

Abstract of “Generic, Deformable Models for 3-D Vehicle Surveillance” by Matthew J. Leotta, Ph.D., Brown University, May 2010.

Vehicle surveillance is the task of measuring moving road vehicles to automatically obtain information about vehicle shape, appearance, identity, path of motion, and, ultimately, driver behavior. While various vehicle sensors exist, none are as versatile as the surveillance camera. Computer vision algorithms can interpret digital images to make a wide variety of vehicle measurements using a single sensor. An ideal algorithm would reconstruct a detailed three-dimensional (3-d) representation of the dynamic traffic scene complete with 3-d vehicle surfaces, trajectories of motion, and identities. Unfortunately, much of the 3-d information is lost during the projection of the world into a 2-d image. As a result, the reconstruction problem is ill-posed. Several researchers have addressed this problem by incorporating prior knowledge about the world to rule out implausible reconstructions. Specifically, in the case of vehicle surveillance, a prior model of 3-d vehicle shape is often used. A constrained alignment of the model to images allows for 3-d shape recovery, tracking, and recognition. Previous 3-d vehicle models are either generic but overly simple or rigid and overly complex. Rigid models represent exactly one vehicle design, so a large collection is needed. A single generic model can deform to a wide variety of shapes, but those shapes have been far too primitive. This thesis presents a new generic 3-d vehicle model that deforms to match a wide variety of passenger vehicles. It is adjustable in complexity between the two extremes. The model is aligned to images by predicting and matching image intensity edges. Novel algorithms are presented for fitting models to images, tracking in video, and learning shape deformation from a collection of detailed rigid models. Experiments compare the proposed model to simple generic models in accuracy and reliability of 3-d shape recovery from images and tracking in video. Standard techniques for recognition are also used to compare the models. The proposed model out performs the existing simple models at each task. Yet, there is still much room for improvement, especially since training data is limited.

Generic, Deformable Models for 3-D Vehicle Surveillance

by

Matthew J. Leotta

B. S., Rensselaer Polytechnic Institute, 2003

Sc. M., Brown University, 2007

Submitted in partial fulfillment of the requirements
for the Degree of Doctor of Philosophy in the
Program in Engineering at Brown University

Providence, Rhode Island

May 2010

© Copyright 2009 by Matthew J. Leotta

This dissertation by Matthew J. Leotta is accepted in its present form by
the Department of Engineering as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date _____

Joseph Mundy, Director

Recommended to the Graduate Council

Date _____

Michael Black, Reader
(Computer Science)

Date _____

Gabriel Taubin, Reader

Approved by the Graduate Council

Date _____

Sheila Bonde
Dean of the Graduate School

Curriculum Vitae

Matthew Joseph Leotta was born on April 25th, 1981. He was raised in the small city of Fulton in Upstate New York. Matthew attended G. Ray Bodley High School where he graduated salutatorian of his class in 1999. In high school, Matthew was a member of the National Honor Society and was very active in extra curricular activities related to mathematics, science, and technology. He was awarded the Rensselaer Medal for achievement in these areas.

After high school, Matthew attended Rensselaer Polytechnic Institute in Troy, New York. He graduated *summa cum laude* in 2003 with a 4.0 GPA and a dual Bachelor of Science in Computer Science and Computer Systems Engineering. At Rensselaer, Matthew joined the Tau Beta Pi Engineering Honor Society and the Eta Kappa Nu Electrical and Computer Engineering Honor Society. He also participated in undergraduate research in the areas of robotics and medial imaging.

In 2003, Matthew joined Brown University in Providence, Rhode Island to pursue a Doctor of Philosophy in Engineering in the area of computer vision. He also received a Master of Science in Applied Mathematics from Brown University in 2007. While at Brown, Matthew published papers on multiple topics in computer graphics and computer vision. His graphics work included a poster at SIGGRAPH, a best paper award at SIBGRAPI, and journal article in the Computer Graphics Forum. His vision work included conference papers at BMVC, ICIP, and CVPR. Matthew also received a commendation from Henrico County, Virginia for applying computer vision to enhance video for a criminal investigation. At Brown, he served as a teaching assistant for a course on *scientific programming in C++* and made significant contributions to the VXL open source C++ computer vision libraries. He is a NASA Rhode Island Space Grant Fellow and a member of IEEE and ACM.

Matthew has accepted a position as a Research and Development Engineer at Kitware, Inc. immediately following the completion of his doctorate in 2009 (degree conferred in 2010). Kitware is located in Clifton Park, New York.

Preface

The research presented in this thesis was conducted in the Laboratory for Engineering Man/Machine Systems (LEMS) at Brown University under the guidance of Professor Joseph Mundy. The work herein was produced primarily between January 2008 and September 2009; however, the ideas for this work evolved over the preceding four year period. Earlier work by the author on vehicle surveillance [68, 69] influenced this thesis as did experience with vehicle recognition in Professor Michael Black's course on *Forensic Computer Vision*.

Early results of this work were presented at the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) in June, 2009. Refer to the following paper:

Matthew J. Leotta and Joseph L. Mundy. Predicting High Resolution Image Edges with a Generic, Adaptive, 3-D Vehicle Model. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1311-1318, June 2009.

This thesis elaborates on the methods used in the CVPR paper, improves upon the algorithms, and extends the work into new application areas.

Acknowledgments

I owe thanks to many people who have supported me and made this thesis possible. My parents have always encouraged me to be creative and strive to meet my potential. The values and work ethic they instilled in me have helped me to reach my goals, and they have always supported me in my efforts.

There have been many great professors who have taught me and guided my work. First and foremost I thank my advisor, Joe Mundy. I am grateful to Joe not only for teaching me about video processing, but more importantly for teaching me how to write a research paper and about the ways of the computer vision community. He always kept me focused on the big picture when I tended to stray. I owe Chuck Stewart at RPI for introducing me to robust estimation, computer vision, VXL, and Joe Mundy. My understanding of probability and random processes benefited greatly from courses with David Cooper and Stuart Geman. Likewise, my understanding of 3-d mesh processing benefited from coursework with Gabriel Taubin. Michael Black gave me a taste of the challenges in applying computer vision to real world problems. In the process he inspired the idea of the deformable vehicle model (or eigen-car), and I thank him for that.

I thank my friends and classmates at Brown for their advice and sympathy. Sometimes only fellow graduate students can understand the stresses that graduate school provides. I also thank my colleagues at Brown and around the world for their collaboration on software development efforts. My work would not have been possible if not for the generous open source contributions to VXL.

Most importantly, I would like to thank my wife, Jennifer. She put her life on hold for me during my six years at Brown. In that time she supported me both emotionally and financially. She helped me keep my sanity when it seemed like I would never be done with graduate school. I don't know what I would have done without her.

Contents

List of Tables	xi
List of Figures	xii
1 Introduction	1
1.1 Applications of 3-d vehicle surveillance	2
1.1.1 Traffic engineering	2
1.1.2 Traffic monitoring	3
1.1.3 Law enforcement	4
1.1.4 Security	4
1.1.5 Military intelligence	5
1.1.6 3-d traffic visualization	6
1.1.7 Traffic video compression	7
1.2 Alternative sensors for vehicle surveillance	8
1.2.1 In-roadway sensors	8
1.2.2 Cooperative sensors	9
1.2.3 Active sensors	10
1.2.4 Passive sensors	11
2 Vehicle Representations	13
2.1 Image formation	15
2.2 Image features	17
2.2.1 Point features	18
2.2.2 Curve features	20
2.2.3 Region features	20
2.3 Geometric vehicle models	23
2.3.1 Agglomerative models	24
2.3.2 Prior models	27

3	A Deformable Vehicle Model	36
3.1	Polygonal mesh definitions	37
3.1.1	Subdivision	41
3.1.2	Texture mapping	43
3.2	Vehicle surface mesh	45
3.3	Vehicle parts	47
3.3.1	Intersections with mesh edges	50
3.4	Image edge prediction	52
3.5	Model deformations	55
3.5.1	Principal components analysis	55
3.5.2	Shape priors	56
4	Fitting to Images	59
4.1	Parameterization of shape and pose	60
4.2	Detection of edge correspondence	61
4.3	Optimization of shape and pose	63
4.3.1	Jacobian matrix computation	64
4.3.2	Regularization	69
4.3.3	Robustness to outliers	71
4.3.4	Multi-scale optimization	76
4.3.5	Multi-view optimization	79
5	Learning the Model	82
5.1	CAD models as training data	87
5.2	Fitting the 3-d mesh	88
5.2.1	Manual initialization	89
5.2.2	Constructing implicit surfaces	91
5.2.3	Shrink-wrapping to implicit surfaces	95
5.3	Fitting the 2-d parts	97
5.3.1	Shrink-wrapping to polygons	98
6	Tracking in Video	102
6.1	Track initialization	103
6.1.1	Video background modeling	104
6.1.2	Sparse optical flow	108
6.1.3	Initializing vehicle pose and shape	112
6.2	Tracking	118
6.2.1	Kalman filter	119
6.2.2	Vehicle state dynamics	123
6.2.3	Foreground edge detection	126

7 Experiments: Convergence of Shape and Pose	129
7.1 Experimental setup	130
7.1.1 Multi-view image data set	130
7.1.2 3-d deformable vehicle models	132
7.1.3 3-d shape error metric	133
7.2 Number of PCA shape parameters	135
7.2.1 Fitting with good initialization	135
7.2.2 Fitting with poor initialization	138
7.3 Region of convergence in pose	142
8 Experiments: Tracking in Video	153
8.1 Video data set	153
8.2 Tracking with fixed shape models	155
8.3 Tracking with shape estimation	160
9 Experiments: Recognition from Shape	170
9.1 Bayesian decision theory	172
9.1.1 Multivariate normal likelihoods	173
9.2 Classification of training data	174
9.3 Classification of recovered vehicle shape	177
9.3.1 Classification from multi-view images	177
9.3.2 Classification from video	181
10 Conclusion	192
10.1 Directions for future work	192
10.1.1 Vehicle model design	192
10.1.2 Learning the vehicle model	194
10.1.3 Fitting to images	196
10.1.4 Tracking	197
10.1.5 Recognition	198
10.2 Final thoughts	199
A IMLS Line Integrals	200
B Software Libraries	203
B.1 Video Streaming Library (VIDL)	204
B.1.1 Introduction	204
B.1.2 Streams	206
B.1.3 Video Frames	210
B.1.4 Pixel Formats	211
B.1.5 Pixel Format Conversion	214

B.2	Probability Distribution Library (VPDL)	218
B.2.1	History	218
B.2.2	Polymorphic Distributions	219
B.2.3	Generic Probability (vpdt)	220
	Bibliography	225

List of Tables

5.1	Training CAD models from the E3D database	83
5.2	Semantic template mesh parameters for manual initialization	91
7.1	Vehicle identities in the multi-view image data set	130
9.1	Confusion matrix for Dodecahedral, 5 parameters, and Gaussian likelihood	176
9.2	Confusion matrix for Ferryman, 5 parameters, and nearest mean	176
9.3	Bayesian classification into five classes using shape from multiple views	178
9.4	Nearest mean classification into five classes using shape from multiple views	178

List of Figures

1.1	Existing 3-d traffic visualizations	7
2.1	An image of two vehicles driving in opposite directions on a roadway	14
2.2	Perspective image projection	15
2.3	Affine image projection	16
2.4	Geometric features of varying intrinsic dimension	17
2.5	Harris corners detected and overlaid on the vehicle image from Figure 2.1	19
2.6	Curves detected and overlaid on the vehicle image from Figure 2.1	21
2.7	Segmentation of foreground and background in the video frame from Figure 2.1 . . .	22
2.8	The plumb line projection method	25
2.9	An agglomerative representation with both regions and points	25
2.10	Tracked 2-d curves	26
2.11	Tracked 3-d curves	27
2.12	Tracked 2-d region boundaries with snakes	28
2.13	Code book vehicle detection	29
2.14	Template matching using the distance transform	30
2.15	Deformable polygon model of vehicle sides	31
2.16	The generic polyhedral vehicle model proposed in Koller <i>et al.</i>	32
2.17	The 29 parameter mesh model proposed by Ferryman <i>et al.</i>	33
2.18	The fixed 3-d mesh models used by Leotta and Mundy	33
2.19	Modeling cast shadow with 3-d polygonal mesh models	34
2.20	A highly detailed CAD model of a sedan	35
3.1	A summary of the deformable vehicle model design	38
3.2	A general mesh with multiple triangulations	39
3.3	Half-edge data structure	40
3.4	Non-oriented and non-manifold meshes	41
3.5	Two levels of mesh subdivision with two different algorithms	42
3.6	Texture mapping with barycentric coordinates	43
3.7	The base 3-d template mesh	45
3.8	Vehicle mesh subdivision	47

3.9	The vehicle mesh texture space	48
3.10	The vehicle parts defined in texture space	49
3.11	The vehicle parts texture mapped onto the 3-d mesh surface	50
3.12	Image edge prediction using part boundaries and occluding contours	54
3.13	Principal components of vehicle deformation	57
4.1	Sampling hypothesis edges for correspondence	62
4.2	Search patterns for finding neighboring edge points	62
4.3	Perpendicular signed error measure for edges	65
4.4	M-estimators and associated weights as function of residual distance	74
4.5	Example error surface at four different residual scales	77
4.6	Multi-scale, multi-view fitting results after various numbers of iterations	81
5.1	Training CAD models	86
5.2	Varying level of detail between different CAD models	88
5.3	Template mesh initialization GUI	90
5.4	A 2-d example of IMLS on points with normals	92
5.5	IMLS (Implicit Moving Least-Squares) Surface of a loader vehicle	94
5.6	Removal of protruding parts	95
5.7	Shrink-wrap in 2-d	95
5.8	Multi-resolution fitting of the vehicle body	96
5.9	Mapping parts from the CAD model to texture space	97
5.10	Initial template parts	98
5.11	Procedure for fitting 2-d parts	100
5.12	Fitting template parts to mapped parts	101
6.1	An example of a 1-d mixture of Gaussians plot	105
6.2	Mathematical morphology operations	109
6.3	KLT flow vectors from the video frame in Figure 2.1	111
6.4	Silhouette prediction	114
6.5	Shadow casting and silhouettes	115
6.6	A summary of track initialization	117
6.7	Detected foreground edges	128
7.1	Images used in multi-view fitting experiments	131
7.2	3-d deformable models compared in experiments	133
7.3	Percent total variation with increasing number of PCA parameters	136
7.4	Error vs. number of PCs for different vehicles with good initialization	137
7.5	Average error vs. number of PCs for different models with good initialization	137
7.6	Average error vs. number of PCs for different image scales with good initialization	137
7.7	Error vs. number of PCs for different vehicles with mean initialization	139

7.8	Average error vs. number of PCs for different models with mean initialization	139
7.9	Average error vs. number of PCs for different image scales with mean initialization .	139
7.10	Fitting overlays: Detailed 3, scale 1, 5 shape parameters	141
7.11	Fitting overlays: Ferryman, scale 1, 5 shape parameters	142
7.12	An inverted plot of error as a function of initial pose with flat plateau	143
7.13	An inverted plot of error as a function of initial pose with bumpy plateau	144
7.14	An inverted plot of error as a function of initial pose with dual plateau	145
7.15	3-d and 2-d views of a joint histogram over distance and angle	146
7.16	Joint histograms with an error threshold of 0.01 meters above the relative best fit . .	147
7.17	Joint histograms with an absolute error threshold of 0.1 meters	149
7.18	Joint histograms with a ground plane constraint and an absolute threshold	150
7.19	Joint histograms with varying number of views and an absolute threshold	152
8.1	Frames from three different video sequences used in tracking	154
8.2	Tracking a white PT Cruiser with fixed shape models in pole view	157
8.3	Tracking a dark red PT Crusier with fixed shape models in aerial view	158
8.4	Tracking a black SUV with fixed shape models in ground view	159
8.5	Tracking a white PT Cruiser with variable shape models in pole view	161
8.6	Tracking a dark red PT Crusier with variable shape models in aerial view	162
8.7	Tracking a black SUV with variable shape models in ground view	164
8.8	Tracking and shape estimation of a dark colored, two-door sedan in pole view	165
8.9	Tracking and shape estimation of a compact crossover SUV in ground view	166
8.10	Single frame overlays from five other easy tracking examples	167
8.11	Single frame overlays from six other challenging tracking examples	168
9.1	Scatter of vehicle classes in the first two principal components	171
9.2	Success rates for classification of training data into five classes	175
9.3	Success rates for classification of training data into three classes	177
9.4	Success rates for classification of shapes recovered from multi-view fitting	180
9.5	Scaling of posterior probabilities for visualization	182
9.6	Three-way classification of the tracked, white PT Cruiser	183
9.7	Five-way classification of the tracked, black SUV	184
9.8	Three-way classification of the tracked, black SUV	185
9.9	Five-way classification of the tracked, red, two-door sedan	186
9.10	Five-way classification of the tracked, crossover SUV	187
9.11	Three-way classification of the tracked, crossover SUV	188
9.12	Five-way classification of a tracked, white, four-door sedan	189
9.13	Five-way classification of the tracked, white pickup	190
10.1	Relearning shape with initialization from truncated PCA projections	195

Chapter 1

Introduction

To understand the concept of *3-d vehicle surveillance from video*, one must start with an understanding of surveillance in general. The term **surveillance** means *to keep watch over* or *to closely observe*. It is most frequently used in the context of suspected criminal activity. Surveillance entails identifying a target, following the actions of the target over time, and interpreting those actions to look for particular behavior (*e.g.* criminal behavior). Thus, it should come as no surprise that the primary targets of surveillance are people.

One type of human behavior that is particularly important in surveillance is travel. Many times one is interested in where a person is going and what route they follow. When people travel over large distances they usually do so in a vehicle (car, boat, airplane, *etc.*). In this case, the person is generally not directly observable, and the vehicle becomes the target of surveillance. Hence, vehicle surveillance is a critical component in the surveillance of people. The technical details of vehicle surveillance are quite different from those of human surveillance; however, the underlying goals are the same. It is the behavior of the people traveling in vehicles that is ultimately of interest.

Surveillance is an innately human task requiring a great deal of cognitive energy. This thesis explores automated surveillance as a means to relieve or eliminate this human burden. Automation in surveillance requires artificial intelligence to replace human intelligence. However, the current state of artificial intelligence comes nowhere near human capability, especially with regard to human behavior interpretation. So it is important to keep a human in the loop to make the final, critical decisions. Nonetheless, automation greatly reduces human labor by handling lower level cognitive tasks in large volumes. Such low level tasks may include preliminary target recognition (subject to human verification in critical situations) and target tracking (*i.e.* following a target's motions over time).

While not always a visual process, surveillance most frequently utilizes the sense of sight. The word *surveillance* itself has French origin and literally means *watch over*. The automated surveillance approach in this thesis stays true to meaning and processes visual information. At the intersection of visual processing and artificial intelligence is the field of **computer vision**. The automated visual surveillance presented here makes use of the work in this field. The computer vision literature

devotes much attention to both people and vehicle surveillance. This thesis addresses the latter.

There are many categories of vehicles that people use for transportation, but by far the most common is the road vehicle. Road vehicles include cars, buses, trucks, motorcycles, *etc.* Of these the most common road vehicles are passenger vehicles: sedans, pickup trucks, minivans, sport utility vehicles (SUVs), *etc.* Passenger vehicles have two axles and typically carry a small number of passengers (2 to 8). This thesis focuses on this collection of most common vehicles, although the same techniques generalize to other classes of vehicles.

The aspect of this work that sets it apart from most other vehicle surveillance methods is a focus on the three-dimensional (3-d) shape recovery of the vehicles. Typically automated vehicle surveillance addresses the problems of recognition and tracking¹. **Vehicle recognition** is the problem of identifying a vehicle, either as a specific vehicle instance (*e.g.* John Doe’s car), or as a member of a category of vehicles (*e.g.* a sedan, broadly; or a Toyota Corolla, narrowly). **Vehicle tracking** is the problem of determining the location of a vehicle as it changes over time. Tracking may include determination of pose (position and orientation), velocity, acceleration, and a recorded log of how these properties have evolved in time. In the case of this thesis, the solution to both of these problems is driven by 3-d shape. **Vehicle 3-d shape recovery** is the problem of estimating a 3-d representation of the vehicle from 2-d imagery. While 3-d shape is not strictly required for surveillance, recognition and tracking may be formulated as a by-product of 3-d shape recovery as will be shown in later chapters. Yet the true advantage of recovering 3-d shape comes from the new applications of vehicle surveillance that are built around the knowledge of 3-d shape.

1.1 Applications of 3-d vehicle surveillance

There are several applications for vehicle surveillance used by very different clients. While the term *surveillance* often has security connotations, it is used here in the more general sense of monitoring vehicle activity for any reason. Vehicle surveillance is used for traffic engineering, law enforcement, traffic management, and military intelligence gathering, to name a few. Some of these applications are already well handled by existing technologies but could be improved using 3-d vehicle surveillance. Other applications are poorly addressed by current state of the art technology or simply are impossible without 3-d shape recovery as proposed in this thesis.

1.1.1 Traffic engineering

Traffic engineering is the branch of civil engineering that is focused on transportation. It is the process of designing roadways—including the placement of traffic signs and operation of traffic lights—to maximize the safety and efficiency of vehicle transportation. A key part of this process is the study of traffic patterns to identify bottlenecks and dangerous conditions. Traffic surveys measure

¹A third common problem is **activity recognition**, the further analysis of the motion patterns to interpret the actions and goals of the target. This problem is beyond the scope of this thesis and is left for the human in the loop.

number of vehicles, vehicle velocity, frequency of lane changes, distance between vehicles, and other parameters of traffic flow. Sometimes vehicles must be classified and counted by class. For example, a survey may wish to estimate the average load exerted on a bridge or roadway. Knowing the class of each vehicle provides a better estimate of its weight. By studying traffic patterns engineers can design improved roadways that are safer and more efficient.

Surveying traffic was once a tedious manual process but has increasingly become automated. Various vehicle surveillance sensors (discussed in Section 1.2) are used to record traffic statistics for later analysis. Some sensors are actively incorporated into the design of traffic systems for real-time use. Traffic light control schemes often use sensors to dynamically adjust the light timing in response to the detected traffic volumes.

Automated traffic surveying is one application of vehicle surveillance that is thoroughly addressed using a barrage of existing sensor systems. Each of these traditional vehicle sensors (Section 1.2) is honed on a specific, limited set of measurements. Counting vehicles and measuring velocity are solved problems, but most sensors are unable to measure the more complex traffic traits like lane changes or acceleration profiles. Computer vision surveillance is well suited to these kinds of measurements since it can recover a detailed track of each vehicle's motion. In fact, a single computer vision system can make almost all of the required measurements. This same feat would otherwise require multiple instances of various types of other more intrusive traffic sensors.

1.1.2 Traffic monitoring

Most large population centers have a Transportation Management Center (TMC) from which traffic on major roadways is monitored. The TMC is a central command center into which many live vehicle surveillance sensors feed. Traffic flow is monitored with automated sensors and live video cameras for manual observation. The video is still referred to as closed-circuit television (CCTV). However, most TMCs now broadcast this video to websites or local television news stations for traffic reports.

Much of the TMC data processing for traffic reports and traffic statistic can be easily automated. Even so, human operators are still needed for other tasks that are difficult to automate. An operator observing a reckless driver might alert a nearby police patrol car. Reckless driving encompasses various activities like speeding, swerving, or traveling the wrong way down a highway. Similarly, an operator might alert emergency medical services if an accident is observed. Automatically detecting and acting on these events requires a more complicated sensor system. Computer vision systems are well suited to this task and can make use of the same video data that is already captured for human for observation.

Computer vision systems have the potential to track vehicles over distances, recognize complex behavior, and report vehicle descriptions. Many sensors are designed to only measure normal vehicle traffic; a vehicle that spins out or goes off of the road would go undetected. On the other hand, a computer vision system can track complicated vehicle maneuvers even as a vehicle leaves the roadway. Furthermore, if the 3-d shapes of the vehicles are recovered then a vehicle collision is easily detected as an event where these shapes intersect in 3-d space. Some commercial systems

claim some limited vehicle behavior detection [19], but this is still an active area of research [46, 95].

1.1.3 Law enforcement

Vehicle surveillance is also used directly for law enforcement purposes. Radar (See Section 1.2.3.) is commonly used by law enforcement officers to detect vehicles traveling faster than the speed limit. The officer can stop the driver and issue a fine. The radar sensor speed measurements also provide evidence for use in court if the fine is contested. Only recently have GPS measurements (See Section 1.2.2.) taken by the driver been admissible as counter evidence [66].

Some traffic law enforcement is even more automated. Cameras are used to automatically capture vehicles in the act of violating traffic laws. These cameras are usually triggered by external sensors that detect actions like speeding or failing to stop at a red light. The trigger sensors may use any of the various traffic sensors. Inductive loop (See Section 1.2.1.) and radar are common triggers for red light and speed cameras respectively. The cameras take high resolution images which are used both for evidence and to identify the perpetrator. Once the evidence is reviewed, the vehicle is identified and a fine is sent to the owner. Some systems use automatic license plate recognition to identify the vehicle owner. In this case, A computer vision system uses optical character recognition (OCR) to read the number on the license plate and then looks up the number in a database of registered vehicles. Such systems may operate from start to finish without any human intervention until the perpetrator receives a fine in the mail with a copy of the image evidence.

Current automated traffic law enforcement systems have some drawbacks. The automated fines are directed at the registered vehicle owner and not the driver. The driver and owner are not always the same person. Sometimes even the vehicle is wrongly identified. People have exploited the fact that only the license plate is used for identification. By creating a fake copy of valid license plate on another vehicle, one can purposefully trigger an automated system and have the fine sent to someone else [96]. While advances in technology may not solve all of these problems, they can improve vehicle identification. Recognizing a vehicle by shape in addition to license plate number could add some redundancy and prevent some of these scams. If the type of vehicle does not match the type registered to the license number, then the system could alert a human to step in and investigate the situation.

1.1.4 Security

Security is the application most commonly associated with video surveillance. Surveillance cameras are frequently used to protect locations of value from crime. The camera acts to deter, to collect evidence, and—in rare cases—to foil criminal activity. As cameras have become cheaper their use has become more widespread. Once used only to protect the most valuable assets, cameras are now used in many small businesses (*e.g.* small banks, gas stations, and supermarkets) and even in private residences. Threat of terrorist activity has also increased interest in video surveillance at public transportations hubs like airports and train stations.

Surveillance cameras work to deter crime by increasing a criminal's fear of being caught. The deterrent factor is sometimes great enough by itself that a realistic but non-functioning camera is used and no data is captured. When data is captured it is usually recorded and only reviewed for evidence after an incident occurs. Surveillance video is rarely monitored live by human operators. Human labor is the most expensive component of a surveillance system. It is only used in the most critical applications to respond to incidents as they occur.

Computer vision techniques can aid video surveillance both during capture and when reviewing evidence. Algorithms operating on live video must run in real-time and without human guidance. They aim to replace or reduce human labor in CCTV monitoring. Such algorithms rely on tracking and recognition, but they also require activity recognition to detect unusual events. Existing algorithms tend to be application specific and limited, but this is another area of active research [32, 45].

Computer vision plays a different role when analyzing prerecorded security video of an incident. Computer vision in live surveillance detects unusual activity and alerts a human operator. The aim is to reduce human labor in analyzing large amounts of data. Conversely, when analyzing recorded video after an incident the amount of data is small and its importance is already known. In such cases, the goal is to extract as much information from the data as possible. The requirements of real-time and automatic processing are relaxed. The need for computer vision arises from the poor quality of some surveillance video. It is often the case with small businesses that video surveillance systems are installed merely to procure reduced insurance premiums. These systems are configured and maintained with little care for the quality of the imagery. When the poor quality video is actually needed as evidence, it may be impossible for detectives to interpret the raw data. One example is a homicide case in Virginia. Detectives were unable to identify the suspect's car in a poor quality gas station security video. A group of scientists assisted with the identification by using computer vision techniques with a 3-d vehicle shape model to extract additional information from the video [64].

1.1.5 Military intelligence

As is often the case in technological development, advances in vehicle surveillance have frequently been a product of research for military applications. CCTV was first used during the 1940's to monitor rocket launches from a safe distance. Cooperative sensors (See Section 1.2.2.) are an adaptation of military technology used for Identification of Friend or Foe (IFF).

A major focus in modern military research is aerial video surveillance. Aerial imagery for military reconnaissance goes as far back as 1859 when it may have been used by the French in the Battle of Solferino [41]. In the early days of both manned flight and photography, still aerial images were taken from a balloon. Today, several types of unmanned aerial vehicles (UAVs) are able to remotely capture and transmit high resolution aerial video. UAVs can hover over an area and provide video surveillance of road vehicles, among other things.

There are several reasons for the military to track vehicles from an aerial platform. They might be gathering intelligence on a suspected terrorist or person of interest. By tracking a suspect's

vehicle they can determine where the person goes, with whom they meet, where meetings are held, *etc.* If all vehicles in an area can be tracked at once, they can trace any vehicle's path in reverse to discover its place of origin. This reverse tracking is useful when a vehicle is discovered to be involved in an incident like a terrorist attack. It is useful to know where such a vehicle came from. Finally, many UAVs, like the MQ-1 Predator, are capable of carrying weapons systems in addition to surveillance systems. When the need arises to destroy a road vehicle, video tracking can assist in targeting a missile strike.

Another military application of vehicle surveillance is facility monitoring. As discussed in Section 1.1.4, surveillance cameras are used to protect valuable assets like military bases. Military applications of security are one of the few cases important enough to require live video monitoring. It is important to track vehicles as they move around a base to identify any unauthorized entry into restricted areas or other suspicious activities. As previously stated, human labor is an expensive component of surveillance. Automated traffic monitoring is highly desirable. With a network of surveillance cameras scattered around a facility, it is possible to use computer vision techniques to identify vehicles and track their movement automatically. Using a 3-d shape model of vehicles, as proposed in this thesis, makes it easier to relate images of the same vehicle when taken by cameras with different viewing directions.

1.1.6 3-d traffic visualization

In addition to the many existing applications that could be improved by 3-d vehicle surveillance, there are several more that require 3-d shape recovery to exist. Consider a wide area under surveillance by an extremely high resolution aerial camera or a network of ground based cameras. With a system built on the ideas presented in this thesis, the 3-d shape of each vehicle is reconstructed as its motion is tracked over time. Such a system could support live 3-d traffic visualization.

Current traffic reports tend to present information in flat 2-d maps that indicate the current average speed on major roadways. Invariably this information is supplemented with live views from various traffic cameras. People like the information overview of the map, but also want to see what the traffic really looks like to know which lanes are moving and which might be backed up due to an accident. This dual view of traffic is the same for television news broadcasts and for internet viewing. The only difference is internet users have a more interactive experience by selecting which part of the map and which video streams to view.

There is currently no system that can show a live 3-d view of the actual vehicles in traffic from arbitrary viewpoints. However, some systems are moving in this direction. Triangle Software produces a system that maps the 2-d traffic map into 3-d and adds scattered 3-d models of buildings and vehicles to represent landmarks and accidents (Figure 1.1 left). The 3-d vehicle models are stock icons to indicate the presence of an accident but do not show the actual configuration of vehicles. Alternatively, Google Earth provides a detailed, realistic 3-d world model complete with buildings, bridges, and other static structures. Unfortunately, vehicles are captured from single, static image and flattened into the ground plane (Figure 1.1 right). Traffic reports are shown by a series of

scattered colored dots (red, yellow, and green) to indicate average traffic speeds, but the flattened vehicles on the road never move.

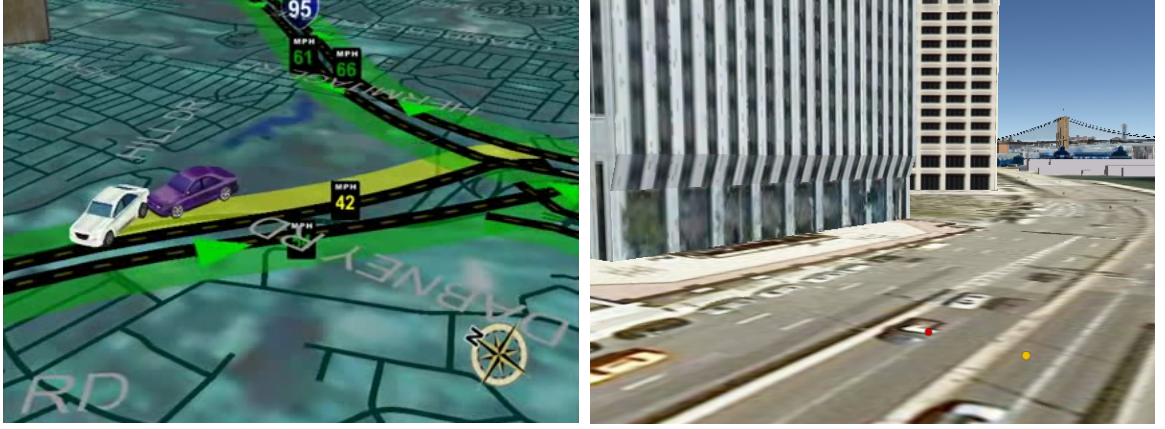


Figure 1.1: Existing 3-d traffic visualizations. Left: an image from Beat the Traffic®, a 3-d traffic reporting tool from Triangle Software. Right: a detailed 3-d world model from Google Earth with flattened, static vehicles.

All of the existing 3-d world models are lacking in dynamic elements. 3-d vehicle surveillance could add live, moving, 3-d vehicle models corresponding to the actual vehicles in the real world. But, a live 3-d traffic visualization is not possible without 3-d vehicle surveillance as described in this thesis. Other computer vision algorithms for vehicle surveillance are not sufficient if they do not recover 3-d vehicle shape.

1.1.7 Traffic video compression

A dynamic 3-d model of traffic, as in Section 1.1.6, is created from large amounts of video data. Transmitting the dynamic 3-d traffic model may require less bandwidth than transmitting video streams. The background of the video is fixed or slowly changing; it can be transmitted once and updated infrequently. Similarly, each vehicle model is fixed in shape and several vehicles may share a single shape model. The only information that must be transmitted at each frame is the vehicle motion. Each frame of the video can be reconstructed (approximately) using computer graphics to render the 3-d world into images.

This idea of using a 3-d scene model for video compression is not new. In fact, it is included in the MPEG-4 standard for video coding (Part 11) [48]. Scene modeling compression is rarely used in practice because it is difficult to recover scene information from video. Compressing multiple videos of multiple views of a dynamic scene with a single 3-d dynamic model can provide huge bandwidth savings, but only if the technology exists to track vehicles and recover a model of their 3-d shape.

1.2 Alternative sensors for vehicle surveillance

There are several technologies used to implement vehicle surveillance applications. This thesis focuses on advancements in computer vision algorithms for vehicle surveillance. A considerable amount of computation and complex reasoning is required to convert image data captured by cameras into the desired vehicle measurements. Other technologies employ sensors designed specifically to measure the desired vehicle data. These devices can obtain vehicle measurements with very little post processing of the raw sensor output. Therefore it is important to review the capabilities and restrictions of these competing technologies and to examine why image and video processing techniques play an important role in vehicle surveillance.

The U.S. Department of Transportation's Federal Highway Administration publishes a handbook [58] describing and comparing all commercially available (and some theoretical) vehicle surveillance technologies. A brief summary of these technologies is provided here. The sensors may be classified by installation type as *in-roadway* or *over-roadway*. In-roadway sensors are installed underneath, inside, or directly on top of the road surface, while over-roadway sensors remotely monitor vehicles from a distance above or off to the side. Many over-roadway sensors do not rely on a road surface *per se* and some work even in off-road applications. Over-roadway sensors are further classified as cooperative (a device in the vehicle communicates with the sensor), active (a signal is sent to the vehicle and reflected back to the sensor), or passive (signals naturally emitted from the vehicle or reflected from the environment are captured).

1.2.1 In-roadway sensors

Sensors installed in the roadway work by detecting one of two phenomena: One, the tires exert pressure on the road surface. Two, the presence of a large mass of ferromagnetic material (the steel engine, chassis, *etc.*) causes a disturbance in magnetic field. In all cases, these sensors are expensive to install and maintain, and traffic must be interrupted. In-roadway sensors are usually permanent in location once installed. Additionally, when roads are resurfaced, some devices need to be removed and reinstalled.

Pressure-based sensors take the shape of a long strip laid transversely across or within the road. One of the simplest examples is a thin rubber tube full of air. Vehicle tires compress the tube and changes in air pressure are registered. Each of the vehicles axles is detected as it passes over the tube. A more modern version uses a fiber optic cable and detects changes in the optical properties of the fiber when stressed. Another variation on this theme is the use of piezoelectric sensors embedded in the road surface. Some sophisticated piezoelectric sensors also feature the ability to measure the weight of a vehicle.

Ferromagnetic sensors include inductive loop and geomagnetic varieties. The inductive loop is the most widely used of all vehicle sensors. A length of wire is formed into a large diameter loop with one or more turns. The wire is buried in the road surface and connected to a sensor that passes current through the loop. The loop acts as an inductor in the circuit. When a vehicle passes over, its

high iron content causes a drop in the inductance of the loop. The change in inductance is detected by the sensor. Using a related idea, geomagnetic sensors detect small perturbations in the earth's magnetic field caused by the iron in the vehicle passing overhead.

In-roadway sensors are some of the oldest, and most well studied traffic sensors in use today. They are very simple and accurate, but also extremely limited in functionality as surveillance devices. In-roadway sensors are primarily useful in simple traffic engineering, traffic management, and law enforcement applications. A single sensor can only detect and count vehicles; a pair of sensors can estimate velocity at one location and count vehicle axles. These sensors certainly have no capability to track vehicles over distance or recover 3-d shape and limited capability to recognize vehicles (only what can be determined from number and separation of axles).

The most severe disadvantage of in-roadway sensors for surveillance is the need to permanently embed them in a fixed location. In most security and military applications one cannot assume that all vehicles will cross a specific location. Consider an off-road surveillance task like securing a military base in the desert or the United States and Mexico border. In both cases, it cannot be assumed that all vehicles will travel across any given location. A single surveillance camera can monitor a large area, but an in-roadway sensor is useless when there are no roadways.

1.2.2 Cooperative sensors

Cooperative sensor are the newest form of vehicle surveillance. A cooperative sensor is one that communicates directly with a device onboard each vehicle. The vehicle identifies itself to the sensor and possibly transmits additional information to the sensor.

RFID (Radio Frequency IDentification) tags are the only cooperative vehicle surveillance technology in widespread use today. RFID tags have been used for electronic toll collection since the late 1980's. An RFID transponder is required in each participating vehicle. The sensor transmits a radio frequency message to the RFID tag as the vehicle passes by. The RFID tag acts as a transponder and replies with a message that uniquely identifies it. The unique identity is used to charge tolls to the driver's account. Since many vehicles now contain RFID transponders for toll collection, attempts have been made to use RFID for traffic surveying [34]. This technology allows an individual vehicle to be tracked over long distances (when using multiple RFID readers).

A second cooperative sensing technology uses an even more ubiquitous transponder device: the mobile phone. The enactment of the *Wireless Communications and Public Safety Act of 1999* required mobile phones in the United States to provide an estimate of location for *Enhanced 911* emergency service. The fact that many drivers carry a mobile phone has sparked research on the use of 911 location services to monitor traffic [108]. Early mobile phone location services computed location by triangulation using signal strength from nearby cellular towers. The estimated location was not accurate enough for most traffic monitoring applications. However, in recent years the cost and size of Global Positioning System (GPS) circuits have decreased dramatically making them suitable for installation in many mobile phones. GPS makes 911 location service (and therefore traffic monitoring) much more accurate. At least one company, IntelliOne Technologies Corp. [47],

has started commercializing traffic monitoring via mobile phone tracking. Another company, Dash Navigation, Inc. [23], has approached this idea from a different angle. Dash sells dashboard mounted GPS navigation systems with the addition of mobile internet connectivity. Each dashboard GPS tracks itself and reports to a central server that aggregates the data and transmits traffic reports back to all equipped vehicles.

Cooperative vehicle surveillance is likely to become a big part of some vehicle surveillance applications in the future. It offers wide area vehicle surveillance using existing wireless network infrastructure. The biggest disadvantage is that it requires the voluntary participation of vehicle owners. Cooperative surveillance is limited to monitoring the participating subset of the traffic. This subset may be sufficient for estimating average road speeds, but it cannot accurately provide a vehicle count. Even if future legislation were to mandate the use of a transponder in every vehicle, this would still not meet all surveillance needs. Mandated sensors could malfunction or could be illegally disabled to evade law enforcement surveillance. Cooperative sensors have an important place in vehicle surveillance, but they will never be acceptable for law enforcement, security, and military applications where the target does not want to be monitored. This Achilles' heel leaves room for sensors that do not rely on cooperation.

1.2.3 Active sensors

Active sensors are like cooperative sensors in that they transmit a signal to the vehicle. They differ in that the active sensors do not require the vehicle to transmit back. Active sensors measure properties of the transmitted signal after it is naturally reflected back or interrupted by the vehicle. A variety of signal modalities are used for this.

Radar (RADio Detection And Ranging) is one of the most common active vehicle surveillance technologies. Radar sensors transmit a microwave radio signal which is reflected by a moving vehicle and recorded by the sensor. The system relies on the Doppler effect to compute vehicle velocity by analyzing the shift in frequency of the reflected signal. Standard continuous wave Doppler radar can only detect moving vehicles, but frequency modulated variants can detect the presence of stopped vehicles. Radar sensors can be mounted above roadways, to the side of roadways, or even in handheld and vehicle mounted devices (radar guns) for law enforcement.

Similar to radar are laser range finders (sometimes called laser radar, ladar, or lidar). Laser range finders use an eye-safe infrared laser beam to measure the distance of a vehicle at a precise point. Some laser range finders transmit a pulse of light and compute distance based on time of flight (TOF). Others offset the transmitter from the sensor and compute distance by triangulation. Laser range finders can compute speed using multiple sensors or multiple measurements over time. Some complex laser systems sweep the laser across the road and take many 3-d measurements as a vehicle passes by. This is the only sensor other than video cameras to address the detailed shape recovery problem.

Ultrasonic is a third signal modality similar to both microwave and laser radar. These sensors operate with sound waves at frequencies above the range of human hearing. Ultrasonic sensors come

in both pulsed and continuous waveform varieties. The pulsed versions make TOF measurements while the continuous waveform versions make use of the Doppler effect. Ultrasonic sensors are used most extensively in Japan where the use of in-roadway sensors is restricted by law.

While active sensors do not require vehicle cooperation, they do have other drawbacks. Installation and maintenance can be complicated by the need to keep the sensor calibrated with respect to the transmitter. Range may be limited by the power and focus of signal transmission. In some security and law enforcement applications it is advantageous for tracking to remain covert. An active signal transmission may be detected by the vehicle causing the driver to adjust behavior. An example of this is the radar detector. Such a device facilitates the need for law enforcement to use a radar detector (e.g. products by Stealth Micro Systems Pty Ltd. [99]). Furthermore, some active sensors can be intentionally tricked by nefarious drivers with jamming devices. Jamming means the target vehicle actively transmits a similar signal to the sensor to cause incorrect measurements. Passive sensors can alleviate many of these problems.

1.2.4 Passive sensors

Passive sensors operate by analyzing signals already present in the environment. These include signals both emitted by vehicles and those reflected from ambient sources. Since the input signal is uncontrolled, passive sensors must address the additional challenge of dealing with changes in the environment. They must adapt to different times of the day, temperatures, or weather conditions. As with active sensors, multiple signal modalities are exploited.

Passive acoustic array sensors measure sound waves produced by traveling vehicles. Sound waves are produced from several sources including engine noise and interaction between tires and the road. An array of microphones captures these sounds. Changes in the magnitude of the sound energy and time delays between microphones are used to count vehicles and estimate velocity.

Passive infrared sensors use optics to collect and focus infrared energy. The simplest sensors (non-imaging) contain a small number of detectors (one to five) in the focal plane. The detectors cover different zones on the ground and respond to energy changes as a vehicle passes. The infrared energy changes are caused by differences in emission (vehicles tend to be warmer than the road) and reflectance (vehicles tend to reflect more atmospheric energy than the road). More complicated passive infrared sensors use a large array of detectors to form an image. Such images require many of the same processing techniques as used with visible light cameras.

Visible light cameras use optics to focus visible light onto an image plane. This is the type of sensor that is explored in this thesis. The images may capture color or may be monochromatic. Still images may be captured sporadically or continually to form a video signal. A big advantage of using cameras for surveillance is that the data is easily interpreted by humans. Many cameras are already installed for the purpose of manual surveillance. The images are televised in traffic reports, recorded for security purposes, and monitored in traffic management centers. These same cameras can be used in automated surveillance. The challenge is implementing an appropriate algorithm to convert the large amount of image data into the desired measurements. Chapter 2 reviews the vast

literature on this topic in more detail. With the right algorithms, imaging sensors have the potential to perform more types of measurements than most other sensor technologies. Image sensors have the potential to recover 3-d shape, classify vehicles, track them over long ranges, estimate speed, detect complex behaviors like lane changes, and much more. Cameras are also relatively expensive to install and maintain when compared to other sensor hardware.

Visible light cameras coupled with computer vision algorithms provide the only tractable technology for all of the outlined 3-d vehicle surveillance applications. It is the only technology for passive acquisition of 3-d vehicle shapes. Passive sensing is important in law enforcement, security, and military surveillance applications. In each case the target may not want to be tracked, or the tracker may not want the target to know it is being tracked. The use of 3-d shape, in addition to enabling exciting new applications, provides important information for computer vision whenever the vehicle is observed from multiple viewpoints. Multiple viewpoints does not imply that multiple cameras are involved or even that a single camera is moving. The viewpoint is a relative measure of the vehicle's orientation with respect to the camera. Hence, even with a fixed camera, the vehicle produces a new viewpoint whenever it rotates. Furthermore, under general imaging conditions (discussed further in Chapter 2), 3-d information is observable when relative translation occurs between the camera and vehicle—even if neither rotates. Therefore, 3-d shape is useful in almost all visual vehicle surveillance applications.

Going forward, this thesis will address the computer vision techniques for 3-d vehicle surveillance. It starts with a thorough treatment of 3-d vehicle shape recovery. The shape recovery technique is then adapted to video for simultaneous tracking and shape recovery. Lastly, the space of vehicle shapes is utilized for recognition.

Chapter 2

Vehicle Representations

In using computer vision as a vehicle surveillance tool, there is no universal algorithm for processing the data. Unlike most other vehicle sensors, which make a small number of targeted measurements, a camera collects a very large number of non-specific measurements. A single image can contain millions of pixels, each a measurement of color or intensity along a narrow shaft of light. Each pixel in isolation is almost meaningless, but combined into an image they provide more useful information than almost any other individual sensor. The difficulty arises in converting this large array of indirect measurements into a select number of direct ones. Critical in this conversion from image data to vehicle measurements is the vehicle representation, the mathematical model that describes a vehicle's appearance and motion in an image.

Given that image input is so general, one might wonder why surveillance algorithms should not also be general. Why not recognize, track, and recover the shape of all objects in the scene, or even all moving objects? Indeed some research has taken a more general path in addressing each of these problems. However, these problems are under constrained and cannot be solved without some assumptions, prior knowledge, or constraints. When the assumptions are general and weak the results are usually quite limited. Better results are achieved with stronger assumptions and more reliance on prior knowledge. The assumption in this thesis is that the targets are a restricted collection of road vehicles with known prior information about 3-d shape and known motion constraints. Using such a vehicle representation makes it much easier to understand otherwise ambiguous image data.

There are an infinite number of 3-d scenes that could have created any 2-d image. Luckily, the vast majority of them can be deemed improbable using prior knowledge of the world. It is difficult to appreciate why computer vision is so challenging when our minds are so efficient at subconsciously applying our own prior knowledge. Consider the image of vehicles in Figure 2.1. Given this single image, most people could describe the 3-d shape of these vehicles, estimate their rough size and position in space, and guess at how these vehicles might be moving. The same people would fail miserably at making the same estimates about the tree branch in the lower right. The difference is the strength of constraints from prior knowledge.

Without prior knowledge, vehicles are difficult subjects to analyze from images. Most vehicles



Figure 2.1: An image of two vehicles driving in opposite directions on a roadway.

have large curved surfaces with uniform color and a shiny, mirror-like finish. Uniform regions do not contain much information. Vehicle bodies, and especially windows, tend to reflect a distorted view of the environment. Reflected features from the environment are dependent on the vehicle pose and are not informative of shape when the environment is unknown. Furthermore, reflected features are locally difficult to distinguish from the useful rigid features on the vehicle.

Luckily, there is usually more data available than a single image like Figure 2.1. Vehicles, by their very nature, are often observed in motion. Using a video camera to capture many images over time helps to separate the relevant information from clutter in the environment and random noise produced by the sensor. Video provides multiple viewpoints of a vehicle that are related in a predictable way given the prior constraints on vehicle motion. Thus, motion is an important part of a vehicle representation.

Even with video, automated vehicle surveillance is difficult. The algorithm must track the changing pose of a 3-d vehicle over video frames when its shape and identity are unknown. Conversely, it must recognize or estimate shape from multiple video frames without knowing how the vehicle is moving. These problems are intertwined and must be addressed simultaneously. To understand how the various vehicle representation models can overcome these challenges, formal definitions of image formation and image features are needed. Some mathematical background on the formation of images and the extraction of geometric image features is followed by a review of the literature on vehicle representation.

2.1 Image formation

An image is formed when light rays enter a camera lens and are focused onto the image plane. These rays have been emitted and reflected by objects in a 3-d scene. Each pixel in a digital image is a measure of the received light energy integrated over a small area on the image plane (the pixel area on the sensor) and over a short period of time (the exposure time). The 3-d surface from which each ray of light was emitted or last reflected could lie at any point along the ray. Thus, all depth information is lost.

Mathematically, the image formation is a projection of 3-d data into a 2-d plane. The camera model that is frequently used in computer vision and is used throughout this thesis is the pinhole model. The model of projection in the pinhole camera is called perspective projection. Under perspective projection, all imaged rays pass through a single point (the pinhole) called the center of projection. As shown in Figure 2.2, the image forms mirrored top-to-bottom and left-to-right (*i.e.* rotated 180 degrees). For simplicity, the equivalent virtual image plane is often used for reasoning because it represents the image without mirroring. The virtual image plane floats in space between the center of projection and the scene.

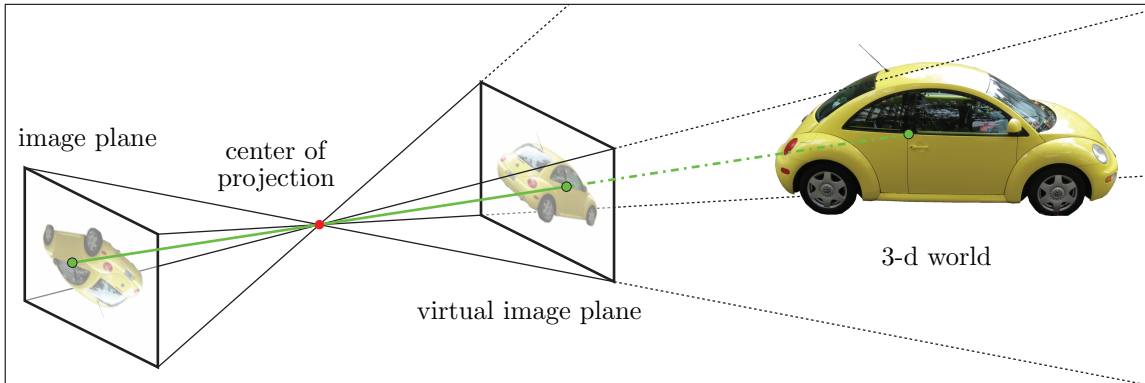


Figure 2.2: Perspective image projection.

Since the rays diverge from the center of projection, objects appear smaller as their distance increases. The different angles of incident light rays also mean that the viewing direction changes continuously across the image. As an object translates with respect to the camera and moves in the image, its viewpoint also changes. Consequently, the effects of 3-d shape are present in the image even when the only motion is translation.

In reality, the camera aperture is not an infinitesimal pinhole. A larger hole is needed to collect more light. Instead, a lens (or series of lenses) focuses the light from a larger aperture onto the image plane. When a lens is used, the entire scene is no longer guaranteed to be in focus. However, when the scene is in focus and a high quality¹ lens is used, pinhole perspective projection remains

¹“High quality” refers to a lens that is designed to not introduce other geometric distortions. If a lens does introduce distortion, the distortion is often modeled as 2-d warping of the image. Lens distortion is not addressed in this thesis.

an accurate model of projection for real cameras.

A simpler model of projection is affine projection. Under affine projection, all imaged rays of light are parallel (See Figure 2.3.). There is no change in the apparent size of objects with distance; the camera can move along its optical axis and produce the exact same image. Translating 3-d objects have no measurable depth information when they do not rotate.

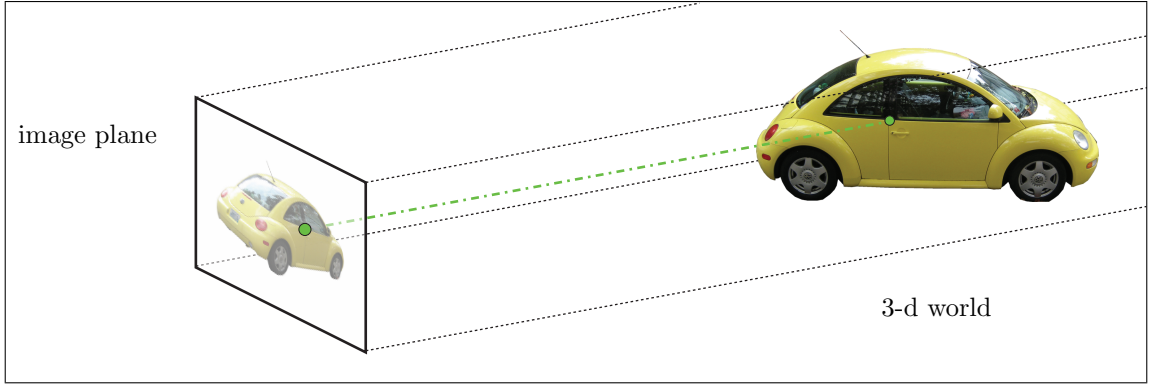


Figure 2.3: Affine image projection.

It is very difficult to manufacture a physical camera that uses true affine projection. Despite that, the affine projection model is useful in some limiting cases of perspective projection. The light rays in the perspective model become more and more parallel as the ratio of focal length to image width increases. Focal length is the distance from the center of projection to the image plane. In an actual camera, the affine model may be appropriate when viewing objects at a great distance with a strong magnifying lens. Affine projection has the advantage of being a linear projection which simplifies some calculations. Yet, perspective projection is more informative about 3-d geometry, especially depth.

Both projection models may be represented numerically in the same matrix form. This requires the use of homogeneous coordinates to handle the nonlinearity of the perspective case. Homogeneous coordinates increase the dimension of a point by one and use the extra dimension as a divisor to the other coordinates. For example, $\mathbf{x} = [X \ Y \ Z \ W]^\top$ represents the 3-d point with coordinates $(\frac{X}{W}, \frac{Y}{W}, \frac{Z}{W})$. Homogeneous points are only defined up to a scale factor. That is, $a\mathbf{x} \equiv \mathbf{x}$ for all scalar $a \neq 0$. The special case where $W = 0$ represents a point at infinity.

The projection of homogeneous 3-d point $\mathbf{x}$ into homogeneous 2-d image coordinate $\mathbf{u}$ is computed as

$$\mathbf{u} = \mathbf{P}\mathbf{x}, \quad (2.1)$$

where $\mathbf{P}$ is a 3×4 projection matrix. The projection matrix can represent either a perspective or affine projection. The projection is perspective if the left 3×3 sub-matrix of $\mathbf{P}$ has full rank. The projection is affine if the bottom row of this sub-matrix vanishes (*i.e.* all values in the row are zero). The reader is referred to [43] for more information on homogeneous coordinates and these projections.

2.2 Image features

All image structure is inherently based in on geometry. All other information, like color or intensity, is meaningful only with respect to that geometry. For example, a pixel's intensity or color is only interesting when attributed to a particular geometric coordinate in the image. Geometry is also the means of relating the 3-d world to the 2-d image as described in the previous section. Therefore, all the primitive features computed from an images are geometric quantities, optionally with other descriptive information attached.

Geometric structures can be classified by their intrinsic and extrinsic dimension. The extrinsic dimension is the dimension of the embedding space (*e.g.* the image plane). Thus, all geometric structures in a 2-d image have an extrinsic dimension of two. The intrinsic dimension is the number of variables needed to access all points on the structure. For extrinsic dimension M , the intrinsic dimension N must satisfy $0 \leq N \leq M$. Therefore, all geometric structures in the image plane must have intrinsic dimension of zero, one, or two. These correspond respectively to points, curves, and regions. Figure 2.4 shows examples of points, curves, and regions extracted from the image of vehicles in Figure 2.1.



Figure 2.4: Geometric features of varying intrinsic dimension. Clockwise from upper left: the original image, detected points (Harris corners), detected curves (sub-pixel Canny edges), detected regions (Stauffer-Grimson foreground regions).

There are many ways to obtain and use each of these types of structures for vehicle representation. Simple geometric features can be obtained directly from the image data. This task is called

feature detection. Usually features are detected using very simple constraints that apply to almost all natural images. For example, rapid changes in intensity tend to be of interest. Geometric structures can also be predetermined using prior knowledge to create a model. However, predetermined geometry is often compared to detected features rather than to the image itself. So feature detection is employed in some form in the use of almost all vehicle representations.

Related to feature detection is the separation of features of interest from the background features. This task is called **segmentation**. When the intrinsic dimension is less than the extrinsic dimension (*i.e.* points and curves) the geometric structures are sparse. That is, in 2-d, they have no area and can never cover the entire space. In these cases, segmentation is usually an agglomerative post-process to detection in which features are assigned to groups that have some semantic meaning (like points belonging to a vehicle). When the intrinsic dimension equals the extrinsic dimension (*i.e.* regions) the geometric features are dense. Regions have area and can potentially cover the entire (finitely bounded) image space. In this case, segmentation may be synonymous with detection. The partitioning of the image into various regions of interest is called segmentation, and this partitioning is also how the regions are “detected”. However, there may be need for additional partitioning or agglomeration of primitive regions into regions of interest with semantic meaning. The term *segmentation* is context dependent and must be clarified by specifying the type of object to be segmented from the image.

It is helpful to lay the groundwork for vehicle representations by discussing some of the common feature detection and segmentation algorithms that have been used in conjunction with vehicle representations.

2.2.1 Point features

Points features (sometimes called interest points) are image features with an intrinsic dimension of zero. They have zero degrees of freedom and can be precisely located. Points are usually detected in areas where the intensity (or color) of an image is rapidly changing in both the vertical and horizontal directions at the same time. One type of interest point is called a corner. Most corner detectors are derived from an algorithm that is now called Harris corners [42]. A corner occurs where the image gradients are maximal in both directions. The other common interest point is the blob. Blobs are peaks in the intensity rather than peaks in gradient; they are often detected with the Laplacian of Gaussian (LoG) operator.

Figure 2.5 shows an example of detected Harris corners on an image of vehicles. Notice that most of the image is made up of uniform, untextured areas, so few points are found. Of those that are found on the vehicles, only a small number are rigidly attached to the vehicle. In other words, many of the points are generated by reflections of the environment or interactions between the boundary of the vehicle and its background. These point do not appear at the same position on the vehicle as it moves and will not be useful in tracking. Points generated by blob detectors have similar properties.

Point features on objects like vehicles can be segmented from the background using a number of criteria. One simple criterion is proximity. Points that are nearby each other are likely to belong to



Figure 2.5: Harris corners detected and overlaid on the vehicle image from Figure 2.1.

the same object. Another criterion is common motion. Points that move consistently according to a motion model are likely to belong to the same object. One last criterion is model agreement. Points can be verified against a prior model of the image appearance in the neighborhood of the point or against a model of the geometric relations between a collection of points. Often a combination of these criteria are used for object segmentation.

A feature descriptor is frequently used conjunction with a feature detector for tracking and matching points across video frames or multiple views. A descriptor is a numerical representation of the image appearance in a local neighborhood around a feature. Effectively each point is associated with a small region. The descriptors are similar, according to some metric, when two points represent the same location on the underlying object. The Scale-Invariant Feature Transform (SIFT) [73] is a popular example of a blob detector with associated feature descriptor.

Another common trait of SIFT and other point detector/descriptors is scale invariance. The image is smoothed and down-sampled repeatedly to form a pyramid of increasingly smaller images. Each image in the pyramid is associated to a different *scale*. Points are detected at each scale. In some detectors, when the same point appears at multiple scales, a single identifiable scale may be chosen. Multi-scale processing is useful when tracking object in perspective views. As the white vehicle in Figure 2.5 approaches the intersection, its apparent size shrinks to that of the black vehicle. Without multi-scale processing, a fixed-sized descriptor region around each point covers a much greater area on the vehicle as the vehicle moves away from the camera. As a result, the descriptors would likely become quite dissimilar.

The literature on interest points and descriptors is too vast to review here. The reader is referred to some detailed studies on interest point detectors [89] and descriptors [78] for further information. Section 6.1.2 also provides some more detail on a specific implementation of interest point tracking that applies to the vehicle tracking algorithm in this thesis.

2.2.2 Curve features

Curves are image features with an intrinsic dimension of one. Since curves have one degree of freedom they can be precisely located, locally, in only one direction. Curves are detected in an image as discrete curve sections called **edges**. Each edge is a local, oriented line segment spanning one pixel in length. After detection, edges can be linked to form curves by following the tangent direction of each edge to the neighboring edge. Some algorithms work directly with the edge primitives and do not make use of curves as continuous 1-d structures.

Edges are typically detected as gradient maxima in one, but not both, image directions. Variations of the Canny edge detector [15] are commonly used to detect them; though, as with points, there is a vast literature on alternatives. Some alternatives detect ridges and valleys in intensities (analogous to blob points), but these are seldom used with vehicle surveillance. The inability to uniquely correspond edges using only local information is called the **aperture problem**. An edge in one image matches equally well to a number of edges along a curve in another image. This problem makes edges more challenging to work with than points. The payoff comes from the number of detected edges being much greater than the number of points (as is evident in Figure 2.6). Edges capture a great deal of the information in a vehicle image.

Figure 2.6 shows an example of a sub-pixel variation of the Canny edge detector with the results linked into curves. Notice that the curves provide a much more descriptive representation of the scene than points. In fact, people can easily recognize vehicles from the curves of an image alone but not so easily from the points. This is more apparent in Figure 2.4 where the image intensities are not shown under the geometry.

Segmentation of curves on a vehicle from those in the background involves criteria analogous to those used for points: proximity, common motion, and model agreement. However, matching or tracking curves is problematic. One cannot match individual edges because of the ambiguity induced by the aperture problem. However, to use linked curves as the atomic units for tracking one must be able to consistently detect them. While the detection of edges is relatively stable, the linking process is much less so. A small amount of noise or outlier edges can dramatically alter the connectivity of curves. Linking can even connect some curves on the vehicle to other curves in the background as can be seen in a close inspection of Figure 2.6.

2.2.3 Region features

Regions are feature that have the maximal image space intrinsic dimension of two. They densely cover a portion of the image with measurable area. The smallest meaningful unit of area in a digital



Figure 2.6: Curves detected and overlaid on the vehicle image from Figure 2.1. Edge are detected with a variation of the Canny detector, estimated to sub-pixel location with parabolic interpolation, and linked into curves.

image is the pixel. Large area regions detected in an image are most frequently created by assigning discrete pixels to a common group. When every pixel in the image is assigned to some group, the image is said to be segmented into regions. Grouped pixels need not be spatially connected. A connected region means that for any two pixels in the region there exists a path through a chain of adjacent pixels such that all pixels in the path are also in the region. A pixel is adjacent to another if it is one step away vertically or horizontally (or in some definitions, diagonally). Spatial coherence is often a desirable trait of regions. So, when a region contains multiple disconnected components, these regions can be further divided into connected components if desired.

The segmentation of a still image into meaningful regions has been a well-studied problem in computer vision. The image is broken into regions with coherent properties like proximity, color, intensity, or texture. When these generic regions are kept relatively small and numerous (*i.e.* over-segmentation) they are denoted *super pixels*. Super pixels are a useful input to computationally intensive algorithms that would otherwise have to operate on the much larger number of individual pixels. Generally, image segmentation algorithms have much higher time complexity and take much longer to compute than most point or edge detection algorithms. This has made them much less suitable to video processing, especially when real-time performance is a goal. More recently, Felzenszwalb and Huttenlocher [29] produced a good segmentation algorithm with only $O(n \log n)$ time complexity for n pixels. Even so, tracking super pixels across video frames suffers from the same

problems as tracking curves. Due to image noise and the changing appearance of moving objects, it is unlikely that each super pixel will uniquely corresponded to a super pixel detected in a later frame.

Rather than deal with the complexities of spacial segmentation, vehicle surveillance applications tend to segment regions based on motion using temporal information in video. When dealing with video, if the camera is stationary then each pixel will corresponded to the same piece of a static scene over time. Therefore each pixel can be examined temporally and independently from all other pixels. Detecting changes in intensity over time allows each pixel to be classified into exactly two regions representing foreground (moving objects) and background (static objects). Figure 2.7 gives an example of foreground/background (FG/BG) segmentation for the familiar two vehicle image. To improve spatial coherence, detections of a small number of connected pixels of one type, isolated within the other type, can be removed with a post processing step based on mathematical morphology.

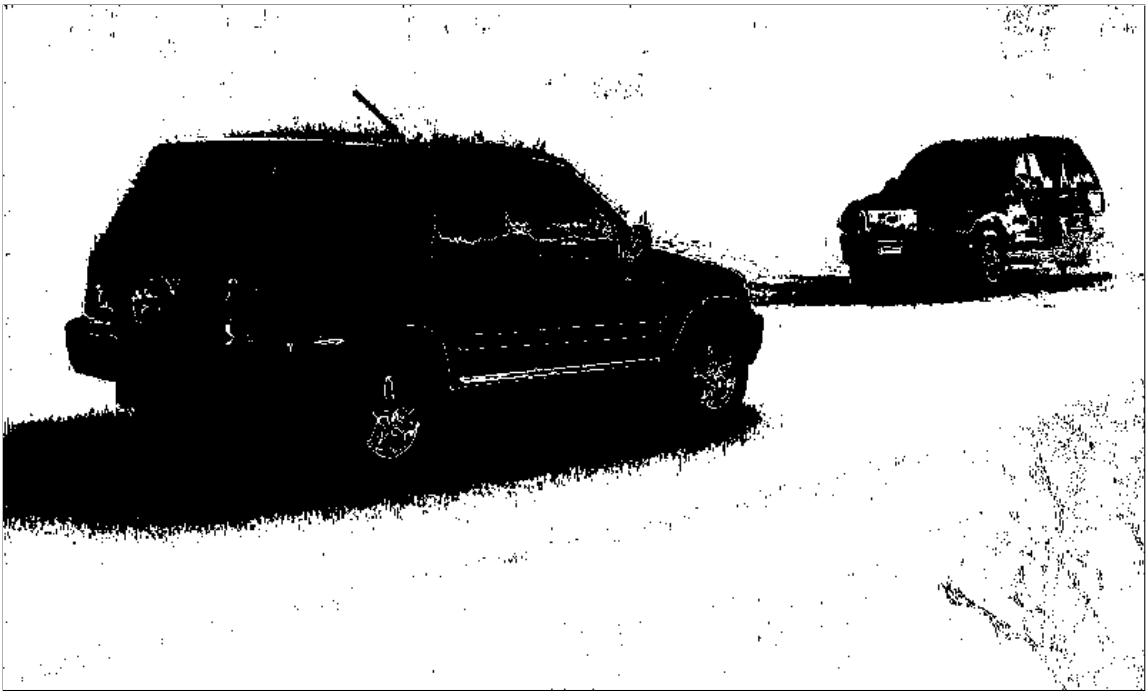


Figure 2.7: Segmentation of foreground and background in the video frame from Figure 2.1. Black pixels are foreground; white pixels are background. Segmentation uses a variation of the Stauffer-Grimson algorithm and no morphological post processing.

To produce FG/BG segmentation for vehicle surveillance, most early algorithms used frame differencing (as in [21]) or a static background image subtraction (as in [28]). Frame differencing thresholds the difference between pixel intensities of adjacent video frames, while background subtraction thresholds the difference between each frame and a common background image containing

no vehicles. These methods are efficient to compute, but both have serious flaws. Frame differencing fails to segment vehicles cleanly since the foreground pixels are generated at the union of the vehicle’s positions in both frames. Additionally, large uniform regions on a slow moving vehicle (or high frame rate video) will overlap in adjacent frames resulting in false background detection. Static background images are often impossible to obtain in high traffic areas, and they do not adapt to gradual changes in scene illumination (*e.g.* changes in sun angle or cloud cover).

Most modern algorithms instead use an adaptive statistical model, often derived from the work of Stauffer and Grimson [98]. The adaptive model in [98] uses a mixture of Gaussians (MoG) at each pixel to learn and adapt the background appearance in an online process. Foreground is detected as pixels that have low probability of belonging to the background distribution. Section 6.1.1 discusses the details of background modeling and morphological post processing in the context of the vehicle tracking algorithm used in this thesis.

2.3 Geometric vehicle models

There are several ways to use geometric information to model the appearance and motion of vehicles. The vehicle representations in the literature of vision-based vehicle surveillance are quite diverse. Each representation leads to a different algorithm for vehicle surveillance, and each has its strengths and weaknesses. These representations are used to solve the problems of vehicle detection, tracking, shape recovery, and recognition. In video surveillance, vehicle detection is the sub-problem of tracking that handles initialization. Some researchers address vehicle detection in still images—a much more challenging problem that is beyond the scope of this thesis.

This section reviews the literature by classifying the vehicle representations into two categories: agglomerative models, and prior models. These approaches vary in how much prior knowledge is used and at what stage of processing the knowledge is applied. Agglomerative models start with feature detection. The primitive features are detected with no prior knowledge about the problem at hand. These features are then segmented into clusters representing each vehicle, usually using some prior vehicle knowledge at this stage. The final vehicle representation is simply a cluster of primitive features. Alternatively, prior models start with a preconceived notion of what features to look for based on vehicle knowledge that is learned from data, imparted by humans, or both. In using prior models, the goal is to relate the existing model to the image data by finding correspondence and estimating model parameters.

Both agglomerative and prior models can be further categorized into 2-d and 3-d models. The 2-d models represent geometry in the image plane. Occasionally the 2-d geometry is mapped onto a world plane, usually the ground plane, to make calculations in world coordinates. The mapping between the image plane and world plane is bijective (one-to-one correspondence), and the geometry is still embedded in a 2-d plane, even if that 2-d plane is a subspace of the 3-d world. On the other hand, 3-d models represent geometry in 3-d world coordinates. The world coordinates are mapped to image coordinates by projection (Section 2.1). This mapping is surjective (many-to-one). A single

image does not constrain the 3-d geometry along the depth direction; addition information is needed in the form of prior knowledge or multiple observations (*e.g.* multiple cameras or video).

2.3.1 Agglomerative models

Agglomerative models are often favored because they are simple and general. The agglomeration of low-level features allow just about any vehicle to be modeled with the same algorithm. This level of generality is suitable for tracking vehicles, but much less helpful for recognition or shape recovery. Recognition using an agglomerative model is usually limited to classification based on overall statistics like total size, average color, or frequency of texture. No agglomerative vehicle representation has been able to recover a dense 3-d shape model that would be suitable for the applications described in Sections 1.1.6 and 1.1.7. Nonetheless, many agglomerative models have been used in the literature, primarily for vehicle tracking and counting applications.

Point features are commonly used in agglomerative vehicle representations. Beymer *et al.* [6] created one of the first real-time vehicle tracking systems using point features. The points, which are Harris corners, are mapped from the image plane onto the ground plane by homography. Tracking is accomplished in the world coordinates of the ground plane. A patch of intensities around each point is used as a descriptor, and tracked points are matched to points in the next frame by correlation of these descriptors. Multi-vehicle clusters of points are formed by an over-grouping segmentation based on proximity alone. These large clusters are later segmented into individual vehicle clusters by observing subtle differences in relative motion of points over time. This method, being inherently 2-d, is only viable in situations where the 3-d aspects of the problem are minimal. Ideal conditions are a straight section of road (so vehicles do not rotate), observation from a far distance (so projection is nearly affine), and viewing the vehicles from above (so distortion of vehicles is minimal when mapped into the ground plane).

The work of Kanhere and Birchfield [55] extends the point-based model into 3-d to loosen the constraints imposed by 2-d tracking. The point features are still corners, though they are now tracked in the image space using optical flow. Instead of mapping points into the ground plane, the points are back projected into 3-d at each frame. Back projection from 2-d to 3-d is ill-defined in general, but Kanhere and Birchfield apply an additional constraint in a method called plumb line projection. Plumb line projection uses a FG/BG region segmentation to constrain the height of a back projected point. A second point is found by tracing downward in the image from the feature point until intersecting the foreground boundary. This boundary point is assumed to correspond to a point on the ground plane that is vertically below the back projected feature point in the 3-d world. The boundary point constrains the height of the back projection as illustrated in Figure 2.8. The accuracy of plumb line projection is compromised by frequent errors in finding the correct ground plane point. However, the errors are shown to be small compared to the errors in projecting all points onto the ground plane, as in the 2-d approach.

Another agglomerative representation that joins points with FG/BG region segmentation is found in [18]. In this work, Choi *et al.* represent vehicles with a combination of SIFT feature points and

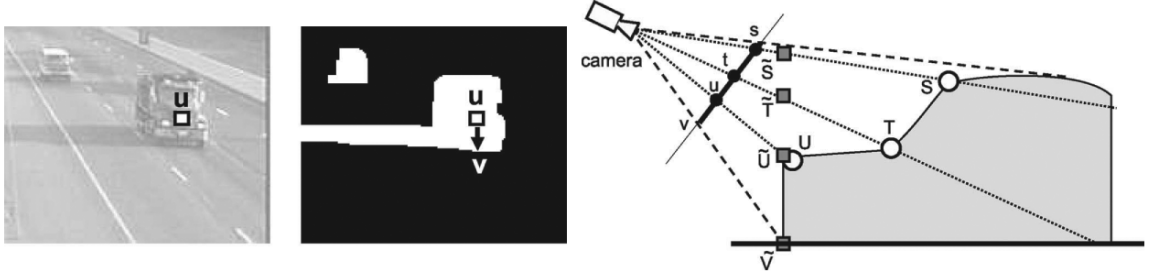


Figure 2.8: The plumb line projection method. Point v is the point on the region boundary below point u . Point u reconstructs accurately in this case, but other points s and t would not be so lucky. Images adapted from Figures 5 and 6 of [55].

connected components of foreground regions from background subtraction. Here the regions play a more active role in the vehicle representation. The regions determine where points are detected, and the regions are used to segment points into vehicle groups. While this method is image-based and purely 2-d, the use of SIFT features for tracking allows for scale invariance as vehicle approaches the camera. Figure 2.9 shows how both point and region features are used in the vehicle representation.

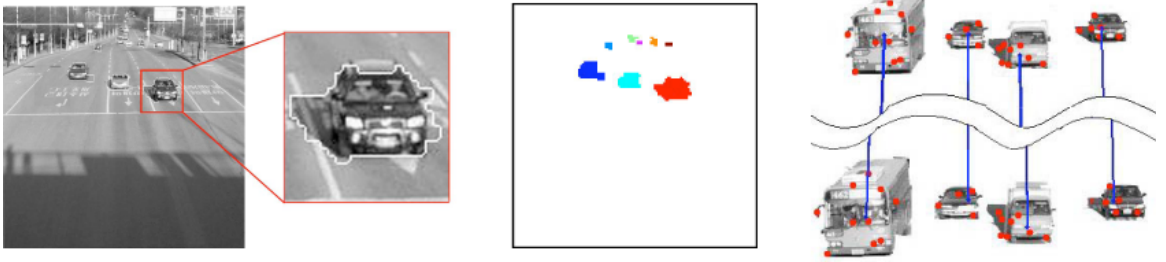


Figure 2.9: An agglomerative representation with both regions and points. Foreground regions are segmented from background, then further segmented into connected components. SIFT features are detected in each region, then matched to points in regions in later frames. Images adapted from Figures 3 and 6 of [18].

In a similar way, edges have been combined with FG/BG region segmentation for a joint representation. Unlike with points, the edges are not usually tracked because of the aperture problem. Instead, they are used in a statistical manner. Fathy and Siyal [28] used the presence of vehicle edges in an early attempt at creating a real-time vehicle detection system. In this work, background subtraction is used with edge detection to identify foreground edges. The presence of foreground edges in designated detection regions of the image is used to count vehicles, but no attempt is made to track specific edge features from frame to frame.

Ha *et al.* [38] use a similar technique of combining background subtraction and edge detection. In this case, the number of foreground pixels and the number of edge features in a detection region are fed into a neural network. The neural network uses the density of foreground edge features to distinguish between a moving vehicle and a shadow cast by a moving vehicle. Cast shadows on

a roadway ordinarily contain little texture and very few edges compared to the vehicle. Yet, cast shadows have darker intensity than background and are classified as foreground regions. This effect is visible in Figures 2.8 and 2.9. Shadow in the foreground detection is often problematic, and edge density is a useful cue in identifying it. But again, the edges are not tracked from frame to frame.

There are some examples of representations that actually involve tracked vehicle edges. Jain *et al.* [50, 52] track curves detected on vehicles in 2-d. Curves are formed by detecting Canny edges, applying sub-pixel interpolation, and linking edges into longer curves. Curves, rather than individual edges, are matched from one frame to the next. The shape of the curves provides one constraint to overcome the aperture problem. An assumption that the vehicle direction of motion is known also provides a constraint. Shape deformations due to 3-d projection are handled by a 2-d elastic matching process. After curves are matched, the vehicle is segmented from the background by finding curves that under go similar transformations. The segmented curves in the right of Figure 2.10, illustrate both the aperture problem and ambiguous linking. The long background curve sticking out of the front of the vehicle was misclassified for two reasons: one, the curve is tangent to the direction of vehicle motion so its velocity is indeterminate (aperture problem); two, the curve was mistakenly linked to a valid curve on the vehicle.



Figure 2.10: Tracked 2-d curves. The left two images show matching curves (color coded by match). The right most image shows vehicle curves segmented from the background by common motion. Images adapted from Figures 11 and 13 of [52].

In earlier work by the author, Leotta and Mundy [69] take a similar approach to Jain *et al.*, but instead model the curves in 3-d. The same sub-pixel edge detection, edge linking, and motion constraints are used. Instead of matching curves by 2-d elastic deformation, curves are matched based on the likelihood that a single 3-d curve explains the 2-d curves by perspective projection. Curves tracked over several frames build up a 3-d curve model by triangulation. Consistent motion is also used here to segment vehicle curves from the background as shown in Figure 2.11. The resulting 3-d curves are used for vehicle recognition in the work of Han *et al.* [40].

Other agglomerative vehicle representations use regions alone without any lower dimensional features. Gupte *et al.* [37] use foreground regions segmented with an adaptive background model. Regions are broken into connected components, but each connected region is not yet assumed to be a vehicle. Realistically, a region could easily represent multiple vehicles in close proximity or a portion of a vehicle due to occlusion or noise. The regions are tracked from frame to frame based on overlap with the previous frame. As regions merge or split in some frames they are grouped

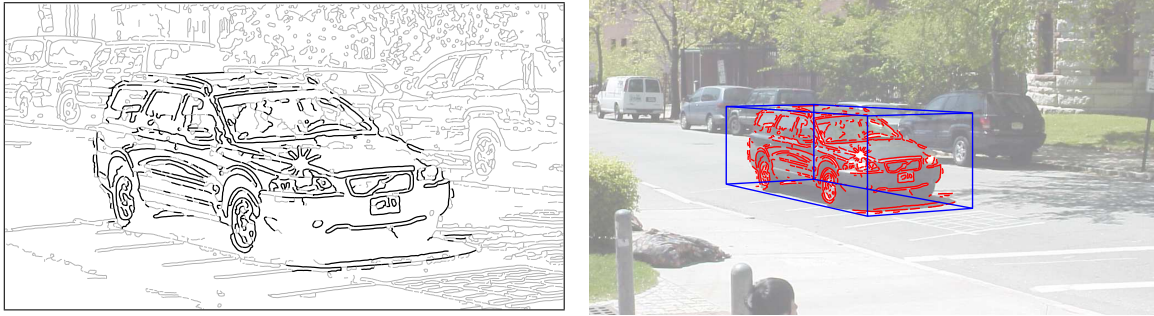


Figure 2.11: Tracked 3-d curves. On the left, tracked 3-d vehicle curves are segmented from the background based on motion. On the right, the same 3-d curves are shown with a 3-d bounding box. Images adapted from Figure 5 of [69].

into a graph structure. Vehicles are later segmented using graph connectivity and classified as car or non-car based on bounding box size. Atev *et al.* [4] expand upon these ideas and use multiple camera views to reduce ambiguities and to create a 3-d bounding box around each vehicle. The vehicle bounding boxes are tracked in 3-d for collision prediction.

Bouttefroy *et al.* [12] use an adaptive MoG background model. Rather than explicitly handle merging and dividing regions, they employ the mean-shift algorithm to cluster the foreground pixels and locate cluster centroids. It is these centroids that are tracked. Vehicles are assumed to be separated well enough for initialization. Magee [75] provides another use of an adaptive MoG model in vehicle representation. In addition to the background model, a separate MoG is used to model the distribution of colors in each connected foreground region.

Koller *et al.* [61] initially detect vehicle regions using an adaptive background model. However, the representation of the vehicle region is then converted from implicit to explicit by fitting the region boundary with a closed cubic spline using 12 control points. The boundary contour is tracked in future frames using the method known as snakes or active contours. Occlusion reasoning is performed by assigning a depth ordering to the 2-d targets. Contours closest to the bottom of the image are assumed to be closest to the camera. This approach also requires vehicles to be well separated for initializations. Figure 2.12 shows an example of these region contours and their motion.

Kato *et al.* [56] use a different kind of background model to address the problem of cast shadow misclassification. A Hidden Markov Model (HMM) is used at each pixel to learn a background model that can classify pixels as vehicle, background, or cast shadow. Unfortunately, this HMM is learned on a fixed sequence of video and is not adapted over time. Tracking of the foreground vehicle regions is not addressed in their work.

2.3.2 Prior models

Unlike agglomerative models, prior vehicle models exist before the data is observed. Rather than construct the model from observations, prior models must be aligned with the data by varying model parameters. The agglomerative models rely on very little prior knowledge about vehicles, so

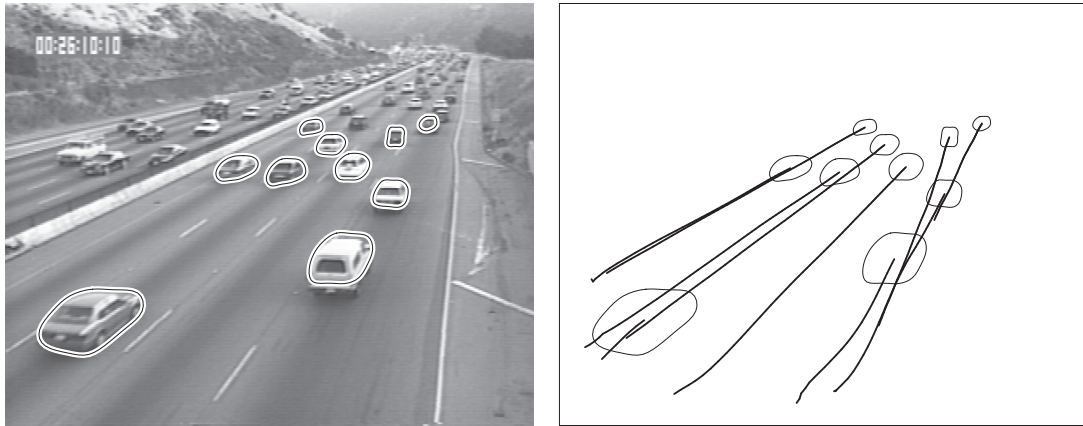


Figure 2.12: Tracked 2-d region boundaries with snakes. On the left, vehicles with their detected region boundaries. On the right, vehicle tracks over 80 frames. Images adapted from Figure 3 of [61].

the same algorithms often apply to almost any moving object with little modification. However, the lack of prior knowledge makes it difficult to recover semantically meaningful vehicle information without extensive post processing on the collection of primitive features. Prior models, on the other hand, focus on a more narrow range of targets, but they are capable of producing a much more meaningful description of those targets. With an agglomerative model it is very difficult to answer questions like “what is the wheelbase (the distance between the two axles)?” and “what is the surface area of the headlights?”. With a prior model that is detailed enough, answering these questions is trivial. As a result, vehicle recognition based on these properties is also trivial. Prior models can also make inferences about parts of the vehicle that are never actually observed by relying prior knowledge like the symmetry of vehicles. Additionally, this prior knowledge can help overcome noisy estimations of depth that plague 3-d features in agglomerative models when the viewpoint baseline is small. Consequently, vehicle shape recovered with a prior model is less noisy and more suitable for rendering in the applications discussed in Sections 1.1.6 and 1.1.7.

In many cases, but not all, a prior model for vehicle representation is a generative statistical model. That is, for many prior vehicle models you can generate samples of vehicle appearance by drawing randomly from a distribution. Other prior models are discriminative. A discriminative model can only classify data based on image observations, but it cannot predict how the classified object might appear in another image. This limits discriminative models to tasks like vehicle detection (classification of a hypothesis as vehicle or non-vehicle) and vehicle recognition (classification as one type of vehicle or another).

As with agglomerative models, several different geometric representations are used for prior models, both in 2-d and 3-d. For the purposes of this thesis, an effective prior model is one that is 3-d, generative, and detailed, yet also generic. A 3-d model is needed to account for varying

viewpoints, and a generative model is needed so that the same model can be used for tracking and shape recovery as well as detection and recognition. A generative model also allows for rendering the recovered vehicle from new viewpoints for visualization. The more detail represented in the model the better the visualization. A more detailed model also takes advantage more information in higher resolution images. More details in the model provide more measurements upon which recognition is based. At the same time, however, an effective model needs to be generic. It should balance strong prior constraints and a high level of detail with the need to generalize to wide variety of vehicles. Too many constraints result in a specific model with limited applicability. Too few result in a vague model lacking in semantic meaning. A compromise is needed. The remainder of this chapter reviews the literature of prior vehicle models. Each prior model addresses these needs to some degree, but none are entirely satisfactory.

Some prior models are based on feature points. These prior models are typically discriminative and are used primarily for vehicle detection in still images. Leibe *et al.* [67] use a prior model consisting of a point descriptor “code book” and an implicit shape model. The model is matched to an image in two stages. First, feature points are detected and assigned descriptors (various detectors and descriptors are considered). Each descriptor is matched against a code book of known vehicle descriptors. If matched, the code book generates one or more hypothesis about what part of a vehicle that point is likely to have come from. Second, each of these hypotheses generates a vote of where the vehicle center should be located relative to that feature point. While there are many wrong hypotheses, the correct hypotheses magnify each other by voting for a common center location. Strong peaks in the voting space translate into vehicle detections. Figure 2.13 shows the progression of these stages to generate a vehicle detection. This prior model for vehicle representation exists in the 2-d image space and assumes that the vehicle is observed from a known viewing direction. The code book itself and the center offsets (*i.e.* implicit shape) are both learned from real training images in which the vehicles have been manually labeled. The learning process is much like agglomerative modeling and it generalizes to many classes of objects. However, once the model is trained on a particular class of object (like passenger vehicles), it becomes a specialized prior model of objects in that class.



Figure 2.13: Code book vehicle detection. Points with descriptors are detected, matched to a code book, and then vote for vehicle center. Images adapted from Figure 5 of [67].

Liebelt *et al.* [70] extend the method of Leibe *et al.* into 3-d to detect the presence and 3-d pose of vehicles in a single image from arbitrary viewpoints. Their code book is a mapping from feature descriptors to 3-d feature points on the surface of a vehicle. The code book is built from a collection

of synthetic images of vehicles with a variety of viewing directions. Computer Aided Design (CAD) 3-d models are used to render the many training images from different viewpoints. The training images are processed to recover local feature points with descriptors. In this case, the feature points are Hessian-Affine, a multi-scale blob detector with a feature descriptor that is affine invariant. The point features are mapped back into 3-d by projecting onto the CAD model surface. The same point features are detected in real images and code book matching is used to find the 3-d pose of vehicles.

Other algorithms use a prior model with explicit geometry. Mundy and Chang [79] track specific vehicle instances using an image representation learned in one frame of video. A region containing the vehicle is selected from one frame of video either manually or by some segmentation process. That region and its image content become the representation for matching in later frames. The image content can consist of multiple modalities of information including: intensity, gradient, color, and polarization. Mutual information is used to fuse these data modalities. Random hypotheses of 2-d transformations are generated to map the region location in the previous frame to the current frame. The transformation is selected that produces the highest mutual information between the model and the region in the current frame. Since this representation is 2-d, the matching will start to break down as the vehicle undergoes 3-d changes not modeled by the space of 2-d transformations.

Many other explicit prior models are anchored to images by matching to detected edges. Detected edges are more informative than points and more robust to illumination changes than regions derived from intensity. Leuven *et al.* [107] use a simple collection of 2-d line segments (called a *template*) as a vehicle representation. Their 2-d model represents vehicle as seen from the front or back. Edges are detected in each frame of video and a distance transform is applied. A distance transform pre-computes the distance to the nearest edge at each pixel location. The template is matched to the image by finding a 2-d transformation that minimizes the average distance of the template lines to the image edges. Edge distance is computed efficiently using the distance transform. Figure 2.14 gives a visual overview of the process.

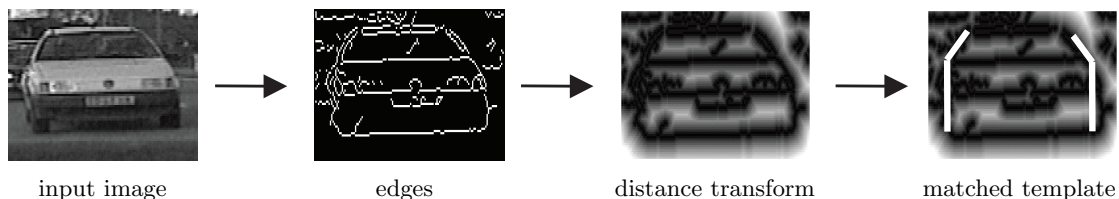


Figure 2.14: Template matching using the distance transform. Images adapted from Figure 1 of [107].

Instead of using an arbitrary collection of lines in a template model, Dubuisson Jolly *et al.* [26] use lines that form the boundary of a 2-d region. Parameterized, irregular polygons are used to model the side view of vehicles (See Figure 2.15.). A different polygon (with either 7 or 8 vertices) represents each of five different classes of vehicles. One of the biggest challenges in using prior models for vehicle surveillance is having a model that is suitable for every observed vehicle instance. It is clearly intractable to create a separate shape model for all possible vehicles. However, it is much

more feasible to create a generic vehicle model with parameters that allow it to deform to match the shape of many vehicles. The polygon vehicle model does just that. Each vehicle class model has a series of manually defined constraints on the shape parameters. Furthermore, a probability distribution on the shape parameters defines the *a priori* likelihood of each shape. This generative model is used for vehicle detection and classification. Simulated annealing is applied to a Gibbs distribution with two likelihood terms. The first attains a maximum when the region contains all foreground pixels (obtained from frame differencing). The second attains a maximum when all lines in the contour align, both in position and orientation, with strong image gradients. As with all the 2-d prior models discussed thus far, this model is only applicable when the vehicle is observed from a particular viewpoint.

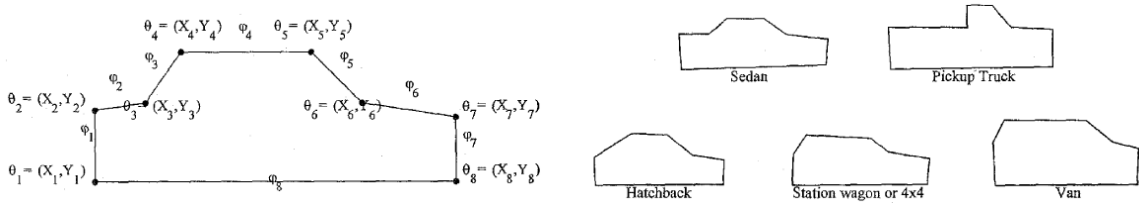


Figure 2.15: Deformable polygon model of vehicle sides. Images adapted from Figures 3 and 4 of [26].

The logical extension of the 2-d polygonal model is the 3-d polyhedral model. A polyhedron or other volume representation has the maximal intrinsic dimension of three in a 3-d space. Analogous to a boundary curve representing a region, the polyhedra is often represented by its boundary surface, a 2-d geometric structure embedded in 3-d. The boundary surface of a polyhedron is represented as a closed, 3-d, polygonal mesh. A polygonal mesh is simply a collection of 2-d polygons, each embedded in 3-d in its own orientation, and each sharing its sides with other polygons to form a connected surface. Since 3-d measurements are not directly available in an image, a surface model requires multiple views, prior knowledge, or both to constrain its shape.

3-d surface models in computer vision extend all the way back to the early 1960's with Lawrence Roberts' Ph.D. thesis [86] on machine perception of 3-d solids using composition of primitive polyhedra. Surprisingly, the 3-d surface models used in vehicle representations today still have much in common with that original work. Almost all 3-d vehicle representations, including the one used in this thesis, use polygonal meshes. The compositional aspect of Roberts' work is not appropriate for vehicles, but the relation of 3-d models to images using intensity edges is still highly applicable. There are many alternative models of 3-d shape used in other areas of computer vision and graphics. A few examples are implicit surfaces, generalized cylinders, shock scaffolds, and voxel arrays. None of these has proven to be more useful in vehicle surveillance than the polyhedral mesh.

The 3-d analog of the polygon model in Figure 2.15 (for sedans only) was proposed in 1991 by Shick and Dickmanns [87]. Their generic model is a 12 sided polyhedron (an irregular dodecahedron). Constraints of symmetry and parallelism are applied to reduce the number shape parameters to 12. This representation was only evaluated for tracking in synthetic videos.

A short time later Koller *et al.* [59, 60] proposed a very similar polyhedral model. It also has 12 faces and 12 parameters with similar constraints (Figure 2.16 left). They did, however, show that the same polyhedral model can represent various classes of vehicles like sedan, hatchback, station wagon, minivan, and pickup (Figure 2.16 right). Unlike the 2-d polygon model, the same topology is used for all vehicle classes. Koller *et al.* also showed working examples of tracking on real video. Projected polyhedral line segments are matched to line segments fit to the image intensity edges.

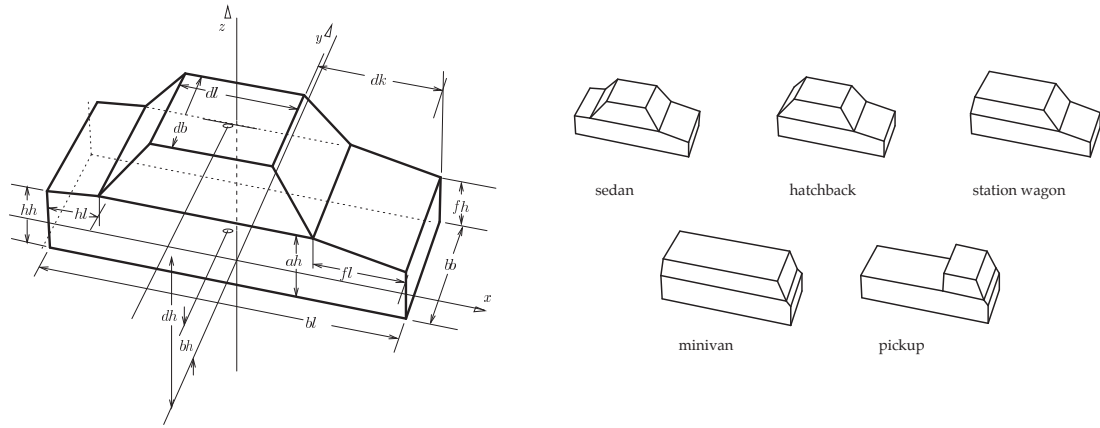


Figure 2.16: The generic polyhedral vehicle model proposed in Koller *et al.* Images adapted from Figures 1 and 2 of [59].

Many other authors have chosen to use this simple vehicle model for vehicle detection, pose estimation, and tracking. In many cases the shape parameters are fixed and a collection of fixed models represent various vehicle classes. Lou *et al.* [72] track a fixed vehicle model by matching to edges. Kollnig and Nagel [62] estimate the pose of a fixed model directly from image gradients. Pece and Worrall [84] track a fixed model using image gradients. Tan *et al.* [104] estimate the pose of a fixed model by matching to line segments fit to edges, and in later work [103], by matching directly to image gradients. Kim and Malik [57] track vehicles with a probabilistic extension of the fixed model to allow for some variation in edge position. Haag and Nagel [39] use optical flow and edges to track a collection of vehicle models. Ambardekar [1] tracks a fixed collection of wire frame models which are initialized with FG/BG segmentation for position and optical flow for orientation. Schlosser *et al.* [88] use a fixed 3-d model for detection in aerial images. Ghosh and Bhanu [33] estimate the 3-d vehicle model parameters incrementally in video. All of these works use the same dodecahedron model or a slight variation of it.

Ferryman *et al.* [30] use a slightly more complex polygon mesh model shown in Figure 2.17. It adds contours representing the wheel wells and is controlled by 29 parameters. The key contributions of this work are learning the prior distribution on model parameters from real data and reducing the parameter space with principal components analysis (PCA) while increasing the model complexity. The parameters are learned for several example vehicles by fitting the model to multiple frames of

video for each. Fitting is user guided in the learning stage. PCA is applied to the training examples in the parameter space, and only the 6 largest principal components are retained. In later work [49], this model is extended to include a PCA model for the intensities on the surface of the 3-d mesh.

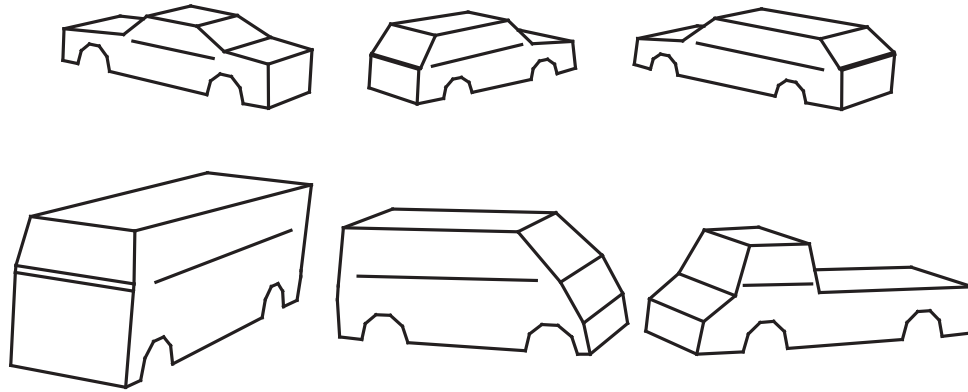


Figure 2.17: The 29 parameter mesh model proposed by Ferryman *et al.* The 6 parameter PCA subspace is only used to model vehicle shapes in the top row. This class of vehicles is referred to as “general cars”. Image adapted from Figure 1 of [30].

More complex vehicle surface models have followed, but all have been fixed in shape. In other earlier work by the author, Leotta and Mundy [68] use a collection of more complex surface meshes (hundreds of faces) to track vehicles while learning an online HMM model for background and shadow. These fixed models are shown in Figure 2.18. This work and many others [39, 57, 59, 60, 62, 88] take advantage of the 3-d surface model to predict shadows cast from the vehicle onto the ground plane. Shadow prediction is straight forward when using 3-d surface models. The sun’s parallel rays and the planar ground act, mathematically, like an affine camera. The shadow is an affine projection of the vehicle model onto the ground plane. Figure 2.19 gives a diagrammatic explanation of this projection, and more details are provided in Section 6.1.3.

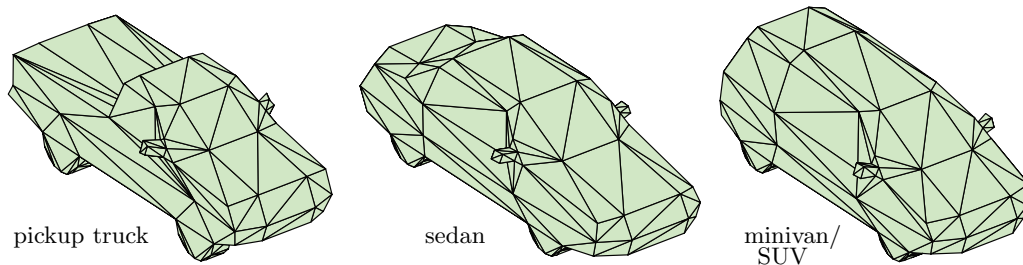


Figure 2.18: The fixed 3-d mesh models used by Leotta and Mundy. Image adapted from Figure 2 of [68].

The most complex surface models used for vehicles are created by Computer Aided Design (CAD). These surface meshes model very fine details of a vehicle using hundreds of thousands or millions of faces. Figure 2.20 shows an example of a CAD model of a sedan. CAD models are frequently

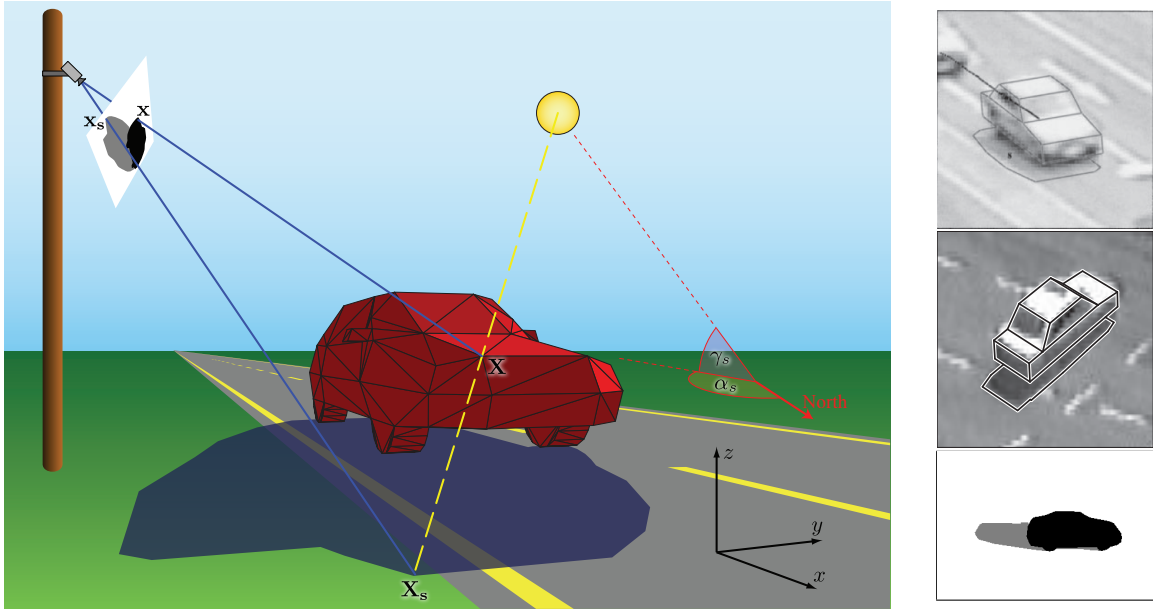


Figure 2.19: Modeling cast shadow with 3-d polygonal mesh models. The diagram on the left shows how shadow is modeled as a projection of the vehicle onto the ground followed by a projection into the camera. The images on the right show examples of this principle in use by various researchers. From the top these images are adapted from [39] Figure 14, [59] Figure 7, and [68] Figure 3.

used for photorealistic rendering (*e.g.* in car advertisements) where this detail is required. Recently CAD models have found their way into computer vision applications for vehicle surveillance. Liebelt *et al.* [70] use them to create training data for their sparse 3-d vehicle model (discussed in earlier). Guo *et al.* [36] use CAD models to predict the appearance of vehicles from different viewpoints in high resolution images. Song and Nevatia [97] use CAD models to create detailed vehicle silhouettes from arbitrary viewing directions. These 2-d silhouettes are used in vehicle detection and tracking.

There are a several driving factors that have spurred the transition to more complex vehicle models. First, the shapes of vehicles are evolving. Manufacturing techniques have improved giving vehicle designers more freedom in creating the shape of vehicle bodies. Vehicle shapes have slowly transitioned away from boxy vehicle shapes and towards more curvy and aerodynamic shapes. The boxy dodecahedral model is much better suited to older boxy vehicles. Second, the resolution of video continues to increase. At low resolution a straight line may still be a good model of a curved vehicle contour. At this scale, pixel quantization smooths away all but the lowest frequencies of curvature. At higher resolutions, however, the details of these curves become more evident and may deviate significantly from a single straight line. In this case, a chain of several line segments better represents the curve. As the image resolution increases, the mesh complexity needs to increase accordingly to maintain the same level of representation accuracy. Third, the processing capability of computers has increased making it more feasible to work with more complex models, even in real-time applications.

The level of detail gained by using CAD models comes at the loss of generality. Each CAD model



Figure 2.20: A highly detailed CAD model of a sedan.

describes exactly one specific vehicle model, and each model is constructed differently by modeling the vehicle's various parts. There is no trivial way to parameterize any one CAD model such that it can deform to match the shape of another. Instead, a collection of CAD models must be used as templates to approximately represent all similar vehicles. The surface model proposed in this thesis aims to overcome this restriction by building a generic, deformable vehicle model that can approximate the level of detail found in CAD models.

Chapter 3

A Deformable Vehicle Model

The vehicle representation proposed in this thesis is a 3-d prior model of vehicle shape and appearance. The purpose of creating this new model is to bridge the gap between the existing dodecahedral and CAD models described in Chapter 2. The dodecahedral models are very generic and can deform in shape to describe a large variety of vehicles spanning many vehicle classes. However, the dodecahedral model only provides a very coarse representation of the vehicle. The CAD models, on the other hand, provide very fine details of vehicle shape. They are also frequently composed of various parts (in the form of disconnected sub-meshes) that represent regions on the vehicle surface that may have different material properties. Examples of parts with different appearance are windows, headlights, taillights, grilles, hubcaps, and license plates. These parts help describe the appearance of the vehicle features that are not due to shape alone. Unfortunately, CAD models are also very specific. Each CAD model represents one vehicle shape corresponding to a specific make (*i.e.* manufacturer), model (*i.e.* product name), and year (*i.e.* first year in a range of years that this style was sold).

The vehicle model described in this thesis extends the generality of the dodecahedral model to the level of detail and part-based description of the CAD models. It is designed to predict the appearance of image edges for the purposes of recovering 3-d vehicle shape and estimating the 3-d vehicle pose. The following properties serve as set of requirements for the proposed vehicle model to meet:

1. **Adaptive mesh complexity:** The number of vertices and faces of the mesh should be variable depending on the problem requirements. A simple mesh allows for faster computation at the expense of representational accuracy. A complex mesh is more accurate, but requires more computation and may even provide more detail than is needed for a given image resolution. Ideally, the model complexity should be allowed to adapt from approximately the level of detail in the dodecahedral model to approximately the level of detail in the CAD models.
2. **Generic 3-d vehicle shape:** The model should be as generic as the dodecahedral model. It should deform in 3-d shape to accurately represent the same diverse set of two axle passenger

vehicles including sedans, pick-up trucks, station wagons, SUVs, and minivans.

3. **Generic 2-d vehicle parts:** The model should describe a collection of 2-d vehicle parts on the surface of the 3-d shape. These parts augment shape information by explaining the appearance of regions on the vehicle that are not due to shape. To fit into the generic framework, these parts must also be consistent across vehicle classes. In addition, these parts must be deformable because they may vary in shape from one vehicle to the next beyond what is explained by 3-d surface deformation.
4. **Edge prediction:** The combination of 3-d shape and 2-d parts should predict the appearance of large number of stable image edges. The edges will drive the estimation of the shape and pose parameters of the model.
5. **Small number of shape parameters:** To make the estimation of the vehicle shape tractable, it must be described by a small number of parameters. Ideally, the number of shape parameters should be variable depending on the problem requirements, just as the mesh complexity is. A smaller number of parameters reduces the computation time at the expense of the model expressiveness. A larger number of parameters leads to a large space of shapes but requires more computation.

Figure 3.1 gives a preview of what the complete model looks like. The rest of this chapter is devoted to explaining the details of the 3-d surface representation, the vehicle parts representation, how the surface and parts combine to predict image edges, and how a small number of parameters is used to describe the deformations. The definition of the vehicle model requires a more formal definition of polygonal meshes. The formal definition of polygonal meshes and related mesh algorithms is given in Section 3.1 before diving into the details of the vehicle representation.

3.1 Polygonal mesh definitions

As alluded to earlier, a polygonal mesh is a collection of polygons with their sides attached to one another. More formally, a polygonal mesh is described by a set of vertices, V , and a set of faces, F . Each **vertex** $v \in V$ is a point (with zero intrinsic dimension) in some space of extrinsic dimension n . All points in V belong to the same n -d space. For meshes used in this thesis, the extrinsic dimension is always either two or three. Each **face** $f \in F$ is a cyclic list of references to a subset vertices in V . In other words, each face specifies the vertices used to form the corners of a polygon. The ordering of vertices in the list is important, but the starting point is not since the list is cyclic. The information contained in F is called the mesh **topology**. It pertains only to how the vertices are connected and is independent of the vertex positions and their extrinsic dimension. The left of Figure 3.2 shows an example of a simple mesh with each face identified by a list of vertices.

For a 3-d mesh, it is desirable if vertices associated with each face are coplanar (*i.e.* lie in a 2-d subspace). When coplanar, it is also desirable that the polygon face be convex. In practice,

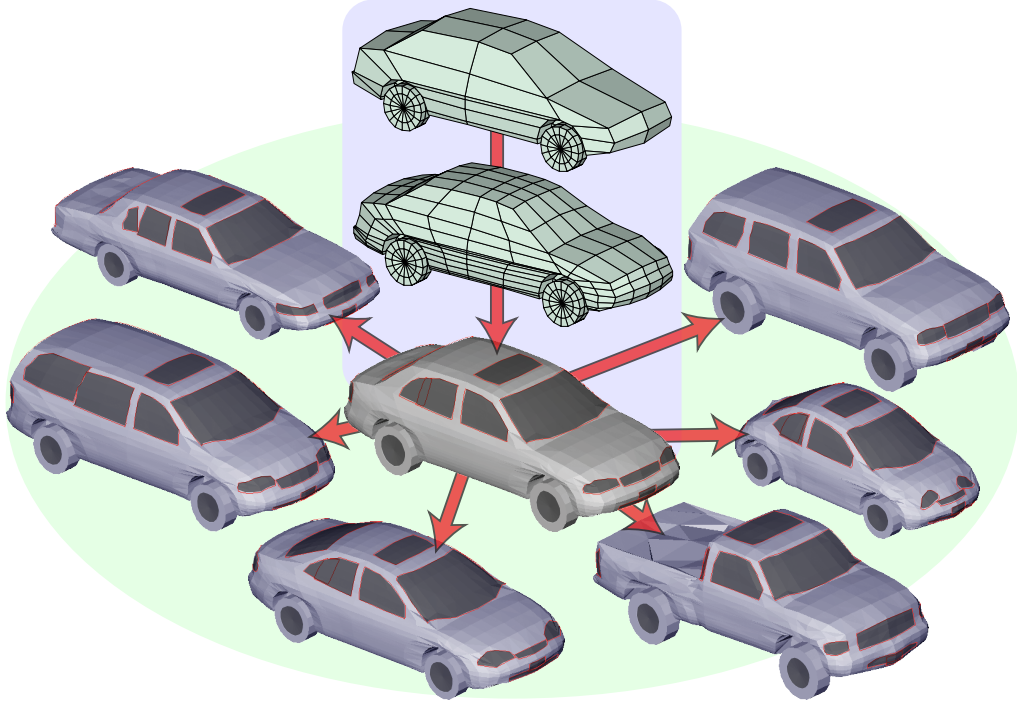


Figure 3.1: A summary of the deformable vehicle model design. The center column shows the mean vehicle shape for the first two mesh resolutions and shape with appearance of parts for the third. The bottom ring depicts a variety of deformations in shape and appearance at the third resolution obtained by varying only 10 model parameters.

these two properties are not always enforced and are only required for certain mesh algorithms. One way of implicitly enforcing these properties is to ensure that each face references exactly three vertices. Such a mesh is called a **triangular mesh** since each face is a triangle. Every triangle is guaranteed to be planar and convex. If, instead, each face references four vertices then the mesh is called a **quadrilateral mesh** and may not satisfy these properties. In general, the polygonal faces may have an arbitrary number vertices which varies from face to face. The meshes used in this thesis are constructed from triangles, quadrilaterals, or a mix of both. Any mesh can be reduced to a triangular mesh by a process called **triangulation**. Triangulation splits each mesh face into triangles. A triangulation of a mesh is not unique since each polygon can be divided in multiple ways. Figure 3.2 shows an example of two different triangulations of the same mesh. All triangulations will retain the same surface geometry as the original mesh when the original mesh consists of planar, convex faces.

A mesh is also said to contain a set of edges, E . Mesh edges are also part of the mesh topology and are derived directly from F . An **edge** is a pair of references to two vertices v_i and v_j in V . The set E of all mesh edges consists of all pairs (v_i, v_j) such that v_i and v_j are found adjacent to each other in at least one face in F . For a general edge, the order in which the adjacent vertices occur does not matter. In other words, edge (v_i, v_j) is equivalent to edge (v_j, v_i) . However, sometimes the

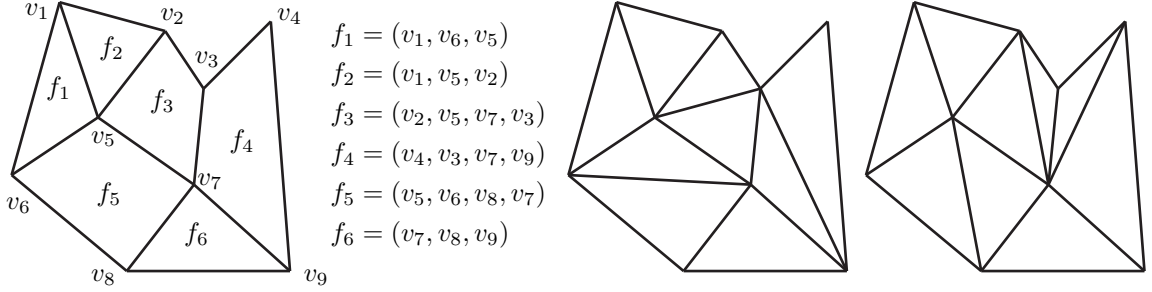


Figure 3.2: A general mesh on the left with 9 labeled vertices and 6 labeled faces. Each face is defined by a cyclic list of vertices. Two different triangulations of the mesh are shown on the right.

order in which the vertices occurs is of interest. A **directed edge** is a restricted type of edge for which (v_i, v_j) and (v_j, v_i) are distinct. The set of edges, E (either directed or undirected), combined with the set of vertices, V , forms a **graph**. The graph of a mesh is a restricted form of a general graph as defined by the graph theoretic branch of mathematics.

There are a basic set of **adjacency** relations between the various mesh structures. An edge and a vertex are adjacent if the edge references the vertex. Likewise, a face and a vertex are adjacent if the faces references the vertex. A face and an edge are adjacent if the face contains the edge (*i.e.* contains the edge's vertices consecutively in the list). Two vertices are adjacent if they are connected by some edge. Two faces are adjacent if they both contain a common (undirected) edge. Two edges are adjacent if they are both contained in a common face and both reference a common vertex. The number of edges adjacent to a vertex is called the **degree** of the vertex.

There are several ways to represent a polygonal mesh in computer memory. The simplest is the **indexed face set**. The indexed face set is an array of vertex coordinates and an array of faces. Each face is simply an array of indices into the vertex coordinates array. This minimal representation closely mirrors the mesh definition given above. It is compact in memory and often used to store a mesh on disk. Unfortunately, it does not explicitly represent edges, nor does it store the adjacency information about a mesh. Edges and adjacency can be computed from an indexed face set, but this requires search through the data arrays. Thus, this representation is very inefficient for many mesh algorithms that need to traverse the mesh by following adjacency relationships.

Several other mesh representations provide a more complete mesh description. There is a trade-off between memory use and computational efficiency. The indexed face set is minimal in memory but increases computation. Alternatively, one could compute and store all adjacency relations for each vertex, edge, and face in a lookup table for maximum efficiency. A good compromise is the half-edge mesh representation. The **half-edge** data structure represents a single directed edge by references to four other adjacent structures: one vertex, one face, and two other half-edges. The half-edges are assumed to come in pairs (hence the name) and are connected to exactly one face. Each half-edge references its pair edge, which is the directed edge in the opposite direction and connected to different face. A half-edge combined with its pair represents one undirected mesh edge. The other referenced half-edge is the next half-edge encountered when traversing counterclockwise

around the referenced face. The referenced vertex is the vertex at the tail of the directed edge. Figure 3.3 illustrates half-edges with an example.

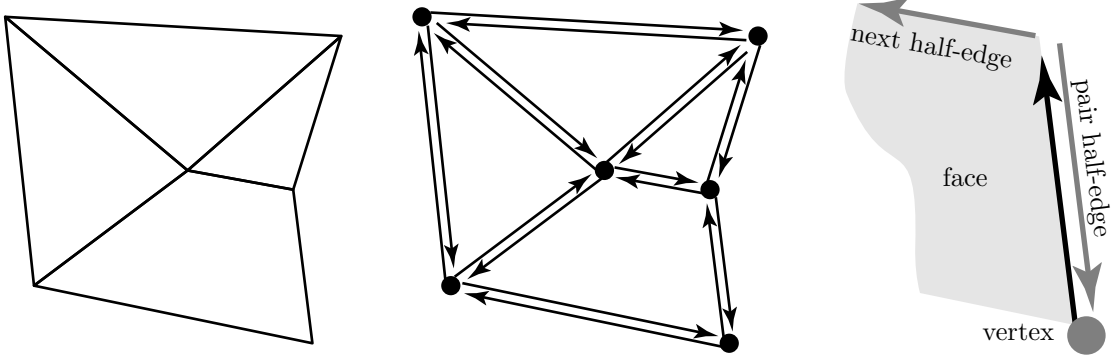


Figure 3.3: Half-edge data structure. The center figure shows the half-edges for the simple mesh on the left. The right figure shows a magnified view of one half-edge (in black) and the other mesh structures that it directly references (in gray).

Given the half-edge data structure, an algorithm can quickly traverse all mesh structures in a connected component of a mesh. A **connected** component of a mesh is the set of all mesh structures (faces, edges, and vertices) such that any two structures are connected by a path of adjacency relationships. This definition is the same as that in graph theory and analogous to connected components in an image segmentation (Section 2.2.3). The half-edge is relatively compact in memory, but can access all adjacency relations in a small number of steps. For example, the vertex at the head of the half-edge is not stored but can be accessed as the tail vertex of the next half-edge (or the tail vertex of the pair half-edge). Following the linked list of next half-edges enumerates all half-edges (and indirectly, vertices) adjacent to a given face. Alternating between following the pair half-edge and next half-edge references enumerates all half-edges (and indirectly, faces) adjacent to a given vertex.

Employing a half-edge representation does impose some minor restrictions on the mesh connectivity. The half-edge representation requires that the mesh is oriented and manifold. An **oriented** mesh is one in which all connected faces have consistent orientation. The orientation of a face is the direction of vertex traversal (clockwise or counterclockwise) around the face. The choice of orientation of an isolated face is inconsequential; however, when joined to another face, the orientation of the second face must agree. More formally, if $f_a, f_b \in F$ are connected by edge (v_i, v_j) then if f_a contains (v_i, v_j) , f_b must contain (v_j, v_i) . Otherwise both half-edges will point in the same direction. The orientation of a face also defines the direction of its normal vector. A face **normal** is a vector perpendicular to the plane containing the face with direction determined by the face orientation using the right-hand rule. The mesh of a consistently oriented surface will have all normals pointing outside the surface (or all pointing inside). A **manifold** mesh is one in which the mesh topology is locally equivalent to a 2-d planar mesh at all points. In a manifold mesh no edge may be adjacent to more than two faces. Also, two surfaces (or two distant points of the same surface) may not join at a single vertex. Figure 3.4 illustrates inconsistent orientation and non-manifold meshes.

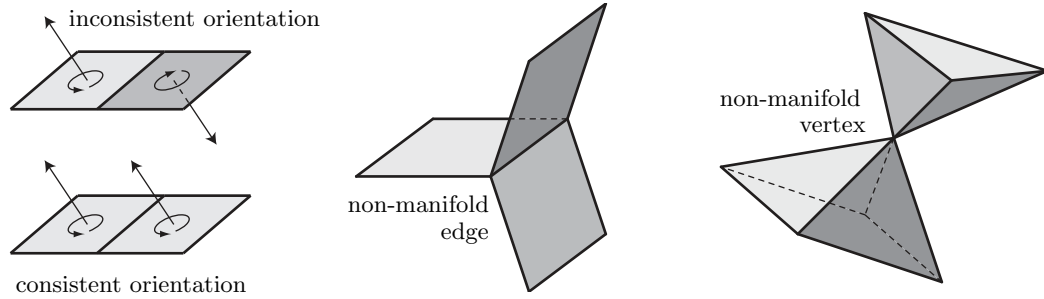


Figure 3.4: Non-oriented and non-manifold meshes. The left figure uses in-plane rotation arrows and normal vectors to illustrate the difference between consistent and inconsistent orientation of adjacent faces. The middle figure illustrates a non-manifold surface caused by three faces sharing an edge. The right figure illustrates a non-manifold surface caused by surfaces joining at a vertex.

The restrictions imposed by the half-edge representation are not a problem for vehicle representations. Every solid 3-d shape is bounded by an oriented, manifold surface. Since the vehicle mesh should represent the boundary of solid shapes approximated by polyhedra, there is no reason the half-edge mesh representation cannot be used. Polygonal meshes that are the boundary of solid polyhedra have an additional property; they are watertight. A **watertight** mesh is one that has no boundary edges. Every face is connected to some other face on all sides forming an enclosure that “holds water”. The water is a metaphor for volume of the 3-d solid. A finite 2-d planar mesh cannot be watertight unless it is also self-intersecting.

3.1.1 Subdivision

A polygonal mesh is often used as a piecewise planar approximation to a continually curving surface. Some amount of detail is invariably lost because of the planar approximation. A given mesh may represent a surface accurately enough for some applications but not enough for others. One solution is use a different polygonal mesh with many more and smaller faces to approximate the same surface. However, this leads to increase memory usage and computational complexity. As a result, both fine and coarse meshes may be useful in different circumstances. Unfortunately, there are an infinite number of way to approximate the same surface with a mesh, and there is no guarantee that any two meshes will have information (mesh topology, or vertex positions) in common. Using multiple unrelated meshes for varying levels of detail is a waste of memory. It can also cause difficulty in corresponding information from one mesh to the other. It would be better if the finely detailed meshes were derived from the coarse ones by simply adding the missing detail. This is the idea behind mesh subdivision. Mesh subdivision takes an existing mesh and splits each face into several smaller faces. The process requires the addition of new vertices and edges. However, the new mesh topology is directly related to the old topology by a well defined algorithm. The new vertices are a superset of the old vertices (though the vertices might also be repositioned). Mesh subdivision forms the basis for meeting the *adaptive complexity* goal for the vehicle mesh representation.

There are many algorithms for mesh subdivision. A naïve approach might try to add new vertices

and edges such that all the original edges remain intact. One way of doing this is to add a new vertex to the center of each face and then connect the new vertex to all the existing face vertices with new edges. Irrespective of the number of sides of each face, the resulting subdivision produces a triangular mesh. Figure 3.5 shows an example of this on the left. This algorithm has the nice property that the graph of the original mesh is a subgraph of the subdivision mesh. No edges are removed or divided. Unfortunately, this subdivision algorithm is not useful for finer surface approximation for exactly this reason. The original, long, straight edges never go away. The mesh also becomes quite imbalanced. One would hope that subdivision would lead to relatively compact faces and most vertices having the same small degree. Instead, the triangle faces become increasingly stretched out and the degree of the vertices grows by three with each subdivision.

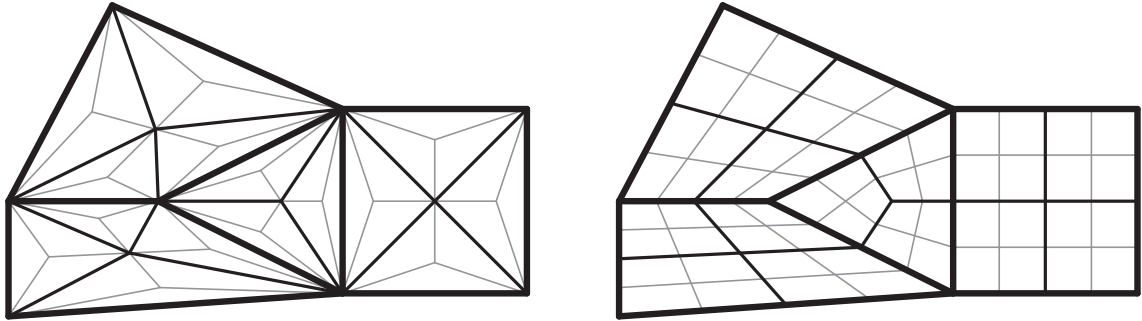


Figure 3.5: Two levels of mesh subdivision with two different algorithms. Each new level is drawn with increasingly lighter line weight. On the left, triangle subdivision without modification of existing edges. On the right, quadrilateral subdivision as computed in Catmull-Clark subdivision surfaces.

A simple variation of the naïve approach can solve these problems. In addition to adding a vertex in the center of each face, one also adds new vertices to center of each edge, splitting each edge into two. Instead of attaching the new center vertex to each original face vertex, the center vertex is attached to each of the new edge center vertices. The result is shown in Figure 3.5 on the right. Irrespective of the number of sides of each face, the resulting subdivision now produces a quadrilateral mesh. In this approach the degree of all added vertices is always four. The faces are compact quadrilaterals that divide evenly into fourths on a grid with each subdivision. This quadrilateral subdivision is commonly used in the computer graphics field to compute subdivision surfaces. The popular Catmull-Clark [17] algorithm uses this approach and adds to it a formula for repositioning the vertices. In Catmull-Clark, vertices are repositioned such that the subdivision surface recursively approximates a bi-cubic uniform B-spline surface with the original mesh vertices used as control points. For vehicle representation, quadrilateral subdivision is used, but vertices are repositioned using a data driven approach described later. The choice of subdivision algorithm is not critical in vehicle representation. Other algorithms could be substituted as long as they result in compact faces and vertices with small degree. Quadrilateral subdivision was selected because it is known to work well when the original mesh consists largely of quadrilaterals. This is true of the vehicle mesh to be discussed in Section 3.2.

3.1.2 Texture mapping

Given any two polygonal meshes with the same topology, one can define a bijective mapping (*i.e.* one-to-one correspondence) from all points on surface of one mesh to the other. Such a mapping is valid even when the vertices of the meshes are embedded in spaces with different extrinsic dimensions. A particular example of this is called texture mapping. In texture mapping, one version of a 3-d mesh is embedded in a 2-d plane. Often it is embedded such that all points lie within a unit square in this plane. The 2-d space containing the flattened mesh is called the **texture space**. The name *texture mapping* comes from computer graphics where this technique is commonly used to map a 2-d image of surface texture onto a 3-d surface for more realistic 3-d rendering. For the vehicle representation it will be used to map 2-d vehicle part geometry onto the vehicle surface in Section 3.3.

While the mapping between meshes is bijective, the mapping between a mesh and a texture space is only bijective if the flattened 2-d mesh faces form a partition of the texture space. In other words, each point in the texture space lies in exactly one 2-d mesh face. If the 2-d mesh does not fill the entire texture space then the mapping is merely injective. Points in the texture space but outside of the 2-d mesh will have no defined mapping onto the mesh. If the 2-d mesh faces overlap in the texture space then points in these regions will map to multiple points on the 3-d mesh. Sometimes a mesh needs to be *cut* open along some edges to avoid overlapping faces when flattening it into 2-d. The resulting map is still valid, but it will be discontinuous across the cut edges. An example of a simple mesh cut open and flattened into texture space is given in Figure 3.6.

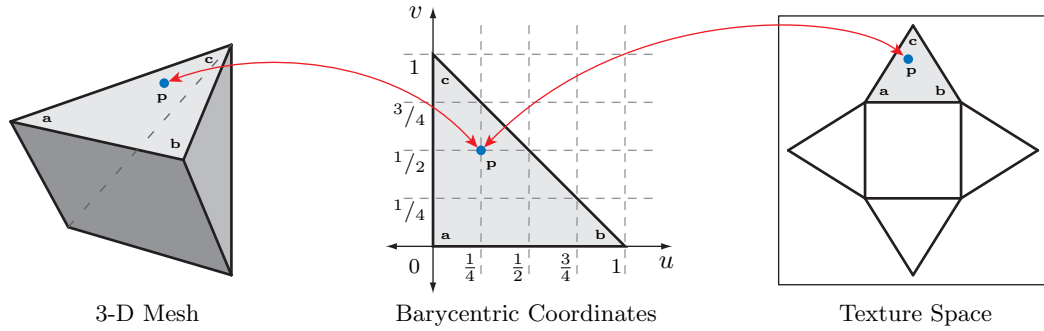


Figure 3.6: Texture mapping with barycentric coordinates. The 3-d mesh of a pyramid on the left is cut open along its edges to form the 2-d texture space mesh on the right. A point is mapped from one space to the other through barycentric coordinates.

Texture mapping is frequently implemented for triangle faces using barycentric coordinates. Assume that a point $\mathbf{p}$ on the 3-d mesh is contained in triangle i with vertices $\mathbf{a}_i$, $\mathbf{b}_i$, and $\mathbf{c}_i$. The point is uniquely represented as

$$\mathbf{p} = (1 - u - v)\mathbf{a}_i + u\mathbf{b}_i + v\mathbf{c}_i, \quad (3.1)$$

where (u, v) are the barycentric coordinates. In 3-d, this expression results in three equations with two unknowns. The solution to (u, v) is over constrained and will not have an exact solution unless $\mathbf{p}$ lies exactly on the plane spanned by triangle i . To ensure a solution, $\mathbf{p}$ is forced onto the triangle

plane by projecting to the closest point on the plane (*i.e.* along the direction orthogonal to the plane). Using the vector operations of cross-product ($\times$) and dot-product ($\cdot$), the solution for the barycentric coordinates is

$$u = \frac{(\mathbf{c}_i - \mathbf{p}) \cdot (\mathbf{n}_i \times (\mathbf{p} - \mathbf{a}_i))}{\mathbf{n}_i \cdot \mathbf{n}_i} \quad v = \frac{(\mathbf{p} - \mathbf{b}_i) \cdot (\mathbf{n}_i \times (\mathbf{p} - \mathbf{a}_i))}{\mathbf{n}_i \cdot \mathbf{n}_i} \quad (3.2)$$

where $\mathbf{n}_i$ is the normal vector of triangle i with length equal to one half the triangle area. This normal vector is computed as

$$\mathbf{n}_i = (\mathbf{b}_i - \mathbf{a}_i) \times (\mathbf{c}_i - \mathbf{a}_i). \quad (3.3)$$

Once u and v are found, (3.1) is applied again—this time in 2-d with $\mathbf{a}_i$, $\mathbf{b}_i$, and $\mathbf{c}_i$ being the 2-d vertices of corresponding triangle i in texture space. The resulting $\mathbf{p}$ is the 2-d texture mapped point. This process is illustrated from left to right in Figure 3.6.

The texture mapping can be inverted as long as the texture space point lies within one of the mesh triangles. The inverse mapping works the same way by mapping from texture coordinates into barycentric coordinates then mapping barycentric coordinates into 3-d with (3.1). Equation (3.2) is simplified somewhat in 2-d since there are now two equations with two unknowns. The 2-d solution for barycentric coordinates is

$$u = \frac{(\mathbf{p} - \mathbf{a}_i) \times (\mathbf{c}_i - \mathbf{a}_i)}{(\mathbf{b}_i - \mathbf{a}_i) \times (\mathbf{c}_i - \mathbf{a}_i)} \quad v = \frac{(\mathbf{p} - \mathbf{a}_i) \times (\mathbf{a}_i - \mathbf{b}_i)}{(\mathbf{b}_i - \mathbf{a}_i) \times (\mathbf{c}_i - \mathbf{a}_i)}, \quad (3.4)$$

keeping in mind that the cross-product of 2-d vectors gives a scalar result.

An arbitrary point, $\mathbf{p}$, in the texture space must be identified with the triangle that contains it before it can be mapped into 3-d. Conveniently, the same calculations for barycentric coordinates are used to test if a point is inside a triangle. Barycentric coordinates have the property that all points within the triangle satisfy all of the following inequalities: $u \geq 0$, $v \geq 0$, and $u + v \leq 1$. If any condition is violated then the point is outside the triangle and a different triangle should be used. An exhaustive search through all the triangles is not always needed. Section 3.3.1 addresses the topic of finding the triangle containing a point in more detail.

The texture mapping methods described so far pertain only to triangular meshes. The vehicle mesh representation requires texture mapping on quadrilaterals as a result of quadrilateral subdivision. Barycentric coordinates have been generalized to arbitrary irregular polygons [77]; however, the equations become significantly more complex. Texture mapping needs to be implemented efficiently because it is one of the core operations that is invoked again and again. A much simpler solution is to bisect each quadrilateral into two triangles and then apply barycentric coordinates to each. This technique is commonly used in computer graphics. There is, however, one caveat. Oliveira [82] points out that when the quadrilateral is not a parallelogram this approach introduces artifacts. The results of texture mapping differ depending on which of the two diagonals is used to bisect the quadrilateral, and neither bisection agrees with the more desirable results produced by a non-linear homography. Fortunately, Oliveira also proves that quadrilateral subdivision dramatically reduces the approximation error. In the limit, subdivided quadrilaterals become parallelograms. Thus the artifacts of bisection become negligible when used with quadrilateral subdivision.

3.2 Vehicle surface mesh

The 3-d vehicle shape is represented by a polygon mesh with five connected components: one for the body and one for each of the four wheels. Each component is a watertight manifold mesh and each is isomorphic to a sphere. In other words, each component represents a solid shape with no holes (*i.e.* genus zero). As shown in Figure 3.7, the mesh faces are primarily quadrilaterals with a few triangles.

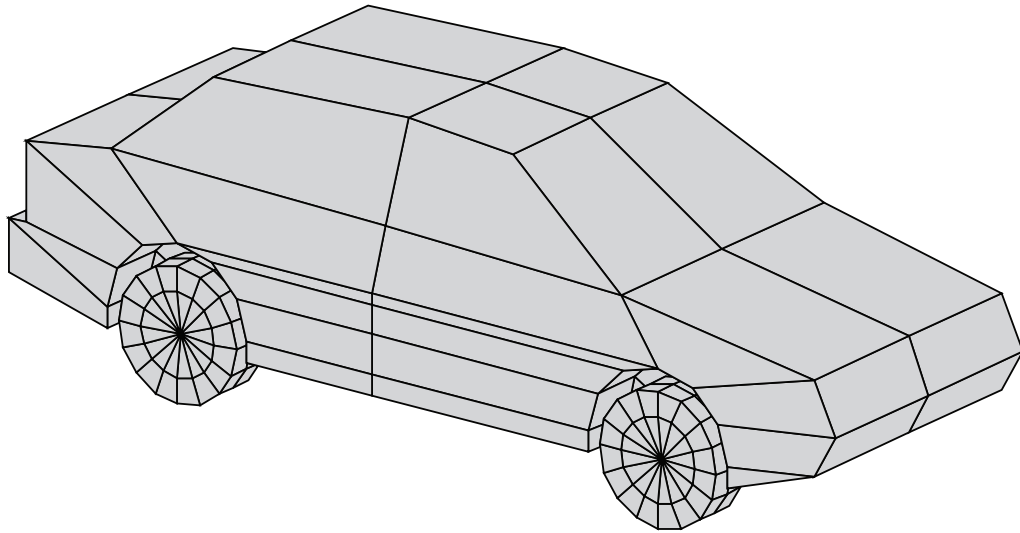


Figure 3.7: The base 3-d template mesh

The coarse 3-d vehicle mesh in Figure 3.7 is called the base **template** mesh. It is called a *template* because it does not represent a fixed 3-d vehicle shape. As with the deformable dodecahedral model, its vertices can be repositioned to capture the rough shape of sedans, minivans, SUVs, pickup trucks, and various other passenger vehicles. The template mesh was designed to represent all of the significant shape features that are consistent across all passenger vehicles while using the fewest polygons possible. This mesh is called the *base* mesh because it is the simplest mesh representation from which all more complicated meshes are derived. Later subdivision of the mesh polygons allows the model to express more detailed shape. Quadrilaterals were chosen because many regions of vehicle surfaces are well approximated by a large quadrilaterals.

The complexity of the body mesh is only slightly greater than that of the simple polyhedral models in used in prior work and shown in Figures 2.16 and 2.17. It is similar to one quadrilateral subdivision of Ferryman’s model (Figure 2.17). The extra subdivisions edges allow for the specification of additional constraints that are not otherwise available. The constraints need to be specified in the base mesh to get proper correspondence of shape features when the mesh is subdivided. For example, the line that separates the vehicle’s front from its back is used to indicate the trailing edge of the vehicle’s front doors. If not specified in the base template mesh, it would cause alignment problems with some 2-d vehicle parts (*e.g.* the side windows). Similarly, the front and rear bumpers

are 3-d shapes clearly present at finer mesh resolutions, so some vertices must represent them in the coarse mesh to enforce proper correspondence.

Wheels are added to the vehicle representation because they are a critical part of vehicle appearance. Wheels are large, have a predictable cylindrical shape, and often have sharp contrast between the bright metal hubcaps and dark rubber tires. This makes them an important addition to the model. Other parts that extend from the body, like side mirrors and luggage racks, have been ignored. These parts are smaller and generally less noticeable, especially at image resolutions for which the base mesh is appropriate. They are also very difficult to model as mesh features connected to the base body mesh. They would require the addition of many small faces that are inappropriate at the resolution of the base mesh. However, if needed, these protruding parts could be added in the future as separate connected components that are disconnected from the body (just as the wheels are). The wheels are treated differently than the body in terms of subdivision. They are modeled with simple cylindrical meshes with 16 sides and fixed at this mesh resolution when the body mesh is subdivided. A subdivision scheme could be applied to the wheels as well, but this step has been avoided for now to focus on the vehicle body mesh which is a much more interesting and challenging surface.

For the body mesh, quadratic subdivision is applied as described in Section 3.1.1. Subdivision is applied recursively to create several mesh resolutions with varying levels of detail. The results of these subdivisions are shown in Figure 3.8. The newly added vertices are repositioned based on vehicle shape data rather than a smoothing function as in Catmull-Clark [17]. The data based shape learning is discussed further in Chapter 5. Recursive subdivision can be applied as many times as needed to produce an arbitrarily close approximation to the vehicle surface. However, for the applications examined in this thesis, two levels of subdivision seem sufficient.

It may seem wasteful to subdivide all faces equally, especially when a large face seems to approximate planar areas of the surface just fine. If this were a mesh for a fixed shape object (like the CAD models are) the typical solution would be adaptive mesh subdivision. Adaptive subdivision only divides the faces in areas of high curvature. But this vehicle mesh does not model a fixed shape. It is a template model that must deform to match the shapes of many different vehicles. The faces chosen for adaptive subdivision of one vehicle shape are not necessarily appropriate for another. It is better to think of the template vehicle mesh as a generalized cartesian grid upon which the values of a shape function are sampled. To represent higher frequency functions, a finer grid sampling is used.

The unit square texture space shown in Figure 3.9 is used for texture mapping the surface of the 3-d vehicle mesh. To avoid the ambiguous mapping described in Section 3.1.2, the texture map does not include the underside of the vehicle or inward sides of the wheels which are generally not observed. The unobserved faces are cut away along mesh edges and the remaining faces are flattened into the texture space keeping the exact same mesh topology for all remaining faces. The texture coordinates of the template vertices are determined once and fixed. They are the same for all configurations of the corresponding 3-d vertices.

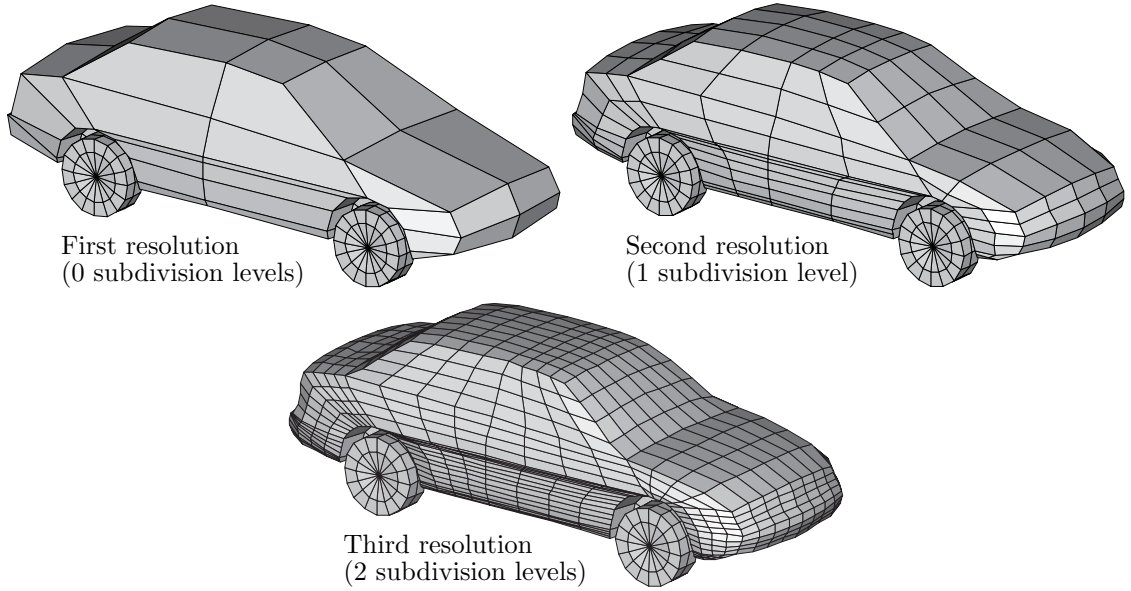


Figure 3.8: Vehicle mesh subdivision. The template mesh (top right) is subdivided to produce the second resolution (top left), and subdivided again for the third resolution (bottom).

When the 3-d mesh is subdivided, the texture space mesh is also subdivided in the exact same way. In the texture space, new vertices from quadrilateral subdivision are added to the center of an edge or the center of a face. The texture space coordinates are not repositioned in any way. The result is a generic vehicle texture space that applies to all vehicle surface resolutions and all vehicle surface shapes. If the 3-d vertices of the template mesh are mapped to corresponding features on the various different vehicle shapes, then a point in the vehicle texture space should correspond to the same point on the surface of all vehicles.

3.3 Vehicle parts

Vehicle parts, in the context of this thesis, are regions of vehicle surface with different material properties than the vehicle body. These parts help to explain the additional aspects of the vehicle appearance that are not predicted by the 3-d surface model. For example, a taillight may appear bright red whereas the vehicle body is another color and may appear darker or lighter than the taillight. This difference in appearance tends to generate edges in a vehicle image between the part and the body. However, the taillight itself is nearly flush with the rest of the body, so these edges are not accounted for by the 3-d surface model. A similar situation exists for several other vehicle parts including windows, headlights, grilles, license plates, and hubcaps. The full list of parts considered for the vehicle model is given on the right of Figure 3.10. The parts that have been selected for modeling are those that are found consistently across the various classes of vehicles. They must also be found in a relatively consistent location on the vehicle surfaces. Some of the selected parts are not

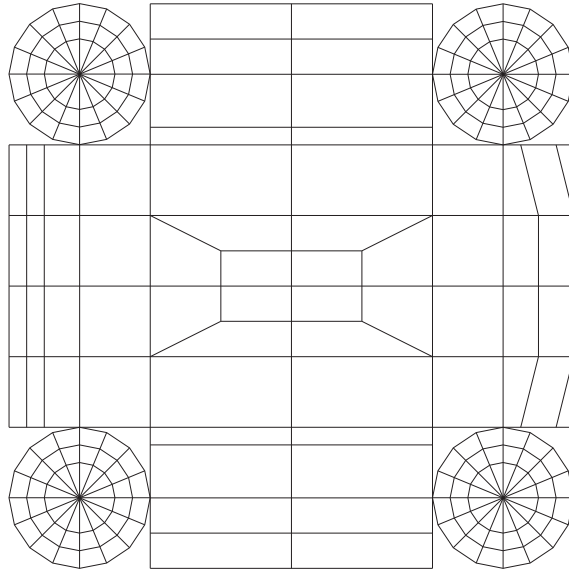


Figure 3.9: The vehicle mesh texture space.

found on all vehicles but are found on enough vehicles that they might provide useful information. For example, not all vehicles have a sunroof, middle side windows, or rear side windows. All of the potentially useful vehicle parts have been included for now. In Chapter 4, algorithms for fitting parts to images are made robust to handle vehicles that do not have all parts.

Previous vehicle models have accounted for some part edges by associating them with 3-d mesh edges. For example, the dodecahedral model (Figure 2.16) has mesh edges that separate the lower half of the body from the upper half. These edges coincidentally appear near the lower boundaries of the vehicle windows and therefore coincide with intensity edges in vehicle images caused by the window appearance. Unfortunately, using the surface mesh edges to model part boundaries is not always so easy. The taillight is, again, a good example. Taillights on vehicles come in all different shapes and size. Sometimes they are small, and sometimes they are large. Sometimes they are boxy, and sometimes they are rounded. Sometimes they are flat on the back of the vehicle, and sometimes they wrap around the sides of the vehicle. All of these variations are possible on vehicles that are otherwise very similar in 3-d shape. Forcing a subset of 3-d mesh edges to correspond to part boundaries can break the correspondence between 3-d shape features. Thus, the part boundaries must be described in a way that is not restricted by the 3-d mesh topology.

Instead of using mesh edges, the parts are described by polygons in the texture space. Figure 3.10 shows a polygon for each part, and the shape of these polygons is defined by the mean over all vehicles in the training data. The concept of mean shape is defined more carefully in Section 3.5. Defining the parts in texture space allows them to be texture mapped onto the vehicle surface independently of the 3-d mesh topology. Figure 3.11 shows the parts from Figure 3.10 mapped onto the surface of the third resolution template mesh. The parts can undergo 2-d shape deformations in the texture space independently of the 3-d surface shape. In many ways, the parts are the 2-d analog of the

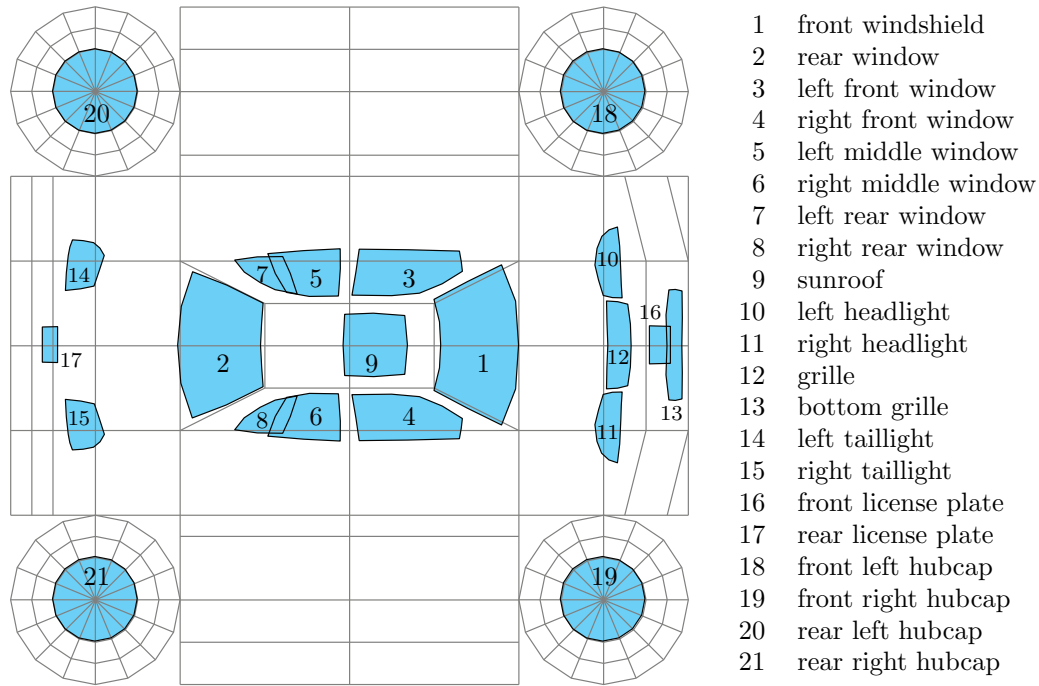


Figure 3.10: The vehicle parts defined in texture space.

3-d surface mesh. The parts are 2-d structures in a 2-d space represented by their 1-d boundary structures. The vehicle body is a 3-d structure in a 3-d space represented by its 2-d boundary structure. The geometry in both the 2-d parts and the 3-d mesh is specified by a finite collection of vertices. The shape of both is deformable by adjusting the position of these vertices.

The process of mapping these polygon parts to the surface of the vehicle is a bit different than conventional texture mapping in computer graphics. The difference is one of raster graphics versus vector graphics. Traditional texture mapping involves raster data in the texture space. The texture space data is an image with color data stored at each pixel. When rendering a new image, a ray from each pixel in the output image is traced back to a point on the surface of the mesh. That point is mapped into the texture space (*i.e.* texture image coordinates) and the color value is interpolated at that point from the image data.

With vector graphics, such as the part polygons, the inverse mapping is more important. The part vertices are the points of interest in the texture space, and one would like to know how they map to the surface of the vehicle (and ultimately, how they map into an image). Mapping the part vertices themselves has already been covered in Section 3.1.2. However, it is not sufficient to merely map each vertex in a part polygon onto the 3-d surface and then connect them with line segments in 3-d. Such line segments would intersect the 3-d surface at their endpoints, but each line would not lie on the surface in general. An arbitrary line segment in the texture space will intersect several mesh edges. To correctly map the segment onto the mesh surface, one must split the line segment at each intersection with a mesh edge forming a chain of connected line segments. When each point

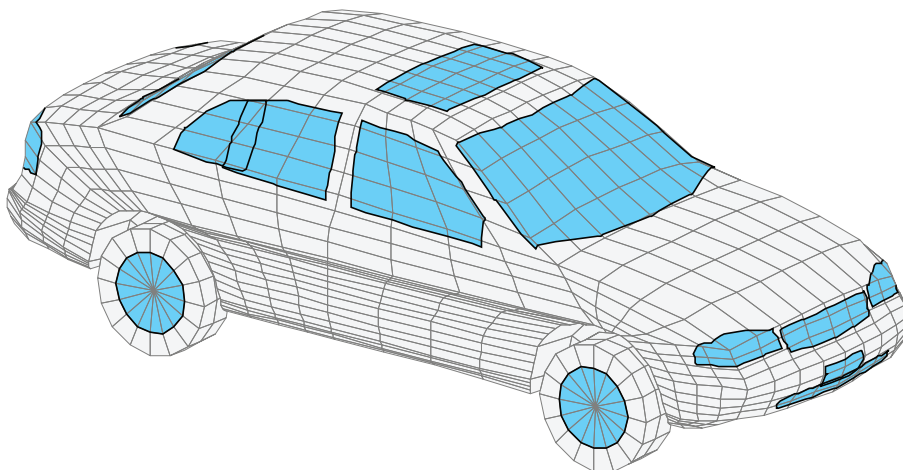


Figure 3.11: The vehicle parts texture mapped onto the 3-d mesh surface.

in the line segment chain is mapped onto the surface, the resulting piecewise linear 3-d curve will lie in the mesh surface.

3.3.1 Intersections with mesh edges

To texture map a polygon part onto the 3-d surface of the vehicle, all intersections must be computed between all mesh edges and all sides of the polygon. Furthermore, all polygon vertices, both original and those resulting from intersection, must be identified with a face of the mesh that contains them. At first glance, finding this set of intersections and face correspondences sounds like it could be a huge combinatorial problem. Fortunately, it can be made much more efficient by taking advantage of adjacency relationships within the mesh and within the polygons.

The strategy for finding all the mesh edge intersection points and mesh face correspondences for a given part polygon is as follows. First, choose an arbitrary part vertex as the starting point. Using exhaustive search, locate the mesh triangle that contains this point. Then, examine the next vertex along the polygon boundary and consider the line segment between the current and next vertices. If the next vertex does not lie in the same mesh triangle, it must intersect one of the three mesh edges adjacent to the triangle. Find the intersection point and then move to the triangle that is adjacent to the current triangle across the mesh edge containing the intersection. If this new triangle does not contain the next part vertex, then continue traversing the mesh, finding intersections along the way, until it does. Once the line segment endpoint is reached, continue tracing the next line segment to the next part vertex. Continue traversing the part boundary and mesh together until returning to the starting vertex. As a byproduct of traversing the mesh to find intersections, the face containing each vertex and intersection point is discovered.

Finding the intersection points is made even more efficient by computing intersections directly in

the barycentric coordinates of the current triangle. Let (u_p, v_p) denote the last vertex or intersection point in the barycentric coordinates of the current triangle. Compute (u_q, v_q) , the ending vertex of the current part line segment in the barycentric coordinates of the same triangle. The difference vector in barycentric coordinates is $(du, dv) = (u_q - u_p, v_q - v_p)$. The goal is to determine which of the three triangle edges is intersected next along this vector and to find the coordinates of the intersection. Let e store the intersected edge from the set $\{\mathbf{ab}, \mathbf{bc}, \mathbf{ca}\}$. These letters refer to edges connecting vertices $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ as labeled in Figure 3.6. Let t be the scaled distance along the difference vector such that the intersection point is located at $(u_i, v_i) = (u_p + t \, du, v_p + t \, dv)$. The intersected edge, e , and intersection distance, t , are easily determined by a few simple tests:

$$(e, t) = \begin{cases} \left(\mathbf{ca}, \frac{-u_p}{du} \right) & \text{if } du < 0 \text{ and } 0 < v_i < 1 \\ \left(\mathbf{ab}, \frac{-v_p}{dv} \right) & \text{if } dv < 0 \text{ and } 0 < u_i < 1 \\ \left(\mathbf{bc}, \frac{(1 - u_p - v_p)}{(du + dv)} \right) & \text{if } dv + du > 0 \text{ and } 0 < u_i \text{ and } 0 < v_i \end{cases} \quad (3.5)$$

These three cases account for all the edge intersections; however, there are also three degenerate cases where the line segment intersects a triangle vertex. The degenerate cases are detected when some of the inequalities in (3.5) become equalities. The intersection lies on vertex $\mathbf{a}$, $\mathbf{b}$, or $\mathbf{c}$ if the barycentric coordinates are $(0, 0)$, $(1, 0)$, or $(0, 1)$, respectively. By chance, one would not expect these degeneracies to ever occur, but the part vertices are not arranged by chance. For example, the hubcap part vertices are designed to lie exactly on mesh vertices. To correctly detect these degenerate cases, a tolerance, ϵ , is needed to account for round-off error in floating point calculations. An intersection is assigned to a vertex if both u_i and v_i are within ϵ of a vertex coordinate. In practice, ϵ is set to 0.01, a value larger than necessary. The larger ϵ helps to snap near vertex intersections to the vertex to avoid having several closely spaced intersections with the multiple mesh edges adjacent to the vertex.

All of this mesh traversal is made very efficient with the half-edge data structure. Traversing triangles in the general case requires traversing a next half-edge reference at most twice (depending on e) followed by a pair half-edge reference once. When a line intersects a vertex, some additional work is needed to make sure it comes out on the other side into the correct triangle. As the half-edges adjacent to the vertex are traversed, their angles in the texture space are computed. The appropriate triangle is the one in which the angle of the line is between the angles of mesh edges adjacent to the triangle. Finally, it is possible that some vertices of a deformed polygon part are moved outside of the valid texture space domain. This situation is discovered during mesh traversal when a line segment crosses a boundary edge of the texture space mesh. When this happens the part is simply clipped by the boundary, and only the valid part is mapped. Returning to the starting vertex, polygon traversal is applied in the other direction to find the other point where the polygon intersects the boundary.

3.4 Image edge prediction

The generic vehicle model is only useful in computer vision applications if it can be related to image observations. As discussed in Chapter 2, detected image edges are good geometric structures for relating a vehicle model to an image. Thus, the vehicle model needs to predict the appearance of enough of these edges to constrain its pose and shape parameters by corresponding the hypothesized edges to the observed edges.

Detected edges on images of vehicles arise from three main sources: 3-d geometry of vehicle surfaces, albedo discontinuities, and specular reflections of the environment. 3-d surface geometry generates edges when the surface normal changes abruptly causing a discontinuity in the light reflected to the camera. Edges are also generated where the 3-d surface is tangent to the viewing direction causing a depth discontinuity and a juxtaposition of foreground and background appearance. Likewise, surface albedo discontinuities map to image intensity discontinuities that produce image edges. Both of these edge sources also occur on background objects, and the mirror-like surface of vehicles generates additional edges by reflecting the background appearance. Each edge is not distinctly generated by only one of these sources. Rather, the superposition of all these sources is involved in each edge. Some edges are more strongly influenced by one source than another. Superposition can be both additive and subtractive for edges. For example, a reflection of the environment in the neighborhood of a predictable albedo discontinuity can produce three different effects: One, additional edges can appear around the albedo edges. Two, the reflection can distort the position or angle of the albedo edges. Three, the reflection can completely override the albedo causing no edge at all.

In addition to the complexities of edges on the vehicles, the background of the image is also typically full of edges from other objects in the scene. A priori, the edges belonging to the target vehicle are not distinguished from the background. Predicting all of the edges in a vehicle scene is intractable and would require a complex model of the surrounding environment. As a result, any algorithm predicting a subset of image edges, must be robust to a large number of outlying edges and the distortion or absence of several expected edges. The vehicle model in this thesis aims to predict the subset of edges that are not only detected reliably, but are also relevant to the majority of passenger vehicles. These hypothesized edges come in two varieties. The 3-d shape of the vehicle mesh is used to predict edges that are caused primarily by the change in appearance due to 3-d shape of the vehicles. The 2-d parts are used to predict edges that are caused primarily by albedo discontinuities on the vehicle surface. The remaining edges, caused by unmodeled features, environment reflection, and background objects, are treated as outliers.

Figure 3.12(a,b) shows an image of a vehicle and the sub-pixel Canny edges detected in that image. Assume, for now, that the template mesh and parts have been deformed to match this vehicle and that a camera model is known with parameters and relative pose matching the image. The image in Figure 3.12(c) is rendered from these known vehicle and camera models. Using the same models, edges are predicted in two ways. The occluding contours of the 3-d mesh (defined below) are used to generate edge hypotheses based on vehicle shape. The part boundaries, texture

mapped onto the vehicle and then projected into the image, are used to generate hypotheses based on albedo discontinuities. These hypotheses for the running example are shown in Figure 3.12(d) with occluding contours in green and part boundaries in red. Unfortunately, not all of these predicted edges are viable. Many of them are self-occluded by the vehicle and are not visible in real images.

The **occluding contours** of a 3-d mesh are the subset of mesh edges that form the boundary between the set of faces that are observed and the set of faces that are occluded. This set of edges is a function of both the mesh and the camera. The set changes as the viewpoint changes. Occluding contours include the shape silhouette and also some contours interior to the silhouette caused by self-occlusion. A superset of the occluding contours called the occluding contour generator is computed using only local mesh properties. If $\mathbf{n}_i$ is the normal vector of mesh face i and $\mathbf{v}$ is a vector in the viewing direction of the camera, then the occluding contour generator is defined as the set of all mesh edges with adjacent faces i and j such that $\mathbf{n}_i \cdot \mathbf{v}$ and $\mathbf{n}_j \cdot \mathbf{v}$ have different signs. That is to say, one face adjacent to the edge is facing the camera and the other is facing away from it. The contour generator forms the boundary of a subset of faces, and thus always exists as closed cycles of edges. The set of green edges in Figure 3.12(d) is the occluding contour generator for the mesh in Figure 3.12(c). Not all contour generator edges are visible in the image because of self-occlusions. Some front-facing faces are occluded by other front-facing faces that are closer to the camera. Thus, determining the occluding contours, the visible subset of the contour generator, requires global mesh properties. Hidden line removal is needed to remove the self-occluded subsets of both the occluding contour generator and the projected part boundaries.

Hidden line removal is a long studied problem in computer graphics and several algorithms exist. Any hidden line removal algorithm should be sufficient to remove the hidden edge hypotheses. Some algorithms, like Appel's algorithm [3], are *object-precise*. They intersect line segments and propagate visibility from segment to segment. These methods are most precise but can be slow, have many special cases to implement, and are unstable. Certain degenerate cases can cause errors that propagate. The algorithm used in this thesis is an *image-precise* method using a depth map (*i.e.* Z-buffer). It is fast, simple, and stable. A floating point depth map at the image resolution is rendered from the 3-d mesh. Figure 3.12(e) shows an example where white pixels are closest and black pixels are farthest from the camera (green pixels have no depth). When each vertex of a hypothesized edge is projected into an image, its depth is computed and compared against the four nearest depth map pixels. If the depth is greater than any of these pixels the point is culled. The remaining hypothesized edges are those that connect two points that were not culled. Figure 3.12(f) shows the remaining set of hypothesized edges after hidden line removal. The results are not perfect. Some small gaps are formed because line segment visibility is treated as all-or-nothing. The gaps could be removed by computing the $O(n^2)$ line segment intersections, but this computation is not justified because the further use of hypothesized edges does not rely on connectivity.

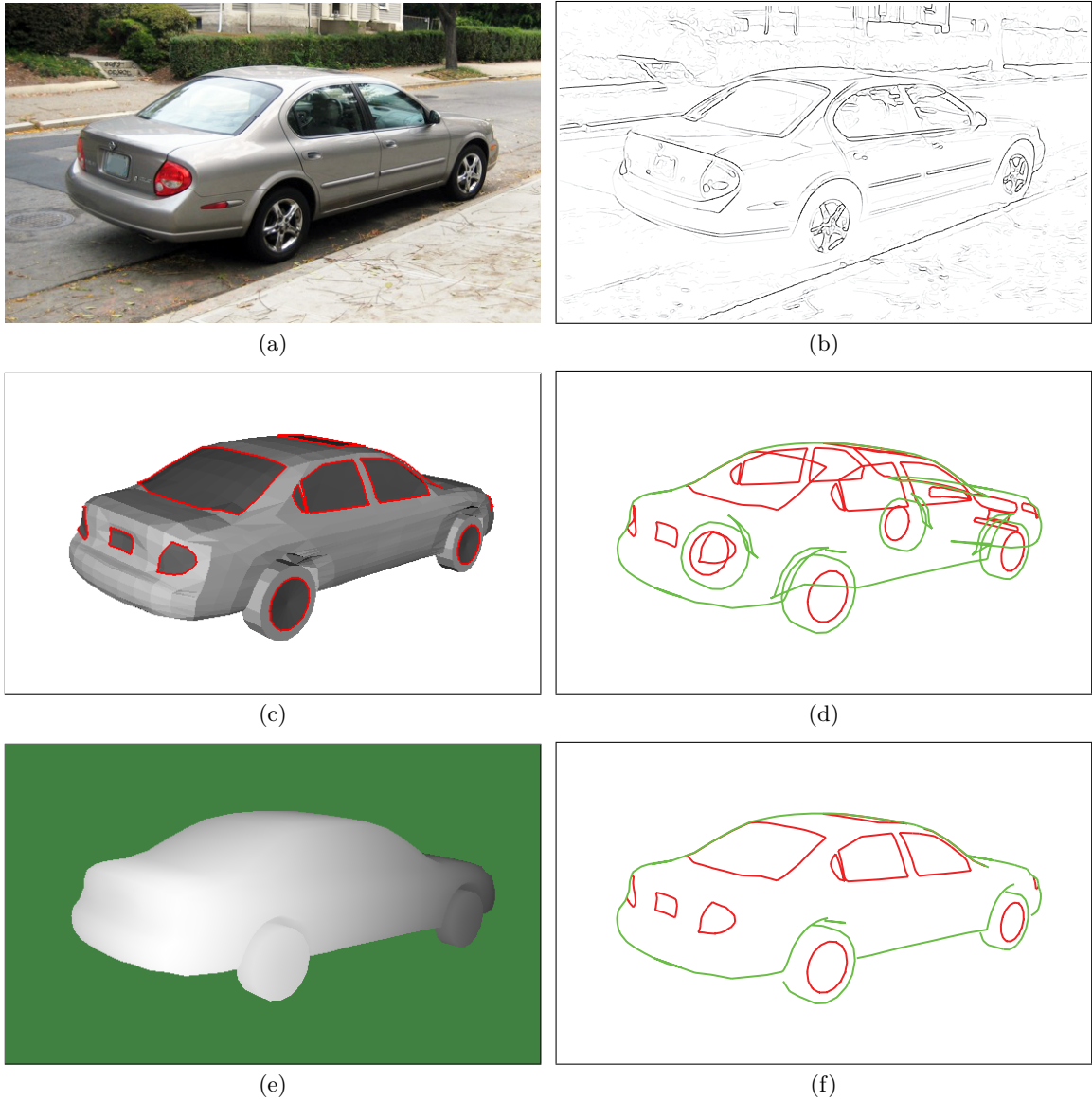


Figure 3.12: Image edge prediction using part boundaries and occluding contours. The edges from image (a) of a Nissan Maxima are shown in (b) with stronger edges rendered darker. The template model and parts are deformed to the vehicle shape and rendered in (c) with a matching camera model. All hypothesized edges from occluding contours and part boundaries are shown in (d) in green and red respectively. A depth-map of the vehicle in (e) is rendered from the mesh and used to remove hidden lines. The final set of visible hypothesized edges is shown in (f).

3.5 Model deformations

The final requirement of the generic vehicle model is a low dimensional parameter space controlling the deformation of the 3-d mesh and 2-d parts. The positions of both the 3-d mesh vertices and 2-d part vertices are adjusted to predict the shape and appearance of arbitrary vehicles. When the number of parameters of shape deformation is too large, estimation of shape becomes intractable. The complete space of vehicle shapes allows each mesh vertex and each part vertex to be independently positioned in 3-d space and 2-d texture space respectively. Each 3-d mesh vertex has three degrees of freedom, and each 2-d part vertex has two degrees of freedom. At the third mesh resolution there are 1444 3-d vertices and 228 2-d vertices resulting in a 4788-dimensional shape space. This is clearly not low dimensional. Furthermore, it is obvious that the vertices are not independent when representing actual vehicles. Adjacent vertices have spatial proximity. Remote vertices may be correlated by vehicle symmetry. A reduced parameter space is a practical way to enforce these constraints and is, in fact, needed to make shape optimization tractable.

With the dodecahedral model, Koller *et al.* [59] and others use a manually specified set of parameters to enforce symmetry and other constraints. Manual constraints quickly become impractical for more complex models. For complex models, the popular solution has been principal components analysis (PCA). PCA learns a parameters subspace that implicitly enforces the constraints present in a collection of training examples. Ferryman *et al.* [30] use PCA to reduce the parameter space of their vehicle model. More complex mesh models in other areas of computer vision and graphics have also adopted PCA parameterization. Blanz and Vetter [9] use PCA on a 3-d human face model, and Anguelov *et al.* [2] use PCA on a 3-d human body model. Both models contain tens of thousands of vertices. PCA provides the necessary dimension reduction while simultaneously providing a prior distribution on the parameters.

3.5.1 Principal components analysis

PCA takes N training samples in a M -dimensional space and transforms the samples into an m -dimensional space ($m = \min(M, N)$) of principal components in which the samples are uncorrelated. In addition, the principal components are sorted such that the first component accounts for the greatest direction of variability in the samples. The n^{th} component accounts for the greatest variation in the subspace orthogonal to the first $n - 1$ components. Thus, for any $n \leq m$, the first n principal components provide the n -d subspace that accounts for the greatest amount of variation in the data.

Applying PCA to the deformable vehicle model requires a collection of training samples. Assume, for now, that the mesh and part vertices have been positioned to accurately represent each vehicle in a database of N vehicles. Chapter 5 will explain how these training data are obtained. Let M_3 be the number of 3-d mesh vertices and M_2 be the number of 2-d part vertices. The coordinates of each 3-d mesh vertex, $\mathbf{v}_i = (v_i^x, v_i^y, v_i^z)$, and 2-d part vertices, $\mathbf{w}_i = (w_i^x, w_i^y)$, are stacked into a single M -d vector ($M = 3M_3 + 2M_2$),

$$\mathbf{s} = \begin{bmatrix} v_1^x & v_1^y & v_1^z & \cdots & v_{M_3}^x & v_{M_3}^y & v_{M_3}^z & w_1^x & w_1^y & \cdots & w_{M_2}^x & w_{M_2}^y \end{bmatrix}^\top. \quad (3.6)$$

This M -d space is the complete shape space, and each training vehicle configuration is represented by a point in this space. The shape vector sample mean, $\mu = \frac{1}{N} \sum_{k=1}^N \mathbf{s}_k$, is the average of all N training samples. The mean vehicle shape and mean vehicle parts are shown when representing the template mesh in Figures 3.7, 3.8, 3.10, and 3.11. For these figures, the database contained $N = 79$ vehicles covering various passenger vehicles (43 four-door sedans, 11 two-door sedans, 10 SUVs, 7 minivans, 6 pickup trucks, and 2 station wagons).

The principal components of vehicle shape are computed with respect to the mean vehicle shape, μ . The mean is subtracted from the training samples which are stacked into a $M \times N$ matrix of deviations from the mean

$$\mathbf{D} = \begin{bmatrix} | & | & & | \\ \mathbf{s}_1 - \mu & \mathbf{s}_2 - \mu & \cdots & \mathbf{s}_N - \mu \\ | & | & & | \end{bmatrix}. \quad (3.7)$$

The singular value decomposition (SVD) of $\mathbf{D}$ is used to find the principal components. SVD decomposes $\mathbf{D}$ as $\mathbf{D} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ such that $\mathbf{U}$ is an $M \times m$ orthonormal matrix, $\mathbf{\Sigma}$ is an $m \times m$ diagonal matrix with nonnegative, monotonically decreasing values, and $\mathbf{V}$ is an $N \times m$ orthonormal matrix. The values on the diagonal ($\sigma_i = \Sigma_{ii}$) provide the magnitudes, and the corresponding columns of $\mathbf{U}$ provide the directions, of most significant variation.

Parameters in the space of these principal components provide a linear combination of the N vehicles. One can interpolate between and extrapolate from the training data. To reduce the parameter space from m to n dimensions, the largest n singular values are retained and the rest are set to zero. By the properties of SVD, this n -d subspace captures the largest amount of the original variation. The $M \times n$ matrix $\mathbf{G}$, formed by the first n columns of $\mathbf{U}$, provides a mapping from a reduced parameter space point $\mathbf{p}$ to the full M -dimensional shape space point $\mathbf{s}$ via $\mathbf{s} = \mathbf{G}\mathbf{p}$. Figure 3.13 shows how the template vehicle deforms when varying the top five PCA parameters.

3.5.2 Shape priors

The top n singular values can also be interpreted as the standard deviations of a multivariate normal distribution fit to the training data in the reduced parameter space. The probability density of the multivariate normal distribution is

$$f_{\mathbf{P}}(\mathbf{p}) = \frac{1}{(2\pi)^{N/2} |\mathbf{\Sigma}_{\mathbf{P}}|^{1/2}} \exp \left(-\frac{1}{2} (\mathbf{p} - \mu_{\mathbf{P}})^\top \mathbf{\Sigma}_{\mathbf{P}}^{-1} (\mathbf{p} - \mu_{\mathbf{P}}) \right). \quad (3.8)$$

Since the PCA subspace is constructed such that each point $\mathbf{p}$ is relative to the mean, $\mu_{\mathbf{P}} = 0$. Furthermore, $\mathbf{\Sigma}_{\mathbf{P}} = \mathbf{\Sigma}^\top \mathbf{\Sigma}$ where $\mathbf{\Sigma}$ is the diagonal matrix of singular values from SVD. Since all the dimensions are uncorrelated, the density is simplified to the product univariate densities in each dimension,

$$f_{\mathbf{P}}(\mathbf{p}) = \prod_{k=1}^n \frac{1}{\sigma_k \sqrt{2\pi}} \exp \left(-\frac{p_k^2}{2\sigma_k^2} \right). \quad (3.9)$$

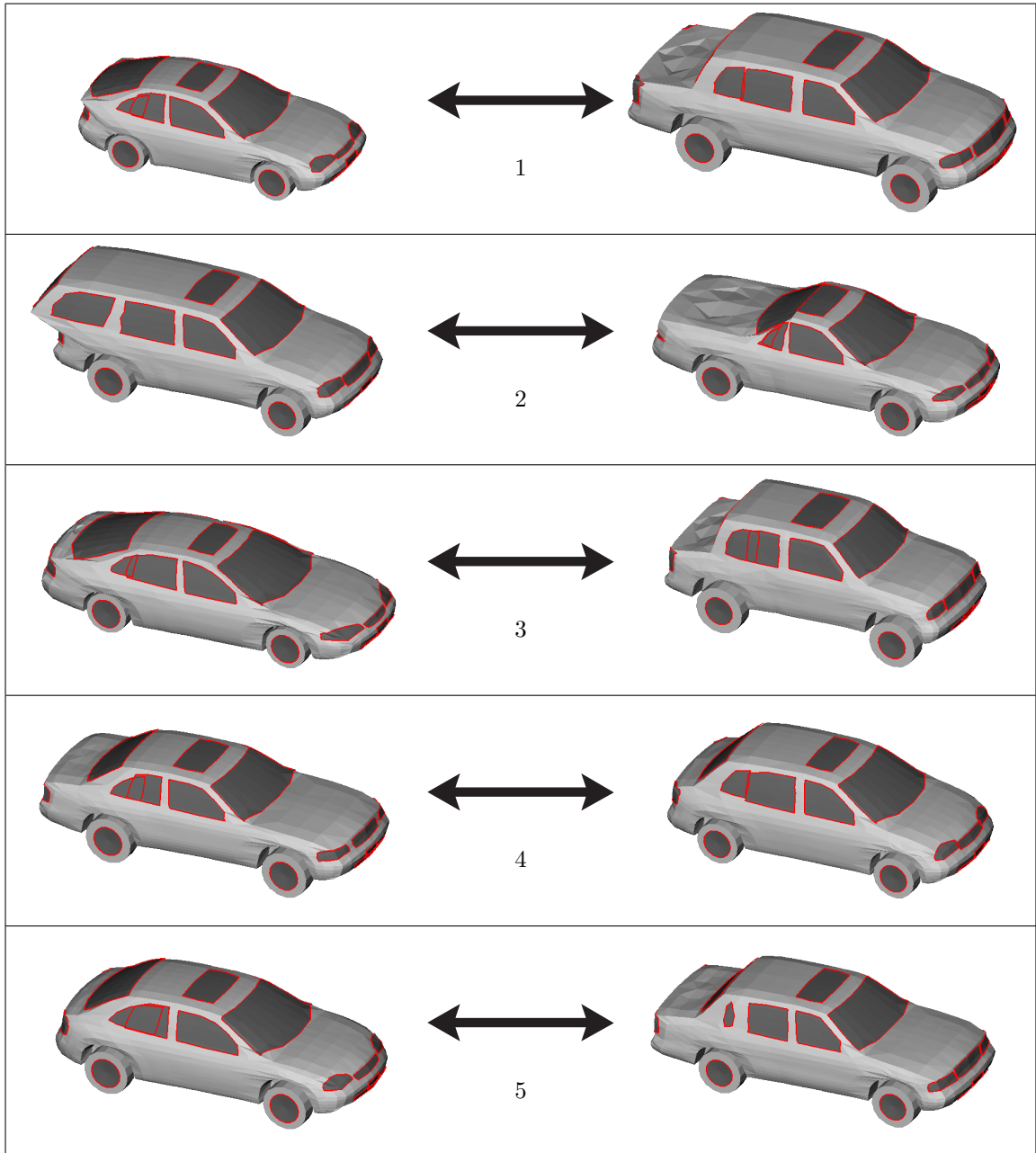


Figure 3.13: Principal components of vehicle deformation. The top five principal components are visualized by rendering the template mesh at $\pm 0.25\sigma$ in each dimension.

In other words, σ_k is the standard deviation of a zero mean normal distribution along principal component k .

That fact that PCA implicitly fits a normal distribution to the data does not imply that a normal distribution is actually a *good* model of the underlying distribution of vehicles. It is not. First of all, the space of vehicle shapes is technically discrete; only a finite number of vehicle shapes have been produced by manufacturers. However, it would be difficult to obtain samples for every vehicle ever produced, and manufacturers continue to produce new designs. In that sense, one could argue that these new (or previously unknown) shapes are drawn from a continuous distribution in vehicle shape space. Even so, a single normal distribution does not seem reasonable to model the distribution of all passenger vehicle shapes. One would expect this distribution to have multiple modes centered around the various vehicle classes. In reality, if a sedan shape is likely, and a pickup truck shape is likely, that does not imply that the average between these shapes is also likely. A half-sedan, half-pickup shape is rather unlikely, but a single normal distribution cannot express this.

A normal distribution is, however, better than nothing. A half-sedan, half-pickup may be unlikely in reality, but at least the shape is physically possible. When extrapolating shape along a principal component, if a parameter deviates too far from the mean then the shape may collapse on itself causing self-intersections, degenerate shapes, and other physically impossible conditions. There are no strict constraints preventing these conditions, but they will be considered less likely according to the normal distribution. Therefore, the normal distribution acts as a prior distribution on valid shape and can be used for regularization during shape estimation as described in Chapter 4. There is still room for a better prior distribution that more accurately captures the likelihood that a shape represents a real vehicle.

Chapter 4

Fitting to Images

The deformable vehicle model described in Chapter 3 can take a small number of shape parameters, pose parameters, and camera parameters and produce a hypothesis of vehicle image edges. Generating these hypotheses is not all that useful in and of itself. It is far more useful to consider the observed image edges of a real vehicle and then determine the model parameter values such that the hypothesis best agrees with the observations. The process of determining these optimal values is called **fitting** the model to the image (or images). Fitting the model parameters effectively recovers lost information about the observed vehicle. The recovered parameter values represent the 3-d shape of the vehicle, and this recovered data can be used to render the vehicle from novel directions or even recognize the vehicle.

Fitting the vehicle model to real image data is very challenging. The most difficult part is obtaining correct correspondence between hypothesized edges and observed edges. As discussed earlier, the deformable vehicle model only predicts a subset of the edges that are observed in real images. So one would expect to observe many edges that must be ignored. The opposite problem is also present. The vehicle model predicts some edges that may not be present in the image. Ideally, these edges should also be ignored and not contribute to the fit. Even if the exact set of edges was predicted, there is still the problem of determining which predicted edge corresponds with which observed edge. If the hypothesis is very close to correct, then it might be acceptable to assign each observed edge to the nearest hypothesis edge. However, this nearest neighbors approach requires good initialization which must be provided somehow, and the correspondence will invariably still contain errors.

To simplify the explanation of the fitting algorithm, the full complexity of the correspondence problem is ignored at first. For simplicity, it is assumed that a good initialization of the shape and pose parameters is provided, and that searching in the neighborhood of hypothesized edges leads to correct correspondences. Under that assumption, the framework for optimizing the vehicle shape and pose parameters is presented. Afterwards, the problems related to incorrect correspondence are revisited. A discussion of parameter initialization postponed until Chapter 6.

4.1 Parameterization of shape and pose

The parameters that are of interest for optimization are the top n PCA parameters described in Section 3.5.1 and up to 6 additional parameters for translation and rotation. The camera parameters are assumed to be known and fixed. In general, the 6 pose parameters consist of 3 translation parameters and 3 rotation parameters. The translation parameters are simply the X , Y , and Z coordinates of the vehicle model coordinate system origin relative to the world coordinate system origin. Likewise, the 3 rotation parameters describe the 3 degrees of freedom in the orientation of the vehicle relative to the world coordinate system.

There are, however, several ways to parameterize these 3 degrees of freedom for rotation. The 3×3 rotation matrix representation is used to efficiently apply the rotation transformation to a large collection of 3-d points. A point $\mathbf{x}$ is rotated to $\mathbf{x}'$ by rotation matrix $\mathbf{R}$ as $\mathbf{x}' = \mathbf{R}\mathbf{x}$. Rotation matrices are a subspace of general 3×3 matrices. They form a mathematical group called the special orthogonal group, $SO(3)$, under the matrix multiplication operation. While useful in applying transformations, it is not a good idea to directly apply optimization techniques in the 9-d space of matrix parameters because unconstrained optimization will generally leave the manifold of proper rotations. A separate operation is needed to map the optimized result back onto the manifold. Applying this constraint after the optimization usually destroys the optimality. For the same reason, other over-parameterizations of rotation, like quaternion and axis-angle (each with 4 parameters), should be avoided in optimization. Euler angles are a minimal parameterization of rotation, but they too should be avoided in optimization. Euler angles describe any rotation as the concatenation of 3 rotations about the X , Y , and Z axes. Unfortunately, the derivatives of these parameters vary across the space of rotations. Some derivatives can even vanish causing a condition known as *gimbal lock*, which leads to unwanted constraints on the optimization.

The rotation parameterization used in this thesis for optimization is the rotation vector (sometimes called Rodrigues vector). The rotation vector, $\omega = [\omega_x \ \omega_y \ \omega_z]^\top$, is a 3-d vector with a direction equal to the axis of rotation and a magnitude, $\theta = \|\omega\|$, equal to the angle of rotation about that axis. This vector also happens to be the Lie algebra representation for the $SO(3)$ matrix group. The exponential map takes a rotation vector ω into a 3×3 rotation matrix $\mathbf{R}$. In the case of 3-d rotation, the exponential map has a closed form expression called Rodrigues' rotation formula:

$$\mathbf{R}(\omega) = e^{\omega_\times} = \mathbf{I} + \frac{\omega_\times}{\theta} \sin \theta + \frac{\omega_\times^2}{\theta^2} (1 - \cos \theta), \quad (4.1)$$

where $\omega_\times$ is the skew-symmetric cross-product matrix of ω :

$$\omega_\times = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}. \quad (4.2)$$

The inverse operation also exists in closed form. The angle of rotation is recovered from $\mathbf{R}$ as

$$\theta = 2 \cos^{-1} \left(\frac{\sqrt{\text{tr}(\mathbf{R}) + 1}}{2} \right), \quad (4.3)$$

where tr indicates the trace of a matrix (the sum of diagonal elements). Then the rotation vector, ω , is recovered from $\mathbf{R}$ as

$$\omega(\mathbf{R}) = \frac{\theta \mathbf{v}}{\|\mathbf{v}\|} \quad \text{with} \quad \mathbf{v} = \begin{bmatrix} R_{32} - R_{23} \\ R_{13} - R_{31} \\ R_{21} - R_{12} \end{bmatrix} \quad (4.4)$$

The rotation vector is the most appropriate parameterization for optimization. It is a minimal parameterization and every rotation vector represents a valid rotation (although the representation is not unique). Furthermore, the derivatives are always well defined. The derivatives are also easily computed since the Lie algebra, by definition, parameterizes the tangent space at the identity rotation. Differentiation is discussed in greater detail in Section 4.3.1.

The complete parameter vector for optimization contains the 3 translation parameters and 3 rotation vector parameters appended to the top n PCA parameters. The resulting parameter space is $n + 6$ -dimensional. It is sometimes useful to further constrain the problem by using only a subset of the pose parameters. Most commonly, one might assume that the vehicles are supported by a known ground plane. In this case, the camera is calibrated with respect to this plane at $Z = 0$. The remaining degrees of freedom are 2 translation (X and Y) and 1 rotation (about the Z axis). With a ground plane constraint, the parameter space is $n + 3$ -dimensional. In any case, the space of shape plus pose parameters is henceforth denoted n' -dimensional.

4.2 Detection of edge correspondence

To optimize the parameters, one must first establish a correspondence between the hypothesized model edges and the observed model edges. It is assumed that an initial guess of the parameter vector is available and is good enough to place the hypothesized edges nearby the correct corresponding edge observations. Therefore, the correspondences are found by local search in a neighborhood around each hypothesized edge. Detected sub-pixel Canny edges are stored in an edge map for the purpose of efficient search. The edge map is a 2-d array with the same dimensions as the original image. Each “pixel” in the array is either null if there is no edge, or contains the gradient magnitude, angle, and sub-pixel offset if there is an edge.

Drummond and Cipolla [25] use the idea of local search to find edge correspondence for real-time pose tracking of rigid 3-d models. They hypothesize image edges from the occluding contours of their model using a method similar to that described in Section 3.4. Points are then sampled along the hypothesized edges with uniform spacing. Each sample point is used as the origin of a 1-d search in the direction perpendicular to the hypothesized edge. Edge map pixels within a distance of κ pixels are examined. Figure 4.1 shows an example of sampling and search from their paper.

The trouble with sampling is that it does not find all the adjacent image edge points for the line segment describing a hypothesized edge. Sparse sampling can be an advantage when optimizing for speed in a real-time system; however, for best results one should use all of the available data.

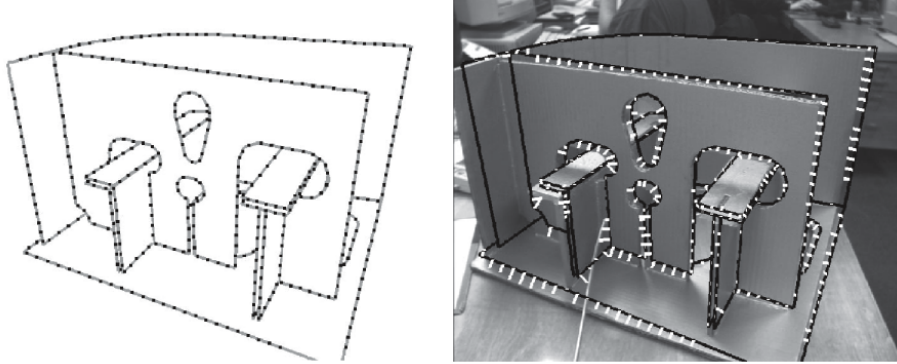


Figure 4.1: Sampling hypothesis edges for correspondence. Drummond and Cipolla use uniform sampling to determine the points from which to search for neighboring edges. Images adapted from Figure 3 of [25].

The trivial solution is to decrease the sample separation to about 1 pixel such that the search lines bunch up and cover all pixels in the neighborhood of a line segment (Figure 4.2 left). This method would work, but the implementation would need to be very careful not to miss pixels or count pixels twice because of roundoff in the search line rasterization. Furthermore, this order of searching the edge map pixels is not well aligned with how the array is stored in memory. The zig-zag-like search pattern about each hypothesized line segment is likely to decrease the cache hit ratio and degrade performance.

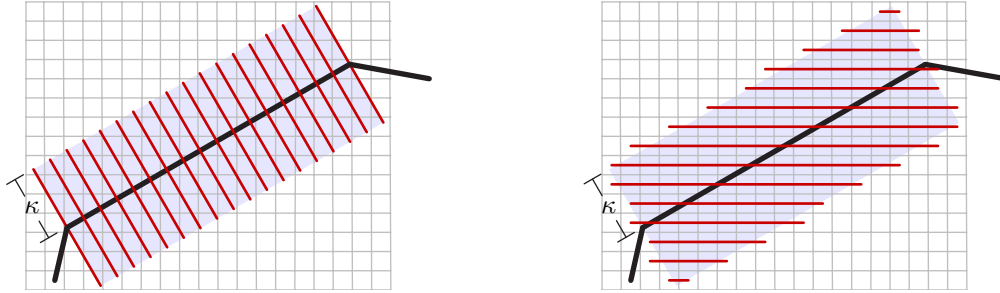


Figure 4.2: Search patterns for finding neighboring edge points. Edge map pixels are searched along red lines. On the left, search perpendicular to the line segment. On the right, search aligned with the image array. Both methods search the same region within κ pixels of the line segment.

The solution used in this thesis is to define an oriented search box around each hypothesized line segment and then raster scan the box along horizontal lines (Figure 4.2 right). The search box has a width of 2κ and a length equal to the length of the line segment. Compared to the sampling algorithm, the same pixels of the edge map are searched, but the order of examining the pixels is changed. Horizontal scanning accesses the pixel in the order they are stored in memory for improved cache performance. The algorithm also guarantees that each pixel in the neighborhood is examined exactly once. Any edge points found in the search region are assigned correspondence with the

hypothesized line segment. The actual point of correspondence on the hypothesized line segment is determined by projecting each edge point to the closest point on the line. This projection is discussed further in Section 4.3.1 and illustrated in Figure 4.3. As with the sampling algorithm, correspondence is perpendicular to the line orientation. The search distance, κ , is parameter of the algorithm. The choice of κ is discussed later when examining robustness to outliers.

4.3 Optimization of shape and pose

The optimization of vehicle shape and pose parameters follows the framework for fitting parameterized 3-d models to images put forth by Lowe [74] in the early 1990's. For some initial set of parameters, a set of hypothesis edges is generated. A collection of correspondences between observed and hypothesized edges is determined by local search. For now, the correspondences are assumed to be correct. Each corresponding edge point produces one error measurement. Each error measurement measures the signed perpendicular distance from one hypothesized edge to the observed edge. Perpendicular distance is used to account for the aperture problem. Correspondence of a point on one edge to another edge is not constrained along the direction parallel to that edge. Any motion along the parallel direction results a different, but equally good, correspondence.

The fitting problem is now formulated as the minimization of the magnitude of the vector of signed error measurements. The solution is obtained by linearizing the problem and iteratively solving a linear system to optimize the parameters in the least squares sense. This is a straightforward application of Gauss-Newton minimization. The problem is posed as

$$\mathbf{J}\delta\mathbf{p} = \mathbf{e}, \quad (4.5)$$

where $\mathbf{e}$ is the vector of signed errors, $\delta\mathbf{p}$ is the vector of incremental parameter updates, and $\mathbf{J}$ is the Jacobian matrix evaluated with respect to the current parameter vector. The elements of $\mathbf{J}$, $J_{ij} = \frac{\partial e_i}{\partial p_j}$, provide a first order approximation of how the i -th error responds to small changes in the j -th parameter. The least squares solution finds the $\delta\mathbf{p}$ that minimizes $\|\mathbf{J}\delta\mathbf{p} - \mathbf{e}\|^2$, the l^2 -norm of the residuals. The well known solution to the least squares problem is to form the normal equations

$$\mathbf{J}^\top \mathbf{J} \delta\mathbf{p} = \mathbf{J}^\top \mathbf{e} \quad (4.6)$$

and then invert the symmetric matrix on the left to get

$$\delta\mathbf{p} = (\mathbf{J}^\top \mathbf{J})^{-1} \mathbf{J}^\top \mathbf{e}. \quad (4.7)$$

The solution is used to update the parameter vector $\mathbf{p}_k$ at the k -th iteration by

$$\mathbf{p}_{k+1} = \mathbf{p}_k + \delta\mathbf{p}. \quad (4.8)$$

The new parameter vector $\mathbf{p}_{k+1}$ generates a new hypothesis for the next iteration. The correspondences, error measurements, and Jacobian matrix are recomputed before solving the linear system again. While the correspondences, errors, and Jacobian are all recomputed with each iteration, the

hypothesis of edges is made only once and held fixed until convergence. That is, selection of occluding contours and visible parts is based on the initial conditions. Small changes in model parameters can cause many edges to appear or disappear. Such rapid changes tend to cause discontinuities in the error function and parameter oscillations that prevent convergence.

The application of Gauss-Newton minimization is fairly straightforward. The only complicated aspect of the procedure (as presented thus far) is how to efficiently compute the Jacobian matrix. As shown below, the partial derivatives for each error measurement only depend on the position and orientation of the hypothesized edge point. They do not depend on the distance to the corresponding observation. Thus, most of the work in computing the Jacobian is independent of the selection of correspondences.

4.3.1 Jacobian matrix computation

The Jacobian matrix, $\mathbf{J}$, is an $m \times n'$ matrix of partial derivatives where m is the number of error measurements and n' is the number of model parameters. Each n' -d row vector, $\mathbf{J}_i$, relates to a different point on the vehicle surface. For computational efficiency, it is better to directly compute $\mathbf{J}^\top \mathbf{J}$ and $\mathbf{J}^\top \mathbf{e}$ for the normal equations in (4.6) instead of first computing $\mathbf{J}$ and $\mathbf{e}$. These matrices are computed incrementally by summation as each Jacobian row, $\mathbf{J}_i$, and corresponding error measure, e_i , are evaluated. The summation formulae are

$$\mathbf{J}^\top \mathbf{J} = \sum_{i=0}^m \mathbf{J}_i^\top \mathbf{J}_i \quad \text{and} \quad \mathbf{J}^\top \mathbf{e} = \sum_{i=0}^m \mathbf{J}_i^\top e_i \quad (4.9)$$

Typically $m \gg n'$, so storing the $n' \times n'$ and $n' \times 1$ matrices of the normal equations provides considerable memory savings over storing $\mathbf{J}$ and $\mathbf{e}$ explicitly. Other advantages of this incremental form are discussed later.

The computation of each $\mathbf{J}_i$ is easier when split into 3 composite functions. Let $E_i(\mathbf{p})$ be a function that generates the i -th error measurement, e_i , as a function of parameter vector $\mathbf{p}$. Hence the Jacobian of E_i is the i -th row vector of $\mathbf{J}$. One can decompose $E_i(\mathbf{p})$ as

$$e_i = E_i(\mathbf{p}) = E_i^1(E_i^2(E_i^3(\mathbf{p}))). \quad (4.10)$$

Working from inside out, E_i^3 generates the relevant i -th 3-d point, $\mathbf{x}_i$, from the model and parameters, $\mathbf{p}$. E_i^2 projects $\mathbf{x}_i$ into 2-d image coordinates $\mathbf{u}_i$. E_i^1 computes the perpendicular signed distance between $\mathbf{u}_i$ and the observation point. Using the chain rule, the Jacobian row vector, $\mathbf{J}_i$ is decomposed into the product

$$\mathbf{J}_i = \mathbf{J}_i^1 \mathbf{J}_i^2 \mathbf{J}_i^3, \quad (4.11)$$

where each term is the Jacobian of one of the composite functions. Specifically, $\mathbf{J}_i^1$ is the 1×2 Jacobian of E_i^1 , $\mathbf{J}_i^2$ is a 2×3 Jacobian of E_i^2 , and $\mathbf{J}_i^3$ is a $3 \times n'$ Jacobian of E_i^3 . These components are called the *perpendicular distance Jacobian*, the *projection Jacobian*, and the *3-d Jacobian* respectively. Each of these component Jacobians is derived below.

Perpendicular distance Jacobian

The signed error as a function of $\mathbf{u}_i$ is $E_i^1(\mathbf{u}_i) = \mathbf{n}_i^\top (\mathbf{u}_i - \mathbf{v}_i)$ where $\mathbf{n}_i$ is the unit normal of the edge at $\mathbf{u}_i$ and $\mathbf{v}_i$ is the fixed corresponding observation point. Assuming the normal is also fixed, the Jacobian with respect to $\mathbf{u}_i$ is simply $\mathbf{J}_i^1 = \mathbf{n}_i$. However, in practice, the hypothesis edge at $\mathbf{u}_i$ actually lies on a longer line segment defined by its endpoints $\mathbf{u}_a$ and $\mathbf{u}_b$. These endpoints are the projections of adjacent control points along a piecewise linear curve on the vehicle mesh surface. There may be many edge points corresponding to this same line segment between $\mathbf{u}_a$ and $\mathbf{u}_b$. Figure 4.3 illustrates this configuration. The unit normal, $\mathbf{n}_i$, is now common among all edges points sharing the line segment. This shared unit normal is denoted $\mathbf{n}_{ab}$ and is defined as perpendicular to $\mathbf{u}_b - \mathbf{u}_a$. In general, each endpoint can move independently, and $\mathbf{n}_{ab}$ is dependent on $\mathbf{u}_a$ and $\mathbf{u}_b$, so both $\mathbf{n}_{ab}$ and $\mathbf{u}_i$ vary with all 4 coordinates of $\mathbf{u}_a$ and $\mathbf{u}_b$. The derivation of this Jacobian is now considerably more complicated, yet the result turns out to be quite simple.

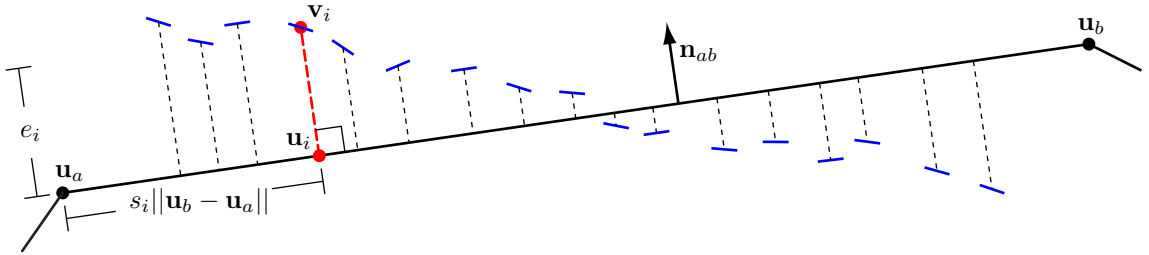


Figure 4.3: Perpendicular signed error measure for edges. Observed edge points (in blue) such as $\mathbf{v}_i$ are matched to $\mathbf{u}_i$, the closest point on the hypothesized line segment from $\mathbf{u}_a$ to $\mathbf{u}_b$. Perpendicular signed error, e_i , is the signed distance between $\mathbf{v}_i$ and $\mathbf{u}_i$ along the direction of the unit normal, $\mathbf{n}_{ab}$. Interpolation parameter $s_i \in [0, 1]$ locates $\mathbf{u}_i$ between $\mathbf{u}_a$ and $\mathbf{u}_b$.

The point $\mathbf{u}_i$ is described relative to $\mathbf{u}_a$ and $\mathbf{u}_b$ by an interpolation parameter, $s_i \in [0, 1]$. Explicitly,

$$\mathbf{u}_i = \mathbf{u}_a + (\mathbf{u}_b - \mathbf{u}_a)s_i. \quad (4.12)$$

The interpolation is determined such that $\mathbf{u}_i$ is the closest point on the line to $\mathbf{v}_i$, so

$$s_i = \frac{(\mathbf{u}_b - \mathbf{u}_a)^\top (\mathbf{v}_i - \mathbf{u}_a)}{\|\mathbf{u}_b - \mathbf{u}_a\|^2}. \quad (4.13)$$

This closest point also ensures that $\mathbf{v}_i - \mathbf{u}_i$ is perpendicular to $\mathbf{u}_b - \mathbf{u}_a$.

Some additional notation is needed to compactly describe the computation of a normal vector. Let $\mathbf{N}$ be the 2×2 matrix representing a 90° rotation in 2-d:

$$\mathbf{N} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}. \quad (4.14)$$

$\mathbf{N}\mathbf{w}$ is the vector normal to a vector $\mathbf{w}$ with the same magnitude as $\mathbf{w}$. The unit vector, $\mathbf{n}$, normal

to $\mathbf{w}$ is $\mathbf{n} = \frac{\mathbf{N}\mathbf{w}}{\|\mathbf{w}\|}$. In particular, the unit normal to the line segment from $\mathbf{u}_a$ to $\mathbf{u}_b$ is

$$\mathbf{n}_{ab} = \frac{\mathbf{N}(\mathbf{u}_b - \mathbf{u}_a)}{\|\mathbf{u}_b - \mathbf{u}_a\|} \quad (4.15)$$

Note that $\mathbf{N}$ has a few special properties: $\mathbf{N}^\top = -\mathbf{N}$, $\mathbf{N}^\top \mathbf{N} = \mathbf{I}$, and $\mathbf{w}^\top \mathbf{N} \mathbf{w} = 0$ for any vector $\mathbf{w}$. Using this notation, one can rewrite the error function E_i^1 in terms of $\mathbf{u}_a$ and $\mathbf{u}_b$

$$E_i^1(\mathbf{u}_a, \mathbf{u}_b) = \frac{(\mathbf{N}(\mathbf{u}_b - \mathbf{u}_a))^\top}{\|\mathbf{u}_b - \mathbf{u}_a\|} (\mathbf{u}_a + (\mathbf{u}_b - \mathbf{u}_a)s_i - \mathbf{v}_i). \quad (4.16)$$

By expanding and applying the properties of $\mathbf{N}$, the above equation simplifies to

$$E_i^1(\mathbf{u}_a, \mathbf{u}_b) = \frac{\mathbf{u}_a^\top \mathbf{N} \mathbf{u}_b + \mathbf{u}_b^\top \mathbf{N} \mathbf{v}_i + \mathbf{v}_i^\top \mathbf{N} \mathbf{u}_a}{\|\mathbf{u}_b - \mathbf{u}_a\|}. \quad (4.17)$$

The partial derivatives of this function with respect to $\mathbf{u}_a$ and $\mathbf{u}_b$ are

$$\frac{\partial}{\partial \mathbf{u}_a} E_i^1(\mathbf{u}_a, \mathbf{u}_b) = \frac{\mathbf{N}(\mathbf{u}_b - \mathbf{v}_i)}{\|\mathbf{u}_b - \mathbf{u}_a\|} + \frac{(\mathbf{u}_b - \mathbf{u}_a) E_i^1(\mathbf{u}_a, \mathbf{u}_b)}{\|\mathbf{u}_b - \mathbf{u}_a\|^2} \quad (4.18)$$

$$\frac{\partial}{\partial \mathbf{u}_b} E_i^1(\mathbf{u}_a, \mathbf{u}_b) = \frac{-\mathbf{N}(\mathbf{u}_a - \mathbf{v}_i)}{\|\mathbf{u}_b - \mathbf{u}_a\|} - \frac{(\mathbf{u}_b - \mathbf{u}_a) E_i^1(\mathbf{u}_a, \mathbf{u}_b)}{\|\mathbf{u}_b - \mathbf{u}_a\|^2} \quad (4.19)$$

These partial derivatives can be simplified further by expressing $\mathbf{v}_i$ relative to $\mathbf{u}_a$ and decomposing into the perpendicular and parallel components. That is,

$$\mathbf{v}_i = \mathbf{u}_a + (\mathbf{u}_b - \mathbf{u}_a)s_i + \mathbf{n}_{ab} E_i^1(\mathbf{u}_a, \mathbf{u}_b) \quad (4.20)$$

Substituting (4.20) into (4.18) and (4.19) and simplifying results in

$$\frac{\partial}{\partial \mathbf{u}_a} E_i^1(\mathbf{u}_a, \mathbf{u}_b) = (1 - s_i) \mathbf{n}_{ab} \quad \text{and} \quad \frac{\partial}{\partial \mathbf{u}_b} E_i^1(\mathbf{u}_a, \mathbf{u}_b) = s_i \mathbf{n}_{ab} \quad (4.21)$$

This solution indicates that the Jacobian $\mathbf{J}_i^1$ is indeed still the unit normal vector (now denoted $\mathbf{n}_{ab}$). Furthermore, it proves that it is sufficient to evaluate the 3-d and projection Jacobians ($\mathbf{J}_i^2$ and $\mathbf{J}_i^3$) only at the control points, and then linearly interpolate the results for each edge point. In other words,

$$\mathbf{J}_i = \mathbf{n}_{ab}^\top ((1 - s_i) \mathbf{J}_a^2 \mathbf{J}_a^3 + s_i \mathbf{J}_b^2 \mathbf{J}_b^3) \quad (4.22)$$

The result of (4.22) allows for considerable computational savings. The number of curve control points is typically much less than m , so the number of the 3-d and projections Jacobians to compute is reduced. Furthermore, one can reorder the summations in (4.9) to reduce computations. Let l be the number of line segments and $m(j)$ be the number of error measures associated with the j -th line segment. The summations can now be written

$$\mathbf{J}^\top \mathbf{J} = \sum_{j=0}^l \left(\mathbf{J}_{a(j)}^\top \mathbf{J}_{a(j)} \left(\sum_{i=0}^{m(j)} (1 - s_i) \right) + \mathbf{J}_{b(j)}^\top \mathbf{J}_{b(j)} \left(\sum_{i=0}^{m(j)} s_i \right) \right) \quad (4.23)$$

$$\mathbf{J}^\top \mathbf{e} = \sum_{j=0}^l \left(\mathbf{J}_{a(j)}^\top \left(\sum_{i=0}^{m(j)} (1 - s_i) \mathbf{e}_i \right) + \mathbf{J}_{b(j)}^\top \left(\sum_{i=0}^{m(j)} s_i \mathbf{e}_i \right) \right). \quad (4.24)$$

The total number of terms remains the same, but now the inner summations are computed over scalar values. These scalar values then multiply matrices for each control point. This factorization reduces the number of outer products, scalar-matrix multiplications, and matrix additions.

Projection Jacobian

The projection Jacobian, $\mathbf{J}_i^2$, describes the change in 2-d coordinates of image point $\mathbf{u}_i$ with respect to changes in the 3-d world coordinates of preimage point $\mathbf{x}_i$. The following derivation is valid for all points in space, but from the last section it is clear that, in practice, i refers to a control point on a piecewise linear curve on the vehicle surface.

Models of image projection were discussed in Section 2.1. The derivation of the projection Jacobian is derived with respect to the general model of projection that covers both affine and perspective projection. This model is the 3×4 projection matrix, $\mathbf{P}$, in homogeneous coordinates. The projection Jacobian is expressed in inhomogeneous coordinates, so the projection equation must be expressed in inhomogeneous form before computing derivatives. To that end, the projection matrix is written as

$$\mathbf{P} = \begin{bmatrix} \mathbf{m}_1^\top & t_1 \\ \mathbf{m}_2^\top & t_2 \\ \mathbf{m}_3^\top & t_3 \end{bmatrix}, \quad (4.25)$$

where $\mathbf{m}_1$, $\mathbf{m}_2$, and $\mathbf{m}_3$ are each 3-d vectors. Assuming that all 3-d world points of interest are finite, the homogeneous form of $\mathbf{x}_i$ is $[\mathbf{x}_i^\top \ 1]^\top$. Therefore, in inhomogeneous coordinates, the projection of $\mathbf{x}_i$ to $\mathbf{u}_i$ is

$$\mathbf{u}_i = E_i^2(\mathbf{x}_i) = \frac{1}{\mathbf{m}_3^\top \mathbf{x}_i + t_3} \begin{bmatrix} \mathbf{m}_1^\top \mathbf{x}_i + t_1 \\ \mathbf{m}_2^\top \mathbf{x}_i + t_2 \end{bmatrix}. \quad (4.26)$$

Taking the derivative with respect to $\mathbf{x}_i$, the projection Jacobian is

$$\mathbf{J}_i^2 = \frac{1}{(\mathbf{m}_3^\top \mathbf{x}_i + t_3)^2} \begin{bmatrix} \mathbf{x}_i^\top (\mathbf{m}_3 \mathbf{m}_1^\top - \mathbf{m}_1 \mathbf{m}_3^\top) + \mathbf{m}_1^\top t_3 - \mathbf{m}_3^\top t_1 \\ \mathbf{x}_i^\top (\mathbf{m}_3 \mathbf{m}_2^\top - \mathbf{m}_2 \mathbf{m}_3^\top) + \mathbf{m}_2^\top t_3 - \mathbf{m}_3^\top t_2 \end{bmatrix}. \quad (4.27)$$

In (4.27) there are several terms that depend only on the camera parameters. The camera parameters are the same for all i , so it is more efficient to compute these terms once and then use them for each $\mathbf{x}_i$. These pre-computed terms are, for $k = \{1, 2\}$, skew-symmetric matrices $\mathbf{A}_k = \mathbf{m}_3 \mathbf{m}_k^\top - \mathbf{m}_k \mathbf{m}_3^\top$ and vectors $\mathbf{b}_k = \mathbf{m}_k^\top t_3 - \mathbf{m}_3^\top t_k$. Using the pre-computed terms

$$\mathbf{J}_i^2 = \frac{1}{(\mathbf{m}_3^\top \mathbf{x}_i + t_3)^2} \begin{bmatrix} \mathbf{x}_i^\top \mathbf{A}_1 + \mathbf{b}_1^\top \\ \mathbf{x}_i^\top \mathbf{A}_2 + \mathbf{b}_2^\top \end{bmatrix}. \quad (4.28)$$

In practice, only the perspective projection model is used, which requires the general form of the equations above. Note, however, that (4.27) simplifies significantly in the special case of affine projection. Recall from Section 2.1 that an affine projection matrix is defined by $\mathbf{m}_3 = \mathbf{0}$ with $t_3 \neq 0$. In this case, $\mathbf{J}_i^2$ reduces to $[\mathbf{m}_1 \ \mathbf{m}_2]^\top / t_3$, which is independent of $\mathbf{x}_i$.

3-d Jacobian

The 3-d Jacobian, $\mathbf{J}_i^3$, describes the change in the 3-d position of $\mathbf{x}_i$ with respect to changes in the parameter vector $\mathbf{p}$. The computation of each element of $\mathbf{J}_i^3$ varies depending on the type of parameter (shape, rotation, or translation) and how $\mathbf{x}_i$ is attached to the vehicle surface (occluding contour

or part boundary). To simplify the notation, the 3-d Jacobian is written as $\mathbf{J}_i^3 = [\mathbf{J}_{s,i}^3 \ \mathbf{J}_{t,i}^3 \ \mathbf{J}_{r,i}^3]$ with the $3 \times n$, 3×3 , and 3×3 sub-matrices corresponding to the shape, translation, and rotation parameters respectively. The use of all pose parameters is assumed here. In the case of a ground plane or other pose constraint, the relevant columns are removed.

To further simplify the 3-d Jacobian, one can always compute the Jacobian relative to the canonical vehicle pose with no translation or rotation. This simplification is possible because one can always find a new camera projection matrix relative to the vehicle coordinate system. That relative camera is computed by a right multiplication of the world camera with a 4×4 homogeneous rigid transformation matrix

$$\mathbf{P}_{\text{rel}} = \mathbf{P} \begin{bmatrix} \mathbf{R}(\omega) & \mathbf{t} \\ 0 & 1 \end{bmatrix}, \quad (4.29)$$

where $\mathbf{t}$ is the current translation and $\mathbf{R}(\omega)$ is the current rotation matrix computed from rotation vector ω by Equation (4.1). Using $\mathbf{P}_{\text{rel}}$ instead of $\mathbf{P}$ in the computation of the projection Jacobian accounts for the absolute vehicle pose in a trivial way. The rest of this section treats $\mathbf{x}_i$ as relative to vehicle coordinate system.

In vehicle coordinates, the 3-d pose Jacobians are trivial. The translation Jacobian is the identity matrix, $\mathbf{J}_{t,i}^3 = \mathbf{I}$. The rotation Jacobian is the transposed cross-product matrix of the position vector, $\mathbf{J}_{r,i}^3 = \mathbf{x}_i^\top \times$ (the definition of a cross-product matrix is given in Equation (4.2)). The latter result follows from the use of the Lie algebra to parameterize rotation. At the identity rotation, the rotation vector lies in the tangent space. In the tangent space, an infinitesimal rotation is computed by small angle approximation to Rodrigues formula (4.1) as

$$\mathbf{R}(\omega)\mathbf{x}_i \approx (\mathbf{I} + \omega \times)\mathbf{x}_i = \mathbf{x}_i + \omega \times \mathbf{x}_i = \mathbf{x}_i + \mathbf{x}_i^\top \times \omega. \quad (4.30)$$

The derivative with respect to ω follows trivially from this expression.

The computation of the 3-d shape Jacobian depends on how $\mathbf{x}_i$ is connected to the vehicle surface. In the simplest case, $\mathbf{x}_i$ is a vertex on the vehicle mesh. In Section 3.5.1, the matrix $\mathbf{G}$ was defined that linearly maps n -d PCA parameter vector $\mathbf{p}_s$ to full shape space vector $\mathbf{s}$ containing the concatenation of all 3-d vertex positions. If $\mathbf{x}_i$ corresponds to the k -th mesh vertex, it follows that there is a $3 \times n$ sub-matrix $\mathbf{G}_k$ of $\mathbf{G}$ such that $\mathbf{x}_i = \mathbf{G}_k \mathbf{p}_s$. Thus, the 3-d shape Jacobian is simply $\mathbf{J}_{s,i}^3 = \mathbf{G}_k$. This simple case covers the occluding contours. As defined in Section 3.4, the occluding contours are always comprised of a subset of mesh edges. Therefore, the control points of these curves are always mesh vertices.

A generalization of the simple case is needed for part boundaries. The vertices of part boundaries are mapped onto the surface from texture space. They can map to arbitrary points on the mesh surface using barycentric coordinates (Section 3.1.2). Since barycentric coordinates and shape deformation are both linear, the 3-d shape Jacobian at an arbitrary point is the barycentric interpolation of the Jacobians at the triangle mesh vertices. Thus,

$$\mathbf{J}_{s,i}^3 = (1 - u - v)\mathbf{G}_{a_i} + u\mathbf{G}_{b_i} + v\mathbf{G}_{c_i} \quad (4.31)$$

where (u, v) are the barycentric coordinates and a_i, b_i, c_i are the indices of the mesh vertices that comprise the triangle containing $\mathbf{x}_i$. Equation (4.31) is still not complete for part boundaries because it does not account for the motion tangent to the surface generated by part deformations in texture space. Another sub-matrix $\mathbf{G}_j$ of $\mathbf{G}$, this time $2 \times n$, describes the linear transformation of the j -th part vertex in the texture space. Each point of interest on a part boundary corresponds to either a part vertex or the intersection of a part polygon with a mesh edge (Section 3.3.1). Analogous to the 3-d case, the $\mathbf{G}_j$ are linearly interpolated at intersection points from the adjacent part vertices. Each column vector of $\mathbf{G}_j$ is mapped into 3-d using barycentric coordinates, and the resulting Jacobian matrix is added to the result in (4.31).

4.3.2 Regularization

Lowe’s optimization framework [74] also includes an extension for regularization using a prior model of the parameter values. For the deformable vehicle model, a normal distribution on the shape parameters is readily available as a byproduct of PCA (see Section 3.5.2). The Gauss-Newton minimization works well when given significantly more correct edge correspondences than parameters and no incorrect correspondences. However, a large number of correct correspondences is initially not available in practice. Instead, a large fraction of the correspondences may be incorrect, and it is initially difficult to separate the good from the bad. With many bad correspondences, Gauss-Newton is likely to pull the solution to an incorrect local minimum that may differ significantly from any realistic vehicle shape and pose. The solution is to incorporate a prior distribution on the parameters as a soft constraint to regularize the optimization.

Regularization is applied by appending a few more rows to the Jacobian and error vector. The regularized linear system is

$$\begin{bmatrix} \mathbf{J} \\ \mathbf{I} \end{bmatrix} \delta \mathbf{p} = \begin{bmatrix} \mathbf{e} \\ \mathbf{d} \end{bmatrix} \quad (4.32)$$

where $\mathbf{I}$ is the $n' \times n'$ identity matrix and $\mathbf{d}$ is a vector of the desired values for $\delta \mathbf{p}$. The least squares solution now penalizes $\delta \mathbf{p}$ from differing from $\mathbf{d}$; however, it does not penalize all parameters equally. A large deviation in one parameter may be more acceptable than a large deviation in another. Furthermore, the error measures in $\mathbf{e}$ are measured in units of pixels while the desired parameter values lie in a space with arbitrary and unrelated units.

To put all the equations on equal footing, each equation in the linear system should be normalized to a unit standard deviation. All the error measurements in $\mathbf{e}$ are in the same image coordinates. One pixel is already a good approximation of the standard deviation in measuring image feature locations, so $\mathbf{J}$ and $\mathbf{e}$ are left unscaled. On the other hand, the shape and pose parameters do need to be normalized. The standard deviations of the shape parameters were computed in diagonal matrix $\mathbf{\Sigma}$ as a byproduct of PCA in Section 3.5.1. Hence, one can normalize the regularization terms by multiplying by $\mathbf{\Sigma}^{-1}$. The standard deviations and desired values for pose parameters have not yet been determined. If no pose prior is available, the pose regularization rows can be omitted from the linear system. In some applications, a prior on pose may be useful to help keep the vehicle on a

known roadway or from deviating too far from its last known pose. In a slight abuse of notation Σ will be used to indicate all parameter standard deviations, both shape and pose.

Even with normalization of the regularization terms, the influence of regularization is uncontrolled. The number of parameters is fixed, but the number of edge error measurements is variable. When the number of error measurements is doubled, the influence of regularization in the linear system is halved. Furthermore, one might wish to control the amount of regularization during the optimization. To that end, a control parameter, λ is introduced to weight all prior model terms and control the influence of regularization.

Taking normalization and regularization control into account, the updated linear system is

$$\begin{bmatrix} \mathbf{J} \\ \lambda \Sigma^{-1} \end{bmatrix} \delta \mathbf{p} = \begin{bmatrix} \mathbf{e} \\ \lambda \Sigma^{-1} \mathbf{d} \end{bmatrix}. \quad (4.33)$$

The least squares solution to this equation amounts to minimizing

$$\|\mathbf{J} \delta \mathbf{p} - \mathbf{e}\|^2 + \lambda^2 \|\Sigma^{-1}(\delta \mathbf{p} - \mathbf{d})\|^2. \quad (4.34)$$

This particular form of regularization is known as ridge regression or Tikhonov [106] regularization. The desired parameters, $\mathbf{d}$, should be set such that the solution $\delta \mathbf{p}$ shifts the parameters to those of the mean vehicle. By construction the PCA parameters are zero mean, $\mu_{\mathbf{p}} = 0$, so it follows from (4.8) that $\mathbf{d} = -\mathbf{p}$. In other words, relative to the current point in parameter space, $\mathbf{p}$, the mean of the parameters is located at $\mu_{\mathbf{p}} = -\mathbf{p}$. The covariance matrix of the parameters as defined in Section 3.5.2 is $\Sigma_{\mathbf{p}} = \Sigma^T \Sigma$. Thus, (4.34) can be written

$$\|\mathbf{J} \delta \mathbf{p} - \mathbf{e}\|^2 + \lambda^2 (\delta \mathbf{p} + \mathbf{p})^T \Sigma_{\mathbf{p}}^{-1} (\delta \mathbf{p} + \mathbf{p}). \quad (4.35)$$

Note the strong similarity between (4.35) and the minimization of log-likelihood terms in a maximum a posteriori (MAP) estimate. For fixed λ , regularization is equivalent to a probabilistic interpretation where the normal distribution, $\mathcal{N}(-\mathbf{p}, \lambda^2 \Sigma_{\mathbf{p}})$, over the parameters is used as a prior in a MAP estimate.

The normal equations formed from (4.33) are

$$(\mathbf{J}^T \mathbf{J} + \lambda^2 \Sigma_{\mathbf{p}}^{-1}) \delta \mathbf{p} = \mathbf{J}^T \mathbf{e} - \lambda^2 \Sigma_{\mathbf{p}}^{-1} \mathbf{p}. \quad (4.36)$$

As a result, regularization adds little to the computational burden. Recall that $\Sigma_{\mathbf{p}}^{-1}$ is a diagonal matrix with the j -th diagonal element equal to σ_j^{-2} . All that is required is adding $(\frac{\lambda}{\sigma_j})^2$ to the diagonal of $\mathbf{J}^T \mathbf{J}$ and subtracting $(\frac{\lambda}{\sigma_j})^2 p_j$ from the elements of $\mathbf{J}^T \mathbf{e}$ before solving the system.

As discussed by Lowe, Tikhonov regularization also bears a strong similarity to the Levenberg-Marquardt algorithm [76] for numerical optimization. Levenberg-Marquardt also uses a control parameter, λ , to add a damping term to the diagonal of $\mathbf{J}^T \mathbf{J}$. The modified normal equations used in Levenberg-Marquardt are

$$(\mathbf{J}^T \mathbf{J} + \lambda \mathbf{I}) \delta \mathbf{p} = \mathbf{J}^T \mathbf{e}, \quad (4.37)$$

where $\mathbf{I}$ is the identity matrix. When λ goes to zero, the algorithm reduces to Gauss-Newton minimization. When λ becomes large, the step size becomes smaller and turns toward the direction

of steepest descent. That is, $\delta \mathbf{p} \approx \lambda^{-1} \mathbf{J}^\top \mathbf{e}$ for very large λ . In the Levenberg-Marquardt algorithm, λ is adapted toward zero by dividing it by a constant as long as the resulting solution leads to a decrease in $\mathbf{e}^\top \mathbf{e}$. Thus, the algorithm acts like Gauss-Newton for fast convergence as long as decreasing error is achieved. When $\mathbf{e}^\top \mathbf{e}$ fails to decrease, λ is increased until a solution is found that does decrease $\mathbf{e}^\top \mathbf{e}$. Such a λ always exists because gradient descent always heads toward the minimum, albeit slowly.

Lowe argues in [74] that the Tikhonov regularization (*i.e.* Equation (4.36)) performs much better than Levenberg-Marquardt (*i.e.* Equation (4.37)) because regularization uses prior knowledge in the form of $\Sigma_{\mathbf{P}}$ that is unavailable to Levenberg-Marquardt. It may be true that $\Sigma_{\mathbf{P}}$ provides useful prior knowledge, but a direct comparison between the algorithms is fundamentally unfair. The critical difference stems from the use of λ on *both* sides of (4.36) and only *one* side of (4.37). Levenberg-Marquardt is designed to minimize $\mathbf{e}^\top \mathbf{e}$. Increasing λ ensures that error decreases monotonically with each step. Tikhonov regularization is designed to stabilize the solution by relying on a prior model. Increasing λ pulls the solution toward the prior model and often increases $\mathbf{e}^\top \mathbf{e}$.

Tikhonov regularization and Levenberg-Marquardt optimization cannot be treated as competing methods. These ideas are orthogonal to each other. In fact, one can even apply Levenberg-Marquardt optimization to a Tikhonov regularized system. Koller *et al.* [59] do just that for fitting a vehicle model to images. The normal equations for the combined methods are

$$(\mathbf{J}^\top \mathbf{J} + \lambda^2 \Sigma_{\mathbf{P}}^{-1} + \alpha \mathbf{I}) \delta \mathbf{p} = \mathbf{J}^\top \mathbf{e} - \lambda^2 \Sigma_{\mathbf{P}}^{-1} \mathbf{p}. \quad (4.38)$$

where α now represents the Levenberg-Marquardt control parameter. Note that the gradient direction is now $\mathbf{J}^\top \mathbf{e} - \lambda^2 \Sigma_{\mathbf{P}}^{-1} \mathbf{p}$, and the error value that Levenberg-Marquardt checks to see if the residual is decreasing is $\mathbf{e}^\top \mathbf{e} + \lambda^2 \mathbf{p}^\top \Sigma_{\mathbf{P}}^{-1} \mathbf{p}$. Levenberg-Marquardt produces the same result as Gauss-Newton for well behaved functions. However, Levenberg-Marquardt is more robust when the initialization is far from the minimum and the error function is not well approximated by a second order Taylor expansion. If the error function is well behaved, then Gauss-Newton produces equally good results and does so with fewer iterations. Section 4.3.4 explains why Levenberg-Marquardt is not needed in the vehicle fitting algorithm.

4.3.3 Robustness to outliers

As described so far, the optimization algorithm assumes that all edge correspondences are correct. Such an ideal situation is never available in reality. For each hypothesized edge it is likely that a large number of the nearby edge observations are not correct matches. These invalid correspondences are **outliers** in the estimation data. Some outliers should be expected for any size search radius, κ , and even more should be expected as κ increases. To make matters worse, a large κ is exactly what is needed initially to compensate for uncertainty in the parameter initialization. It is well known that even a small number of outliers can have a large negative impact on the solutions to linear systems, so robust estimate techniques are needed.

There is not enough evidence available to make a hard decision about whether a given measurement is an outlier or not. Such a decision cannot be made until the model parameters are recovered (and even then outlier classification is not always correct). There are two common approaches used to side step this problem. The first approach is to randomly select subsets of measurements to use in the optimization. These algorithms are based on RANSAC (Random Sample Consensus) [31]. Each subset leads to a solution, and the solution that agrees with the most measurements is chosen. The second approach is to use all measurements but assign a score to each that represents the confidence that the measurement is an inlier. The scores are then used as weights in the linear system. In least squares estimations this is called iteratively reweighted least squares (IRLS). At each iteration the current solution is used to reevaluate the weights.

The RANSAC approach requires a large number of random subsets to ensure a correct solution with high probability. The number of subsets increases with both the number of unknown parameters and the proportion of outliers in the data. RANSAC is best suited for problems with relatively few parameters and a fast optimization routine for each measurement subset. Optimizing vehicle shape and pose does not fit this description. The optimization is already an iterative process and lends itself more to IRLS, so this is the approach that is taken.

Assume, for now, that a weight $w_i \in [0, 1]$ has been determined for each residual i . One can arrange these weights into a diagonal matrix $\mathbf{W}$. Now the weighted version of the Equation (4.33) is

$$\begin{bmatrix} \mathbf{WJ} \\ \lambda \mathbf{\Sigma}^{-1} \end{bmatrix} \delta \mathbf{p} = \begin{bmatrix} \mathbf{We} \\ -\lambda \mathbf{\Sigma}^{-1} \mathbf{p} \end{bmatrix}, \quad (4.39)$$

and the weighted normal equations (with regularization) are

$$(\mathbf{J}^\top \mathbf{W}^2 \mathbf{J} + \lambda^2 \mathbf{\Sigma}_{\mathbf{p}}^{-1}) \delta \mathbf{p} = \mathbf{J}^\top \mathbf{W}^2 \mathbf{e} - \lambda^2 \mathbf{\Sigma}_{\mathbf{p}}^{-1} \mathbf{p}. \quad (4.40)$$

The increase in computation for incorporating weights is insignificant because $\mathbf{W}^2$ is also a diagonal matrix. In practice, the scalar terms in the inner loops of Equations (4.23) and (4.24) are simply multiplied by w_i^2 . The number residuals and their respective weights are likely to change with each iteration. This means the influence of regularization will also change if not accounted for by adjusting λ . To keep regularization consistent, the weights are normalized by dividing $\mathbf{W}$ by $\|\mathbf{W}\| = \sqrt{\sum_i w_i^2}$. This operation is equivalent to multiplying λ by $\|\mathbf{W}\|$. With normalized weights λ need not be adjusted between IRLS iterations.

There are many ways to assign weights in IRLS. One particularly well-studied approach is the use of M-estimators [100] for robust statistics. M-estimators are generalizations of the least squares problem where the quadratic function is replaced by a more general function, ρ , that grows sub-quadratically. For example, one would seek to minimize $\sum_i \rho(r_i)$ instead of $\sum_i \|r_i\|^2$, where r_i is the i -th residual. In the case of fitting the vehicle model r_i is the i -th element of $\mathbf{J} \delta \mathbf{p} - \mathbf{e}$ ¹. The M-estimator equation is minimized by equating the derivative to zero, $\sum_i \rho'(r_i) = 0$, and solving.

¹At each iteration, k , the Jacobian is computed relative to the current parameter estimate $\mathbf{p}_k$. Thus, the current estimate of $\delta \mathbf{p}$ (before minimization) is always $\mathbf{0}$. It follows that the residuals reduce to $r_i = -e_i$.

Observe that if $w_i(r_i) = \rho'(r_i)/r_i$ and the w_i are assumed to be temporarily fixed, then the M-estimator problem is equivalent to the weighted least squares problem of minimizing $\sum_i \|w_i r_i\|^2$. The weights are a function of the residuals, but the updated residuals are a function of weights (via the weighted least squares solution). Thus, IRLS alternates between estimating the weights and minimizing the residuals following the standard expectation-maximization (EM) framework. The weights should decrease as a function of increasing residual since ρ is sub-quadratic. Intuitively, small residuals indicate trustworthy data which should be weighted more strongly. Large residuals indicate data that are more likely to be outliers and should be down-weighted. In standard least squares, these distant data points have a large influence on the solution.

There are several common choices for the M-estimator function. Drummond and Cipolla [25] use a variant of the Cauchy distribution to robustly weight their edge matches. The Cauchy distribution, like the normal distribution, has tails that tends toward zero in the limit as the residual distance increases to infinity. This means that every edge match should be considered with some (possibly small) weight no matter how large the distance. Such distributions do not agree with the practice of limiting correspondence search to a finite distance, κ . If the weighing function has a non-zero value at κ then the minimization function will have discontinuities whenever a correspondence crosses the κ threshold.

One special class of M-estimators generate weights that smoothly reach zero after a finite distance. These are called *redescending* M-estimators. A popular redescending M-estimator is the Beaton-Tukey biweight [5]. This M-estimator is the one used in this thesis. The Beaton-Tukey function and corresponding weight are

$$\rho(r) = \begin{cases} \frac{a^2}{6} (1 - (1 - (\frac{r}{a})^2)^3) & \text{if } |r| \leq a \\ \frac{a^2}{6} & \text{if } |r| > a \end{cases} \quad \text{and} \quad w(r) = \begin{cases} (1 - (\frac{r}{a})^2)^2 & \text{if } |r| \leq a \\ 0 & \text{if } |r| > a \end{cases}. \quad (4.41)$$

Figure 4.4 shows the Cauchy and Beaton-Tukey functions and weights compared to standard least squares. The parameter a is the value above which the Beaton-Tukey weight is zero. All potential edge matches greater than a pixels from a hypothesis edge will have no influence in the optimization, so there is no need to even search for them. Thus, the search radius, κ , is set equal to a . Since the weights are zero at a distance of κ there will be no discontinuities in the minimization function as edges pass beyond the search radius.

The choice of a (and therefore κ) is determined by the scale of the residuals, σ_r . Typically $a \approx 4\sigma_r$ is used [44]. The scale is a measure of the standard deviation of the true inlier data. Earlier it was assumed that 1 pixel is an appropriate standard deviation for all edge positions. When the vehicle model is fit perfectly to the image, $\sigma_r = 1$ is also a reasonable estimate for all residuals. However, the standard deviation will be considerably larger initially when the model is not correctly aligned. In the optimization routine, σ_r is initially determined by the uncertainty in the vehicle model parameters. The scale is gradually reduced to a minimum value of $\sigma_r = 1$ (See Section 4.3.4.). Throughout the process $\kappa = a = 4\sigma_r$.

In addition to weights determined by the M-estimator, there are other characteristics of edges

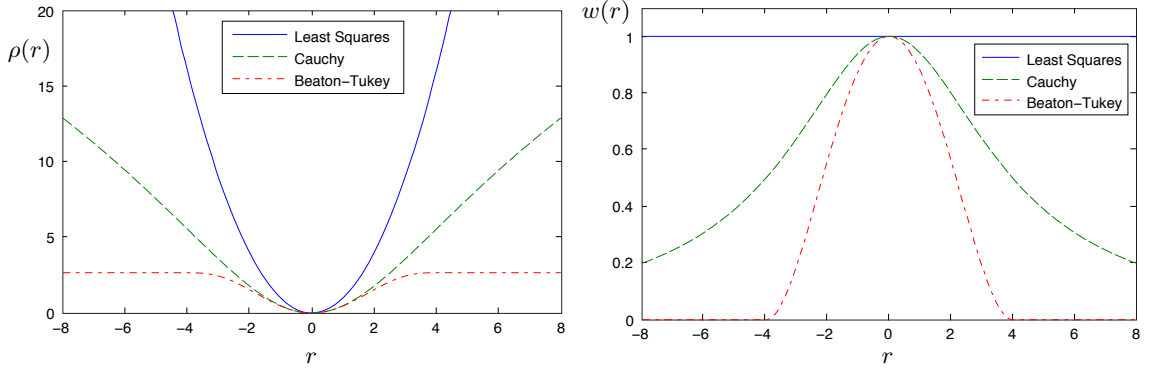


Figure 4.4: M-estimator, $\rho(r)$, and associated weight, $w(r)$, as functions of residual, r . For Beaton-Tukey, $a = 4$ in this plot.

that should factor into the weight of each correspondence. Most of these weight modifications were proposed by Drummond and Cipolla [25] and are used with little modification in this thesis. These characteristics are edge strength, edge orientation, multiple correspondence, and proximity to image boundaries. Each characteristic generates its own weight in the range $[0, 1]$. These weights are multiplied into the M-estimator weight. In practice, any edge that is assigned a zero weight is ignored for efficiency.

Edge strength

Every detected edge has an associated edge strength determined by the magnitude of the intensity gradient. Very weak edges are likely to be corrupted by sensor noise. Some weak edges may have significant error in position, others may be generated solely from noise with no underlying structure. Strong edges are less susceptible to noise corruption and should therefore have more weight in the optimization. However, there is no reason to assume that very strong edges should be any more trustworthy than moderately strong edges. All edges should be treated equally if they are strong enough to overcome sensor noise. To that end, two thresholds are used T_{s1} and T_{s2} with $T_{s1} < T_{s2}$. All edges with strength below T_{s1} are assumed to arise solely from noise and are assigned a weight of zero. All edges with strength above T_{s2} are assumed to be negligibly affected by noise and are assigned a weight of one. All edges with strength between T_{s1} and T_{s2} are assigned a weight that varies linearly from zero to one. These values are set manually to $T_{s1} = 5$ and $T_{s2} = 30$ (relative to intensities measured in the range $[0, 255]$).

Edge orientation

Every detected edge also has an associated orientation determined by the direction of the intensity gradient. The orientation of the hypothesized edges should agree with the orientation of corresponding detected edges up to a sign reversal. The gradient direction of some edges may reverse under

different conditions (*e.g.* vehicle color, or lighting conditions) so only 180° of orientation are considered. The gradient direction is known to be a poor approximation of the true edge normal, and more complex algorithms are known to produce better orientation estimates [102]. However, the extra computation is not likely to provide much benefit since there is also significant error in the initial orientation of the hypothesized edges. The absolute difference in angle between the hypothesized and detected edges is computed (in the range $[0, \pi/2]$). Correspondences with an angle difference less than 0.5 radians ($\approx 28^\circ$) are assigned a weight of one. Correspondences with a larger angle difference are assigned a weight of zero.

Multiple correspondence

A single point on a hypothesized line segment may correspond to more than one detected edge point. Clearly no more than one of these correspondences can be the correct match, but there is no way to identify which is correct. The solution is to divide the consideration equally amongst the matches by assigning each a weight of $1/N$ where N is the number of correspondences at the point.

The concept of multiple correspondences at a point needs some clarification. As discussed earlier, correspondences are computed between detected edge points and a hypothesized edge line segment. While each detected edge point is associated with the closest point on the line segment, it is unlikely that these closest points will ever be exactly the same down to machine precision. Instead the concept of multiple correspondence refers to the same point at pixel resolution. If searching for correspondences perpendicular to the hypothesis (Figure 4.2 left) then N is the number of detected edges found along each search line. Since an image aligned search pattern is used instead (Figure 4.2 right), the multiplicity must be computed in a different way. The projected closest points are collected into one pixel wide bins along the hypothesized line segment. The bin occupancy determines N .

Proximity to image boundaries

When fitting a vehicle model to sequential frames of video (See Chapter 6.), edges will appear or disappear when they cross an image boundary. This rapid change in edge visibility can cause a large change in the optimization equations. The same applies to hypothesized edges that cross a boundary during optimization in a single image. Hypothesized edges could be maintained outside the image boundaries, but they are not since the appropriate corresponding edges are not likely to be available. Both cases can be solved by gradually reducing the weight of detected edges as they approach the image boundaries. Any detected edge more than T_b pixels away from all boundaries is assigned a weight of one. Edges near a boundary are assigned a weight that varies linearly from zero to one with the distance to the boundary. $T_b = 40$ is used for this threshold (as in [25]).

4.3.4 Multi-scale optimization

The complete system of robust and regularized equations for fitting the vehicle model were developed in the previous subsections culminating in Equation (4.39). This system of equations is used in IRLS. IRLS alternates between solving the system and reestimating the Jacobian, correspondences, and weights. In Section 4.3.3 it was made clear that the selection of correspondences and computation of weights are dependent on σ_r , the scale of the residuals. The scale of the residuals is determined by the standard deviation of the residuals of the true (and unknown) correspondences. Not only is σ_r unknown, it also varies as the model parameters change. When the model is perfectly fit to the image it is assumed that the scale reduces to a minimum value of $\sigma_r = 1$. The selection and adjustment of σ_r is critical to achieving correct convergence from a wide variety of initial parameters. Figure 4.5 shows an example of the magnitude of the residual, $\mathbf{e}^\top \mathbf{W}^2 \mathbf{e} + \lambda^2 \mathbf{p}^\top \Sigma_{\mathbf{p}}^{-1} \mathbf{p}$, plotted as a function of the first two shape principal components at four different residual scales. At large σ_r , the error function is smooth and there are few minima, each with a wide basin of attraction. However, the error function in the neighborhood of a minimum is somewhat flat, and the optimal parameters are not precisely located. At small σ_r , there are many local minima but each is more precisely located.

The choice σ_r is very closely related to the notion of scale-space [71] commonly used in computer vision. Scale-space methods frequently create an image pyramid by smoothing and down-sampling the image recursively. Each level of the pyramid represents one scale. Typically, features are detected at each level of the pyramid [73, 78]. Fitting algorithms start with coarse scale images. The fit to the coarse scale image initializes the next finer level, and fitting proceeds through scale space until reaching the original fine scale image. Starting with the coarse scale allows for convergence over a wide region in parameter space. Finishing with the fine scale allows the converged result to be as accurate as possible.

Using the image pyramid approach for vehicle fitting would involve setting $\sigma_r = 1$ at each level of the pyramid; the scale change is built into the pyramid itself. The alternative approach used in this thesis is to fix the scale of the image and adapt σ_r . This approach is possible because the detected edge features of interest are assumed to be sharp intensity discontinuities that manifest over a wide range of scales. The pyramid approach offers the computational advantage of a small search radius over all scales. However, additional computation is required to compute the image pyramid and detect edges at each level. It is unclear whether the computational savings outweigh the pyramid construction cost. A detailed comparison of these two approaches is left for future work.

The multi-scale fitting algorithm used in this thesis takes an approach that is closely related to the method of graduated non-convexity [8]. First, initialize σ_r to an appropriately large scale (discussed below). Keeping σ_r fixed, run several iterations of IRLS while monitoring the magnitude of the residuals, $\mathbf{e}^\top \mathbf{W}^2 \mathbf{e} + \lambda^2 \mathbf{p}^\top \Sigma_{\mathbf{p}}^{-1} \mathbf{p}$. Continue to iterate as long as the magnitude of the residuals continues to decrease. When the magnitude of the residuals fails to decrease, revert to the previous minimal solution, decrease σ_r , and try again. σ_r is decreased by dividing by a constant greater than one. This constant defines the sampling rate in scale-space. The value of $\sqrt{2}$ provides sufficient resolution in scale for this algorithm. When σ_r is reduced to less than one, the IRLS iterations stop.

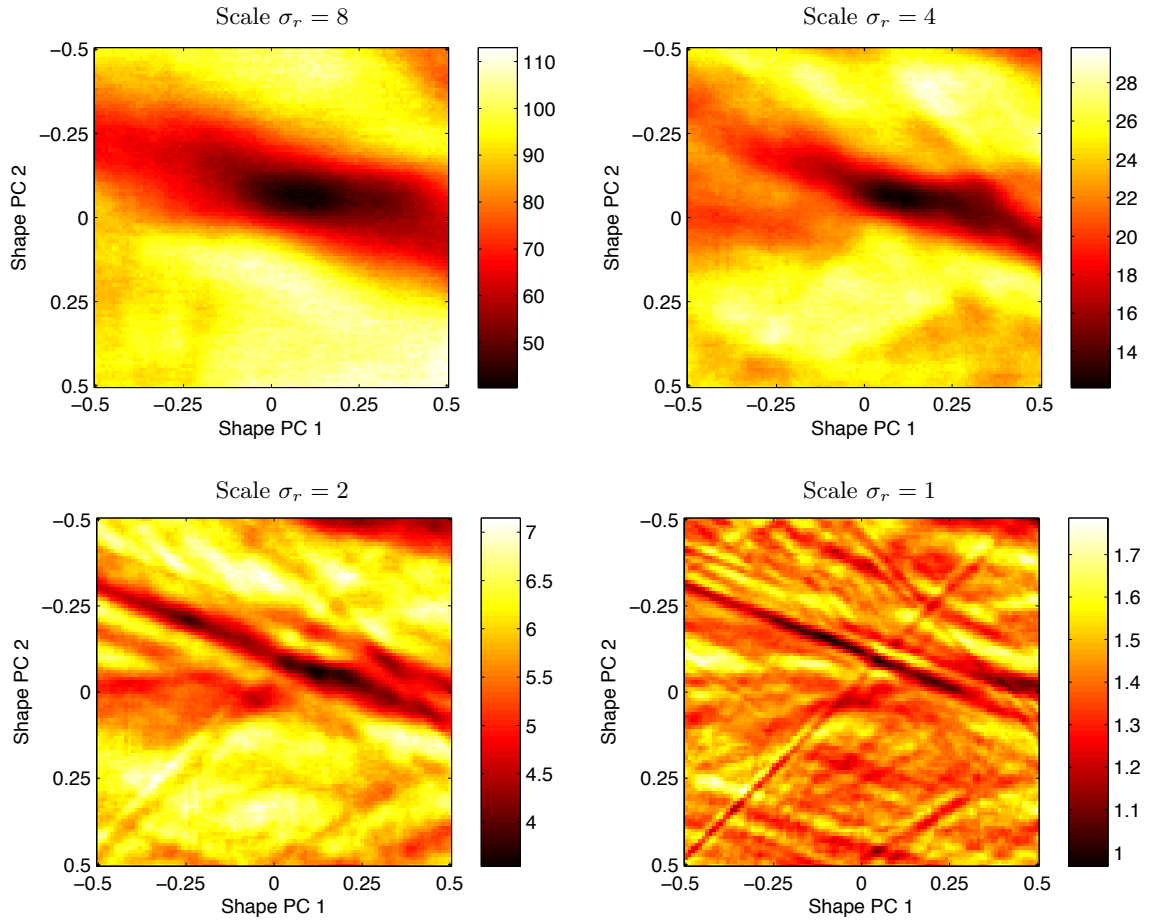


Figure 4.5: Example error surface at four different residual scales over the first two PCA shape parameters (normalized by standard deviation).

The optimization procedure is summarized in Algorithm 4.1.

Algorithm 4.1 Optimize shape and pose parameters

Require: $\mathbf{p}_0, kmax, \lambda, \Sigma_P$

$k \leftarrow 0$

initialize σ_r

$lastres \leftarrow \infty$

while $\sigma_r \geq 1$ and $k < kmax$ **do**

 hypothesize edges at $\mathbf{p}_k$

 find correspondences within $4\sigma_r$ pixels

$\mathbf{J} \leftarrow$ Jacobian computed at $\mathbf{p}_k$

$\mathbf{e} \leftarrow$ signed perpendicular correspondence errors at $\mathbf{p}_k$

$\mathbf{W} \leftarrow$ diagonal matrix of correspondence weights at $\mathbf{p}_k$ with scale σ_r

$\mathbf{W} \leftarrow \mathbf{W}/\|\mathbf{W}\|$

if $\mathbf{e}^\top \mathbf{W}^2 \mathbf{e} + \lambda^2 \mathbf{p}^\top \Sigma_P^{-1} \mathbf{p} \geq lastres$ **then**

$\sigma_r \leftarrow \sigma_r / \sqrt{2}$ {Move to a finer scale}

$lastres \leftarrow \infty$ {Reset: residuals not comparable over scale change}

$k \leftarrow k - 1$

$\mathbf{p}_k \leftarrow \mathbf{p}_{k+1} - \delta \mathbf{p}$ {Revert to the previous parameters with smaller errors}

else

 Solve $(\mathbf{J}^\top \mathbf{W}^2 \mathbf{J} + \lambda^2 \Sigma_P^{-1}) \delta \mathbf{p} = \mathbf{J}^\top \mathbf{W}^2 \mathbf{e} - \lambda^2 \Sigma_P^{-1} \mathbf{p}_k$ for $\delta \mathbf{p}$

$\mathbf{p}_{k+1} \leftarrow \mathbf{p}_k + \delta \mathbf{p}$

$k \leftarrow k + 1$

$lastres \leftarrow \mathbf{e}^\top \mathbf{W}^2 \mathbf{e} + \lambda^2 \mathbf{p}^\top \Sigma_P^{-1} \mathbf{p}$

end if

end while

Notice that Algorithm 4.1 has a very similar form to Levenberg-Marquardt optimization. (as described in Section 4.3.2). When a Gauss-Newton iteration fails to reduce the error term, that iteration is rejected, and a scale parameter is adjusted. Both optimization techniques aim to overcome trouble spots for the Gauss-Newton iteration. Figure 4.5 shows that, at $\sigma_r = 8$, the error function is very smooth and has a wide, paraboloid-like shape near the minimum. These are ideal conditions for Gauss-Newton. As σ_r decreases, the error surface becomes more noisy but still remains similar to a paraboloid in a smaller neighborhood around the minimum. Thus, the Gauss-Newton iterations are successful at all scales if properly initialized by the coarser scales. Levenberg-Marquardt is best suited for traversing long narrow trenches like those apparent at $\sigma_r = 1$ in Figure 4.5. However, such trenches are not traversed over long distances in the multi-scale algorithm. The previous scale places the initialization near the bottom to begin with. Nonetheless, it is perfectly reasonable to combine the scale space approach with Levenberg-Marquardt optimization.

The combination of Levenberg-Marquardt and scale space optimization is tested experimentally. Levenberg-Marquardt is applied at each scale instead of Gauss-Newton. As expected, the Levenberg-Marquardt parameter, α , quickly shrinks and all nearly-Gauss-Newton iterations succeed until reaching the minimum. At the first failed iteration, the gradient is almost zero. α quickly grows again until the iteration is dominated by the gradient direction. In most cases, α , can grow to a magnitude of 10^{10} and still not produce a better solution. Occasionally, a slightly better solution is found when α

is large. However, if any additional error reduction occurs, it is usually less than 1% of the reduction provided by the original Gauss-Newton iterations. Applying Levenberg-Marquardt never produces a worse solution, but the solution is either identical or very marginally better. While this sounds appealing, it is probably not worth the run-time cost in most cases. The Gauss-Newton iterations typically converge in less than eight iterations per scale. However, Levenberg-Marquardt requires many times that number and, more often than not, reaches the same result. Most of the extra iterations are spent searching for an appropriately small step that may or may not lead to a slight improvement in the end. In this case, the large increase in computation does not justify the slight improvement in results, so Levenberg-Marquardt is not used.

The initial σ_r is dependent on uncertainty in the initial vehicle shape and pose. It is not appropriate to arbitrarily pick a fixed initial σ_r because the scale may vary greatly with changes in position. The same vehicle shape at different positions could have drastically different image scales because of perspective projection. A more principled solution is to start with an initial estimate of error in the parameter space and then use the Jacobian to estimate the error in the image space. It may be desirable to estimate the scale without computing correspondences, so the image Jacobians of the control points, $\mathbf{J}_i^2 \mathbf{J}_i^3$ (See (4.11).), are used in place of the error Jacobians, $\mathbf{J}_i$. For the i -th control point, the j -th column of $2 \times n'$ matrix $\mathbf{J}_i^2 \mathbf{J}_i^3$ is a linear approximation to the vector describing the image motion of point i with a unit change in parameter j . Assuming an initial uncertainty of σ_j in the j -th parameter, one can find the magnitude of uncertainty in the image at the i -th point as the magnitude of the scaled vector, $\|\sigma_j [\mathbf{J}_i^2 \mathbf{J}_i^3]_j\|$, where $[\cdot]_j$ indicates the j -th column. The initial value of σ_r is then computed as an average of this uncertainty over all points and all parameters,

$$\sigma_r = \frac{1}{M} \sum_{i=1}^M \frac{1}{n'} \sum_{j=1}^{n'} \|\sigma_j [\mathbf{J}_i^2 \mathbf{J}_i^3]_j\|. \quad (4.42)$$

The initial uncertainties in the shape parameters are closely related to the shape standard deviations estimated in PCA, especially if starting with the mean shape. The uncertainties in pose parameters will depend on the pose initialization procedure.

The optimization algorithm is fairly robust to changing the initial value of σ_r as long as it is sufficiently large. Relating σ_r to the parameter space is more important when extending Algorithm 4.1 to multiple view estimation.

4.3.5 Multi-view optimization

The optimization algorithm described so far estimates the shape and pose of a vehicle from a single image. This is a very challenging problem in real imagery. Environments where vehicles are found are often highly cluttered with texture. Texture in the environment produces a large number of spurious edges both in the background and on the vehicle surface by specular reflection. However, when viewed from another viewpoint, most background texture changes position, and the apparent texture on the vehicle surface due to reflection also moves. The consistent edges over multiple views arise primarily from vehicle structure. Multiple views also help to constrain the 3-d geometry, which

is ambiguous from one view. Optimization from one view must rely heavily on the prior model. Thus, if multiple views are available it is beneficial to include them in a joint optimization. Conveniently, multi-view optimization only requires a minor modification to the single view algorithm.

The multi-view configuration considered here assumes that two or more cameras have been calibrated with respect to a common world coordinate system. Hence, the number of parameters is still the same as before. The n' parameters include n shape parameters and either 6 or 3 pose parameters expressed relative to the world coordinates. Assume the number of views is L . Each view has its own camera matrix $\mathbf{P}_l$ such that 3-d point $[\mathbf{x}^\top 1]^\top$ projects to corresponding point $\mathbf{u}_l = \mathbf{P}_l[\mathbf{x}^\top 1]^\top$ for each view $l \in L$. For any configuration of parameters one can now hypothesize edges in all L views. Likewise, one can compute $\mathbf{J}$, $\mathbf{e}$, and $\mathbf{W}$ for hypotheses in each view. Once computed, it does not matter which view generated the linear equations. All of the equations can be stacked into a single linear system. That is, all Jacobian rows computed from all correspondences in all views are stacked into a single matrix $\mathbf{J}$. The same is done for the signed errors in vector $\mathbf{e}$ and weights on the diagonal of $\mathbf{W}$. Since the normal equations are computed directly in practice, this simply means that $\mathbf{J}^\top \mathbf{W}^2 \mathbf{J}$ and $\mathbf{J}^\top \mathbf{W}^2 \mathbf{e}$ are now computed by summation over data in multiple views. No additional memory is required to store the linear system, and the computation to solve the linear system remains the same.

One significant difference between the data observed in multiple views is scale. Consider two identical perspective cameras observing a vehicle, one from a distance D and the other from a distance of $2D$. At twice the distance, the vehicle in the second image appears at half the scale. Changes in parameters cause hypothesized edges to move twice as far in the first image compared to the second. Clearly error measures from different views should not be treated the same. This issue is resolved by associating a different residual scale, σ_r , with each view. The initial σ_r estimation in Equation (4.42) is computed separately for each view using the same set of initial parameter space uncertainties. One would expect the estimated σ_r in the first view of the above example to be roughly twice that of the second view. Each view then maintains its relative scale during the optimization. When moving to a finer scale, all σ_r are updated according to $\sigma_r \leftarrow \max(1, \sigma_r/\sqrt{2})$. The IRLS termination condition is also modified such that iteration continues until the σ_r for all views reaches 1 or until reaching the maximum number of iterations allowed. Figure 4.6 illustrates the multi-view, multi-scale fitting process by showing an example of intermediate results after various numbers of iterations. Many more fitting experiments are provided in Chapter 7.

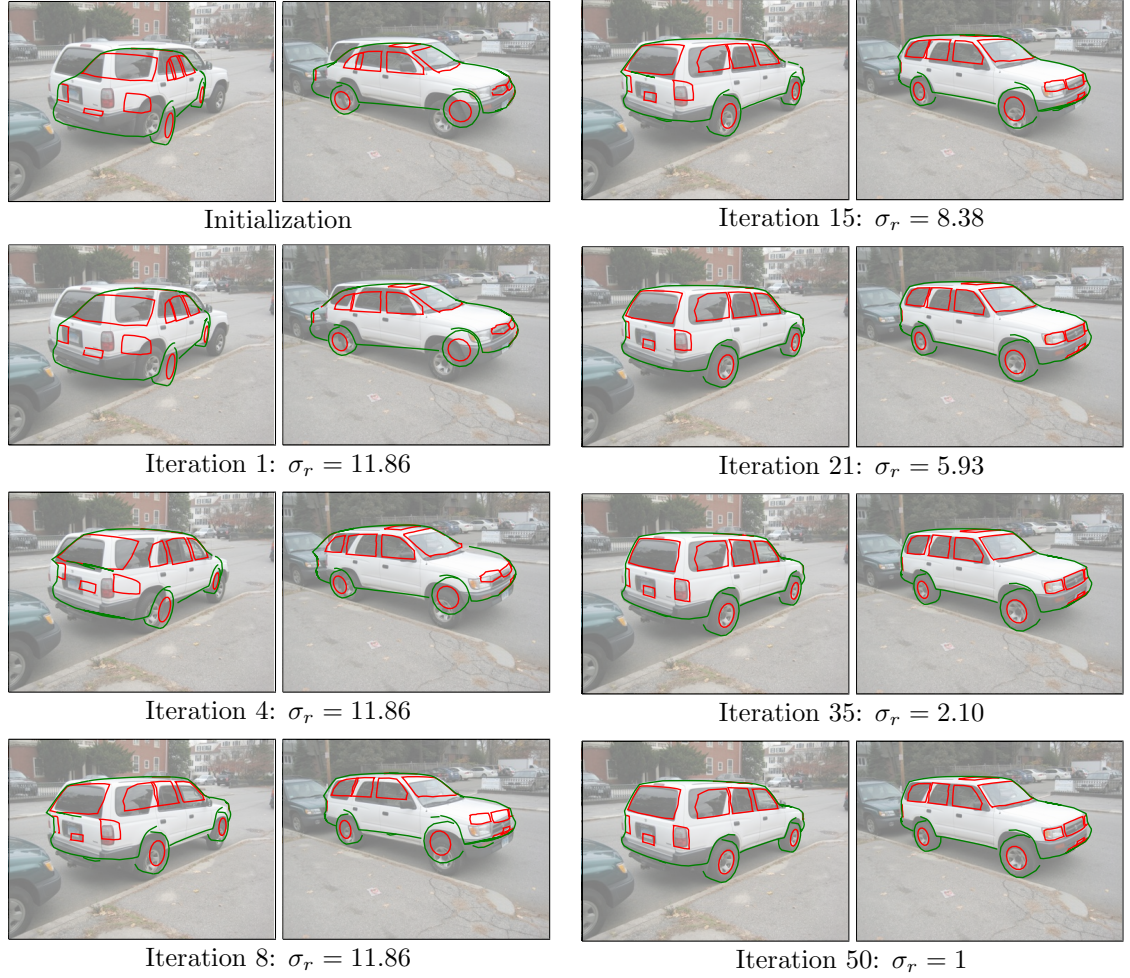


Figure 4.6: Multi-scale, multi-view fitting results after various numbers of iterations. The resulting model is overlaid on 2 of 7 images from a multi-view dataset (See Chapter 7). The initial shape is the mean over the training data, and the initial pose is shown in the upper left. The model is roughly aligned after the first 8 iterations at the initial residual scale of $\sigma_r = 11.86$. Additional iterations refine the alignment at finer scales until convergence at scale 1.

Chapter 5

Learning the Model

In Section 3.5.1 an assumption was made that the template mesh and parts were deformed to match the shape of each vehicle in a database of N vehicles. That assumption allowed PCA to form a reduced parameterization of the shape space and a prior distribution on vehicle shapes. In Chapter 3, the question of how to obtain these training deformations was left open. This chapter returns to that question to fill in the critical missing piece.

Ideally one would like to automatically learn the training deformations from data. An obvious source of training data is images of vehicles. One could try to automatically fit the template model to a collection of images with multiple views of each vehicle. Unfortunately, that is not easy. In fact, fitting the template mesh to images of vehicles is exactly the problem that learning a reduced parameterization and prior distribution is intended to help solve. Without the constraints of the prior model, automated learning from images is intractable. Without the learned training deformations, there is no data to compute a prior model. It is an example of the classic *chicken-or-egg* problem, each part is needed to produce the other.

Alternatively, one could align the template mesh to images manually or semi-automatically. This is the approach taken by Ferryman *et al.* [30] to learn the PCA training data for their model. User-guided learning from images is feasible for a simple vehicle model like that of Ferryman *et al.* However, when the vehicle mesh is subdivided to contain thousands of vertices, it is no longer feasible to have a user position them by hand. In fact, it may not even be possible for a human to do this task accurately. There are too many ambiguities caused the projection from 3-d to 2-d. Not every vertex on the mesh is associated with an identifiable feature that can be matched across multiple views. This makes determining the depth of these points difficult, even for a human operator working with multiple images. It might be feasible to use such user-guided techniques to learn new vehicle shapes once an initial prior model is available for guidance, but some other data source is needed to bootstrap the learning process.

Instead of trying to learn 3-d shape from ambiguous 2-d data, it is much easier to learn directly from 3-d data. This is precisely the solution used to produce deformable models of the human face and body (as in [9] and [2], for example). A template model is deformed to match a cloud of 3-d

points measured on the face or body surface. Often a small amount of user guidance is needed to enforce correspondence between the template and data. To obtain 3-d surface data of human subjects, a 3-d scanner system is used. There are several commercially available 3-d scanners. The technology is often very similar to the laser range finders discussed in Section 1.2.3, but there are other scanning technologies available as well. Most commercial 3-d scanning systems are built for indoor use under controlled lighting conditions and for objects of human-size or smaller. There are outdoor 3-d scanners as well, but they tend to capture data from a single direction. For example, lidar scanners mounted over roadways or on aerial platforms tend to sample many more points on the top of vehicles than on the sides. In contrast, human body scanners can usually capture data points densely on all sides. It is difficult to come by 3-d scans of passenger vehicles with dense sampling on all sides. Luckily, there is an easier way.

Unlike the human body, vehicles are manufactured to precise specifications. While the dimensions of natural objects are unknown and must be measured to produce a 3-d model, the dimensions of vehicles are specified by the designer and are known even before the vehicle is manufactured. Modern vehicles are designed with Computer Aided Design (CAD) software, so a 3-d model is available from the start. These engineering design models are not made available to the public, but more and more commonly a simpler variation is. With advances in computer graphics, it has become very common to generate advertisements for vehicles and movie special effects using photorealistic rendering. A detailed 3-d vehicle CAD model is designed just for that purpose. The same models also have uses in video games where graphics engines are now advanced enough to render detailed models in real time. As a result of the applicability of 3-d CAD models to the entertainment and marketing industries, many vehicle CAD models are easily obtained on the internet both for free and for sale. The Defense Advanced Research Projects Agency (DARPA) has even collected many of these vehicle CAD models for use as training data in the *Exploitation of 3-D Data* (E3D) project [22]. It is these CAD models that are used as training data to learn the template mesh deformations for the generic vehicle model. Table 5.1 lists the 79 E3D CAD models of passenger vehicles used as training data in this thesis. Each model is also pictured in Figure 5.1.

Table 5.1: Training CAD models from the E3D database. Each model is pictured in Figure 5.1.

#	E3D category	Make	Model	Year
1	minivan	Chevrolet	Astro	1986
2	minivan	Dodge	Caravan	1991
3	minivan	Dodge	Caravan	1996
4	minivan	Dodge	Grand Caravan	2001
5	minivan	Mazda	MPV	1992
6	minivan	Mercury	Villager	1999
7	minivan	Plymouth	Grand Voyager LE	1996

Continued on Next Page...

... Continued from Previous Page

#	E3D category	Make	Model	Year
8	pickup	Chevrolet	S10 4 × 4	1994
9	pickup	Chevrolet	S10	1998
10	pickup	Chevrolet	Silverado Twincab	2001
11	pickup	Dodge	Ram	1999
12	pickup	Nissan	Frontier	1999
13	pickup	Toyota	Tundra	2000
14	2-door sedan	Acura	Integra	2001
15	2-door sedan	Chevrolet	Cavalier	1993
16	2-door sedan	Chevrolet	Cavalier	1998
17	2-door sedan	Chrysler	Sebring	2001
18	2-door sedan	Dodge	Neon	1996
19	2-door sedan	Dodge	Stratus	2001
20	2-door sedan	Lincoln	Mark VII	1984
21	2-door sedan	Plymouth	Neon	1997
22	2-door sedan	Pontiac	Grand Am	1999
23	2-door sedan	Toyota	Tercel	1995
24	2-door sedan	Volkswagen	Beetle	1998
25	wagon	Volvo	V40	2000
26	wagon	Volvo	V70XC	2001
27	SUV	Chevrolet	S10 Blazer	1998
28	SUV	Chevrolet	Trailblazer	2001
29	SUV	Dodge	Durango	1998
30	SUV	Geo	Tracker	1993
31	SUV	GMC	Yukon	2000
32	SUV	Jeep	Cherokee (2-door)	1994
33	SUV	Jeep	Grand Cherokee	1994
34	SUV	Mercedes	G-Class	2002
35	SUV	Nissan	Xterra	1999
36	SUV	Toyota	4Runner	1999
37	4-door sedan	Acura	Integra	1994
38	4-door sedan	BMW	318	1994
39	4-door sedan	BMW	528i	1997
40	4-door sedan	BMW	540i	2001
41	4-door sedan	Buick	LeSabre	1998
42	4-door sedan	Buick	Regal	1997
43	4-door sedan	Cadillac	Catera	1999

Continued on Next Page...

... Continued from Previous Page

#	E3D category	Make	Model	Year
44	4-door sedan	Cadillac	Deville	1994
45	4-door sedan	Chrysler	300M	1998
46	4-door sedan	Chrysler	PT Crusier	2001
47	4-door sedan	Chrysler	Sebring	2001
48	4-door sedan	Dodge	Intrepid	1993
49	4-door sedan	Dodge	Intrepid	1998
50	4-door sedan	Dodge	Stratus	1998
51	4-door sedan	Dodge	Stratus	1999
52	4-door sedan	Honda	Accord	1991
53	4-door sedan	Honda	Accord	1993
54	4-door sedan	Honda	Civic	1999
55	4-door sedan	Honda	Civic	1997
56	4-door sedan	Hyundai	Elantra	2001
57	4-door sedan	Infiniti	I30	1999
58	4-door sedan	Infiniti	J30	1993
59	4-door sedan	Lexus	400GS	1998
60	4-door sedan	Lincoln	Continental	1997
61	4-door sedan	Lincoln	Towncar	1993
62	4-door sedan	Mazda	626	1999
63	4-door sedan	Mercedes	W210	1996
64	4-door sedan	Mercury	Sable LS	1996
65	4-door sedan	Mitsubishi	Carisma	1998
66	4-door sedan	Nissan	Altima	1994
67	4-door sedan	Nissan	Maxima	1994
68	4-door sedan	Nissan	Maxima GLE	1999
69	4-door sedan	Olds	Cutlass	1991
70	4-door sedan	Olds	Intrigue	1999
71	4-door sedan	Plymouth	Neon	2000
72	4-door sedan	Pontiac	Grand Prix GT	2000
73	4-door sedan	Toyota	Avalon	1995
74	4-door sedan	Toyota	Corolla	1998
75	4-door sedan	Toyota	Mark II	1997
76	4-door sedan	Volvo	S40	2000
77	4-door sedan	Volvo	S80	2000
78	4-door sedan	Volkswagen	Jetta	1991
79	4-door sedan	Volkswagen	Jetta	1998



Figure 5.1: Training CAD models, grouped by categories used in the E3D project. The numbers refer to Table 5.1.

In broad terms, the template mesh deformations are learned by fitting the template to the surface of the vehicle CAD models. Some minimal user guidance is needed to position the template in approximate correspondence with the CAD model. Then an optimization routine tightly fits the template surface to the CAD model surface in a *shrink-wrap*-like process. Fitting the template parts uses the analogous process in 2-d. First the user identifies the parts on the CAD model. The boundaries of these parts are projected to the closest points on the fitted template mesh, and then mapped into texture space. The template parts are automatically shrink-wrapped to the mapped parts in texture space. The basic idea behind learning mesh deformations is simple, but the implementation is rather complex.

5.1 CAD models as training data

Vehicle CAD models designed for rendering are represented as a polygonal meshes. Unfortunately, none of the nice properties of the template mesh are guaranteed for a CAD model. These CAD models are designed for high quality rendering, not for 3-d shape representation. A CAD model is typically comprised of many disconnected sub-meshes. Each connected component does not usually represent the boundary of a solid shape. The meshes are rarely watertight. While the connected components tend to be oriented and manifold, even these properties do not always hold.

The separate connected components may represent different vehicle parts and may be assigned different material properties for rendering. The CAD parts do not correspond directly to the deformable surface parts of the generic template described in Section 3.3. The number of connected mesh components is an arbitrary choice of the CAD model designer. The number varies widely from model to model. Some models treat the entire vehicle body as one connected component. Others have a separate component for each physical part (doors, hood, trunk, *etc.*). Fortunately, since the deformable template parts represent different surface properties, they are usually included as a subset of the CAD model connected components. Each template part corresponds to one or more CAD model connected components in general. This facts makes user-guided learning of template part shapes much easier.

The amount of detail in the CAD meshes also varies widely both between vehicle models and between connected components. Some simple CAD models contain mesh components for only the exterior surfaces of the vehicle, much like the template mesh does. More complex CAD models may have mesh components for interior structures like seats, a steering wheel, and a dashboard, as well as undercarriage components like exhaust pipes, a muffler, and axles. Some CAD models are detailed enough to have separate components for each lug nut, wiper blade, and audio system control knob. Large flat surfaces are modeled with few large polygons, and fine detailed parts are modeled with many more small polygons. As a result, the vertices of CAD model meshes are not evenly distributed across the surface, and many vertices can be found inside the vehicle’s outer surface. This means it is not possible to treat the mesh vertices as a point cloud and directly apply the point cloud fitting methods that are common with 3-d scanned data. Instead, surface fitting needs to include

the polygon faces specified in the CAD model. Figure 5.2 illustrates the variation in detail between two CAD models that differ in number of faces by almost two orders of magnitude.

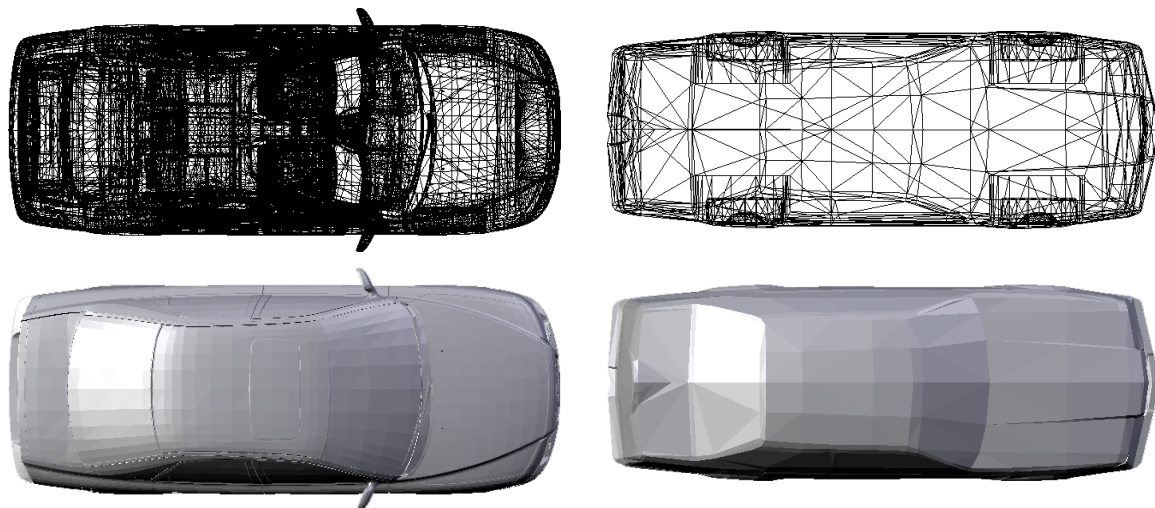


Figure 5.2: Varying level of detail between different CAD models. Top-down views of two different sedan CAD models rendered in wireframe (top) and solid shading (bottom). The left vehicle mesh models interior and exterior details with 226,604 faces. The right vehicle mesh only models the exterior with 2,750 faces.

Another complication of working with CAD models are gaps between mesh components. Since the mesh components are not connected, there will inevitably be gaps or even intersections between different components. Most gaps between components are too small to notice in a rendering of the model. A few gaps are intentionally large, for example, to represent the openings in a grille. All gaps and face intersections can introduce problems for algorithms that need to interpret the collection of components as a single closed surface.

5.2 Fitting the 3-d mesh

Shrink-wrapping the template mesh to the CAD model surface involves repositioning each template vertex to lie on the CAD model surface. Ideally, the normals of the CAD model surface and the normals of the template mesh should also agree. If the vertices of template mesh are spaced close enough to each other relative to the magnitude of CAD surface curvature, then the normals will naturally agree. This is a result of the Nyquist sampling theorem. Low curvature is a low frequency signal and requires fewer samples to accurately represent the shape. Higher curvature requires more samples to accurately represent the shape. Fitting the base template mesh (which has few vertices) directly to a CAD model (which has high frequency curvature) can lead to undesirable effects like aliasing. The solution is to apply a low-pass filter to the CAD model shape to smooth away high frequency curvature when fitting a template with a small number of vertices. The filter can pass higher frequencies as the template mesh is subdivided.

Taubin [105] pioneered the field of mesh smoothing using low-pass filtering techniques in the frequency domain. Much additional work has been done on mesh smoothing since then. Unfortunately, all of these techniques operate on a single connected mesh. They do not operate correctly on disconnected groups of mesh components that collectively represent a single surface. The jumble of disconnected polygon groups found in the CAD models are commonly called *polygon soup*. It is notoriously difficult to apply mesh manipulation algorithms to a polygon soup. The solution proposed by Shen *et al.* [93] is to re-mesh the surface with a watertight mesh that closely approximates the original mesh while filling in holes and gaps. Their method generates a 3-d scalar field from a mesh such that the original mesh faces lie in a level-set of this field. The algorithm also has a parameter that smooths the field, thereby smoothing the isosurface. Shen *et al.* use an algorithm to build a new explicit mesh from this implicit surface representation. Remeshing the implicit surface is not of interest when deforming a template mesh. One can use the implicit surface directly to guide the repositioning of the template mesh vertices. The implicit surface algorithm of Shen *et al.* is used to simultaneously smooth the CAD model surfaces while filling in holes and gaps.

It is not sufficient to map template mesh vertices to arbitrary points on the CAD isosurface. Assume, for some starting configuration of the template mesh, that each vertex is mapped to a nearby point on the CAD isosurface. This deformation of the mesh may represent the CAD model shape reasonably well, but if the initial configuration is arbitrary there will be no correspondence between features from one vehicle to the next. Correspondence is crucial when linearly interpolating between shapes, as in PCA. When a vertex in the template corresponds to a point on the front of one vehicle and on the back of another, the average position is inside both vehicles, and the average vehicle shape looks nothing at all like a vehicle. To enforce correspondence, the initial template configuration should be roughly aligned with the CAD model shape and pose such that each vertex consistently starts near a corresponding part on the vehicles. This initialization requires some user guidance. Consider a point on top of the trunk of a sedan. Where does this point correspond to on a minivan? There is no well defined answer to this question. Instead this correspondence is a choice in the model design. Whatever the choice may be, it must be enforced in a consistent way across all vehicles. Consistency is provided by manual initialization of the base template mesh. Approximate correspondence is maintained during subdivision because each new vertex is positioned relative to several existing vertices that have correct correspondence.

5.2.1 Manual initialization

Manual initialization of the base template mesh allows for some degree of control over how the model corresponds across vastly different vehicle shapes. Initialization starts by placing all CAD models into the same reference pose as the template model. In this pose, the driving direction is aligned with the X axis and the vertical direction with the Z axis. The origin is placed halfway between the front and rear axles in X , halfway between the left and right sides in Y , and on the ground plane in Z (*i.e.* the minimum Z value over all vertices is zero).

Specifying the initial position of each vertex in the base template is too much work for manual

labor. There are still a few hundred vertices in the base template with three degrees of freedom each. Additionally, many of the vertices obey symmetry constraints that apply to all vehicles. For example, all vertices are mirror symmetric about the XZ plane. Wheels and wheel wells are radially symmetric about the axles. The work involved in manually initializing the vertices should be minimized by taking these constraints into account. To that end, a set of 30 semantically meaningful, axis-aligned vehicle measurements is defined. These parameters define the initial template mesh deformation much like the 12 parameters in Koller *et al.* [59] or the 29 parameters in Ferryman *et al.* [30]. A graphical user interface (GUI) is built that assigns a slider to each parameter allowing easy adjustment of the 30 parameters. The interface is a Python plugin to the open source 3-d modeling program called Blender. The template mesh wireframe is rendered on top of the CAD wireframe and updated in real time as the parameters change. Figure 5.3 shows a screen capture of the GUI. A list of the 30 parameters and brief description of their semantic meanings is given in Table 5.2.

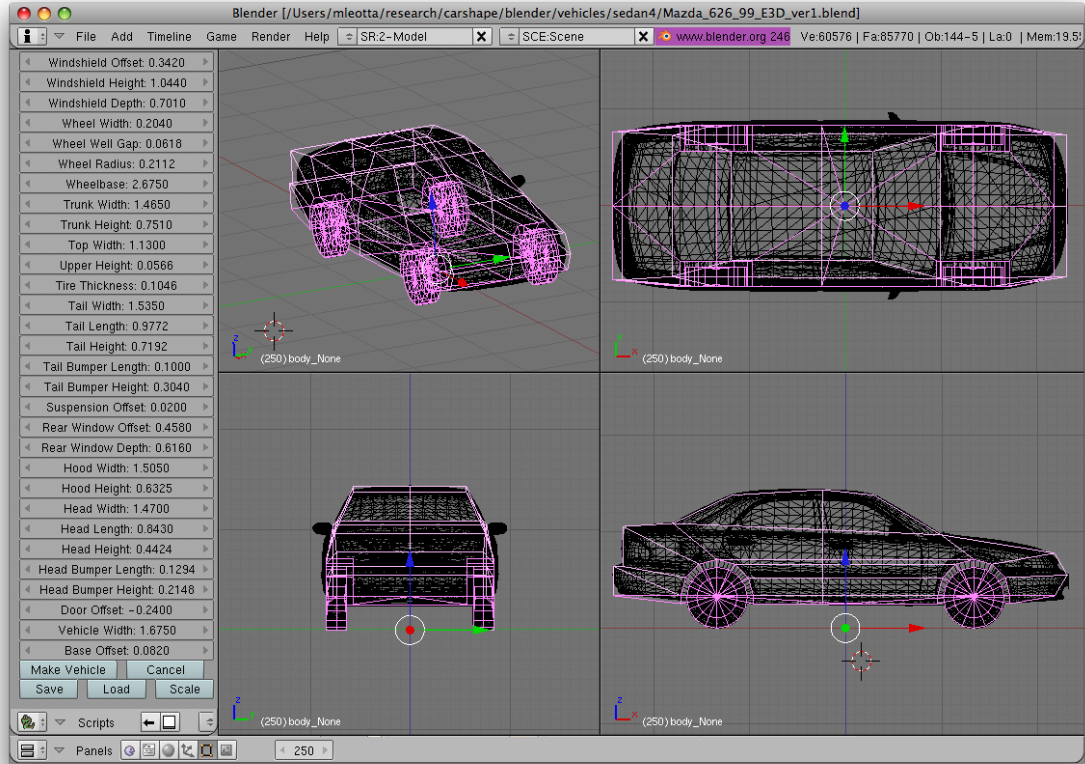


Figure 5.3: Template mesh initialization GUI. The template is rendered orthographically in pink wireframe over the black wireframe of the target CAD model. The sliders on the left vary each of the 30 semantic template mesh parameters.

The template initialization parameters are set manually for each of the CAD vehicle models in the training database. After this initialization phase, the shape fitting is automated for all subdivided mesh resolutions by shrink-wrapping the vertices to the CAD models surface. The initialization

Measurement	Axis	Description
Wheelbase	X	Distance between axles
Wheel Radius	$X \& Z$	Radius of each hubcap (<i>i.e.</i> inner radius of each tire)
Tire Thickness	$X \& Z$	Difference between inner and outer tire radii
Wheel Width	Y	Width of tire tread
Vehicle Width	Y	Overall width of vehicle body at center
Suspension Offset	Z	Vertical distance between axle centers and base of body
Wheel Well Gap	$X \& Z$	Radial distance between tires and wheel well
Head Length	X	Distance from front axle to head of vehicle
Tail Length	$-X$	Distance from rear axle to tail of vehicle
Head Height	Z	Vertical distance between body base and hood at vehicle head
Tail Height	Z	Vertical distance between body base and trunk at vehicle tail
Head Width	Y	Overall width of vehicle body at head
Tail Width	Y	Overall width of vehicle body at tail
Upper Height	Z	Vertical distance between top of windshield and center of roof
Top Width	Y	Overall width of vehicle at roof
Tail Bumper Height	Z	Vertical distance between body base and top of tail bumper
Tail Bumper Length	$-X$	Distance tail bumper protrudes from tail
Head Bumper Height	Z	Vertical distance between body base and top of head bumper
Head Bumper Length	X	Distance head bumper protrudes from head
Windshield Offset	X	Horizontal distance between front axle and bottom of windshield
Windshield Depth	X	Horizontal distance between bottom and top of windshield
Hood Height	Z	Vertical distance between body base and hood at windshield bottom
Hood Width	Y	Overall width of vehicle where hood meets windshield
Rear Window Offset	X	Horizontal distance between rear axle and bottom of rear window
Rear Window Depth	X	Horizontal distance between bottom and top of rear window
Trunk Height	Z	Vertical distance between body base and trunk at rear window bottom
Trunk Width	Y	Overall width of vehicle where trunk meets rear window
Windshield Height	Z	Vertical distance between body base and tops of windshield and rear window
Base Offset	$-Z$	Body base offset between axles and below normal base level
Door Offset	X	Distance from center to trailing edge of front door

Table 5.2: Semantic template mesh parameters for manual initialization. Each parameter is a measurement aligned with one of the coordinate axes. The exceptions are the radial wheel measurements in the XZ plane.

parameters do not cover the complete degrees of freedom in the base template model. Indeed, there is always a configuration of the same base template vertices that better describes the shape of the vehicle but lies outside the 30 parameter subspace. Thus, even the base template mesh is shrink-wrapped to the CAD model surface.

5.2.2 Constructing implicit surfaces

An implicit surface of the CAD model body is produced using the implicit moving least-squares (IMLS) method described by Shen *et al.* [93]. In their paper, Shen *et al.* describe a generalized mathematical framework for IMLS surfaces; however, they only apply a special case that uses constant basis functions for their experiments. This simple case was found to be sufficient in practice, and it also has a much more intuitive interpretation. The constant basis function case of IMLS is used to build the CAD model implicit surface. This special case is described here. Readers interested in the more general framework are referred to [93].

It is easiest to first define the IMLS surface for an oriented point cloud (*i.e.* points with normal vectors) and then extend it to meshes. Consider a cloud of 3-d points sampled from a surface. Each point has a normal vector perpendicular to the surface at that point. The goal is to construct an

implicit surface as an isosurface of a scalar field function, $f(\mathbf{x})$, such that the surface passes through all the points and the point normals agree with the surface normals. Intuitively, IMLS accomplishes this task by assigning a field function, $S_i(\mathbf{x})$, to each point, $\mathbf{p}_i$, that enforces the tangent constraints of the surface in the neighborhood of that point. It then computes a moving average over the fields of all points. At any point in space, the field $f(\mathbf{x})$ is simply a weighted average over the $S_i(\mathbf{x})$ of each point. The average is *moving* because the weighting function assigns more weight to close points and the weight falls off to zero for points that are far away. Figure 5.4 illustrates the process in the analogous 2-d case. This is somewhat similar to fitting a surface with radial basis functions, but much more local. Radial basis functions have the counterintuitive property of giving more weight to distant points.

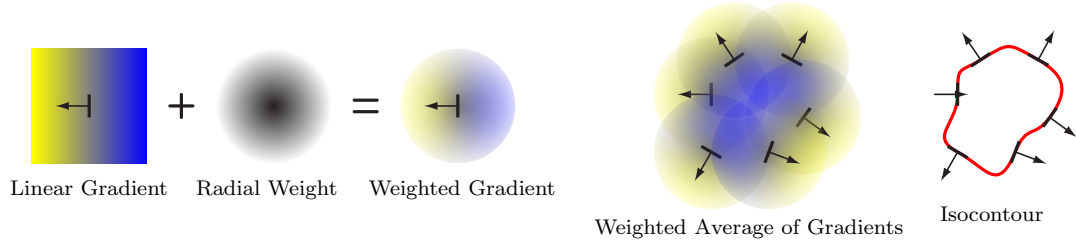


Figure 5.4: A 2-d example of IMLS on points with normals. Each point is associated with a gradient field aligned with the normal direction and a radial weight field. Both fields have infinite extent, but the weight quickly decays toward zero. A weighted average of the fields around each direct point produces the implicit surface field from which an isocontour is extracted.

The local field function, $S_i(\mathbf{x})$, assigned to each point is simply a linear gradient in the direction of the normal vector. The function is defined as

$$S_i(\mathbf{x}) = \phi_i + (\mathbf{x} - \mathbf{p}_i)^\top \mathbf{n}_i, \quad (5.1)$$

where $\mathbf{n}_i$ is the normal vector at point $\mathbf{p}_i$ and ϕ_i is the value the field takes at that point. $S_i(\mathbf{x})$ is the signed distance from $\mathbf{p}_i$ to $\mathbf{x}$ along the direction $\mathbf{n}_i$ plus a constant ϕ_i . In the simplest case, ϕ_i is set to zero for all i . Then the resulting field $f(\mathbf{x})$ is negative inside the surface and positive outside with the desired isosurface defined at $f(\mathbf{x}) = 0$. There is, however, a reason to use nonzero ϕ_i , so this term will not be dropped. The reason will become apparent later.

The weighting function used in [93] and applied in this thesis is

$$w(r) = \frac{1}{r^2 + \epsilon^2}. \quad (5.2)$$

It is radially symmetric about each point $\mathbf{p}_i$ where r is the radius. Therefore, the weight as a function of $\mathbf{x}$ and center point $\mathbf{p}$ is

$$w(\mathbf{x}, \mathbf{p}) = \frac{1}{\|\mathbf{x} - \mathbf{p}\|^2 + \epsilon^2}. \quad (5.3)$$

The parameter ϵ is used to control the smoothness of the field and resulting isosurface. Shen *et al.* note that ϵ has the same units as distance, and the isosurface approximation with this weighting function tends to smooth away any surface structures with length less than ϵ .

In the general framework for IMLS, the field function is evaluated as

$$f(\mathbf{x}) = \mathbf{b}(\mathbf{x})^\top \mathbf{c}, \quad (5.4)$$

where $\mathbf{b}(\mathbf{x})$ is the vector of basis functions and $\mathbf{c}$ is a vector of unknown coefficients. These coefficients are computed using the least-squares solution to

$$\begin{bmatrix} w(\mathbf{x}, \mathbf{p}_1) & & \\ & \ddots & \\ & & w(\mathbf{x}, \mathbf{p}_N) \end{bmatrix} \begin{bmatrix} \mathbf{b}(\mathbf{p}_1) \\ \vdots \\ \mathbf{b}(\mathbf{p}_N) \end{bmatrix} \mathbf{c} = \begin{bmatrix} w(\mathbf{x}, \mathbf{p}_1) & & \\ & \ddots & \\ & & w(\mathbf{x}, \mathbf{p}_N) \end{bmatrix} \begin{bmatrix} S_1(\mathbf{x}) \\ \vdots \\ S_N(\mathbf{x}) \end{bmatrix} \quad (5.5)$$

for a set of N points. In the special case of constant basis functions, $\mathbf{b}(\mathbf{x}) = [1]$, the solution reduces to

$$f(\mathbf{x}) = \frac{\sum_{i=1}^N w^2(\mathbf{x}, \mathbf{p}_i) S_i(\mathbf{x})}{\sum_{i=1}^N w^2(\mathbf{x}, \mathbf{p}_i)} \quad (5.6)$$

This equation agrees with the intuitive interpretation that $f(\mathbf{x})$ is a weighted average of the constraint fields, $S_i(\mathbf{x})$. However, the least-squares solution causes the weights to be the squares of the function given in Equation (5.3). Division by the sum of all weights acts as a normalization. Notice that as $\epsilon \rightarrow 0$ the weight function evaluated at its center, $w(0)$, blows up. This singularity in the weight is removable in $f(\mathbf{x})$ and in the limit $f(\mathbf{p}_i) \rightarrow \phi_i$. When $\epsilon = 0$ and all $\phi_i = 0$ the isosurface at zero passes exactly through all data points. For numerical stability it is best use a very small $\epsilon > 0$ to avoid singular points.

So far this algorithm generates an isosurface for a point cloud. However the real data of interest is a polygonal mesh. A naïve extension from points to a mesh is to simply scatter a large number of points across the mesh faces and apply the point cloud method. Shen *et al.* show that scattering points on the mesh can lead to artifacts in the form of high frequency surface oscillations. The appropriate solution is to take the limit as the number of scattered points goes to infinity. That is, integrate the equations over each polygon and then sum over all polygons. Thus, the version of Equation (5.6) adapted for a mesh is

$$f(\mathbf{x}) = \frac{\sum_{i=1}^M A_i}{\sum_{i=1}^M a_i} \quad A_i = \int_{\Omega_i} w^2(\mathbf{x}, \mathbf{p}) S(\mathbf{x}, \mathbf{p}) d\mathbf{p} \quad a_i = \int_{\Omega_i} w^2(\mathbf{x}, \mathbf{p}) d\mathbf{p} \quad (5.7)$$

where M is the number of polygons and Ω_i represents the area of polygon i . In the continuous case, the constraint field $S(\mathbf{x}, \mathbf{p})$ becomes a function of $\mathbf{p}$ because the value ϕ can vary across the polygon. In practice, a ϕ is assigned to each vertex and linearly interpolated across each polygon, so it may indeed vary. The other terms of $S(\mathbf{x}, \mathbf{p})$ are fixed because each face is planar. There are no known closed form solutions to the area integrals in Equation (5.7). Shen *et al.* do, however, provided closed form solutions of these integrals over a line. Unfortunately, the derivations reported in [93] seem to be incorrect. The correct derivations of the closed form line integrals are provided in Appendix A. The area integral is computed by a special form of numerical quadrature that takes into account the singularity in the weighting function. Polygons are split into triangles and then divided into smaller triangles if necessary such that the closest point on each triangle to $\mathbf{x}$ is at one of

the triangle’s vertices. The outer integration is computed numerically along one axis in barycentric coordinates using special quadrature points. The inner integration is computed analytically along the barycentric iso-lines of the other axis. Refer to [93] for quadrature details.

Increasing the value of ϵ has the desired effect of smoothing away structures of length less than ϵ , but it also has the undesirable effect of ballooning the surface approximately an ϵ distance away from the mesh. Shen *et al.* counteract this effect by computing the isosurface at $f(\mathbf{x}) = -\epsilon$ instead of zero. This correction helps the overall size, but the shape is no longer as accurate. Some parts of the mesh may lie significantly outside of this new isosurface. The solution to this problem is to iteratively adjust the ϕ values at these poorly represented vertices and recompute the isosurface until all vertices lie within the isosurface. Figure 5.5 is a result from Shen *et al.* showing the isosurface of a loader construction vehicle after iterative adjustment for three different values of ϵ .

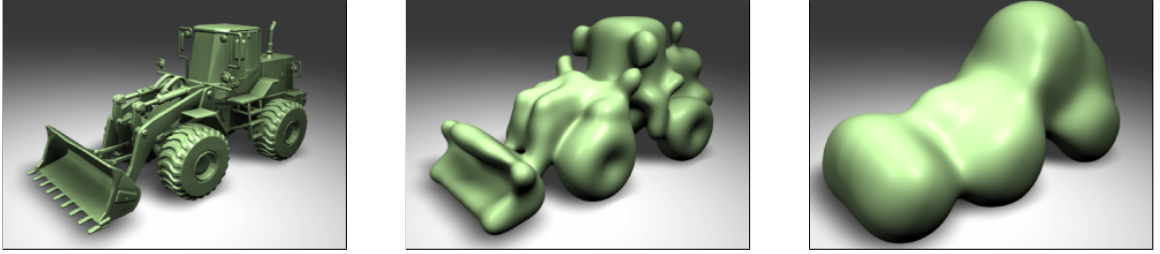


Figure 5.5: IMLS (Implicit Moving Least-Squares) Surface of a loader vehicle with increasing values of weighting function parameter ϵ . Images adapted from Figure 1 of [93].

A few preprocessing steps are needed to make this algorithm work on the CAD vehicle models. Only the exterior vehicle surface is of interest, so as many interior polygons as possible should be removed to prevent their influence on the exterior surface and to reduce computation. Since the CAD models are a polygon soup, there is no topological notion of interior vs exterior. Instead, interior polygons are defined to be those that are occluded from almost all viewpoints. The CAD model is rendered at high resolution from many viewing directions sampled across the viewing sphere. For each rendered image, each pixel is associated with an observation of one mesh polygon. Polygons that receive no observations from any of the sampled viewing directions are removed. It is also beneficial to make the orientation of all polygons as consistent as possible. When a polygon is observed only from the back in all sample viewing directions, the orientation of that polygon is reversed.

Since the resulting isosurface is used for shrink wrapping the template mesh to the vehicle body, the input CAD model mesh should contain only the vehicle body. The mesh components representing the wheels and other protruding structures like side mirrors, luggage racks, and wiper blades are manually identified and removed. Figure 5.6 shows an example CAD model before and after removing protruding structures. The wheels of the template mesh are not deformed beyond the three manual initialization parameters (wheel radius, tire thickness, and wheel width) since wheels do not tend to exhibit much additional shape deformation. Other protruding parts including side mirrors and luggage racks are relatively small and complex in shape. They are too complex and not

important enough to the overall shape to expend effort in modeling at this time. These parts are ignored for now, but could be modeled at a later time if necessary.

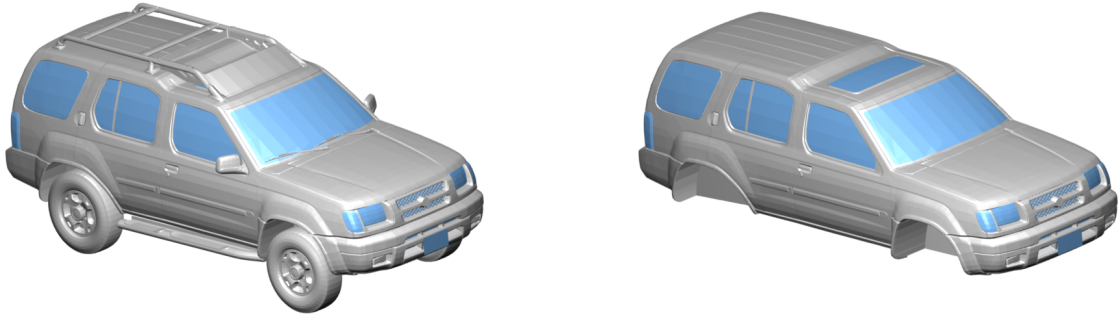


Figure 5.6: Removal of protruding parts leaving only the mesh body. The original mesh is on the left. The IMLS surface is estimated from only the body sub-mesh on the right.

5.2.3 Shrink-wrapping to implicit surfaces

After manually initializing the template mesh body and building the IMLS implicit surface of the CAD model body, the template is deformed to agree with the IMLS surface. This optimization of the template shape is reminiscent of the shrink-wrap process in commercial packaging in which a thin plastic film is heated and formed to exactly fit the shape of an object. The template mesh takes the place of the plastic film and the isosurface of the CAD model takes the place of the target object. Each vertex of the template mesh is moved to lie on the isosurface. The direction of this motion is normal to the template mesh surface. Normal vectors at each vertex of the template mesh are computed by averaging the normals of each face adjacent to the vertex. The intersection point with the isosurface is found along the vertex normal vector, and the vertex is moved to this intersection point. Figure 5.7 illustrates the process in the analogous 2-d case.

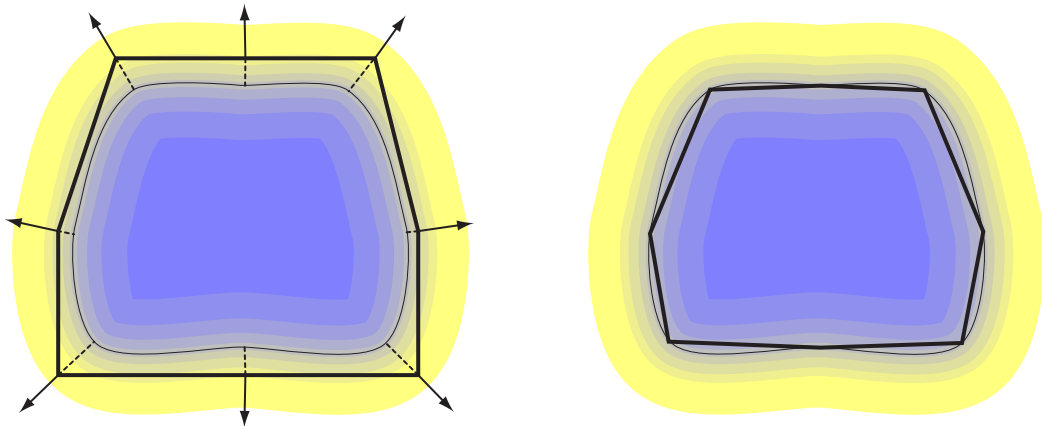


Figure 5.7: Shrink-wrap in 2-d. Template vertices follow the field gradient along the normal direction until they meet the isosurface.

Finding the intersection with the implicit surface along a ray requires numerical optimization. Newton's method is used to follow the gradient of the field function $f(\mathbf{x})$ projected onto the normal direction. The same techniques are used in computer graphics to ray trace implicit surfaces [10]. Newton's method requires the computation of $\nabla f(\mathbf{x})$. Computing the gradient of an IMLS field goes beyond the work described in [93]. One could easily compute $\nabla f(\mathbf{x})$ by finite difference, but this calculation requires three additional evaluations of $f(\mathbf{x})$. Each evaluation of $f(\mathbf{x})$ involves integration over many faces and is costly. It is more efficient to compute the gradient analytically at the same time as evaluating $f(\mathbf{x})$. The analytical solution requires a few new integration terms in addition to those used in evaluating $f(\mathbf{x})$. The derivation is provided in Appendix A.

The base template mesh body is deformed to the CAD body isosurface using a smoothing parameter of $\epsilon = 0.1$ meters. Effectively all surface detail smaller than 0.1 meters is smoothed away. After fitting the base template mesh, the mesh is subdivided and fit again. Each time the mesh is subdivided the vertex sampling is doubled in both directions of the surface tangent subspace. Accordingly, the value of ϵ is halved with each subdivision. More detail in the isosurface becomes available as more vertices are produced to model it. Figure 5.8 shows the initialization and three resolutions of fitting for the vehicle body in Figure 5.6.

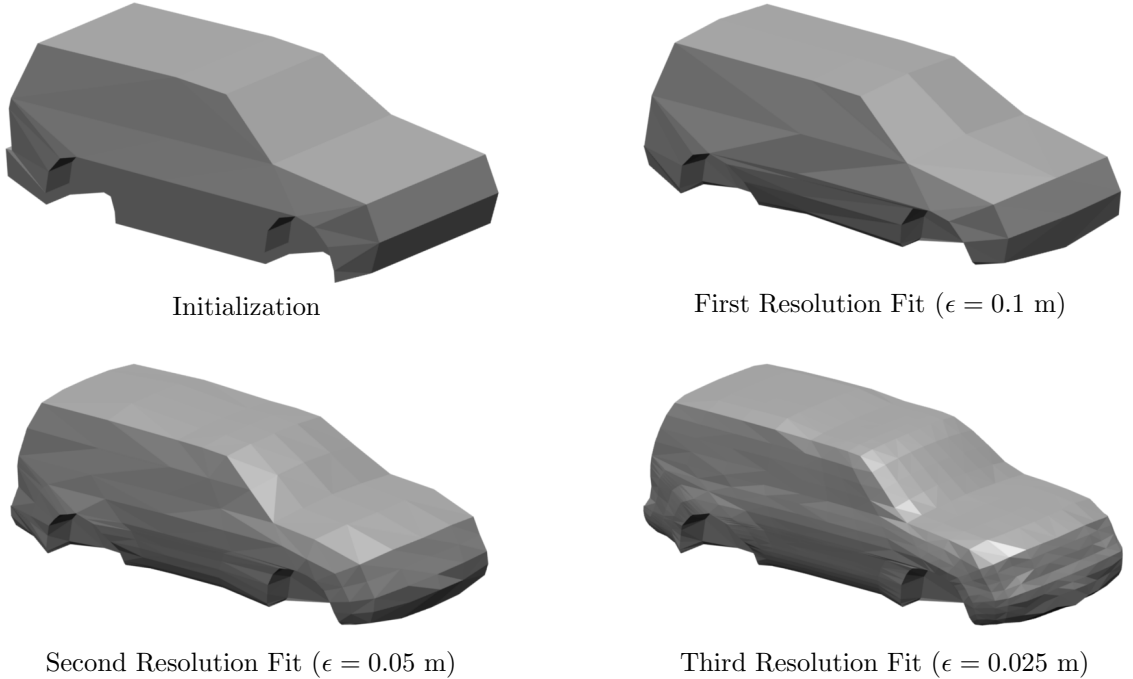


Figure 5.8: Multi-resolution fitting of the vehicle body.

5.3 Fitting the 2-d parts

Once the 3-d shape of a vehicle has been learned, the 2-d deformable template parts are learned on that surface. Luckily, each template part is typically represented as one or more connected components of the CAD model mesh. Taking advantage of this fact, these components are identified on the CAD models (shown in blue in Figure 5.9) and their 3-d mesh boundaries are extracted. All boundary points are mapped into texture coordinates by a two step process as shown in Figure 5.9. First, a point on the CAD model is projected to the closest point on the shrink-wrapped template mesh. Second, the point is mapped into texture space using barycentric coordinates.

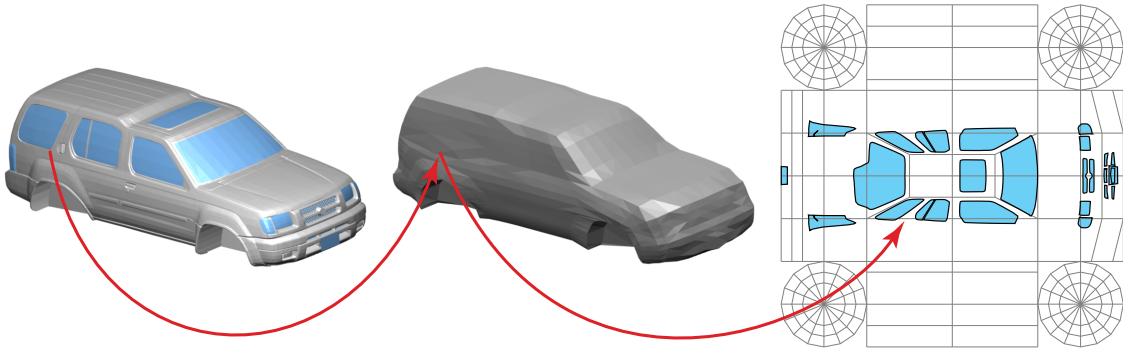


Figure 5.9: Mapping parts from the CAD model to texture space.

The next step is to fit the template mesh parts to the mapped CAD parts in the texture space. This problem is the 2-d analog of fitting the template mesh in 3-d. Each of the template parts listed in Figure 3.10 is independently deformed to match the shape of the mapped CAD part (if the part exists for the given vehicle). Unlike the 3-d template, the 2-d template parts do not require any manual shape initialization. An automatic initialization method is used instead. The exception is the hubcaps which are attached to the wheels and not the body. The hubcaps happen to be already represented by vertices in the mesh. Hubcap parts are manually added to the texture space to coincide with the mesh vertices. They are added to fixed texture space coordinates for all vehicles and only deform as a result of 3-d mesh deformation.

The implicit surface technique used in 3-d is replaced by a much simpler and more efficient algorithm that maps the template parts directly to the target polygon data in 2-d. Similar techniques were tried unsuccessfully in the 3-d case. The reason they work for the 2-d shapes and not the 3-d shape is that the 2-d shapes are much better behaved. The 2-d parts are generated from the boundaries of regions, so they are guaranteed to be closed polygons. The 2-d parts also tend to be convex (or mostly convex) with fewer ambiguities in the correspondence of shape. Some parts do vary considerably in location and scale, and parts may consist of multiple closed polygons, but these issues can be handled automatically.

5.3.1 Shrink-wrapping to polygons

The initial parts used for automatic initialization are shown in Figure 5.10. These template parts are the approximate size, shape, and location of the parts mapped from CAD models. However, they are not close enough in all cases to use as is. To help with location, each initial template part is moved such that its centroid of vertices aligns with the centroid of vertices in the corresponding mapped part. Aligning location is especially important for parts like the license plates that tend to change position significantly between vehicles. The size of the vertices is also scaled up by a factor of 1.5 (relative to the centroid) prior to shrink-wrapping. Scaling helps to ensure that the template part lies outside of the target mapped part. If the template vertices lie inside the target part it is possible that they could map to the wrong side of the polygon causing the shape to collapse.

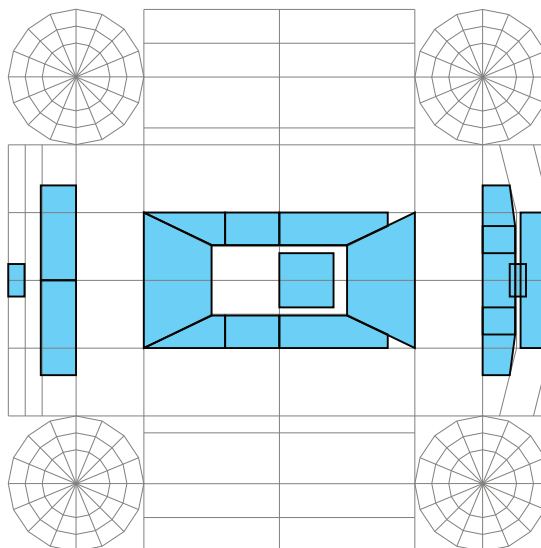


Figure 5.10: Initial template parts

Some mapped parts are represented by a cluster of closed polygons rather than a single one. This tends to happen most frequently with headlights, taillights and grilles. Headlights and taillights often are composed of several different colored pieces, and grilles often are divided into sections. CAD model designers tend to assign separate mesh components to these sub-parts. The number of sub-parts varies from vehicle to vehicle, so only the boundary of the group is modeled for consistency. A single template part can be fit to a cluster of polygons to represent the cluster boundary, but internal edges can cause some trouble. To avoid these problems, the template parts are first fit to the convex hull of the mapped part. In addition to creating a single closed polygon, the convex hull also tends to smooth the boundary reducing the sampling problems related to fitting a simple template part to a complex mapped part shape. Since the parts tend to be mostly convex, using the convex hull is good first approximation to the shape.

The 3-d shrink-wrapping procedure maps points to the target surface along the computed normal directions. This process only works well when the initial shape is very close to the target shape.

The automatic initialization of 2-d parts is not always good enough for this method. If the shapes differ significantly, the computed normals will also differ and shrink-wrapping along the normal directions can be disastrous. A dual space optimization of part shape based on the work of Ohtake and Belyaev [81] was found to work much better. First, the template vertices are mapped to the closest points on the convex hull of the target polygon. This first pass may be a poor approximation of the convex hull because of poorly distributed samples. The centers and normals of each template polygon side are then computed. This is the dual representation of the polygon. The line centers are projected along their normal directions to intersect the convex hull. The centers are moved to the intersection points and the line orientation is changed to be tangent to the convex hull at these intersection points. The positions of the original primal mesh vertices are recomputed as the intersection of the adjacent lines. Occasionally the lines are nearly parallel and a good intersection point does not exist. In this case, the algorithm reverts to the previous vertex location. This dual mesh procedure redistributes the template vertices to better match the target shape. These new vertex locations may now lie off of the convex hull, so they are mapped once more to closest point on the convex hull. The middle row of Figure 5.11 illustrates the processing so far. The final vertex positions after this first stage of processing correspond better to corners features than the original closest point projections. This dual mesh mapping procedure is particularly important for vertically oriented taillights, since the initial template taillight has a horizontally oriented shape.

After the first stage of fitting to the convex hull, the template part is well aligned with the overall part shape. At this point it is safe to expose the concavities of the part by switching back from the convex hull to the original polygon (or cluster of polygons). As long as the gaps between the sub-parts are not too large and the concavities are not too complex, the template shape can safely deform further to capture some of the concavities of the part. Doing so may require a finer level of detail in the template part, so the polygon is subdivided by adding new vertices at the center of each side. This subdivision is analogous to the subdivision of mesh faces in 3-d. The dual space fitting process is now repeated starting with the step that maps dual vertices along the normal directions. The bottom row of Figure 5.11 illustrates this second stage processing. As with the mesh subdivision, it is possible to recursively subdivide the template parts and refit to get closer and closer approximations to the shape. In practice, only one level of subdivision is needed to get a good representation of the part shape. Figure 5.12 shows the mapped CAD model parts and the corresponding fit of the template parts for the vehicle in Figure 5.6.

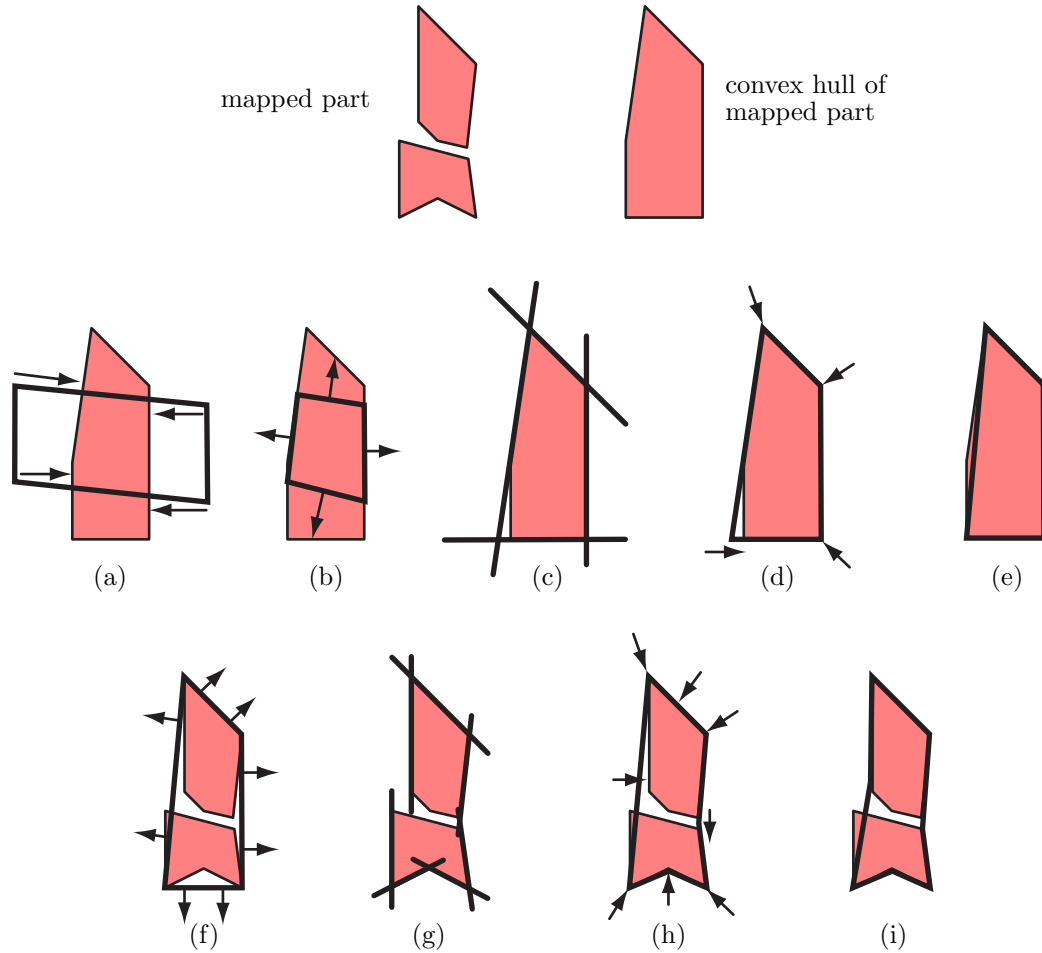


Figure 5.11: Procedure for fitting 2-d parts: (a) Map initial part vertices to closest points on convex hull. (b) Map dual vertices (line centers) to convex hull along normal directions. (c) Make lines tangent to convex hull and intersect to recover vertices. (d) Map vertices to the closest points on convex hull. (e) First stage result; subdivide the template polygon sides. (f) Map dual vertices along normal direction to original target part. (g) Make lines tangent to target part and intersect to recover vertices. (h) Map vertices to closest points on target part. (i) Final result.

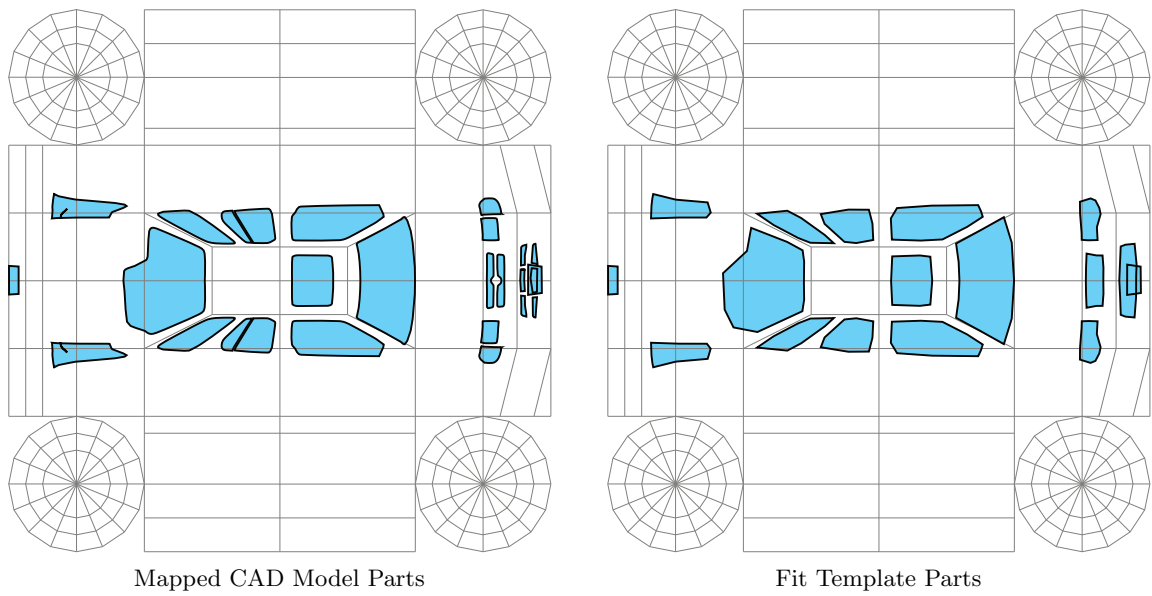


Figure 5.12: Fitting template parts to mapped parts. On the left, original parts mapped directly from the CAD model. On the right, template parts deformed to represent the parts on the left.

Chapter 6

Tracking in Video

More interesting than fitting a vehicle model to a still image is the task of fitting the model to a sequence of images from a video. After all, vehicles are designed for transportation, and motion is an inherent vehicle characteristic. In some surveillance applications, measuring the trajectory of a vehicle can be even more important than recovering shape and identity. The task of recovering vehicle pose as it evolves over time is called **tracking**.

The term *tracking* is used broadly in the computer vision literature to refer to any process that estimates change over time. Tracking can take on a different meaning depending on the object being tracked and the space in which it is tracked. With respect to vehicle tracking in video, the objects may be low level features or models of an entire vehicle, and the space may be 2-d image coordinates or 3-d world coordinates. See Chapter 2 for a review of the literature and examples of different types of tracking. In this thesis, tracking refers to estimation of the 3-d pose of a vehicle model as it evolves over time.

In contrast to most other 3-d, model-based, vehicle tracking approaches [1, 80, 39, 46, 62, 72, 84, 87, 49], the model shape is not assumed to be known *a priori*. Instead, the shape is estimated simultaneously with the motion. In most previous tracking work, a fixed model (or fixed set of shape parameters) is used during tracking. Either the shape is assumed to be known, or the shape is selected from a small number of models, usually one from each class of vehicles. In all of these cases, the model is the same as (or a slight variation of) Koller’s deformable model shown in Figure 2.16. The only study found on simultaneous tracking and shape estimation was performed by Koller *et al.* [59]. Ferryman *et al.* used their vehicle model to study both shape estimation [30] with manual pose initialization and tracking [49] with fixed shape, but not both shape estimation and tracking together. This chapter does not aim to make significant new strides in the area of vehicle tracking. Rather, the goal is to describe the standard algorithm used by Koller and modify it to work with the vehicle fitting framework described in Chapter 4. The end result is to study the effects of the choice of vehicle model on the performance of joint tracking and shape estimation.

To achieve optimal joint tracking and shape recovery results, one could pose the problem as simultaneous estimation of shape and time varying pose from a fixed collection of images. In other

words, one could select a range of frames from a video and apply a slight modification of multi-view optimization (Section 4.3.5) on all frames at once. A fixed camera imaging a moving vehicle is equivalent to a fixed vehicle imaged by a moving camera, and a moving camera generates multiple camera viewpoints. The only relevant difference compared to multi-view fitting is that the vehicle track is unknown so the cameras do not have relative extrinsic (*i.e.* pose) calibration. However, temporal continuity ensures that the pose change is continuous and small between adjacent frames. In terms of multi-view optimization, one would need to provide a model of pose variation over time (*i.e.* a motion model) and estimate its parameters in the optimization.

Unfortunately, this batch tracking framework is not usually practical or desirable in real applications. It is not typically known in advance which frames contain an entire vehicle track. Also, the complete set of frames containing a given vehicle can potentially become quite large. As the number of frames grows, the time and memory requirements for multi-view optimization can easily become prohibitively large. Furthermore, video data often comes from live streaming sources. It is desirable to have real-time algorithms estimate the changing vehicle pose as each new frame arrives. Such algorithms are called **online** trackers (as apposed to batch processing trackers which are called **offline**). Although some work has been devoted to offline tracking (including earlier work by the author [68]), the vast majority of interest is in online trackers. The rest of this chapter is devoted to online tracking.

Online tracking is the process of finding the pose of a vehicle in the current frame given the tracking results from previous frames and possibly a model of motion. When shape is also involved, the current frame should help refine the current shape estimate such that shape eventually converges after many observations. However, this recursive definition, like all recursive processes, needs a base case. This base case is a separate process called **track initialization**. To start the recursion, one must obtain an initial estimate of the vehicle state from some other algorithm. The initialization process must detect the vehicle and provide an initial guess of pose, shape, and motion. This chapter proceeds by first discussing initialization, and then following with a discussion of online tracking.

6.1 Track initialization

Track initialization relies on the ability to detect and orient a vehicle in one frame of video. Vehicle detection algorithms have been a popular research area in recent years, and detector performance is getting quite good. Some detectors [88, 97] use a 3-d vehicle model to generate templates for vehicle detection. Again, a fixed vehicle model is used. Another popular approach is to detect local features (*e.g.* interest points) and match their local descriptors to a code book of learned vehicle features. The matched features then vote for a vehicle centroid using a generalized Hough transform. See Leibe *et al.* [67] for a detailed explanation. This approach was initially limited to 2-d detection for a fixed viewpoint, but recent work [70, 83, 101] has extended feature-based matching to handle multiple viewpoints and estimation of orientation.

Single image vehicle detection algorithms are very useful for the case of still images. However,

they tend to be difficult to implement and typically require a large database of segmented images for training. Since detection is not a major focus in this thesis, these techniques are avoided in favor of a simpler method.

The goal of detection is to initialize vehicle tracking in video. A video sequence has additional temporal information not available to single image detectors. One can exploit the temporal information to obtain a simple detector for moving vehicles. The general concept is this: a group of pixels with different appearance from the background, consistent motion, and approximate vehicle size and shape are considered for vehicle track initialization. This detection criteria requires a static camera, is sensitive to occlusion, and does not handle multiple vehicles in close proximity. However, similar approaches are used by many other authors, [1, 12, 18, 80, 39, 63, 61] including Koller [59], and it will be sufficient for the experiments in this thesis.

Before diving into the details of the track initialization, a brief review of background modeling and feature point tracking (sparse optical flow) is needed.

6.1.1 Video background modeling

Background modeling was already introduced in Section 2.2.3, and used to produce the regions in Figure 2.7. To review, the goal is to segment each frame of video into foreground (FG) and background (BG) pixels ¹. Assuming the camera is stationary, each pixel will observe the same portion of a static scene over time, and each pixel can be modeled independently. These pixels will have little variation in appearance while observing a static background. Moving objects, however, do not fit this model. The pixel appearance tends to change rapidly as an object moves across the scene. Thus, the static (or slowly varying) background appearance is modeled statistically, and observations that do not fit the background model are classified as foreground.

The background appearance is not entirely stationary so the model does need to be updated to account for gradual illumination changes. As with tracking, it is most useful if the background model is learned in an online fashion. Each new frame is used to update the model and the model slowly adapts to changes over time. The seminal and most cited work in adaptive background modeling is by Stauffer and Grimson [98]. This work uses a mixture of Gaussians (MoG) to represent the background model independently at each pixel in the image,

$$P(\mathbf{o}_t) = \sum_{k=1}^K w_{k,t} \mathcal{N}(\mathbf{o}_t | \mu_{k,t}, \Sigma_{k,t}), \quad (6.1)$$

where $\mathbf{o}_t$ is the pixel observation at time t , $w_{k,t}$ is a weight for each of K components, and $\mathcal{N}(\mathbf{o}_t | \mu_{k,t}, \Sigma_{k,t})$ is a Gaussian (*i.e.* normal) density with mean $\mu_{k,t}$ and covariance $\Sigma_{k,t}$ evaluated at $\mathbf{o}_t$. Each new observation is matched to one Gaussian component in the mixture if it is within a threshold on Mahalanobis (*i.e.* covariance normalized) distance. This threshold is typically 2.5. If

¹Two general purpose C++ software libraries were written to support this thesis and applied in the implementation of background modeling in video. The first is a video steaming library, and the second is probability distribution library. These libraries are now among the core libraries in the VXL project. Refer to Appendix B for details and documentation.

matched, the observation is used to update the Gaussian component's mean, covariance, and weight. If not matched to any component, a new component is created with high covariance, low weight, and a mean at the new observation. A new component may replace the least important component if a maximum number of components has been reached. The importance of components is determined by the ratio of weight to standard deviation, $w_{k,t}/\sigma_{k,t}$. Note that Stauffer and Grimson use a 3-d observation vector (in RGB color space) but assume the covariance is spherical (*i.e.* $\Sigma_{k,t} = \sigma_{k,t}^2 \mathbf{I}$).

Background segmentation is also determined by mixture component importance. Components are kept sorted by decreasing importance. High weight, low standard deviation components are first. The first B components are selected as background components; the rest are foreground. B is adaptively chosen to be the minimum number such that the percentage of total weight of the B components exceeds a threshold, T_B . Therefore,

$$B = \arg \min_b \left(\sum_{k=1}^b w_k > T_B \right), \quad (6.2)$$

where a reasonable value for T_B is 0.7 (assuming all weights sum to 1). Observations that match a background component are classified as background. Observations that match a foreground component or do not match any component are classified as foreground. Figure 6.1 illustrates an example of a 1-d (greyscale) MoG and shows the FG/BG decision boundaries. Refer to [98] for more details.

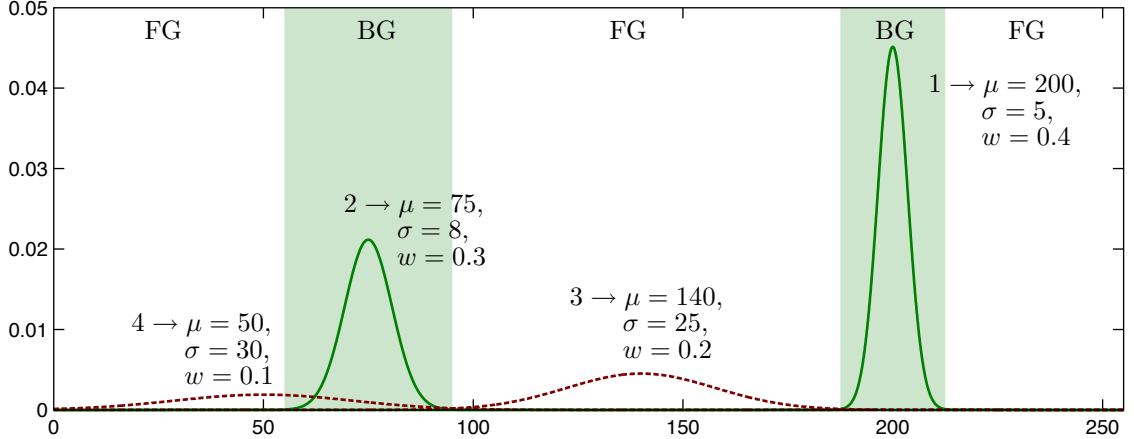


Figure 6.1: An example of a 1-d mixture of Gaussians plot. This mixture has 4 components sorted by importance (w/σ). The first 2 Gaussians represent the background. The green areas are background (BG), and the white is foreground (FG).

The common justification for the MoG model is that some oscillating parts of the background, like leaves on a tree or a flag in the wind, will be multimodal. This observation is true, but in practice moving leaves, flags, etc. exhibit a wide distribution of appearance that is not captured by a small number of sharply peaked Gaussians. As a result, these periodic motions are still frequently detected as foreground. Only a few real world phenomena, like blinking traffic lights, are well modeled by two sharply peaked modes. Almost all background pixels are ultimately represented by

a single Gaussian. The real power of the MoG model comes from the ability to model many different foreground observations with temporary, disposable Gaussians. This has two key advantages over a single Gaussian appearance model. One, the outlying foreground points do not corrupt background model. Two, a foreground object can become part of the background if stationary for long enough (*e.g.* a parking car). This transition can occur abruptly. The background model is then bimodal (old background, new background), and can resume learning the old background quite quickly if the object then moves again after a short time (*e.g.* the car leaves the parking space).

Stauffer and Grimson's background model is still commonly used today, mostly in its original form. In the original paper, the update equations are

$$\rho_{k,t} = \alpha M_{k,t} \mathcal{N}(\mathbf{o}_t | \mu_{k,t}, \sigma_{k,t}) \quad (6.3)$$

$$w_{k,t} = (1 - \alpha)w_{k,t-1} + \alpha M_{k,t} \quad (6.4)$$

$$\mu_{k,t} = (1 - \rho_{k,t})\mu_{k,t-1} + \rho_{k,t}\mathbf{o}_t \quad (6.5)$$

$$\sigma_{k,t}^2 = (1 - \rho_{k,t})\sigma_{k,t-1}^2 + \rho_{k,t}(\mathbf{o}_t - \mu_{k,t})^\top (\mathbf{o}_t - \mu_{k,t}) \quad (6.6)$$

where $w_{k,t}$, $\mu_{k,t}$, and $\sigma_{k,t}$ are the weight, mean, and standard deviation respectively of component k at time t . $\mathbf{o}_t$ is the current pixel observation, α is a small constant learning rate, and $M_{k,t}$ is 1 when component k is the match, 0 otherwise. The learning rate for $\mu_{k,t}$, and $\sigma_{k,t}$ is $\rho_{k,t}$, which is α weighted by the normal density at $\mathbf{o}_k$. This choice of $\rho_{k,t}$ does not seem to be well justified and can lead to serious problems. If the data is scaled to the range $[0,1]$ (instead of $[0,255]$), the normal density will frequently evaluate to values much greater than 1. In turn, $\rho_{k,t}$ can become greater than 1, and this is unacceptable for a learning rate.

Several other researchers have proposed minor alterations to the original equations. These alterations have focused mostly on the slow convergence rate at initialization. The constant α means online learning is slow. A slow learning rate is fine in the steady state when the model is already accurate, but it does not work well at initialization. The model is biased by its initial conditions and may take hundreds of frames to overcome them. Conversely, taking the sufficient statistics approach to online learning, the learning rate is effectively $\alpha_t = 1/(t + 1)$ at the t -th observation. So learning is initially very fast and then rapidly slows down and goes to zero in the limit as $t \rightarrow \infty$. KaewTraKulPong and Bowden [53] propose a compromise in which the sufficient statistics approach is followed while $t < L$ and then $\alpha_t = 1/L$ is used in the steady state when $t \geq L$ for some window size L . Power and Schoonees [85] take a more principled approach to the derivation of the update equations and show that a better choice for $\rho_{k,t}$ is $\rho_{k,t} = \alpha_t M_{k,t} / w_{k,t}$. Intuitively, $w_{k,t}$ is large for well established components causing them to adapt more slowly, and it is small for new components causing them to adapt more quickly. Lee [65] takes this idea one step further and actually accounts for the number of observations seen by each mixture component. He reverts to the use of a constant α , but assigns

$$\rho_{k,t} = M_{k,t} \left(\frac{1 - \alpha}{c_{k,t}} + \alpha \right) \quad (6.7)$$

where $c_{k,t} = c_{k,t-1} + M_{k,t}$ is a count of the number of observations matched to component k . This method starts with a learning rate of $\rho_{k,t} = 1$ and then gradually transitions to $\rho_{k,t} = \alpha$ in the limit

as $c_{k,t} \rightarrow \infty$. The separate observation count per mixture component is important in ensuring that the learning is consistent at all stages of operation, since new components can be created at any time. New components should always start with the same fast learning rate even if thousands of frames of video have been observed before their creation.

The update equations in this thesis come from the authors earlier work [68] on estimating background models that account for shadow. The equations combine the varying α_t of [53] and [85] with the count-per-component of [65]. Therefore

$$\alpha_t = \max\left(\frac{1}{t}, \frac{1}{L}\right) \quad \rho_{k,t} = M_{k,t} \left(\frac{1 - \alpha_t}{c_{k,t}} + \alpha_t \right) \quad (6.8)$$

where L is the sliding window size in the steady state. In practice, $L = 300$ frames is used (a 10 second window at 30 frames per second). The update equations themselves are

$$w_{k,t} = (1 - \alpha_t)w_{k,t-1} + \alpha_t M_{k,t} \quad (6.9)$$

$$\mu_{k,t} = (1 - \rho_{k,t})\mu_{k,t-1} + \rho_{k,t}\mathbf{o}_t \quad (6.10)$$

$$\Sigma_{k,t} = (1 - \rho_{k,t})(\Sigma_{k,t-1} + \rho_{k,t}(\mathbf{o}_t - \mu_{k,t-1}) \circ (\mathbf{o}_t - \mu_{k,t-1})). \quad (6.11)$$

There are a few subtle changes in Equation (6.11). First, (6.11) is now a matrix update. It is assumed that $\Sigma_{k,t}$ is still diagonal (*i.e.* independent channels), but each channel may now have a different variance. The $\circ$ operator indicates element-wise vector multiplication. In this case, $\Sigma_{k,t}$ represents the vector on the diagonal of the matrix. Equation (6.11) also generalizes to full covariance matrices if $\circ$ is replaced by an outer product. It also specializes to scalar variance if $\circ$ is replaced by an inner product. Since the covariance is no longer scalar, the original measure of component importance no longer applies. Hence, the importance measure is generalized to $w_{k,t}^3 / \det(\Sigma_{k,t})$ for three color channels. Second, the data term now differs by a factor of $(1 - \rho_{k,t})$ (notice the intentional rearrangement of parentheses), and it also uses the mean from the previous time step. These changes are to bring (6.11) inline with the correct equations for sufficient statistics. If one takes $\alpha = 0$ then (6.11) produces same results as

$$\Sigma_{k,t} = \frac{1}{t} \sum_{i=1}^t (\mathbf{o}_i - \mu_{k,t}) \circ (\mathbf{o}_i - \mu_{k,t}) \quad (6.12)$$

for all values of t . Equation (6.6) does not have this property. In the steady state, as $\rho_{k,t}$ becomes small, (6.11) approximates (6.6). However, if the number of observations is small and $\rho_{k,t}$ is large, then (6.6) is a significantly biased.

A typical result of applying background modeling and FG/BG segmentation is shown in Figure 2.7. There is some noticeable noise in the segmentation that manifests itself as scattered misclassifications. Both false positives (pixels on moving objects labeled BG) and false negatives (background pixels labeled FG) are present. False positives arise from moving pixels that happen to look like the background. False negative arise from sensor noise at the tails of the distribution and small background motions that are not captured by the background model. Most of the errors are sparsely scattered throughout the image, while the true segmentation tends to involve large regions

with consistent labels. Since each pixel is modeled independently, there is no constraint that enforces local continuity of labels between adjacent pixels.

Morphological post-processing

It is beneficial to reduce the scattered errors in FG/BG segmentation because connected components of FG regions will later be used for vehicle detection. One good solution is to construct a Markov random field (MRF) over adjacent pixels and solve for the optimal segmentation using graph cuts [13]. The investigation into the use of this method is left for future work. For simplicity and efficiency, a common fast solution is post-processing with mathematical morphology operations [90].

Mathematical morphology operations on binary images are composed of two complimentary operations called **erosion** and **dilation**. These operations are defined in terms of a set of local neighborhood relations called the **structuring element**. While the structuring element can be defined arbitrarily, a simple 3×3 box for 8-connectivity is used in this thesis. A morphological operator is applied at each pixel in a binary image and all neighbors specified by the structuring element are considered. The erosion operator sets a pixel to zero if any of its neighbors are zero. The dilation operator sets a pixel to one if any of its neighbors are one. As a result, image regions defined by pixels of value one (*i.e.* the background) will erode with the erosion operator and dilate with dilation operator. The top row of Figure 6.2 shows the erosion and dilation operations applied to Figure 2.7.

The erosion operation expands the foreground region and swallows up isolated false positives. Applying dilation after erosion shrinks the foreground region back approximately to its original form without restoring the isolated false positives. This combined operation of erosion followed by dilation is called the **opening** operation. Similarly, the dilation operation expands the background region and swallows up isolated false negatives. Applying erosion after dilation shrinks the background region back approximately to its original form without restoring the isolated false negatives. This combined operation of dilation followed by erosion is called the **closing** operation. The opening and closing operations are shown in the middle row of Figure 6.2.

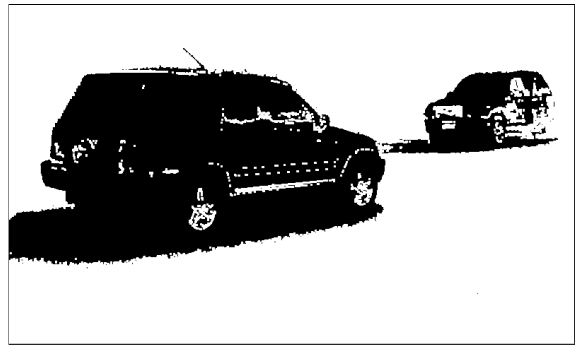
Since it is desirable to reduce both false positives and false negatives, both opening and closing operations are used. As with erosion and dilation, opening and closing are not commutative. The bottom row of Figure 6.2 compares the results of applying opening followed by closing to applying closing followed by opening. Opening followed by closing removes more false positives but leaves some false negatives. Closing followed by opening removes more false negatives but leaves some false positives. The next stage of track initialization involves finding connected components of foreground pixels. For this purpose extra foreground detections are more problematic than extra holes in the foreground regions. Therefore, closing followed by opening is applied to clean up the segmentation.

6.1.2 Sparse optical flow

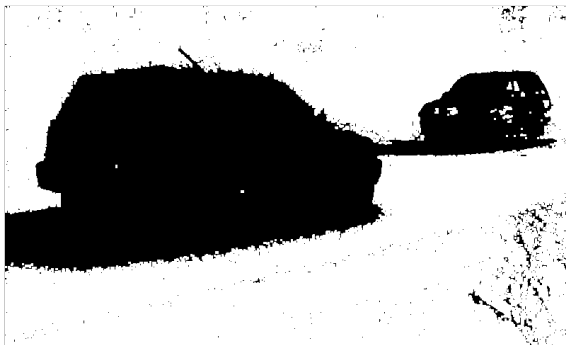
The detection of foreground regions is helpful in determining the location of moving objects, but it does not reveal anything about the direction of motion. For track initialization it is critical to also



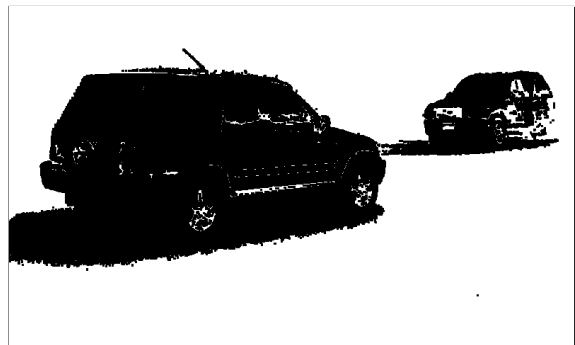
Erosion



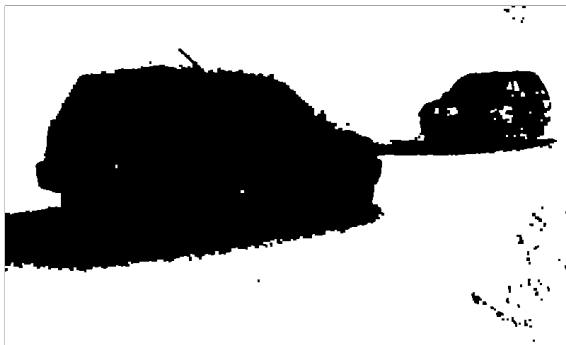
Dilation



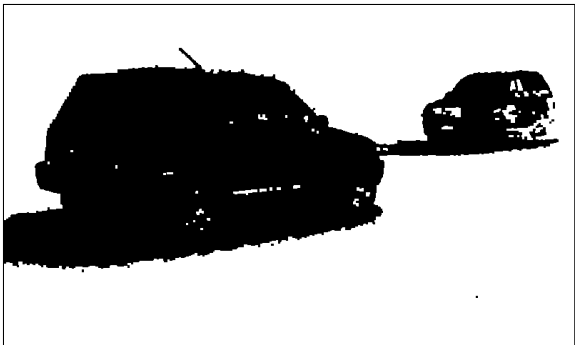
Opening



Closing



Opening followed by Closing



Closing followed by Opening

Figure 6.2: Mathematical morphology operations applied to the FG/BG segmentation in Figure 2.7. Top: erosion and dilation. Middle: opening and closing. Bottom: opening followed by closing and closing followed by opening.

estimate an approximate orientation of the vehicle model. Under the assumption that the vehicle is driving forward, one can make use of estimated motion in the image plane to reason about motion in 3-d and therefore about orientation of the moving vehicle. To obtain image motion information one must track image features over time. When a sparse set of interest points is tracked between a pair of video frames it is called **sparse optical flow**.

Optical flow is a well studied problem in the computer vision literature. Most of the work focuses on dense optical flow, the estimation of motion vectors at every pixel in a video frame. While optical flow is well-conditioned at interest points, it is ill-conditioned along edges (aperture problem), and even less constrained within uniform regions. The challenge in dense optical flow is to propagate the constraints from interest points to the rest of the image to generate a piecewise smooth flow field.

For the purpose of vehicle track initialization it is sufficient to only estimate flow at the sparse set of well-conditioned points. The sparse optical flow algorithm used in this thesis is the Kanade-Lucas-Tomasi (KLT) feature tracker [94]. The existing C code implementation of KLT written by Stan Birchfield [7] is used in this thesis. The points detected for tracking in KLT are very closely related to the Harris corners shown in Figure 2.5. Both Harris corners and KLT start by computing the structure matrix

$$\mathbf{A} = \begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix} \quad (6.13)$$

at every pixel, where I_x and I_y are the horizontal and vertical image gradients and $\langle \cdot \rangle$ indicates an average over a small local neighborhood. The eigenvalues (λ_1, λ_2) of $\mathbf{A}$ are indicative of the structure at that point. The eigenvalues are computed according to

$$\lambda_{1,2} = \frac{1}{2} \left(\text{tr}(\mathbf{A}) \pm \sqrt{\text{tr}^2(\mathbf{A}) - 4 \det(\mathbf{A})} \right). \quad (6.14)$$

Intuitively, two large eigenvalues indicate a corner (*i.e.* interest point), one large and one very small eigenvalue indicates an edge, and two very small eigenvalues indicate a flat region. The Harris corners measure of “cornerness” is computed as $\det(\mathbf{A}) - \kappa \text{tr}^2(\mathbf{A})$ where κ is a small constant typically in the range $[0.04, 0.15]$. This measure was chosen to avoid the computation of the square root needed to obtain the eigenvalues (a square root was considered relatively expensive in the 1980’s). In KLT, the eigenvalues are computed directly, and the measure of cornerness is computed as $\min(\lambda_1, \lambda_2)$. A recent study by Bruce and Kornprobst [14] compares these measures in more detail.

The set of points with a minimum eigenvalue greater than a threshold of 10 are considered for tracking in this thesis. These points are also estimated to sub-pixel accuracy using parabolic interpolation. A more standard approach to point selection is to sort all of the points by decreasing cornerness, and then select points in a greedy fashion such that no selected point is within some small distance of any previously selected points. The selection continues until reaching a maximum number of points, often 1000. The selection criterion is intended to find spatially distributed points that will track unambiguously over several consecutive frames. This property is not required for vehicle pose initialization. Instead the goal is to approximately track many points for no more than two frames to get a sense of the average instantaneous motion. For this task, a simple threshold is sufficient and far more efficient.

Once the set of trackable points is determined, the motion is estimated according to the brightness constancy constraint. This constraint assumes that the image brightness remains constant over time but changes in position. Assuming small motion, a first order Taylor expansion of the image as a function of space and time leads to the equation:

$$I_x \delta x + I_y \delta y = -I_t \quad (6.15)$$

that relates the unknown motion vector $(\delta x, \delta y)$ to the spatial derivatives I_x and I_y and temporal derivative I_t . Equation (6.15) provides only one constraint for two unknowns, so KLT further assumes constant flow in a local neighborhood and constructs a system of equations using the pixels in a small region around the point. One additional challenge is that these linear equations quickly break down for motion larger than about a pixel in length. KLT solves this problem using a coarse-to-fine scale-space approach (scale-space was discussed briefly in Section 4.3.4). At coarse enough scales, the image motion will reduce to less than a pixel. The estimated flow at the coarse scale is used to initialize the algorithm within a pixel at the next finer scale. This process repeats until reaching the original fine scale image. Refer to [94, 7] for more details on the KLT algorithm.

Figure 6.3 shows the resulting flow vectors on the frame of video from Figure 2.1. In practice, the flow is computed from frame t to frame $t - 1$ since future frames have not yet been observed. The resulting flow vectors point backwards, but they are easily corrected by negating them. Notice that most of the vectors on each vehicle point in the approximate direction of motion.

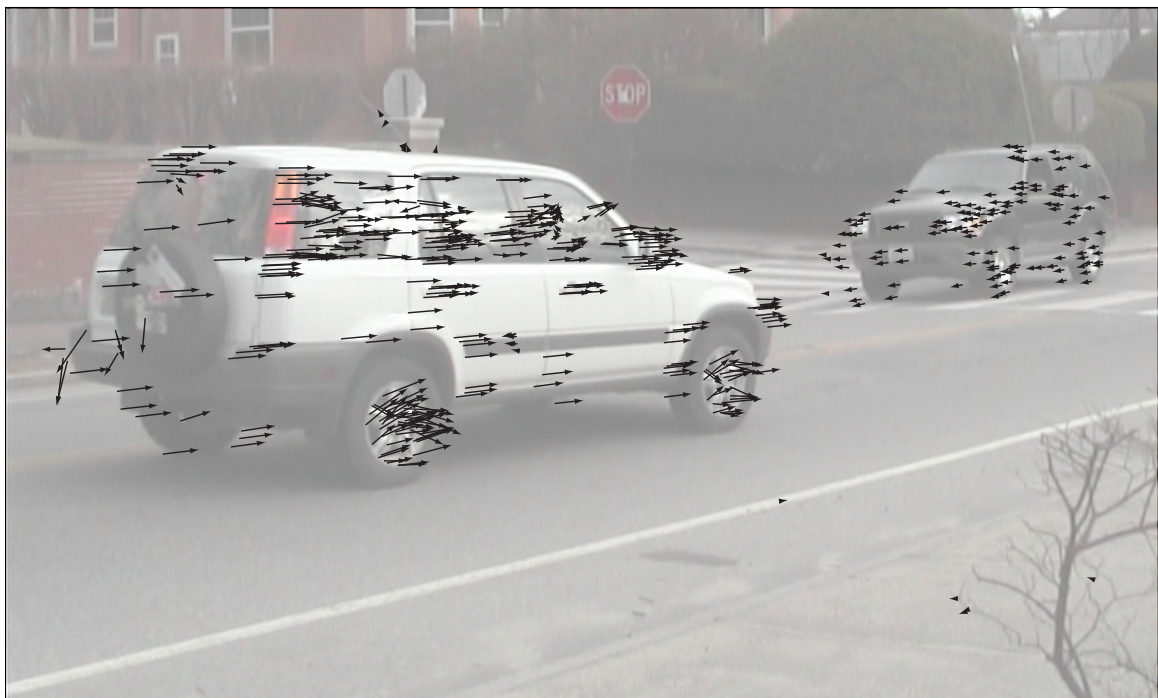


Figure 6.3: KLT flow vectors from the video frame in Figure 2.1.

6.1.3 Initializing vehicle pose and shape

Combining foreground region detection with sparse optical flow provides enough cues to generate a rough vehicle pose and shape estimate in a calibrated 3-d world. For both track initialization and tracking itself, it is assumed that the camera is calibrated relative to a known ground plane on which the vehicles drive. The strategy is to place the 3-d vehicle model on the ground plane and adjust its orientation such that it is consistent with the optical flow. Then, with a fixed orientation, adjust the translation and rough shape such that the silhouette of the model projection aligns well with a foreground region. This strategy is quite similar to that proposed in the Master Thesis of Ambardekar [1], although he considers fixed shape models.

The first step of initialization is to associate connected foreground regions with vehicles. All connected components are found in the morphologically post-processed foreground detection. The exterior boundary of each region is traced as a closed contour (holes in the region are ignored). Then the area and circumference (*i.e.* boundary length) of each region are computed. A few heuristics help to quickly reject implausible or unusable regions. Thresholds are determined pragmatically.

Completeness Only complete, closed regions are considered. Regions touching an image boundary are ignored because it is difficult to judge how much of the vehicle lies outside of the image.

Minimum area Regions must have an area larger than a minimum threshold (4000 pixels is used). Otherwise, they are either not vehicles, or are considered too small to track.

Circumference compared to area Vehicles silhouettes are relatively compact with slowly curving boundaries. The ratio of the circumference to the square root of area should be smaller than a threshold (10 is used). For comparison, this ratio is minimized with a circle which has a ratio of $2\sqrt{\pi} \approx 3.5$, and a square has a ratio of 4.

Area compared to bounding box area This heuristic also evaluates compactness. The area of the region should be no less than one half the area of the bounding box around the region.

The last two heuristics help to reject unusable regions created by the union of regions from multiple moving objects. The regions in Figure 6.2 are an example of this situation. Both vehicles are joined into a single foreground region that is rejected by these heuristics.

Any region that passes these tests is a candidate for vehicle track initialization. For each region, all optical flow vectors originating in the region are found and associated with that initialization candidate.

Vehicle orientation

The next step of initialization is to estimate the vehicle orientation. For a given initialization candidate, the associated optical flow vectors are considered. Since the vehicle can only move parallel to the ground plane, the 3-d vectors corresponding to the 2-d optical flow vectors must be parallel to the ground plane. Only the direction of these 3-d vectors is of interest. There is no need

to recover the 3-d position of the interest points or magnitude of the vectors at this time. One can map each optical flow vector from the image plane to the ground plane (or any plane parallel to the ground plane) and then compute the orientation in world coordinates.

The mapping between the image plane and any world plane is computed by a 3×3 matrix in homogeneous coordinates called a homography. Assume that the world coordinates are such that the Z -axis is perpendicular to the ground. Let $\mathbf{c}_i$ be the 4 columns of the 3×4 camera matrix $\mathbf{P} = [\mathbf{c}_1 \ \mathbf{c}_2 \ \mathbf{c}_3 \ \mathbf{c}_4]$. Then the homography mapping the world plane at $Z = h$ to the image is

$$\mathbf{H} = \begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & (h\mathbf{c}_3 + \mathbf{c}_4) \end{bmatrix} \quad (6.16)$$

The homography mapping from the image back to the $Z = h$ plane is $\mathbf{H}^{-1}$. For simplicity, one can use the ground plane at $h = 0$.

The head and tail of each optical flow vector are mapped by $\mathbf{H}^{-1}$, and the orientation of the resulting ground plane vector is computed. To account for outliers, the orientation angles are tabulated in a histogram with eight bins, each of width 45° . The angles in the most populous bin are averaged to obtain the orientation estimate. If any adjacent bin contains more than 75% of the peak population then its angles are also averaged in. Considering adjacent bins accounts for an orientation near a bin boundary. The estimated orientation is held fixed for the rest of the initialization process.

Shadow and Silhouette prediction

With the orientation fixed, it remains to initialize the position and shape using the detected foreground region geometry. To that end, a prediction of the foreground region must be generated from the 3-d model. For any hypothesized pose and shape, one can compute the projection of the mesh occluding contours as described in Section 3.4. Computing the self-intersections of these contours and tracing the outermost edges results in a prediction of the vehicle silhouette. The silhouette contour is always a subset of the occluding contour generator. Therefore, the same Jacobian computations from Section 4.3.1 describe changes in the silhouette shape. Figure 6.4 shows an example of the predicted image edges and the traced silhouette.

Unfortunately, it is immediately obvious from Figure 6.2 that the vehicle silhouette is not sufficient to predict the detected foreground region. A large contribution to the foreground comes from the shadow cast by the vehicle onto the ground. It is very difficult to accurately separate the true foreground from the shadowed background based on local appearance alone. Fortunately, there is no need to make such a separation. Modeling the vehicle as a solid shape in the 3-d world makes it easy to predict the appearance of the shadow as well. In fact, the existence of shadow can even be helpful. The shadow cast by the sun onto the ground plane is much like a second camera viewing the vehicle from the direction of the sun, but only silhouette information is imaged onto the ground plane. Figure 2.19 illustrates the image projection of a point and the shadow of that point (with respect to a rigid vehicle model used in the author's earlier work [68]).

One can construct a 3×4 camera projection matrix, $\mathbf{P}_s$, that projects the 3-d vehicle model

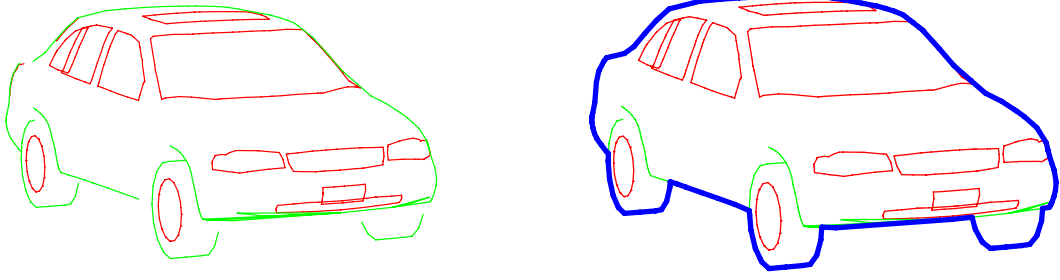


Figure 6.4: Silhouette prediction. On the left, visible occluding contours and parts as described earlier. On the right, the silhouette traced from the outer occluding contour generator edges, drawn in blue.

directly to the image of its ground plane shadow. Henceforth, $\mathbf{P}_s$ will be called the **shadow camera**. The shadow camera is constructed from the concatenation to two projections. First, the projection of the vehicle into the ground plane by the light source. Second, the projection of the ground plane points into the image plane of the real camera. The latter projection is, of course, just the matrix $\mathbf{P}$ used to project the vehicle directly into the image plane. The former projection is a degenerate 4×4 homography, $\mathbf{H}_s$, that flattens the vehicle into the $Z = 0$ plane. Thus, $\mathbf{P}_s = \mathbf{P}\mathbf{H}_s$. This thesis will only consider shadows from a single directional light source, namely the sun. However, these concepts are easily generalized to point lights (*e.g.* street lamps) and multiple light sources. A point light source is the dual of a perspective camera, and a directional light source is the dual of an affine camera.

It is assumed that the direction of the sun light is known and represented by 3-d vector $\mathbf{d} = [d_x \ d_y \ d_z]^\top$. This is a reasonable assumption because position of the sun, although continually varying, is easy to predict. An example of a sinusoidal model for solar position is that of Carruthers *et al.* [16]. It computes the azimuth and altitude of the sun given latitude, longitude, date, and time. This model was designed for architectural design, but it was adapted for vehicle shadow prediction in the author's earlier work [68]. For a directional light source like the sun (*i.e.* with all rays parallel), the vehicle flattening homography takes the simple form

$$\mathbf{H}_s = \begin{bmatrix} 1 & 0 & -d_x/d_z & 0 \\ 0 & 1 & -d_y/d_z & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (6.17)$$

Instead of computing the matrix multiplication $\mathbf{P}\mathbf{H}_s$, one can simply form

$$\mathbf{P}_s = \begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & -(\frac{d_x}{d_z}\mathbf{c}_1 + \frac{d_y}{d_z}\mathbf{c}_2) & \mathbf{c}_4 \end{bmatrix} \quad (6.18)$$

using the columns, $\mathbf{c}_i$, of $\mathbf{P}$.

The shadow camera can be treated just as another view of the vehicle. The occluding contours are computed (with respect to the light direction) and projected into the image. All the Jacobian computations of Section 4.3.1 still apply. The vehicle silhouette and shadow silhouettes will overlap in the image, so a single silhouette is formed by tracing the outer boundary of the union of the occluding contours as observed by both cameras. The resulting silhouette polygon has sections coming from both sources. Information is retained about which mesh vertex generated which silhouette point and by which projection in order to facilitate Jacobian computation. Figure 6.5 shows examples of vehicle silhouettes generated by the union of the model and its cast shadow.

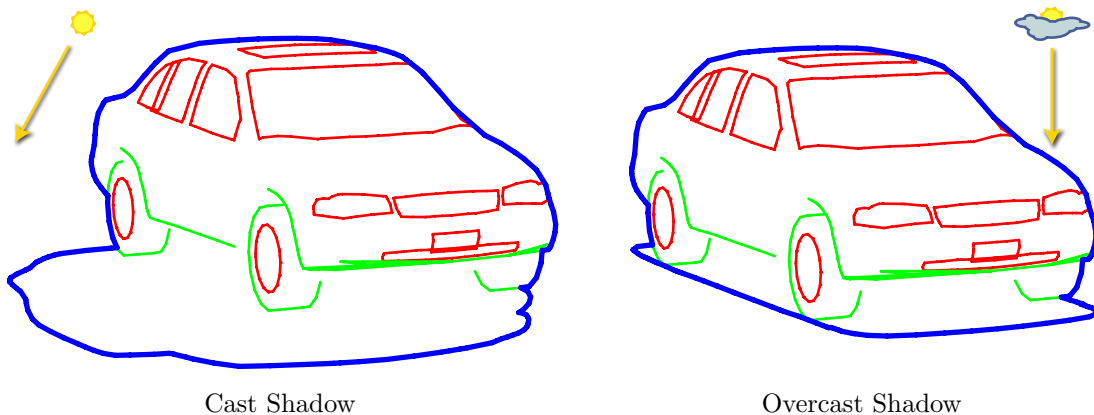


Figure 6.5: Shadow casting and silhouettes. Left, a silhouette of a vehicle and cast shadow under directly illumination. Right, an approximation to shadow formed under indirect illumination using a vertical direct lighting source.

The silhouette from the combined vehicle and shadow sources is a much better predictor of the detected foreground region, especially when the sunlight is unobstructed by clouds. On an overcast day, the sunlight is scattered by the clouds causing the light to reach the vehicle from all directions. The light source is distributed almost uniformly over the hemisphere. This does not mean that no shadow is cast. Vehicles have a small space between the chassis bottom and the road that receives little light. A shadow forms directly under the vehicle body that is strong enough to trigger foreground detection. Using the same model of shadow formation, a good approximation to this phenomenon is the vertical light direction $\mathbf{d} = [0 \ 0 \ -1]^\top$. The right side of Figure 6.5 shows this configuration. On partly cloudy days it may be necessary to switch or interpolate between these two extremes. However, this variable situation is not studied further in this thesis.

In addition to predicting the foreground region, the cast shadow contour is also added to the set of hypothesized edges used in fitting the model to images. In overcast lighting conditions, the shadow boundary appears very soft, and the edge detector generates a weak edge or no edge at all. However, in directly sunlight, the shadow edge is very crisp and is often one of the strong edges for dark colored vehicles. Using the shadow edge makes the difference between tracking success and failure in these cases.

Vehicle position

The next step in initialization is the determination of the approximate vehicle position on the ground plane. The position should be set such that the predicted silhouette aligns with the detected foreground region. Assume for now that the shape is fixed and is a reasonable approximation to the true shape. A first approximation to the region alignment is an alignment of the region centroids. Recall that the centroid of any polygon can be computed efficiently as a summation evaluated at the polygon vertices (a discreet application of Green's theorem). The centroid of the foreground is the target, and the centroid of the predicted silhouette is varied to reach the target by translating the vehicle.

To form an initial guess of the vehicle position, the foreground centroid is back-projected onto a horizontal world plane. The homography in Equation (6.16) is used to map the centroid to a height of $h = 1\text{m}$ (assumed to be approximately in the center of most vehicles). The vehicle is positioned at this back-projected location. The accuracy of this first prediction may vary significantly depending on the viewing direction, size of shadow, shape of vehicle, *etc.* However, it places the model roughly in the right part of the image. Subsequent refinements in position will not significantly change the shape of the predicted silhouette. From this starting position, the silhouette of the model and shadow are rendered, and the centroid of the silhouette is computed. The image space vector from the model silhouette centroid to the foreground region centroid has nonzero length in general. Figure 6.6 shows an example image (a), the detected foreground region and centroid (b), and the initial centroid misalignment after back-projection (c). This error in centroid alignment can be reduced by a simple iterative refinement. Mapping the error vector from the image plane to the ground plane gives an estimate of the correction needed in world coordinates. The vehicle model is translated by this correction vector, and the updated silhouette and centroid are computed. The new centroid error vector is computed and the process continues as necessary. Only a few iterations are needed in practice to reduce the error to less than a pixel. Centroid alignment within one pixel is sufficient for the next alignment stage. Figure 6.6(d) shows the model after centroid alignment.

Refined shape and position

At this point, the vehicle model is positioned in space such that its silhouette is centered over the foreground region. The alignment of region centroids does not ensure accurate 3-d position estimation if the model shape is not also reasonably accurate. This is especially true at oblique angles. For example, if the true vehicle is large but the initial shape estimate is much smaller (and therefore has a lower centroid), then the model may be translated far off toward the horizon to force the centroids to align. To correct such an error, one needs to refine both the rough shape and position such that the region boundaries themselves align, and not just the centroids.

Aligning the silhouette to the region boundary can be viewed as a simpler version of edge alignment process covered in Chapter 4. The process is simplified because there is much less ambiguity in the correspondence problem, and the desired accuracy of the solution is much less. It only needs to provide a reasonable initialization for the main model fitting algorithm. Since the regions are already

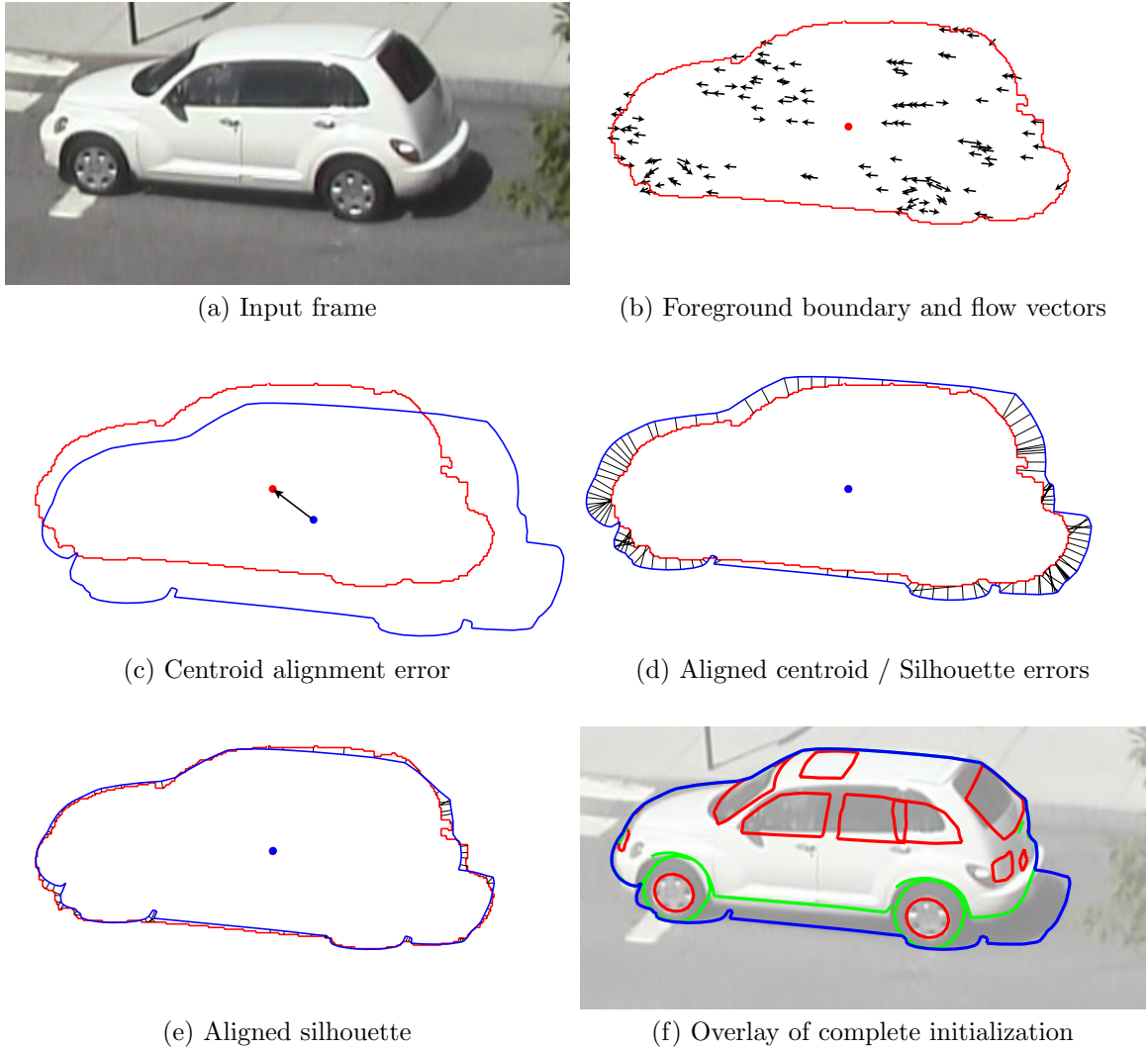


Figure 6.6: A summary of track initialization. Input frame (a) has its foreground boundary (b) detected, centroid found, and optical flow vectors associated. A test shape (c), mean minivan/SUV in this case, is oriented by flow vectors and positioned by back projection of the foreground centroid. The error in centroids is minimized by translation and the silhouette error vectors (d) are found. Silhouette error vectors are minimized (e) by least squares on translation and shape parameters. The final initialization (f) is shown.

aligned by centroid, searching for correspondences along lines normal to the silhouette curve gives good results. Furthermore, the region boundary curves are fully oriented (inside and outside directions are known), so it is easy to avoid matching to the wrong side of a polygon. Figure 6.6(d) shows an example of silhouette correspondences. For each vertex of the silhouette polygon a search in the normal direction finds the closest intersection (if one exists) with the foreground region boundary. Some mismatches can occur in regions of concavity, so searches are ignored if they intersect with another part of the silhouette before finding a match. The rest of the correspondences are given equal weight in a linear system with the same form as Equation (4.33). Recall that the silhouette polygon consists of edges that are a subset of occluding contours (possibly from more than one viewpoint), so Jacobian computation is exactly the same. Optimization is performed over the two translation parameters and the top three PCA shape parameters. Only three shape parameters are used to avoid over fitting since only silhouette information is used. Regularization with the prior shape parameters is used as before. A single least squares iteration provides a reasonable solution (as shown in Figure 6.6(e)).

In practice, boundary correspondence in the above procedure may be poor when the initial shape differs significantly from the true shape. The mean vehicle shape is not always sufficient to initialize all passenger vehicles. To solve this problem, three initial models are tested with the initialization procedure: the mean sedan shape, the mean pickup truck shape, and the mean SUV/minivan shape. The model that has the lowest final boundary alignment error is selected. Boundary alignment error is measured as the mean of the squared distances.

6.2 Tracking

Once a track is initialized, the goal of traditional (known shape) model-based tracking is to estimate the pose in the initial video frame and every frame thereafter. The simplest way to accomplish this goal is to estimate the pose in the current frame, and then use the current pose to initialize the estimation of the pose in the next frame. The pose in each subsequent frame is computed independently in the same way, relying on the previous frame only for initialization. Ferryman *et al.* [49] use exactly this method for tracking vehicle models when the shape parameters are assumed to be known. This naïve method can be quite successful when the motion between frames is small and the parameter estimation is well constrained by a single image. A tracking result computed in this way is often quite erratic because there are no explicit continuity of motion constraints between frames.

When the shape of the vehicle is also variable, the simple frame-independent tracking becomes much less attractive. Since there are many more parameters, it is much more difficult to get an accurate estimate from a single frame without a strong prior. A better solution is to use the previous estimate and its uncertainty as a prior distribution to regularize the current estimate. The previous estimate can be propagated through time by employing a motion model. Koller *et al.* [60] use this approach to estimate vehicle shape parameters while tracking pose. If all goes well, the uncertainty

in the shape parameters should decrease over time causing the shape estimate to converge.

This section adapts the tracking framework of Koller *et al.* [60] to the vehicle model and fitting procedure described in Chapters 3 and 4. Koller *et al.* use a predict and correct framework based on a generalization of the Kalman filter. The Kalman filter is described next, including the generalizations needed to adapt it to vehicle tracking. Following the Kalman filter, the vehicle motion model is discussed. Such a model of state dynamics is required in Kalman filtering. Finally, a method for detecting foreground edges is explained. Ignoring background edges helps to remove outliers in the edge matching process.

6.2.1 Kalman filter

Variations on the Kalman filter have long been used for tracking applications in various different fields. The original Kalman [54] filter describes any linear dynamical Markov chain with a Gaussian noise model. A Markov chain describes a sequence of states such that the transition to the next state depends only on the current state and is independent of the past. For example, if $\mathbf{x}_t$ is a vector describing the state at time t , then this state is determined by the linear equation ²

$$\mathbf{x}_t = \mathbf{F}_t \mathbf{x}_{t-1} + \mathbf{w}_t \quad (6.19)$$

where $\mathbf{F}_t$ is a state transition matrix and $\mathbf{w}_t$ is random noise drawn from a zero mean Gaussian distribution with covariance $\mathbf{Q}_t$. Both $\mathbf{F}_t$ and $\mathbf{Q}_t$ may be time varying but are assumed to be known. The true state, $\mathbf{x}_t$ is never directly observed. Instead, a measurement vector, $\mathbf{z}_t$, is obtained by another linear equation

$$\mathbf{z}_t = \mathbf{H}_t \mathbf{x}_t + \mathbf{v}_t \quad (6.20)$$

where $\mathbf{H}_t$ is the observation matrix and $\mathbf{v}_t$ is observation noise drawn from another zero mean Gaussian distribution with covariance $\mathbf{R}_t$. Again, $\mathbf{H}_t$ and $\mathbf{R}_t$ may vary with time but are assumed to be known. This linear model is not directly applicable to the nonlinear vehicle tracking problem. However, conceptually, one could think of $\mathbf{x}_t$ as the true vehicle shape and pose, $\mathbf{F}_t$ as the vehicle dynamics, $\mathbf{z}_t$ as an image of the vehicle, and $\mathbf{H}_t$ as the image formation process.

The goal of Kalman filtering is to continually produce estimates, $\hat{\mathbf{x}}_t$, of the underlying state as well as the uncertainty (covariance matrix Φ_t) of those estimates. Kalman filtering produces the optimal estimates under this linear model given only a stream of observations, $\mathbf{z}_t$, and the parameters of transition and observation. The Kalman filter makes these estimates in two stages. First, the **prediction** stage estimates the change in state from the system dynamics. Then, the **correction** stage uses the observation to update the predictions. Results from prediction stage are denoted $\hat{\mathbf{x}}_t^-$ and Φ_t^- , and results from the correction stage are denoted $\hat{\mathbf{x}}_t^+$ and Φ_t^+ . The superscripts indicate whether the result is from before or after correction.

²The most general form of the state transition equation also includes a term for input that controls the system. Control is irrelevant here because the vehicle tracking task is purely passive observation.

The prediction stage of the Kalman filter uses dynamics from Equation (6.19) to make an optimal prediction. The prediction equations are

$$\hat{\mathbf{x}}_t^- = \mathbf{F}_t \hat{\mathbf{x}}_{t-1}^+ \quad (6.21)$$

$$\mathbf{\Phi}_t^- = \mathbf{F}_t \mathbf{\Phi}_{t-1}^+ \mathbf{F}_t^\top + \mathbf{Q}_t. \quad (6.22)$$

In much the same way, Equation (6.20) is used to make a prediction of the observation. The predicted observation is compared to the actual observation, and the difference between them is called the Kalman innovations

$$\mathbf{y}_t = \mathbf{z}_t - \mathbf{H}_t \hat{\mathbf{x}}_t^- \quad (6.23)$$

$$\mathbf{S}_t = \mathbf{H}_t \mathbf{\Phi}_t^- \mathbf{H}_t^\top + \mathbf{R}_t. \quad (6.24)$$

These Kalman innovations are used in the correction stage to update the estimates. To make the correction, a matrix is needed that maps the innovations from the observation space back to the state space. This matrix is called the optimal Kalman gain

$$\mathbf{K}_t = \mathbf{\Phi}_t^- \mathbf{H}_t^\top \mathbf{S}_t^{-1}. \quad (6.25)$$

Using the optimal Kalman gain, the correction equations are ³

$$\hat{\mathbf{x}}_t^+ = \hat{\mathbf{x}}_t^- + \mathbf{K}_t \mathbf{y}_t \quad (6.26)$$

$$\mathbf{\Phi}_t^+ = (\mathbf{I} - \mathbf{K}_t \mathbf{H}_t) \mathbf{\Phi}_t^- \quad (6.27)$$

Extended Kalman filter

The Kalman filter provides the optimal tracker for the restricted set of problems that fall into its linear framework. Unfortunately, many interesting problems, including vehicle tracking, are nonlinear. The most common method used to circumvent this limitation is to linearize the nonlinear equations and then apply the Kalman filter. This variation of the Kalman filter for nonlinear problems is called the extended Kalman filter (EKF). Unlike the linear Kalman filter, the EKF is not guaranteed to be optimal. The nonlinearities can easily cause the estimates to diverge from the correct values. However, the EKF works well for many problems and is frequently used.

The EKF generalizes the underlying model to use arbitrary functions for state dynamics and observation. Equations (6.19) and (6.20) become

$$\mathbf{x}_t = \mathbf{f}_t(\mathbf{x}_{t-1}) + \mathbf{w}_t \quad (6.28)$$

$$\mathbf{z}_t = \mathbf{h}_t(\mathbf{x}_t) + \mathbf{v}_t. \quad (6.29)$$

where $\mathbf{f}_t(\cdot)$ and $\mathbf{h}_t(\cdot)$ are the nonlinear functions for dynamics and observation respectively. $\mathbf{w}_t$ and $\mathbf{v}_t$ are zero mean random Gaussian noise, as before. To apply the Kalman filter, $\mathbf{F}_t$ and $\mathbf{H}_t$

³Equation (6.27) is a simplification of a more general correction equation. This simplification is only valid for $\mathbf{K}_t$ as defined in Equation (6.25).

are replaced by the Jacobians of their respective functions evaluated at the current state estimate. Therefore,

$$\mathbf{F}_t = \left. \frac{\partial \mathbf{f}_t}{\partial \mathbf{x}} \right|_{\hat{\mathbf{x}}_{t-1}^+} \quad \text{and} \quad \mathbf{H}_t = \left. \frac{\partial \mathbf{h}_t}{\partial \mathbf{x}} \right|_{\hat{\mathbf{x}}_t^-}. \quad (6.30)$$

Iterated extended Kalman filter

The variation of the Kalman filter used by Koller *et al.* [60] extends even beyond the EKF. Solving a nonlinear system with a single linear approximation is always going to result in some amount of error. After a single linear solution, the corrected estimate is still not likely to agree with the observation. Typically, the solution to nonlinear equations is found by iteratively solving linear approximations until convergence. Bringing these ideas together, the iterated extended Kalman filter (IEKF) iteratively computes the observation Jacobian and solves the correction equations until convergence. Iteration in the IEKF occurs at each time step and before moving to the next time step.

Koller *et al.* explain that the iterative solution to the correction equations is equivalent to maximizing the posterior distribution of the state estimate given the observation:

$$P(\hat{\mathbf{x}}_t^+ | \mathbf{z}_t) = \frac{1}{C} \exp \left(-\frac{1}{2} (\mathbf{z}_t - \mathbf{h}_t(\hat{\mathbf{x}}_t^+))^T \mathbf{R}_t^{-1} (\mathbf{z}_t - \mathbf{h}_t(\hat{\mathbf{x}}_t^+)) \right) \exp \left(-\frac{1}{2} (\hat{\mathbf{x}}_t^+ - \hat{\mathbf{x}}_t^-)^T \boldsymbol{\Phi}_t^{-1} (\hat{\mathbf{x}}_t^+ - \hat{\mathbf{x}}_t^-) \right) \quad (6.31)$$

In the above expression, C is the distribution normalizing constant, which need not be known for a MAP estimation. The MAP estimate is equivalent to the following minimization problem:

$$\arg \min_{\hat{\mathbf{x}}_t^+} (\mathbf{z}_t - \mathbf{h}_t(\hat{\mathbf{x}}_t^+))^T \mathbf{R}_t^{-1} (\mathbf{z}_t - \mathbf{h}_t(\hat{\mathbf{x}}_t^+)) + (\hat{\mathbf{x}}_t^+ - \hat{\mathbf{x}}_t^-)^T \boldsymbol{\Phi}_t^{-1} (\hat{\mathbf{x}}_t^+ - \hat{\mathbf{x}}_t^-) \quad (6.32)$$

Using Gauss-Newton minimization as in Chapter 4, the minimization is solved iteratively. At each iteration a correction vector, $\delta \hat{\mathbf{x}}_t^+$, is computed by evaluating the Jacobian, $\mathbf{H}_t$, of $\mathbf{h}_t$ and solving a linear least squares problem. In this case, the Gauss-Newton equations are

$$(\mathbf{H}_t^T \mathbf{R}_t^{-1} \mathbf{H}_t + \boldsymbol{\Phi}_t^{-1}) \delta \hat{\mathbf{x}}_t^+ = \mathbf{H}_t^T \mathbf{R}_t^{-1} (\mathbf{z}_t - \mathbf{h}_t(\hat{\mathbf{x}}_t^+)) - \boldsymbol{\Phi}_t^{-1} (\hat{\mathbf{x}}_t^+ - \hat{\mathbf{x}}_t^-) \quad (6.33)$$

A comparison of Equation (6.33) with the earlier fitting Equation (4.40) reveals that the form is exactly the same. Only the notation differs between the IEKF and the method in Chapter 4. The various terms equate as follows

- $\mathbf{H}_t$ equates to $\mathbf{J}$, the Jacobian of the image measurement function.
- $\mathbf{R}_t^{-1}$ equates to $\mathbf{W}^2$, the inverse covariance of the image measurements.
- $\lambda^2 \boldsymbol{\Sigma}_P^{-1}$ equates to $\boldsymbol{\Phi}_t^{-1}$, the inverse covariance of the parameters.
- $\delta \hat{\mathbf{x}}_t^+$ equates to $\delta \mathbf{p}$, the unknown, incremental, parameter update vector.
- $(\mathbf{z}_t - \mathbf{h}_t(\hat{\mathbf{x}}_t^+))$ equates to $\mathbf{e}$, the residual vector (Kalman innovation) between the current prediction of image measurements and the actual image measurements

- $-(\hat{\mathbf{x}}_t^+ - \hat{\mathbf{x}}_t^-)$ equates to $-\mathbf{p}$, the vector between current value of the parameters and the mean of the prior distribution on parameters.

Notation aside, the key difference between the fitting algorithm of Chapter 4 and the IEKF corrections is the prior distribution on parameters. In Chapter 4, the prior is a Gaussian centered on the mean vehicle with a wide covariance covering the variation over all vehicles. The parameter λ is available to uniformly scale the covariance and adjust the amount of regularization. In the IEKF, the prior is also a Gaussian, but the mean and covariance are predicted from previous estimated values by the Kalman prediction equations. Propagating and evolving the prior through time allows information about past estimations to influence the current one. Ideally, a large covariance in the estimated shape parameters should decrease over time.

A more minor difference between Chapter 4 and the IEKF is that the IEKF state vector may include a few additional parameters related to the motion model. Since Kalman filtering operates on a first-order Markov chain, all information needed to predict the next state must be contained in the current state. Pose itself is not necessarily a good predictor of future pose, so the state vector is augmented with extra parameters like velocities and accelerations. These extra parameters are closely tied to the dynamics but not the image measurements. The derivative of image measurements is zero with respect to velocity, acceleration, *etc.* However, these parameters are correlated to the pose by dynamics Jacobian matrix $\mathbf{F}_t$, so the extra parameters are estimated nonetheless. The specific model of state dynamics used in this thesis is discussed in Section 6.2.2.

It no longer makes sense to use the optimal Kalman gain in the IEKF. The nonlinearity of the problem is explicitly addressed by the iterative solver for the corrected state vector. That leaves an open question about how to correct the state covariance. Koller *et al.* [59] suggests the following update using the residual covariance computed from the observation Jacobians evaluated at the point of convergence:

$$\Phi_t^+ = \left(\mathbf{H}_t^\top \mathbf{R}_t^{-1} \mathbf{H}_t + \Phi_t^{-1} \right)^{-1} \quad (6.34)$$

Regularized iterated extended Kalman filter

There are two problems with directly applying Koller’s IEKF using the fitting algorithm in Chapter 4. The first relates to the use of scale space, and the second relates to a lack of a time independent prior. Both problems are addressed by a few proposed changes to the IEKF.

It is well known that the IEKF tends to underestimate the parameter covariance, Φ_t , as a result of linearization. The solution is often to multiply by a gain matrix or increase the dynamics uncertainty, $\mathbf{Q}_t$. In the case of multi-scale fitting, the underestimation can be extreme. The final estimate at convergence always occurs with a residual scale set to $\sigma_r = 1$, and the M-estimator down-weights large errors relative to this scale. So the residual covariance, $\mathbf{H}_t^\top \mathbf{R}_t^{-1} \mathbf{H}_t$, always corresponds to errors on the sub-pixel level. Furthermore, the underestimation is directly related to the scale of the vehicle in the image. At coarse scales, when the vehicle appears small, a unit change in a parameter may correspond to an image motion less than one pixel. In this case, the covariance estimate at $\sigma_r = 1$ is appropriate. At fine scales, when the vehicle appears large, the same unit

change corresponds to much larger change in the image space. The scale change of the vehicle is not accounted for when estimating the covariance at $\sigma_r = 1$. To compensate, one can divide $\mathbf{H}_t$ by the vehicle scale. Vehicle scale is the initial σ_r estimate used to start optimization and is computed in (4.42). However, dividing by $\sigma_{r,\text{init}}$ is not quite enough. The M-estimator down-weights all errors by some amount, even those less than one. Thus, weighting with M-estimators shrinks the covariance further. In practice, dividing $\mathbf{H}_t$ by another factor of 2 provides good compensation. Revising (6.34) to account for scale invariance results in

$$\Phi_t^+ = \left(\frac{\mathbf{H}_t^\top \mathbf{R}_t^{-1} \mathbf{H}_t}{4\sigma_{r,\text{init}}^2} + \Phi_t^{-1} \right)^{-1}. \quad (6.35)$$

A second problem with the IEKF is drift in the vehicle shape prior. In multi-view fitting, the shape prior is determined by a normal distribution learned from the training data. Tracking is initialized with this shape prior, but at each subsequent frame the prior is a normal distribution centered on the solution of the previous frame. This tracking prior reduces rapid changes in shape, but over time the shape may drift away from any meaningful vehicle shape. If the shape estimate is bad at time t then the tracking prior favors the same bad shape at time $t + 1$. There is no time independent regularization to enforce prior knowledge about vehicle shape regardless of the current tracking state. To solve this problem, a new variation of the Kalman filter is proposed that uses *both* a time-dependent prior for recursive estimation and a time independent prior for regularization. The regularized iterated extended Kalman filter (RIEKF) reintroduces the learned shape prior at each frame. The learned shape prior is a zero mean normal distribution with inverse covariance $\Sigma_{\mathbf{P}}^{-1}$. Inverse covariance is as computed in Section 3.5.2 but expanded to all parameters with zeros in all non-shape rows and columns. Equation (6.33) becomes

$$(\mathbf{H}_t^\top \mathbf{R}_t^{-1} \mathbf{H}_t + \Phi_t^{-1} + \lambda^2 \Sigma_{\mathbf{P}}^{-1}) \delta \hat{\mathbf{x}}_t^+ = \mathbf{H}_t^\top \mathbf{R}_t^{-1} (\mathbf{z}_t - \mathbf{h}_t(\hat{\mathbf{x}}_t^+)) - \Phi_t^{-1} (\hat{\mathbf{x}}_t^+ - \hat{\mathbf{x}}_t^-) - \lambda^2 \Sigma_{\mathbf{P}}^{-1} \hat{\mathbf{x}}_t^+ \quad (6.36)$$

and the covariance correction equation (including scale invariance) is now

$$\Phi_t^+ = \left(\frac{\mathbf{H}_t^\top \mathbf{R}_t^{-1} \mathbf{H}_t}{4\sigma_{r,\text{init}}^2} + \Phi_t^{-1} + \lambda^2 \Sigma_{\mathbf{P}}^{-1} \right)^{-1}. \quad (6.37)$$

As before, λ is a parameter to control the amount of regularization. However, λ is set to one in all experiments. For most feasible vehicle shapes, the Φ_t^{-1} prior dominates over the $\lambda^2 \Sigma_{\mathbf{P}}^{-1}$ prior. When the shape parameters start to deviate too far from mean vehicle, the time independent prior pulls them back. To summarize, the RIEKF tracking algorithm is applied as described in Algorithm 6.1.

6.2.2 Vehicle state dynamics

The regularized iterated extended Kalman filter requires a state vector, $\mathbf{x}_t$, that contains all information about the vehicle state at time t . It also requires a motion model, a function $\mathbf{f}_t(\cdot)$ that maps the state from the previous frame at time $t - 1$ to the current frame at time t . The parameter vector,

Algorithm 6.1 Track a vehicle and estimate shape with the RIEKF

Require: $\hat{\mathbf{x}}_0^-$ from track initialization
 initialize Φ_0^{-1} with shape prior Σ_P^{-1} (with zeroed out non-shape rows and columns)
 $t \leftarrow 0$
loop
 Correct: Solve for $\hat{\mathbf{x}}_t^+$ using Algorithm 4.1 (revised according to (6.36))
 Correct: Solve for Φ_t^+ using (6.37)
 $t \leftarrow t + 1$
 Predict: Solve for $\hat{\mathbf{x}}_t^-$ using (6.21)
 Predict: Solve for Φ_t^- using (6.22)
end loop

$\mathbf{p}$, from Chapter 4 is almost adequate for the state vector requirements. It contains the parameters describing both pose and vehicle shape; however, it does not contain any parameters describing how the pose should evolve over time. Vehicles do not always move at the same rate, so the parameter vector must be augmented with rate of change parameters (*e.g.* velocities, accelerations) to form the state vector. Exact specification of these extra parameters depends on the choice of motion model.

The motion model used in this thesis is based on that of Koller *et al.* [60]. It is a simple, idealized model of vehicle dynamics using the assumption that vehicles follow circular arcs on the ground plane at a constant rate of speed. Furthermore, the vehicle is always orientated tangent to the direction of motion, and it rotates about a central axis to keep the orientation tangent. The vehicles have a fixed (but uncertain) shape, so the motion model predicts no change in the shape parameters over time. This motion model requires only two additional parameters in the state vector: a constant translational velocity, v , and a constant angular velocity, Ω . Clearly, true vehicle dynamics are much more complicated. Even if drivers did keep the steering wheel at a fixed angle and drive at a fixed speed, this model is only an approximation to the true motion. On top of that, drivers are constantly accelerating both in angle and translation. However, the circular arc motion model is able to handle these deviations by accounting for them in the prediction noise. The correction stage accounts for deviations by correcting the pose and adjusting v and Ω .

In the motion model of Koller *et al.*, the state vector is

$$\mathbf{x}_t = \begin{bmatrix} X_t & Y_t & \theta_t & v_t & \Omega_t & \mathbf{p}_{s,t}^\top \end{bmatrix}^\top \quad (6.38)$$

where X_t and Y_t are the ground position, θ_t is the orientation, v_t and Ω_t are the translational and angular velocities, and $\mathbf{p}_{s,t}$ is the vector of shape parameters. The state predictions are

$$\mathbf{x}_{t+1} = f_{t+1}(\mathbf{x}_t) = \begin{bmatrix} X_t + v_t \tau \frac{\sin(\theta_t + \Omega_t \tau) - \sin(\theta_t)}{\Omega_t \tau} \\ Y_t - v_t \tau \frac{\cos(\theta_t + \Omega_t \tau) - \cos(\theta_t)}{\Omega_t \tau} \\ \theta_t + \Omega_t \tau \\ v_t \\ \Omega_t \\ \mathbf{p}_{s,t} \end{bmatrix}, \quad (6.39)$$

where τ is the time interval between frames t and $t + 1$. For a 30 frame per second video, $\tau = \frac{1}{30}$ seconds. Refer to [59] for the derivation of (6.39) using differential equations. Computing Jacobian $\mathbf{F}_t$ of $\mathbf{f}_t$ evaluated at state $\mathbf{x}_t$ is rather straightforward and results in a sparse matrix

$$\mathbf{F}_t(\mathbf{x}_t) = \begin{bmatrix} 1 & 0 & v_t\tau C & \tau S & v_t\tau \frac{\partial S}{\partial \Omega_t} & \mathbf{0}^\top \\ 0 & 1 & v_t\tau S & -\tau C & -v_t\tau \frac{\partial C}{\partial \Omega_t} & \mathbf{0}^\top \\ 0 & 0 & 1 & 0 & \tau & \mathbf{0}^\top \\ 0 & 0 & 0 & 1 & 0 & \mathbf{0}^\top \\ 0 & 0 & 0 & 0 & 1 & \mathbf{0}^\top \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}, \quad (6.40)$$

where

$$S = \frac{\sin(\theta_t + \Omega_t\tau) - \sin(\theta_t)}{\Omega_t\tau} \quad (6.41)$$

$$C = \frac{\cos(\theta_t + \Omega_t\tau) - \cos(\theta_t)}{\Omega_t\tau} \quad (6.42)$$

$$\frac{\partial S}{\partial \Omega_t} = \frac{\cos(\theta_t + \Omega_t\tau) - S}{\Omega_t} \quad (6.43)$$

$$\frac{\partial C}{\partial \Omega_t} = \frac{-\sin(\theta_t + \Omega_t\tau) - C}{\Omega_t}. \quad (6.44)$$

Pose and shape initialization are described in Section 6.1.3, but now that there are velocity parameters in the vehicle state, velocity initialization is also needed. Velocity is initialized using the difference between pose in the first two tracked frames. In the first frame, pose is initialized as in Section 6.1.3, and velocities are assigned prior values of 5 meters per second and 0 radians per second. In the second frame, the translation rows and columns of Φ_1^- are set to zero to remove the prior on position. The model is fit in the second frame and the difference between pose in the first and second frames is used to initialize velocity. After the second frame, velocities are determined as part of the optimization and finite differencing is no longer used.

Lastly, the covariance matrix $\mathbf{Q}_t$ represents the uncertainty in the state prediction. The values in $\mathbf{Q}_t$ specify how much the dynamics model is relied on. When $\mathbf{Q}_t$ is too small the tracker may give priority to predicted motion over the observed data, and the track can diverge from the observed motion. Conversely, when $\mathbf{Q}_t$ is too large the current observation takes priority over the prediction, and the track becomes erratic and is easily lost when the vehicle is occluded. If $\mathbf{Q}_t$ is a zero matrix then the motion model is assumed to predict state dynamics perfectly. Only residual uncertainty of the previous frame estimate contributes to uncertainty in the current estimate. Clearly, the circular motion model is not perfectly accurate. Koller *et al.* [59] only specify uncertainty in the velocity parameters to account for acceleration, which is not modeled directly. However, the constant velocity assumptions are not the only deficiencies of this model. For example, vehicles do not rotate exactly about their central vertical axis. An additional small amount of variance is warranted in the pose parameters as well. To that end, $\mathbf{Q}_t$ is specified as a diagonal matrix, and the following values were found to be appropriate in practice: $\text{var}(X_t) = \text{var}(Y_t) = 0.01 \text{ m}^2$, $\text{var}(\theta_t) = 0.005 \text{ radians}^2$,

$\text{var}(v_t) = 0.1 \text{ (m/s)}^2$, $\text{var}(\Omega_t) = 0.05 \text{ (radians/s)}^2$. These values offer a good trade-off between the motion model and observed data in the steady state, when the track is well-established.

Unfortunately, the same $\mathbf{Q}_t$ is not appropriate right after initialization. State prediction is much less accurate when the parameters are not well established. The tracker needs to trust the observations much more in the first few frames of tracking. Therefore, a larger $\mathbf{Q}_t$ is needed when t is small, but this extra covariance should decay over time. It also helps to add covariance to the shape parameters right after initialization since the initialized shape estimate may be poor. The following formula is used to generate $\mathbf{Q}_t$:

$$\mathbf{Q}_t = \text{diag} \left(\begin{bmatrix} 0.01 & 0.01 & 0.005 & 0.1 & 0.05 & \mathbf{0} \end{bmatrix} + \frac{1}{t} \begin{bmatrix} 0.1 & 0.1 & 0.05 & 1.0 & 0.5 & 0.1\mathbf{\Sigma_P} \end{bmatrix} \right). \quad (6.45)$$

Note that $t = 1$ at the first prediction, and the initial prediction covariance is about ten times higher than the steady state (excepting shape). The extra added prediction covariance decays with t^{-1} .

6.2.3 Foreground edge detection

The correction stage of the RIEKF aligns the vehicle model with detected image edges as described in Chapter 4. Normally, there is no way to distinguish detected edges on the vehicle from those in the background, so all edges are considered. However, in this tracking formulation, one can do better. Recall that the camera is stationary as required for background modeling (Section 6.1.1) in track initialization. With a stationary camera, one can also detect and remove background edges, leaving only foreground edges for vehicle model fitting. There is still ambiguity in edge correspondence, but it can be greatly reduced since all background edges are outliers in the fitting process.

The detection of foreground edges is closely related to the background modeling procedure in Section 6.1.1. In fact, one way to obtain foreground edges is to use the background detection result (Figure 6.2) to mask out the detected edges (Figure 2.6) that lie in the background. This procedure extracts most foreground edges. However, segmentation can be inaccurate along the boundaries between foreground and background. Some foreground edges along these boundaries may be lost, and these edges, the occluding contours, are some of the most important for fitting.

The alternative used in this thesis is to directly use a background model of sub-pixel edges. This idea is primarily based on the work of Jain *et al.* [51] with some minor modifications. Background modeling of edges is nearly the same as background modeling of colors, but with a different input image. In color background modeling, each pixel contains a 3-d point in the color space (*e.g.* RGB). In sub-pixel edge background modeling, each pixel contains a 3-d point where the first two components represent the sub-pixel image coordinates, and the last component is the edge orientation angle. Variance in each of these components is assumed to be independent. The only difference in the Gaussian learning equations is that the distance metric in the orientation component is cyclical. The distance between two angles is smallest angle between them.

In the method of Jain *et al.* [51], the concept of *no edge present* is not modeled. Each pixel

contains a mixture of Gaussians representing different edge positions and orientations, but no component represents the probability that no edge exists. As a result, the model will quickly converge for background edges, but will not forget edges that have not been observed for a long time. An undesirable side effect of this model is that vehicles will quickly disappear into the background as soon as they come to rest (*e.g.* at a stop sign) even if many frames without the vehicle were recently observed. Even worse, some edges may disappear while the vehicle is still slowly moving. Vehicles contain many long edges that run parallel to the direction of motion. As a consequence of the aperture problem, these edges locally appear stationary over several frames and may disappear into the background. Finally, although less common, vehicles traveling along the same path at a later time may also produce similar edges to those learned from the first vehicle. Since there is no forgetting in the absence of vehicles, these foreground edges may also be lost.

The solution used in this thesis is to add a special component to the mixture to represent the absence of any edge. Only the weight of the *no edge* component in the mixture is important; the mean and variance are meaningless. However, a special 3-d data point is assigned to avoid rewriting the background modeling code for this special case. All pixels in which no edge is detected are replaced by the same special edge point. This point is located outside the image boundaries to prevent interference with any real edges. Arbitrarily, the *no edge* point is assigned to $(-100, -100, 0^\circ)$. Real image coordinates are always positive. This special point always matches the *no edge* component of the mixture because the point is always exactly the same. A small lower bound is set on the variance to prevent it from vanishing due to the observation of identical points.

A result of the foreground edge detection is shown in Figure 6.7. Compared to the full set of detected edges in Figure 2.6, almost all vehicle edges are retained while most background edges are removed. There are still many scattered false foreground detections in the background. Currently, no attempt is made to post process these results. The reduction of outlier edges is already significant enough to benefit the vehicle fitting algorithm.



Figure 6.7: Detected foreground edges. Compare to Figure 2.6.

Chapter 7

Experiments: Convergence of Shape and Pose

This chapter presents an experimental evaluation of the vehicle model in Chapter 3 and fitting procedure in Chapter 4. The purpose of these experiments is to compare the proposed vehicle model to previous deformable vehicle models by examining the results of fitting to real images. Specifically, these experiments compare the accuracy of the recovered 3-d shape and the size of the parameter space region within which valid convergence occurs. Ideally, one would like to see the model converge to the same, very accurate vehicle shape starting from a wide variety of initial parameters. These experiments analyze the effects of a large number of system variables including:

- choice of model (and mesh resolution)
- image resolution
- class of vehicle
- color of vehicle
- number of PCA shape parameters
- number of pose parameters (*i.e.* use of a ground plane constraint)
- number of views in multi-view fitting

The next section sets up the experiments by describing the image data set, the vehicles models, and the fitting error measure. This is followed by an extensive study on the impact of varying the number of PCA shape parameters. Lastly, experiments illustrate the range of convergence in parameter space.

7.1 Experimental setup

The experiments in this chapter compare the recovered 3-d vehicle model shape to a ground truth shape. Ground truth for shape is provided by the E3D CAD models (Figure 5.1). Three things are needed to carry out these experiments: a collection of images showing vehicles for which the ground truth shape is available, a set of deformable vehicle models to compare, and a metric for measuring the error of the resulting shape in 3-d. Each of these requirements is discussed in turn.

7.1.1 Multi-view image data set

The devised experiments require images from multiple viewpoints of known vehicles for which a ground truth 3-d CAD model is available. Vehicles parked on the street were inspected to find those with available ground truth. Images of these vehicles were taken from several viewpoints with a hand-held digital camera. Each image was taken with the same lens configuration. The resulting multi-view data set contains images of vehicles from several different classes and with several different colors. The vehicle identifications are summarized in Table 7.1, and Figure 7.1 shows thumbnail images of the collected data.

Vehicle	E3D Class	Color	CAD #	# of Views
Dodge Stratus	4-door sedan	white	51	7
Toyota 4Runner	SUV	white	36	7
Dodge Caravan	minivan	white	3	6
Toyota Tundra	pickup	black	13	6
Nissan Maxima GLE	4-door sedan	gray	68	5
Chevrolet S10 Blazer	SUV	tan	27	5
Volvo V70XC	wagon	blue	26	5
Volkswagen Beetle	2-door sedan	yellow	24	4
Volkswagen Beetle	2-door sedan	black	24	4

Table 7.1: Vehicle identities in the multi-view image data set. The CAD number identifies the E3D CAD model used for ground truth. CAD numbers index into Figure 5.1 and Table 5.1.

Each collected image is smoothed and recursively down-sampled by a factor of two to generate images at various resolutions. The resulting image resolutions are 1536×1152 , 768×576 , 384×288 , 192×144 , and 96×72 . Hereafter, these resolutions are referred to as image scales 1-5, respectively. Note that scale 1 is close to 1920×1080 , the full 1080p HD video resolution. However, the vehicles in these test images take up a large portion of the image. The resolution on the vehicles is even greater than would be found in most real world vehicle surveillance applications using 1080p video. At scale 1, the resolution on the vehicle ranges between about 0.25 cm per pixel and 1 cm per pixel. At scale 5, the resolution scales accordingly to between 4 and 16 cm per pixel.

Camera calibration

Multiple view parameter estimation, as described in Section 4.3.5, requires calibrated cameras. Estimation of camera calibration parameters is completed in two steps. First, a common set of

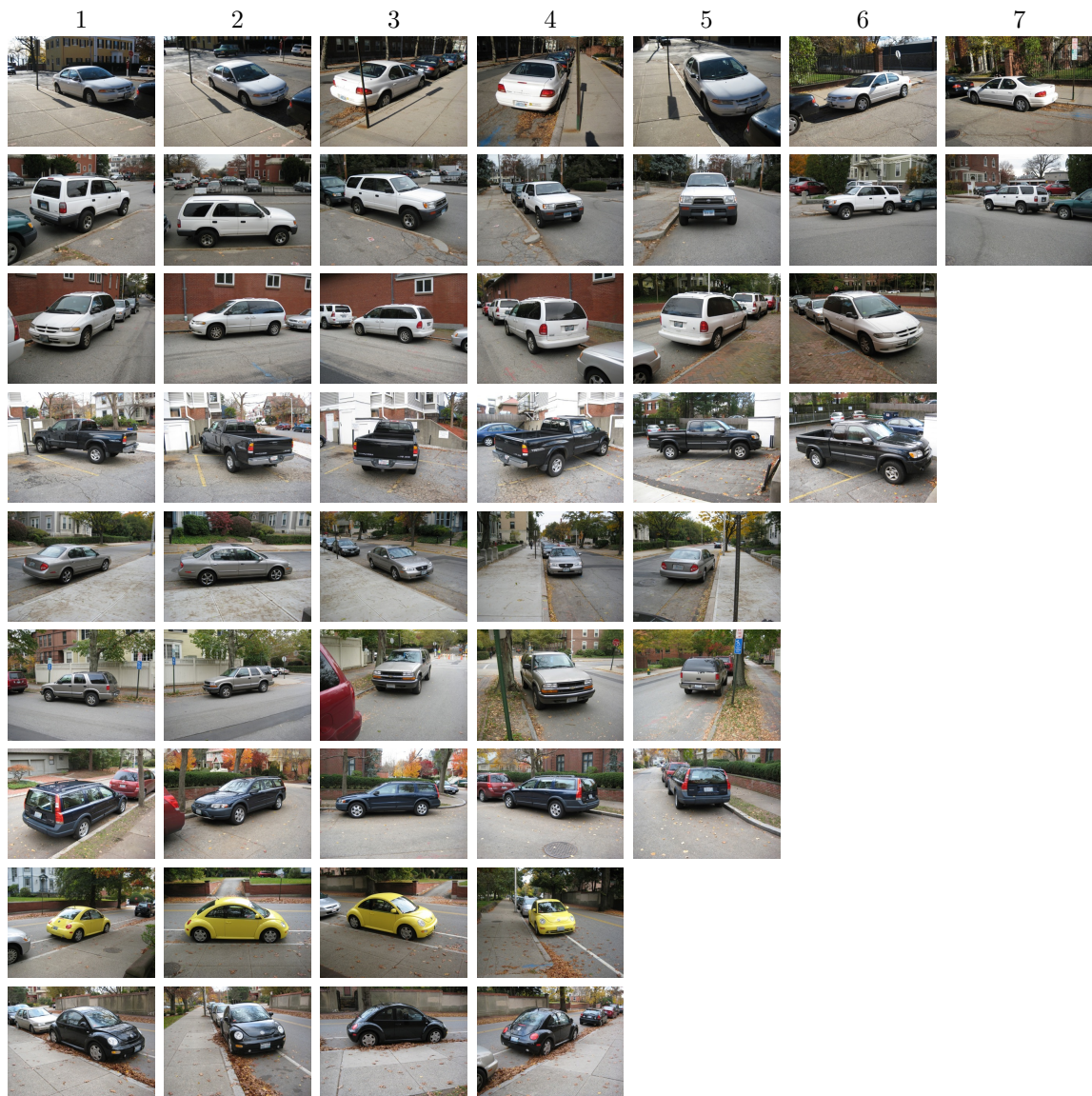


Figure 7.1: Images used in multi-view fitting experiments. From top to bottom these vehicles are: Dodge Stratus, Toyota 4Runner, Dodge Caravan, Toyota Tundra, Nissan Maxima, Chevrolet S10 Blazer, Volvo V70XC, VW Beetle (yellow), and VW Beetle (black).

intrinsic parameters are determined for all images. Second, extrinsic camera calibration is computed for each image using the ground truth vehicle model as a calibration object.

A common intrinsic calibration is possible because the same camera and lens configuration was used when taking all images. Intrinsic calibration is computed using a checkerboard pattern and the Camera Calibration Toolbox for Matlab [11] by Jean-Yves Bouguet. In general, there are five intrinsic degrees of freedom in the perspective projection matrix. These degrees of freedom are represented by the focal length, aspect ratio, skew, and 2-d principal point. The Calibration Toolbox estimates all of these parameters and also coefficients of radial distortion. The digital camera used to take the images was manufactured to have an aspect ratio of one, a skew of zero, and a principal point in the center of the image. The only variable left to the photographer is the focal length. Calibration results verify that the intrinsic parameters are indeed very close to the manufacturer's specifications and that radial distortion is negligible. As a result, aspect ratio, skew, and principal point are set to the ideal values, and focal length is estimated by itself assuming no radial distortion. Focal length units are relative to the resolution of image, but field of view angles are resolution independent. The estimated field of view is about 53° horizontally and 41° vertically.

With an intrinsically calibrated camera, the remaining extrinsic parameters are estimated using the vehicle itself as a calibration object. There are six remaining degrees of freedom in the projection matrix. These degrees of freedom cover the pose of the camera relative to the world coordinates. Unlike the intrinsic parameters, the pose is different for each camera. If the ground truth 3-d vehicle model is placed in the world coordinate system, and all cameras are fully calibrated, then the model projected by each camera should align perfectly with each image. In other words, the extrinsic camera calibration problem is identical to the vehicle pose estimation problem. Extrinsic camera calibration can be solved by a special case of Algorithm 4.1 where the vehicle shape is known and fixed. This simply amounts to removing all the PCA shape parameters and corresponding Jacobian columns. To use Algorithm 4.1 directly, the ground truth CAD model is replaced by the shrink-wrapped vehicle model (third mesh resolution) and parts as described in Chapter 5. Note that this model is still highly accurate. There is no projection onto a shape subspace; every vertex lies on the CAD model surface. For each camera, the pose is manually initialized and then refined by running a pose only version of Algorithm 4.1. For some viewing directions, like frontal views, the pose is poorly constrained. The optimization results are supervised, and any poor solution is manually adjusted.

7.1.2 3-d deformable vehicle models

The experiments aim to compare the performance of the proposed deformable vehicle model to prior deformable vehicle models in the literature. The most common deformable vehicle is an irregular dodecahedron, such as that used by Koller *et al.* [59]. The only other deformable 3-d model of passenger vehicles known to the author is that of Ferryman *et al.* [30]. Ferryman's model is almost identical to the dodecahedral model, except that it also contains a semi-decagonal arch for each wheel well. Experiments evaluate the dodecahedral model and Ferryman's model in comparison with the proposed, more detailed model. Furthermore, the experiments consider the detailed model

at each of the first three mesh subdivision resolutions. In each case, one subdivision of part polygons is used. Figure 7.2 shows the five vehicle models considered. The models are henceforth referred to as Dodecahedral, Ferryman, Detailed 1, Detailed 2, and Detailed 3, where Detailed 1 is the coarsest and Detailed 3 is the finest mesh. Both Dodecahedral and Ferryman use all mesh edges to predict image edges. The detailed meshes use a combination of occluding contours and part boundaries as described in Section 3.4.

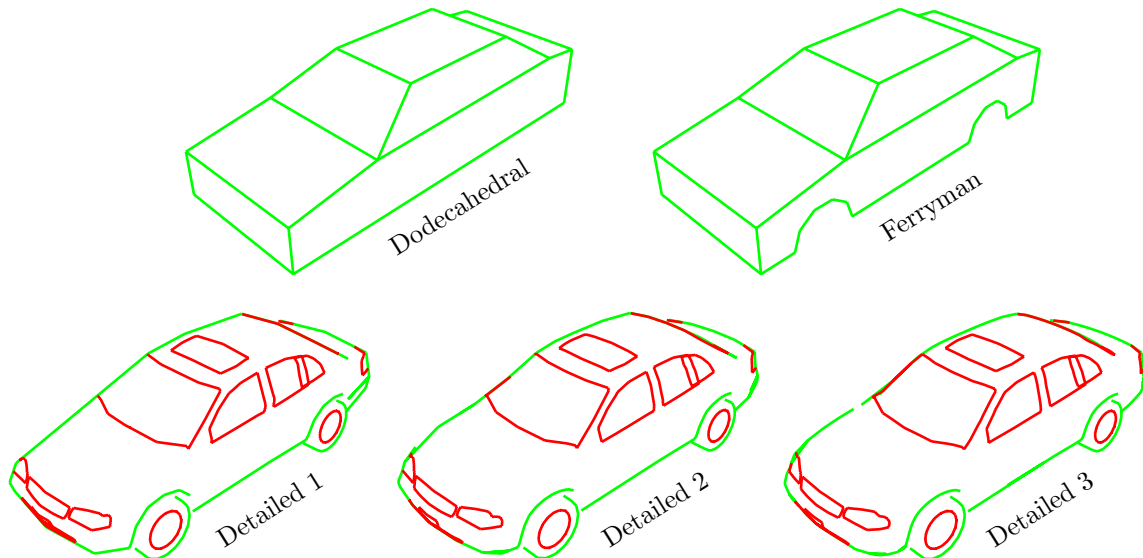


Figure 7.2: 3-d deformable models compared in experiments.

For comparison on equal footing, the shape deformation parameters for all models are learned from the same database of E3D CAD models. The learning procedure has already been explained in Chapter 5 for the detailed models. In the case of the Dodecahedral and Ferryman models, the vertices are positioned according to the 30 semantic vehicle measurements listed in Table 5.2. Constructing the shape from these measurements results in a Dodecahedral model with 13 degrees of freedom. Koller’s dodecahedron had only 12 parameters. The extra degree of freedom is found in the line segment below side windows. Koller *et al.* restrict this line segment to be parallel to the X -axis. Here, this line segment may be angled in the XZ -plane. The Ferryman model gains an additional three degrees of freedom corresponding to the front axle X position, rear axle X position, and wheel well radius.

7.1.3 3-d shape error metric

After a 3-d model is fit to a set of images, one would like a measure of the quality of that fit. An obvious measure that is readily available is the magnitude of the image fitting residual. That is, $\mathbf{e}^\top \mathbf{W}^2 \mathbf{e}$ evaluated at the final iteration of Algorithm 4.1. However, this measure doesn’t directly relate to the quality of the fit because the true edge correspondences are never known. An edge

distance can go zero if it matches nothing within the search radius or if it perfectly aligns with any edge, even an incorrect one. Instead, a good measure of the quality of a fit should examine the resulting 3-d shape.

Another way of measuring the quality of a fit is to compare the resulting shape to the true shape using a distance metric in the shape space. This approach is also flawed. It does not take into account any error in the estimated pose. A good measure of quality should account for error in both shape and pose.

A third approach compares the fit model in 3-d world coordinates to the same model with ground truth shape and pose parameters. Each 3-d mesh vertex, with position determined by estimates of shape and pose, has a corresponding vertex, with position determined by the ground truth shape and pose. The average distance between all these corresponding vertices is a measure of fit quality. This metric may be reasonable in some cases, but it still has a few flaws. First, it requires the ground truth to use the same 3-d model for point-wise correspondence. So ground truth varies between experiments with different models. Second, it will not be very accurate for Dodecahedral or Ferryman because the ground truth vertices are not generally on the true vehicle surface. Third, one can usually move the points tangent to the surface and still have an equally good approximation of the 3-d shape. In other words, there exists a 3-d analog to the aperture problem that can artificially degrade the quality score if using this error metric.

The actual error metric used in this thesis is a variation on the previous metric that does not assume predetermined correspondence. Each point on the estimated mesh surface (as determined by shape and pose) is matched to the *closest* point on the ground truth mesh. The ground truth mesh need not be the same model as the estimated mesh. Error is computed as the root mean square (RMS) distance between all point matches. That is,

$$E = \sqrt{\frac{1}{M} \sum_{i=1}^M (\mathbf{v}_i - \text{cp}(\mathbf{v}_i))^{\top} (\mathbf{v}_i - \text{cp}(\mathbf{v}_i))}, \quad (7.1)$$

where M is the number of points, $\mathbf{v}_i$, on the estimated mesh surface, and $\text{cp}(\cdot)$ is a function that returns the closest point on the ground truth surface. Finding the closest point on a complex mesh can be an expensive operation. A naïve implementation requires a distance computation between the point and each mesh face. However, a significant speedup is obtained by using a kd-tree data structure.

Ideally, the number of measurement points should be large, and the points should be distributed over the entire vehicle surface. The vertices of Detailed 3, satisfy this condition. There are over a 1000 vertices and they are distributed fairly uniformly across the surface as a result of quadrilateral subdivision (Section 3.1.1). Other models have far fewer vertices. The Dodecahedral model has only 16, and these corner points tend to be the farthest from the surface. To correct for this imbalance, quadrilateral subdivision is applied recursively to all models after fitting until the number of vertices is greater than 1000. Recall that subdivision does not alter the shape of the mesh, but it does add more vertices on the planar mesh faces. The total set of subdivided vertices (at least 1000) is used

in the computation of RMS error. Only the vehicle body surfaces are compared by the error metric. Wheels and undercarriage are ignored because not all models have these surfaces.

One could use the E3D CAD models as the ground truth in the RMS closest point error metric. However, as discussed in Section 5.1, many CAD models contain interior structures as well as the outer surface. If the pose is incorrect, the estimated model could easily match to many nearby interior surfaces. The RMS error should measure only error in recovering the outer surface shape. Thus, the CAD models are again replaced by a shrink-wrapped mesh (using Detailed 3) for ground truth.

7.2 Number of PCA shape parameters

Vehicle shape parameter space is determined by PCA on the set of training examples. Section 3.5.1 mentions that the maximum number of degrees of freedom, m , can be reduced to any $n \leq m$ by retaining only the top n principal components. However, no criteria have been established for choosing n . A small n is favored because it reduces computation, may provide a more meaningful fit in noisy or poorly conditioned data, and can more easily overcome the influence of outliers. However, fewer model parameters also results in a more constrained and less expressive model. A larger number of parameters can always provide a better fit to the data, but extra parameters may result in over fitting depending on the data. In short, a reasonable value of n depends on the data and should be determined pragmatically. This dependence is on both the 3-d training data and 2-d image data.

Figure 7.3 plots the percentage of total variation¹ in the learned Detailed 3 model as the dimension of the PCA subspace is increased. It is clear that most of the variation is packed into a small number of top principal components. The top five components cover 94.3% of the variation, and the top ten cover 97.8%. The “knee” of the curve is right around five components. Each new component makes a significant contribution before the fifth component. After the fifth, their contribution is much less. This first plot gives some insight, but does not yet take into account how many parameters are useful in recovering shape from real image data.

7.2.1 Fitting with good initialization

The first set of experiments examine how accuracy of recovering 3-d shape varies with the number of PCA parameters. These experiments are not concerned with effects of suboptimal initialization. Initialization is addressed in later experiments. Instead, these experiments are concerned with the accuracy achieved in the best of circumstances when very good initialization is provided. For all experiments in the rest of this section, the initial pose is the ground truth pose, and the initial shape uses the ground truth shape parameters with all parameters above the current n truncated. Such an initialization does not guarantee the minimal 3-d error, but it should be very close. With this

¹Percent of total variation is the sum of variances over the first n principal components compared to the sum of variances over all components.

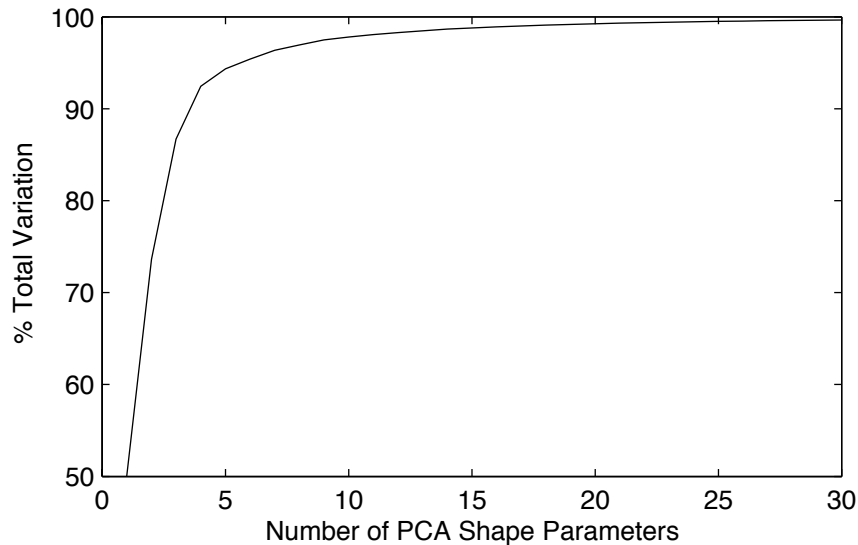


Figure 7.3: Percent total variation with increasing number of PCA parameters for the Detailed 3 vehicle model.

initialization, the fitting algorithm is run for each of the nine vehicles in Figure 7.1 using all available images. The experiments are repeated for all combinations of the five deformable models, the five image scales, and the number of PCA shape parameters from 0 to 15. Along with the n PCA shape parameters, all six pose parameters are estimated. Since good initialization is assumed, the initial uncertainties in the parameters (See Section 4.3.4.) are also set low. The initial shape uncertainties are each set to 0.125 times the PCA singular values, the translations uncertainties are set to 0.125 meters, and the orientation uncertainties are set to 0.0625 radians (about 3.5°). Fitting accuracy in all experiments is computed using the RMS error described in Section 7.1.3.

Figure 7.4 shows two plots of the error as a function of number of PCA parameters for each of the different vehicle image sets. Both plots use image scale 1. The left plot uses the Ferryman model and the right uses Detailed 2. Similar trends are found for all other vehicle models and scales. As expected, most of the error is reduced by the fourth or fifth PCA component. However, the error reduction depends on the vehicle model. Some vehicles, like the Dodge Stratus and Nissan Maxima, are very close to the mean shape of Detailed 2. With Detailed 2, these vehicles start with, and maintain, low error. Their error only decreases slightly with the first few shape parameters. Other vehicles, like the Volkswagen Beetle, are very unusual in shape for any of the models. They start with high error and may require several more parameters to reduce the error to a comparable level with other vehicles. Their error also tends to fluctuate more than other vehicles. Boxy vehicles, like the Toyota Tundra and Toyota 4Runner, seem to consistently maintain lower error using the Ferryman model.

Figure 7.5 shows two more plots that compare the choice of deformable 3-d model. In these plots, the error has been averaged over all vehicle image sets to produce a single curve for each

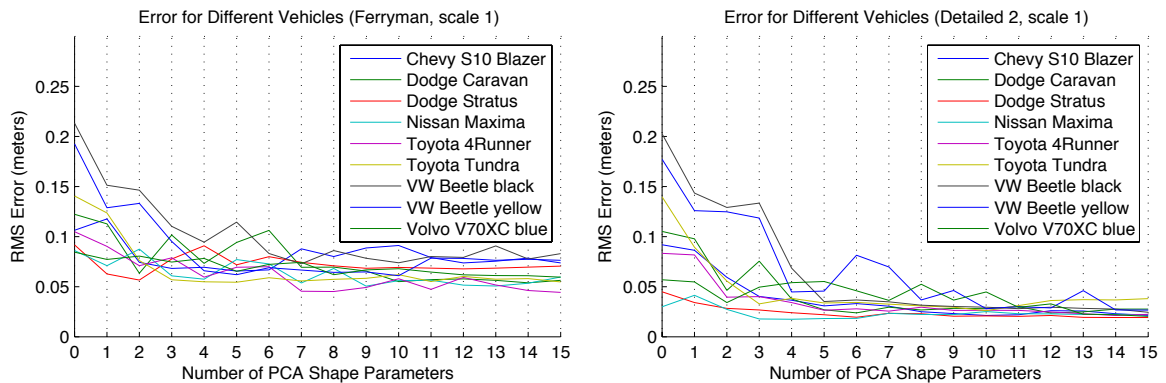


Figure 7.4: Error versus number of PCA parameters for different vehicles. Initialization uses the ground truth with low uncertainty in parameters.

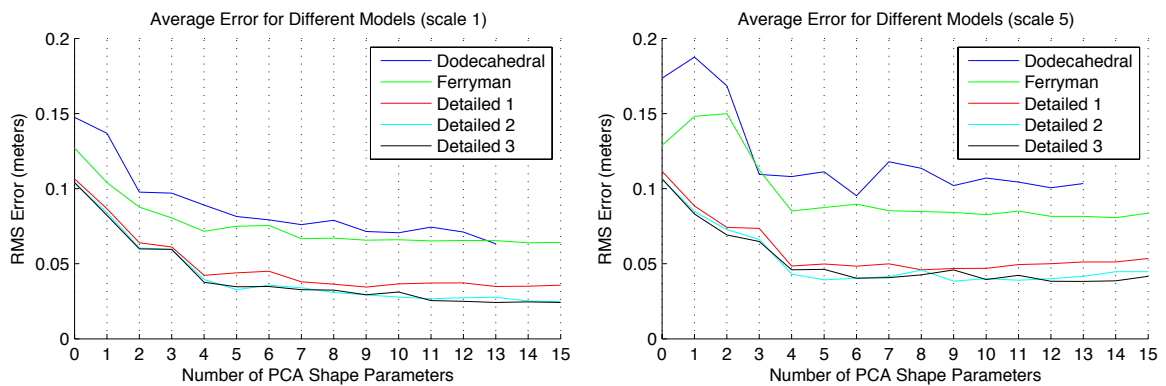


Figure 7.5: Average error versus number of PCA parameters for different models. Initialization uses the ground truth with low uncertainty in parameters.

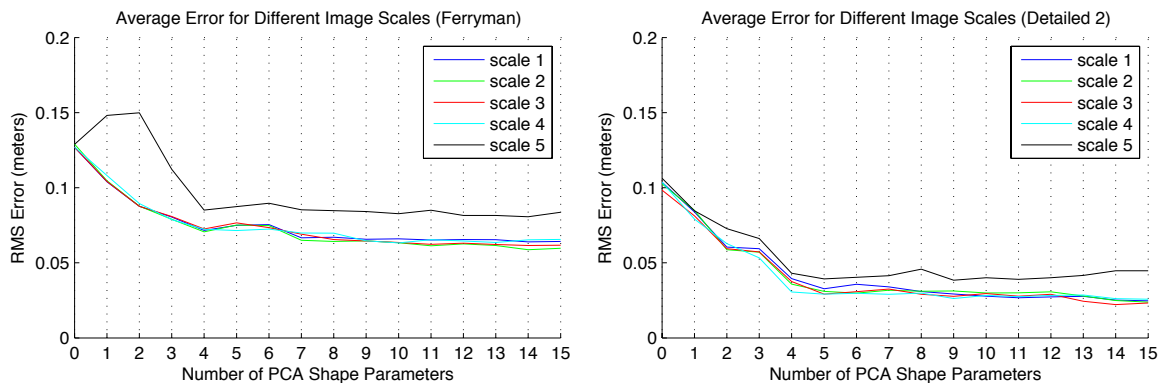


Figure 7.6: Average error versus number of PCA parameters for different image scales. Initialization uses the ground truth with low uncertainty in parameters.

deformable model. The left plot shows the results at scale 1, the right plot at scale 5. Intermediate scales show the same general trends. The Dodecahedral model is comparable to the Ferryman model with Ferryman doing slightly better. All the Detailed models are significantly more accurate than Dodecahedral and Ferryman, as expected. The Detailed models are all very close, with Detailed 3 better than Detailed 2, which is better than Detailed 1. However Detailed 2 is extremely close to Detailed 3 and sometimes is more accurate.

Figure 7.6 shows another pair of plots, this time varying the scale of the images. Again, the error is averaged over all vehicle image sets to produce one curve per scale. The left plot uses the Ferryman model, and the right plot uses Detailed 2. Accuracy is very similar for all scales except for scale 5. This indicates that the fitting algorithm is largely scale independent until reaching a breakdown point at a minimum image resolution. Degradation in performance at scale 5 provides evidence that scale 5 is near this breakdown point.

In summary, with good initialization, a few important trends are noticeable. First, on the average, the error tends to decrease rapidly with the first few principal components and then levels off in dimensions greater than four or five. Second, the Detailed models consistently have lower RMS error than both the Dodecahedral and Ferryman model. On average, the error is around 0.05 meters less for the Detailed models. Third, at finer scales (*i.e.* finer than scale 5), the Detailed 2 and Detailed 3 models do continue to decrease in error on average, although very slightly, as the number of shape parameters increases beyond five. The remaining models (and all models at scale 5) do not exhibit this gradual decrease, but they do not exhibit an increase either.

Judging from these experiments alone, there seems to be no harm in using more shape parameters, even as many as 15. In these experiments there is no apparent penalty for using more parameters, and in some cases there may even be a gain. However, these experiments only consider the best case scenario of very accurate initialization. Usually, the initialization is less accurate and has more uncertainty.

7.2.2 Fitting with poor initialization

In realistic situations, the parameter initialization is not nearly as good as that used in the previous section. Therefore, the same set of experiments from the previous section are repeated with poor and uncertain initialization. Instead of starting near the ground truth shape, the vehicle is initialized to the mean vehicle shape. The pose is still initialized to the ground truth for now, but initial pose is varied later in Section 7.3. In practice, an initialization routine, such as that in Section 6.1, can do much better than the mean shape. Some vehicles are significantly different in shape from the mean vehicle, and optimization starting with the mean shape can be quite challenging even when the pose is accurate. To properly compensate for the poor initialization, the initial uncertainty in all pose parameters are increase by a factor of four. The initial shape uncertainties are set to 0.5 times the PCA singular values, the translation uncertainties are set to 0.5 meters, and the orientation uncertainties are set to 0.25 radians (about 14°). Figures 7.7, 7.8, and 7.9 show the results of the exact same experiments as Section 7.2.1 but with mean shape initialization and higher uncertainty.

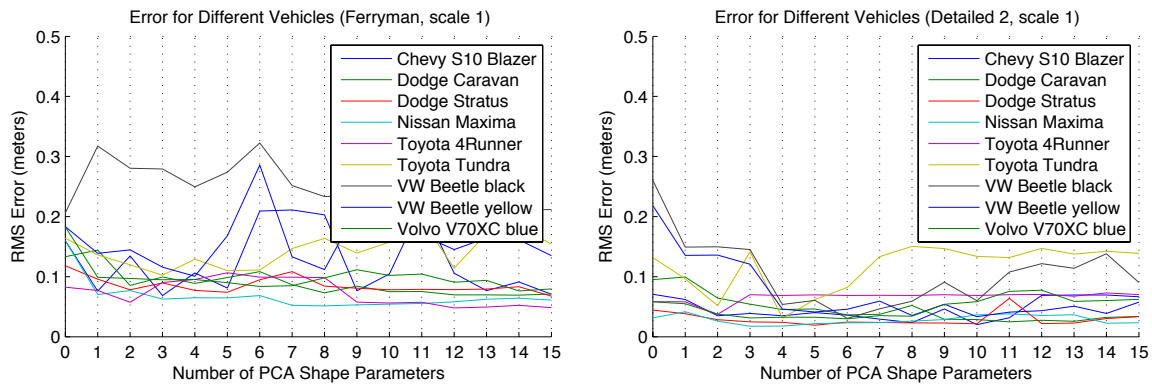


Figure 7.7: Error versus number of PCA parameters for different vehicles. Initialization uses the mean vehicle with high uncertainty in parameters.

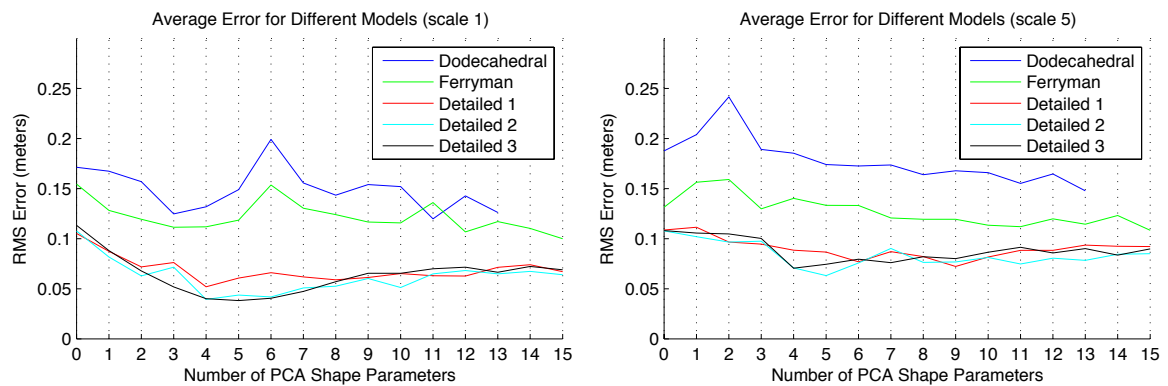


Figure 7.8: Average error versus number of PCA parameters for different models. Initialization uses the mean vehicle with high uncertainty in parameters.

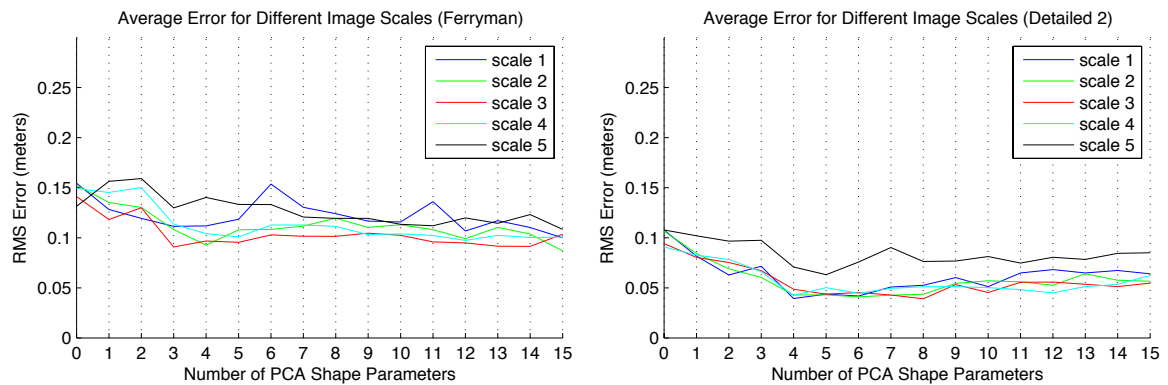


Figure 7.9: Average error versus number of PCA parameters for different image scales. Initialization uses the mean vehicle with high uncertainty in parameters.

Some of the trends in these experimental results are the same as the previous experiments. Others are quite different. In Figure 7.7, most vehicles still tend to decrease in error when adding the first few parameters. Yet, the errors are not quite as low as before, and some vehicles now increase in error after adding more shape parameters. Note that the scale of the vertical axes has changed in these new plots to account for increased error. In the case of the Ferryman model, some vehicles have very large error at every number of shape parameters. There does not seem to be any number of parameters for the Ferryman model that works well with all vehicles. In Figure 7.8, the five vehicle models still show the same ranking in accuracy with all three Detailed models significantly more accurate than Dodecahedral and Ferryman. Although, there is less difference in accuracy between the three Detailed models than before. In Figure 7.9, scales 2-5 are still very similar for the Detailed 2 model, and scale 1 is still noticeably less accurate. However, with the Ferryman model, the scales are a bit more separated now. Scale 3 is most accurate, and scales 1 and 5 tend to be the least accurate. The Ferryman model seems to be less scale independent and seems to favor scale 3.

Perhaps the most important trend that has changed with poor initialization is the optimal number of shape parameters. With very good initialization, it is observed that the fit continues to improve in accuracy with increasing shape parameters when using fine enough scales and the Detailed 2 or 3 model. With poor initialization, this trend no longer occurs. The error now decreases at first and then increases again as more parameters are added. This new trend is most clear with the Detailed 3 model at scale 1 shown in the left of Figure 7.8. The curve reaches a minimum at five shape parameters. Fewer shape parameter result in a model that is not expressive enough, so the error is higher. Additional shape parameters provide too many degrees of freedom and the model is more easily trapped by outliers in a local minimum. Five shape parameters seems to provided a good tradeoff between these two extremes. In the rest of the experiments in this thesis, the initialization is assumed to be uncertain, and five shape parameters are used.

To obtain a visual sense of the fitting accuracy, Figure 7.10 shows the converged models rendered on top of the images. One image of each test vehicle is shown, and the results of convergence are drawn for the Detailed 3 model at scale 1 with 5 shape parameters. Most of the vehicle shapes are recovered accurately and the model aligns well with the images. In all cases, the edges from headlights, taillights, and license plates have the worst alignment with the true underlying features. The trouble with these features arises from the larger variance in shape and position of these parts. When truncating to five shape parameters, expressiveness in these details is the first to go. Furthermore, the small size and uncertain location of these parts makes it difficult to latch onto the correct inlier curves.

There are also two interesting failure modes evident in Figure 7.10. First consider the pickup (Toyota Tundra) and wagon (Volvo V70XC). In both of these vehicles, the rear window is not properly aligned. Part of the problem is that these are both dark colored vehicles. The edges due to background reflection are often much stronger than the boundary edges between the windows and body. Part of the problem is also underrepresentation in the training database. Station wagons and pickup truck have the least number of models in the training data (two and six respectively), so the

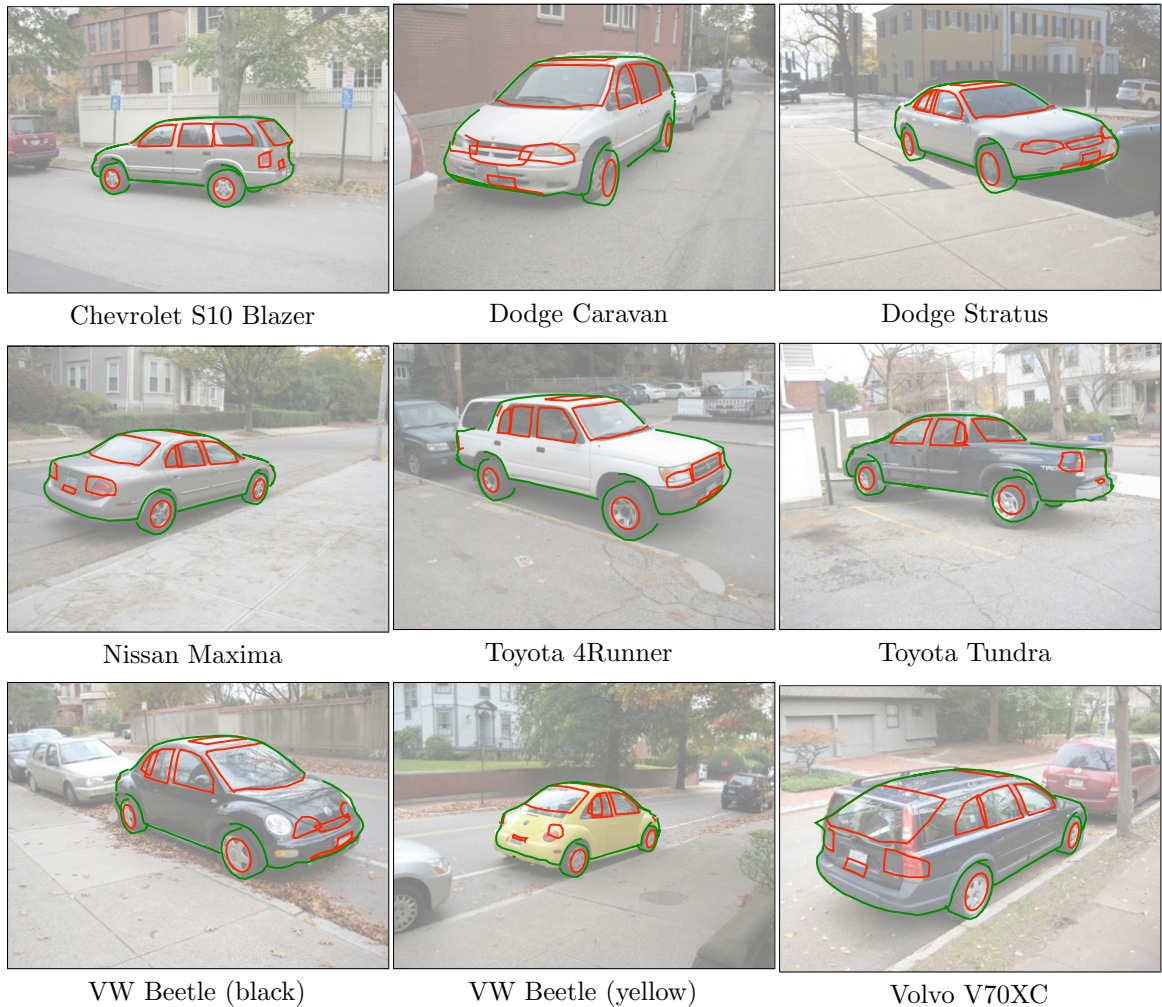


Figure 7.10: Overlay of the fitting results using Detailed 3 at scale 1 with 5 shape parameters. Initialization uses the mean shape and correct pose.

model is biased toward the shapes of the other vehicle classes, especially sedans.

An even more interesting failure mode is apparent in the Toyota 4Runner. The estimated shape is incorrect, but it still aligns very well with image data. Ignoring the rear side windows, this SUV looks very much like a pickup truck (albeit with a very short bed). This local minimum is reached before the true minimum when starting with the mean shape. Unlike the local minima of the pickup and wagon, this local minimum is also very stable. The error for the 4Runner shown in Figure 7.7 increases after two shape parameters and remains incredibly constant as the number of parameters is increased to 15. That is, this same local minimum is found for all numbers of parameters from 3 to 15 when starting with mean shape. Finding the correct shape is not a lost cause. Other initial conditions lead to the correct shape. However, this example illustrates that, in some cases, solutions can be provided an excellent fit to the data but still recover the wrong shape.

For comparison, Figure 7.11 shows the overlays with the Ferryman model results. The conditions

of the experiments are the same except for the model. In general, these shape estimates look reasonable. Although, some vehicles, like the VW Beetles and Toyota Tundra, are much worse than others. It is hard to visually judge the accuracy of these estimates since many of the Ferryman edges do not correspond to any real vehicle features. The wheel well curves are one feature that does have a well defined meaning. In many cases, only one of the two wheel well curves is correctly aligned. The Toyota 4Runner suffers from the same local minima as before.

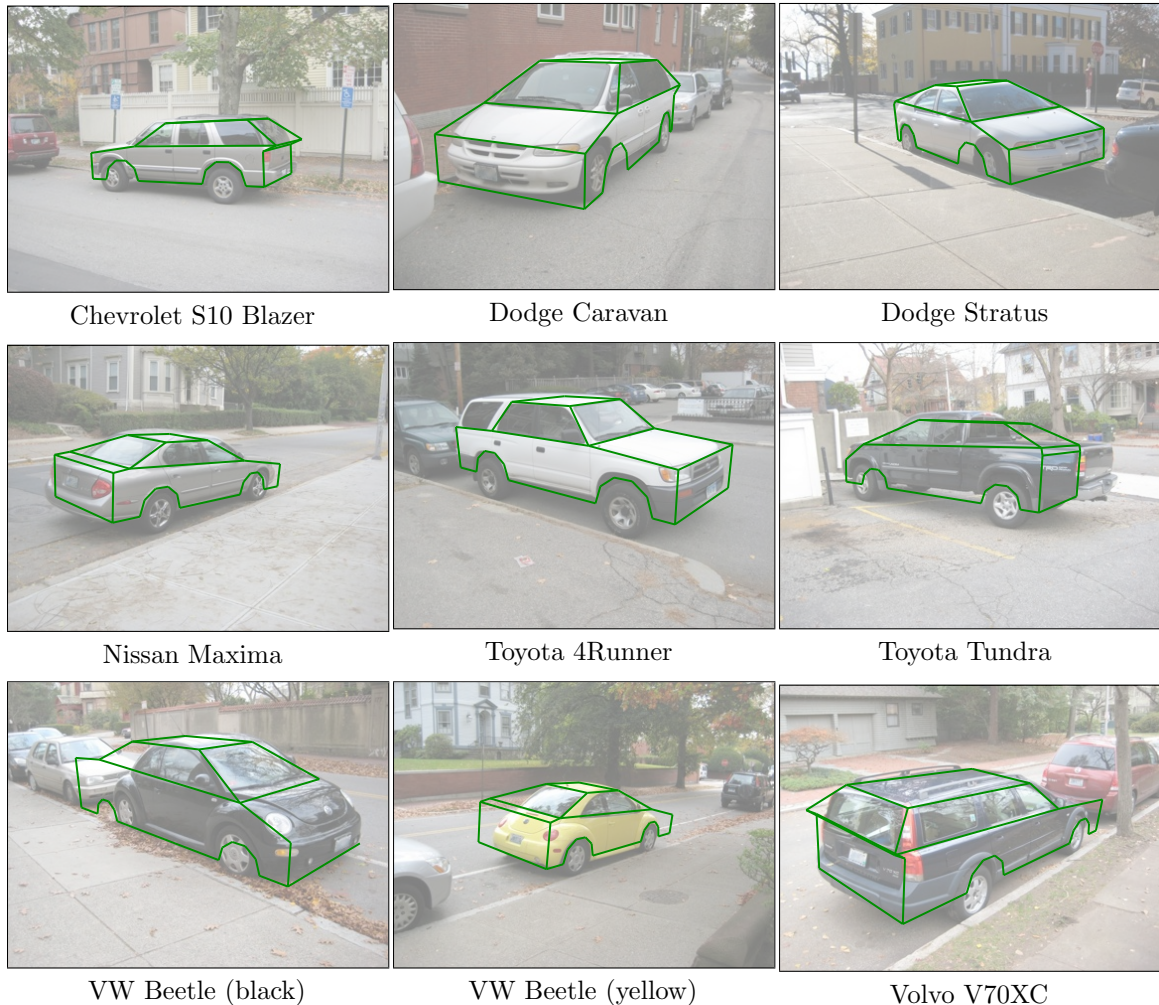


Figure 7.11: Overlay of the fitting results using Ferryman at scale 1 with 5 shape parameters. Initialization uses the mean shape and correct pose.

7.3 Region of convergence in pose

In the previous section, the experiments started with fixed initial conditions. Pose was initialized to the ground truth, and shape was initialized either to the ground truth or mean shape. These experiments showed that the fitting algorithm usually converges to an accurate shape with good

pose initialization and five shape parameters. An obvious question is how far away from the true pose can the initialization be such that the algorithm still converges to the same accurate result? Clearly the initialization cannot be arbitrary. Algorithm 4.1 is a local optimizer that relies on good initialization. At least some of the initial edge correspondences need to be correct, or there is no hope of success. Therefore, the next experiments study the effect of initial pose on convergence.

The following experiments all use five shape parameters and are initialized with the mean vehicle shape. Initial pose is varied and the convergence results are compared. One would hope that the fitting algorithm finds the same accurate solution for a wide range of initial conditions. In other words, within some region around the true pose, the algorithm should converge to the correct shape with high probability. A simple experiment, helps to illustrate this region of convergence. The fitting algorithm is run as in Section 7.2.2, but instead of ground truth pose, the initial pose is sampled on a regular grid. For each initial pose, the resulting RMS error of the fit is determined. Figure 7.12 plots the negative RMS error as a function of the initial pose sampled on a grid in the subspace of X translation and Z axis rotation. All other pose parameters are kept at the ground truth.

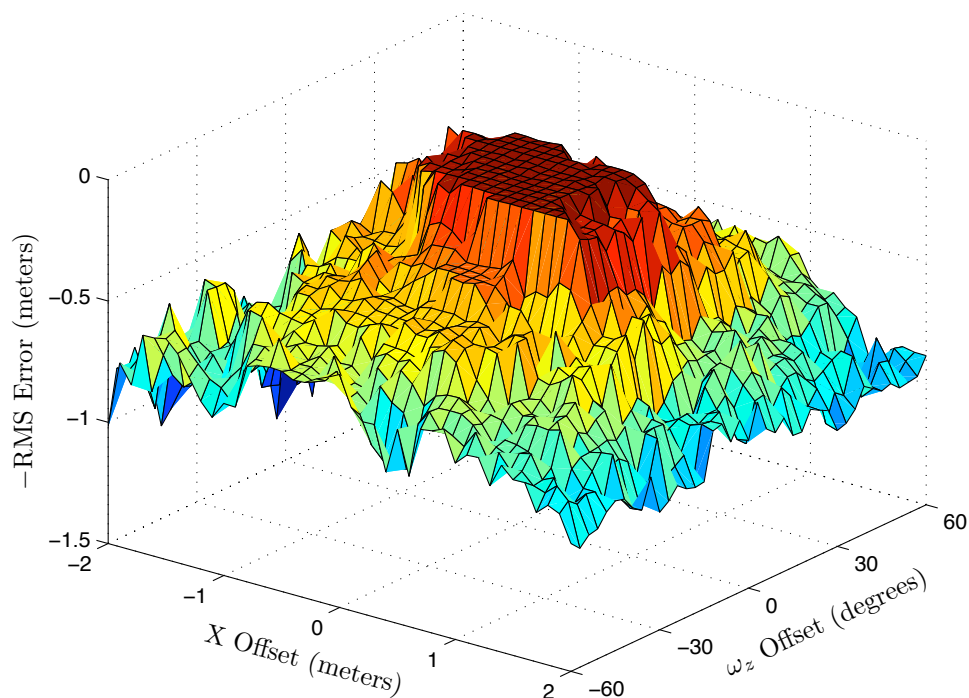


Figure 7.12: An inverted plot of RMS error at convergence as a function of two initial pose parameters: X translation and Z -axis rotation. This plot is for the Detailed 3 model with Dodge Stratus images at scale 3 using mean shape initialization. The error forms a clear plateau in the neighborhood of the true pose.

A clear plateau forms around the minimum error in Figure 7.12. Thus, all initial poses on the plateau converge to the same low error solution. The plateau is a cross-section of the region of convergence in the six dimensional pose space. In this case, the region spans about ± 0.5 meters in

X and $\pm 15^\circ$ in ω_z . Some form of plateau appears for all vehicle image sets, image scales, and vehicle models. However, the size of the plateau varies in each case, and in the case of the Dodecahedral and Ferryman models, the plateau is less clearly defined. Figure 7.13 shows the same experiment as in Figure 7.12 but with Detailed 3 replaced by Ferryman. The plateau is similar in shape but significantly more noisy and with less clearly defined boundaries. A bumpy plateau indicates that there is no single optimal solution for these boxy vehicle models. Rather, there are several local minima that each represent the vehicle reasonably well. Many of the boxy model's predicted edges do not correspond to any real vehicle features. It is not surprising that these line segments would latch onto different vehicle edges when subjected to different initialization conditions.

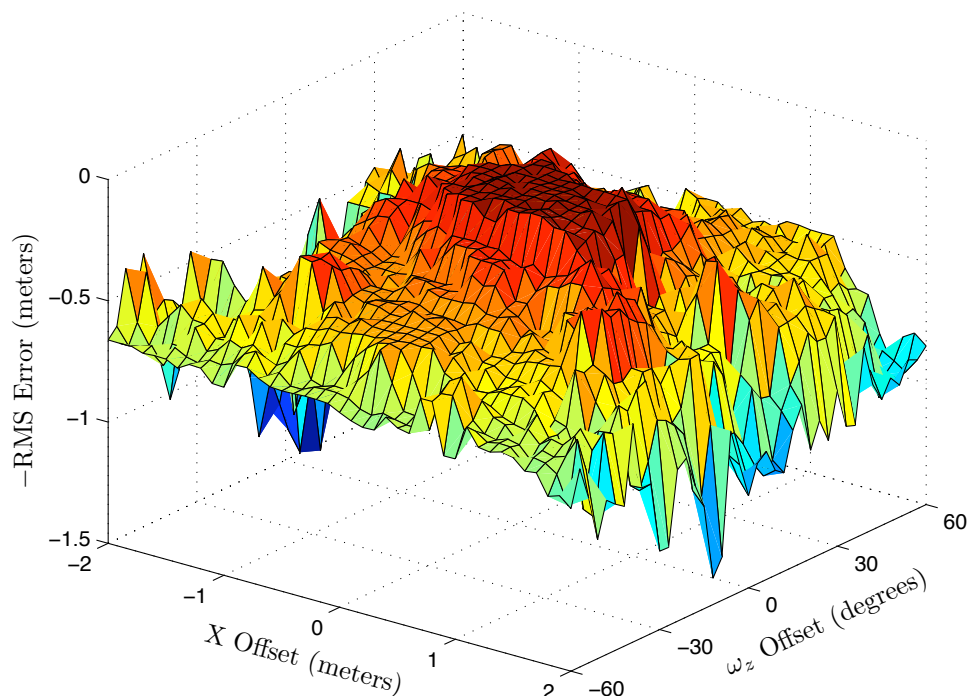


Figure 7.13: The same experiment as in Figure 7.12 (Dodge Stratus, scale 3, mean initial shape) except with the Ferryman model instead of Detailed 3. The plateau for Ferryman is bumpy and less clearly defined.

Outside of the plateau (*i.e.* convergence region), the algorithm converges to incorrect local minima. Error in regions off of the plateau are quite noisy. A noisy surface indicates that these incorrect local minima are not very stable and are very sensitive to initial conditions. However, it is possible obtain a plateau at an incorrect solution. The Toyota 4Runner in Figure 7.10 illustrates an incorrect solution that is well supported by the image data. Figure 7.14 show the inverted error plot corresponding to that experiment. Two distinct plateaus appear in the plot. The lower plateau covers the origin and represents the stable but invalid solution found in Figure 7.10. The higher plateau (lower error) represents the correct solution. Notice that the correct solution is obtained when the vehicle is translated backward around 0.2 to 0.5 meters. When moved backward, the rear window of

the mean vehicle is initially placed closer to the rear window in the images. This initial placement causes the rear window to lock onto the correct set of curves.

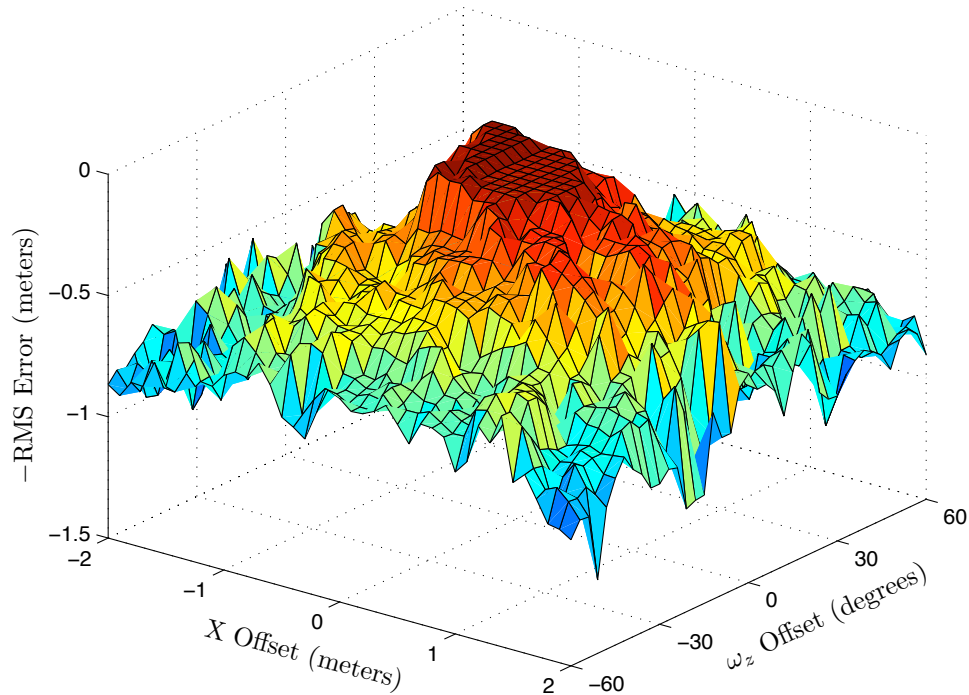


Figure 7.14: An inverted plot of RMS error plot is for the Detailed 3 model with Toyota 4Runner images at scale 1 using mean shape initialization. Two distinct plateaus indicate two stable solutions. The higher plateau (lower error) is the correct one.

Extending the convergence region analysis to the full six dimensional pose space is not trivial. In the 2-d cross-sections, the grids contained 41 samples on each axis. Therefore, $41^2 = 1681$ convergence experiments are required to generate a plot for each combination of image set, image scale, and vehicle model. Sampling on the same regular grid in the 6-d hyper-volume requires $41^6 \approx 4.75$ billion experiments, an intractable number. Furthermore, there is no meaningful way to visualize a plateau in a 6-d space. The data need to be reduced to a lower dimensional quantity for interpretation. Measuring the hyper-volume of the 6-d plateau is one option, but the space has mixed units of distance and angle. The hyper-volume would be biased toward angles or distances depending on the choice of units. However, half of the dimensions have distance units (meters) and half have angle units (degrees). Thus, the translation and orientation components of pose can each be reduced to a meaningful 1-d measurement. For translation, the Euclidean distance measures distance between two points. When comparing to the origin, Euclidean distance is just the magnitude of the translation vector. For orientation, a minimum angle of rotation exists between any two orientations. When comparing to the identity, the minimum angle is just the magnitude of the rotation vector, as explained in Section 4.1.

A Monte Carlo approach (*i.e.* random sampling) is used to produce a meaningful measure of

the region of convergence using a tractable amount processing. Initial poses are selected at random in the space of all poses. Translation samples are drawn independently from orientation samples. Each type of sample is drawn from a distribution that is uniform in direction over the sphere and uniform in distance from the origin over a fixed interval $[0, D]$. For translation, $D = 1$ meter. For orientation, $D = 30^\circ$. These ranges are chosen to be about twice as large as the largest plateau widths observed in the 2-d cross-sections (*i.e.* ± 0.5 meters and $\pm 15^\circ$ as in Figure 7.12). A joint 2-d histogram is created with distance on one axis and angle on the other. Each axis is divided into 10 bins for 100 2-d bins in total. Random pose samples are drawn from the translation and orientation distributions until each bin contains 10 random samples. Hence, the total number of samples is 1000. Optimization is run from each initial pose using five shape parameters set to the mean shape. A threshold on the final RMS error determines if the optimization succeeded. The number of successful convergences in each bin is plotted in the 2-d histogram. An example is shown in Figure 7.15. The 3-d view of the joint histogram is easiest to interpret, but the 2-d view of the histogram is far more compact and allows easier comparison of multiple histograms.

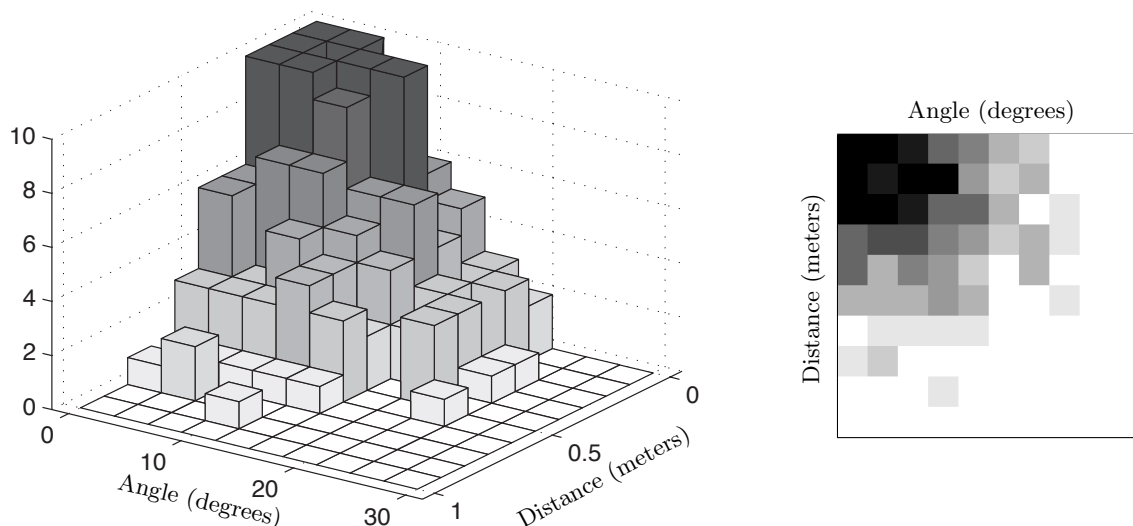


Figure 7.15: 3-d and 2-d views of a joint histogram over distance and angle. The left shows the histogram as a 3-d bar plot where the height is the number of correct convergences. The right shows the same data in image form with black representing correct convergence of all 10 samples and white representing correct convergence of none. This plot corresponds to the images, model, and scale of Figure 7.12

To measure the extent of the plateau, the threshold for convergence should only accept solutions within a very small tolerance of the correct solution. However, that threshold will vary between image sets, image scales, and vehicle models since the height of the plateau varies. Experiments in Section 7.2.1 use ground truth initialization and low initial uncertainty to determine the best fit for each model. The final RMS error values for each of these best case experiments determines the residual error, E_R , after correct convergence (*i.e.* the height of each plateau). Therefore, correct

convergence is determined to be any solution with an error less than $E_R + \epsilon$, where ϵ is a small tolerance set to 0.01 meters. Figure 7.16 shows the 225 resulting histograms for all combinations of image sets, image scales, and vehicle models. The same 1000 random poses are used in each configuration of images, scale, and model.

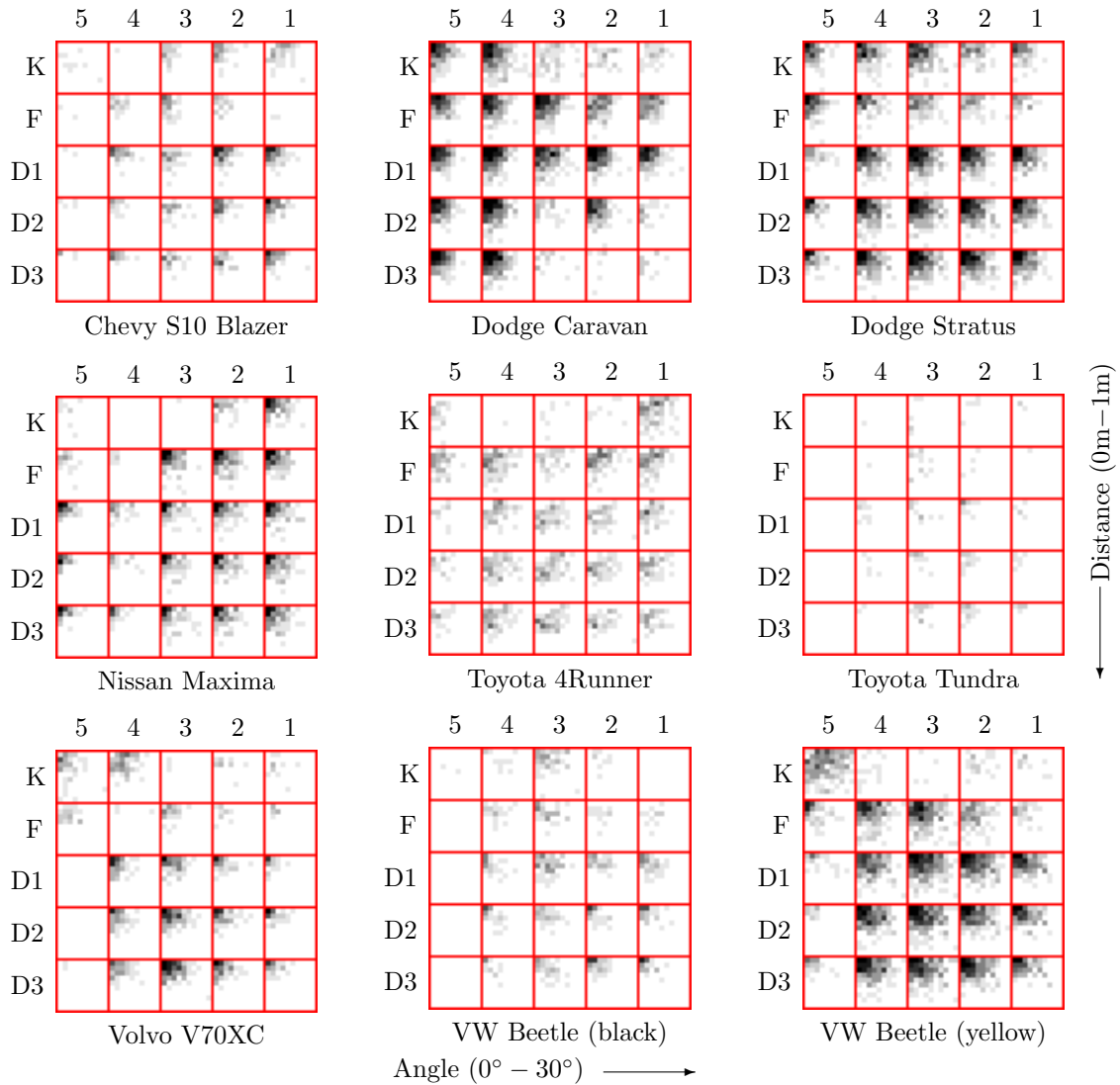


Figure 7.16: Joint histograms with an error threshold of 0.01 meters above the relative best fit. Histograms are grouped into 5×5 grids by image set. In each grid, image resolution increases left to right (scales 5 – 1), and model complexity increases top to bottom (Dodecahedral (K), Ferryman (F), Detailed 1 (D1), Detailed 2 (D2), Detailed 3 (D3)).

Dark bins in the upper left of each histogram indicate stable convergence around the true pose (with mean initial shape). Many dark bins spreading down and to the right indicate the convergence is stable over a wide range of initial poses. A number of trends are apparent in Figure 7.16. First, some vehicles are significantly more stable than others over all models and image scales. The least

stable convergence occurs with the black vehicles: the Toyota Tundra and Volkswagen Beetle. Note that the yellow Volkswagen Beetle does significantly better than the black one even though all other variables besides color are nearly the same. In fact, the vehicles with the widest stable convergence regions are all brightly colored: the yellow Volkswagen Beetle, the white Dodge Stratus, and the white Dodge Caravan. Brightly colored vehicles are more stable because their bright color contrasts sharply with windows and background that are often very dark. This contrast leads to detection of feature edges that are stronger than most of the background clutter. Thus, the outlier edges have less relative weight in the optimization. Conversely, with dark colored vehicles, the feature edges are often very weak. Reflections of the background often generate the strongest edges. As a result, the outliers may overpower the modeled edges making fitting a model to dark vehicles very difficult.

The Toyota 4Runner is also a white vehicle, but does not show the same characteristics of the other brightly colored vehicles. Using most models and scales, correct convergence occurs at many points away from the true pose, but the upper left bins are all white instead of black. This, of course, is a result of the two stable solutions for this vehicle discussed previously. These experiments measure convergence to the lowest error solution. Initialization with the true pose and mean shape results causes convergence to the less optimal solution. However, the correct solution is found with the mean shape for several poses away from the true pose.

Another common trend is that the convergence regions tend to be larger for the proposed models (Detailed 1, 2, and 3) than the Dodecahedral or Ferryman models, especially at finer scales. All three Detailed models tend to have convergence regions similar to each other. Convergence regions of the Detailed models are quite similar over the scales 1-4, with noticeably worse results at scale 5, the most coarse image resolution.

The Dodge Caravan histograms seem to disagree with many of the trends previously mentioned. Convergence is actually more stable at the lower scales and with simpler models. Camera calibration of this particular image set was also found to have worse alignment with the ground truth vehicle shape when compared to other image sets. From Figure 7.10 one can see that the Caravan alignment is not as good as with the other vehicles. Similar poor alignment also occurs in the ground truth calibration, not just this fitting result. After further research, it was discovered that the 1996 Dodge Caravan was manufactured with two different body sizes with different wheelbases and different overall lengths. It is likely that the ground truth CAD model does not exactly match the vehicle in these images. Therefore, the best case fit that defines the convergence threshold is itself not very accurate and not very stable over different scales and models.

Figure 7.16 only shows how each configuration converges to what is assumed to be the minimal error for that configuration. Since the error threshold is different for each configuration, the histograms do not directly compare how well each configuration convergences to a common low error solution. In other words, one configuration (*e.g.* Caravan, scale 5, Dodecahedral) may converge very well to a common solution over many initial poses, but that solution itself may have high error. Another configuration (*e.g.* Caravan, scale 1, Detailed 3) may not converge very well to the lowest error solution, but may converge well to as slightly suboptimal solution that still has much less

error than the first configuration. An absolute error threshold is needed to compare the error of the convergence results between different histograms. Figure 7.17 shows the exact same experiments as Figure 7.16, but with an absolute threshold on error, $E_A = 0.1$ meters.

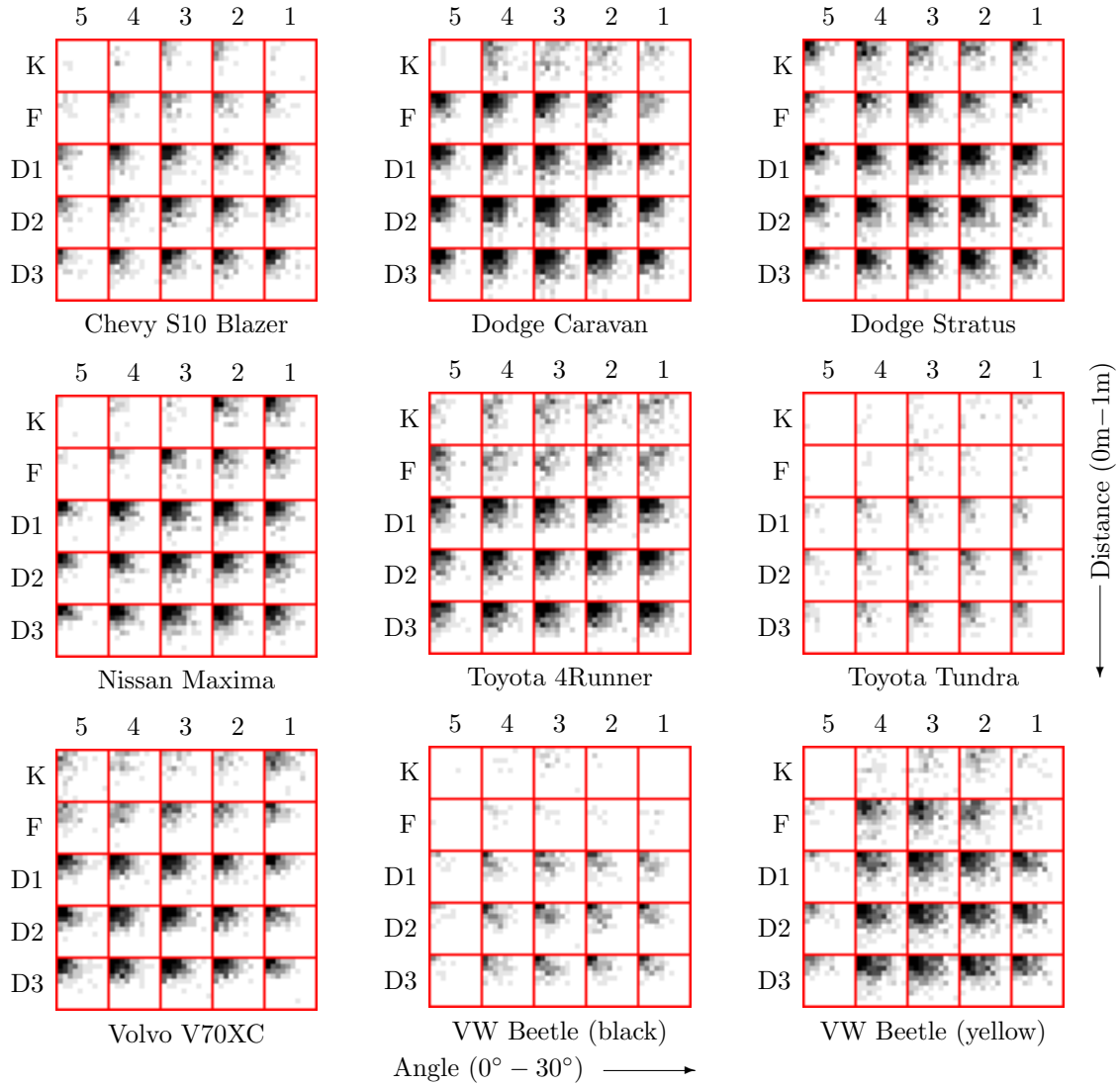


Figure 7.17: Joint histograms with an absolute error threshold of 0.1 meters.

An absolute error threshold no longer measures the convergence to the minimum error plateau, but instead measures convergence below a global tolerance for error. The two closely spaced plateaus of the Toyota 4Runner are not distinguished in these histograms. However, the histograms do show more consistently that the Detailed models converge to a lower error solution over a wider range of initial poses than Dodecahedral and Ferryman. Figure 7.16 also demonstrates more clearly that wheel wells in the Ferryman model lead to better convergence than the Dodecahedral model which is the same but without wheel wells.

One variable that has not yet been examined is the number of pose parameters. A ground plane constraint is often available to reduced the pose degrees of freedom from six to three. To test the effect on convergence, a new set of 1000 ground plane constrained poses is randomly selected, and optimization is applied using only the remaining three pose variables. These constrained random poses are sampled from analogous uniform distributions. That is, Z -axis rotation is drawn from a uniform distribution in the range $[-30^\circ, 30^\circ]$, and (X, Y) position is drawn from a uniform distribution in polar coordinates, (r, θ) such that r is uniform on $[0, 1]$ meters and θ is uniform on $[0^\circ, 360^\circ]$. Figure 7.18 shows the ground plane constrained histograms with absolute threshold.

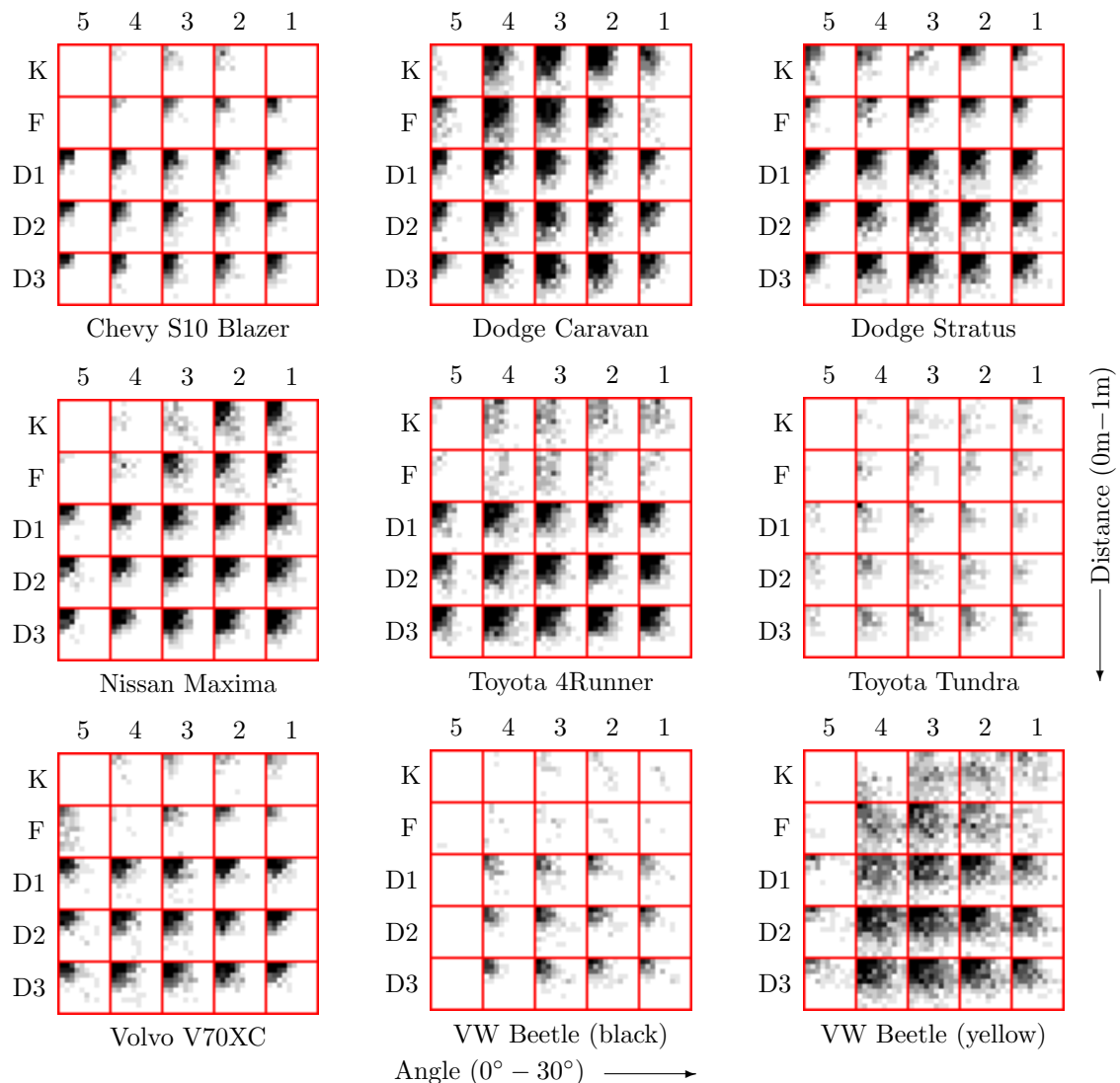


Figure 7.18: Joint histograms with a ground plane constraint and an absolute error threshold of 0.1 meters.

Convergence regions with a ground plane constraint are very similar to those with all six pose parameters. However, the number of successful converged results increases in almost all cases. The

shape of the histograms do not change very much, but the bins appear darker on average. This means that a ground plane constraint does not generally make the convergence region larger, but does make it more stable. In other words, it does not enable convergence from more distant poses; instead, it makes convergence more likely for nearby poses.

A final variable to consider is the number of views used in multi-view fitting. In previous experiments, all available views are used. The number of views differ between images sets, ranging from four to seven, but this does not illustrate how the algorithm responds to varying the number of views within an image set. Especially interesting is how the algorithm performs as the number of views reduces to one. With only one view, there is no longer any depth information provided by multi-view stereo constraints. The algorithm relies totally on prior knowledge to recover 3-d shape from only one view.

To examine the effects of number of views, one image set with the most views is selected, and images are removed one at a time. The Dodge Stratus image set is used for this experiment. The seven views are removed in an order such that similar views are removed first leaving a wide baseline between remaining views. The images are reordered in the sequence (2, 3, 6, 7, 5, 3, 1), where the numbers refer to the ordering in Figure 7.1. Images are removed one at a time from the end of the sequence (image #1 first, image #2 last), and optimization is run on each subset. Figure 7.19 shows the resulting histograms. Notice that the convergence regions are quite similar for two or more views. The Dodecahedral model starts to deteriorate first, especially at fine scales, when the number of views drops to five or four views. The Ferryman model does better with noticeable deterioration starting around three or two views. The Detailed models perform best with stable convergence regions all the way down to two views. However, with only one view, the convergence region suffers for all models and all scales.

The monocular case is by far the most challenging, but also very important in real applications. Single camera surveillance is much more common in most real world applications than multi-view surveillance. The video tracking experiments in the next chapter consider only the monocular case. While it is true that multiple video frames over time do provide a form of multi-view stereo, online tracking algorithms are recursive and must start by fitting to a single frame. Figure 7.19 demonstrates that accurate, single frame fitting is possible from mean shape parameters, but the region convergence may be quite small. However, as long as the resulting shape and pose are moderately accurate, the tracking will succeed and shape will be refined over time. Initialization with more accurate shape parameters also helps in the effort.

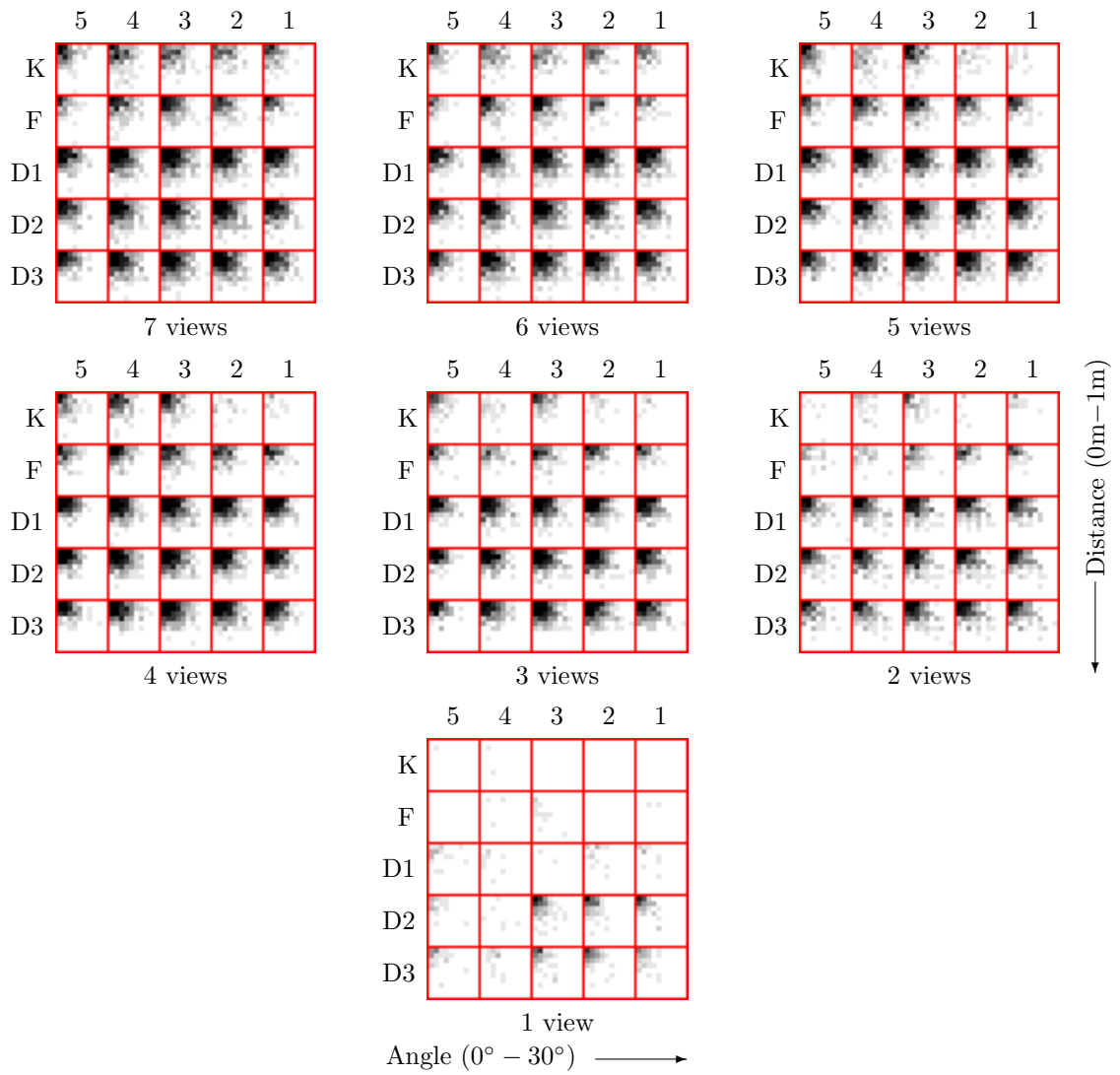


Figure 7.19: Joint histograms with varying number of views and an absolute threshold of 0.1 meters. These result are produced from the Dodge Stratus image set.

Chapter 8

Experiments: Tracking in Video

This chapter presents an experimental evaluation of the tracking algorithm described in Chapter 6. Unlike the multi-view experiments in Chapter 7, the vehicles are observed from a single viewpoint and generally do not correspond to a CAD model in the shape database. As previously shown, single view shape estimation is far more challenging than simultaneous estimation from two or more views. Furthermore, the reconstruction now must be generalized to previously unknown vehicle shapes. Thus, these experiments consider much more challenging circumstances.

In the computer vision literature, only Koller *et al.* [60] attempt to simultaneously track vehicles in real video data while estimating their 3-d shape. Moreover, Koller *et al.* present only three successful examples. All three examples are bright white, boxy shaped sedans and hatchbacks that are well represented by the Dodecahedral model. On the other hand, many papers [1, 39, 46, 49, 59, 62, 72, 80, 84] have been devoted to vehicle tracking using a rigid 3-d vehicle model. Therefore, tracking is first evaluated using fixed shape parameters for each of the deformable models. The fixed shape tracking results are then compared to tracking while estimating the shape.

8.1 Video data set

Three different high resolution video sequences were collected under different viewing conditions. Each video sequence contains a number of vehicles to track. Figure 8.1 shows an image from each sequence. All sequences observe intersections since the most interesting behavior occurs in these areas. Vehicles accelerate at intersections both in position and angle. They also may rapidly change direction instead of traveling along a straight path. All video sequences were captured with the same camera and have the same resolution of 1280×720 (*i.e.* HDTV 720P) and same frame rate of 30 frames per second. However, the sequences differ in the height of the camera above the ground, the viewing direction, the field of view, and the illumination conditions. The three sequences are labeled ground view, pole view, and aerial view.

The ground view camera is positioned at a height of 3.3 meters above the ground with an elevation angle just 5° below the horizon (measured relative to the principal ray emitted from the center of



Ground view
height: 3.3 m
elevation angle: -5°
angle of view: 49°
illumination: overcast



Pole view
height: 10 m
elevation angle: -18°
angle of view: 49°
illumination: sunny



Aerial view
height: 84 m
elevation angle: -41°
angle of view: 21°
illumination: overcast

Figure 8.1: Frames from three different video sequences used in tracking. The cameras are placed at different heights above the ground with different elevation angles (angle between principal ray and ground plane) and different angles of view (measured horizontally).

the image). The lens produces a 49° angle of view measured horizontally, similar to that of the still images in Chapter 7. This sequence represents the viewing conditions typically found in a camera mounted on the side of a roadway, on top of a single-story structure for example. Vehicles are observed primarily from the side and undergo significant scale change as they drive towards or away from the camera. Vehicles in the distance have around 3 cm per pixel resolution, and the resolution increases to about 1 cm per pixel as they approach the camera. This range of resolutions is similar to image scale 3 in the experiments of Chapter 7. Illumination is overcast in this sequence, so strong cast shadows are not present.

The pole view sequence raises the camera to a height of 10 meters above the ground and lowers the elevation angle to 18° below the horizon. This sequence represents the viewing conditions typically found in a camera mounted atop a tall utility pole. More of the top surface of the vehicles is visible. The lens is the same as that in the ground view resulting in a 49° angle of view, but the viewing distance is greater so the image resolution on vehicles is less on average. Vehicles still undergo significant scale change when traveling along the road that runs from top to bottom in the image. In the far upper right corner, the resolution is about 8 cm per pixel, but this section of road is ignored. Vehicles are not considered for tracking until they emerge from behind the tree at a resolution around 4 cm per pixel. At the bottom left corner, the resolution increases to about 1.5 cm per pixel. However, vehicles turning the corner from right to left (as shown in Figure 8.1) maintain a fairly constant resolution around 1.5 cm per pixel. For comparison to Chapter 7, this sequence is also closest to image scale 3. This sequence has direct illumination by the sun causing crisp cast shadows and occasional flares when the sun is directly reflected in the vehicle surface.

The aerial view sequence increases the camera height to 84 meters above the ground and lowers the elevation angle to 41° below the horizon. This sequence simulates the viewing conditions found in a stabilized aerial video capture. The distance is great enough that vehicle projection is locally well-modeled by an affine projection. Over the entire scene, however, a perspective projection is still more appropriate and is still used. From this great distance, a lens with greater magnification is needed to produce usable vehicle resolution. The resulting angle of view is 21° , less than half that of the other sequences. In this sequence, the resolution on the vehicles is fairly constant around 4 cm per pixel at all points in the image. In Chapter 7, image scale 4 is the closest match. Illumination is also overcast as in the ground sequence.

8.2 Tracking with fixed shape models

Some of the vehicles observed in the video sequences are identified as corresponding to a model in the CAD shape database. For these vehicles, tracking can be testing with fixed, ground truth shape. Other observed vehicles are similar size and shape to a CAD model, but not an exact match. These approximate shapes are also examined in fixed shape tracking. Track initialization is the same for fixed shape vehicles except that the initialization is complete after alignment of silhouette centroids. There is no need to minimize the difference between the silhouette boundaries

because the models are already known to have the correct size and shape. In all experiments, the background modeling is run for 100 frames to initialize the background model before enabling track initialization. Track initialization is enabled at a frame when the vehicle is fully visible, because the simple track initialization method is currently unable to cope with partial occlusion. Once the track is initialized, the background model is no longer used in tracking, and the algorithm is much more robust to occlusions (as demonstrated below).

First consider a fairly easy tracking example. Tracking results for a white PT Cruiser are shown in Figure 8.2. This example comes from the pole view sequence, and the vehicle is tracked for 180 frames. An exact model of the vehicle is available: model number 46 in Table 5.1 and Figure 5.1. To compactly represent a summary of the video result, a montage of several key frames is shown as a single image. Tracking results for these key frames are overlaid onto the montage. Since Dodecahedral and Ferryman are similar and all three Detailed models are similar, only the Ferryman and Detailed 3 models are shown as overlays. However, the plots at the top of Figure 8.2 show results for all five models. The first two plots show estimated translation velocity and angular velocity as it evolves over time. These velocities are obtained from the optimization at each frame and are not necessarily equal to the time difference between estimated poses. The estimated velocities are the values used in the pose prediction at each frame. Initial estimates of velocity may be poor and it may take several frames to converge to the true velocities. Yet, this does not indicate that the track itself is off course. It only indicates that the pose predictions have more significant error before correction. Notice that the Dodecahedral and Ferryman models tend to have worse initial velocity estimates. This is because the models align less precisely to the image, which translates into larger error in the initial two-frame pose difference estimate. For the same reason, the velocity estimates are noisy for the simple models and more smooth for the Detailed models, especially Detailed 2 and 3. The final plot shows the recovered path of the vehicle on the ground plane. A dot is placed on the Detailed 3 curve at one second intervals for a time reference. The plot axes are unevenly scaled to better show the difference between models. All models produce a similar and relatively smooth track in this case. There is just a small amount more oscillation in the positions determined by Dodecahedral and Ferryman.

A more challenging example of tracking is shown in Figure 8.3. This example contains 140 frames from the aerial view sequence and involves the exact same vehicle, a PT Cruiser, but in a dark red color and at a much lower resolution. Furthermore, this sequence contains multiple examples of partial occlusion and even one example of total occlusion. The same collection of montage, overlays, and graphs is provided for comparison to the previous example. This montage shows seven key frames from the video sequence. In the second key frame, the front wheel is occluded by a traffic light. In the third key frame, the back wheel is occluded by a road sign and the front third of the vehicle is occluded by a tree. In the sixth key frame, the entire vehicle is completely occluded by a large tree. In the seventh key frame the front half of the vehicle is occluded. Remarkably, tracking is able to overcome these occlusions. Again, the velocities are relatively smooth with Detailed 3 being the smoothest. Some small spikes are seen in the parameters around frame 30 where the first partial



Figure 8.2: Tracking a white PT Cruiser with fixed shape models in pole view. Plots show estimated translation velocity, angular velocity, and path for all models. The top image is a montage of four key frames. The middle and bottom images overlay Ferryman and Detailed 3 results, respectively.

occlusion occurs. Much larger spikes occur around frame 120 when total occlusion occurs. All three Detailed model survive the total occlusion, but the Dodecahedral and Ferryman models both lose track when the model is fully occluded. The simple models fail because they do not predict the image edges as accurately. Poor edge alignment keeps the estimated covariance higher than that of the Detailed models. Thus, fitting relies more on the current observation and less on the dynamics. When total occlusion occurs, the tracker does not trust the dynamics enough to maintain the track.

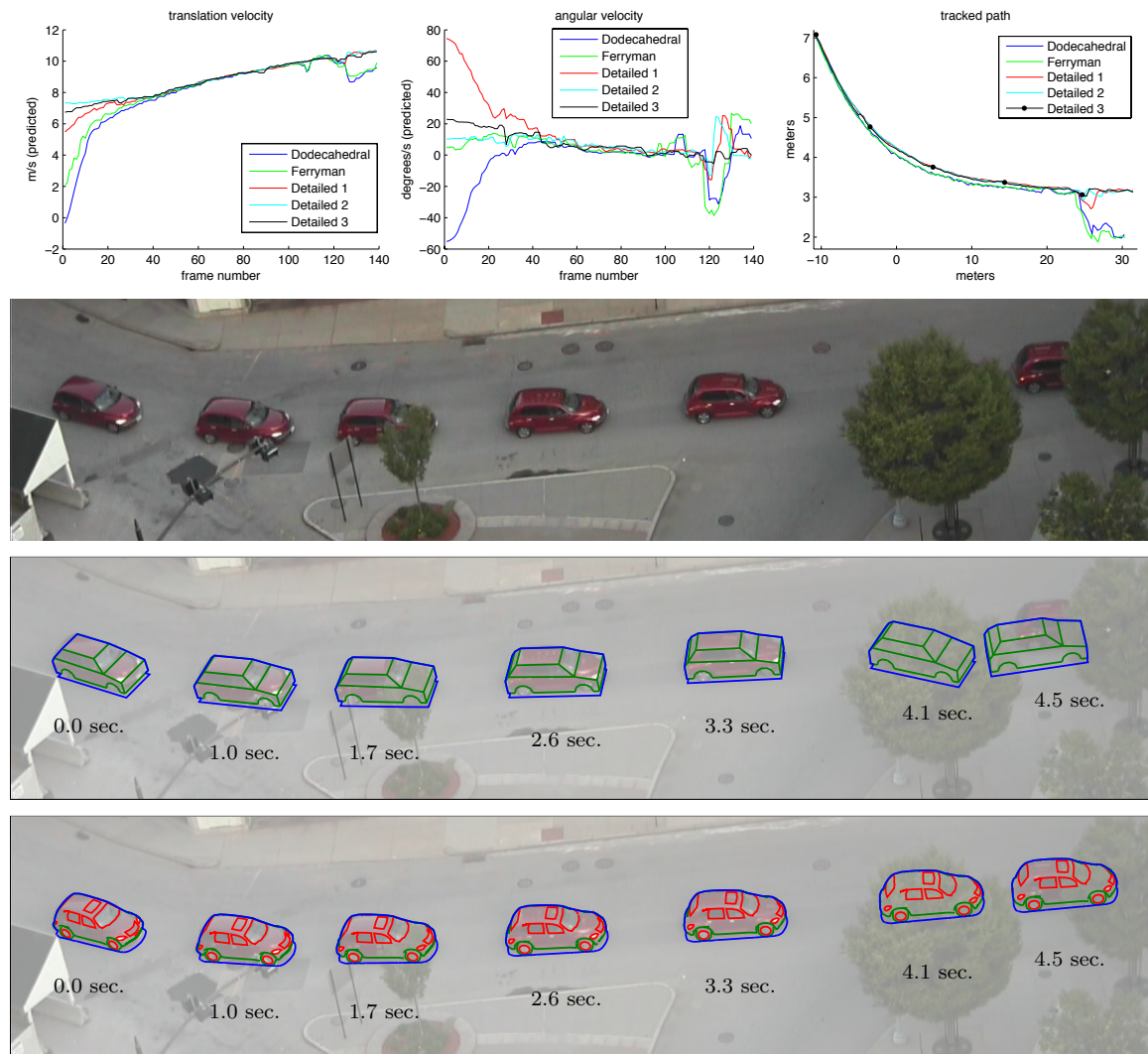


Figure 8.3: Tracking a dark red PT Crusier with fixed shape models in aerial view. Results are shown in the same format as Figure 8.2. Overlays use a seven frame montage with Ferryman in the middle and Detailed 3 on the bottom.

A third, and even more challenging, example of tracking is shown in Figure 8.4. This example contains 150 frames from the ground view sequence. The vehicle appears to be a Nissan Pathfinder, a vehicle that is not in the E3D database. The closest model in overall size, shape, and appearance seems to be the Chevrolet Trailblazer (model 28). So this experiment tests the ability to track with a

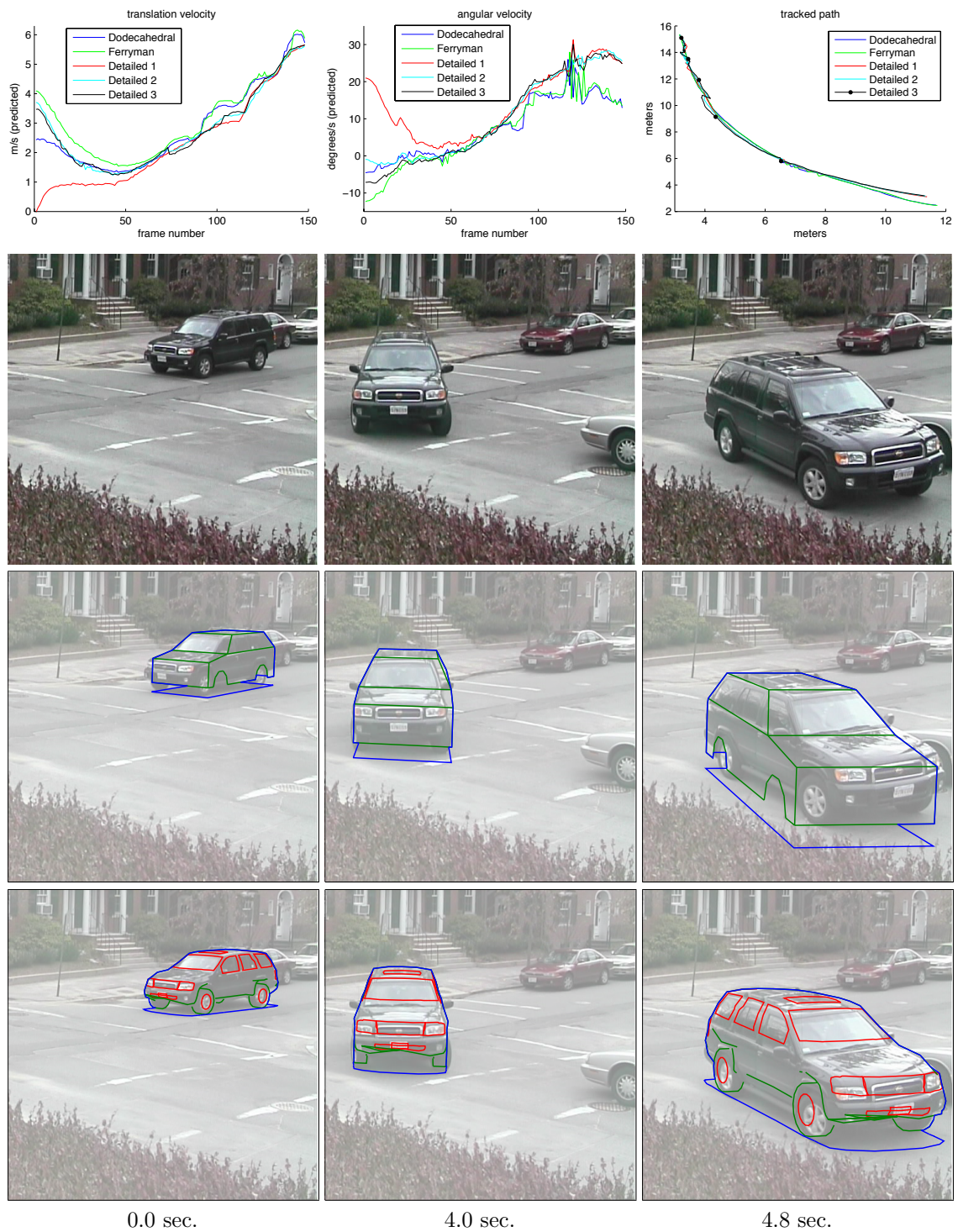


Figure 8.4: Tracking a black SUV with fixed shape models in ground view. Results are shown in the same format as Figure 8.2. Overlays use three separate frames with Ferryman in the middle and Detailed 3 on the bottom.

fixed shape model that is not exactly correct. In addition to the approximate model, this example is also challenging because the vehicle is black in color. Reflections from the environment are significant in this sequence, and many outlier edges are stronger than the feature edges. Yet, the biggest factor that makes this sequence challenging is the vehicle's orientation relative to the camera. Figure 8.4 shows three key frames from the video side-by-side. A montage is not used here because the vehicle would occlude itself. Notice that the vehicle is heading directly toward the camera in the second key frame. From this direction, the vehicle pose (and later, shape) are very poorly constrained. Only the front surface of the SUV is visible. Fairly significant changes in orientation and forward motion will have a very small impact on the alignment of curves. Tracking results are quite reasonable before the vehicle turns toward the camera. While facing the camera, the angular velocity estimates become quite erratic, especially in the Dodecahedral and Ferryman models. After facing the camera, the track is not completely lost, but model alignment is quite poor. The alignment is close enough that is likely that the Detailed models would eventually regain correct correspondence if the field of view was expanded to the right. However, the fate of the Dodecahedral and Ferryman tracks is less certain. The front of these models is significantly misaligned. Also, notice that the path of the Detailed 3 model is not nearly as smooth in this case. The model bounces around a bit trying to align features that cannot be aligned because the shape model is not accurate.

8.3 Tracking with shape estimation

The same three sequences now are revisited in experiments with unknown shape. First, the same three vehicles tested Section 8.2 are tracked again assuming unknown shape. Direct comparisons are made between tracking results with and without shape estimation. Then, several other unknown vehicles are tracked with shape estimation. Conclusions are drawn about the limits of simultaneous tracking and shape estimation.

The white PT Cruiser from Figure 8.2 is tracked again with simultaneous shape estimation. The new results are shown in Figure 8.5. These results are provided in exactly the same way as Figure 8.2 for direct comparison. Immediately, it is apparent that the tracking results look almost identical. The velocity profiles and tracked paths look almost the same with the exception of different large initial errors in the velocities of Dodecahedral and Ferryman. If anything, these curves are actually smoother than those with fixed shape. Additionally, parameters in the model allow compensation of alignment error by adjusting shape, for better or worse. Only two small deviations in the tracked path are noticeable. The Ferryman model deviates a bit in the first several frames (evident in the first keyframe overlay), and the Dodecahedral model deviates for several frames around 4.5 seconds. Both models recover. Deviations at the end of the sequence are caused by the vehicle leaving the camera field of view.

The shape is fairly consistent over the length of the white PT Cruiser track. Initially, the predicted window edges do not align, but this is quickly corrected. In the later half of the sequence, the trunk extends slightly beyond the rear window making the shape look a little bit more like a



Figure 8.5: Tracking a white PT Cruiser with variable shape models in pole view. Results are directly comparable with Figure 8.2. The only difference is the addition of shape estimation.

sedan. Deviation in this back edge occurs in all models. The reason is clear. As the vehicle turns the corner, the left side of the vehicle becomes shaded, and the darker appearance of the vehicle almost exactly matches the intensity of the gray pavement. The trailing edge of the PT Cruiser simply disappears. Since there is no edge to constrain the shape, the model starts to shift slightly back toward the prior mean shape, which is sedan-like.

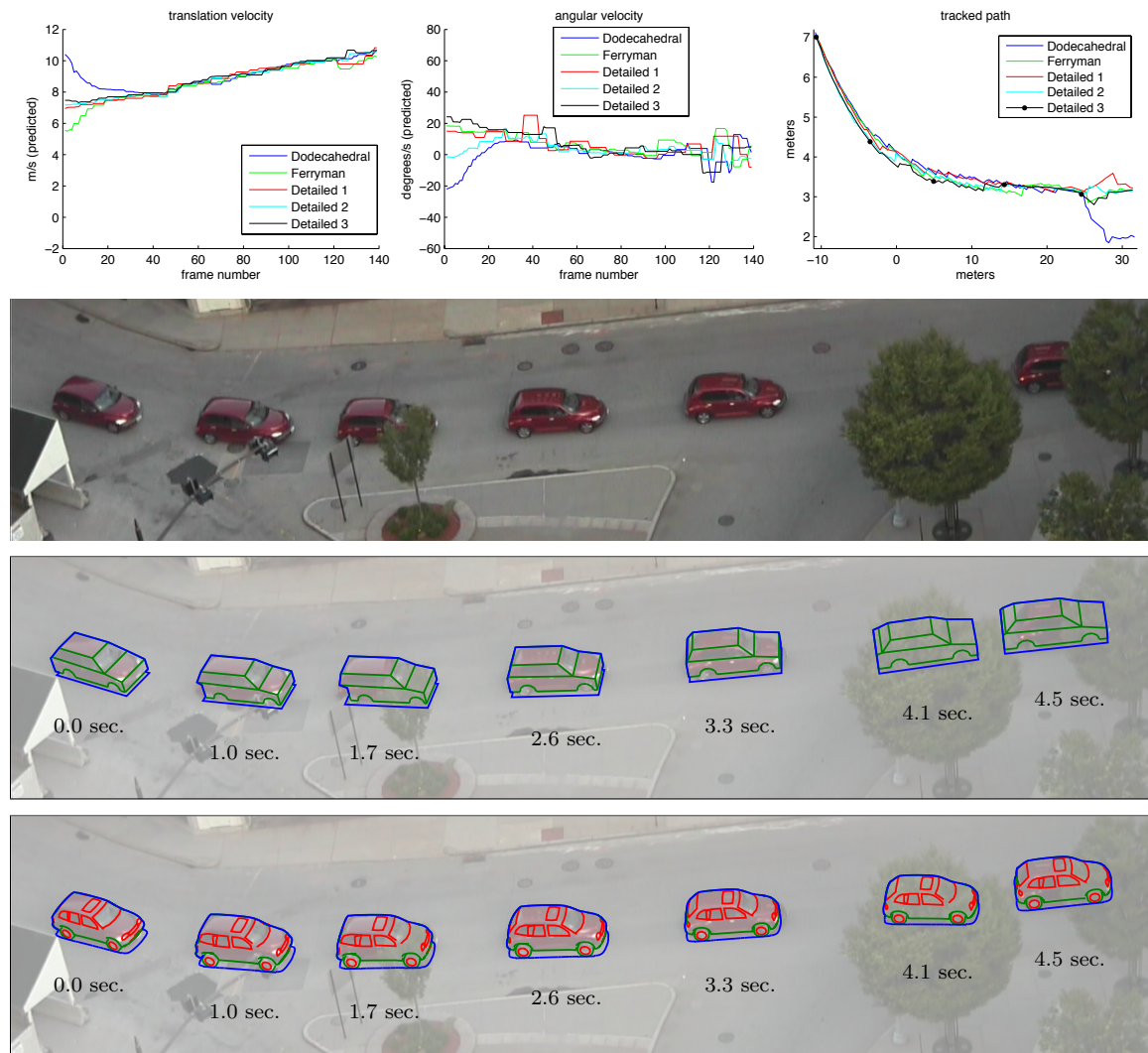


Figure 8.6: Tracking a dark red PT Cruiser with variable shape models in aerial view. Results are directly comparable with Figure 8.3. The only difference is the addition of shape estimation.

The red PT Cruiser, shown in Figure 8.6, also performs quite similarly to its fixed shape counterpart. In this case, the velocity profiles and tracked paths are a bit more noisy than with fixed shapes. The piecewise continuous appearance of the velocity profiles is a telltale sign that the parameter covariance is moderately underestimated. The predictions are overconfident and followed precisely for a few frames until the model alignment deviates significantly and the model snaps back to the

observed data suddenly. This result is an indication that the tracker is not perfectly scale invariant. A larger $\mathbf{Q}_t$ matrix or a greater divisor in Equation (6.37) would correct this problem. Nonetheless, tracking still performs well and handles the occlusions gracefully. Only the Dodecahedral model loses track in the total occlusion. However, the shape of the Ferryman model seems to be significantly affected by the occlusion. Key frames two and three show misalignment in the rear of the Ferryman model. The model also elongates during the total occlusion, and is poorly aligned when the vehicle reappears. The shape of the Detailed 3 model only changes slightly in height during frequent occlusion of the wheels, and the shape remains very stable during total occlusion with good post-occlusion alignment.

The black SUV in the ground view was challenging for fixed shape tracking. It is even more challenging for simultaneous shape estimation. Results are shown in Figure 8.7. During the first portion of the sequence, when the vehicle is observed from the side, tracking and shape estimation are quite good for a black vehicle. The Dodecahedral and Ferryman models undergo significant pose and shape oscillations (apparent in the velocity profiles) because the interior edges they predict are weak and there are many outliers. The Detailed models are able to latch onto the strong hubcap and headlight contours; thus, they track more stably. Some shape oscillation is observed in the upper rear of the Detailed models during the first few frames. The SUV is passing by another dark, parked vehicle, and the rear edges are not strongly detected in some frames. The path of the Detailed 3 model is now much smoother than with the incorrect, fixed shape model. Adaptation to the correct shape allows for better estimation of pose. However, once the vehicle turns toward the camera, estimation quickly breaks down. Not only is pose poorly constrained from the frontal viewpoint, but shape is poorly constrained as well. From the front, this could be almost any shape of vehicle. All vehicle models degrade in shape under this degenerate image data. They return to sedan-like shapes under the influence of the time independent prior of the RIEKF. Only the front windshield remains properly aligned. Without this prior (*i.e.* with the IEKF) the models completely collapse into meaningless shapes. This example is considered a tracking failure even though the incorrect shapes continue to follow the approximate path of the vehicle. This is, in fact, a very common failure mode. When model alignment is lost, the incorrect shape, sometimes with incorrect orientation, is often dragged along in the image space. Consequently, the 3-d track and shape are invalid, but tracking of rough position in the image continues to be valid, often for many frames.

Figure 8.8 shows a similar challenging vehicle in the pole view sequence. A dark colored, two-door sedan is tracked for 230 frames as it turns a corner toward the camera. In this case, when the vehicle is heading toward the camera, more of the top surface of the vehicle is observed, and the degeneracy is not quite as severe. Another important difference in this example is the illumination. Early in the sequence, direct illumination shades the left (driver's side) of the vehicle and washes out all the part features and the occluding contours at the base of the sedan. Then the vehicle passes partially through the shadow of a tree which washes out most of the remaining occluding contours on the top of the vehicle. Only the cast shadow boundary is detected as a strong and stable edge throughout the first half of the sequence. In this example, if the shadow boundary is not included

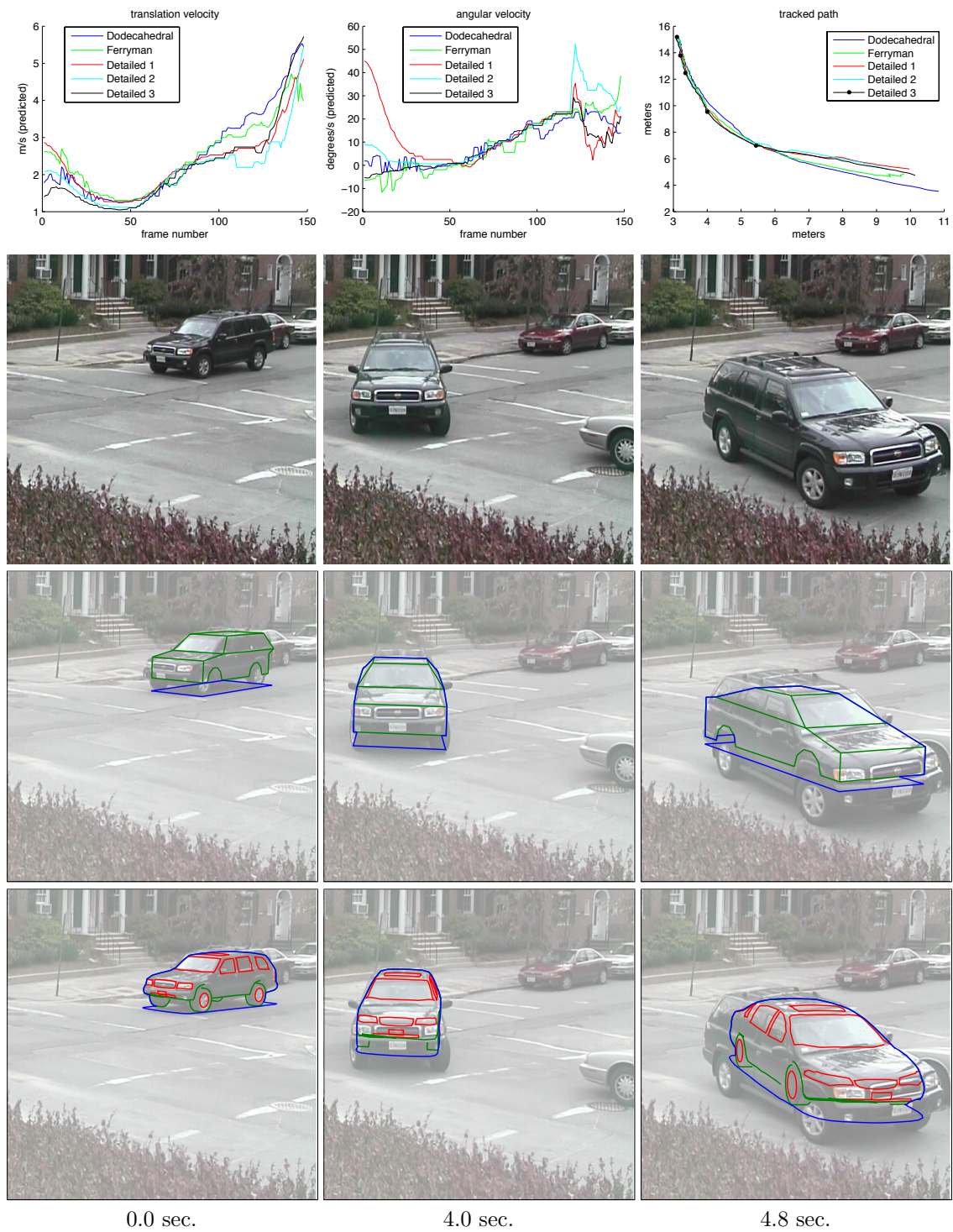


Figure 8.7: Tracking a black SUV with variable shape models in ground view. Results are directly comparable with Figure 8.4. The only difference is the addition of shape estimation.

as a hypothesized edge, then tracking fails even before the vehicle turns toward the camera. There are simply not enough other edges to track at the start of the sequence. Adding the shadow contour provides a strong shape constraint from the viewpoint of the sun, and the tracker succeeds. Tracking with the Detailed 3 model does quite well, but shape is somewhat distorted after the degenerate viewing direction. In the second half of the sequence, the vehicle is stretched in length slightly, the front occluding contour is mismatched to the shadow boundary, and the back window is shifted backward. However, the Dodecahedral and Ferryman models fare worse. The back window of these models is misaligned from the beginning, and both models fail at the end of the sequence when the sedan enters the shadow of another tree. The leading edge of these boxy models latches onto an occluding tree trunk, and the models are compressed into the shapes of compact SUVs.

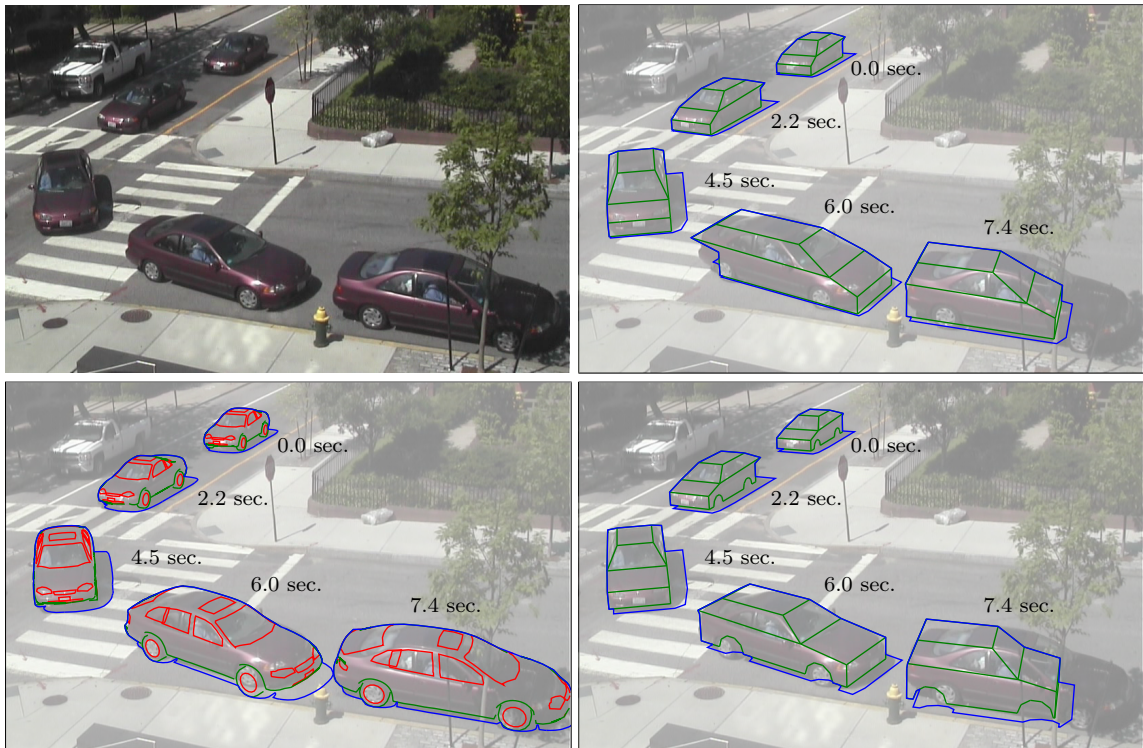


Figure 8.8: Tracking and shape estimation of a dark colored, two-door sedan in pole view. This montage shows five key frames. Clockwise starting from the upper right, the overlays are of Dodecahedral, Ferryman, and Detailed 3.

Another interesting example is the ground view sequence shown in Figure 8.9. In this example, a gray crossover SUV (it appears to be a Toyota RAV4) is tracked for 250 frames. The two-toned color scheme of the vehicle makes its edges more reliable than those of the black SUV in Figure 8.7. However, this vehicle is not similar in shape to anything in the CAD training database. Crossover SUVs are a newer class of vehicle not in the E3D database. They are similar in shape to traditional SUVs, but built on a sedan chassis rather than a truck chassis. Thus, they tend to be smaller and more curvy than traditional SUVs. This sequence also differs from the previous ones in the path



Figure 8.9: Tracking and shape estimation of a compact crossover SUV in ground view. This montage shows seven key frames. From top to bottom the overlays are Dodecahedral, Ferryman, and Detailed 3. The Dodecahedral model diverges in shape early on and loses track after 2.5 seconds.

of the vehicle. Instead of turning, this vehicle travels almost in a straight line. In the process, it covers the full range of scales in the ground view. There is also a small partial occlusion by a stop sign, an almost complete occlusion by a pair of utility poles, and several more partial occlusions by tree branches. The Dodecahedral model fails on sequence. Initialization is poor, and the alignment is never recovered. Yet, the misaligned model is dragged along for 2.5 seconds (75 frames), and the shape continues to distort. Eventually, the shape is deemed too unlikely and the track is terminated. Surprisingly, the Ferryman model does much better. In fact, both the Ferryman and Detailed 3 models produce similar trajectories. The estimated shape also aligns well with the images even though no similar model is found in the training data. Both Ferryman and Detailed 3 models survive the first two occlusions, and both tracks start to fail when the image resolution becomes too low. In both cases, the shape starts to deform and the vehicle mistakenly rotates to the right when encountering tree branch occlusions at the end of the sequence.

Several other vehicles are also tracked with results similar to those already shown. The major issues with tracking have already been illustrated, and there is no need to show the rest of the experiments in such detail. Instead, Figures 8.10 and 8.11 summarize the results by showing a single cropped frame for each vehicle and the alignment of the Dodecahedral, Ferryman, and Detailed 3 models. All tracking attempts are successful except where noted below, and none of these vehicles are known to be in the training database except for two Volkswagen Beetle examples.

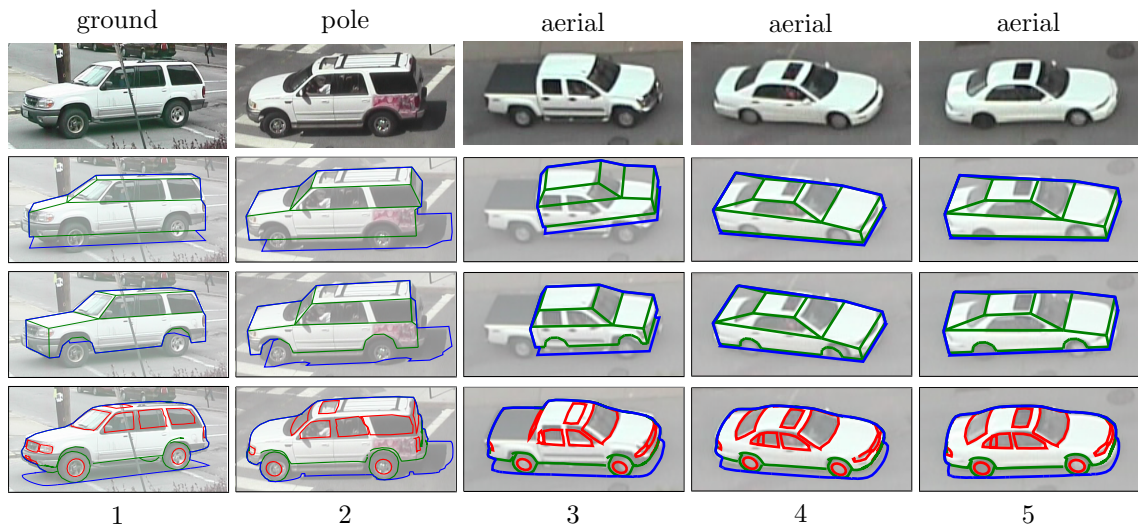


Figure 8.10: Single frame overlays from five other tracked vehicles. These vehicles are all white and are considered to be easy tracking examples. From top to bottom overlays are Dodecahedral, Ferryman, and Detailed 3.

Figure 8.10 shows examples of white vehicles with high contrast edges. These examples are expected to be the easiest to track. In all of these examples the Detailed 3 model performs very well. For vehicle number 1, only the Dodecahedral model has some trouble with model orientation from this side view. The Ferryman model is constrained by the wheel wells. All models perform well on vehicle number 2. Vehicle number 3 is a pickup truck, and both Dodecahedral and Ferryman

fail to capture the correct shape. Both simple models miss the bed of the truck entirely and fit a small SUV-like shape to the truck cab. Dodecahedral is also poorly aligned in this frame, but quickly recovers the pose. Tracking pose is successful with all models even if shape estimation is not. Vehicle numbers 4 and 5 are both white sedans (complete with sunroofs) and are the easiest of all examples. All three models perform very well, and the shape parameters are extremely stable over the length of the tracks. However, both of the simple models seem to have a hard time fully capturing the shape around the trunk and hood. The resulting 3-d models appear somewhat flattened like a race car.

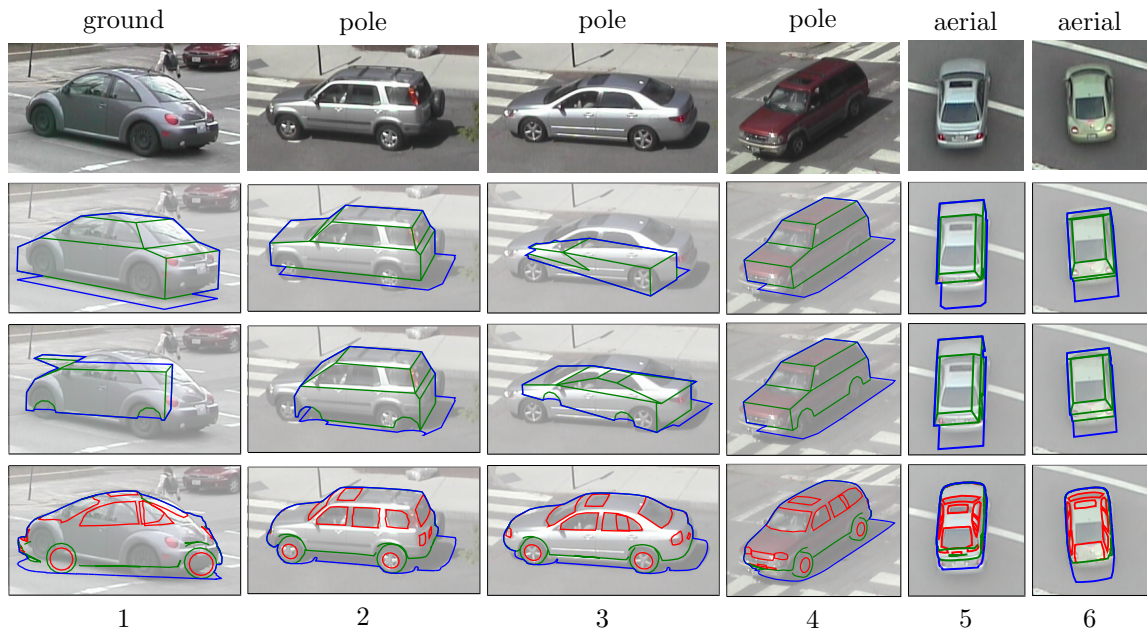


Figure 8.11: Single frame overlays from six other tracked vehicles. These vehicles are considered to be more difficult tracking examples. From top to bottom overlays are Dodecahedral, Ferryman, and Detailed 3.

Figure 8.11 shows examples of more difficult vehicles to track. Vehicle number 1 is a dark gray Volkswagen Beetle. This example illustrates one of the few total failures for the Detailed 3 model. Shape and orientation quickly diverge after initialization. This image is the ninth frame after initialization. The first frame shows good alignment, but the second frame already has significant orientation error. Such a failure suggests that the velocity initialization method is to blame. The Ferryman model suffers a similar fate; yet, the Dodecahedral model somehow manages a successful track. Vehicle number 2 is successful with all models. The Ferryman model shape estimate is poor in this frame but eventually recovers. Vehicle number 3 is tracked very well with the Detailed 3 model. However, for the simple models, this is another case of failure immediately after initialization. This image is the third frame after initialization, and both models have nearly collapsed. Starting the initialization at later frames produces the same, immediately diverging failure for Dodecahedral and Ferryman. Vehicle number 4 is successful for all models but is another example that requires

the shadow edge for successful tracking early in the sequence. The scenario is exactly the same as in Figure 8.8 except this SUV continues on a straight path. Alignment is a little less stable in the shadowed region compared to the sedan, but it stabilizes afterward. The final alignment is better than the sedan since the SUV does not pass through a degenerate view. Vehicle numbers 5 and 6 are both examples of tracking from a nearly nadir (top-down) view. Vehicle number 6 is another Volkswagen Beetle. In both of these examples, the shape initialization is poor and never fully recovers. Both initializations stretch the back window beyond the trunk of the vehicle. The nadir view is nearly as degenerate for shape estimation as the frontal view. Pose estimation is initially reasonable, but noisy. Eventually, the trajectory diverges for the VW Beetle with Detailed 3. Clearly, the proposed initialization method is not well suited to nadir views. Using a fixed shape model, vehicle number 6 is tracked successfully with the Detailed 3 model. Initialization still provides a misalignment, but the rigid model recovers several frames later.

In summary, model-based tracking is quite successful, with or without shape estimation. However, if shape is not estimated, the fixed model needs to accurately represent the shape to ensure smooth tracking results. All tested vehicle models track successfully in most cases. The Detailed models (especially Detailed 3) usually provide a more stable trajectory and are less likely to be derailed by occlusion. There are four primary causes of tracking and shape estimation failure: dark and/or shadowed vehicles with few stable edges, significant occlusion of the vehicle, poorly constrained viewing directions, and poor track initialization. The first cause is sometimes avoided using a prediction of cast shadow, which generates a multi-view constraint. Considering the multi-view results of Chapter 7, it is highly likely that multiple, synchronized videos from different viewpoints would avoid the second and third failure causes and greatly improve stability all around. More advanced initialization algorithms would address the fourth cause.

The shape estimates continue to evolve over the duration of each track, so it is difficult to compare recursive shape estimation during tracking to a global operation as in the multi-view fitting. The uncertainty in the estimate never goes away during tracking, so the shape can adjust at any time. One could enforce convergence of the shape by artificially reducing the covariance to zero over time. Forced convergence could be both good and bad. On one hand, if the shape is accurately estimated at the beginning of the sequence then it would be advantageous to lock in that shape in case later views poorly constrain the shape. On the other hand, if the shape is poorly estimated in the beginning of the sequence then it would be advantageous to correct the shape later on when more information is available. Unfortunately, the true feature correspondences are never known with certainty, so one cannot expect the shape uncertainty to ever vanish on its own.

Chapter 9

Experiments: Recognition from Shape

Recognition is one of the primary reasons for shape recovery in vehicle surveillance applications. Each recovered vehicle shape is compactly represented as a point in a low dimensional shape space. The conjecture is that vehicle shape is a good indicator of vehicle identity, and therefore the identity can be determined from the shape space parameters. While this may not be entirely true with regard to identifying specific vehicle instances, it is more likely to be true with regard to generic vehicle classes.

Recently, much progress has been made with regard to recognition of specific vehicle instances [92, 91, 35]. Often called *vehicle fingerprinting*, the goal of this type of recognition is to identify the same vehicle instance when observed in different images. For example, a vehicle is first observed by one surveillance camera. The goal is to recognize the same vehicle when it later appears in the video captured by another camera. Note that the true identity of the vehicle is never determined by this process. These algorithms do not identify the vehicle by make and model (*e.g.* Toyota Corolla, Dodge Ram, *etc.*) or even by generic class (*e.g.* sedan, pickup, SUV, *etc.*). It does not matter what the vehicle is as long as it can be found in another image.

Shape space is much better suited to generic class recognition. Since sedans have a similar shape, one would expect sedans to cluster in shape space. Likewise, one would also expect pickup trucks to cluster in shape space but at a different location than sedans. Figure 9.1 plots each training vehicle by E3D class in the first two principle components of shape space. Clustering is already apparent, even in 2-d. The task of recognition is to exploit this clustering in shape space to identify the generic class of a reconstructed vehicle shape. It may even be possible to identify the specific make and model of a vehicle by searching the training data for the nearest neighbor in shape space. However, such recognition assumes that the observed vehicle is contained in the database of 3-d models. Often the observed vehicle is not in the database but enough similar models are available to determine the generic vehicle class. Thus, the primary focus of this chapter is studying generic class recognition in

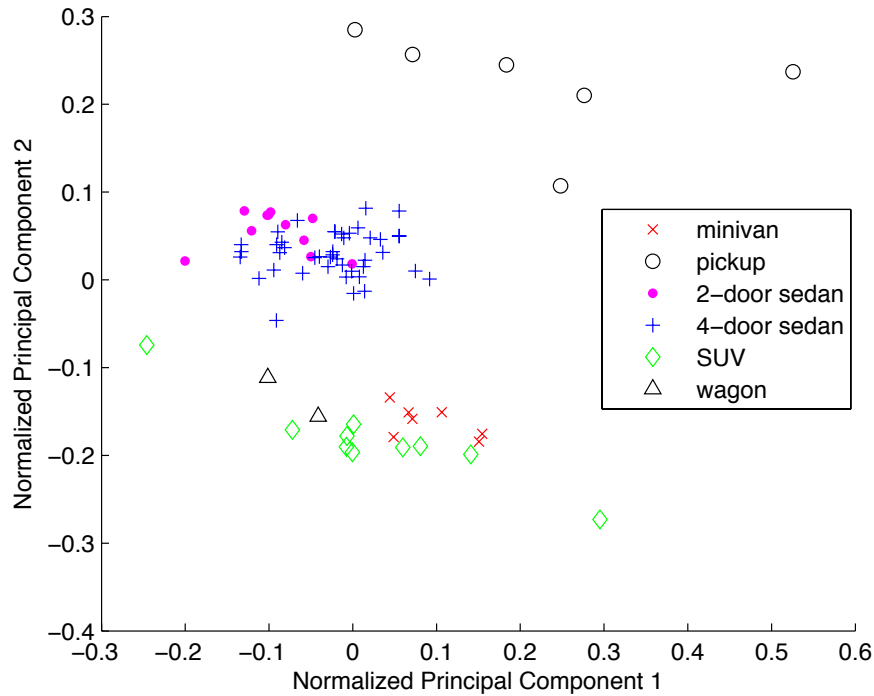


Figure 9.1: Scatter of vehicle classes in the first two principal components normalized by standard deviation. Each point represents the shape learned from a vehicle in Table 5.1. This plot uses the Detailed 2 model, but the plot is very similar for the other models considered in Chapter 7.

vehicle shape space.

Adding additional dimensions allows for more class separation, but it is already clear from Figure 9.1 that some vehicle classes may be better separated than others. Pickup trucks are well separated from everything. Two- and four-door sedans overlap significantly but are separated from other classes. SUVs, minivans, and wagons are also tightly grouped, but overlap somewhat less than two- and four-door sedans. In fact, in 2-d, the data seems to naturally cluster into three classes: pickups, sedans, and the union of minivans, SUVs, and wagons. It is helpful to keep in mind that the taxonomy of vehicle classification is not based on shape alone. Vehicle classes are also largely determined by function. A minivan has a third row of seats to carry extra passengers. An SUV uses the same space to carry extra cargo. Both vehicles could potentially have the same exterior shape. Experiments in this chapter attempt to recognize the E3D specified vehicle classes but also consider the three composite classes listed above.

Once the vehicles are mapped to shape space, any classification algorithm for N -d data is applicable. As with tracking, the goal of this chapter is not to push the envelope in the recognition field. Rather, the goal is to evaluate the standard recognition techniques for comparison to previous work on recognition with deformable vehicle models. Once again, the key players are Ferryman *et al.* [30] and Koller *et al.* [60], and once again, Ferryman takes the simple approach while Koller takes the theoretically grounded approach. Ferryman’s approach is to compute the mean of each class in

the training data. A new point is classified by finding the distance to each of the class means and selecting the closest. Koller's approach is to use Bayesian decision theory with each class represented as a multivariate normal distribution. A review of Bayesian decision theory reveals that Ferryman's approach is just a simple, special case of Koller's approach.

9.1 Bayesian decision theory

A full explanation of Bayesian decision theory can be found in Duda *et al.* [27]. What follows in this section is only a brief overview of the necessary background for the experiments to come.

Bayesian decision theory uses a probabilistic framework to select the most probable vehicle class given the position of a point in shape space. Let the variable c_i represent the event that vehicle belongs to the i -th class ($i \in [1, K]$), and let $\mathbf{x}$ represent the observed point in vehicle shape space. Bayesian decision theory assumes that the events c_i and observation $\mathbf{x}$ are random variables with associated probability distributions. In particular, the distribution $P(c_i|\mathbf{x})$ is the probability that the vehicle belongs to class i given the observation point $\mathbf{x}$. This distribution is called the **posterior**. If one can evaluate the posterior distribution for each of the K classes, then the decision is made by selecting the class with the highest posterior probability. More formally, this decision rule results in the lowest probability of making a classification error.

The posterior distribution is usually not directly available. More often one has a model for $P(\mathbf{x}|c_i)$, called the **likelihood** distribution. Given a known vehicle class, the likelihood distribution describes the likelihood of occurrence of each shape space point. A likelihood distribution is typically learned by fitting a distribution model to all the training data with a given class label. A separate likelihood distribution can be estimated for each vehicle class in this way.

The **joint** distribution, $P(\mathbf{x}, c_i)$, describes the probability of jointly selecting both a class label and a position in shape space. The order of the variables is unimportant, so $P(\mathbf{x}, c_i) = P(c_i, \mathbf{x})$. A joint distribution can be decomposed into the product

$$P(\mathbf{x}, c_i) = P(\mathbf{x}|c_i) P(c_i), \quad (9.1)$$

where $P(\mathbf{x}|c_i)$ is a likelihood and $P(c_i)$ is a **prior**. A prior distribution on c_i is the probability of the i -th class occurring before even considering any observations. For example, sedans may appear more frequently than pickups because more people own sedans. Before observing a vehicle, one can already guess that the vehicle is more likely to be a sedan than a pickup. The vehicle class prior can be estimated from the frequency of occurrence of class labels in training data.

Reversing the variables in the joint distribution, the same decomposition applies:

$$P(c_i, \mathbf{x}) = P(c_i|\mathbf{x}) P(\mathbf{x}). \quad (9.2)$$

Equating (9.1) and (9.2) leads to an expression known as Bayes rule:

$$P(c_i|\mathbf{x}) = \frac{P(\mathbf{x}|c_i) P(c_i)}{P(\mathbf{x})}. \quad (9.3)$$

Bayes rule expresses the desired posterior distribution in terms of the estimated likelihood and prior distributions. The denominator in (9.3) is the probability of observing any point in shape space. This too can be computed in terms of known quantities by marginalizing the joint distribution over the class labels. That is, sum the joint distribution over all possible classes:

$$P(\mathbf{x}) = \sum_{i=1}^K P(\mathbf{x}, c_i) = \sum_{i=1}^K P(\mathbf{x}|c_i) P(c_i). \quad (9.4)$$

However, making a Bayesian decision about the class label does not require computation of the actual value of the posterior distribution. It suffices to select the maximum posterior over all K classes. Note that the denominator, $P(\mathbf{x})$ is the same for each class. Therefore, the posterior distribution is proportional to the numerator in Bayes rule, and the Bayes decision rule becomes

$$\arg \max_i (P(\mathbf{x}|c_i) P(c_i)). \quad (9.5)$$

All that remains is to learn a model of the likelihood and prior distributions from training data.

9.1.1 Multivariate normal likelihoods

A common choice for the likelihood distributions is the multivariate normal (*i.e.* Gaussian) distribution. The normal distribution is popular both for its analytical tractability and because it often is a good model for the distribution of real data. Koller *et al.* [60] use the multivariate normal distribution and the same is used in this thesis.

Normal distributions have already made an appearance in earlier chapters (Refer to Sections 3.5.2 and 6.1.1.). To review, the expression for the likelihood as a multivariate normal distribution is

$$P(\mathbf{x}|c_i) = \mathcal{N}(\mathbf{x}|\mu_i, \Sigma_i) = \frac{1}{(2\pi)^{N/2} |\Sigma_i|^{1/2}} \exp \left(-\frac{1}{2} (\mathbf{x} - \mu_i)^\top \Sigma_i^{-1} (\mathbf{x} - \mu_i) \right), \quad (9.6)$$

where μ_i and Σ_i are the mean and covariance for the i -th vehicle class, and N is the dimension of the shape space. The mean and covariance are estimated by computing their expected values from the training data

$$\mu_i = \frac{1}{M_i} \sum_{j=1}^{M_i} \mathbf{x}_{i,j}, \quad (9.7)$$

$$\Sigma_i = \frac{1}{M_i} \sum_{j=1}^{M_i} (\mathbf{x}_{i,j} - \mu_i) (\mathbf{x}_{i,j} - \mu_i)^\top, \quad (9.8)$$

where M_i is the number of training examples in the i -th class, and $\mathbf{x}_{i,j}$ is the j -th example from the i -th class.

Recall that the outcome of the decision rule is not affected by any positive scale or shift that is the same for all classes. Therefore, the $(2\pi)^{N/2}$ in the denominator of (9.6) can be safely removed. Taking this one step further, any monotonically increasing function applied to the likelihood will not change the maximum. Logarithm is a monotonically increasing function, and it simplifies the

computation by inverting the exponential. After applying the logarithm and some manipulation the decision rule reduces to

$$\arg \max_i (\mathbf{x}^\top \mathbf{A}_i \mathbf{x} + \mathbf{a}_i^\top \mathbf{x} + a_{i0}), \quad (9.9)$$

where

$$\mathbf{A}_i = -\frac{1}{2} \boldsymbol{\Sigma}_i^{-1}, \quad (9.10)$$

$$\mathbf{a}_i = \boldsymbol{\Sigma}_i^{-1} \mu_i, \quad (9.11)$$

$$a_{i0} = -\frac{1}{2} (\mu_i^\top \boldsymbol{\Sigma}_i^{-1} \mu_i + \ln |\boldsymbol{\Sigma}_i|) + \ln P(c_i). \quad (9.12)$$

The terms $\mathbf{A}_i$, $\mathbf{a}_i$, and a_{i0} do not depend on $\mathbf{x}$, so they are pre-computed for each class. Then, for each point, classification only requires the computation of the quadratic form in (9.9) for each class. The quadratic functions evaluated in (9.9) are called the discriminant functions.

Interestingly, the nearest-mean approach used by Ferryman *et al.* [30] is also covered by a special case of Bayesian classifier with normal likelihoods. Two simplifying assumptions are required. First, the prior distribution is assumed to be uniform (*i.e.* $P(c_i) = 1/K$ for all i). Second, the covariance matrices for all classes are assumed to be spherical with a common variance σ^2 (*i.e.* $\boldsymbol{\Sigma}_i = \sigma^2 \mathbf{I}$ for all i). With these constraints, many of the terms in the discriminant functions become constant over class and do not contribute to the determination of a maximum. What is left are linear discriminant functions that amount to picking the minimum distance to the mean. Notice that the class clusters in Figure 9.1 are elongated and not very spherical. Thus, Koller's approach is a better model for the data and should out perform Ferryman's approach.

9.2 Classification of training data

One way to evaluate the feasibility of a classifier is to test it on the training data from which it was learned. If the classifier performs well on the training data that gives hope that the classes are indeed distinguishable using the available measurements. Yet, it makes no guarantee of how well the classifier will generalize to additional data. Especially since the training data is generated by pristine 3-d models, and the testing data comes from fitting the model to images. However, if the classifier fails miserably on the training data, then there is no hope of generalization. Either the classifier needs improvement, or the classes are not distinguishable from the data.

Training classifiers usually requires a large number of training samples for good results, especially as the dimension of the space increases. The number of samples should be much larger than the dimension. Computing an invertible covariance matrix requires that the number of samples in each class be at least one larger than the dimension. However, the number of available samples in the E3D database is quite small. There are only six pickup trucks and only two wagons. This limits Koller's approach to one dimension if using all six E3D classes. There are ways around the covariance degeneracy. For example, applying regularization similar to Section 4.3.2 or other matrix constraints. Instead of deviating from the standard theory, the wagon class is discarded. Only up

to five dimensions are considered for the remaining five classes (pickup, 2-door sedan, 4-door sedan, minivan, and SUV) to satisfy the constraints imposed by the pickup class. For comparison, the three composite classes defined above (pickup, sedan, minivan+SUV+wagon) are also studied.

In the following experiments, classifiers are learned using the approaches of both Ferryman and Koller. A uniform prior distribution on vehicle classes is assumed in both cases. Learning is applied in the shape spaces associated with each of the five vehicle models defined in Section 7.1.2. In each space, subspaces from one to five dimensions are considered. Figure 9.2 shows the percentage of correct classification of the training data into five vehicle classes. Clearly, increasing the dimension improves classification in almost all cases. Interestingly, when using only the first principal component, the simpler models out perform the more complex ones. This trend quickly reverses when adding more dimensions. The Bayesian classifiers with Gaussian likelihoods tend to outperform the nearest mean classifiers by about 10%. In four dimensions, both the Detailed 2 and Detailed 3 models reach perfect classification of the training data under the Bayesian classifiers. All Detailed models reach 100% correct classification in five dimensions using the Bayesian classifiers. The nearest mean classifiers come close to perfect in four and five dimensions, but they are always stuck with a small number of misclassifications. The Dodecahedral and Ferryman vehicle models max out around 90% and 80% for the Bayesian and nearest mean classifiers, respectively.

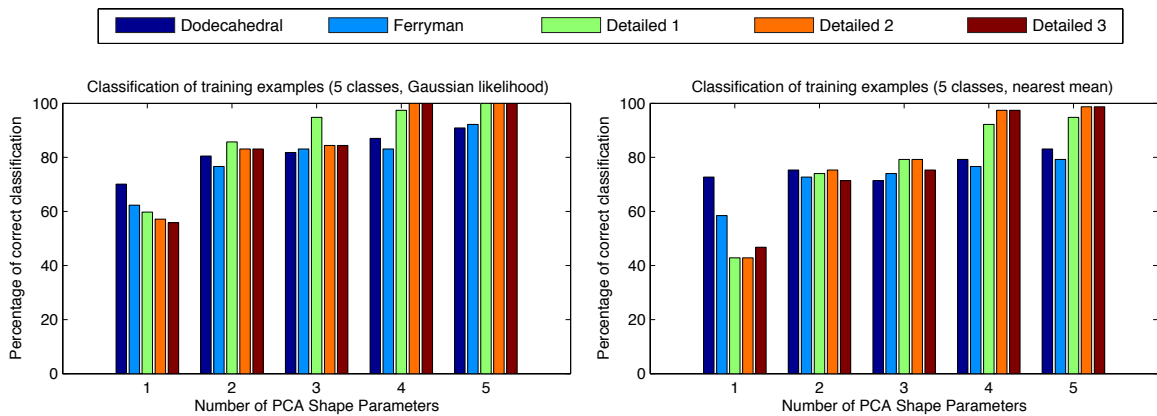


Figure 9.2: Success rates for classification of training data into five classes. On the left, a Bayesian classifier with Gaussian likelihoods. On the right, a nearest mean classifier.

Looking at the confusion matrices for some of these experiments helps to better understand the modes of failure. In particular, consider the approach of Koller *et al.* with the Dodecahedral model and Bayesian classifier. The confusion matrix for five vehicle classes and five dimensional shape space is shown in Table 9.1. The only misclassifications occur between two- and four-door sedans. Such an error is not surprising because the Dodecahedral model makes no measurement of the position of doors. The only information about the number of doors comes from weak correlation with the overall vehicle dimensions. Conversely, the Detailed models contain parts representing side windows, and measurements of side windows have much higher correlation with the number of doors. This extra information allows the Detailed models to succeed where Dodecahedral model fails.

		Predicted				
		minivan	pickup	sedan 2	sedan 4	SUV
Actual	minivan	7	0	0	0	0
	pickup	0	6	0	0	0
	sedan 2-door	0	0	10	1	0
	sedan 4-door	0	0	6	37	0
	SUV	0	0	0	0	10

Table 9.1: Confusion matrix for Dodecahedral, 5 parameters, and Gaussian likelihood.

A similar situation arises with the approach of Ferryman *et al.* Table 9.2 shows the confusion matrix for the Ferryman model with a nearest mean classifier of five vehicle classes in a five dimensional shape space. The Ferryman model is also not suited for determining the number of doors and most of the failures come from confusion between two- and four-door sedans. Additionally, the nearest mean classifier also makes some errors in distinguishing minivans and SUVs. Ambiguity between minivans and SUVs is the result of weakness in the nearest mean classifier not the choice of vehicle model. Both Ferryman and Dodecahedral models confuse minivans and SUVs under the nearest mean classifier but not under the more general Bayesian classifier. However, both Ferryman and Dodecahedral models confuse two- and four-door sedans under both classifiers.

		Predicted				
		minivan	pickup	sedan 2	sedan 4	SUV
Actual	minivan	6	0	0	0	1
	pickup	0	6	0	0	0
	sedan 2-door	0	0	10	1	0
	sedan 4-door	0	0	13	30	0
	SUV	1	0	0	0	9

Table 9.2: Confusion matrix for Ferryman, 5 parameters, and nearest mean.

For comparison, the same experiments are run on the three-way classification problem discussed above. The three classes are formed by merging the two- and four-door classes, and merging the minivans with SUVs and wagons. Figure 9.3 shows the percentage of correct classifications of the training data into these three vehicle classes. As expected, classification into these three composite classes is much easier. For two or more dimensional spaces, both classifiers achieve perfect or near perfect classification of the training data using any of the vehicle models.

In summary, the vehicle shape spaces appear to contain sufficient information to allow for the classification of vehicles. The Bayesian classifier with Gaussian likelihoods suggested by Koller *et al.* makes fewer errors than the nearest mean classifier suggested by Ferryman *et al.* Yet, success on the training data does not ensure success on the test data. The Bayesian classifier provides a better fit to the training data, but a better fit could result in over fitting, especially since there are so few training examples. Choice of vehicle model also has an effect on the success of classifiers. The Dodecahedral and Ferryman models have trouble distinguishing between two- and four-door sedans, while the Detailed models are much better at this task. One would expect the deficiencies

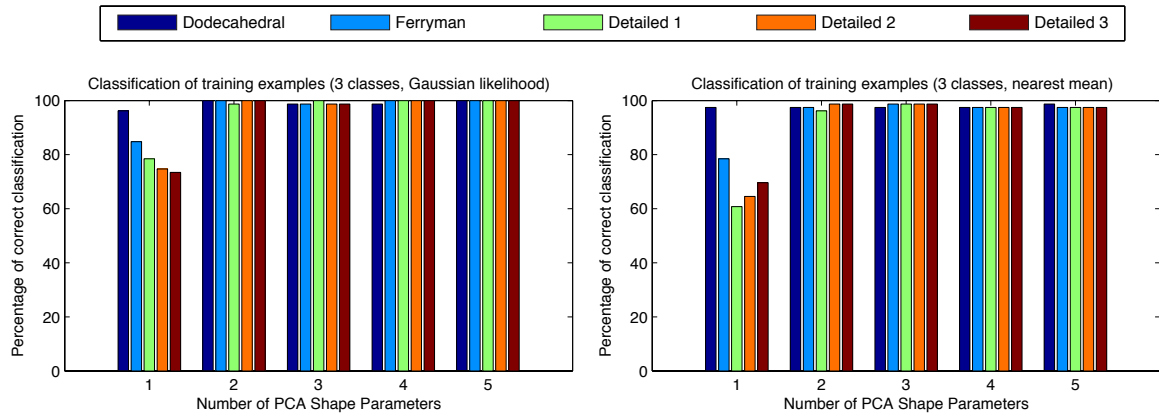


Figure 9.3: Success rates for classification of training data into three classes. On the left, a Bayesian classifier with Gaussian likelihoods. On the right, a nearest mean classifier.

of the Dodecahedral and Ferryman models to carry over to classification of test data. These simple vehicle models are not well suited for making such subtle class distinctions.

9.3 Classification of recovered vehicle shape

The same classification techniques applied to training data in Section 9.2 are now applied to shape parameters recovered by fitting the models to images. Only a five-parameter shape space is considered for recognition with each of the five models. Five shape parameters are recovered in all fitting experiments, and using fewer parameters does not benefit recognition. For each experiment in this section, the classified shape parameters come from the results of an experiment in Chapters 7 or 8.

9.3.1 Classification from multi-view images

In Chapter 7 a multiple view image data set is presented. Refer to Figure 7.1. Each of these images correspond to a CAD model in the training database. Section 9.2 already showed good recognition performance on the training data. Therefore, the results in this section provide insight into how well the classifiers generalize to noisy reconstructions of known shapes.

First consider the recovered shapes from Section 7.2.1 on fitting with good initialization. To review, the models are fit to the images using ground truth pose and shape for initial parameters and low initial uncertainty. These shape parameters are expected to be close to the ground truth parameters used to train the classifiers. There are no gross errors resulting from incorrect local minima. Classifications using the Bayesian classifier with Gaussian likelihoods are shown in Table 9.3 for each image set, image scale, and vehicle model. The set of five vehicle classes is used in these results. Similarly, Table 9.4 shows the classifications of the nearest mean classifier on the same data. The Volvo V70XC image set is included, but will never have correct classification because station wagons are not represented by any of the five vehicle classes.

	5	4	3	2	1
K	U	U	S	S	U
F	U	U	U	U	U
D1	U	U	U	U	U
D2	U	U	U	U	U
D3	U	U	U	U	U
Chevy S10 Blazer					
	5	4	3	2	1
K	S	U	U	M	M
F	U	U	U	U	S
D1	M	M	M	M	U
D2	U	U	U	U	U
D3	M	U	U	M	U
Dodge Caravan					
	5	4	3	2	1
K	S	S	S	S	S
F	S	C	S	S	S
D1	S	S	S	S	S
D2	S	S	S	S	S
D3	S	S	S	S	S
Dodge Stratus					
	5	4	3	2	1
K	S	S	S	S	S
F	P	P	P	P	P
D1	P	C	P	S	S
D2	P	P	P	P	P
D3	P	P	P	P	P
Toyota Tundra					
	5	4	3	2	1
K	S	U	U	U	U
F	U	U	U	U	U
D1	U	U	U	U	U
D2	U	U	U	U	U
D3	U	U	U	U	U
Toyota 4Runner					
	5	4	3	2	1
K	S	S	C	S	S
F	S	C	C	C	C
D1	S	C	C	C	C
D2	S	C	C	C	C
D3	S	C	C	C	C
VW Beetle (black)					
	5	4	3	2	1
K	S	S	S	S	S
F	S	C	C	C	C
D1	S	C	C	C	C
D2	S	C	C	C	C
D3	S	C	C	C	C
VW Beetle (yellow)					
	5	4	3	2	1
K	S	U	U	U	U
F	U	U	M	U	U
D1	U	U	U	U	U
D2	U	U	U	U	U
D3	U	U	U	U	U
Volvo V70XC					

Table 9.3: Bayesian classification into five classes using recovered shape from multiple views. Good initialization (Section 7.2.1) and five shape parameters are used for each vehicle model and image scale (labeled the same as Figure 7.16). Vehicle classes are minivan (M), SUV (U), 4-door sedan (S), 2-door sedan (C), and pickup truck (P). Incorrect classifications are identified in gray.

	5	4	3	2	1
K	U	U	S	M	U
F	U	U	U	U	U
D1	U	U	U	U	U
D2	U	U	U	U	U
D3	U	U	U	U	U
Chevy S10 Blazer					
	5	4	3	2	1
K	U	M	M	M	M
F	U	M	M	M	M
D1	M	M	M	M	M
D2	M	M	M	M	M
D3	M	M	M	M	M
Dodge Caravan					
	5	4	3	2	1
K	C	C	S	S	S
F	S	C	S	S	S
D1	S	S	S	C	S
D2	S	S	S	S	S
D3	S	S	S	S	S
Dodge Stratus					
	5	4	3	2	1
K	S	U	U	U	U
F	U	C	U	U	U
D1	S	S	S	S	S
D2	S	S	S	S	S
D3	S	S	S	S	S
Nissan Maxima					
	5	4	3	2	1
K	C	C	C	C	C
F	C	C	C	C	C
D1	M	S	C	C	C
D2	M	C	C	C	C
D3	M	C	C	C	C
Toyota 4Runner					
	5	4	3	2	1
K	P	P	P	P	P
F	P	P	P	P	P
D1	P	P	P	P	P
D2	P	P	P	P	P
D3	P	P	P	P	P
Toyota Tundra					
	5	4	3	2	1
K	U	U	S	S	M
F	M	S	S	S	S
D1	S	U	S	S	S
D2	S	U	S	S	S
D3	S	S	S	S	S
Volvo V70XC					

Table 9.4: Nearest mean classification into five classes using recovered shape from multiple views. Data and labels are the same as in Table 9.3.

Most of the classifications in Table 9.4 are correct. Notably, the Volkswagen Beetles and Dodge Caravan seem to have the highest rate of misclassifications. It is not surprising that the Volkswagen Beetles are frequently misclassified. The Beetle is a very unusual shape, quite different from all other two-door sedans. However, one would not expect many misclassifications of the Dodge Caravan. Most of these misclassifications choose SUV, a reasonable mistake. Yet, one might also expect to see the Blazer and 4Runner frequently misclassified as minivan. The Blazer and 4Runner are only misclassified as four-door sedans using the Dodecahedral model. They are correctly classified as SUV in all other cases. The poor performance on the Dodge Caravan is likely caused by an anomaly with this data set discussed in Section 7.3. It seems that the images are of a Dodge Caravan with different dimensions than those of the CAD model. Thus, the CAD model fits poorly to the images and the model-based camera calibrations are also affected.

Interestingly, the nearest mean classifier (Table 9.4) results in fewer misclassifications for many of the image sets. This outcome is contrary to the classification of the training data. The nearest mean classifier is a restricted form of the Bayesian classifier with Gaussian likelihoods. Clearly the Gaussian likelihoods are over fit to the training data. Over-fitting is not surprising given the very limited size of the training data. Also interesting is the difference in classification of the Volvo V70XC wagon. The Bayesian classifiers assign a label of SUV in most cases, while the nearest mean classifiers usually assign a label of four-door sedan. Neither of these categories is correct, but these are probably the two closest options, semantically, given the choices.

Classification results for these classifiers using the three composite vehicle classes are not interesting enough to show in detail. As expected, three-class results are even better than five-class results. In all 225 combinations of image sets, image scales, and vehicle models, there are only 8 misclassifications by the Bayesian classifier and 7 by the nearest mean classifier.

Next consider the recovered shapes from Section 7.3. These results use poor initialization in both shape and pose and a high initial uncertainty. The initial shape is the mean vehicle in all cases, and the initial pose is selected at random from a distribution centered around the true pose. As the initial pose shifts farther from the true pose, the fitting algorithm becomes more likely to get stuck in an incorrect local minimum. Using all random initial poses from Section 7.3 for recognition would mean that most of the shape parameters represent failed convergence attempts. Instead, recognition is applied to the results of the 40 closest poses to the true pose. That is, all initial poses within 0.2 meters and 6° of the correct pose. These initial conditions can still generate bad shape estimates but do not do so as frequently. Each of the image sets is initialized with the 40 poses and mean shape and optimized as in Section 7.3. Each estimated shape is classified and compared to the true class label. Recognition rates are averaged over vehicle image sets and plotted in Figure 9.4. The Volvo V70XC image set is removed from consideration in the five-class case because station wagon is not present in the class labels. The recognition rate corresponding to random chance is also plotted as a horizontal dashed line. Recognition rates below this line are no better than selecting a class label at random with equal probability.

The results in Figure 9.4 also show over fitting as seen in the previous results. Recognition

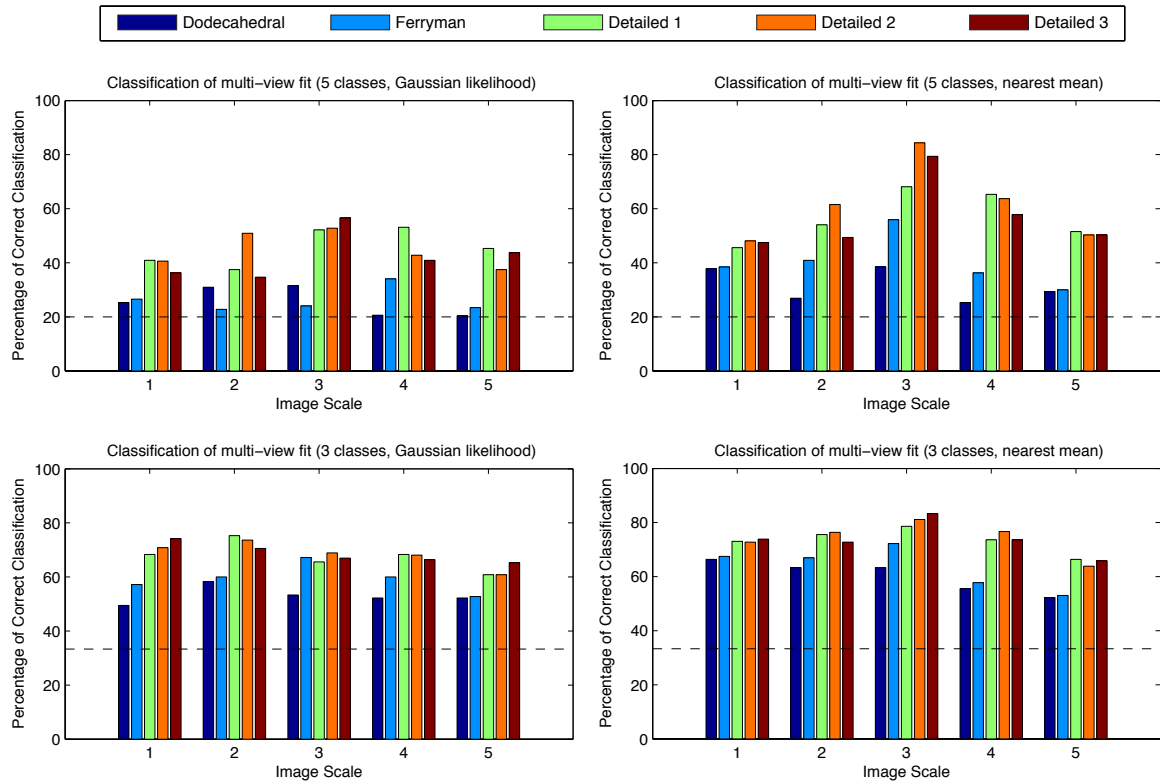


Figure 9.4: Success rates for classification of shapes recovered from multi-view fitting. Top plots use the five vehicle classes, bottom plots use three. Left plots use a Bayesian classifier with Gaussian likelihoods, right plots use a nearest mean classifier. Dashed lines indicate random chance rate (*i.e.* $1/5$ and $1/3$).

rates for nearest mean classification tend to be somewhat higher than rates for Gaussian likelihoods. Recognition rates are also higher for the proposed Detailed models than for Dodecahedral and Ferryman. Ferryman is typically better than Dodecahedral (with a couple of exceptions). However, there is no clear winner between the three resolutions of the Detailed model. Each of the Detailed models performs better than the others in some of the experiments. There seems to be a slight peak in recognition performance at the third image scale. This peak is especially noticeable when the nearest mean classifier is applied to five classes. Since the classifiers are the same for different scales, this peak must be caused by differences in the quality of shape recovery. A similar trend is faint, but apparent, in the pose convergence results of Section 7.3. The evidence suggests that the Detailed models are best suited to predicting vehicle edges at image resolutions found in image scale 3 (*i.e.* about 1 to 4 cm per pixel). At lower resolutions, image edges are not as precisely located, and some nearby edges merge or are lost. At higher resolutions, many additional outlier edges are present, and some edges may split into multiple parallel edges as image detail is added.

Finally, these recognition rates may not seem very impressive. In many cases they are not. Some the results using the Dodecahedral and Ferryman are barely better than chance for five-class recognition. At least the Detailed models are always quite a bit better than chance. In some cases,

the recognition rates exceed 80%, but there is still much room for improvement. Part of the problem comes from instances of poor shape estimation, and part of the problem comes from simple and over fit classifiers. Both areas could be improved if more training examples were available. To put these results in perspective, Ferryman *et al.* [30] report only a 48% recognition rate in their paper for a different three-class problem (sedan, hatchback, wagon). Koller *et al.* [60] only show three positive examples of successful recognition while tracking. Their setup uses five classes (sedan, hatchback, wagon, minibus, pickup), but two of their examples are sedans, one is a hatchback, and all three are bright white in color.

9.3.2 Classification from video

Now consider recognition from the shape recovered during simultaneous tracking and shape estimation. As discussed in Chapter 8, shape estimation continually evolves over the course of tracking. While shape may or may not stabilize, it never really converges to a fixed value. Thus, it makes sense to look at vehicle classification as it changes over time. One way to do this is to plot the posterior class probabilities (Equation (9.3)) for each class as a function of frame number. At each frame, the posterior probabilities of all classes are between zero and one and sum to one. Unfortunately, such plots are not always very informative. The top plots of Figure 9.5 show examples of actual posterior probabilities for the Gaussian likelihood and nearest mean classifiers. The Gaussian likelihoods are sharply peaked around the class means, so one class always has a posterior probability of almost one while the rest are almost zero. When shape crosses a decision boundary, the probabilities shift abruptly. The nearest mean classifier has the opposite problem. Distance to mean is mapped to the Bayesian framework assuming an identity covariance matrix on all classes. These Gaussians are relatively wide. As a result, all classes have almost equal posterior probability.

Recall that the decision boundaries between vehicle classes do not change with an arbitrary scaling of the covariance matrices applied equally to all classes. One can scale all the covariance matrices such that the resulting plots are more informative, and the resulting classifications are still the same. In the end, the recognition algorithm only cares about finding the class with the maximum posterior probability. Scaling does not change the end result, but produces a plot that better separates the probabilities to illustrate ranking. The bottom plots in Figure 9.5 show the same classification results plotted with scaled covariances. Gaussian likelihood covariances are all multiplied by a factor of 20 and nearest mean covariances are all divided by a factor of 20. The same scale factors are applied to all results in the rest of this section. The ordering of the classes is now more clear, and one can see by comparison to the top plots that the resulting decisions are exactly the same.

Those scaled plots in Figure 9.5 happen to show the five-class posterior probabilities for tracking the white PT Cruiser (from Figure 8.5) using the Detailed 3 model. The Bayesian classifier with Gaussian likelihoods selects a label of four-door sedan for most of the sequence. SUV is the second best choice and overtakes four-door sedan for a few frames. Two-door sedan is a distant third choice and both pickup and minivan are not even in the running. On the other hand, the nearest mean

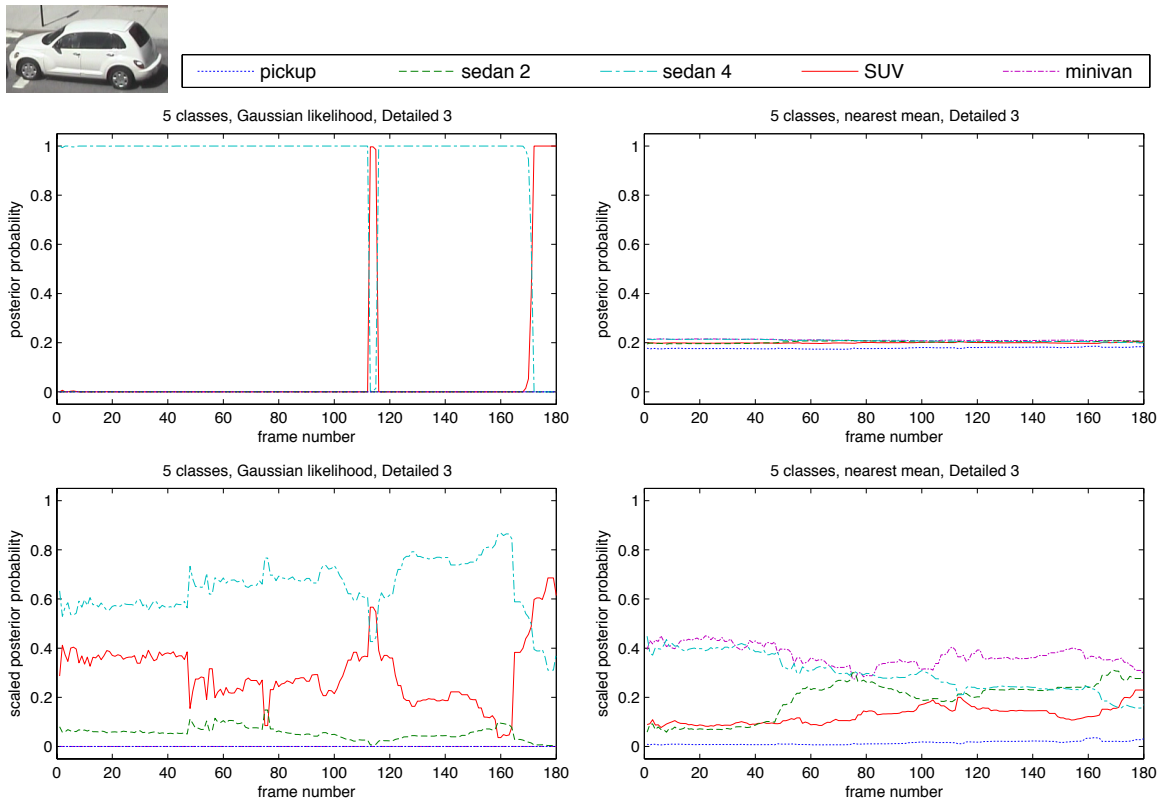


Figure 9.5: Scaling of posterior probabilities for visualization. Actual posterior probabilities are on the top, and scaled versions are on the bottom. The plots classify the white PT Cruiser (from Figure 8.5) using the Detailed 3 model and five classes.

classifier chooses the minivan class, although the four-door sedan is a close second and takes the lead for a few frames. This raises an obvious question: what is the correct class of a PT Cruiser? The answer is not simple and illustrates some of the problems in the very definition of the generic vehicle class recognition problem. Not every vehicle fits neatly into one of a small number of categories. The correct answer, according to the E3D classification, is four-door sedan. However, it is certainly an outlier compared to other sedans. Station wagon would probably be more accurate, but the shape is very not similar to other wagons either. A PT Cruiser is the size of a sedan, but with shape features similar to a small SUV or minivan. The classifiers are trained to recognize a PT Cruiser as a four-door sedan. So technically, the Gaussian likelihoods correctly classify the vehicle in most frames, and the nearest mean mistakenly chooses minivan most of the time. Yet, it is hard to argue that a PT Cruiser looks more like a sedan than a minivan. Figure 9.6 shows the same classification results when considering the three-class problem. With three classes, both classifiers oscillate between sedan and SUV/minivan, the pickup class is not even close to consideration. This ambiguous result seems appropriate for this particular vehicle. Recognition results are similar for the red PT Cruiser in aerial view, and the simpler models also show similar confusion between sedan, SUV, and minivan classes.

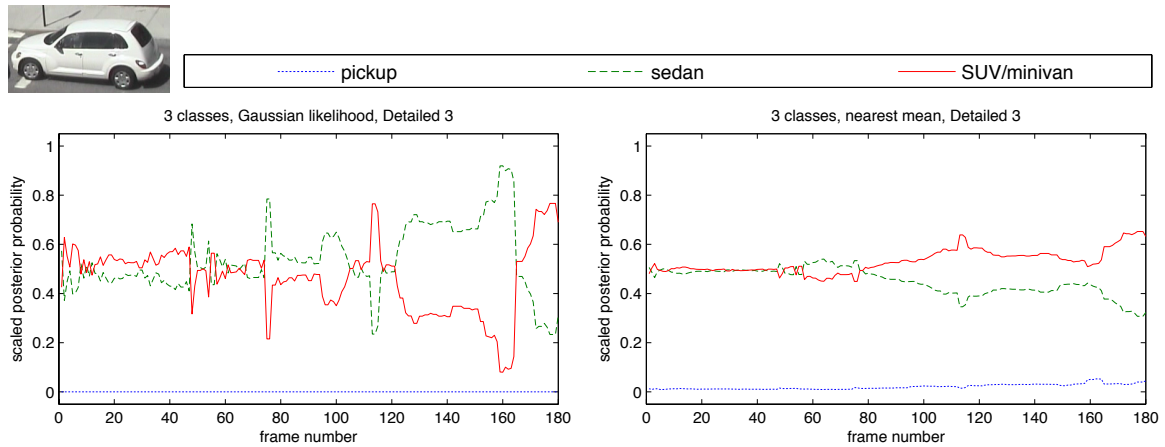


Figure 9.6: Three-way classification of the white PT Cruiser from tracking results. Classification oscillates between sedan and SUV/minivan since this vehicle shape is a little of both.

Other vehicles do fit well into the defined vehicle classes. For example, the black SUV from Figure 8.7 is definitely in the SUV class. Figure 9.7 plots the (scaled) posterior probabilities for both classifiers and for the Detailed 3 and Ferryman models. Recall that the SUV passes through a poorly constrained viewing direction around frame 120, and shape alignment is lost after that time. Both classifiers correctly classify the SUV for the first half of the sequence using the Detail 3 model. Classification is strongly determined, with little ambiguity between other classes. After tracking fails, the model is classified as a four-door sedan. One can verify in Figure 8.7 that the failed shape at the end of the sequence does indeed look like a four-door sedan.

The same experiment run with the Ferryman model does not produce such a clear classification. There is more ambiguity between the SUV and four-door sedan classes early in the sequence for the Gaussian likelihoods. Also, the remaining three class each have significantly higher probabilities at various points in the sequence. The Ferryman model with the nearest mean classifier does well at the beginning but gets confused with the minivan class toward the middle of the sequence. Some of the ambiguity is reduced by switched to three vehicle classes. These results are shown in Figure 9.8. Notably, the Ferryman model does almost as well as Detailed 3 using the nearest mean classifier. However, Detailed 3 hangs onto the correct classification longer while passing through the poorly constrained viewpoint. Detailed 3 is still clearly better with Gaussian likelihoods. The Ferryman model continues to confuse the SUV with a sedan early in the sequence. Overall, recognition is quite good for this challenging sequence when using the Detailed 3 model and mediocre with the Ferryman model.

Another challenging example is the red, two-door sedan from Figure 8.8. Not only is this a dark vehicle, but it passes through a poorly constrained viewing direction, and the difference between two- and four-door sedan is very subtle. Figure 9.9 plots the (scaled) posterior probabilities for all five vehicle models and for both classifiers. This vehicle passes through the poorly constrained view around frame 135. After this time, the models remain mostly aligned, but there is some misalignment

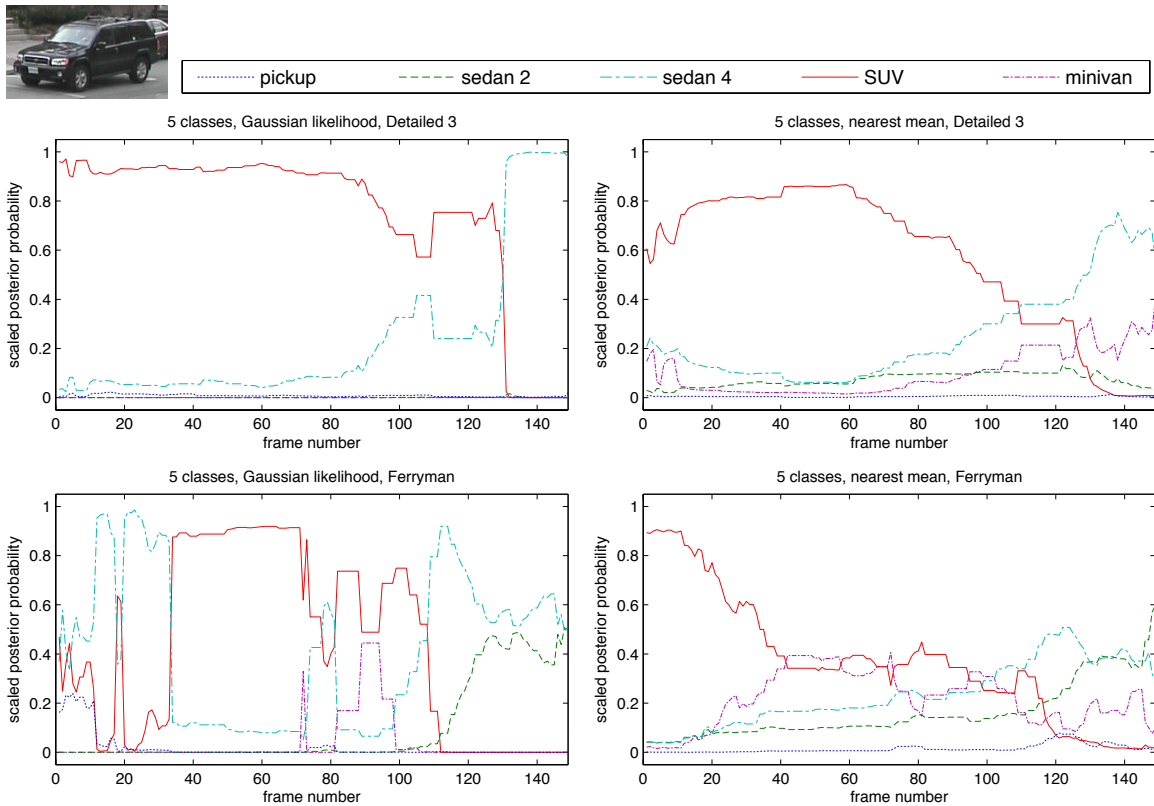


Figure 9.7: Classification of the black SUV from tracking results (Figure 8.7) using the Detailed 3 and Ferryman models with five classes.

in both the front of the model and the rear window. All three Detailed models, perform quite well under both classifiers. The primary confusion is between two- and four-door sedans, as expected. Before frame 135, the classification with the Detailed models is almost always correct. Detailed 3 has the most confidence in its choice of two-door sedan. After frame 135, the Detailed models are incorrectly stretched in length by a small amount. This shifts the classification to four-door sedan. However, the nearest mean classifier tends to keep the correct classification longer than the Bayesian classifier with Gaussian likelihoods.

Ferryman and Dodecahedral models do not perform nearly as well for recognition of the red, two-door sedan. The Ferryman model performs reasonably for the nearest mean classifier, but it erroneously selects SUV for the majority of frames when using the Gaussian likelihoods. The Dodecahedral models does even worse. It rarely selects the correct class when using either of the classifiers. Ironically, the only frames that are correctly classified are those that occur after the track is lost (around frame 210). Using the Bayesian classifier with Gaussian likelihoods, the Dodecahedral model selects four-door sedan for most of the sequence and SUV near the end. Using the nearest mean classifier, the Dodecahedral model selects minivan for almost the entire sequence.

The crossover SUV from Figure 8.9 is another example of a vehicle that does not fit neatly into one of the vehicle classes. Although, the crossover SUV is not as ambiguous as the PT Cruiser. It

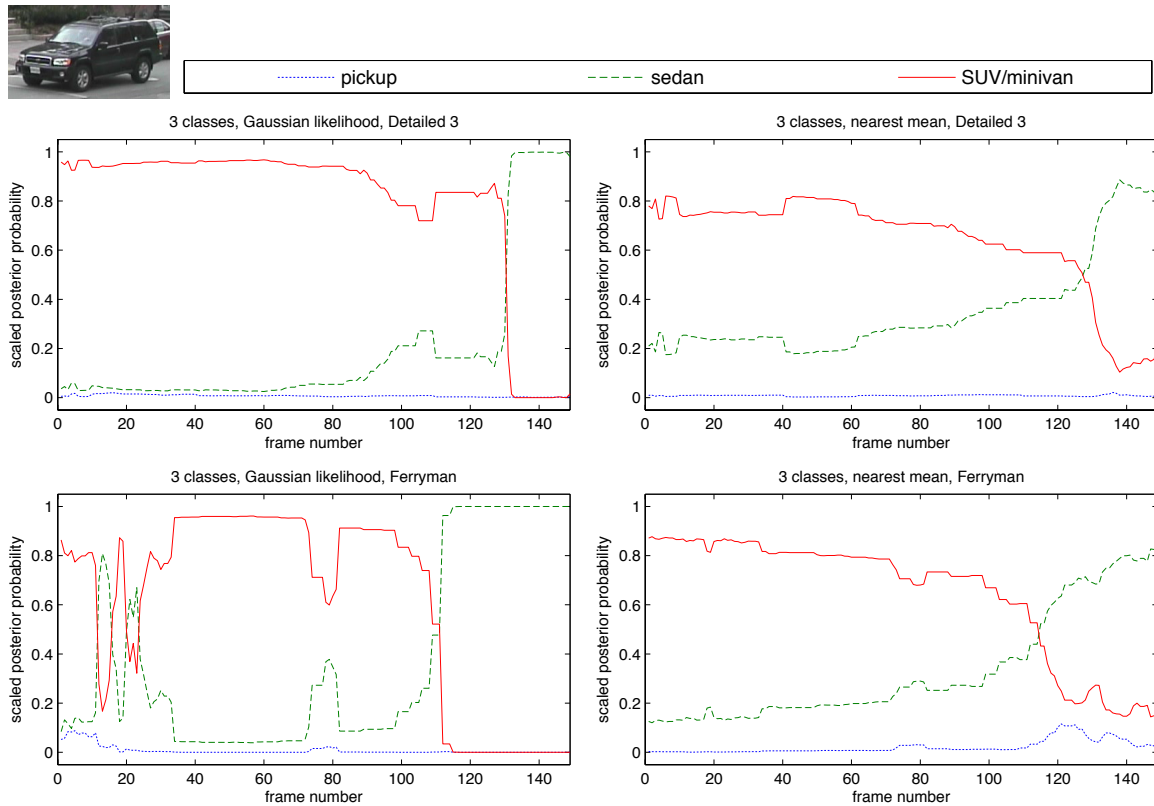


Figure 9.8: Classification of the black SUV from tracking results (Figure 8.7) using the Detailed 3 and Ferryman models with three classes.

seems clear that SUV is the most appropriate class. Despite that, this vehicle still has a different shape from SUVs used in training the classifiers. Thus, this is another challenging recognition example given the available training data. Figure 9.10 plots the five-class posteriors for the Detailed 3 and Ferryman models using both classifiers. Tracking with Dodecahedral fails immediately, so recognition is not shown. In this example, both Detailed 3 and Ferryman perform comparably. In most frames, both models select the SUV class using the Bayesian classifier with Gaussian likelihoods. The four-door sedan is second most likely in most frames, probably because this SUV is so small. In several frames, four-door sedan overtakes SUV under the Ferryman model. Under the Detailed 3 model, four-door sedan never overtakes SUV, but the minivan class overtakes both of them for a few frames. All classes except pickup are in the running when using the nearest mean classifier. Yet, the nearest mean classifier chooses either SUV or minivan in almost all frames for both the Detailed 3 and Ferryman models.

Recognition results are quite a bit more stable for the crossover SUV when the number of vehicle classes is reduced to three. Figure 9.11 plots the three-class (scaled) posterior probabilities for the crossover SUV. With only three classes, the combined SUV/minivan class is selected reliably. In fact, using the Detailed 3 model, both classifiers select SUV/minivan in every single frame, and do so by a wide margin. The Ferryman results are almost as good, but the sedan class takes the lead at

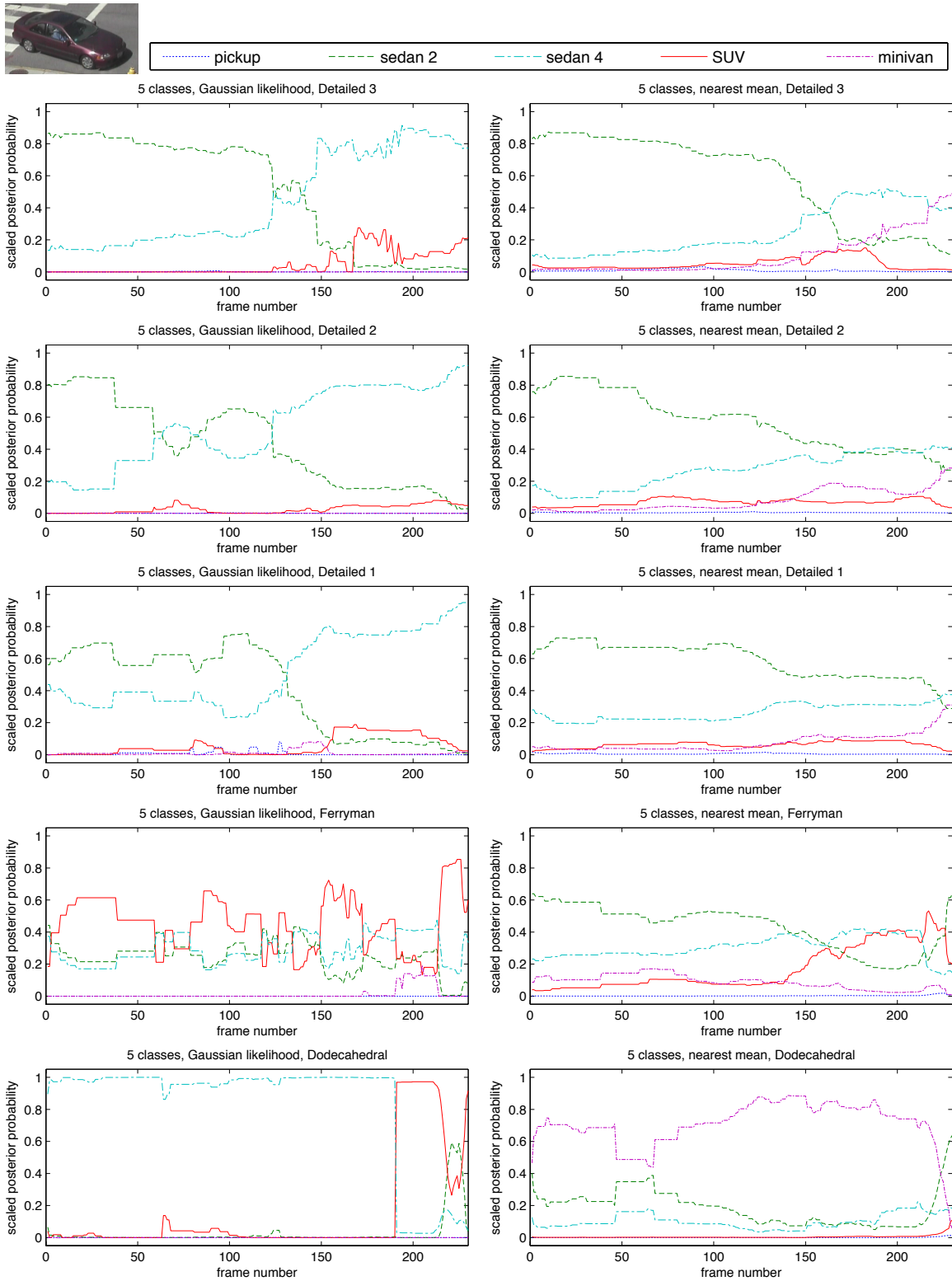


Figure 9.9: Classification of the red, two-door sedan from tracking results (Figure 8.8) using each of the vehicle models and five classes.

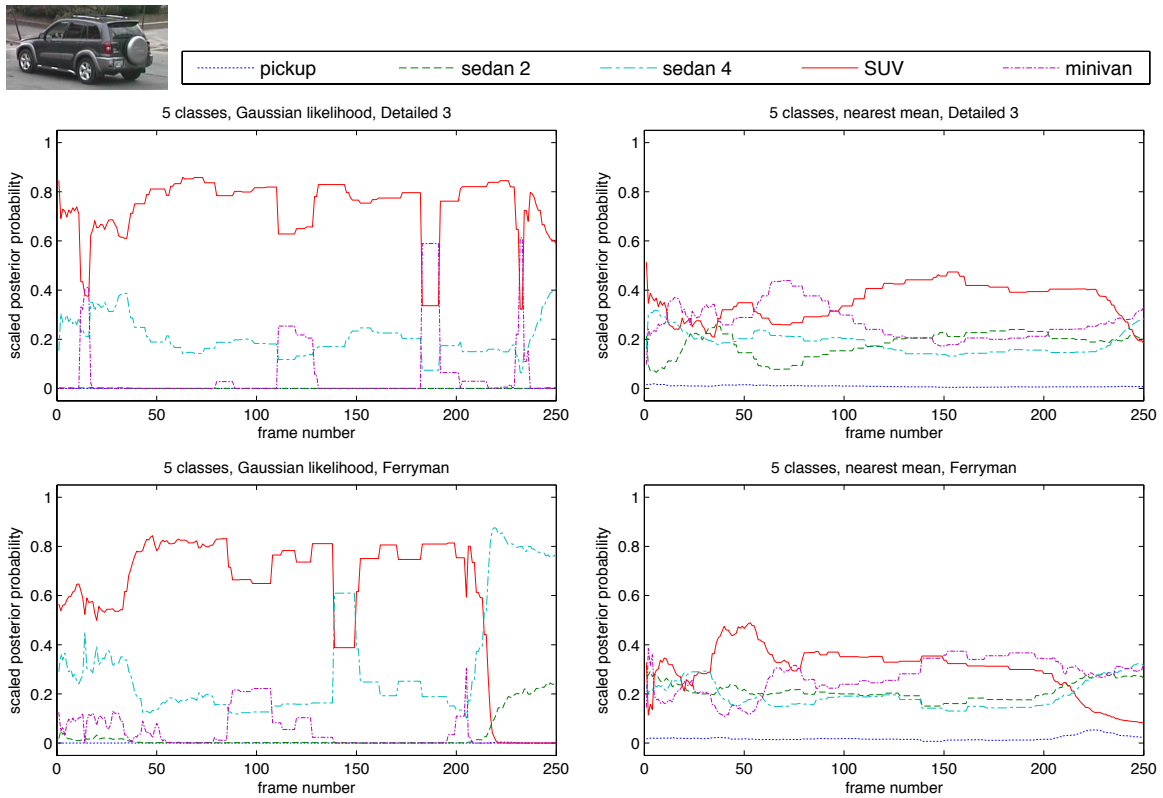


Figure 9.10: Classification of the crossover SUV from tracking results (Figure 8.9) using the Detailed 3 and Ferryman models and five classes.

the end of the sequence. Image resolution on the vehicle is extremely low at the end of the sequence, and the track is eventually lost. This explains why the SUV/minivan posterior probability drops off at the end.

Highly stable recognition results are not only obtainable using three classes. All of the results presented so far pertain to vehicles that are difficult to track, difficult to classify, or both. With easily tracked and classified vehicles, one can obtain stable classification results in the five-class problem. The easiest vehicles to track and recognize are those that have well defined class membership, high contrast edges, and no poorly constrained viewing conditions. Examples of such easy vehicles are given in Figure 8.10. Consider one of these vehicles, vehicle number 4 in Figure 8.10, a white sedan in aerial view. Recognition results for this vehicle are shown in Figure 9.12 for the five-class problem. The correct class label for this vehicle is four-door sedan. Four-door sedan is selected as the class label, often by a wide margin, in almost every frame for every classifier with every model. The only misclassifications come at the very end of the sequence when using the Gaussian likelihoods and Dodecahedral or Ferryman models. Tracking failure explains these two errors. The vehicle is almost completely occluded by the large tree at the end of the sequence, and both Dodecahedral and Ferryman lose track. These exceptions aside, the posterior probabilities are very consistent over the length of the track.

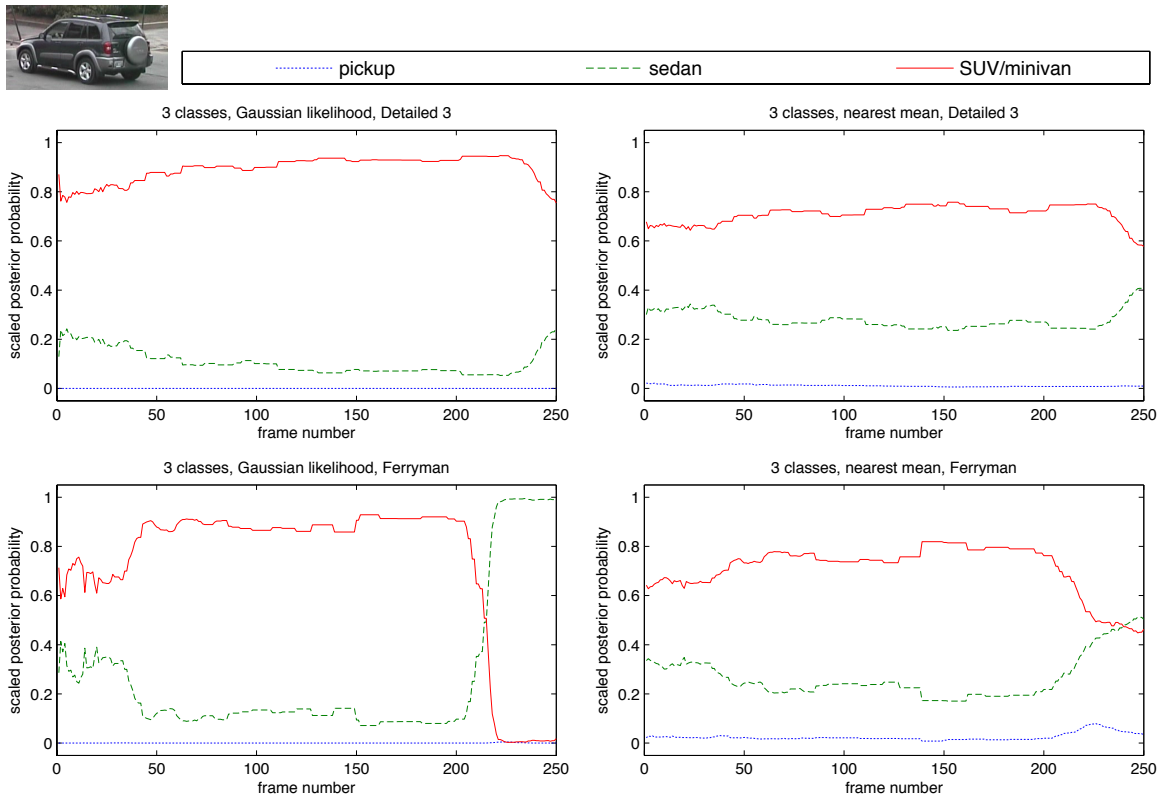


Figure 9.11: Classification of the crossover SUV from tracking results (Figure 8.9) using the Detailed 3 and Ferryman models and three classes.

Another vehicle that would be expected to give very stable results is the white pickup truck, vehicle number 3 in Figure 8.10. However, this example serves as a reminder that recognition is a process dependent on correct shape recovery. It is possible for tracking of pose to be successful without recovering the correct vehicle shape. This scenario plays out for the white pickup using the Dodecahedral and Ferryman models. Detailed 3 accurately fits the entire vehicle and recovers a pickup truck shape. Dodecahedral and Ferryman only fit to the front half of the vehicle and produce an SUV-like shape. Recognition results are shown in Figure 9.13 for five classes. Gaussian likelihoods using Detailed 3 select the correct label by a wide margin in every frame of the video. Nearest mean selects the correct label most of the time, but it has some confusion with the four-door sedan class. Switching to the Dodecahedral and Ferryman models, recognition fails at every frame. Not only that, but the pickup class is considered *least* likely in almost every frame. Both of these models using both classifiers tend to choose the SUV class instead.

In conclusion, these experiments have shown that class recognition is possible using shape recovered while simultaneously tracking a vehicle from a single camera. The quality of recognition depends both on the accuracy of fitting the 3-d model to images and on the ability of the shape space to separate the classes. The proposed Detailed models, especially Detailed 3, are superior on both fronts to the boxy Dodecahedral and Ferryman models. Furthermore, all experiments in this section,

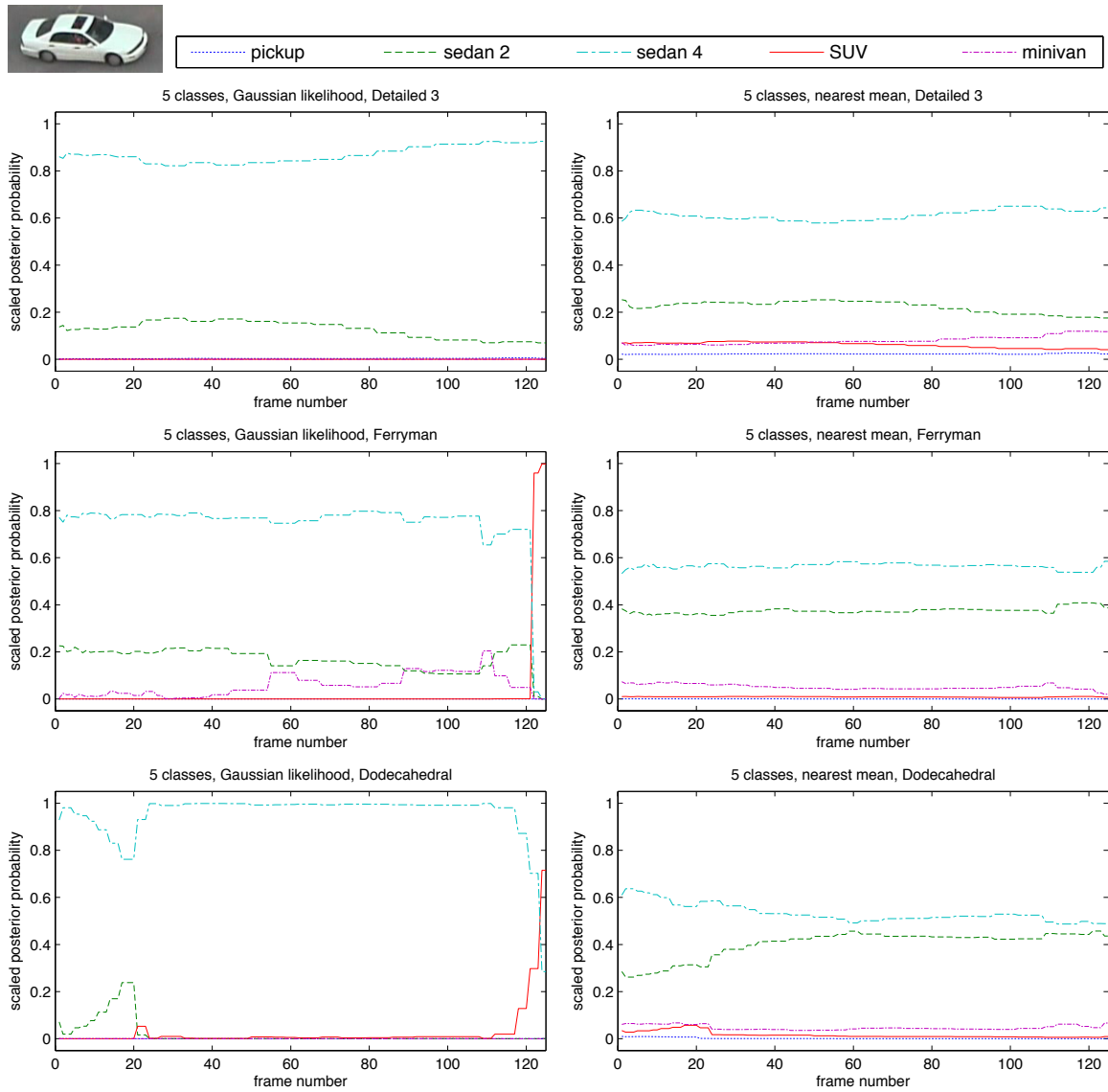


Figure 9.12: Classification of a white, four-door sedan from tracking results (Figure 8.10, vehicle number 4) using the Detailed 3, Ferryman, and Dodecahedral models and five classes.

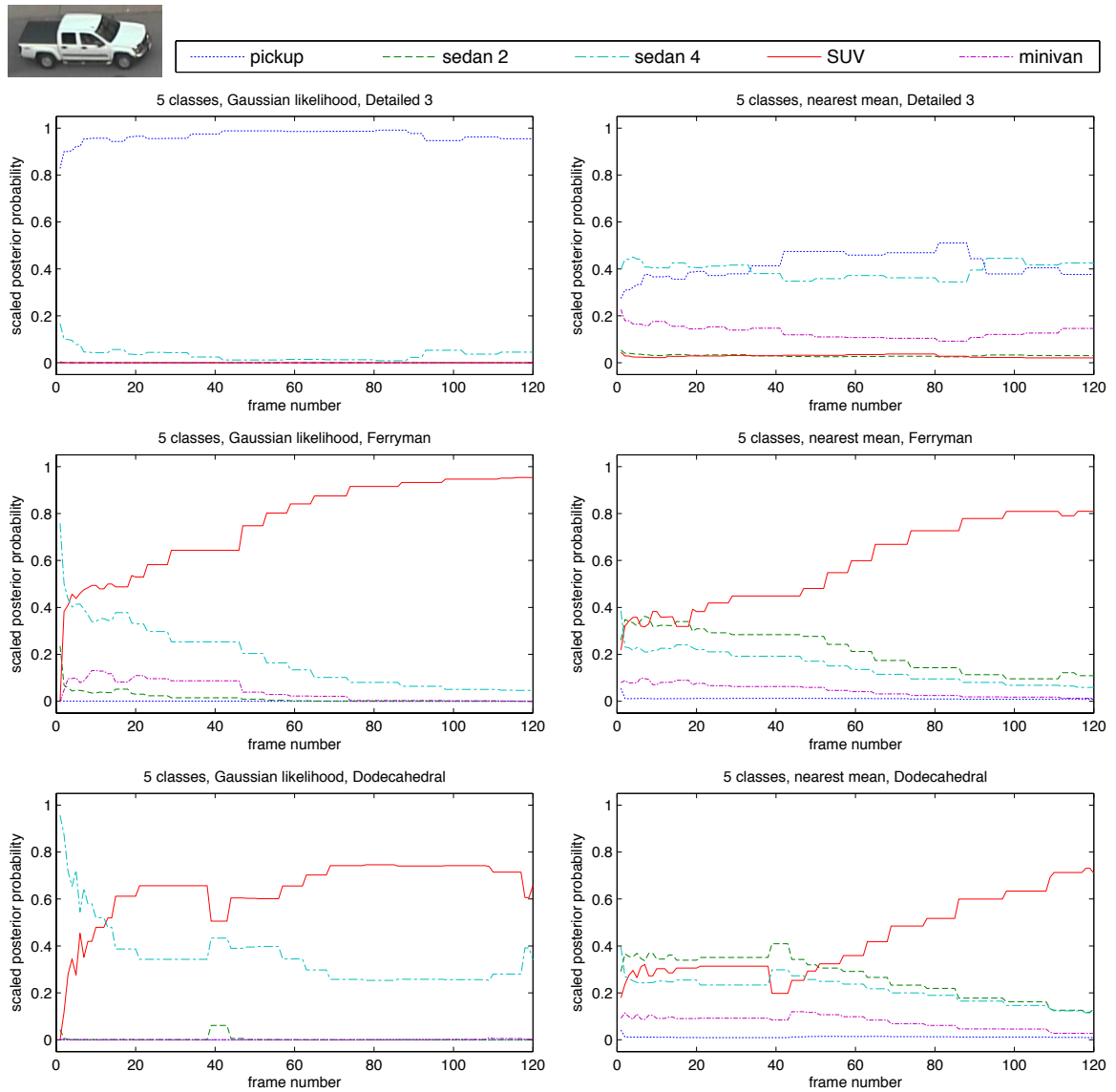


Figure 9.13: Classification of the white pickup truck from tracking results (from Figure 8.10, vehicle number 3) using the Detailed 3, Ferryman, and Dodecahedral models and five classes.

with the exception of the PT Cruiser, apply to vehicles that are not in the E3D training data. This shows that recognition does indeed generalize beyond the training data. The more general Bayesian classifier with Gaussian likelihoods usually outperforms the nearest mean classifier. Over fitting does not seem to be as big of an issue with regard to novel vehicle shapes. Recall from the previous section that the nearest mean classifier is more robust to shape estimation errors caused by poor shape initialization. Initialization in these tracking experiments is actually quite good, and shape variation is more due to the variety of vehicle shapes than errors in reconstruction. One can see, from Figure 9.9 for example, that when the shape alignment does become poor, the nearest mean classifier does better than Gaussian likelihoods.

Chapter 10

Conclusion

This thesis has addressed the automated vehicle surveillance problem. Specifically, it has framed the vehicle tracking and generic class recognition problems as by-products of dense 3-d shape recovery from images and video. 3-d shape was recovered using a deformable vehicle model by jointly estimating the parameters of shape and pose. The primary contribution of this thesis was the improved generic, deformable, 3-d vehicle model. Novel algorithms have been presented for learning the model shape from CAD models and estimating model parameters by fitting to images.

The proposed model is more detailed and predicts image edges more accurately than simpler models in the related work of Koller *et al.* [59] and Ferryman *et al.* [30]. It extends the concept of a generic vehicle model to nearly the level of detail found in rigid CAD models. The proposed vehicle model has been applied to the problems of shape recovery, tracking, and recognition. Experiments have shown that the proposed model outperforms the simple boxy vehicle models in all of these tasks. However, the experiments also show that there is much room for improvement.

10.1 Directions for future work

Automated vehicle surveillance can be improved by advancing several different aspects of the ideas described in this thesis. Obviously, the design of the vehicle model itself is a key contributing factor in the performance of shape estimation, tracking, and recognition. However, the way in which the prior model of shape is learned is also critical. Improving the method of fitting the model to images would further advance all three tasks. Finally, the tracking and recognition algorithms used in this thesis are not, by any means, state of the art. Applying more advanced algorithms would greatly benefit these tasks. Ideas for future work in each of these areas are discussed below.

10.1.1 Vehicle model design

It is important to remember that, while the model vertex positions are learned, the model topology is designed. The number of vertices, their semantic meaning, and their interconnectivity are all

manually determined. There is nothing wrong with using a designed model, but the design of this model is not optimized in any quantifiable way. Subjectively, the base mesh was designed to cover all critical shape features with fewest possible vertices connected in way that closely resembles a quadrilateral grid. Likewise, the parts were selected to cover the most commonly occurring, semantically meaningful, closed regions that occur on the training vehicles. Yet, these are all design decisions, and different designs may produce better or worse results.

For example, instead of a quadrilateral mesh, one could use a triangular mesh. In addition, quadrilateral subdivision could be replaced with a triangular subdivision scheme. It is not clear if triangular subdivision would produce better or worse results. Going one step further, it may be possible to incorporate some form of adaptive subdivision. Adaptive subdivision on a deformable mesh is somewhat tricky because moving vertex positions change the mesh curvature. Nonetheless, when the vertices are in good correspondence, some vertices are likely to have less curvature under all deformations of the model. Adaptive subdivision would need to take the range of deformations into account.

Related to adaptive subdivision is the concept of a model hierarchy. At higher mesh resolutions, some vehicle classes need more vertices in some areas than others. For example, it has not been possible to represent the bed of a pickup truck with the standard quadrilateral subdivision of the base mesh. Pickup truck shapes currently appear to have a covered bed in the deformable model. One solution is to have different subdivision schemes for different vehicle classes. A single mesh could cover all vehicles at the coarse mesh resolution, and then subdivide in different ways for each vehicle subclass. Generic fitting could start with the coarse model and then switch to an appropriate class model after the rough shape is established.

Mesh parts are also fair game for redesign. One simple option is to determine which parts make the most significant contribution to performance. Experimentally, one could try removing different combinations of parts and comparing the effects on shape estimation, tracking, and recognition. One could hypothesize that large parts with limited variation, like windows, are more useful than small parts with large variation, like headlights. Experiments in this thesis already suggest a few changes to part design. The middle and back side windows should probably be joined into a single part. Many vehicles do not have both windows, and rarely does the divider between these windows align with corresponding features. These windows also tend to overlap in the training data when a missing window is replaced by the mean shape. License plates would be better represented as a separate planar mesh of fixed size that is attached tangent to the vehicle surface at a point specified in the texture space. License plates are known to have a fixed size on all vehicles. Representing their four corners in the texture space allows the plates to deform slightly and bend along curved vehicle surfaces.

Even the texture space itself is subject to redesign. The texture space projection of the mesh vertices was determined somewhat arbitrarily. The current projection (Figure 3.9) tries to keep the model symmetric in texture space and assigns relatively equal areas to the various mesh faces. However, it is likely that a different arrangement of mesh vertices in the texture space would have

an impacted on model fitting.

Finally, the detailed shape of the wheels has been somewhat neglected in the current model. There is no reason not to extend mesh subdivision to the wheel sub-meshes and also fit them to the CAD models of wheels. Along the same lines, one could consider adding additional sub-meshes representing side mirrors, luggage racks, spoilers, external spare tires, and other shape features. These extra surfaces would need to be optional with some mechanism for detecting their presence. It is unlikely that these additional surfaces would benefit the current fitting algorithm, but with an improved fitting algorithm these parts could help distinguish between a larger set of more subtly different vehicle classes.

10.1.2 Learning the vehicle model

In addition to the changes in the model itself, there are some improvements that could be made to the learning of a prior model of shape. The most obvious direction to pursue for improving the prior model is acquisition of more training data. Assuming a larger database of CAD models is obtained, extending the deformable models to the new CAD models need not require the same amount of manual effort. The existing deformable model can bootstrap the process. Given a new CAD model, one would still need to label corresponding parts and position the model in the canonical pose. However, the coarse 3-d shape should no longer require manual initialization. Instead, the 3-d shape could be initialized by fitting the deformable model to the CAD model. Such a fitting process is similar to image fitting, but much less ambiguous. Given an initial fit, the detailed shape is learned as before by shrink-wrapping. The resulting shape feeds back into the prior shape estimation.

A large set of CAD models may not be easy to obtain, since each CAD model itself takes a great deal of manual labor for someone to create. Instead, one could consider extending the model using alternative sources of training data. Again, the existing deformable model would help to bootstrap the process and make up for some of the inadequacies of other data sources. For example, one could consider learning new shapes by fitting the existing model to LIDAR scans of vehicles, or even multi-view image sets. Presently, automated learning from images would be difficult and would require supervision to help correct poor fitting results. Although, improvements to the fitting algorithm could make this option more feasible.

Along similar lines, one could extend the prior model using a much simpler set of measurements. Each year the Canadian government releases an updated database of measurements for all vehicles sold in Canada. The Canadian Vehicle Specifications (CVS) [20] contains 13 length measurements of each vehicle that directly relate to a subset of the 30 measurements proposed in Section 5.2.1. These measurements alone are not enough to build a detailed vehicle model, but the measurements could be transferred to the existing vehicle shape space. Each measurement places constraints on the relative positions of various mesh vertices. One could estimate the point in the current shape space that best addresses these constraints. These recovered shape points would not add anything to the expressiveness of the shape subspace; however, they would give a better representation of the distribution of real vehicles in that space.

Given a larger collection of training examples in shape space, one might also consider using a more complex prior distribution model. PCA could still be applied to the space of all vehicles to reduce the dimensionality, but the single Gaussian distribution it induces is probably not the best prior model of shape. It might be better to fit a Gaussian model to each vehicle class, and then represent shape prior distribution as a mixture of Gaussians. Such a multimodal distribution would help reduce the likelihood of estimating a shape that is part way between two classes and not likely in reality.

Finally, there are some improvements that could be made to learning process even with the existing training examples. Recall that during fitting, each vertex is mapped to a nearby point on the vehicle surface. The shrink-wrap process only constrains the vertices along the direction normal to the vehicle surface. Position of the vertices tangent to the surface is determined only by initial conditions. Most vertices could be moved slightly in the tangent plane with little effect on the overall shape. Therefore, some of the variation in the PCA shape space accounts for vertex variations tangent to the surface. These tangent variations are essentially correspondence errors and may bias the true modes of variation between vehicles. One way to improve point correspondences is to truncate the PCA shape and shrink-wrap again. That is, project each recovered vehicle into a low dimensional shape subspace (*e.g.* the top five PCA parameters) and use that shape as initialization to compute shrink-wrapping again. This process is straight forward for the 3-d mesh vertices. Figure 10.1 shows preliminary results indicating that much of the variation beyond five PCA components can be reduced by this method. However, to do this correctly, one must also account for the deformation of parts. The 3-d vertex deformations should be repositioned such that the parts contain less variation in texture space. Properly handling both surface and part vertices jointly requires further investigation.

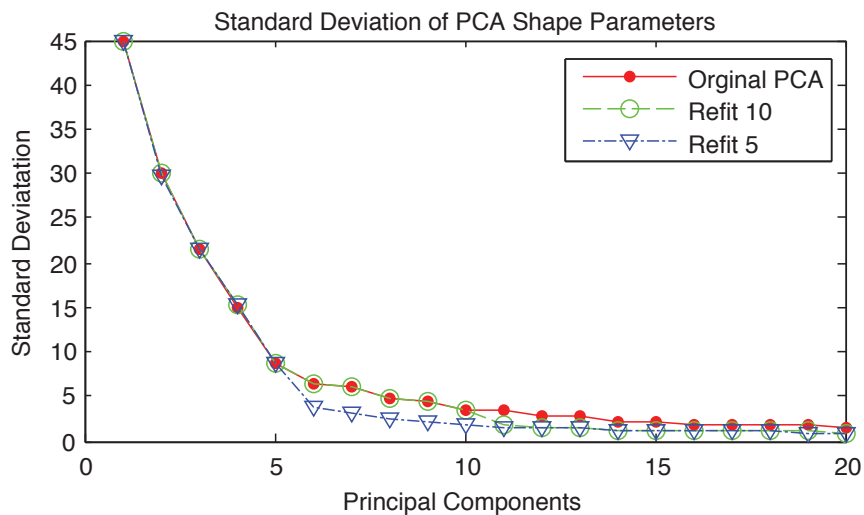


Figure 10.1: Relearning shape with initialization from truncated PCA projections. Standard deviations of the original PCA space (3-d mesh only, no parts) are compared to those after truncating shapes to 5-d and 10-d subspaces and refitting with the truncated models as initialization.

10.1.3 Fitting to images

Even with improvements to the vehicle model and prior shape distribution. The current algorithm for fitting to images is still likely to fail in certain challenging cases. It has been shown that some cases are difficult for the fitting algorithm, particularly dark vehicles with low edge contrast, poor initialization, and extremely low image resolution. Fitting to a single view is also challenging, especially if the view does not show much 3-d relief (like frontal views or top down views). It may be possible to improve the fitting algorithm to better handle some of these cases.

One interesting area of exploration is the choice of edges, both on the model and in the image. As already suggested, one could determine which parts predict the most useful edges for fitting. Alternatively, some optional parts (*e.g.* sunroof) could be enabled or disabled dynamically during fitting based on image evidence. This could happen continuously by applying additional weights to parts as a whole. There are also other 3-d surface edges to consider. For example, suggestive contours [24] are ridges that are *almost* occluding contours from the current view direction. They are local minima in $\mathbf{n} \cdot \mathbf{v}$ (refer to Section 3.4). Suggestive contours might predict some edges that are otherwise missed. Similarly, one could consider high curvature ridges on the surface.

A bigger problem is that there are sometimes just not enough strong edges detected on a dark vehicle. The Canny edge detector on grayscale images is not sufficient to exploit all available information. In some cases, an important edge may be the boundary between regions of different color but similar intensity. An edge detector that accounts for color might be helpful. Along the same lines, other types of edge detectors might be useful, such as those that define edges between different textures. Vehicles have little inherent texture, but some surfaces reflect texture from the background differently.

Of course, edges are not the only game in town. Regions and point features may augment edges in the fitting process. The deformable model was specifically designed such that each part is the boundary of a region, but these regions have not yet been exploited. With each model hypothesis, the parts predict regions in the image. One could learn a statistical appearance model for the colors and intensities in these regions. Such a model might help to better determine which image edges represent true part boundaries. For example, taillights are very likely to appear with the color red on all vehicles.

Point features could also be useful in applying additional constraints. Liebelt *et al.* [70] have shown that it is possible to use code book matching of feature point descriptors to detect and initialize the 3-d pose of vehicles. While such initialization itself would be useful, the same matched points could also provide constraints on the fitting process. Liebelt *et al.* also train their detector on CAD models, so each detected point corresponds to a known 3-d point on the vehicle surface.

Finally, it would be worth taking a more in-depth look at the efficiency and performance of different approaches to scale space discussed in Section 4.3.4. The construction of a scale space pyramid is avoided in favor a dynamically scaling the correspondence search at a fixed image resolution. It would be interesting to examine the difference in speed of each approach and also see if the results differ significantly.

10.1.4 Tracking

Vehicle tracking is quite successful, but there are a few conditions, like occlusion and poorly constrained views, that tend to disrupt a successful track. It would be interesting and very straight forward to try tracking in multi-view video. That is, use two (or more) synchronized video cameras mounted far apart, such as on opposite sides of a street. From the results of multi-view still image experiments, it seems highly likely that multi-view video, even with only two views, would greatly improve stability where the vehicle is the poorly constrained from a single view. However, a synchronized and calibrated multi-camera system is much more complex to install and maintain in practice, so it is still important to focus on advances in the single view case.

When viewing a vehicle from a single, poorly constrained direction, recursive shape estimation is not enough avoid shape degradation. One way of avoiding shape degradation is to abandon the purely recursive framework and find a compromise between the online and offline algorithms. For example, one could retain a limited number key frames from earlier in the sequence, and run a batch optimization on multiple frames at once. Frames could be retained whenever the vehicle changes significantly in pose or scale from previous key frames. The challenges of such a method would be selecting appropriate frames to constrain the optimization. A larger number of frames provides more constraints, but a smaller number of frames makes online processes more feasible.

The tracker could also consider other sources of information, like the background model, in constraining the shape and pose. Preliminary experiments examined the use of foreground region boundaries as additional tracking constraints. The same boundary constraints used in track initialization were added into the tracking optimization as additional terms. Foreground regions did help in some cases with degenerate viewpoints, but in several other cases the tracking performance was degraded. The foreground boundary is not reliable enough to use in every frame, especially near occlusions. Yet there might be a better way to make use of this information instead of relying on boundary alignment.

Another area that could use some improvement is track initialization. Velocity initialization appears to be the culprit in several of the observed tracking failures. It might be necessary to extend velocity initialization to an estimate of more than two frames. Another alternative is to use the optical flow vectors to estimate the world space velocities. Other than occasional velocity failures, the current method of track initialization works quite well. However, it requires a stationary camera for background modeling and a completely unoccluded view of the vehicle. Such constraints are not always practical in real situations. Vehicles often appear in higher volumes and in close proximity. It would be interesting to test other 3-d vehicle detection algorithms [70, 101, 83] that do not have these requirements.

A few other aspects of real world vehicle tracking have not been addressed with the proposed model. For example, a 3-d vehicle tracker is well suited to handling occlusions between vehicles. Such occlusions are easily predicted by the simultaneous projection of multiple vehicle models. Occlusion predictions could limit hypothesized edges to those that are not occluded and thereby avoid mismatching edges in these circumstances. Also, non-planar roads have not been considered.

Many roads are not well modeled by a single plane. One could consider a more complex ground model, or possibly even estimate the ground plane dynamically using vehicle alignment.

Finally, it would be interesting to test the benefits of more complex motion models. The simplest improvement might be the addition of accelerations to the motion model. Constant velocity might be a good model in highway traffic, but at intersections acceleration is significant. A constant acceleration model might lead to better predictions in these cases. Advancing the model further, one could abandon the circular arc model in favor of one that takes into account the motion of a four-wheel, two-axle vehicle. The position of all wheels is accounted for in the shape model, and one could even model the changing orientation of the wheels relative to the vehicle body.

10.1.5 Recognition

Vehicle recognition has the largest opportunity for advancement. The simple recognition experiments presented in this thesis are designed to show that recognition is indeed possible in the vehicle shape space and that the proposed vehicle model is better suited for recognition than previous models. However, the classifiers are quite simple and the training data is quite limited. The first step required in improving vehicle recognition is to obtain more training data. Additional training data could come from more CAD models or other sources as discussed in Section 10.1.2. Additional training data would help to reduce over fitting and allow the classifiers to generalize better to new examples. It may also be helpful to train classifiers on estimated 3-d shape from labeled vehicle images instead of on pristine shapes fit to CAD models.

One simple way of adjusting the current classifiers to better handle the limited training data is to apply the shrinkage approach to covariance estimation. Shrinkage estimators are essentially another name for the regularization techniques already used in this thesis. The standard covariance estimate is combined with prior covariance such the identity matrix. The result is something part way between the standard Gaussian likelihoods and nearest mean classifier. Over fitting can be reduced without completely assuming independence of the dimensions.

Classification with Gaussian models really only scratches the surface of possible recognition techniques. A whole field of research addresses the general problem of learning classifiers for points in an n -d space. Some of the relevant machine learning techniques include support vector machines, decision trees, and neural networks. Once the observations are mapped into a meaningful low-dimensional space, an enormous number of classification techniques are applicable. A thorough examination of various recognition algorithms is beyond the scope of this thesis.

Finally, regardless of the choice of classifier and number training examples, one must carefully consider the selection of vehicle classes. There is no universal list of vehicle classes. Some vehicle classes like van, hatchback, and crossover SUV are not considered in this thesis (training examples are not available). Some vehicle classes are also more closely related than others. A hierarchy of vehicle classes might be of some use. There will always be a few unique vehicles with shape that does not fit well into any class. The Volkswagen Beetle and PT Cruiser are popular examples that have been studied in this thesis. If recognition is good enough, the best vehicle classes might be the

manufacturer's make, model, and year. Each vehicle shape becomes its own class. The down side is that some such classes would be nearly impossible to distinguish.

10.2 Final thoughts

This thesis relies on fitting a deformable, 3-d vehicle model to drive various surveillance tasks. A similar model-based vehicle surveillance framework was popular in the computer vision community in the early 1990's. It has since fallen out of favor. Sparse feature-based methods are now in vogue for vehicles and many other objects. However, model-based shape estimation, tracking, and recognition have remained popular in areas of computer vision related to the human face and human body. Human 3-d models have continued to advance in complexity and accuracy. Advancement has been driven, in large part, by computer graphics and the needs of the entertainment industry. Meanwhile, generic 3-d models for vehicles have been left in the dust.

Computer vision research on 3-d model-based methods frequently borrows from the products of the computer graphics community. Deformable human models are useful in graphics research because human shapes vary continuously, and humans actually deform while in motion. Deformable vehicle models are rarely useful in graphics research because vehicles are manufactured to precise specifications and are rigid. One can simply build a CAD model for each vehicle one wishes to render. Thus, model-based computer vision has been helped along for human shape estimation, but vision researchers are left with a collection of rigid models when dealing with vehicles.

Old ideas about deformable vehicle models have been revisited in this thesis with an updated look at the choice of 3-d model. Vehicles may not continuously deform in shape in the real world, but a deformable model is still useful in recovering shape when shape is unknown. Such deformable models need not be simple boxy shapes just because they are easy to create and deform. The models also need not be rigid in order to capture the fine details of shape. Detailed models may be available from the graphics community, but that does not mean they are the best choice for computer vision problems. Sometimes it is best to build a 3-d model specifically for vision tasks. Yet models for computer vision can still learn from the graphics models to help get beyond primitive shapes.

The author hopes that future vehicle surveillance work will continue to examine detailed, deformable, 3-d models that are suitable for computer vision. Much effort has been made to create a vehicle model that is as detailed as possible while also being as generic as possible. However, the propose model is still just a proof of concept. There are many way of improving, not only the model, but also the algorithms used in shape learning, fitting to images, tracking, and recognition. This thesis only scratches the surface. Hopefully, the ideas presented here have laid the groundwork for more advanced models and algorithms to come.

Appendix A

IMLS Line Integrals

In Section 5.2.2 the evaluation of the IMLS implicit surface function $f(\mathbf{x})$ involves some area integrals given in Equation (5.7). There is no known closed-form expression for these area integrals, but there is a closed-form solution to the line integrals. In practice, the area integral is written as a double integral in barycentric coordinates of a triangle. The outer integral is computed with numerical quadrature, and the inner line integral is computed with the analytic expression derived here.

Assume that the line segment of interest is specified by a starting point, $\mathbf{p}_0$, and an ending point, $\mathbf{p}_1$. Associated with these points are target function values ϕ_0 and ϕ_1 . The ϕ at any point along the line segment is linearly interpolated. If s is the interpolation parameter in the range $[0, 1]$ then $\phi(s) = \phi_0 + s(\phi_1 - \phi_0)$. Likewise, the position of any point along the segment can be expressed as a function of s . That is, $\mathbf{p}(s) = \mathbf{p}_0 + s(\mathbf{p}_1 - \mathbf{p}_0)$. The normal, $\mathbf{n}$, to the triangle containing the line segment is constant over s . Using these terms and changing the variable of integration to s , the integrals in question are

$$l(\mathbf{x}) = \int_{\mathbf{p}_0}^{\mathbf{p}_1} w^2(\mathbf{x}, \mathbf{p}) d\mathbf{p} = \int_{\mathbf{p}_0}^{\mathbf{p}_1} \frac{1}{(\|\mathbf{x} - \mathbf{p}\|^2 + \epsilon^2)^2} d\mathbf{p} \quad (\text{A.1})$$

$$= \int_0^1 \frac{1}{(\|\mathbf{x} - \mathbf{p}(s)\|^2 + \epsilon^2)^2} \|\mathbf{p}_1 - \mathbf{p}_0\| ds. \quad (\text{A.2})$$

$$L(\mathbf{x}) = \int_{\mathbf{p}_0}^{\mathbf{p}_1} w^2(\mathbf{x}, \mathbf{p}) S(\mathbf{x}, \mathbf{p}) d\mathbf{p} = \int_{\mathbf{p}_0}^{\mathbf{p}_1} \frac{\phi(\mathbf{p}) + (\mathbf{x} - \mathbf{p}_0)^\top \mathbf{n}}{(\|\mathbf{x} - \mathbf{p}\|^2 + \epsilon^2)^2} d\mathbf{p} \quad (\text{A.3})$$

$$= \int_0^1 \frac{\phi(s) + (\mathbf{x} - \mathbf{p}_0)^\top \mathbf{n}}{(\|\mathbf{x} - \mathbf{p}(s)\|^2 + \epsilon^2)^2} \|\mathbf{p}_1 - \mathbf{p}_0\| ds \quad (\text{A.4})$$

After factoring these expressions in terms of s and pulling constants out of the integration, $l(\mathbf{x})$ and $L(\mathbf{x})$ can both be written as a linear combination of two integral forms:

$$I_{0,2} = \int_0^1 \frac{1}{((s + k_1)^2 + k_2)^2} ds \quad \text{and} \quad I_{1,2} = \int_0^1 \frac{s}{((s + k_1)^2 + k_2)^2} ds. \quad (\text{A.5})$$

Specifically,

$$l(\mathbf{x}) = \frac{I_{0,2}}{\|\mathbf{p}_1 - \mathbf{p}_0\|^3} \quad \text{and} \quad L(\mathbf{x}) = \frac{(\phi_0 + (\mathbf{x} - \mathbf{p}_0)^\top \mathbf{n})I_{0,2} + (\phi_1 - \phi_0)I_{1,2}}{\|\mathbf{p}_1 - \mathbf{p}_0\|^3} \quad (\text{A.6})$$

with

$$k_1 = \frac{-(\mathbf{p}_1 - \mathbf{p}_0)^\top (\mathbf{x} - \mathbf{p}_0)}{\|\mathbf{p}_1 - \mathbf{p}_0\|^2} \quad \text{and} \quad k_2 = \frac{\|\mathbf{x} - \mathbf{p}_0\|^2}{\|\mathbf{p}_1 - \mathbf{p}_0\|^2} - k_1^2 \quad (\text{A.7})$$

All that remains is to provide the analytic solutions to the integrals in (A.5). Shen *et al.* provide these solutions in the appendix of [93]. Unfortunately, the solutions provided by Shen *et al.* are incorrect. Their analytic expressions do not evaluate to the correct values as verified by numerical integration. The correct analytic expressions are derived below. Shen *et al.* define their interval of integration as $[0, a]$. This a parameter is redundant since a can always map to 1 with an appropriate change of variables. The correct analytic expressions for the interval $[0, 1]$ are

$$I_{0,2} = \int_0^1 \frac{1}{((s + k_1)^2 + k_2)^2} ds = \frac{-\beta}{2(k_2)^{\frac{3}{2}}} + \frac{k_2 - k_1(k_1 + 1)}{2k_2(k_1^2 + k_2)((k_1 + 1)^2 + k_2)} \quad (\text{A.8})$$

$$I_{1,2} = \int_0^1 \frac{s}{((s + k_1)^2 + k_2)^2} ds = \frac{\beta k_1}{2(k_2)^{\frac{3}{2}}} + \frac{k_1 + 1}{2k_2((k_1 + 1)^2 + k_2)} \quad (\text{A.9})$$

where

$$\beta = \left(\tan^{-1} \left(\frac{k_1}{\sqrt{k_2}} \right) - \tan^{-1} \left(\frac{k_1 + 1}{\sqrt{k_2}} \right) \right) \quad (\text{A.10})$$

and this β is the same as the β in the appendix of [93].

Computing the gradient of the field reuses much of the computation already done while computing the field itself. Using the quotient rule

$$\nabla f(\mathbf{x}) = \frac{d}{d\mathbf{x}} \frac{\sum_{i=1}^M A_i}{\sum_{i=1}^M a_i} = \frac{\left(\sum_{i=1}^M \frac{d}{d\mathbf{x}} A_i \right) \left(\sum_{i=1}^M a_i \right) - \left(\sum_{i=1}^M A_i \right) \left(\sum_{i=1}^M \frac{d}{d\mathbf{x}} a_i \right)}{\left(\sum_{i=1}^M a_i \right)^2}. \quad (\text{A.11})$$

The only new terms are $\frac{d}{d\mathbf{x}} A_i$ and $\frac{d}{d\mathbf{x}} a_i$ for each triangle i . One can interchange integration and differentiation in these terms and compute the area integrals using the exact same quadrature of line integrals scheme. So in addition to line integrals $l(\mathbf{x})$ and $L(\mathbf{x})$, the following integrals are needed

$$\frac{d}{d\mathbf{x}} l(\mathbf{x}) = \int_0^1 \frac{d}{d\mathbf{x}} \frac{1}{(\|\mathbf{x} - \mathbf{p}(s)\|^2 + \epsilon^2)^2} \|\mathbf{p}_1 - \mathbf{p}_0\| ds \quad (\text{A.12})$$

$$= \int_0^1 \frac{-4(\mathbf{x} - \mathbf{p}(s))}{(\|\mathbf{x} - \mathbf{p}(s)\|^2 + \epsilon^2)^3} \|\mathbf{p}_1 - \mathbf{p}_0\| ds \quad (\text{A.13})$$

$$\frac{d}{d\mathbf{x}} L(\mathbf{x}) = \int_0^1 \frac{d}{d\mathbf{x}} \frac{\phi(s) + (\mathbf{x} - \mathbf{p}_0)^\top \mathbf{n}}{(\|\mathbf{x} - \mathbf{p}(s)\|^2 + \epsilon^2)^2} \|\mathbf{p}_1 - \mathbf{p}_0\| ds \quad (\text{A.14})$$

$$= \int_0^1 \frac{\mathbf{n}(\|\mathbf{x} - \mathbf{p}(s)\|^2 + \epsilon^2) - 4(\mathbf{x} - \mathbf{p}(s))(\phi(s) + (\mathbf{x} - \mathbf{p}_0)^\top \mathbf{n})}{(\|\mathbf{x} - \mathbf{p}(s)\|^2 + \epsilon^2)^3} \|\mathbf{p}_1 - \mathbf{p}_0\| ds \quad (\text{A.15})$$

$$= \mathbf{n} l(\mathbf{x}) + (\mathbf{x} - \mathbf{p}_0)^\top \mathbf{n} \frac{d}{d\mathbf{x}} l(\mathbf{x}) + \int_0^1 \frac{-4(\mathbf{x} - \mathbf{p}(s))\phi(s)}{(\|\mathbf{x} - \mathbf{p}(s)\|^2 + \epsilon^2)^3} \|\mathbf{p}_1 - \mathbf{p}_0\| ds \quad (\text{A.16})$$

The remaining integral terms in (A.13) and (A.16) have a very similar form to the previously computed integrals. The exponent of denominators is now three instead of two, and the $\mathbf{p}(s)\phi(s)$ product in (A.16) means that the numerator is now a quadratic polynomial in s . These are the only important differences as far as integration is concerned. Thus, the integrals in (A.13) and (A.16) can be rewritten in terms of a new set of integral basis functions

$$I_{0,3} = \int_0^1 \frac{ds}{((s+k_1)^2 + k_2)^3}, \quad I_{1,3} = \int_0^1 \frac{s ds}{((s+k_1)^2 + k_2)^3}, \quad I_{2,3} = \int_0^1 \frac{s^2 ds}{((s+k_1)^2 + k_2)^3}. \quad (\text{A.17})$$

In terms of this basis, the line integrals are expressed as

$$\frac{d}{d\mathbf{x}} l(\mathbf{x}) = 4 \frac{(\mathbf{p}_0 - \mathbf{x})I_{0,3} + (\mathbf{p}_1 - \mathbf{p}_0)I_{1,3}}{\|\mathbf{p}_1 - \mathbf{p}_0\|^5} \quad (\text{A.18})$$

$$\begin{aligned} \frac{d}{d\mathbf{x}} L(\mathbf{x}) = & \mathbf{n} l(\mathbf{x}) + (\mathbf{x} - \mathbf{p}_0)^\top \mathbf{n} \frac{d}{d\mathbf{x}} l(\mathbf{x}) \\ & + 4 \frac{(\mathbf{p}_0 - \mathbf{x})(\phi_0 I_{0,3} + (\phi_1 - \phi_0)I_{1,3}) + (\mathbf{p}_1 - \mathbf{p}_0)(\phi_0 I_{1,3} + (\phi_1 - \phi_0)I_{2,3})}{\|\mathbf{p}_1 - \mathbf{p}_0\|^5} \end{aligned} \quad (\text{A.19})$$

The analytic expressions for these basis integrals are

$$\begin{aligned} I_{0,3} = & \int_0^1 \frac{1}{((s+k_1)^2 + k_2)^3} ds \\ = & \frac{1}{8} \left(\frac{-3\beta}{(k_2)^{\frac{5}{2}}} + \frac{5k_2^3 - (k_1(k_1+1) - 3)k_2^2 - k_1(k_1+1)(9k_1(k_1+1) + 5)k_2 - 3k_1^3(k_1+1)^3}{k_2^2(k_1^2 + k_2)^2((k_1+1)^2 + k_2)^2} \right) \end{aligned} \quad (\text{A.20})$$

$$\begin{aligned} I_{1,3} = & \int_0^1 \frac{s}{((s+k_1)^2 + k_2)^3} ds \\ = & \frac{1}{8} \left(\frac{3\beta k_1}{(k_2)^{\frac{5}{2}}} + \frac{(k_1^2 + k_2)((3(k_1+1) + 1)k_2^2 + (k_1+1)(6k_1^2 + 3k_1 + 2)k_2 + 3k_1^2(k_1+1)^3)}{k_2^2(k_1^2 + k_2)^2((k_1+1)^2 + k_2)^2} \right) \end{aligned} \quad (\text{A.21})$$

$$\begin{aligned} I_{2,3} = & \int_0^1 \frac{s^2}{((s+k_1)^2 + k_2)^3} ds \\ = & \frac{1}{8} \left(\frac{-\beta(3k_1^2 + k_2)}{(k_2)^{\frac{5}{2}}} - \frac{(k_1^2 + k_2)^2(k_2^2 + (k_1+1)(4k_1-1)k_2 + 3k_1(k_1+1)^3)}{k_2^2(k_1^2 + k_2)^2((k_1+1)^2 + k_2)^2} \right) \end{aligned} \quad (\text{A.22})$$

where k_1 , k_2 , and β are the same as above in Equations (A.7) and (A.10) respectively. These expressions may seem rather complex and expensive to compute. However, upon closer inspection it becomes clear that many terms are reused and shared among the expressions. These equations are implemented more efficiently by defined temporary variables for the most commonly reused terms. There is much common computation between $f(\mathbf{x})$ and $\nabla f(\mathbf{x})$, so these are computed together for maximum efficiency.

Appendix B

Software Libraries

This appendix contains documentation for key C++ software libraries written for use in this thesis. These libraries were written in a very general way such that they might be useful in other computer vision projects and possibly even projects in other fields. The code described here has been made publicly available as part of the VXL core vision libraries. The libraries contributed by this thesis work have already found use in other academic and corporate institutions. Such widespread use is a stipulation for publication of code in the VXL core.

Each section in this appendix has been published as a chapter in the online VXL book. The online edition of this documentation is available at <http://vxl.sourceforge.net/>. Since the online publication of this documentation is continually updated as the code evolves. Please refer to the online version for the latest version of this documentation.

While I am the primary designer of the framework in these libraries and the primary author of this documentation, there are several other people who have contributed code and documentation. Collaboration is a key component of developing a software library for wide spread use. I would like to thank the following people for their contributions: Joe Mundy (Brown University), Amitha Perera (Kitware Inc.), Miguel A. Figueroa-Villanueva (University of Puerto Rico), Brendan McCane (University of Otago), Ian Scott (imorphics ltd. and University of Manchester), and all the other contributors that are too numerous to mention.

B.1 Video Streaming Library (VIDL)

Summary: vidl is a library for managing video streams.

The purpose of this library is to provide a unified interface for reading and writing video data in VXL. Unlike the previous vidl library, this library treats video as a data stream. The input streams are derived from `vidl_istream` and can provide data from disk or from a live capture (a camera, for example). The output streams are derived from `vidl_ostream` and can write data to disk. Potentially, output streams could also display on a screen or transmit an internet video stream, but these capabilities are not implemented yet.

B.1.1 Introduction

An example of simple, compact video transcoding with vidl is given below.

```
#include <vidl/vidl_image_list_istream.h>
#include <vidl/vidl_ffmpeg_ostream.h>

...

// create an input stream
vidl_image_list_istream my_istream("*.png");

// create an output stream
vidl_ffmpeg_ostream my_ostream("video.avi",
    vidl_ffmpeg_ostream_params()
        .encoder( vidl_ffmpeg_ostream_params::DVVIDEO )
        .frame_rate( 29.95 ) );

// transcode the video
vidl_frame_sptr frame;
while (bool(frame = my_istream.read_frame()) &&
    my_ostream.write_frame(frame) );
```

This example has three parts: open an input stream, open an output stream, and transfer the data. The streams are opened by constructing a stream instance. The constructor arguments are different for each stream type.

In the example, the `vidl_image_list_istream` is constructed with a file glob string (See `vul_file_iterator` for details). The input video will consist all PNG files in the current directory sorted alphanumerically by file name. The images should all have the same size because the output format requires this.

The `vidl_ffmpeg_ostream` is constructed with the name of the output video file and a parameters class that specifies all the encoding parameters. The `vidl_ffmpeg_ostream_params` class uses a design

that allows the parameters to be set on an anonymous instance by chaining together function calls that each return a reference to the instance to the next function. Here the encoder `enum` is set to `DVVIDEO` and the frame rate is set to 29.95 frames per second. All other parameters take default values.

In the transcoding loop, all frames are read from the input stream one at a time. Each frame is written to the output stream until an invalid frame is detected indicating the end of the input stream. All of this processing is done within the loop condition test, and the body of the loop is empty (there are, of course, several other ways to structure this code). The `read_frame()` member function advances to the next frame in the input sequence and then returns that frame. It returns a null smart pointer when no more frames are available. The cast to `bool` causes the loop to terminate when the end of input is reached (i.e. a null pointer evaluates to false). Because of lazy evaluation, `write_frame()` is only called on the output stream when the frame is valid. This member function writes the frame to the output and returns true on a successful write. Thus, a failed write will also terminate the loop. The streams are closed when the stream instances go out of scope and are destroyed.

Comparison with Previous Video Library

This design has the following advantages over the old VXL video library (now called `vidl1`) that treated videos as container objects.

- Better support for large video files
- Better support for encoding options
- Support for live video streams
- Better integration with `vil`

Streams vs. Codecs

Various stream subclasses are used to interface with different types of video streams. Note that these are no longer called *codecs*. The codec in the old `vidl1` library was somewhat of a misnomer. A codec (COder/DECoder) provides encoding and decoding algorithms for a specific video compression format (e.g. MPEG2, DV, H.264, etc.). `vidl` (and its predecessor) provided wrappers to other APIs (like DirectShow and FFmpeg) which may encode or decode with a variety of codecs. Each `vidl_istream` and `vidl_ostream` subclass is a wrapper for another video API that may or may not provide encoding, and may support many different codecs. For example, `vidl_ffmpeg_istream` provides a input using the FFmpeg library. The FFmpeg library can open a large variety of video file formats (avi, mpeg, etc.) and decode using an even larger variety of codecs (MPEG2, MPEG4, MJPEG, H.264, Cinepak, etc.).

The video stream design also allows for streams that receive images from live video feeds. This could be from a camera, network video feed, or other source. Thus, video capture and video transcoding are both accomplished by directing an input stream into an output stream.

Several APIs that `vidl` interfaces allow for both input and output. For these there are both a corresponding `vidl_istream` and `vidl_ostream`. Other APIs are designed only for one or the other. For example, video capture APIs (like `libdc1394`) only provide an interface for input, so only a `vidl_istream` is needed. The separation of input and output was not allowed by the previous `vidl` library design.

B.1.2 Streams

All `vidl` input streams are derived from the abstract base class `vidl_istream`. Likewise all output streams are derived from `vidl_ostream`. Once a stream is created, all critical operations are controlled by invoking base class virtual functions. This polymorphic design allows video processing algorithms to operate on any combination of input and output streams configurable at run time.

The following subsections give an overview of the input and output stream base classes followed by an overview of the different derived stream types.

Input Streams

The `vidl_istream` API defines several methods to access properties of the stream, control the stream state, and access frame data. The functionality of most methods is obvious from the name. `width()` and `height()` return the video resolution in pixels, and `frame_rate()` returns the video frame rate in frames per second. Frame rate may not be define in some streams, like a sequence of image files, and 0 is returned in this case. `format()` returns the `vidl_pixel_format` of the frames. `num_frames()` returns a count of the number frames in a finite length video clip or -1 in a live video stream. Note that some video files do not store an accurate count of all video frames. In this case, `num_frames()` must scan the entire video to count the number of frames. Accessing the entire video can be costly, so use this function only when you really need an accurate frame count. `frame_number()` returns the current frame number in the sequence of video frames (starting at 0 for the first frame).

The method `is_open()` returns true when a video stream has been opened successfully and has not been closed by calling `close()`. The method `is_valid()` has a slight different meaning. A video stream is valid when its frame counter refers to a valid video frame. Video streams should open in an *invalid* state, referring to the position *before* the first valid frame. When first opened, `frame_number()` should return `static_cast<unsigned int>(-1)` to indicate the invalid frame before frame 0. The stream is also invalid when it advances past the last valid frame. The `advance()` method advances the frame counter and may invalidate the previously release frame, but does not necessarily acquire or decode the next frame. If the advance was successful it returns true. The `current_frame()` method returns a smart pointer to the current frame, acquiring or decoding it the first time this method is called. The `read_frame()` method advances the frame counter and returns the resulting current frame in one function call.

Live streams only support advancing through frames sequentially, but most other streams allow traversal forward or backward through the video to find a particular frame. This operation is called *seeking*. If an input stream supports seeking, the method `is_seekable()` will return true. If supported, the method `seek_frame(frame_number)` will seek forward or backward to set the current frame number to `frame_number`. Keep in mind that, even when supported, seeking is almost always much slower than advancing. For some video decoders it may even be error prone. Try to avoid seeking when possible.

Input Stream Types

At the time of this writing, the following input stream classes have been implemented.

`vidl_image_list_istream`

Treats an ordered list of image file paths as a video stream.

`vidl_ffmpeg_istream`

Uses the FFMPEG (<http://ffmpeg.mplayerhq.hu/>) library to decode many common video file formats.

`vidl_v4l_istream`

Use a video for Linux input stream.

`vidl_dshow_live_istream`

Use the DirectShow API to stream video directly from camera and frame-grabber devices in Windows using native Windows codecs.

`vidl_dshow_file_istream`

Use the DirectShow API to encode video files in Windows using native Windows codecs

`vidl_dc1394_istream`

Use libdc1394 v2 to stream video directly from IEEE 1394 (firewire) based cameras (Windows not supported yet)

Output Streams

The interface for output streams is much simpler than for input streams. Once an output video stream is open, its sole job is to accept frames and append them to the output video. The key method is `write_frame(frame)` which takes a frame smart pointer and tries to write it to the end of the output stream. It returns true if successful. The other methods are `is_open()` which returns true if the stream is open for writing, and `close()` which closes the stream.

The most complicated part of using output streams is typically their construction. The constructors for output streams differ depending on the particular stream type, but tend to involve

many parameters especially when video encoding and compression are involved. When the number of parameters is very large, a special parameters struct may be used to encapsulate the parameter collection. For example, `vidl_ffmpeg_ostream_params`.

Output Stream Types

At the time of this writing, the following output stream classes have been implemented.

`vidl_image_list_ostream`

Writes a video to disk as a sequence of numbered image files using any image file format supported by `vil`.

`vidl_ffmpeg_ostream`

Uses the FFMPEG (<http://ffmpeg.mplayerhq.hu/>) libraries to encode to many common video file formats with many encoding options for video quality and compression.

Image List Streams

The image list streams are the simplest streams in `vidl`. They require no third party libraries and rely only on `vil`. The `vidl_image_list_istream` treats a vector of paths to image files as a video (with unknown frame rate). The stream is constructed with a vector of paths or a file glob string that is used to match files on disk and produce a vector of paths in alphanumeric order. When the stream is “opened”, each file path is tested to see if `vil` can open it as an image (image data is not read yet). The first frame defines the video resolution. Any frame that does not open in `vil` as a `vidl` compatible format or does not match the video resolution is ignored.

The `vidl_image_list_ostream` works in the opposite way. Opening a stream requires an output directory, a name format, a file format, and an initial index. For each frame that is written to the stream, `vil` is used to write an image file to the specified directory in the specified image file format. The name of the files is produced by applying the frame number, starting at the specified initial index, to the name format using `printf` syntax. For example

```
vidl_image_list_ostream my_ostream("video", "frame%03d", "tiff", 7);
```

will write TIFF images with the following paths

```
video/frame007.tiff
video/frame008.tiff
video/frame009.tiff
video/frame010.tiff
...
```

FFMPEG Streams

FFMPEG (<http://ffmpeg.mplayerhq.hu/>) is a collection of cross-platform, open source, C libraries for encoding and decoding video. It is primarily used on Linux and Mac OS X, but it can be compiled for Windows too. It requires GCC and will not compile with Microsoft's compiler. The primary libraries are libavcodec and libavformat. Newer versions of FFMPEG also use libavutil and libswscale. For several years prior to the version 0.5 release, FFMPEG evolved continually with an ever-changing API but no formal releases. `vidl` has tried to support several "versions" of the API by supplying multiple stream implementations. The selection of implementation and its interface with FFMPEG is encapsulated and hidden from the public `vidl` stream interfaces. Going forward, `vidl` will try to support any official release versions of FFMPEG. If official release versions continue to be few and far between, a subversion snapshot of FFMPEG may also be supported and updated only when the need arises for new features. In the future, a version of FFMPEG may also be added to `v3p` to make a common version easily accessible to all VXL users.

Using the `vidl_ffmpeg_istream` is trivial. The constructor only requires the path to a video file. FFMPEG can decode almost any video file format with any video codec that is thrown at it. Be warned that some malformed files will still open but may produce errors during seeking.

Using the `vidl_ffmpeg_ostream` is also straightforward. The constructor requires the path to the output video file and a `vidl_ffmpeg_ostream_params` struct to specify the encoding parameters. The `vidl_ffmpeg_ostream_params` is a struct that mirrors the enormous set of parameters available to FFMPEG for encoding. The current set of parameters is not complete and will need to be updated in the future. Leaving most of the parameters at their default value will produce reasonable results in most cases. The most important parameters to set are encoder, frame rate, and bit rate. The encoder is the selection of codec to use from those supported by FFMPEG. An `enum` specifies a list of codecs allowed by `vidl`. This list needs to be expanded and should ultimately vary depending on the FFMPEG configuration since some codecs are optionally enabled during FFMPEG compilation. The frame rate is a `double` specifying the number of frames per second in the output video. The bit rate is an `unsigned int` specifying the target compression level in number of bits per second of video. Another set of important parameters is the frame size, the width (`ni`) and height (`nj`) of each image. However, `vidl_ffmpeg_ostream` does not require that you specify the frame size directly. Instead, it uses a delayed stream opening scheme. The constructor stores the parameters but does not immediately open the stream. When the first frame is written, the size of this frame is used to specify the size of the output video and the output stream is opened at this point. If necessary, one can force the stream to open without writing a frame by calling the `open()` method on the stream.

DirectShow Streams

Not written yet.

1394 (Firewire) Camera Streams

There are several cameras that capture and transmit video over the 1394 (aka Firewire) interface. Many of these cameras conform to a standard of camera configuration and streaming parameters. This standard is known as the IIDC or DCAM Specifications. Note that this refers to live capture 1394 cameras and does not generally include camcorders that transfer saved video using the 1394 interface.

There are several driver packages that implement the IIDC Specifications. `vidl` supports an open source package called `libdc1394` (<http://damien.douxchamps.net/ieee1394/libdc1394/>). The `vidl_dc1394_istream` uses the `libdc1394` version 2 API to capture live video from 1394 cameras. `libdc1394` currently runs on linux and Mac OS X. It may also run on Windows in the future. Other examples of IIDC camera APIs are the CMU 1394 Driver (<http://www.cs.cmu.edu/~iwan/1394/>) for Windows and the Point Grey FlyCapture SDK (<http://www.ptgrey.com/products/pgrflycapture/index.asp>) for Windows. Point Grey adds additional features beyond the standard for its own line of cameras. Separate `vidl_istream` classes could be written for these in the future.

This `vidl_dc1394_istream` is opened with a parameters struct called `vidl_iidc1394_params` that specifies all the IIDC parameters needed to configure the camera. Since the parameters are standard and not specific to `libdc1394`, the `vidl_iidc1394_params` struct is intended to be shared with any other IIDC 1394 stream classes written in the future. Additional parameters that are specific to `libdc1394` are kept out of `vidl_iidc1394_params`. Namely, the `num_dma_buffers` parameter is passed in separately when opening the stream. This parameter specifies the number of frame buffers to use for Direct Memory Access (DMA) transfer. `libdc1394` will create a separate thread to continually read frames into these buffers asynchronously from the main program. Refer to the `libdc1394` documentation for details.

The `vidl_iidc1394_params` struct also contains a nested struct called `valid_options`. This struct is used to represent the available devices and their parameters found by probing the 1394 bus. The `vidl_dc1394_istream` class has a static member function called `valid_params` that can be used to probe the bus and populate this struct.

Video 4 Linux Streams

Not written yet.

B.1.3 Video Frames

The `vidl_frame_sptr` is a data structure used to transport video frame data from a `vidl_istream` to a `vidl_ostream`. The `vidl_frame` contains a pointer to the frame buffer, the image resolution, the pixel format, and may eventually contain a time stamp. A `vidl_frame` is more restrictive than a `vil_image_view` since it requires contiguous memory and is limited in pixel data types. However, it also handles many pixels formats not supported by `vil`, such as those with sub-sampled chrominance (i.e. YUV 4:2:2).

In some cases the `vidl_frame` data can be wrapped by a `vil_image_view`, but usually this is not a good idea. The `vidl_frame` should be considered a volatile data structure. The image buffer is only guaranteed to be valid until the next call of `vidl_istream::advance()`. The `istream` will keep a pointer to the released frame and then invalidate it before reusing memory. Use the functions in `vidl_convert.h` to obtain a non-volatile `vil_image_view`. More details are given below in the section on conversions.

B.1.4 Pixel Formats

The supported pixel formats are enumerated in `vidl_pixel_format` enum. The naming convention is `VIDL_PIXEL_FORMAT_color_num[P]`, where *color* is the color encoding, *num* is the bit number, and the P character is optionally appended to indicate a planar encoding. There are also two special formats. `VIDL_PIXEL_FORMAT_UNKNOWN` is used when the pixel format is not known or does not matter. `VIDL_PIXEL_FORMAT_ENUM_END` is an end of enum marker and should never be used in a `vidl_frame`. This marker is used to count the number of pixel formats. All new formats should be added before the end marker in the enum.

The *color* of a pixel format specifies color encoding. A separate enum named `vidl_pixel_color` enumerates the available color modes. These are

`VIDL_PIXEL_COLOR_MONO`

Monochrome or grayscale (1 color channel)

`VIDL_PIXEL_COLOR_RGB`

Red Green Blue (3 color channels)

`VIDL_PIXEL_COLOR_RGBA`

Red Green Blue Alpha (4 color channels)

`VIDL_PIXEL_COLOR_YUV`

Luma Chroma-Blue Chroma-Red, aka YCrCb (3 color channels)

The multichannel color formats are not always encoded in the order given above. The characters are permuted in the `vidl_pixel_format` names to indicate the actual ordering. For example, `BGR` indicates RGB color stored in reverse order.

The *bit number* indicates how the bits are laid out. This is most important in packed color frames where a group of 2 or 4 pixels is encoded together and differing numbers of bits are added to different color channels. When no packing occurs, the number in the `vidl_pixel_format` name is the total number of bits (i.e. `RGB_24` is RGB with 8 bits for each channel). When the number of bits per channel is different, the standard chroma sub-sampling numbering is used (i.e. 422 for 4:2:2). See Wikipedia (http://en.wikipedia.org/wiki/Chroma_subsampling) for more details. A separate enum named `vidl_pixel_arrangement` enumerates the available arrangements. These are

`VIDL_PIXEL_ARRANGE_SINGLE`

Each pixel is listed in single file with equal bit allocation to each channel (e.g. RGBRG-BRGBRGRGB...)

VIDL_PIXEL_ARRANGE_PACKED

Small blocks of data represent a group of pixels and bit allocation generally differs between channels

VIDL_PIXEL_ARRANGE_PLANAR

All data from one channel is provided before providing all data from the next channel (e.g. RRRR... GGGG... BBBB...). Chroma sub-sampling may still apply.

VIDL_PIXEL_ARRANGE_PALETTE

(Not currently used) Each pixel is an index into a look-up table of colors.

The Pixel Traits Table

To determine properties of each pixel type, `vidl` relies heavily on template specialization and meta-programming. The template `vidl_pixel_traits_of<FMT>` is specialized for each pixel format `FMT`. The traits class provides several static members that describe the properties of each format. The trait specializations are created using the macro `vidl_pt_mac`. This macro lets all the properties appear in a nice table. It also makes it easy to add a new pixel format.

The following is a subset of the pixel traits table found in `vidl_pixel_format.h`.

//	format	name	type	bpp	color	arrange	xcs	ycs
//	-----	-----	----	---	-----	-----	---	---
<code>vidl_pt_mac(</code>	<code>UNKNOWN,</code>	<code>"unknown",</code>	<code>void,</code>	<code>0,</code>	<code>UNKNOWN,</code>	<code>UNKNOWN,</code>	<code>0,</code>	<code>0);</code>
<code>vidl_pt_mac(</code>	<code>RGB_24,</code>	<code>"RGB 24",</code>	<code>vx1_byte,</code>	<code>24,</code>	<code>RGB,</code>	<code>SINGLE,</code>	<code>0,</code>	<code>0);</code>
<code>vidl_pt_mac(</code>	<code>RGB_24P,</code>	<code>"RGB 24P",</code>	<code>vx1_byte,</code>	<code>24,</code>	<code>RGB,</code>	<code>PLANAR,</code>	<code>0,</code>	<code>0);</code>
<code>vidl_pt_mac(</code>	<code>BGR_24,</code>	<code>"BGR 24",</code>	<code>vx1_byte,</code>	<code>24,</code>	<code>RGB,</code>	<code>SINGLE,</code>	<code>0,</code>	<code>0);</code>
<code>vidl_pt_mac(</code>	<code>RGBA_32,</code>	<code>"RGBA 32",</code>	<code>vx1_byte,</code>	<code>32,</code>	<code>RGBA,</code>	<code>SINGLE,</code>	<code>0,</code>	<code>0);</code>
<code>vidl_pt_mac(</code>	<code>YUV_444P,</code>	<code>"YUV 444P",</code>	<code>vx1_byte,</code>	<code>24,</code>	<code>YUV,</code>	<code>PLANAR,</code>	<code>0,</code>	<code>0);</code>
<code>vidl_pt_mac(</code>	<code>YUV_422P,</code>	<code>"YUV 422P",</code>	<code>vx1_byte,</code>	<code>16,</code>	<code>YUV,</code>	<code>PLANAR,</code>	<code>1,</code>	<code>0);</code>
<code>vidl_pt_mac(</code>	<code>YUV_420P,</code>	<code>"YUV 420P",</code>	<code>vx1_byte,</code>	<code>12,</code>	<code>YUV,</code>	<code>PLANAR,</code>	<code>1,</code>	<code>1);</code>
<code>vidl_pt_mac(</code>	<code>YUV_410P,</code>	<code>"YUV 410P",</code>	<code>vx1_byte,</code>	<code>10,</code>	<code>YUV,</code>	<code>PLANAR,</code>	<code>2,</code>	<code>1);</code>
<code>vidl_pt_mac(</code>	<code>YUYV_422,</code>	<code>"YUYV 422",</code>	<code>vx1_byte,</code>	<code>16,</code>	<code>YUV,</code>	<code>PACKED,</code>	<code>1,</code>	<code>0);</code>
<code>vidl_pt_mac(</code>	<code>MONO_1,</code>	<code>"Mono 1",</code>	<code>bool,</code>	<code>1,</code>	<code>MONO,</code>	<code>SINGLE,</code>	<code>0,</code>	<code>0);</code>
<code>vidl_pt_mac(</code>	<code>MONO_8,</code>	<code>"Mono 8",</code>	<code>vx1_byte,</code>	<code>8,</code>	<code>MONO,</code>	<code>SINGLE,</code>	<code>0,</code>	<code>0);</code>
<code>vidl_pt_mac(</code>	<code>MONO_16,</code>	<code>"Mono 16",</code>	<code>vx1_uint_16,</code>	<code>16,</code>	<code>MONO,</code>	<code>SINGLE,</code>	<code>0,</code>	<code>0);</code>

The meaning of each column in the pixel traits table are as follows.

`format`

The format identifier symbol. Appended to `VIDL_PIXEL_FORMAT_`, this should match the `vidl_pixel_format` enum value.

name

The string name of the format for human readability.

type

The data type to be used when converting to a `vil_image_view`.

bpp

The number of bits per pixel. Not always an even multiple of `sizeof(type)` because pixels may be packed or sub-sampled in some planes (see `xcs` and `y cs`).

color

The color format. Appended to `VIDL_PIXEL_COLOR_` this determines the `vidl_pixel_color`.

arrange

The pixel arrangement.

Appended to `VIDL_PIXEL_ARRANGE_` this determines the `vidl_pixel_arrangement`.

xcs

The X color sub-sampling factor. In YUV color space, the U and V channels have a horizontal resolution (`ni`) that is 2^{-xcs} times that of the Y channel.

y cs

The Y color sub-sampling factor. In YUV color space, the U and V channels have a vertical resolution (`nj`) that is $2^{-y cs}$ times that of the Y channel.

A new pixel format may be introduced by simply adding an entry in this table and a corresponding `enum` addition to `vidl_pixel_format`. The traits of a pixel format are used in the automatic generation of conversion routines. The ultimate goal of the `vidl` pixel format framework is that new formats added to the table should automatically work (possibly slowly) without writing any additional conversion routines.

Run-time Pixel Traits

The template specializations used to define traits are very useful at compile-time, but less useful given a pixel format that is not known until run-time. A second (non-templated) struct named `vidl_pixel_traits` is available to dynamically hold traits at run-time. The function `vidl_pixel_format_traits` provides dynamic look-up of traits from a `vidl_pixel_format` variable.

```

// find the pixel format of a frame
vidl_pixel_format format = my_frame_sptr->pixel_format();

// look up the pixel traits
vidl_pixel_traits pt = vidl_pixel_format_traits(format);

// print out some information
vcl_cout << "The frame has format " << pt.name
          << " with " << pt.bits_per_pixels << " bits per pixel"
          << " and " << pt.num_channels << " channels" << vcl_endl;

```

To simplify this process there are several convenience functions, such as `vidl_pixel_format_bpp`, to directly look up a pixel trait given the format without the need for the traits struct. There is also a `vcl_ostream` operator for `vidl_pixel_format` so that the human readable string name is printed. The following example

```

vidl_pixel_format format = VIDL_PIXEL_FORMAT_RGB_24;
vcl_cout << "pixel format is " << format << vcl_endl;

```

produces the output

```

pixel format is RGB 24

```

There is also a function that provides a look up of a pixel format from its string name. Use `vidl_pixel_format_from_string` for this purpose. The expression

```

vidl_pixel_format_from_string("RGB 24") == VIDL_PIXEL_FORMAT_RGB_24

```

should evaluate to true.

B.1.5 Pixel Format Conversion

When passing a frame from an input stream to an output stream, pixel format conversion is generally not something the user needs to worry about. The `vidl_istream` should provide a `vidl_frame_sptr` that uses whatever pixel format is native to the video data. If the `vidl_ostream` requires a different pixel format, then it performs the required conversion automatically.

If for some reason manual conversion is required, the appropriate functions are found in header `vidl_convert.h`. The function is called `vidl_convert_frame`, and it comes in two forms. The first

```

bool vidl_convert_frame(const vidl_frame& in_frame,
                       vidl_frame& out_frame);

```

takes an input frame and an output frame. The output frame must have the appropriate size and pixel format for the output. If the size is not appropriate then the function returns false. This version is useful when you want to store the converted data into an existing memory buffer; for example, reusing the same output frame for each new input frame. The second form of the function

```
vidl_frame_sptr vidl_convert_frame(const vidl_frame_sptr& in_frame,
                                   vidl_pixel_format format);
```

takes a smart pointer to an input frame and allocates new memory for the output frame using the specified pixel format. If the conversion is possible, a smart pointer to the new frame is returned. Otherwise, a null smart pointer is returned. This version requires less preparation by the user, but does not allow for the reuse of frame memory.

Conversion Between Frames and Image Views

One case where the user usually does want to convert a frame is in the conversion to and from a `vil_image_view`. This type of conversion is needed to do any image processing on the data. There is more than one way to do such a conversion. The fastest way, if possible, is to share the memory and wrap the `vidl_frame` in a `vil_image_view` or vice versa. Wrapping is only possible when the memory layout of the `vil` memory format is compatible with a `vidl` pixel format.

For writing to output streams, wrapping is usually safe because the data is immediately used and then no longer needed. The function `vidl_convert_to_frame` takes a `vil_image_view_base_sptr` and always tries first to wrap the memory. If wrapping fails, new memory is allocated and the image data is copied (and converted if necessary) into that memory. The result is returned as a `vidl_frame_sptr`.

When reading frames from video, wrapping can be more dangerous. Processing images from video often involves multiple frames. To reduce repeated reallocation of memory, most input streams will reuse the same memory. A member function on the `vidl_frame` allows an input stream to *invalidate* the frame object it previously released in order to reclaim the memory for the next frame. The invalidated frame will now have a null pointer to image data, but any `vil_image_view` that was sharing the memory will not be notified. The image sharing the memory will be overwritten when the next frame is read from the video stream. It is safer to copy memory when converting a frame to an image, but it is potentially much slower than wrapping. Functions are provided for both copy conversion and wrapping conversion. The appropriate conversion depends on the situation.

When in doubt, the safest way to get a `vil_image_view` from a `vidl_frame` is by copying with

```
bool vidl_convert_to_view(const vidl_frame& frame,
                          vil_image_view_base& image,
                          vidl_pixel_color require_color);
```

This function resizes the image (if necessary) and then copies the frame data into it with the appropriate conversion (if necessary). The conversion is based on the data type of the image view. The last function argument, `require_color`, is optional and defaults to `VIDL_PIXEL_COLOR_UNKNOWN`. In the default case, the color encoding is kept the same as the input frame. If `require_color` is specified, then it will be used as the resulting color encoding. Videos are often stored and transmitted in YUV, but `vil` images are usually assumed to be in RGB. The `require_color` is used to force that conversion.

`vidl_convert_to_view` will return true if the conversion is successful.

The fastest way to obtain a `vil_image_view` is by wrapping the frame memory. The function `vidl_convert_wrap_in_view` takes a `vidl_frame` and tries to wrap its memory with an appropriate image view. If the wrapping is possible, the function returns a `vil_image_view_base_sptr` to the view. Otherwise, it returns a null smart pointer.

Adding Optimized Conversion Routines

The pixel traits table is used to automatically generate conversion routines between any pair of listed pixel formats. There are two types of automatically generated conversion functions. The first, and simplest, is applied when the input pixel format matches the output pixel format. In this case, no conversion is needed so the frame data is copied directly to the output with `vcl_memcpy`. The second, and most general, is applied when the pixel formats do not match and an actual conversion is needed. In this case, the function uses a `vidl_pixel_iterator` for the input and output frames to step through each pixel and convert it. A `vidl_pixel_iterator` uses virtual functions to provide a standard interface for accessing or setting the pixel data as an array of bytes, even when multiple pixels are packed together. The data at each pixel is also passed through a color conversion function to convert the pixel data type and color representation when necessary.

It is obvious that this generic conversion process is not optimal. For each pixel, multiple virtual functions are called, and the pixel data is copied through multiple intermediate variables. Pixel iterators introduce additional overhead in packed formats where data shared across pixels is accessed multiple times. Since fast pixel format conversions are required in many applications, the generic routine is not acceptable. Unfortunately, it is intractable to write an optimized conversion routine for all pairs of pixel formats. That would require hundreds of specialized functions. Yet, not every conversion combination needs optimal performance. Only a small subset of conversions are commonly used in practice. `vidl` allows the default, generic conversion routine to be overridden by an optimized implementation for the most common format pairs.

Optimized conversion routines are added by creating a template specialization within the designated area in `vidl_convert.cxx`. The specialization should follow the example below.

```
VCL_DEFINE_SPECIALIZATION
struct convert<IN_FMT, OUT_FMT>
{
    enum { defined = true };
    static bool apply(const vidl_frame& in_frame,
                     vidl_frame& out_frame)
    {
        assert(in_frame.pixel_format()==IN_FMT);
        assert(out_frame.pixel_format()==OUT_FMT);

        // convert the data here
    }
}
```

```
        return true;
    }
};
```

In this example, `IN_FMT` should be replaced by the input pixel format `enum` value (for example, `VIDL_PIXEL_FORMAT_RGB_24`). Likewise, `OUT_FMT` should be replaced with the output pixel format `enum` value. Several optimized conversion routines are already provided. The existing functions can be used as an example when adding new optimizations.

B.2 Probability Distribution Library (VPDL)

Summary: `vpdl` is a library of probability distributions.

This library contains data structures to represent univariate and multivariate probability distributions and provides algorithms to operate on them. This includes estimating a distribution from data points and sampling from a distribution. `vpdl` is built on top of `vn1` and `vn1_algo`.

`vpdl` is comprised of two programming paradigms: generic programming and polymorphic inheritance. The generic programming part is in its own sub-library called `vpdt`. `vpdt` is a template library (like STL). There is no compiled library for `vpdt`, only a collection of header files in subdirectory `vpdl/vpdt`. `vpdt` works with `vn1` types, but in many cases can generalize to other types. The goal of `vpdt` is to provide generic implementations that are both time and memory efficient when types are known at compile time.

The rest of `vpdl` uses a polymorphic design and provides greater run time flexibility and easy of use. It is limited to distributions over scalar, `vn1_vector`, and `vn1_vector_fixed` types.

B.2.1 History

`vpdl` is the combination of two different design patterns for probability distributions. It was formed from the merger of three contrib libraries: `mul/vpdf1`, `mul/pdf1d`, and `br1/bbas/bsta`.

Created by Manchester, `vpdf1` provided a polymorphic hierarchy (using virtual functions) for multivariate distributions based on `vn1_vector` and `vn1_matrix` types. For univariate distributions, `pdf1d` mirrored the design of `vpdf1`, but used scalar types (i.e. `double`). These libraries were very flexible at run time. Both distribution type and, in the case of `vpdf1`, dimension could be selected at run time.

Created by Brown, `bsta` provided a generic programming hierarchy (using templates) for both univariate and multivariate distributions. Template parameters specified scalar type (`float` or `double`) and dimension. Templates allowed the same code base to use scalars in the univariate case and `vn1_vector_fixed` and `vn1_matrix_fixed` in the multivariate case. The goal of `bsta` was to be very efficient. Many optimizations are possible by assuming types and dimension are known at compile time.

`vpdl` was designed as a core library to meet the needs of both original designs. The main library generalizes Manchester's polymorphic design with a templated distribution library. This creates polymorphic inheritance hierarchy for each choice to template parameters. The templates allow hierarchies based on scalar, `vn1_vector`, and `vn1_vector_fixed` types. When possible, the implementations of `vpdl` distributions are wrappers around corresponding `vpdt` distributions. The `vpdt` distributions provide even more generalized implementations but without virtual functions or inheritance.

B.2.2 Polymorphic Distributions

Each polymorphic distribution is derived (directly or indirectly) from a common templated base class called `vpdl_distribution<T,n>`. Template parameter `T` specifies the scalar numeric type (`float` or `double`) and `n` specifies the dimension. In particular:

- `n==0` uses `vn1_vector<T>` and `vn1_matrix<T>` (dimension specified at run time)
- `n==1` uses `T` (scalar computations)
- `n>1` uses `vn1_vector_fixed<T,n>` and `vn1_matrix_fixed<T,n,n>` (fixed dimension of `n`)

The distribution class is an abstract base class. The most important functions it defines are:

```
virtual T density (const vector &pt) const=0;
```

Evaluate the unnormalized density at a point.

```
virtual T norm_const () const=0;
```

The normalization constant for the density.

```
virtual T prob_density (const vector &pt) const;
```

Evaluate the probability density at a point. Defaults to `density(pt) * norm_const()`.

These functions allow the density of the distribution to be evaluated independently from the normalization constant. This is important because the normalization constant is sometimes expensive to compute and not required for some calculations. Furthermore, the normalization constant is independent of the evaluation point so it can be computed once in advance and reused. Alternatively, the function `prob_density` returns the normalized density directly.

The type `vector` is a typedef defined in the class that selects the appropriate vector class depending on `T` and `n`. Similarly, the type `matrix` is defined to the appropriate matrix type.

Several other functions are defined on the distribution base class including: `gradient_density`, `cumulative_prob`, `compute_mean`, and `compute_covar`. All of these functions provide a polymorphic interface to probability distributions. However, in most cases, the `vpdl` classes are simply light wrappers around generic `vpdt` implementations that do the real work.

One additional component of the polymorphic wrappers is a class hierarchy. In `vpdt`, the distribution classes need not have any inheritance relations, but in `vpdl` the currently implemented distributions are arranged into the following class hierarchy. Each top level class below is derived directly from `vpdl_distribution` (template arguments are omitted).

- `vpdl_gaussian_base` - a base class for all Gaussian varieties.
 - `vpdl_gaussian` - a general Gaussian with full covariance.
 - `vpdl_gaussian_indep` - a Gaussian with independent (i.e. diagonal) covariance.
 - `vpdl_gaussian_sphere` - a Gaussian with spherical (i.e. scaled identity) covariance.

- `vpdl_multi_cmp_dist` - a base class for all multi-component distributions
 - `vpdl_kernel_base` - a base class for kernel density distributions.
 - * `vpdl_kernel_fbw_base` - a base class for fixed bandwidth kernel densities
 - `vpdl_kernel_gaussian_sfbw` - a fixed bandwidth spherical Gaussian kernel distribution
 - * `vpdl_kernel_vbw_base` - a base class for variable bandwidth kernel densities
 - `vpdl_mixture` - a mixture distribution (linear combination of arbitrary distributions).
 - `vpdl_mixture_of` - a mixture distribution where each component has the same type (but may have different parameters).

B.2.3 Generic Probability (`vpdt`)

The subdirectory `vpdl/vpdt` contains the generic programming components of this library. `vpdt` is a self-sufficient template library that does not depend on the rest of `vpdl`. However, there is nothing to stop `vpdl` distribution classes from being used as `vpdt` distribution types as long as they meet the requirements. Whenever possible, `vpdl` distributions should be wrappers around a generic `vpdt` implementation. This strategy allows the code to be used both in generic and polymorphic designs.

Generic programming uses *concepts* to implement compile time type polymorphism rather than inheritance to implement run time object-oriented polymorphism. Concepts are sets of requirements that a data type must meet. To satisfy a concept a class must define the required typedefs, static constants, member functions, or related global functions. Classes that satisfy a concept need not be related by inheritance. The Standard Template Library (STL) is built upon the definition of concepts. In STL, some concepts (like Iterators) are general enough to include built-in types (like pointers). In `vpdt`, there are also some basic concepts which can be satisfied by built-in types. However, the most useful concepts (like Distributions) are a bit more specific and require a class implementation.

Basic Concepts

A collection of more basic concepts will form the building blocks of the probability distribution concept. The relevant concepts are:

- Scalar
- Field
- Vector
- Matrix

`vn1` data structures satisfy these concepts and are used in practice. However, other representations could be used if the appropriate functions are provided. These concepts are all interrelated

by various function requirements. The type that meets the Field concept is considered the primary type. For each Field type there should be a unique associated Scalar, Vector, and Matrix type. The `vpdt_field_traits<F>` struct defines these types for each F. `vpdt_field_traits<F>` needs to be specialized for each Field type. This traits struct defines the following typedefs:

`vpdt_field_traits<F>::dimension`

This is actually a `static const unsigned int` and indicates the fixed dimension of the Field type

`vpdt_field_traits<F>::scalar_type`

The Scalar type associated with the Field type

`vpdt_field_traits<F>::field_type`

The Field type itself (`F==field_type`). Redundant, but provided for consistency.

`vpdt_field_traits<F>::vector_type`

The Vector type associated with the Field type

`vpdt_field_traits<F>::matrix_type`

The Matrix type associated with the Field type

In the special case of univariate distributions, all four of these types become equal, and `dimension==1`. There is also a special typedef `vpdt_field_traits<F>::type_is_scalar` defined to `void` that only exists if F is scalar. Similarly, `vpdt_field_traits<F>::type_is_vector` is defined to `void` only if F is non-scalar. The existence of these special types is used to disambiguate some template specializations, and to provide faster implementations for the scalar case.

Scalar

A probability distribution is defined over some scalar field. The Scalar concept represents this value at each point in space. It is satisfied by floating point types. For now this is likely to be limited to `double` or `float`. Scalars must support all built in arithmetic operators (+, -, *, /, etc.) as well as all standard cmath functions like `sqrt`, `exp`, and `log`. If the type does not support these functions directly, it must be able to implicitly cast itself to and from a type that does.

Field

Each point in a probability space is represented by the Field concept. When used in `vpdl` distributions the Field is satisfied by either a scalar, a `vn1_vector`, or a `vn1_vector_fixed`. The Field has an associated Scalar and dimension. For dimension `n`, a Field contains `n` values of its Scalar type. A Field actually has two measure of dimension: a fixed (compile time) dimension and a variable (run time) dimension. These dimensions may or may not be equal. Field types with data allocated on the stack will have size specified at compile time. The fixed dimension and variable dimension will

both equal this fixed size. Field types with data allocated on the heap will have a fixed dimension of zero and a variable dimension equal to the number of currently allocated Scalar objects.

A Field type will be equal to its Scalar type in the case of univariate distributions. Since Scalar types are usually built-in C++ types, they can not be required to implement any member functions. Instead, they are required to implement a set of overloaded global functions to perform the required set of actions. For the standard types used in `vpdl`, these functions are implemented in `vpdl/vpdt/vpdt_access.h`. For some types that satisfy the concept these functions are trivial or even empty. In the table below the Field type is denoted `field` and its corresponding Scalar type is `scalar`.

```
unsigned int vpdt_size(const field&)
```

Access the run time size (i.e. dimension)

```
void vpdt_set_size(field&, unsigned int)
```

Change the run time size for heap-based structures, otherwise do nothing.

```
void vpdt_fill(field&, const scalar&)
```

Fill all dimensions of the field with the scalar value

```
const scalar& vpdt_index(const field&, unsigned int)
```

Access a scalar element reference (const) by index

```
scalar& vpdt_index(field&, unsigned int)
```

Access a scalar element reference (non-const) by index

The Field type should also support the subtraction operator on two Field types. The return type for the difference of two Field types is a Vector type.

Vector

The Vector concept represents the difference between two Field values. In the case of the default `vn1` types, the Field and Vector types are the same. The requirements for the Vector are a superset of the requirements for a Field. A Vector should support all the operation of Field plus several other arithmetic operations:

- `vector = vector + vector` (and equivalent for `-`)
- `vector += vector` (and equivalent for `-=`)
- `field = field + vector` (and equivalent for `-`)
- `field += vector` (and equivalent for `-=`)
- `vector = scalar + vector` (and equivalent for `-`)
- `vector += scalar` (and equivalent for `-=`)

- `vector = scalar * vector`
- `vector *= scalar`

Vectors should also define these global functions

- `scalar dot_product(const vector&, const vector&)`
- `vector element_product(const vector&, const vector&)`
- `matrix outer_product(const vector&, const vector&)`

Matrix

The Matrix concept actually refers to a square matrix. It is used for second order statistics (like covariance). Since it is square, the matrix also has a single (fixed or variable) dimension like the Field and Vector. It requires all the same `vpdt_access.h` functions except the signature of the `vpdt_index` function now has two indices

- `const scalar& vpdt_index(const matrix&, unsigned int, unsigned int)`
- `scalar& vpdt_index(matrix&, unsigned int, unsigned int)`

A few arithmetic operations are also needed:

- `matrix = matrix + matrix` (and equivalent for `-`)
- `matrix += matrix` (and equivalent for `--`)
- `matrix = scalar + matrix` (and equivalent for `-`)
- `matrix += scalar` (and equivalent for `--`)
- `matrix = scalar * matrix`
- `matrix *= scalar`

Distribution Concept

The Distribution concept is the central concept in `vpdt`. The Distribution is a probability distribution class that is analogous to `vpdl_distribution<T,n>`. In fact, `vpdl_distribution<T,n>` and all its subclasses satisfy the Distribution concept. The Distribution requires a subset of the functions that are declared virtual on `vpdl_distribution<T,n>`. The following example lays out the required API that a Distribution is required to have. In this example, it is assumed that a consistent set of basic types have been defined such that `T` satisfies Scalar, `F` satisfies Field, `V` satisfies Vector, and `M` satisfies Matrix.

```

class vpdf_distribution
{
public:
    //: The field type
    typedef F field_type;

    //: Return the variable (run time) dimension
    unsigned int dimension() const;

    //: Evaluate the unnormalized density at a point
    // This must be multiplied by norm_const() to integrate to 1
    T density(const F& pt) const;

    //: Compute the gradient of the density function, returned in \a g
    // The return value of the function is the density itself
    T gradient_density(const F& pt, V& g) const;

    //: Compute the normalization constant (independent of sample point).
    T norm_const() const;

    //: Evaluate the cumulative distribution function at a point
    // This is the integral of the density function from negative infinity
    // (in all dimensions) to the point in question
    T cumulative_prob(const F& pt) const;

    //: Compute the mean of the distribution.
    void compute_mean(F& m) const;

    //: Compute the covariance matrix of the distribution.
    void compute_covar(M& c) const;
};

```

The other member functions that are defined in `vpdf_distribution<T,n>` are not needed for a `vpdf` Distribution because they can be implemented in terms of the required functions. These extra functions are provided as global functions instead. For example,

```
T vpdf_prob_density(const dist& d, const F& pt)
```

is a function that returns `d.density(pt) * d.norm_const()`. These functions can be overloaded when a more efficient implementation exists for a particular distribution class. For example, the default `vpdf_log_density` function applied to a Gaussian distribution would compute the log of an exponential. An overload of `vpdf_log_density` for Gaussians can eliminate this extra unnecessary step.

Bibliography

- [1] Amol A. Ambardekar. Efficient vehicle tracking and classification for an automated traffic surveillance system. Master's thesis, University of Nevada, Reno, December 2007.
- [2] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, and J. Davis. Scape: Shape completion and animation of people. *ACM Transactions on Graphics*, 24(3):408–416, 2005.
- [3] Arthur Appel. The notion of quantitative invisibility and the machine rendering of solids. In *Proceedings of the ACM National Conference*, pages 387–393, New York, NY, USA, 1967. ACM.
- [4] S. Atev, H. Arumugam, O. Masoud, R. Janardan, and N.P. Papanikolopoulos. A vision-based approach to collision prediction at traffic intersections. *IEEE Transactions on Intelligent Transportation Systems*, 6(4):416–423, December 2005.
- [5] Albert E. Beaton and John W. Tukey. The fitting of power series, meaning polynomials, illustrated on band-spectroscopic data. *Technometrics*, 16(2):147–185, May 1974.
- [6] D. Beymer, P. McLauchlan, B. Coifman, and J. Malik. A real-time computer vision system for measuring traffic parameters. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 495–501, Jun 1997.
- [7] Stan Birchfield. Klt: An implementation of the kanade-lucas-tomasi feature tracker. <http://www.ces.clemson.edu/~stb/klt/>.
- [8] Andrew Blake and Andrew Zisserman. *Visual Reconstruction*. MIT Press, 1987.
- [9] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 187–194, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.
- [10] James F. Blinn. A generalization of algebraic surface drawing. *ACM Trans. Graph.*, 1(3):235–256, 1982.
- [11] Jean-Yves Bouguet. Camera calibration toolbox for matlab. http://www.vision.caltech.edu/bouguetj/calib_doc/, June 2008.

- [12] Philippe Loic Marie Bouttefroy, Abdesselam Bouzerdoum, Son Lam Phung, and Azeddine Beghdadi. Vehicle tracking by non-drifting mean-shift using projective kalman filter. In *IEEE Conference on Intelligent Transportation Systems*, pages 61–66, Oct. 2008.
- [13] Y. Boykov and G. Funka-Lea. Graph cuts and efficient n-d image segmentation. *International Journal of Computer Vision*, 70(2):109–131, 2006.
- [14] Neil Bruce and Pierre Kornprobst. Harris corners in the real world: A principled selection criterion for interest points based on ecological statistics. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2160–2167, 2009.
- [15] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.
- [16] D. Carruthers, C. Uloth, and G. G. Roy. An evaluation of formulae for solar declination and the equation of time. Research Report RR17, School of Architecture, University of Western Australia, January 1990.
- [17] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological surfaces. *Computer-Aided Design*, 10(6):350–355, 1978.
- [18] Jae-Young Choi, Kyung-Sang Sung, and Young-Kyu Yang. Multiple vehicles detection and tracking based on scale-invariant feature transform. In *IEEE Intelligent Transportation Systems Conference*, pages 528–533, October 2007.
- [19] Citilog, Inc. <http://www.citilog.fr/>.
- [20] Collision Investigation and Research Division of Transport Canada. Canadian vehicle specifications. <http://www.carsp.ca/index.php?0=documents&1=47>, March 2009.
- [21] D.J. Dailey, F.W. Cathey, and S. Pumrin. An algorithm to estimate mean traffic speed using uncalibrated cameras. *Intelligent Transportation Systems, IEEE Transactions on*, 1(2):98–107, Jun 2000.
- [22] DARPA. Exploitation of 3-d data (e3d). <http://www.darpa.mil/IPT0/programs/e3d/e3d.asp>.
- [23] Dash Navigation, Inc. <http://www.dash.net/>.
- [24] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive contours for conveying shape. *ACM Trans. Graph.*, 22(3):848–855, 2003.
- [25] T. Drummond and R. Cipolla. Real-time visual tracking of complex structures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):932–946, Jul 2002.

- [26] M.-P. Dubuisson Jolly, S. Lakshmanan, and A.K. Jain. Vehicle segmentation and classification using deformable templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(3):293–308, Mar 1996.
- [27] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. Wiley New York, 2001.
- [28] M. Fathy and M. Y. Siyal. An image detection technique based on morphological edge detection and background differencing for real-time traffic analysis. *Pattern Recognition Letters*, 16(12):1321 – 1330, 1995.
- [29] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, September 2004.
- [30] J. M. Ferryman, A. D. Worrall, G. D. Sullivan, and K. D. Baker. A generic deformable model for vehicle recognition. In *BMVC '95: Proceedings of the 1995 British conference on Machine vision (Vol. 1)*, pages 127–136, Surrey, UK, UK, 1995. BMVA Press.
- [31] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.
- [32] Florent Fusier, Valéry Valentin, François Brémont, Monique Thonnat, Mark Borg, David Thirde, and James Ferryman. Video understanding for complex activity recognition. *Machine Vision and Applications*, 18(3-4):167–188, August 2007.
- [33] N. Ghosh and B. Bhanu. Incremental vehicle 3-d modeling from video. In *International Conference on Pattern Recognition (ICPR)*, volume 3, pages 272–275, 2006.
- [34] Jason Gorss. Portable, solar-powered tag readers could improve traffic management. <http://news.rpi.edu/update.do?artcenterkey=1828>, Nov 2006.
- [35] Y. Guo, S. Hsu, Y. Shan, H. Sawhney, and Rakesh Kumar. Vehicle fingerprinting for reacquisition & tracking in videos. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 761–768, June 2005.
- [36] Yanlin Guo, Cen Rao, S. Samarasekera, J. Kim, R. Kumar, and H. Sawhney. Matching vehicles under large pose transformations using approximate 3d models and piecewise mrf model. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, June 2008.
- [37] S. Gupte, O. Masoud, R.F.K. Martin, and N.P. Papanikolopoulos. Detection and classification of vehicles. *Intelligent Transportation Systems, IEEE Transactions on*, 3(1):37–47, Mar 2002.
- [38] D. M. Ha, J. M. Lee, and Y. D. Kim. Neural-edge-based vehicle detection and traffic parameter extraction. *Image and Vision Computing*, 22(11):899–907, 2004.

- [39] Michael Haag and Hans-Hellmut Nagel. Combination of edge element and optical flow estimates for 3d-model-based vehicle tracking in traffic image sequences. *International Journal of Computer Vision*, 35(3):295–319, December 1999.
- [40] D.J. Han, M.J. Leotta, D.B. Cooper, and J.L. Mundy. Vehicle class recognition from video-based on 3d curve probes. In *IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance (PETS)*, pages 285–292, October 2005.
- [41] John Hannavy. *Encyclopedia of Nineteenth-century Photography*, chapter Aerial Photography, page 14. CRC Press, 2008.
- [42] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, pages 147–151, 1988.
- [43] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, first edition, 2000.
- [44] P. W. Holland and R. E. Welsch. Robust regression using iteratively reweighted least-squares. *Communications in Statistics - Theory and Methods*, 6(9):813–827, 1977.
- [45] Anthony Hoogs and Amitha G. Perera. Video activity recognition in the real world. In *AAAI Conference on Artificial Intelligence*, pages 1551–1554, 2008.
- [46] Weiming Hu, Xuejuan Xiao, D. Xie, Tieniu Tan, and S. Maybank. Traffic accident prediction using 3-d model-based vehicle tracking. *IEEE Transactions on Vehicular Technology*, 53(3):677–694, May 2004.
- [47] IntelliOne Technologies Corp. <http://www.intellione.com/d/iotraffic.html>.
- [48] ISO/IEC 14496-11:2005. Coding of audio-visual objects – part 11: Scene description and application engine, 2005.
- [49] S. J. Maybank J. M. Ferryman, A. D. Worrall. Learning enhanced 3d models for vehicle tracking. In *BMVC*, 1998.
- [50] Vishal Jain, Benjamin B. Kimia, and Joseph L. Mundy. Figure-ground segregation of object in video using curve-matching. In *European Conference on Computer Vision (ECCV)*, Prague, Czech Republic, May 2004.
- [51] Vishal Jain, Benjamin B. Kimia, and Joseph L. Mundy. Background modeling based on subpixel edges. In *IEEE International Conference on Image Processing (ICIP)*, volume 6, pages 321–324, 2007.
- [52] Vishal Jain, Benjamin B. Kimia, and Joseph L. Mundy. Segregation of moving objects using elastic matching. *Computer Vision and Image Understanding*, 108:230–242, 2007.

- [53] P. KaewTraKulPong and R. Bowden. An improved adaptive background mixture model for real-time tracking with shadow detection. In *2nd European workshop on Advanced Video Based Surveillance Systems*, September 2001.
- [54] R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.
- [55] N. K. Kanhere and S. T. Birchfield. Real-time incremental segmentation and tracking of vehicles at low camera angles using stable features. *IEEE Transactions on Intelligent Transportation Systems*, 9(1):148–160, March 2008.
- [56] J. Kato, T. Watanabe, S. Joga, J. Rittscher, and A. Blake. An hmm-based segmentation method for traffic monitoring movies. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(9):1291–1296, Sep 2002.
- [57] Z. Kim and J. Malik. Fast vehicle detection with probabilistic feature grouping and its application to vehicle tracking. In *International Conference on Computer Vision (ICCV)*, volume 1, pages 524–531, October 2003.
- [58] Lawrence A. Klein, Milton K. Mills, and David R.P. Gibson. *Traffic Detector Handbook*. U.S. Department of Transportation - Federal Highway Administration, third edition, October 2006.
- [59] D. Koller, K. Daniilidis, and H.H. Nagel. Model-based object tracking in monocular image sequences of road traffic scenes. *International Journal of Computer Vision*, 10(3):257–281, June 1993.
- [60] Dieter Koller. Moving object recognition and classification based on recursive shape parameter estimation. In *Israel Conference on Artificial Intelligence, Computer Vision*, pages 27–28, December 1993.
- [61] Dieter Koller, Joseph Weber, and Jitendra Malik. Robust multiple car tracking with occlusion reasoning. In *In Proc. Third European Conference on Computer Vision*, pages 189–196, Stockholm, Sweden, 1994.
- [62] Henner Kollnig and Hans-Hellmut Nagel. 3d pose estimation by directly matching polyhedral models to gray value gradients. *International Journal of Computer Vision*, 23(3):283–302, June 1997.
- [63] P. Kumar, S. Ranganath, Huang Weimin, and K. Sengupta. Framework for real-time behavior interpretation from traffic video. *IEEE Transactions on Intelligent Transportation Systems*, 6(1):43–53, March 2005.
- [64] Wendy Y. Lawton. Whodunit? computer scientists bring clarity to grainy surveillance video. Inside Brown <http://www.brown.edu/Administration/InsideBrown/2006/093006a.html>, Fall 2006.

- [65] Dar-Shyang Lee. Effective gaussian mixture learning for video background subtraction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27:827–832, May 2005.
- [66] Lisa Leff. Teen’s ticket hinges on gps vs. radar. USA Today http://www.usatoday.com/tech/news/techpolicy/2007-10-25-gps-vs-radar_N.htm, October 2007.
- [67] Bastian Leibe, Aleš Leonardis, and Bernt Schiele. Robust object detection with interleaved categorization and segmentation. *International Journal of Computer Vision*, 77(1-3):259–289, May 2008.
- [68] M.J. Leotta and J.L. Mundy. Learning background and shadow appearance with 3-d vehicle models. In *Proceedings of the British Machine Vision Conference (BMVC)*, volume 2, pages 649–658, September 2006.
- [69] M.J. Leotta and J.L. Mundy. Epipolar curve tracking in 3-d. In *IEEE International Conference on Image Processing (ICIP)*, volume 6, pages 325–328, September 2007.
- [70] J. Liebelt, C. Schmid, and K. Schertler. Viewpoint-independent object class detection using 3d feature maps. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, June 2008.
- [71] T. Lindeberg. *Scale-space theory in computer vision*. Kluwer Academic Print on Demand, 1994.
- [72] J. Lou, T. Tan, W. Hu, H. Yang, and S.J. Maybank. 3-d model-based vehicle tracking. *IEEE Trans. on Image Processing*, 14(10):1561–1569, October 2005.
- [73] David G. Lowe. Object recognition from local scale-invariant features. In *International Conference on Computer Vision (ICCV)*, pages 1150–1157, 1999.
- [74] D.G. Lowe. Fitting parameterized three-dimensional models to images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 13(5):441–450, May 1991.
- [75] Derek R. Magee. Tracking multiple vehicles using foreground, background and motion models. *Image and Vision Computing*, 22(2):143 – 155, 2004. Statistical Methods in Video Processing.
- [76] Donald W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):431–441, June 1963.
- [77] Mark Meyer, Haeyoung Lee, Alan Barr, and Mathieu Desbrun. Generalized barycentric coordinates on irregular polygons. *Journal of Graphics Tools*, 7(1):13–22, 2002.
- [78] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, Oct. 2005.

- [79] Joseph L. Mundy and Chung-Fu Chang. Fusion of intensity, texture, and color in video tracking based on mutual information. In *Applied Image Pattern Recognition Workshop*, pages 10–15, Los Alamitos, CA, USA, 2004. IEEE Computer Society.
- [80] Nathalie El Nabbut. Vehicle tracking in outdoor environments using 3d models. Master’s thesis, University of Waterloo, Waterloo, Ontario, Canada, July 2008.
- [81] Yutaka Ohtake and Alexander G. Belyaev. Dual/primal mesh optimization for polygonized implicit surfaces. In *Symposium on Solid modeling and applications (SMA)*, pages 171–178, New York, NY, USA, 2002. ACM.
- [82] M.M. Oliveira. Correcting texture mapping errors introduced by graphics hardware. In *Pacific Conference on Computer Graphics and Applications (PG)*, pages 31–38, 2001.
- [83] Mustafa Özuysal, Vincent Lepetit, and Pascal Fua. Pose estimation for category specific multiview object localization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 778–785, 2009.
- [84] Arthur E.C. Pece and Anthony D. Worrall. Tracking without feature detection. In *International Workshop on Performance Evaluation of Tracking and Surveillance*, pages 29–37, 2000.
- [85] P.W. Power and J.A. Schoonees. Understanding background mixture models for foreground segmentation. In *Imaging and Vision Computing New Zealand*, Auckland, NZ, November 2002.
- [86] Lawrence G. Roberts. *Machine Perception of Three-Dimensional Solids*. PhD thesis, Massachusetts Institute of Technology, 1963.
- [87] J. Schick and E.D. Dickmanns. Simultaneous estimation of 3d shape and motion of objects by computer vision. In *Proceedings of the IEEE Workshop on Visual Motion*, pages 256–261, Oct 1991.
- [88] C. Schlosser, J. Reitberger, and S. Hinz. Automatic car detection in high resolution urban scenes based on an adaptive 3d-model. In *GRSS/ISPRS Joint Workshop on Remote Sensing and Data Fusion over Urban Areas*, pages 167–171, May 2003.
- [89] Cordelia Schmid, Roger Mohr, and Christian Bauckhage. Evaluation of interest point detectors. *International Journal of Computer Vision*, 37(2):151–172, 2000.
- [90] J. Serra. *Image analysis and mathematical morphology*. Academic Press, Inc. Orlando, FL, USA, 1983.
- [91] Ying Shan, H.S. Sawhney, and R. Kumar. Unsupervised learning of discriminative edge measures for vehicle matching between non-overlapping cameras. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 894–901, June 2005.

- [92] Ying Shan, H.S. Sawhney, and R. Kumar. Vehicle identification between non-overlapping cameras without direct feature matching. In *IEEE International Conference on Computer Vision (ICCV)*, volume 1, pages 378–385, Oct. 2005.
- [93] Chen Shen, James F. O’Brien, and Jonathan R. Shewchuk. Interpolating and approximating implicit surfaces from polygon soup. In *Proceedings of ACM SIGGRAPH 2004*, pages 896–904. ACM Press, August 2004.
- [94] J. Shi and C. Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 593–600, 1994.
- [95] Tomas Singliar and Milos Hauskrecht. Towards a learning traffic incident detection system. In *in Proc. Workshop on Machine learning Algorithms for Surveillance and Event Detection*, 2006.
- [96] Joe Slaninka. Local teens claim pranks on county’s speed cams. The Montgomery County Sentinel <http://www.thesentinel.com/302730670790449.php>, December 2008.
- [97] Xuefeng Song and R. Nevatia. A model-based vehicle segmentation method for tracking. In *Tenth IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 1124–1131, October 2005.
- [98] C. Stauffer and W.E.L. Grimson. Adaptive background mixture models for real-time tracking. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 246–252, 1999.
- [99] Stealth Micro Systems Pty Ltd. <http://www.spectre-rdd.com/>.
- [100] Charles V. Stewart. Robust parameter estimation in computer vision. *SIAM Reviews*, 41(3):513–537, 1999.
- [101] Min Sun, Hao Su, Silvio Savarese, and Li Fei-Fei. A multi-view probabilistic model for 3d object classes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1247–1254, 2009.
- [102] A. Tamrakar and B.B. Kimia. No grouping left behind: From edges to curve fragments. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1–8, Oct. 2007.
- [103] T.N. Tan and K.D. Baker. Efficient image gradient based vehicle localization. *Image Processing, IEEE Transactions on*, 9(8):1343–1356, Aug 2000.
- [104] T.N. Tan, G.D. Sullivan, and K.D. Baker. Model-based localisation and recognition of road vehicles. *International Journal of Computer Vision*, 27(1):5–25, March 1998.
- [105] Gabriel Taubin. A signal processing approach to fair surface design. In *SIGGRAPH ’95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 351–358, New York, NY, USA, 1995. ACM.

- [106] Andrey N. Tikhonov and Vasiliy Y. Arsenin. *Solutions of Ill Posed Problems*. Winston, Silver Spring, Maryland, U.S.A., 1977.
- [107] J. van Leuven, M.B. van Leeuwen, and F.C.A. Groen. Real-time vehicle tracking in image sequences. In *IEEE Instrumentation and Measurement Technology Conference*, volume 3, pages 2049–2054, 2001.
- [108] D. Work, O.-P. Tossavainen, S. Blandin, A. Bayen, T. Iwuchukwu, and K. Tracton. An ensemble kalman filtering approach to highway traffic estimation using gps enabled mobile devices. In *47th IEEE Conference on Decision and Control*, pages 5062–5068, December 2008.