Fusion of Laser Range-Finding and Computer Vision Data for Traffic Detection by Autonomous Vehicles

Stephen James Cacciola

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science
in
Mechanical Engineering

Dr. Charles F. Reinholtz, Chairman

Dr. Alfred L. Wicks, Committee Member

Dr. Dennis W. Hong, Committee Member

3rd December 2007
Blacksburg, VA

Keywords: Autonomous Vehicles, Mobile Robotics, Sensor Fusion, Vision Processing

Fusion of Laser Range-finding and Computer Vision Data for Traffic Detection by Autonomous Vehicles

Stephen James Cacciola

ABSTRACT

The DARPA Challenges were created in response to a Congressional and Department of Defense (DoD) mandate that one-third of US operational ground combat vehicles be unmanned by the year 2015. The Urban Challenge is the latest competition that tasks industry, academia, and inventors with designing an autonomous vehicle that can safely operate in an urban environment.

A basic and important capability needed in a successful competition vehicle is the ability to detect and classify objects. The most important objects to classify are other vehicles on the road. Navigating traffic, which includes other autonomous vehicles, is critical in the obstacle avoidance and decision making processes. This thesis provides an overview of the algorithms and software designed to detect and locate these vehicles. By combining the individual strengths of laser range-finding and vision processing, the two sensors are able to more accurately detect and locate vehicles than either sensor acting alone.

The range-finding module uses the built-in object detection capabilities of IBEO Alasca laser rangefinders to detect the location, size, and velocity of nearby objects. The Alasca units are designed for automotive use, and so they alone are able to identify nearby obstacles as vehicles with a high level of certainty. After some basic filtering, an object detected by the Alasca scanner is given an initial classification based on its location, size, and velocity. The vision module uses the location of these objects as determined by the ranger finder to extract regions of interest from large images through perspective transformation. These regions of the image are then examined for distinct characteristics common to all vehicles such as tail lights and tires. Checking multiple characteristics helps reduce the number of false-negative detections. Since the entire image is never processed, the image size and resolution can be maximized to ensure the characteristics are as clear as possible. The existence of these characteristics is then used to modify the certainty level from the IBEO and determine if a given object is a vehicle.

Acknowledgements

It has been a long and difficult road, and although this thesis is complete I can already see new challenges on the horizon. But its completion does mark a significant milestone in my career which would not be possible without the tremendous support of my family and friends.

First and foremost, I'd like to acknowledge the support given by my advisor Dr. Charles Reinholtz, without whom I would not be where I am today. He saw potential in me, even at times when I did not, and I am honored to have had the opportunity to work with him the past three years. I also want to thank my other advisors Dr. Alfred Wicks and Dr. Dennis Hong. Dr. Wick's sound advice has helped me through this project, and he always makes sure I have data logs. And Dr. Hong, with his magical personality, has been a teacher and mentor of mine since my undergraduate years here at Virginia Tech.

I would also like to thank my family back at home in Rhode Island. Specifically I want to thank my mom Linda, who has always been a pillar of strength for me, and has encouraged me to succeed in whatever my future may hold. I also want to thank my dad James, who has supported me every step of the way, in all aspects of my life. And I can't forget my siblings Lauren, Billy, and Denise, who have helped fill my life with love and laughter.

I have also been gifted with a great group of friends, who are now scattered all over the country. While some have been closer to me at times, all have contributed to my life. I truly am a part of all that I have met. In this category, I especially want to thank the KHB, the best group of scoundrels that one could be a part of. I can't imagine having a better group of lifelong friends. I only hope that we can keep up the pace we've already set for the rest of our lives.

I have had the privilege of working with a great group of people on all of the Grand Challenges. There is not a more intelligent and dedicated group out there, and their hard work has served to push me even harder to achieve my goals. So thank you to Andrew Bacha, Brett Leedy, Cheryl Bauman, Mike Webster, Ruel Faruque, Dave Anderson, Pat Currier, Grant Gothing, Peter King, and all of the others I missed but have worked with along the way.

Last, but certainly not least, I want to thank my wonderful girlfriend Jenn. I can't possibly convey how important your support has been to me. Even through the early mornings, frustrating days, and late nights, you have always been there to help me continue onward. So for the laundry folded on my bed, the dinners in the microwave, and the little notes of support everywhere I look, I thank you from the bottom of my heart.

Table of Contents

List of Figures

List of Tables

List of Acronyms

CAN = Controller Area Network
CCD = Charge-Coupled Device
CMOS = Complementary Metal-Oxide Semiconductor
DARPA = Defense Advanced Research Projects Agency
DOD = Department of Defense
ECU = Electronic Control Unit
FOV = Field Of View
FPGA = Field-Programmable Gate Array
IEEE = Institute of Electrical and Electronics Engineers
GPS = Global Positioning System
INS = Inertial Navigation System
JAUS = Joint Architecture for Unmanned Systems
Laser = Light Amplification by Stimulated Emission of Radiation
LIDAR = LIght Detection And Ranging
LLC = Limited Liability Company
LRF = Laser Range-finder
MDF = Mission Definition File
NQE = National Qualification Event
OCR = Optical Character Recognition
ROI = Region Of Interest
RNDF = Route Network Definition File
SUV = Sports Utility Vehicle
UCE = Urban Challenge Event
UGV = Unmanned Ground Vehicle
VT = Virginia Tech

Chapter 1
Background and Introduction

Unmanned vehicles have had a long history of utilization on the battlefield. The first recorded use of an unmanned vehicle on the battlefield was in an Austrian attack on Venice in 1849. The Austrians, who controlled much of Italy at the time, were having trouble getting artillery in range of the city due to the numerous canals and lagoons. In response, the Austrians launched around 200 unmanned balloons carrying bombs with timed fuses over the city. Although some of the bombs were dropped on the city as planned, others were sent back over the Austrian lines due to a change in the winds [Naughton 2007]. Since then, unmanned aerial vehicles have been present in numerous conflicts from World War I to Vietnam to the present day conflicts in the Middle East.

A more recent advance in the field of unmanned systems is the development of unmanned ground vehicles (UGVs). In general, ground vehicle require greater intelligence in order to successfully navigate their more complex surroundings. A key component of successful navigation is the ability to accurately identify and classify objects in the environment. To this end, a novel object identification and classification system is presented for the specific use in the DARPA Urban Challenge. The system uses the advantages of range-finding and monocular vision technologies to locate and classify objects of interest in the environment, specifically other vehicles.

This thesis examines the design of the object classification system for team *V*ictor *T*ango's entry to the Urban Challenge. A brief history of the Grand and Urban Challenges are given in chapter one. Chapter two details team *V*ictor *T*ango and its entry vehicle, Odin. Chapter three reviews some existing methods for sensor fusion and visual vehicle detection. Chapters four and five then describe the individual details of the range-finding and vision processing system, respectively. Chapter six details the overview of the classification module as a whole, as well as many special features used in the Urban Challenge software architecture. Finally, Chapter seven reviews the test results of the module and possible future improvements to the software.

1.1 The Grand Challenges

The DARPA challenges were created as the result of the National Defense Authorization Act for Fiscal Year 2001, Public Law 106-398. This mandate states that: “It shall be a goal of the Armed Forces to achieve the fielding of unmanned, remotely controlled technology such that…by 2015, one-third of the operational ground combat vehicles are unmanned” [DARPA 2007]. To spur the accelerated development of unmanned group vehicles, DARPA created the Grand Challenges.

The objective of the first two Grand Challenges was to create a vehicle that could navigate more than 150 miles of rough, desert terrain without any human interaction. The competitions were extremely well received, with hundreds of teams applying to compete in the two races and worldwide coverage of the events. The first competition ran on March 13th 2004, and ended as a moderate success with no vehicles traveling more than eight miles. Virginia Tech’s entry, Cliff, was ranked fifth after the qualification event, but had its brakes lock up shortly after starting the race. The second Grand Challenge, held on October 8th 2005, saw five teams complete the 132 mile course. Virginia Tech’s two entries Cliff and Rocky, shown in Figure 1-1, finished 8th and 9th after traveling 38 and 43 miles respectively. The technologies developed in preparation for these Challenges are vital to the future of unmanned ground vehicles for both government and commercial applications.

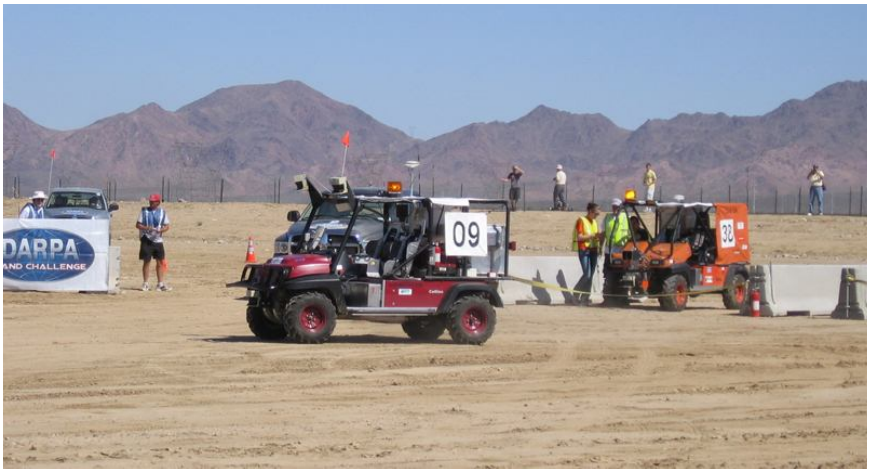


Figure 1-1: The start of the second DARPA Grand Challenge. Rocky is in the foreground, having just left the chute, and Cliff getting ready to leave in the background.

1.2 Urban Challenge Overview

After the success of the Grand Challenges, DARPA announced the creation of the Urban Challenge on May 1st, 2006. The previous competitions required the vehicles to travel long distances over rugged terrain, which emphasized building a robust platform and software solution. The Urban Challenge required autonomous vehicles to complete 60 miles of urban missions in less than six hours.

Two different data files were given to each team that defines the environment and objectives. The first is a Route Network Definition File (RNDF), an example of which can be seen in Figure 1-2. This text file contains information about all of the segments (roads) and zones (lots) on the course. Each segment is made up of multiple lanes, which in turn contains multiple waypoints. The spacing and path between these waypoints can vary greatly; therefore, the vehicle must have the ability to stay on course using more than just its global position. A certain number of these waypoints are flagged as checkpoints and given a unique numbered identifier. It is also possible that segments may be temporarily or permanently blocked, which requires the vehicle to have the ability to autonomously re-plan an alternate route to its destination.

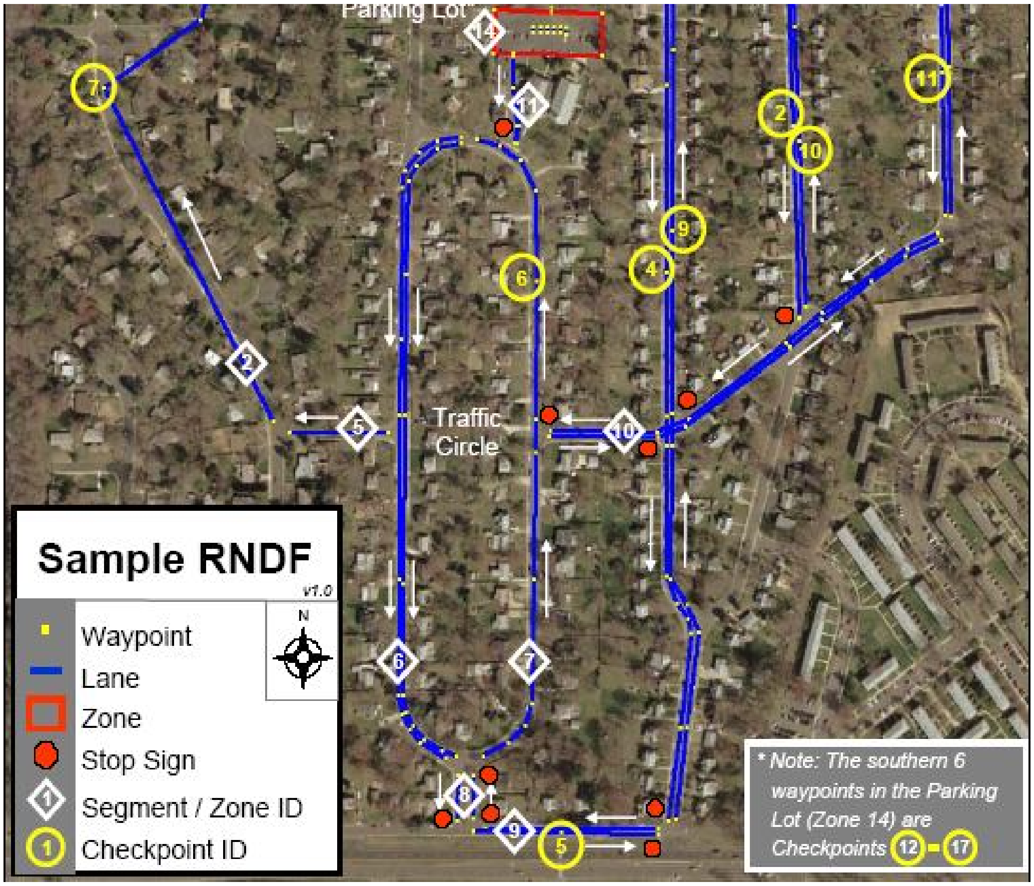


Figure 1-2: A section of the sample RNDF supplied by DARPA.

The second file, the Mission Definition File (MDF), defines a route the vehicle must complete within the course. It contains a list of checkpoints the vehicle must cross in sequential order as well as the speed limit that must be maintained in each section of the course. While the RNDF is supplied to the teams at least 24 hours in advance, the MDFs must be received and loaded only five minutes before each mission begins. Vehicles must complete these missions while navigating through stop-and-go traffic, merging with other vehicles, and avoiding static and dynamic obstacles.

1.3 Classification Requirements

The focus of the Urban Challenge was on intelligent behaviors and decision making in urban environment. To accomplish the required tasks, the vehicles had to first be able to perceive their surroundings. A flawless decision making process can still cause the vehicle to fail if there is misinformation from the perception software. Odin therefore had to be able to accurately locate all objects in its vicinity and correctly classify them.

There were a number of simplifications to the Challenge that differentiate it from autonomous driving on actual city streets. There were no traffic signs or lights of any kind, only stop lines whose locations are explicitly defined in the RNDF. Due to safety reasons, there were no dynamic obstacles on the course other than manned and unmanned vehicles, and these vehicles were only present on segments and zones listed in the RNDF. These two pieces of information greatly reduced the number of unique obstacles that needed to be identified, as well as the locations that they can legally exist. Finally, the autonomous vehicles had to be built on a full-sized chassis with a documented safety record, which implied that they have the same physical features of typical cars such as lights, signals, and tires.

The official rules stated that the speed limits on the course would be monitored closely, and did not exceed 30 miles per hour. The distance at which Odin must detect a vehicle is dependent on this speed as well as the amount of time required to sense and avoid it. With a maximum differential speed of 60 miles per hour, an oncoming vehicle must be detected and classified at least 176 feet (54 meters) away assuming Odin can navigate to avoid the object in two seconds. However, a greater distance is required to perform any advanced planning around the same vehicle.

The system must also be able to detect obstacles all around the vehicle. All objects can be defined as either a static object that will never move or a dynamic object that is, or has the potential to be, in motion. From the previous assumptions all large moving objects are assumed to be manned or unmanned vehicles. Static objects must be examined closely to determine if they are a vehicle capable of motion. The classification of these stationary dynamic objects is particularly important in cases such as vehicles being queued at an intersection or being temporarily disabled. However, only stationary vehicles in front of Odin affect its planning, and therefore only these require precise classification.

While running the course Odin interacted with many different vehicles. Depending on the situation, these vehicles can be oriented any number of ways with reference to the base platform. Manned traffic vehicles were typical cars such as those seen on everyday streets. Although unmanned vehicles must be built on a full-sized chassis, they may have a number of alterations that modify their characteristics. Therefore the system must be able to correctly classify vehicles independent of size, color, and orientation.

Chapter 2
Team *V*ictor *T*ango

A large task such as the Urban Challenge could not be successfully accomplished by a small group. It requires dozens, if not hundreds, of individuals focusing on tasks ranging from vehicle systems to electronics to navigational software. Any single hardware system or software module is a large enough problem to require the time and effort of multiple team members. Likewise, each system has the potential to end the final competition run if it fails.

Team *V*ictor *T*ango was created shortly after the announcement of the creation of the Urban Challenge, although the name was not coined until a few months later. Using the combination of phonetic alphabet words for Virginia Tech, it represents the VT core of the team while leaving room for others to join. This section briefly describes the structure of team *V*ictor *T*ango, as well as the summary of the hardware and software systems of its entry vehicle, Odin.

2.1 Team Summary

Team *V*ictor *T*ango was formed through a partnership between Virginia Tech and TORC Technologies LLC. TORC is a start-up company founded by a group of faculty and ex-graduate students from Virginia Tech with experience in intelligent robotics. These members were integral to the success of the previous Virginia Tech Grand Challenge and IGVC teams. *V*ictor *T*ango also identified a number of leaders in the robotics community and associated industries to serve as an advisory board for the team. While they are not official team members, these advisors bring their past expertise to assist the team in identifying key technologies and developing sound planning strategies.

The most unique element of the team is the involvement of the Virginia Tech students. Four faculty advisors, eight graduate students, and the TORC employees advise almost 50 undergraduate students participating in the project as part of their senior capstone design class. Most undergraduate students have never been involved with any aspect of unmanned systems, so some initial time and effort is spent making them familiar with topics in unmanned system and team practices. By the end of the academic term they have learned enough to contribute a large number of useful man-hours to the project. The overall team structure is shown in Figure 2-1.

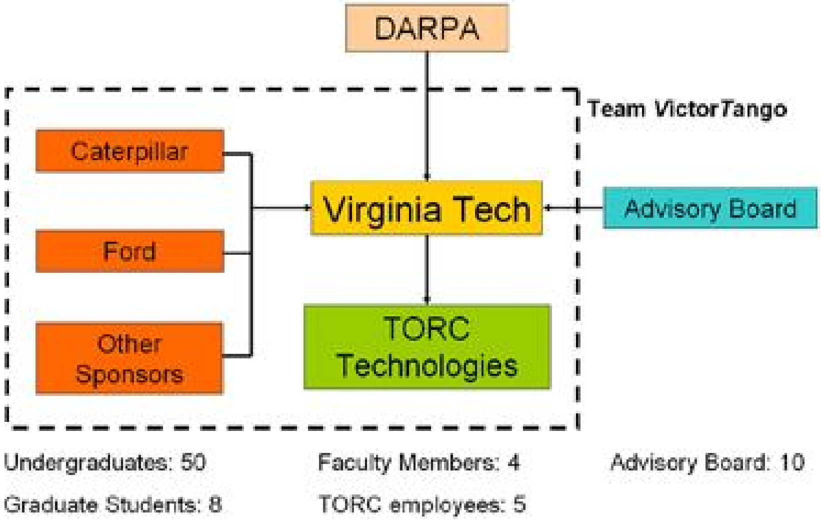


Figure 2-1: An overview of the structure of team *V*ictor *T*ango.

Team *V*ictor *T*ango is one of only eleven teams which have received up to one million dollars in funding through DARPA for the Urban Challenge. However, a number of other sponsors have also signed on to donate money, equipment, and technical expertise to the team. Some of these sponsors include National Instruments, Caterpillar, and Ford Motor Company. With their support, *V*ictor *T*ango has managed to develop its entry to the Urban Challenge: Odin. Named after the chief god in Norse mythology, Odin was the god of many things such as inspiration, poetry, and victory. He was also a collector of information, sending his two ravens Huginn (Thought) and Munin (Memory) out to collect tidings from around the world. A persisting theme in the stories of Odin is self-sacrifice for the chance to gain knowledge and power, a concept the entire team can relate to.

2.2 Odin Vehicle Systems

Odin's base platform is a 2005 Hybrid Ford Escape, a medium-sized SUV. As a stock vehicle it satisfies all of the platform requirements for the competition, while still having ample space to convert it to autonomous control. The platform contains numerous technical advantages that distinguish it as an ideal choice.

The largest advantage is the on-board power systems that preexist on a hybrid vehicle. A 330 volt battery pack is contained in the rear of the car that provides enough power to run all of the sensors and computers on board while maintaining all of the stock functions. Since the entire vehicle's necessary power can be drawn from this source, there is no need for an onboard generator that takes up valuable space and requires extra fuel to operate. It also allows all power systems to be drawn from the same source. This is accomplished by stepping down the voltages as shown in Figure 2-2. A distribution box was created for a 12 and 24 volt systems that allow each component to be powered individually.

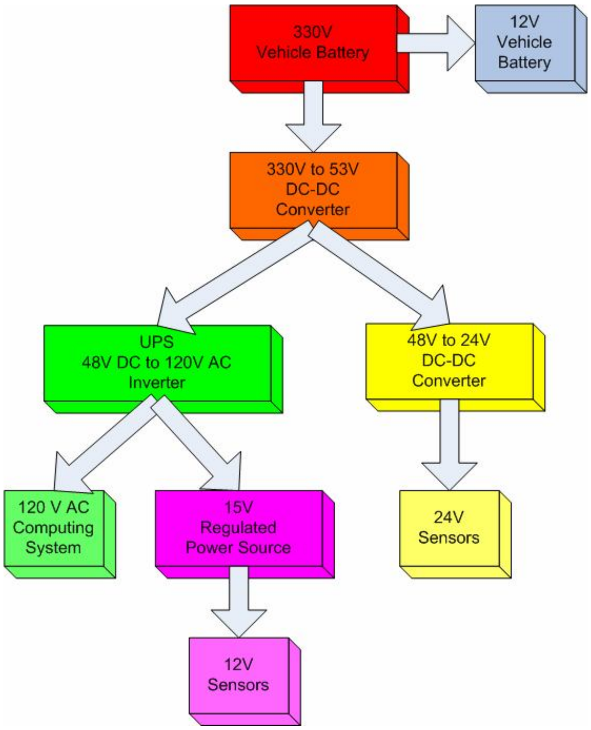


Figure 2-2. Flowchart of the power system available on Odin. All power originates from the 330 V hybrid batteries in the rear of the vehicle.

Another extremely useful feature of the Hybrid Escape is the preexisting drive-by-wire systems. In order to convert a vehicle to autonomous control, the braking, steering, throttle, and shifting must be modified or replaced with an electrically controlled system or actuator. Hybrid Escapes comes with a drive-by-wire throttle, shifting, and braking that are controlled by electric signals generated by the gas pedal, steering wheel, and brake pedal respectively. This allows the team to simply tap into the wires that transmit the controlling signal and replicate it through the team's computers. Ford has supplied the team with information on how to generate these signals. However, the vehicle comes with a number of fail-safes, one of which creates a fault if an equivalent mechanical action is not used to depress the brake pedal. Because the team could not work around this fault, an electrically controlled mechanical actuator was installed that physically depresses the brake pedal instead of using the built-in braking system. Finally, while the steering system is not directly drive-by-wire, it can be controlled through the torque assist motor on the existing steering assist system.

There are three main computing systems on board Odin. The first is a National Instruments Compact RIO, an integrated FPGA and controller. This device is mounted in the glove box and generates all of the signals that control the motion of the vehicle. The other two computers are two quad core rack mounted servers in the rear of the vehicle, shown in Figure 2-3. The main difference between the two servers is that one, nicknamed Bill, runs a Windows operating system while the other, nicknamed Linus, is running a Linux operating system. Due to a number of functions only available on Windows, any software that deals with vision processing must run on the Bill computer. The Linus computer is sectioned into multiple virtual machines, or discrete partitions of the overall processing power of the computer. By creating these virtual machines (named Alpha, Bravo, Charlie, etc), each is able to have one or more of the available processing cores dedicated to it, which allow multiple processes to be run without stealing processing power from one another.

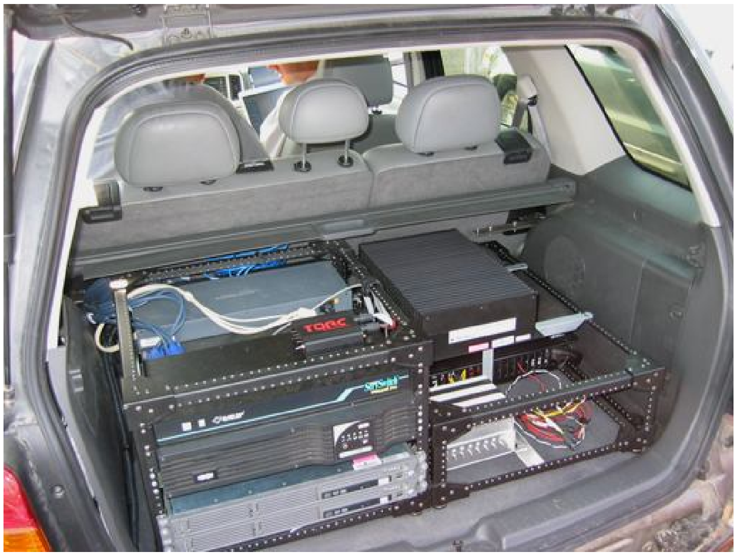


Figure 2-3. The power and computing systems mounted in the rear of Odin. The two servers Bill and Linus can be seen mounted on the bottom-left corner of the rack.

2.3 Odin Software Architecture

The software architecture, depicted in Figure 2-4, was designed to take advantage of the large size of team *V*ictor *T*ango. Each module is a sub-divided task of the problem as a whole, and therefore can be worked on independently. While this structure makes each module easier to work with, it complicates the interactions between the different components. To better define those interactions, an extensive software architecture document was created detailing the inputs, outputs, and functional requirements of each module.

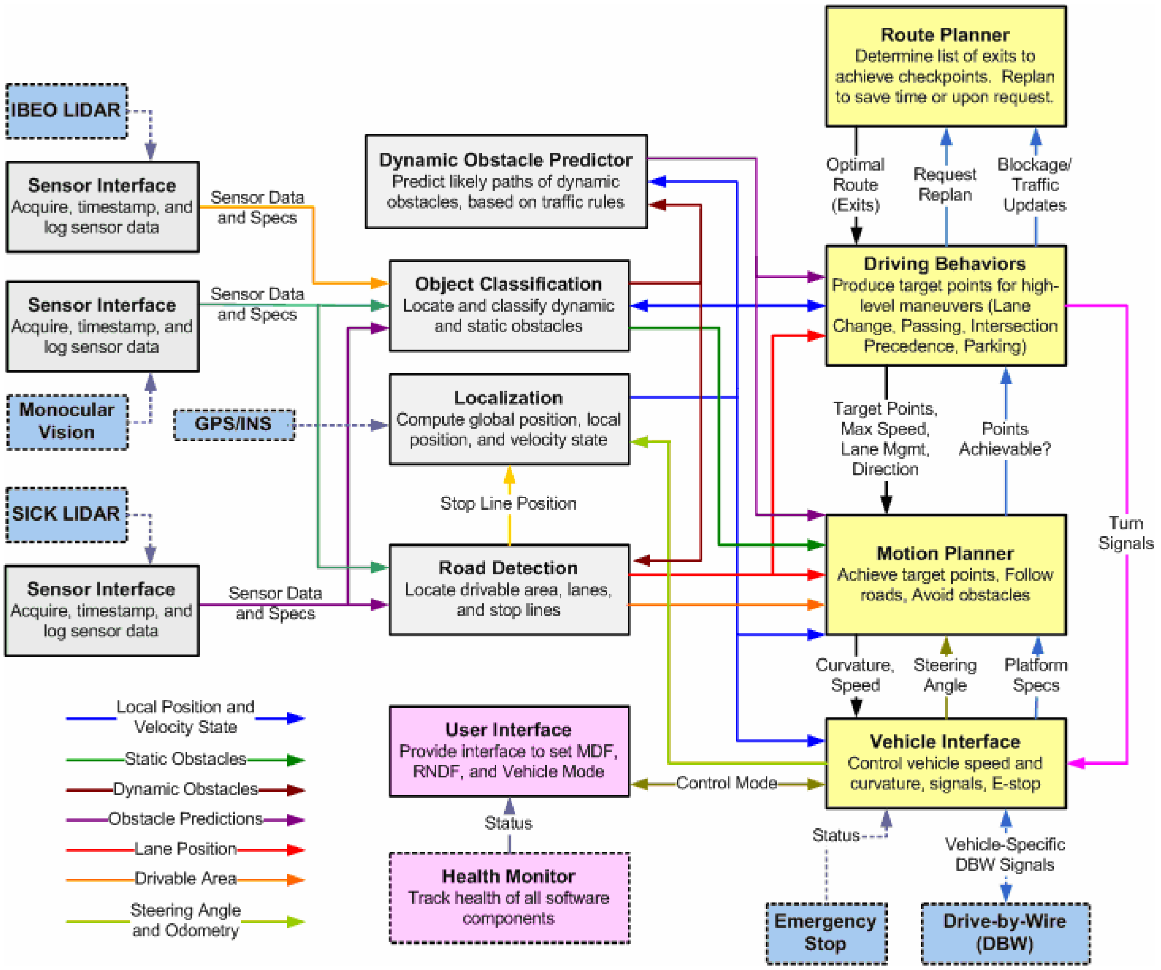

IBEO LIDAR
Sensor Interface
Acquire, timestamp, and log sensor data
Sensor Data and Specs
Sensor Interface
Acquire, timestamp, and log sensor data
Sensor Data and Specs
Monocular Vision
GPS/INS
SICK LIDAR
Sensor Interface
Acquire, timestamp, and log sensor data
Sensor Data and Specs
Dynamic Obstacle Predictor
Predict likely paths of dynamic obstacles, based on traffic rules
Object Classification
Locate and classify dynamic and static obstacles
Localization
Compute global position, local position, and velocity state
Stop Line Position
Road Detection
Locate drivable area, lanes, and stop lines
Route Planner
Determine list of exits to achieve checkpoints. Replan to save time or upon request.
Optimal Route (Exits)
Request Replan
Blockage/ Traffic Updates
Driving Behaviors
Produce target points for high-level maneuvers (Lane Change, Passing, Intersection Precedence, Parking)
Target Points, Max Speed, Lane Mgmt, Direction
Points Achievable?
Turn Signals
Motion Planner
Achieve target points, Follow roads, Avoid obstacles
Curvature, Speed
Steering Angle
Platform Specs
Vehicle Interface
Control vehicle speed and curvature, signals, E-stop
Control Mode
Status
Vehicle-Specific DBW Signals
Emergency Stop
Drive-by-Wire (DBW)
User Interface
Provide interface to set MDF, RNDF, and Vehicle Mode
Status
Health Monitor
Track health of all software components
Local Position and Velocity State
Static Obstacles
Dynamic Obstacles
Obstacle Predictions
Lane Position
Drivable Area
Steering Angle and Odometry

Figure 2-4. The overall software architecture for Odin. Light blue boxes are perception modules, while yellow boxes denote decision making modules. Color coded arrows show the flow of information.

The flow of information starts with the input of sensory data. Here a number of perception modules take in data describing the vehicle and the surrounding world. After performing their functions, the perception modules output discrete information for the decision making modules to use. Localization takes in INS and odometer inputs in order to establish Odin's best known position in the world. This module is the only one to know the vehicles global position. All other modules work in either the local or vehicle coordinate frames, which are further described in Figure 2-5. The road detection module uses information from the cameras, LIDAR, and the RNDF to determine the location of nearby drivable areas. Finally, obstacle classification takes in LIDAR and camera data to identify and classify objects in the vicinity of Odin, and will be the focus of the remainder of this paper. A secondary program, the dynamic object predictor, uses all of the processed sensory data to calculate future paths for nearby dynamic obstacles.

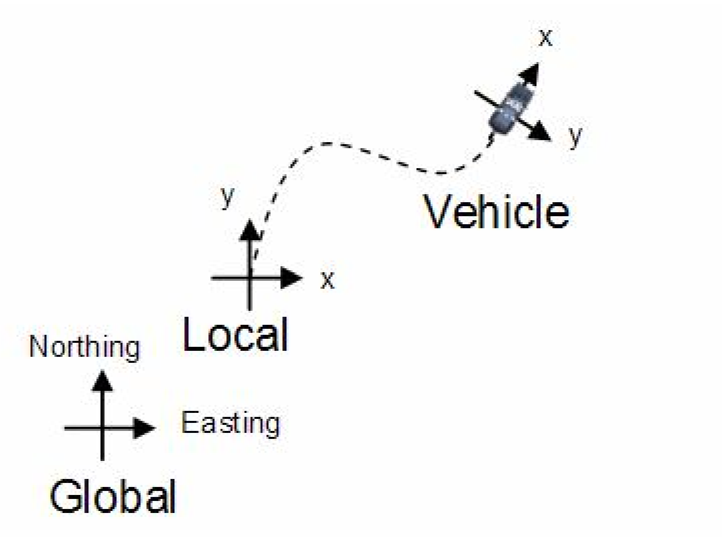


Figure 2-5. The global, local, and vehicle coordinate frames. The local frame origin is defined as where Odin is first started, and the vehicle frame origin is located at the center of the vehicles rear axle.

Once the perception modules define the location of Odin, the roads, and nearby objects, the decision making modules use this information to plan Odin's way through the environment. This process begins with the router planner, which takes in the RNDF and MDF to determine the shortest path to the vehicles destination. This module can be compared to a map search program such as Mapquest that outputs the optimal route to achieve a destination. To ensure a more accurate determination of the quickest route, the planner can also be supplemented with information discovered while navigating the course, such as if certain segments are blocked. Using this general route around the course, the driving behaviors module determines what high-level vehicle maneuvers must be applied while navigating the course and outputs an array of target points for the vehicle to achieve. For each target point, a number of Boolean flags are set which dictate if the vehicle must stay in its lane, if it can go in reverse, and how fast it is allowed travel. The motion planning module then takes these target points and translates them into vehicle actions, while also avoiding any static objects in its path. This is accomplished by creating a vehicle profile array that defines the detailed vehicle controls to achieve over time. These profiles are passed to the vehicle interface, which executes the commands.

A number of other functions exist in the software architecture that interact with many of the modules. The user interface module is the main method for communicating with the base platform. Along with controlling the mode of the vehicle, it also is the interface for loading the RNDFs and MDFs. A health monitoring process collects all error messages from each module. With this information it determines if the vehicle is completely functional, is able to run at reduced functionality, or should stop for safety reasons. Finally, an emergency stop function monitors the vehicle and remote e-stops, and puts the vehicle in the appropriate emergency state if either is activated.

Almost all messaging traffic is sent using the Joint Architecture for Unmanned Systems (JAUS). This architecture is an emerging standard for unmanned systems that specifies messages and transfer protocols to use between multiple systems, computers, and programs. While the team used this framework to give structure to the software architecture, it was not limited to the current messaging set, and created a number of user-defined messages when needed. These messages will eventually be proposed to the JAUS working group for inclusion into the standard. Certain types of data, such as images, were not sent through the JAUS standards due to their large bandwidth requirements. Because all of the data for these few cases all existed on the same computer, a shared memory interface was created for modules to write to and read from quickly.

Chapter 3
Methods of Object Detection and Classification

The successful integration of data is fundamental to all of Odin's perception modules. The classification module uses multiple sensors to produce a single output, and so a suitable method is required to fuse these multiple inputs. While this concept is simple in theory, the process to determine how to use each sensor to its fullest potential can be complicated in practice. Combining, or fusing, this data can be just as complicated, if not more so. There are several different ways of fusing data from multiple sensors that will be presented in this chapter, along with the advantages and disadvantages of each.

The main focus of the classification module is to identify any vehicles Odin sees while traversing an Urban Challenge RNDF. The classification module must also be able to locate any other static obstacles that could impede the mission. Static objects only need to be defined by their position and outline. However, due to the complexity that mobile vehicles can exhibit, they must be defined by their location, size, and velocity. This also makes it extremely important that every vehicle which could interact with Odin be correctly located and classified. Although laser range-finders can locate objects with accurate results, images contain the most information to classify them correctly. This chapter will describe various methods for visual vehicle detection under a variety of circumstances.

3.1 Sensor Fusion

The idea of fusing multiple sources of information into a singular decision is not a recent development. In fact, it is easy to trace a number of fusion methods to biological origins. Humans rely on many different senses to perceive the objects in their surroundings. Where eyesight may be enough to determine if a round green object in a refrigerator is an apple, the senses of touch, smell, and taste can also be used to differentiate between a rotten apple and one that is still edible. All five human senses are fused together in the brain, which also takes into account past experience of similar situations. Knowing that an apple found in a similar way the day before was rotten can often override current sensory information that is contradictory.

Whenever multiple sensors are used towards a singular purpose, a method to fuse the data is required. The most fundamental definition of data fusion is the hierarchical transformation between the observed parameters and a resulting decision of the location, characteristics, and identity of an entity [Hall 1997]. The definition of what an entity is depends on the application of the fusion. Different applications also can require various levels of inference about the detected objects. An overview of these different inference levels is shown in Figure 3-1. The higher an entity is on this scale, the more information is required about it. For example the existence of an entity only requires a Boolean output that can be generated by any number of various sensors without any prior knowledge of what the object is. Meanwhile a threat analysis decision requires information such as an interpretation of an objects characteristics and movements in reference to its intent towards the viewing platform.

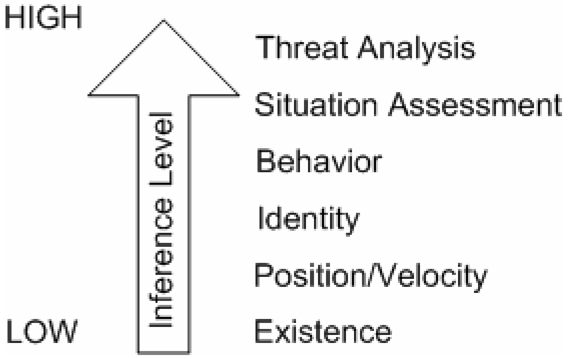


Figure 3-1. An overview of the levels of inference required by fusion applications.

Observational data can be combined at three levels. At the first level, raw sensor data can be combined directly, providing the sensors are measuring the same physical quantity. For example, the raw data from two range-finding sensors could be combined to achieve a level of redundancy. The second level, feature-level fusion, requires that multiple sensors extract different observations of the same object. These features are combined into a single feature vector that is examined by a central process. Often this process attempts to match the object to prior models using techniques such as pattern recognition. The third and final level, decision-level fusion, requires each sensor type to make an initial classification of an entity. The results of each sensor's classification are then combined in a central system, usually involving some form of voting method.

For a decision-level fusion, the reliability of each individual sensor is an important consideration. When multiple sensors are used towards the same goal it is often difficult to determine which one, if either, is correct. For example if a range-finder and camera are both used to find the relative position of an object, it is important to know that the range-finder is much more accurate in this task than the camera. The typical method for combining multiple sensors using decision-level fusion is to determine the accuracy of each individually, then develop a weighting function that decides which is the most accurate in various cases. A more advanced technique involves using aprioi knowledge of how environmental conditions affect the sensors' reliability. This technique can adjust the accuracy of each sensor depending on external conditions, such as an area of low light or the presence of precipitation. To implement this technique a database of rules for various conditions can be created and a different set of rules are used for each condition [Futoshi 2001].

While using two sensors that measure the same physical characteristic can result in more precise classification, the real strength of decision level fusion is exhibited when heterogeneous sensors are used. In this way the strengths of multiple types of sensors can be used to more accurately develop an accurate classification. Using this principle, a number of systems have been developed for specific applications that fuse the data coming from laser range-finding and monocular vision. One such fusion uses a range-finder and monocular camera to independently find nearby corners and planes in a typical indoor structured environment. In this case the range-finding unit is used to locate wall corners and "semiplanes". Simultaneously a CCD camera uses a segment extraction process to find visual edges. The results of both are fused together by attempting to match the laser features to a corresponding visual feature [Castellanos 1996].

However, this method only uses two different sensors to examine the same physical characteristic. A second system uses the data from both a CCD and a ranging camera. This system is able to enhance a range map produced by the ranging camera with color and texture from the monocular camera. The result is the ability to discern a physical characteristic, such as an edge, by looking at where the distance, color, and texture all change in the same location of the image [Jasiobedzki 1997].

3.2 Visual Vehicle Detection

There are numerous available methods for detecting and classifying traffic vehicles using computer vision. However, most existing algorithms are designed for specific situations. These applications can range from the detection of moving vehicles by a stationary camera for analysis of traffic patterns, to the detection of vehicles from a moving platform for research towards adaptive cruise control systems. Existing methods of vehicle detection do not address the general problem of classifying stationary and moving vehicles from a generic angle. Regardless, some of these techniques do provide a baseline for developing a visual classification routine.

One of the simplest cases involves vehicle detection and classification from a stationary camera. If the camera itself is motionless, stationary objects will appear the same from one frame to the next. In this situation it is a relatively simple process to segment moving objects from a stationary background across multiple frames. One method that uses these concepts divides portions of the image based on observed motion. These moving segments are detected by running a threshold through the spatial and time gradients of the image. The resulting regions are considered to be moving objects and surrounded by closed cubic splines. The motion of each object can be approximated using an affine transformation based on the objects translation and difference in scale along the plane of the road. Finally, partially occluded objects can also be accounted for using this process by noting that closer objects will always block further objects. Therefore occlusions of bounding contours can be allowed if the objects are processed starting at the bottom of the image [Koller 1994]. However, this method is not able to distinguish a stationary car from the background of the image, and as previously mentioned only works if the camera is stationary. An example of this process at work can is shown in Figure 3-2.

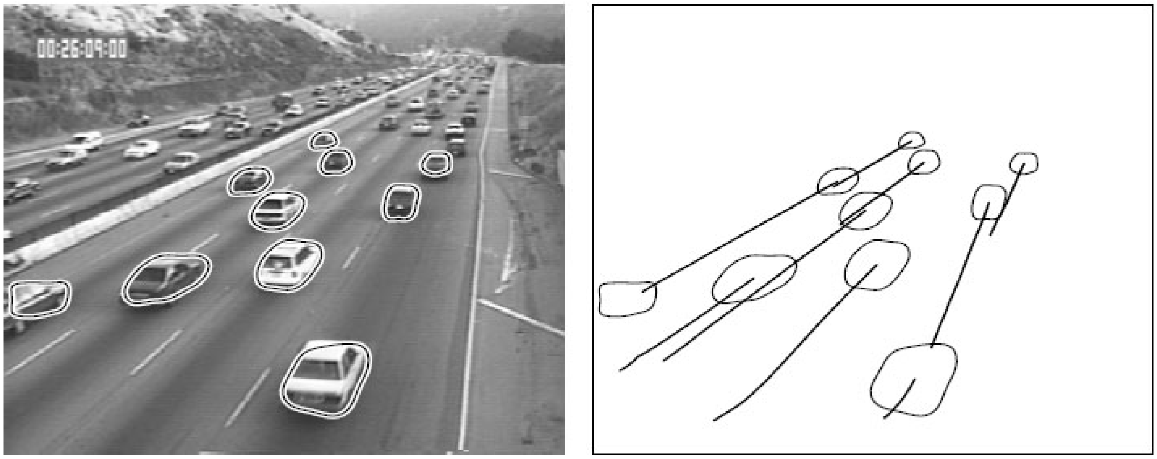


Figure 3-2. A frame of an algorithm based on motion segmentation from a static camera [Koller 1994].

If the camera is mounted on a moving platform, visual vehicle detection becomes more difficult due to the lack of a constant background image to extract from. There have been a number of methods suggested to achieve accurate, efficient vehicle detection using only monocular cameras. One traditional computer vision technique requires the creation of a large database of samples. This set serves as a learning tool, and all future objects are checked against the learned set. For the case of vehicle detection, there are many different types of different road vehicles that can be seen, and therefore each one must be represented in this learned set. In one algorithm, each candidate region is split into three sub-regions. Each region is run through a Canny edge detection routine to enhance the edges. The Canny edge detector operates by blurring the image with a Gaussian convolution filter to eliminate any noise. It then finds the edge strength of each pixel using the magnitude of two 2-D Sobel convolution matrices, and relates the edge direction of each pixel to a traceable path in the pixilated image. After this operation is complete, the sub-regions are each checked against the pre-learned vehicle data set [Sotelo 2005]. Another similar system uses the same concept of training images, but uses a combination of vehicle and non-vehicle sample windows. A feature vector for each window is created through a Raster scan of the pixels. To successfully detect a vehicle, the feature vector must closely match that of one in the training set [Kato 2002]. Both of these methods require a large number of samples to account for all possible vehicle models, and they only function from a single orientation. They also can not handle if the vehicle is partially blocked from view.

Although algorithms based on learned sample sets can be successful in specific situations, other methods have been developed that are able to detect a vehicle without any preprocessing. One such method detects the road area in front of the platform, and then searches that area for candidate regions. Within each region, the system searches for a combination of horizontal and vertical edges that could signify an object. These objects are checked for characteristic edges and corners, such as those made by the lower edge of the vehicles body or the outer corner of the wheels. If a significant number of these features are found, and if their relative locations are as expected, the object is considered to be recognized as a car [Graefe 1993]. Although this method has only been applied to preceding vehicles, it could theoretically be expanded to be able to classify vehicles from other orientations as well.

A common thread through all the visual methods is that it is challenging to develop a technique to visually detect edges for all operating environments. Many times the edges between the vehicle and the background objects are unclear due to similar color or lighting. One possible way to separate objects using monocular vision that does not suffer from this shortcoming is through the concept of optical flow. Optical flow is the apparent visual motion of static objects that occurs when an observing body moves through the world. It is also possible to fuse 2-D pixel motion with ranging information to obtain a true 3-D pixel velocity. In either case, by observing the discontinuities in pixel motion, objects can be segmented and located in the world. If the motion of the platform is known, then this same concept can be applied to detect moving vehicles while in motion as well. In this case moving objects can be detected by observing discontinuities in the motion flow due to the platforms motion [Talukder 2003]. This serves to isolate objects within the image, but can not be directly used to classify any as a vehicle based on anything besides its size.

Chapter 4
Range-Finding Systems

The base of the object classification module is the laser range-finding units. Though they may look like the eyes of the vehicle, their function more closely mimics the sense of touch than that of sight. Also called LIDAR (LIght Detection And Ranging), laser range-finding functions much like sonar and radar in that pulses of energy are emitted and their time of flight is measured upon return. However where sonar uses sound waves and radar uses electromagnetic waves, LIDAR uses laser pulses. By knowing the amount of time the pulse takes to return, and knowing the speed of light for the medium, the distance to the target can be calculated.

In order to measure multiple distances with a single range-finding unit, many models incorporate rotating mirrors that reflect the laser pulses at various angles. An example of the components used in this process is shown in Figure 4-1. An infrared laser diode emits the laser through a lens. A rotating mirror is used to direct the laser pulse out into the environment at a specific angle. The angle is known through an encoder attached to the motor that spins the mirror. Once the pulse contacts an object it will partially reflect back to the unit, bouncing off the mirror and passing through the lens where it is detected by a receiver. Knowing the distance to the object and its relative angle to the sensor, the object's position relative to the sensor can be calculated.

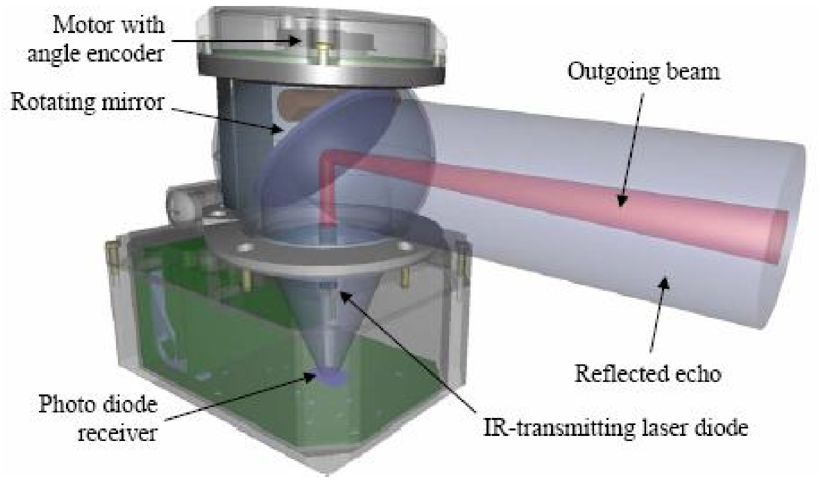


Figure 4-1: The components of a rotating-mirror laser range-finder [IBEO 2006].

Since LIDAR units operate using the speed of light, the cycle time for a single pulse is extremely small compared to sonar units. An improved angular resolution and longer measurement range over radar can also be obtained due to the coherent nature of laser light. In coherent light the particles are all in phase in both space and time, resulting in the narrow beam of light expanding only over long distances. However laser range-finding units do have a number of disadvantages, the main one being a limited field of view. A wide field of view can only be obtained in one dimension with a rotating mirror. It is also possible for the laser pulses to completely reflect off objects, resulting in holes in the scan area. Finally, the scans only present a thin profile of the objects in view, which makes any observations about the object other than its position and size difficult to formulate.

This chapter details the range-finding portion of the object classification module. The technology behind Odin's range-finding units, the IBEO XT and IBEO A0, is described. An overview of the software used to interact with the range-finders, along with the processes that attempt to determine the validity and classification of the detected objects, is also presented.

4.1 IBEO Hardware

The range-finders used on Odin are designed by IBEO, a German automotive sensors company. Two of their feature products, the IBEO XTs, are mounted on the front two corners of Odin as seen in Figure 4-2. These two units are part of a system called the Fusion, and can be coordinated so that objects seen in their overlapping scan areas produce a single output. This is accomplished by precisely calibrating them so that the position of an object relative to a common coordinate system is the same for both sensors. These two sensors must always be used together, as one will not function if the other is not present. They are contained in enclosures that allow a 320 degree horizontal field of view at a one-degree horizontal scan resolution. However, due to the placement and mounting of the sensor, they are only able to see 260 degrees in front and to the side of Odin. Their advertised maximum range is 200 meters, although most objects past 100 meters are unable to be accurately distinguished. At this distance each scan point is 1.75

meters apart, resulting in only two to three points reflecting off a vehicle, which is insufficient for accurate object detection. A third IBEO sensor is also mounted on the center of the rear bumper to observe vehicles behind Odin. This unit, an older IBEO model called the A0, is in an enclosure that only allows an approximate 150 degree horizontal field of view. Other than the modified housing, the A0 still contains all of the same hardware and software as the XTs. The Fusion system can operate at multiple refresh rates, although 12.5 Hz (80 ms) is the default factory setting. Due to a hardware issue with the older internal motor, the A0 unit can only operate with a refresh rate that is a multiple of 10, and so it runs at 10 Hz.

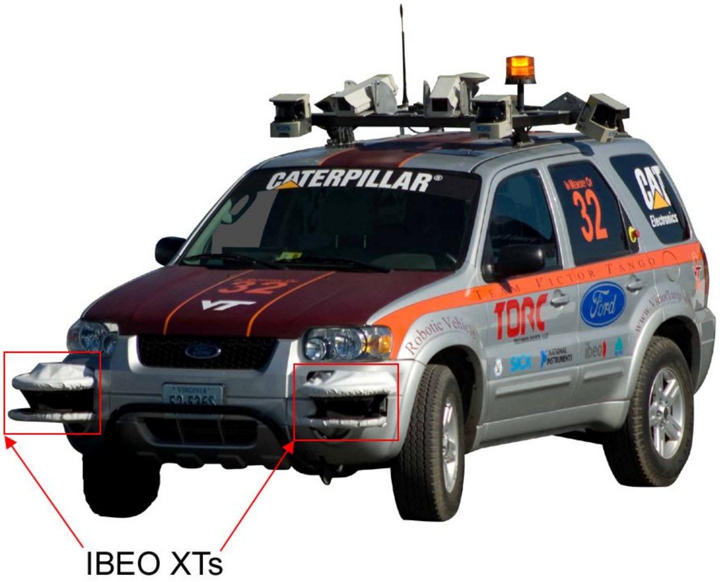


Figure 4-2: The IBEO XT units mounted on the front corners of Odin. The A0 unit is mounted at the center of the rear bumper and can not be seen in this figure.

The biggest advantage the IBEO units hold over other range-finders is their multi layer technology. The sensors have a single transmitting laser diode and four independent photo diode receivers. These diodes are stationary, and by placing them in a linear array the diodes are able to read the laser returns at multiple vertical angles. Using this configuration, each layer has a unique field of view shaped as shortened cycloids seen in Figure 4-3. The result is that the scanner has a 3.2 degree vertical field of view directly in front and at the back. This angle is reduced down to zero directly to the left and right of the sensor. While this vertical field of view is still relatively small, it does also allow for four times as many returns as a single plane scanner to be detected and processed.

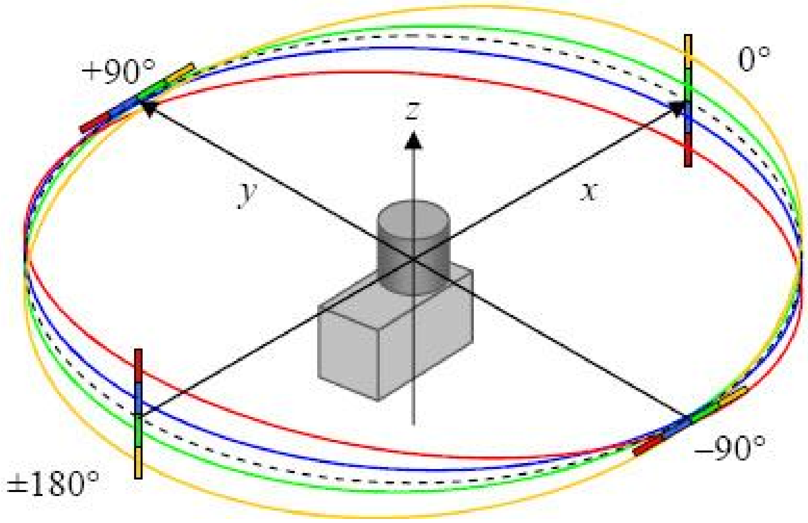


Figure 4-3: A three dimensional view of the IBEO range-finders' four vertical scan planes [IBEO 2006].

Another useful feature of the IBEO laser range-finders is their multi-target capability. When a reflected laser pulse returns to the diode the intensity of the return is converted to a voltage. This voltage must be above a certain threshold to register as a valid return in order to prevent noise from within the scanner from causing false readings. However the intensity of return for small objects in the air, such as dust or precipitation, can be large enough to register it as a valid reading. To offset this limitation the receivers are able to accept up to three echoes, or secondary returns, per pulse. The result is that the scanner can effectively see through clear materials such as glass or water without neglecting objects further away. It also results in a better definition of object corners, since a single beam could return both surfaces that define the corner.

The Fusion system and the A0 each come with their own Electronic Control Unit (ECU). The ECU is a computer which handles communications with the laser scanners, as well as performing all signal processing tasks. Scan data for one or more sensors is read in through the ARCnet interface, and can be sent out as CAN or Ethernet messages. Using signal processing algorithms, the ECU is able to classify individual returns as precipitation or dust in the air. Also, because the ground has a characteristic scan pattern, the ECU is also able determine if sets of scans are contacting the ground with moderate success. This is important to ensure that when the scanner does not classify the ground as an object the vehicle needs to avoid when approaching or leaving a hill. A limitation of this feature is that it often fails if the scan pattern for the road is broken up due to nearby objects.

The ECU has the ability to send the information it collects out in two different forms. The first is as an array of scan points that detail the relative location of each object that caused the return. The raw intensity of each return is not available to the user. The second output of the ECU is a list of discrete objects. These objects are created by clustering groups of scan points that the ECU believes to be part of the same entity. Characteristics such as position, size, and number of scan points are calculated for these objects. A second parallel process attempts to predict the movement of each object using a Kalman filter. All objects seen in the current scan are matched to the objects and predictions from the previous scan. It is possible that more than one segment can be assigned to a single object to take into account the case of a foreground object obscuring the view of a further one [IBEO 2006]. A flowchart for this object tracking algorithm can is shown in Figure 4-4.

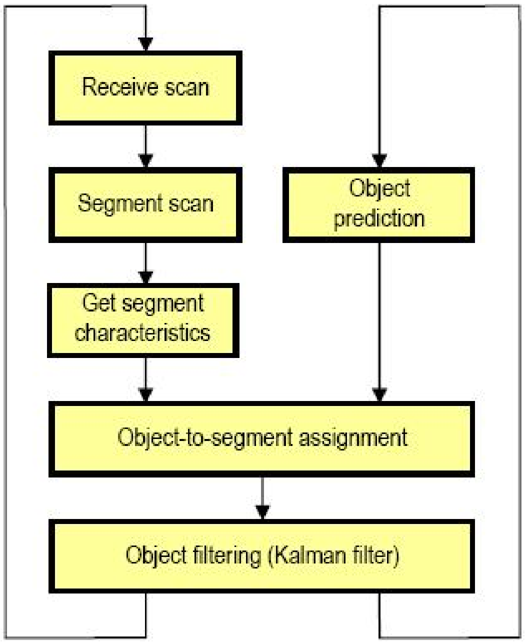

Receive scan
Segment scan
Object prediction
Get segment characteristics
Object-to-segment assignment
Object filtering (Kalman filter)

Figure 4-4: A flowchart detailing the ECU algorithm for object segmentation and tracking [IBEO 2006].

4.2 LRF Classification Processes

As previously stated, laser range-finding is only able to supply limited information to assist in vehicle detection and classification. The three main characteristics that can be determined are size, shape, and velocity. Although these attributes do not fully describe a vehicle, they do provide enough information for an initial classification. All information obtained from the IBEO is referenced to its local origin at the imaginary center of the front axle, and it is immediately translated to vehicle coordinates as previously described in Figure 2-5. The first step in detecting objects with the laser range-finders is filtering out bad or inconsequential returns. Being the first step of classification, it is important that these filters are as general as possible to prevent the exclusion of valid objects. There are two ways that an object could be considered invalid: by age or by location.

Due to cases of slight variations in the scan returns to the IBEO, at times objects can appear and disappear rapidly. Often this is caused by pieces of ground momentarily being incorrectly identified as objects. The module’s age filter handles these cases of objects flashing in and out of existence. It checks the object age coming from the sensors, and if the object has only been seen for a small number of scan iterations it is ignored. However, due to the ECU’s built-in tracking routines, objects that flicker in and out retain their old information, which includes object age. The result is that these false objects appear and disappear, but are not removed from the module’s object list by checking the ECU age alone. A secondary age filter exists within the classification software that counteracts this unintended feature. This filter functions solely on if the object was seen in previous iterations of the classification software, and so it is independent of the ECU calculated object age.

There are two more filters which remove objects based on their position in the world. The first, coarsest filter exists within the ECU’s software. A region of interest (ROI) is established that defines an area to observe. The size of this region is set based on the requirements of the competition previously detailed in section 1.3, and can be resized during operation based on the current requirements of the decision making software. Although a range of approximately 30 meters is all that is required, the size of the ROI is expanded past this range to ensure no objects of interest are missed as they enter the region. The second filter runs on any objects within this range. Since Odin is always operated on defined RNDF roads, the classification component takes in the drivable area coverage map determined by the road detection module and removes objects if they are not on or near the road. An object is only removed if all visible points are located out of the road area to account for the case of a vehicle being partially on the road. An overview of what objects would be removes by these filters is shown in Figure 4-5. Although not to scale, the red trapezoid shows the shape of the IBEO region of interest. The red boxes are vehicles that would be filtered out, while the green ones would be passed on for further analysis.

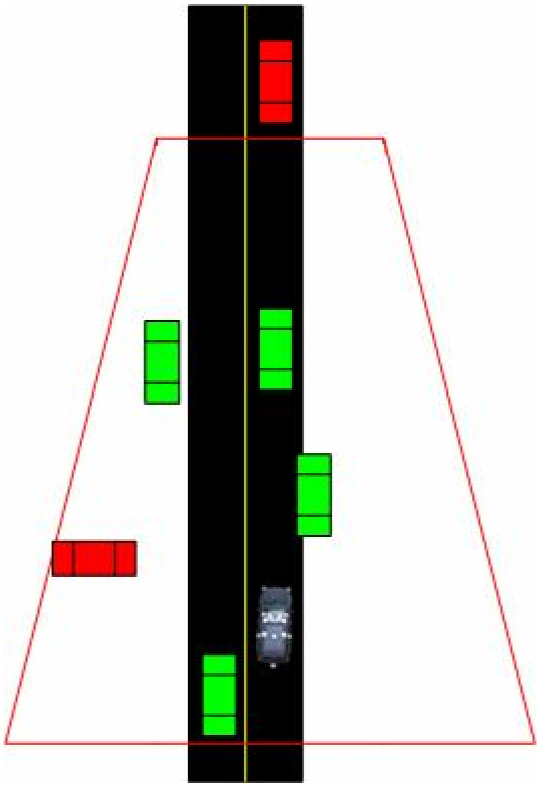


Figure 4-5: A visualization of which vehicles would be filtered out by the classification location filters.

Once an object has been detected and verified to be of interest, the range-finder software attempts to make an initial classification. These confidence values were generated through observing how often IBEO objects with various characteristics were actually vehicles during test runs. The largest indication of a vehicle is its velocity. Due to the safety constraints of the competition, the only moving objects on the Urban Challenge course were full-sized vehicles. Therefore if an object is moving and on the course, there is an extremely high chance that it is a vehicle. This reasoning could fail if small objects such as debris are seen by the range-finders. Similarly, an object that is not moving could still in actuality be a stationary vehicle. To compliment the velocity check, the size and shape of the object is also examined. Using these tools, the classification software assigns the object an initial confidence value as seen in Table 4-1. The dimension sizes refer to the size of both the x and y dimension of the object as seen by the sensors. Because the range-finders can only operate based on line of sight, a vehicle can only have one or two visible sides. Confidence values close to 10 represent a high chance of the object being a car, while values close to 0 represent a very small chance that the object is a car. Since the classification of stationary cars behind Odin is not a concern, the confidence values for objects seen by the A0 are based on much more general size characteristics than preceding objects.

Table 4-1. Confidence values for objects seen by the IBEOs.

Sensor and Situation	Confidence values based on object velocity		
	Velocity < 2 m/s	2 < Velocity < 4	Velocity > 4 m/s
A0: both dimensions < 1.3m	0	0	0
A0: one dimension > 1.3m	0	10	10
XT: both dimensions < 1.3m	0	0	0
XT: one dimension > 8m	1	3	10
XT: one dimension < 1.3m && one dimension between 1.3 and 8m	5	7	10
XT: both dimensions between 1.3 and 2.5m	6	8	10
XT: one dimension between 1.3 and 2.5m && one dimension between 2.5 and 8m	7	9	10

Odin's software architecture only requires that the position of static obstacles is defined. However vehicles must be defined by their length, width, and velocity vector. To determine how many sides of the car are seen, the size of the object in both the x and y dimensions are examined. If either dimension is close to zero, it is assumed that dimension is invalid, and only one side of the vehicle is seen. For this case the software attempts to fit the length of the contour points in the valid dimension to that of a length or width of a typical vehicle. The length of the unseen dimension is then estimated based on the size of the visible side.

If both the x and y dimensions of the contour points are not close to zero, then it is assumed that two sides of the vehicle are visible. In this case the range-finders seen an L-shaped object and must calculate the length and width of each side. To accomplish this, the three corner points of the L are found, and the distance between all three are calculated as seen in Figure 4-6. Assuming the shape is approximately that of a right triangle, the longest side is the hypotenuse, while the short and middle sides are the width and length of the vehicle, respectively.

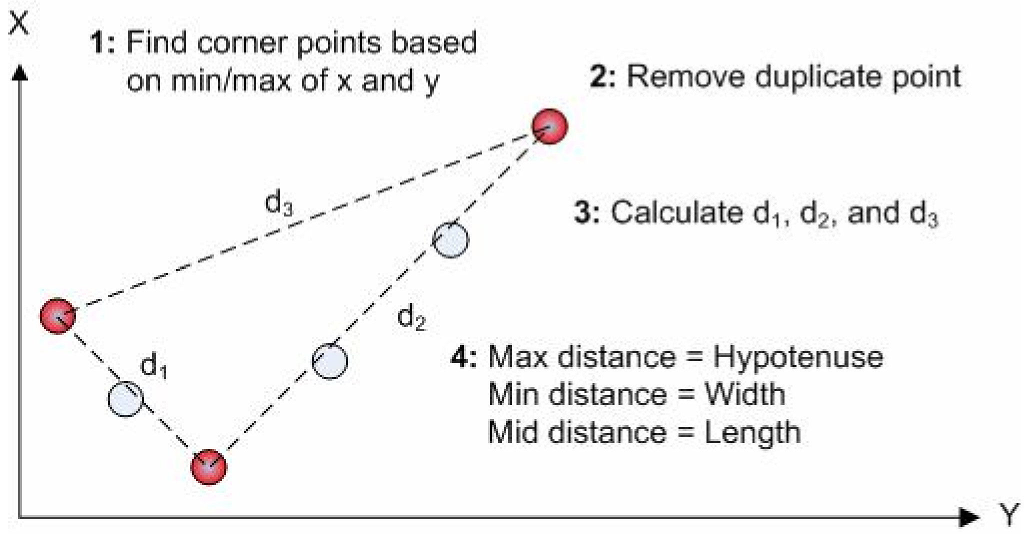

X
1: Find corner points based on min/max of x and y
2: Remove duplicate point
d3
3: Calculate d1, d2, and d3
d2
d1
4: Max distance = Hypotenuse
Min distance = Width
Mid distance = Length
Y

Figure 4-6: The process to determine the length and width of a vehicle when two sides are seen.

Although the speed of a car can be obtained from the ECU's software, the heading is not readily available. The calculation of the heading for a moving vehicle is straight forward, since the ECU can estimate the x and y components of its velocity. However, the heading of a stationary vehicle must be calculated based on its shape alone. At best these calculations result in two possible headings that are 180 degrees apart due to the ambiguity between the front and back of a vehicle. In the case of two visible sides, the heading can be estimated by finding the heading of the contour points that make up its length. Determining the heading for a vehicle with only one visible side is more complicated. Knowing whether the visible side is a length or width, and the relative position of the vehicle, an estimate of the heading can be determined as shown in Figure 4-7. The result is an estimate of the heading based on a determination of how the vehicle would have to be oriented to produce the object as perceived by the IBEOs.

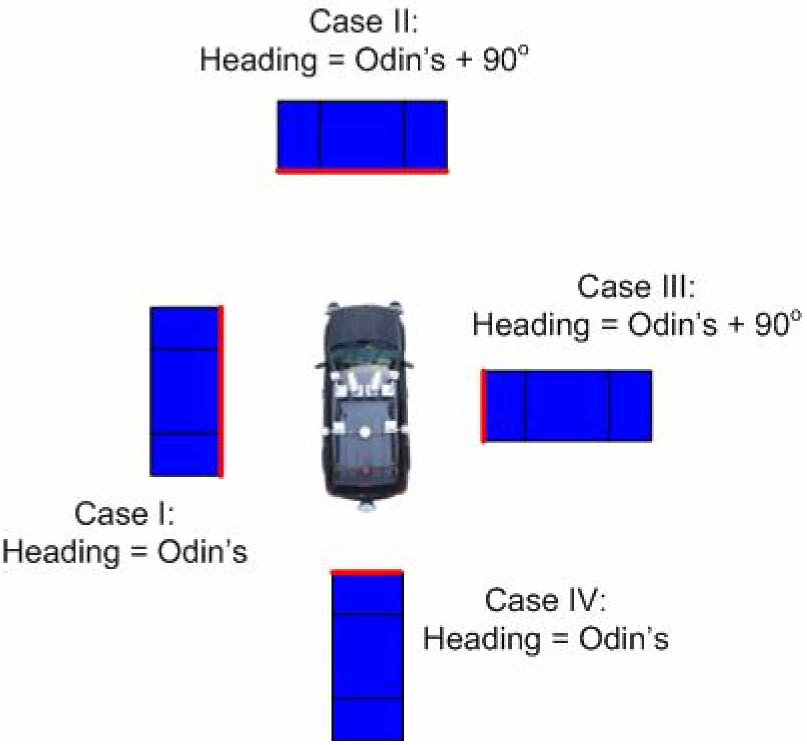


Figure 4-7: Heading estimates of a stationary vehicle based on the location and size of the visible side. The red lines represent the edge the scanner can see for each configuration.

Chapter 5
Visual Vehicle Identification

While laser range-finders can be used to accurate detect any nearby objects, they lack the information needed to accurately classify them. It has been said that a picture is worth a thousand words, and nowhere is this truer than the field of computer vision. While size and shape alone can not distinguish a car from a dumpster or other similarly sized objects, images contain this necessary information. However, this process is complicated by the wide variety of sizes, shapes, colors, and features present on road vehicles. Because of this variety, there are only a handful of characteristics present in all vehicles. These characteristics can be searched for within an image through image processing routines.

Image processing is a form of information processing that extracts information from a digital image. These steps can output either a modified image, or a set of features or characteristics from the original picture. Although image processing is performed on an entire image, most functions actually examine or modify the individual pixels within the image. In the case of digital images, these pixels represent the intensity and color of light that is detected by an array of sensors contained on a chip within the camera. By looking at the values for these pixels, or how they change over space and time, valuable information can be collected about the image and what it contains. Large images can consist of hundreds of thousands of pixels, and running image processing operations on these images can be extremely computationally intensive if operations are performed for each pixel.

There are two main types of digital camera chips currently available to choose from: CCD and CMOS. While both function on similar principles, they differ by the quality of image they are able to display. Both chips contain pixilated metal oxide semiconductors that accumulate an amount of charge proportional to the illumination intensity when the camera’s shutter is opened. A CCD chip transfers each pixel’s charge to a common output structure, and these outputs are converted to a voltage to be sent off the chip. However, in a CMOS chip the charge-to-voltage conversion takes place at each individual sensor. This minute difference results in all of the variability between the two chip types. In general, CCD chips offer the advantages of a lower noise ratio, which results in a higher image quality, as well as a higher uniformity of darks and lights and easier shutter control. On the other hand CMOS chips generally are much faster and have a number of features unavailable to CCDs, such as the ability to send regions of the images independently of the whole, or the ability to level out local overexposures [Litwiller 2001].

Another important decision to make is which color space for the processing routines to work in. A color space is a model that attempts to define as many color variations as possible through three or four base components. Color spaces are named after the components that define them. The most common, RGB, stands for the three colors of red, green, and blue that are mixed to obtain all other possible colors. This color space can be visualized of as a three dimensional Cartesian coordinate system with each color on one axis as seen in Figure 5-1. While this space is suitable for many applications, an objects RGB definition can change drastically in different lightning conditions.

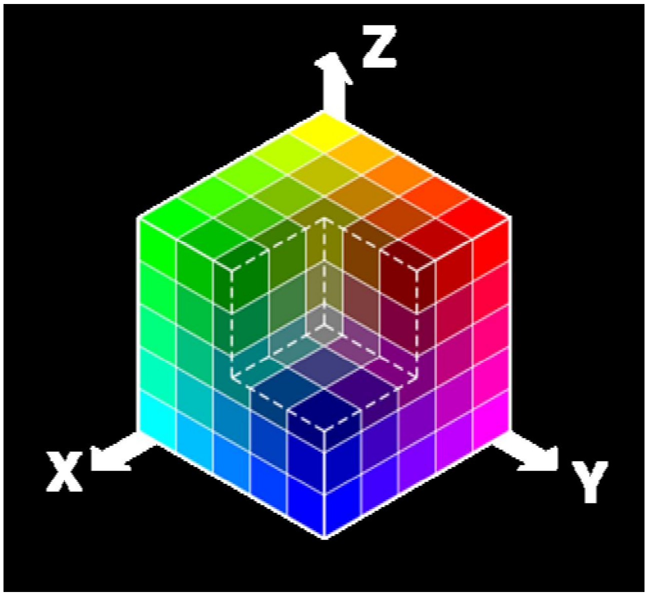


Figure 5-1: The RGB color space shown on a Cartesian coordinate frame.

For variable lighting conditions, the HSL color space is vastly superior to the RGB space. The HSL space is actually a non-linear deformation of the RGB space, and defines colors as a function of their hue, saturation, and luminosity. This color space can be depicted as a cone on a cylindrical coordinate axis as seen in Figure 6-2. The hue represents the pure color seen, and changes on the cone as the radial angle changes. In order words, the hue is the base color. The saturation is the amount of this color that is seen, and increases by moving along the radial dimension of the cone. If there is no base color the saturation is zero, and the color space is reduced to a grayscale representation. Finally, the luminosity is the amount of light seen in the color, which can be represented by moving up or down the z-axis of the cone. The higher the luminosity, the brighter the color appears. Because all of the color information is contained in the hue component, base colors are not changed in different lightning conditions, and only the saturation and luminosity vary.

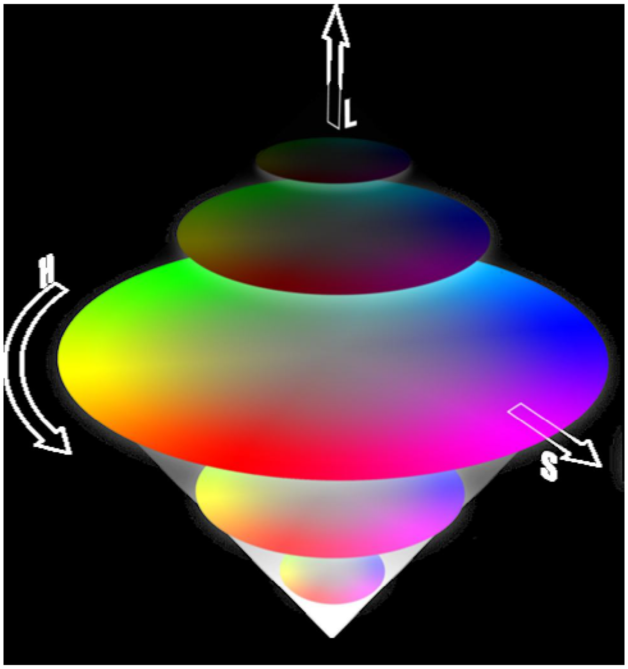


Figure 5-2: The HSL color space shown on a cylindrical coordinate frame.

This chapter continues to detail the vision processing component of the module. First, the hardware used to obtain and transmit the images is described. Next, the concept of perspective is discussed, along with how it is used to extract regions from the image. Finally, the specific characteristics that are used for vehicle detection are described, along with how they are identified within an image.

5.1 Camera Hardware

Odin is equipped with two Imaging Source color CCD cameras. CCD cameras were chosen based on their higher imaging quality than that of similar CMOS cameras. These cameras are capable of outputting 1024 by 768 pixilated images at speeds of up to 30 frames per second for grayscale images, and 15 frames per second for 24-bit RGB images. This pixel resolution is extremely large for most image processing applications. However, with a higher the resolution, an object occupies more pixels in the image. The result is that small objects can be seen at further distances with an increased image quality. The disadvantage is that with so many pixels, any image processing done on the entire image is extremely computationally intensive.

As stated in section 1-3, objects in front of Odin require the most precise classification. To supplement the IBEO classification for these objects, the two cameras are mounted on the roof rack of Odin as seen in Figure 5-3. Both cameras are angled approximately fifteen degrees below horizontal, and each is angled twenty degrees out from the center of the vehicle. Using this configuration the field of view of each camera is slightly overlapped, but the overall field of view is significantly larger than would be possible with a single camera. Since the cameras are mounted outside of the vehicle, they are each placed within an extruded aluminum enclosure designed to house security cameras. These enclosures come equipped with a system that monitors the temperature inside and activates either a heater or blower if the temperature becomes too cold or hot, respectively. Keeping the temperature moderated reduces the chances of water condensing on the lens, therefore obscuring the view of the cameras.

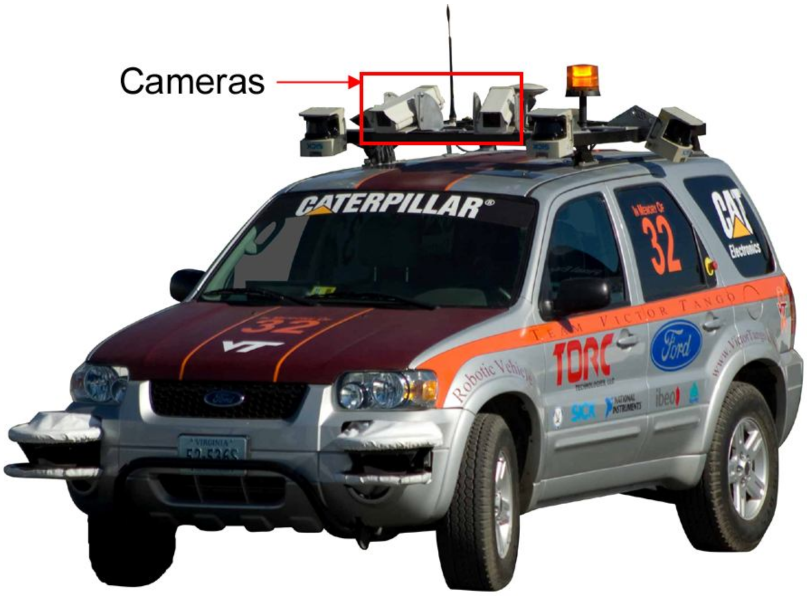


Figure 5-3: The CCD cameras mounted on the top of Odin's roof rack.

The field of view for each camera is dependent on the focal length of the lens and the size of the camera's chip. The larger the image, or the smaller the focal length, the wider the field of view obtained. The field of view of a single camera can be calculated by

$$\alpha = 2 * \tan^{-1}\left(\frac{w}{2f}\right)$$

where "w" is the width of the CCD chip and "*f*" is the focal length of the lens. From this field of view value, and knowing the relative orientation of the cameras, the overall field of view of the system can be calculated by

$$f.o.v = \alpha + 2\theta$$

where θ is the horizontal rotation of each camera. In order to scan all areas through which Odin could travel, an overall system field of view of at least 80 degrees is required, which corresponds to a 40 degree field of view requirement for each camera. Therefore, with the 5.8 mm chip inside the Imaging Source camera, the focal length of the lens must be less than or equal to approximately 8 mm. Using this information, two Pentax 6 mm lenses were selected for the system, resulting in an overall field of view of approximately 90 degrees.

Another important consideration with this setup is the blind spot caused by overlapping the cameras. The distance out that this blind spot covers can be found by

$$l = {}^{d}\!/_{2} * \tan\left(90^{o} - {}^{\alpha}\!/_{2} + \theta\right)$$

where “d” is the baseline distance between the two cameras. A summary of the physical quantities used to make these calculations can be found in Figure 5-4. Measuring the baseline distance to be 2.03 m, the point at which the fields of view begin to overlap for the Imaging Source cameras and Pentax lenses is 3.2 m past the cameras, or 1.17 m past the front bumper. Considering that the DARPA rules specify that no objects can ever be within one meter of the vehicle, the area that this blind spot covers is not of concern.

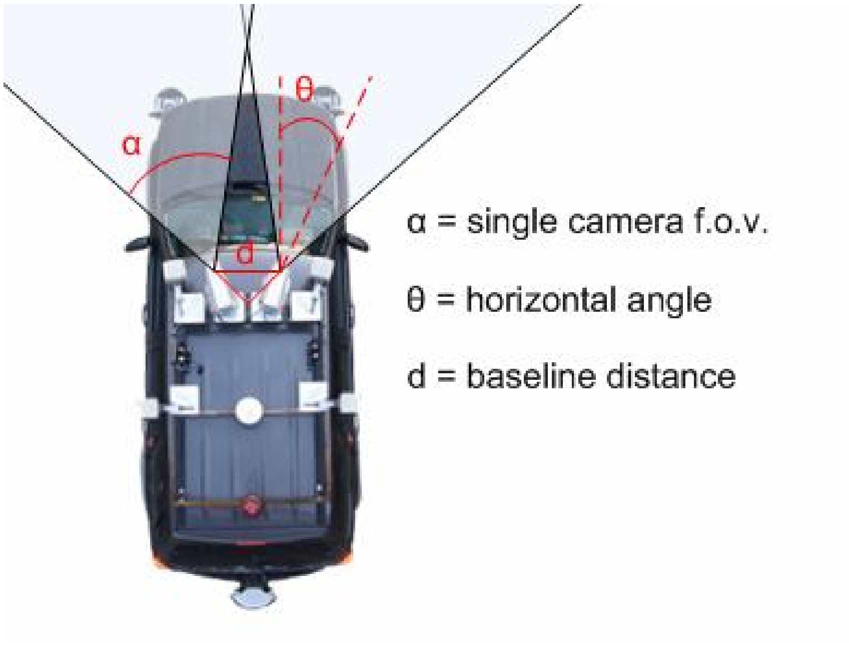


Figure 5-4: Values used to determine the vision systems field of view and blind spot.

The IEEE 1394 standard, specifically Apple Inc.’s Firewire brand, is the interface through which the onboard computers communicate with the cameras. It is a serial bus interface that supports transfer rates of up to 400 megabits per second, or 4096 bytes per packet. The image size, pixel format, and frame rate all effect the amount of information sent in each packet, and various combinations of these three values are shown in Table 5-1. Cells highlighted in light blue denote settings that the Firewire standard can support, while those in red are not achievable. The Imaging Source cameras output 1024 by 768 images, and support both the YUV color and Mono 8 raw formats. While a single YUV image can be passed at 15 frames per second with Firewire, there is not enough bandwidth to pass a second at this rate over the same bus. There is sufficient bandwidth to send two grayscale images through the Mono8 format at this size and speed, resulting in the loss of important color information.

Table 5-1. Bytes per packet required for image sizes and formats [NI 2006].

Image Type | Frame Rates	240	120	60	30	15	7.5	3.75	1.875
800x600 YUV (4:2:2)	32000	16000	8000	4000	2000	1000	500	
800x600 RGB		24000	12000	6000	3000	1500		
800x600 Y (Mono 8)	16000	8000	4000	2000	1000	500		
1024x768 YUV (4:2:2)		24576	12288	6144	3072	1536	768	384
1024x768 RGB			18432	9216	4608	2304	1152	576
1024x768 Y (Mono 8)	24576	12288	6144	3072	1536	768	384	192
800x600 Y (Mono 16)	32000	16000	8000	4000	2000	1000	500	
1024x768 Y (Mono 16)		24576	12288	6144	3072	1536	768	384

The color images sent by the cameras are created through the use of a Bayer filter. The Bayer filter is a mosaic of light sensors that are arranged on an image sensor as seen in Figure 5-5. Each filter is designed to detect red, green, or blue light, and only outputs an 8-bit grayscale representation of the magnitude of that one color. The RGB color representation for each individual pixel can be calculated knowing the intensity of light at that filter and all adjacent filters. For the YUV color format, these calculations are performed on the camera. By instead sending the raw information through the Mono 8 format and performing this conversion after the image is received, both images can be sent using the Firewire interface at 15 frames per second on the same bus while still retaining all color information. However, it is important to note that due to this local averaging of color intensities, all edges in the image are slightly blurred.

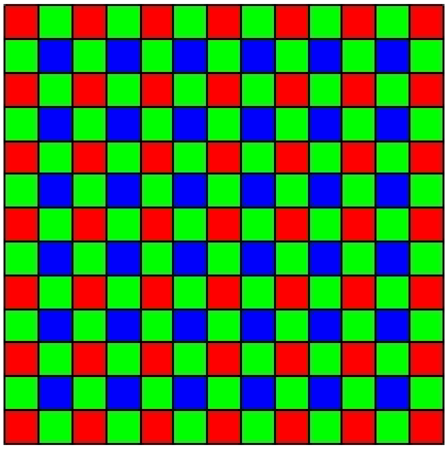


Figure 5-5: A Bayer filter mosaic. Each square represents what color the receiver is sensitive to [Allen 2007].

When using two cameras, especially when their views overlap, it is important to ensure the two images are as close to identical as possible. To accomplish this, a custom interface was created to control both cameras. The interface controls the initialization of camera settings, reading the images off the camera, and sending the images to the classification component through shared memory. While the initialization process is the same for both cameras, the interface is able to adjust light dependent settings on the fly for a single camera. For example, the color saturation and hue settings are fixed for both cameras, since their effect on the image is independent of the lighting conditions. However, the camera gain and white balance settings are set to auto-adjust to the current lighting conditions to ensure the ideal image brightness and quality. Shutter speed is the one exception to this rule since it also affects the sharpness of moving objects in the image. Because the two cameras can experience different lighting conditions depending on their orientation to the sun, it is important that these adjustments are done individually to each.

5.2 Region of Interest Extraction

The biggest limitation for most vision processing routines is the large amount of computations required. Most routines are performed on every pixel in the image, resulting in hundreds of thousands of operations for a single process. To reduce the number of these operations, the object classification module only examines small regions of the image at a time, rather than the image as a whole. These regions correspond to where the range-finding software believes there is a vehicle present.

The first step in this process is to determine where in the image the object exists. This can be accomplished through a form of perspective transformation. Perspective transformation is the process through which three dimensional objects are projected onto a two dimensional surface. For this transformation a pinhole camera model, shown in Figure 5-6, is applied to each camera. In this model the camera opening is modeled as a single point, and all light entering the enclosure must pass through this point. Normally, since the image plane is behind the focal point, images appear to be mirrored horizontally. For all calculations, a virtual image plane projected in front of the focal point is instead used, so that this mirroring effect can be ignored.

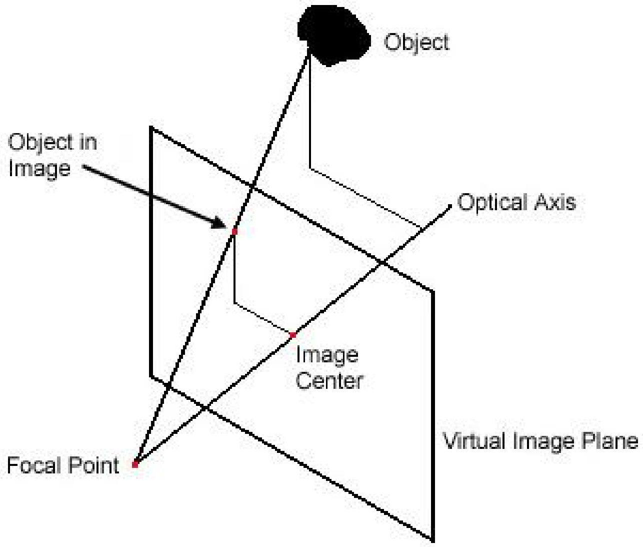


Figure 5-6: The pinhole camera model. In this model all light must pass through the single focal point.

Before the location of the object point in the image can be calculated, the real world distances to the object must be adjusted to take into account any yaw angle in the camera. By adjusting these distances first, the remaining calculations can be greatly simplified by acting as if the cameras were pointing straight ahead. This is most easily accomplished by converting the Cartesian distances into a cylindrical radius and angle, then subtracting the camera yaw from the angle. Once converting these new values back into Cartesian space, the adjusted object distances are determined by

$$x_v = \sqrt{x_o{}^2 + y_o{}^2}\cos\left(\arctan\left({}^{y_o}\!\big/_{x_o}\right) - \theta_c\right)$$

$$y_v = \sqrt{x_o{}^2 + y_o{}^2}\sin\left(\arctan\left({}^{y_o}\!\big/_{x_o}\right) - \theta_c\right)$$

where x_o and y_o are the actual distances to the object and θ_c is the yaw angle of the camera.

The image row in which the object point exists can be found first. This is accomplished by looking at the side view of the camera as seen in Figure 5-7. The angle alpha can be calculated as

$$\alpha = \arctan\left({}^{x_v}\!\big/_{-z_{cam}}\right) - \psi_{cam}$$

where ψ_{cam} is the pitch angle of the camera below horizontal. Since z-coordinates above the ground are negative in the right-handed vehicle coordinates system, the camera height must be negated. Knowing this angle, as well as some basic camera properties, the row of the image in which the object point exists can be found by

$$row = \frac{h}{2} - \frac{f * \tan(\alpha)}{s}$$

Here “h” is the overall height of the image in pixels, *f* is the focal length of the lens, and s is the physical size of a single square pixel on the CCD chip.

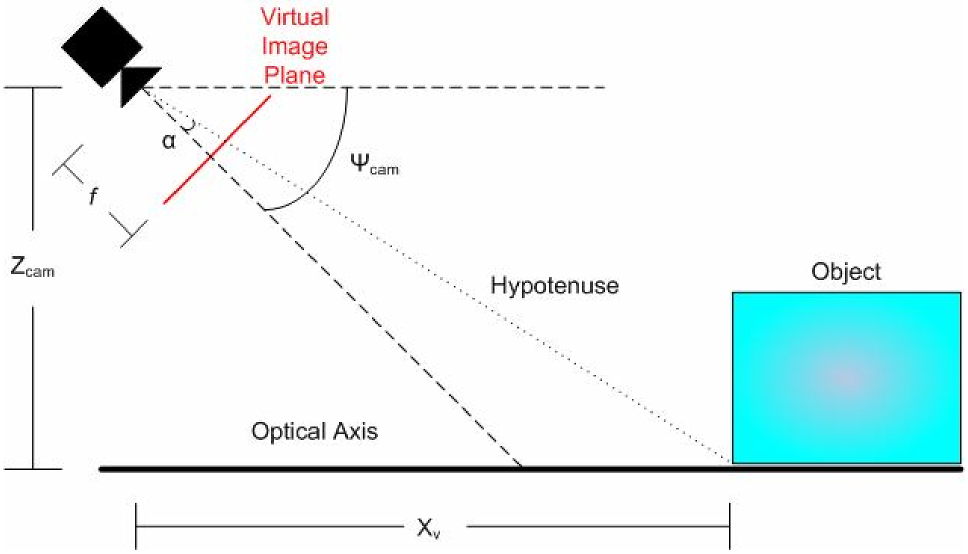


Figure 5-7: The side used to determine in what row the point exists within the image.

A number of trigonometric distances must be calculated to find the column of the image in which the object point exists. An overview of this point of view can is shown in Figure 5-8. The projection of the focal length onto the X-Y plane must be first determined using

$$f' = \frac{f}{\cos(\alpha)}$$

The projection of the hypotenuse from the focal point to the real-world object point in the X-Y plane can also be found through

$$hypot = \frac{z_{cam}}{\cos\left(\arctan {}^{x_v}\!/\!_{-z_{cam}}\right)}$$

Knowing these distances, the column of the image in which the object point exists can be found similarly to the row using

$$col = \frac{w}{2} + \frac{y_v * f'}{s * hypot}$$

where w is the width of the image in pixels.

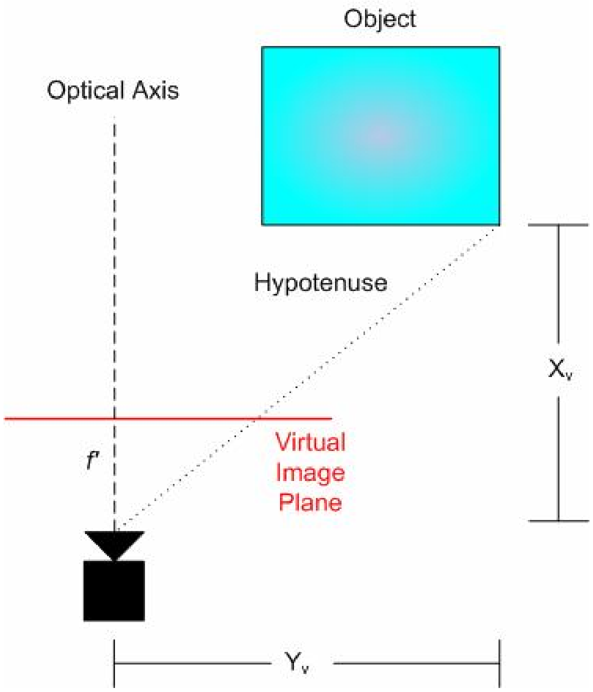


Figure 5-8: The top view of the camera and object. Note that the focal length used is actually the projection of the focal length into the X-Y plane.

A concern of this process is that it uses a flat plane assumption. In other words, all calculations assume that the point of interest is located on a flat area of ground. While this assumption is valid for most physical ground conditions, a problem arises when attempting to map object points that are not actually located on the ground. To handle this issue, the approximated height of the point is subtracted from the height of the camera in order to create a "false" ground to work with.

This process allows any point in vehicle coordinates to be mapped to an image pixel, assuming the point in question is within the camera's field of view. In order to extract a whole region instead of just a point, the vertical and horizontal size of a bounding box must be found. Since the objects are coming from the range-finders, the horizontal bounds of this box can be easily determined by finding the perspective transformation of the contour points with the minimum and maximum y-dimension values.

However, the IBEO does not give an accurate value for the z-dimension of an object. To find the vertical bounds of the region, the height of the object in the image is estimated based on the expected size of a vehicle at certain distances away from the camera. This relationship was determined experimentally. A large truck was parked in front of the cameras, and images were collected as the truck drove away. The approximate height of the truck in these images was determined at known distances. The resulting curve can be very closely approximated by an exponential function as shown in Figure 5-9. If it is assumed that the IBEO scan points hit the vehicle at approximately the same height as the sensor itself, the vertical bounding box can be placed using this exponential function and the average x-dimension of the vehicle.

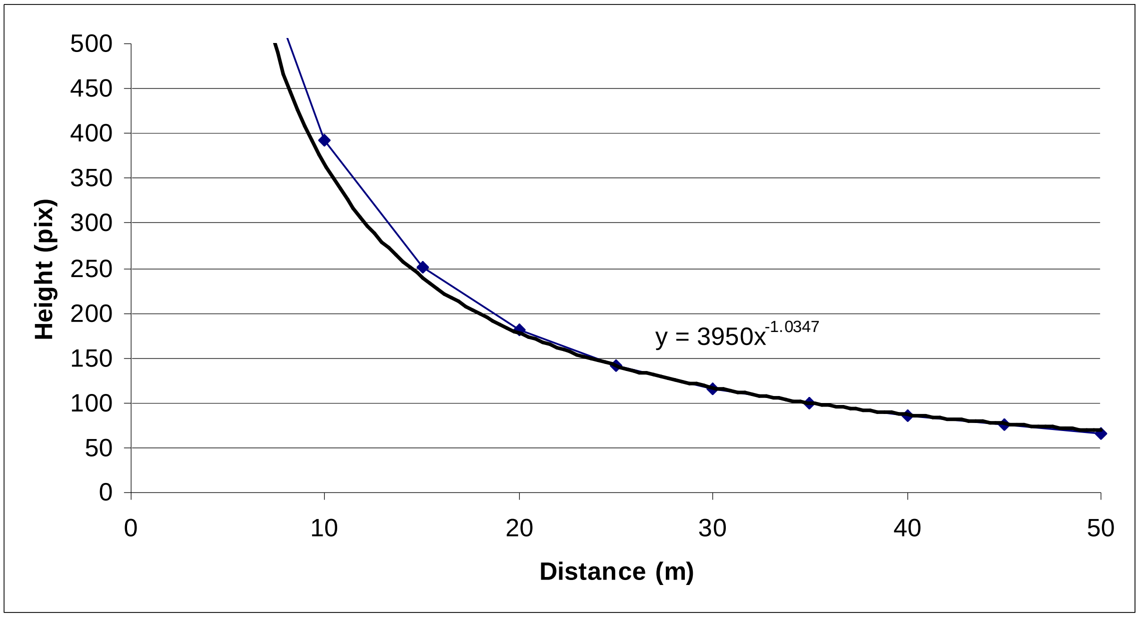


Figure 5-9: Plot showing the relationship between the height of an 8 foot truck in an image based on its distance away from the camera.

Even though the module collects both camera and range-finding inputs at the same time, it only reads the most recent scan data. Therefore the time difference between the collection of both the camera images and IBEO scan data needs to be taken into account. It is possible that the two inputs may be off by as much as the cycle time of the slowest input, or approximately 80 ms. There can also be situations where, due to the resolution of the IBEO scans, small sections at each end of the object are missed. To account for both of these possibilities, a factor of safety of 15% is added to the horizontal length of the region of interest. This factor increases the width of the region of interest to ensure that the entire object is contained in the extracted image.

5.3 Vehicle Characteristic Detection

Traffic vehicles come in a variety of shapes, sizes, and colors. This variety makes it difficult to develop a vision-based method of locating vehicles within an image. While vehicles in the Urban Challenge are required to be smaller than a certain size, they also have the possibility of not conforming to certain features common to passenger vehicles. It is therefore important to determine characteristics that are common to all vehicles, including robotic entries to the Urban Challenge. Two basic characteristics were chosen to be the basis of the visual classification routine based on their ease of identification and prevalence on all vehicles: brake lights and tires. By checking for multiple features, the chance of not finding any of them on a road vehicle decreases, resulting in a reduction of false-negative vehicle detections.

As previously detailed, precise classification is only required for vehicles in front of Odin. If the module is still uncertain of an objects identity before vision processing, the location of the object must be able to be mapped back into the image through the perspective transformation steps detailed in Section 5.2. If an object is not in the image, vision processing can not assist in classifying the object. Also, even with the relatively high resolution images obtained from the cameras, there is a certain distance past which it is impossible to distinguish vehicle characteristics. From the requirements detailed in Section 1-3, objects further than 54 meters away from Odin should not be a concern of the planning and avoidance software if traveling under 30 miles per hour. Since moving vehicles are detected successfully without the use of vision processing, the cameras are only required to classify stationary or slow moving vehicles, effectively cutting this required range in half.

The first characteristic examined was brake lights. This characteristic was an ideal choice because not only are all vehicles required to have tail lights, they are also a unique color that can be easily distinguished in an image. Additionally tail lights come in pairs, which enables for the software to examine not only an individual particle, but compare pairs of particles to known characteristics of tail lights.

The first step of the tail light detection software is processing the image. The three major steps of this process can is shown in Figure 5-10. A threshold is run on the

HSL image to extract red and deep orange hues with relatively high saturation and luminance values. A particle filter removes any small particle noise resulting from this threshold. To remove any holes in the particles, a series of dilations and erosions are run. While it is possible to simply close holes within particles, this can often result in the unintended destruction of particles that are completely contained within other hollow particles. Finally, a convex hull function is performed to smooth out any irregular particle edges. Since tail lights can come in many shapes and sizes, the exact shape of the particle is not important, so this convex hull step does not remove any useful information about the particles.

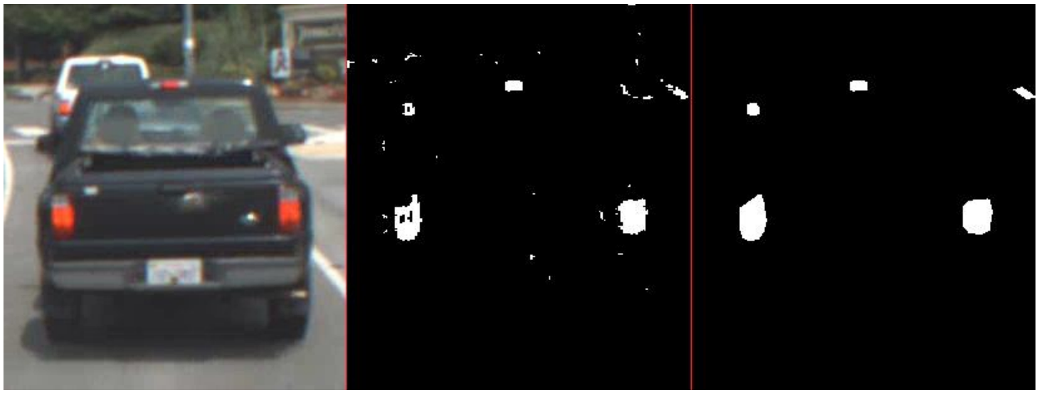
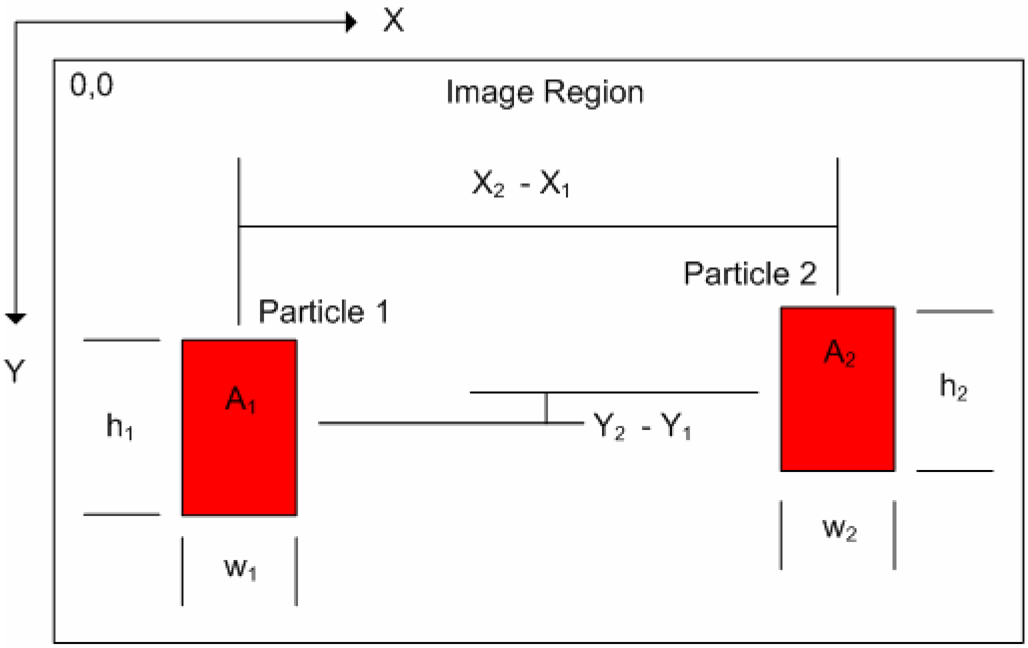


Figure 5-10: Three panels showing an example initial HSL region, the result of the threshold operation on it, and the final particle image before analysis of the tail light particles begins.

Even after the morphological operations, there often are extraneous particles that remain in the image. To further filter these results, a matching process is performed which checks pairs of particles in order to verify that together they could be a set of tail lights. An overview of the five physical quantities examined can is shown in Figure 5-11. The area of the particles must be approximately the same, the center of mass for both particles must appear in approximately the same height of the image, and the overall height of the particles must be approximately the same. Two more checks also examine how far apart the particles are in the x-direction of the image and how close their width-to-height ratios are. If all five of these checks are true, the particle pair is flagged as a tail light pair.



Level check: $Y_2\text{-}Y_1 < h_{max}$
Height check: $h_2\text{-}h_1 < 20$
Ratio check: $R_1\text{-}R_2 < 1.25$ where $R = w/h$
Size check: $A_1\text{-}A_2 < A_{diff}$ where $A_{diff} = 0.7{*}A_{max}$
Distance check: $X_2\text{-}X_1 < 22{*}W_{avg}$ where $W_{avg} = (w_1 + w_2)/2$

Figure 5-11: The five particle checks that must be true for the pair to be flagged as tail lights. All constant values in the checks were determined from a set of sample cars.

While the tail light detection software works in most situations, a red car often causes it to fail. This is due to that fact that during the threshold step the tail lights can not be separated from the rest of the vehicle. The result is a large blob near the center of the image instead of a pair of smaller particles. To account for this case, a second check takes place after the initial particle analysis. If no tail light pairs are found, the checker looks for any particles that take up a large percentage of the image and are located near the center. If this check is successful, the analysis result is marked as a special case instead of a successful pair found. The different effects that these two results have on the object confidence will be explained further in Chapter 6.

While tail lights are an ideal characteristic to examine, they can only be seen in the rear of a vehicle. To supplement the tail light checker, the software also looks for tires seen from the side of cars. Tires share a similar advantage to tail lights in that particle pairs can be examined to filter out false positives. However they are more difficult to extract from their surroundings due to the close proximity in color they share to asphalt. Instead of pulling out the tire itself, the software looks for the hubcap, which is generally much brighter than the rest of the tire.

The steps to extract the tires are similar to the tail light process, and can is shown in Figure 5-12. A threshold is applied to obtain all light areas of the image. Since there is no specific color information, the results of this threshold is often a majority of the image. However, because the dark rubber of the tire gives the light center a buffer, the center particle is segmented from the other areas. In certain situations light can bleed over the tire, connecting the center particle to other areas. To ensure this does not happen, a segmentation function is used to split particles at thin areas. After this segmentation all extremely large particles are removed. Remaining particles are dilated to account for any size loss due to the segmentation, after which all small noise particles are removed. Finally, a convex hull operation is run the remaining particles to smooth out their contours.

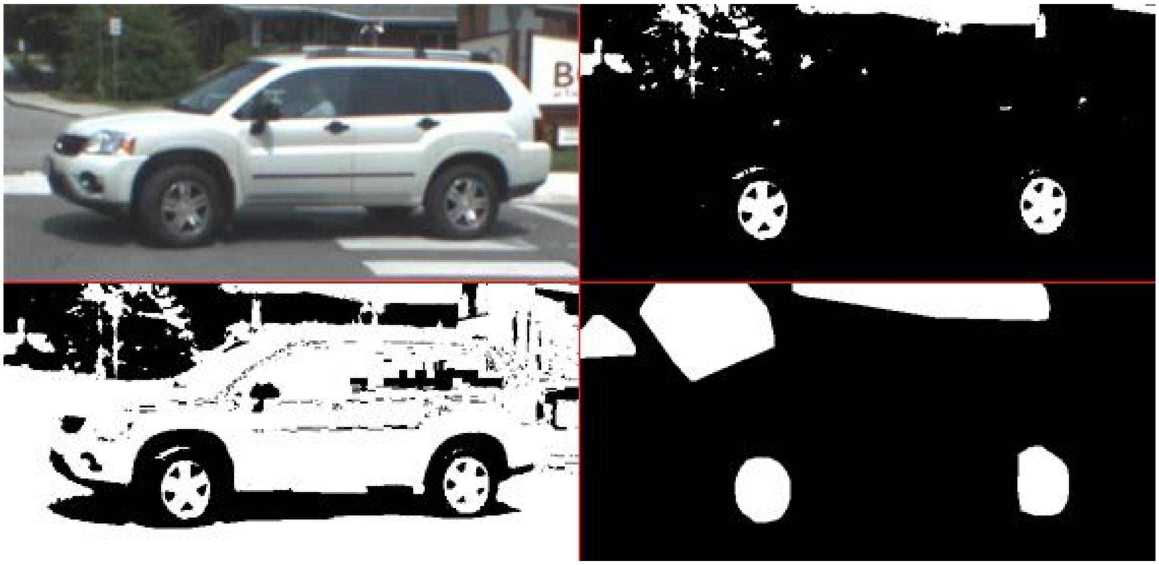


Figure 5-12: Panels showing the HSL region (upper-left), the threshold operation (lower-left), the particles after segmentation and large particle filter (upper-right), and the final particle image (lower-right).

The final image goes through two steps of particle analysis filters. The first step examines individual particles to determine if they have the correct geometry to be tires. In order to remain, the particles must have a width-to-height ratio that is approximately one. The XX and YY moments of inertia are also checked, and the ratio of these two moments must also be approximately one. Finally, the ratio

$$\frac{4 * \pi * A}{P^{2}}$$

where “A” is the area of the particle and “P” is the perimeter of the particle, must be close to one. This ratio only approaches one if the object is a perfect circle.

The second step runs the remaining particles through a matching process similar to the tail light analysis. Four characteristics are checked as seen in Figure 5-13. For a set of particles to be marked as a tire pair their areas, height, and width-height ratios must be approximately the same. The particles must also be a specified distance apart in the x-dimension of the image. In general these checks are more lenient than the tail light checks due to the individual particle filter that is performed prior to this step.

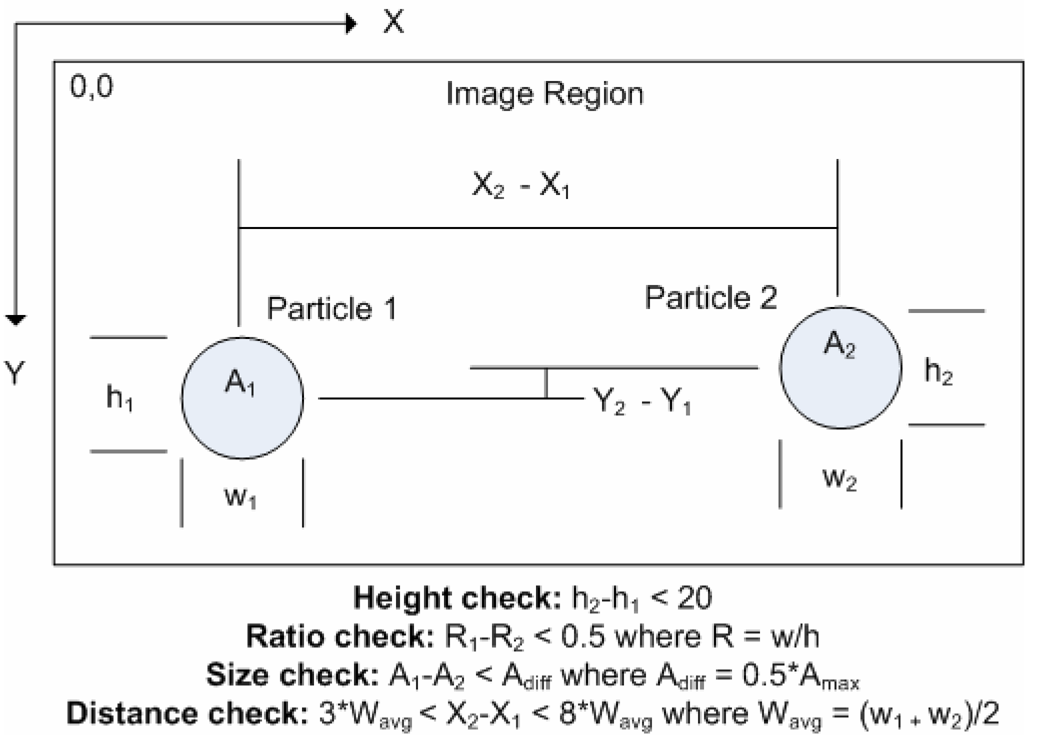

X
Y
0,0
Image Region
X_2 - X_1
Particle 1
Particle 2
A_1
A_2
h_1
h_2
w_1
w_2
Y_2 - Y_1
Height check: $h_2-h_1 < 20$
Ratio check: $R_1-R_2 < 0.5$ where $R = w/h$
Size check: $A_1-A_2 < A_{diff}$ where $A_{diff} = 0.5*A_{max}$
Distance check: $3*W_{avg} < X_2-X_1 < 8*W_{avg}$ where $W_{avg} = (w_1 + w_2)/2$

Figure 5-13: The four particle checks that must be true for the pair to be flagged as tires. Again all constant values were determined from a set of sample cars.

Other vehicle features were also examined as possible identifying characteristics. While none of these characteristics are currently present in the software, it is possible for some to be incorporated in a manner similar to tail lights and tires. This is especially true for future use of the module in traffic identification outside of the scope of the Urban Challenge. For example, license plates are required on all traffic vehicles. While all states do not require plates on the front of vehicles, they are always required on the rear. License plates are also a constant size and shape. But since they are not required on Urban Challenge vehicles, and many sensor configurations on robotic vehicles block them, they were not included as a vehicle identification characteristic. Also, the body of most vehicles is a uniform color. Even if that color is not constant between multiple vehicles, the shape of the vehicle could be determined if the color could be identified. However, many Urban Challenge vehicles are multiple colors, with numerous sponsor logos scattered along the body, making this check impossible.

While other autonomous vehicles present a larger challenge to identify based on their lack of usual road-vehicle qualities, they did present a number of their own unique characteristics. A number of these characteristics were examined as possible markers for classifying autonomous vehicles. All entries were required to have flashing lights to identify when they were in autonomous control, and it could be possible to use these amber lights a vehicle characteristic. However, there was no requirement for the size or location of these lights on the vehicle, and so it would be difficult to successfully identify them. Also, all entries were numbered, and the number was required to be placed on all sides of the vehicle. Optical Character Recognition (OCR) was examined as a possible way to not only identify autonomous vehicles but to determine which specific vehicle Odin is interacting with. These digits provided a better possibility since they were designed to stand out from the body of the vehicle. However, it was determined that it would be too difficult and computationally intensive to actively look for numbers that varied in size and font.

Chapter 6
Classification Module

The object classification module designed for Odin is a feature-level based fusion of laser range-finder and monocular vision data. It combines the locating strengths of laser range-finders with the large amount of information available in images to detect, locate, and classify vehicles which have the possibility of interacting with Odin during the Urban Challenge.

An overview of the steps to identify and classify a vehicle can is shown in Figure 6-1. Objects segmented by the IBEO ECUs are collected by the module, which initially filters out any invalid or inconsequential objects as described in section 4-2. An initial classification of static or dynamic is made based only on this IBEO data. These initial objects are passed to a classification center. Using visual information from the monocular cameras, any object in front of Odin is examined for the visual characteristics described in section 5-3. Based if any of these characteristics present, as well as the information collected from the IBEOs, a final classification of the object is made. At this point the static and dynamic objects converted to a common format, and are available to all decision making components.

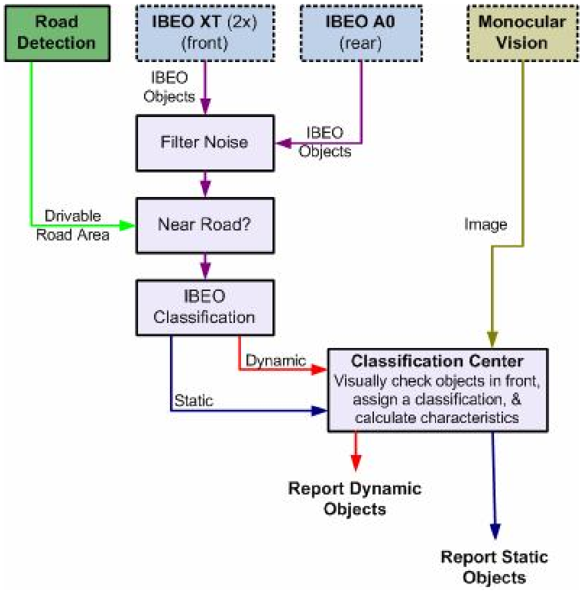


Figure 6-1: The flow of information used to classify vehicles through the object classification module

This section covers some of the details of the classification module. First, an overview of the flow of classification is presented, which describes the interplay between the classifications made by both the range-finding and vision processing routines. Afterwards, a number of module features created to improve the validity and usefulness of observed objects is discussed. Also covered here are additional requirements that the module must handle before making objects available to the entire system.

6.1 Flow of Classification

The core function of the object classification module is to correctly detect and identify nearby vehicles. Though the module itself is a variation of feature-level sensor fusion, the interaction between the range-finder and camera classifications is fairly simple. This is due to the fact that information from the IBEO is used to determine what camera objects to examine. The camera software does not currently pick out objects to examine independently of the IBEO object data, though this is a possible future extension of this research. Instead, the software modifies the classification obtained from the IBEO based on the results of the vision processing routines.

As previously described in Chapter 4, the range-finding software is able to make an initial object classification based on its size, shape, and speed. In many cases the range-finders are able to determine the identity of an object without any visual cues. An objects and size and shape are often such that it is impossible for it to be the profile of a vehicle. Therefore, objects with a very low confidence are assumed to be static objects and are not checked through vision. Similarly, vehicles can often be detected using the IBEO alone based on their speed and size, so any object with a high enough confidence coming from the IBEO is assumed to be a vehicle and is not checked.

If the cameras are used to examine an object, then its confidence is modified by the results of the vision processing routines as detailed in Table 6-1. These modifications apply to each characteristic checked by the software. Also, since the only two characteristics currently implemented can not be seen at the same time, only the maximum modification between the two is applied. For example, if the vision processing routines find a single pair of tail lights in the region, the confidence value is increased by two and the image is not checked for tires. However, if no tail lights are found and multiple possible tires are found the confidence value is increased by one. If neither is found, the confidence gets reduced by one. The special case modification takes care of situations where the initial analysis fails but a secondary check succeeds, such as with the red car check performed during the tail lights analysis.

Table 6-1. Visual processing modifications to object confidences.

Processing Result	Modification
Single Object	+2
Multiple Objects	+1
Special Case	+1
No Objects	-1
Unused	+0

Once the final confidence value has been determined, a threshold is applied to make the final distinction for the object. If the confidence value is equal to or above this threshold value, the object is considered a vehicle, and the process to determine its position, dimensions, and velocity is run. If the confidence value is below the threshold, the object is considered a static obstacle, and only its position is considered.

While the vision processing routines assist in identifying an objects identity, it never functions perfectly. Individual frames may have aberrations that cause an object to be classified incorrectly. Due to the previously described age and reverse-age filters, iterations where an object is not classified as a vehicle can results in unintended delays in informing the navigation software of the objects existence. Therefore, a simple memory filter was created to offset any temporary incorrect classifications. This filter takes the confidence of an object for the four previous iterations and averages it with the current result. If this average is significantly different than the current iterations result, the average confidence value is used to determine the objects classification instead of the current confidence value. In this way single incorrect classifications are smoothed out over time, while it is still possible for the classification of the object to change if new information is obtained by the module. Once an object is not seen and has passed through the reverse-age filter, the ID's memory is cleared to prevent future objects from being classified based on old information.

6.2 Module Features

One of the biggest challenges working with sensor data is dealing with the noise that often comes with it. This is especially true with the IBEO object data. Small variations in laser pulses' time of return can cause a disparity in an object's reported position. As these points shift, appear, and disappear, the contour points that define cluster of scan points vary as well. This phenomenon can result in objects splitting apart or merging together if these point shifts are substantial. The ECU has a built-in system to handle these appearing and disappearing objects. If an observed object is no longer seen, the ECU holds its information in memory for approximately two seconds. If during this time another object is seen in the same location, the ECU assumes it is the same object and use the remembered information.

While this feature is useful to keep track of objects that become occluded, it also makes filtering out noise more difficult. In general, the object age field is a good indication of how long the object has been seen, and a simple low pass filter can be used to ignore spurious objects that appear. However these false objects that rapidly appear and disappear remain in memory due to the built-in tracking of the ECU, and therefore the age field is retained as well. To counter this, a custom age filter was added to the classification module. This filter internally keeps track of objects seen in the previous iteration and adds them to an array. Objects remain in this array for four iterations, or approximately one-third of a second, during which time no calculations or classifications are performed on them. If the object still exists after the required time they are analyzed as usual. The result is that objects that continue to appear and disappear rapidly are always filtered out of the final object list.

A consequence of this process is that valid objects that disappear also do not reappear until they pass the custom age filter. Therefore, if an object splits or disappears for a single iteration, it did not reappear until after it has passed the entire custom age filter again. A second filter, nicknamed the reverse age filter, attempts to handle this problem. When an object is no longer seen, the reverse age filter holds the object in memory for a short time, and the classification module continues to send the object to other modules as if the object were still seen. The timeout for this filter is set slightly higher than that of the custom age filter. In this way the custom and reverse age filters overlap and the object is continually output as shown in Figure 6-2.

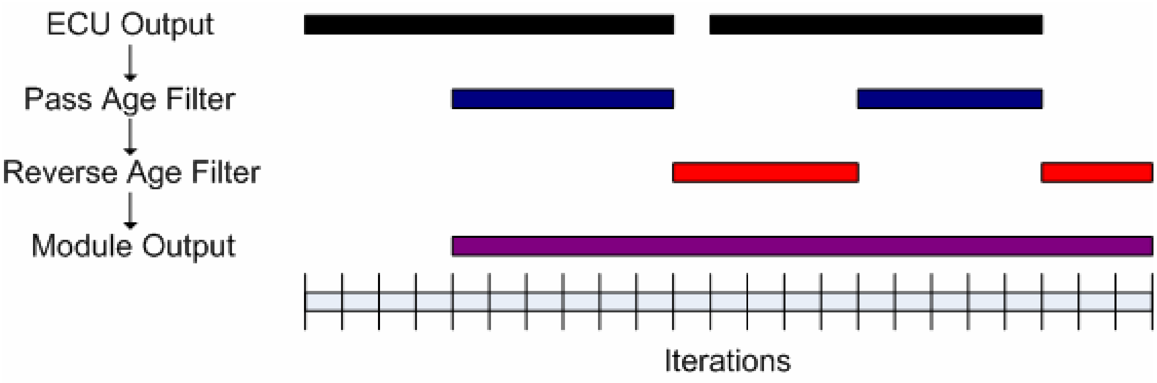


Figure 6-2: A timeline showing the custom age and reverse age filters. The black bar represent when an object is seen in the ECU, while the colored bars are shown if the object passes its respective filter.

Even if noisy scan data does not cause objects to disappear, it often causes distortions in their profiles. These distortions are usually relatively minor, but they can have a significant effect on the module's calculations of a vehicles heading and speed. Object information from previous iterations is used to smooth out these drastic changes. Knowing that the size of a car does not change, any large variation in width or length can be ignored, and the values from the previous iterations can be used instead. Likewise, the heading of a vehicle can not change instantaneously. Therefore if the heading changes drastically, especially if the vehicle is stationary, that variation can be ignored as well.

Another source of noisy scan data is the presence of dirt or imperfections on the lens, which result in scan returns in front of the scanner. These false objects are a major concern, especially with the Fusion system, since the module is telling the behavior software that an object is directly in front of the vehicle at all times. To counter this, a filter is used to remove any objects that are in a rectangular box located at the lens of the scanners. If an object's profile is contained entirely within this region it is completely ignored. However, if Odin moves close to an object, the false contour points at the scanner merge with the objects points. The result is a single object that is in front of the vehicle, but not contained entirely within the region as shown in Figure 6-3. As a solution, a secondary filter was created that only trims contour points within region. Contour points that define objects outside of the region remain, while those within the region are removed.

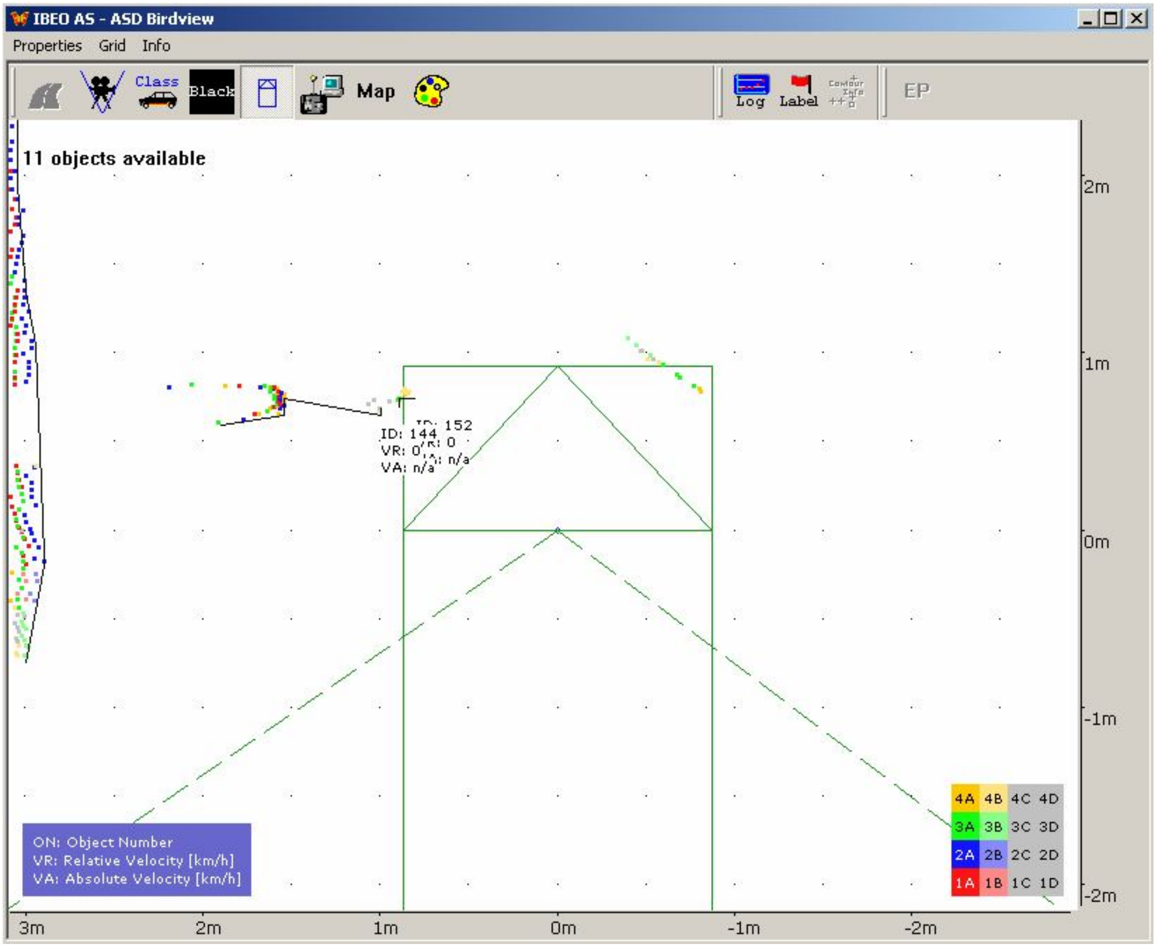


Figure 6-3: A valid object merged with noise at the IBEO lens.

Once an object has passed all of the filters, the calculations for the position, size, shape, and heading are performed in the vehicle coordinate frame as described in section 2-3. Since all of the other components work in local coordinates, all object positions and orientations must be converted to this frame. To do perform this transformation, the additional information of Odin's local position and orientation is required. The local heading of an object can be determined by adding its heading in the vehicle frame with Odin's heading. The local position can be calculated using trigonometry as shown in

Figure 6-4. The result is that both static and dynamic objects are placed in a stationary coordinate frame which is invariable to vehicle motion and GPS errors.

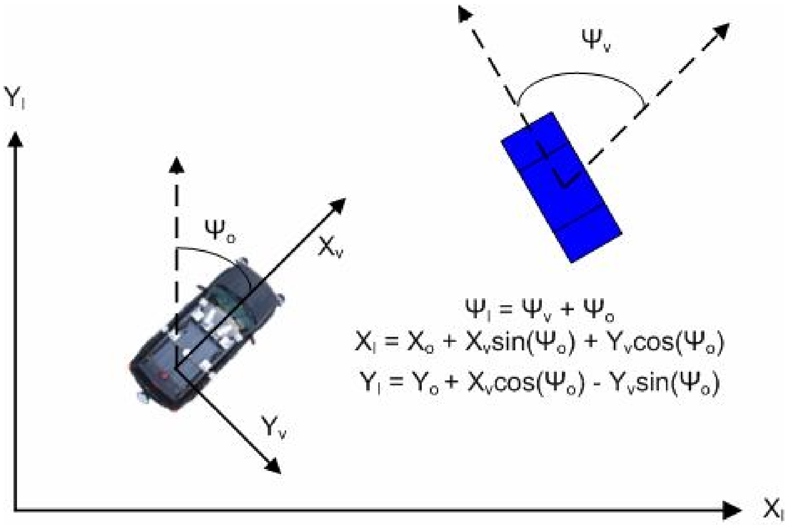

Ψ_v
Y_l
Ψ_o
X_v
$\Psi_l = \Psi_v + \Psi_o$
$X_l = X_o + X_v\sin(\Psi_o) + Y_v\cos(\Psi_o)$
$Y_l = Y_o + X_v\cos(\Psi_o) - Y_v\sin(\Psi_o)$
Y_v
X_l

Figure 6-4: The conversion of an objects position and heading from vehicle to local coordinates.

Finally, a useful feature to the decision making processes is the determination of disabled vehicles. While the higher level driving behaviors module will tell Odin to stop behind a stopped preceding vehicle, eventually Odin needs to be able to navigate around this vehicle if it remains motionless for an extended period of time. To assist in this decision making process, the classification module tracks how long since each of the observed vehicles has moved. If a vehicle has a velocity close to zero, the iteration time of the classification module is added to that vehicle ID’s previously calculated stopped time. If instead the vehicle is observed to have a velocity, the stopped time is reset to zero. It is up to the decision making modules to determine how long Odin must wait before initiating a pass around the motionless vehicle.

Chapter 7
Results, Future Work, and Conclusions

The DARPA Urban Challenge was held on November 3rd 2007, after a week long qualifying event (NQE) consisting of several grueling tests. Thirty four teams traveled to Victorville, California to compete for 3.5 million dollars in prizes. The qualification event consisted of three course, named "course A", "course B", and "course C". Course A tested the vehicle's ability to initiate a left turn across moving traffic and merge with moving traffic, and was a good indication of the overall safety of the vehicle. Course B was a long navigation challenge designed to determine a vehicle's ability to successfully navigate a course and return. Finally, course C tested how well a vehicle could navigate four-way intersections and re-plan a path if a road was blocked off.

At the end of the week, only eleven vehicles were invited to participate in the final event due to safety concerns with the other vehicles. Odin was included in this group, and performed so well during qualification that the team earned the second pole position. In the Urban Challenge Event (UCE) Odin, seen in Figure 7-1, was the first to enter the course due to a GPS failure on Carnegie Mellon's robot. The event itself was split into three main missions, each of which was split into multiple sub-missions. After each main mission the vehicle would return to the chute to receive the next MDF.

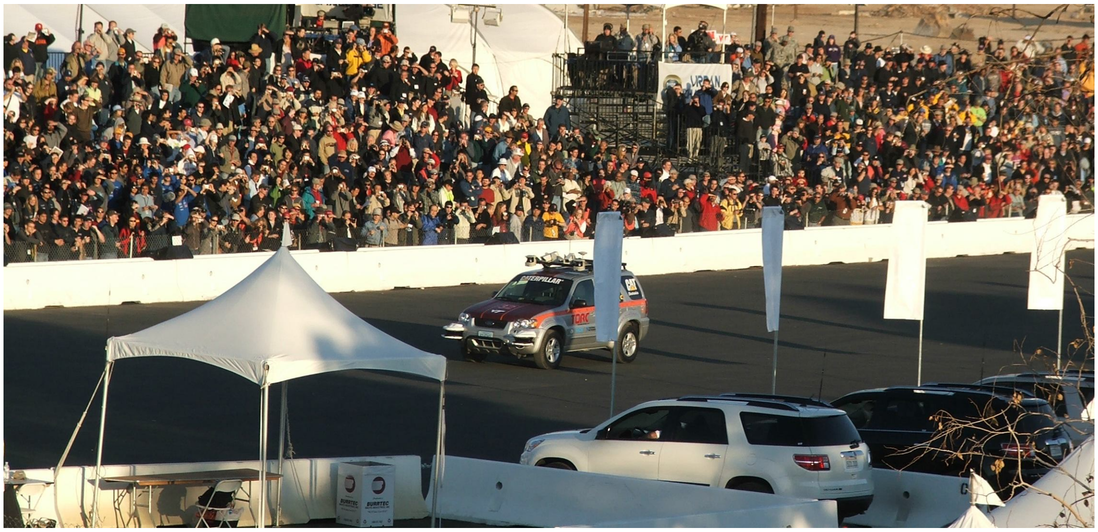


Figure 7-1: Odin leaving the start chutes first the morning of the Urban Challenge Event.

Throughout the day Odin performed well, navigating successfully through winding roads, intersections, open parking lots, and even unmarked dirt roads. Odin was the first vehicle to complete both missions one and two, with Stanford and Carnegie Mellon never more than ten minutes behind. However, by the end of the event both teams had made up time on Odin. Stanford crosses the finish line first, followed by Carnegie Mellon minutes later. Odin finished approximately five minutes after Carnegie Mellon, the third of six vehicles to complete the course.

At the closing ceremonies the next day, DARPA announced that Odin had won third place behind Stanford and Carnegie Mellon, earning $500,000 as shown in Figure 7-2. Though DARPA said that none of the top three teams had traffic infractions that modified their overall time, there were a number of incidents involving all three vehicles. Throughout the day, numerous DARPA officials told the team that they believe Odin was driving the safest, performing moves such as driving down the aisles of the parking lot rather than cutting across spots.

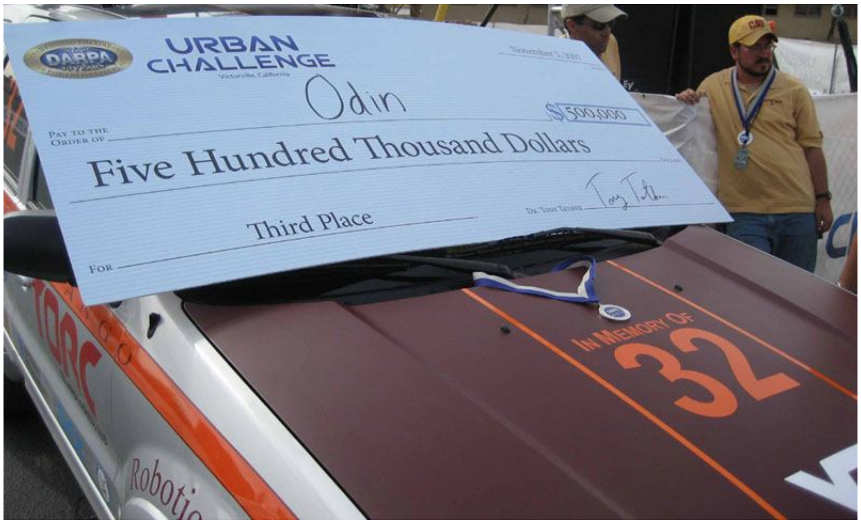


Figure 7-2: Odin at the closing ceremonies with the third place check.

This rest of this chapter covers the results of testing the classification module, both before and during the Urban Challenge. First will be a discussion on what parts of the module were used for the Urban Challenge Event and why this decision was made. Also covered here is how the rest of the module would have improved the vehicle’s performance due to the improved classification capabilities. Next, some suggestions for future improvements to the system will be presented. Finally, some conclusions on the results of the module as a whole will be discussed.

7.1 System Test Results

There were months of testing prior to the Urban Challenge, but the real tests of the classification module were during the qualification and final Urban Challenge events. During the team's testing sessions factors were controlled, and often only specific portions of the code were being monitored closely. During the actual events, the software had to run flawlessly without anyone monitoring or debugging it.

The timeline between the start of work on the project and the start of the competition was extremely short. Shortly before the qualification event the module was complete, but only the range-finding portions had been sufficiently tested to ensure one hundred percent reliability. Also while false positive results for vehicle detection would not be ideal, the chance of a false negative detection could be disastrous. Without months of testing, it could not be confirmed that a false negative detection would not occur. For this reason, only the range-finding software was run during the Urban Challenge, even though the vision portions were complete as well.

During the qualification and final event, the module recorded a one hundred percent success rate at detection vehicle within the field of view and distance requirements detailed in Section 1-3. It was also able to run at approximately 10 Hz, allowing ample time for the decision making software to handle static and slow moving objects. However, static objects were often misclassified as vehicles. This was a result of the vision module not running, and in most cases it did not affect the overall behavior of the vehicle. Odin was extremely successful in navigating the qualification and final event runs. There were a few problems did occur with the module during some runs.

On the first run on the NQE "area A" course, after successfully completing three complete loops of merging with traffic, the vehicle got stuck attempting to make a left hand turn across traffic. Once DARPA officials allowed the team to approach the vehicle it was determined that the retro reflective tape used to make the lane boundaries were reflecting the IBEO scan returns, creating what looked like the profile of a vehicle in the approaching lane. The result was that there was always a vehicle approaching, and Odin was never clear to turn. While it would not have removed the object, if the vision module was running it should have detected that the object had no vehicle characteristics, and therefore was not an approaching vehicle. A fix was implemented on the behavior side of the software to take care of this problem.

On the second run of NQE course A run, Odin appeared to be cutting off more vehicles than normal during the merge with moving traffic. After the completion of the run it was discovered that there currently exists a software bug within the IBEO ROI software. The ROI software allows the user to specify a trapezoidal region within which objects are detected. However, there is a bug where only the thinnest rectangle of the trapezoid is actually used as the region. A visual representation of this region bug is shown in Figure 7-3. As a result, all IBEO regions had to be modified to be rectangular to ensure the correct region was being examined

.

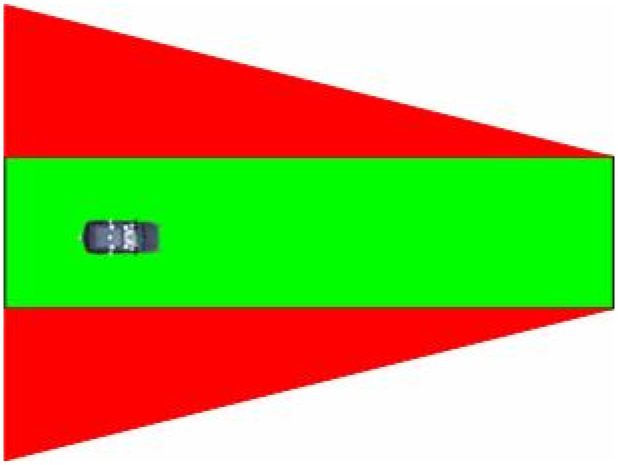


Figure 7-3: A visual representation of the IBEO ROI bug. Because of this bug only the green portion of the region would return objects, instead of the entire area.

During the final run, Odin was said to have temporarily stopped during the off-road portion of the course. This area consisted of a winding, motor-graded dirt road that sloped down at varying angles. At each side of this road were dirt burms ranging from a few inches to two feet tall. The reason for Odin's delay was that a section of these burms had an L-shaped profile, and was being classified as a vehicle in Odin's lane. This problem alone would not have resulted in Odin being stuck forever, as the burms were far enough off the road that Odin would have been able to pass. However, a simultaneous small drift in the global position of the vehicle placed Odin further to the side of the road than it actually was, resulting in the object being placed in the center of the road. Eventually the position drift disappeared, and Odin went around the fake vehicle as expected. While the vision module would have resulted in this object being correctly classified as static, it alone would not have prevented this delay.

A known problem with the IBEO sensors was that occasionally the internal software would freeze, which results in no scan or object data getting to the software module. This problem occurred during the 3rd and final mission of the Urban Challenge, when Odin came to a stop before a four-way intersection. To remedy this situation, a health monitoring routine had been built-into the module code. If the module fails to receive information from the IBEOs for a full five seconds, the vehicle cycles the power to the ECUs and pauses 90 seconds until they are running again.

Finally, although the vision processing module was not sufficiently tested for the final event, it had been tested in many situations and lighting conditions. A sample set of these tests can be found in Appendix B. These tests all assume that the vehicle in question is static, even in cases when the actual vehicle is moving. This is due to the fact that a moving vehicle is not be examined by the vision processing software as described in Section 6-1.

7.2 Future Improvements

There are a number of modifications or changes that can be made to the classification module. The most basic change would be the addition of more vehicle characteristics that the vision processing routines could look for. License plates and a uniform vehicle body would be the main characteristics available to on traffic vehicles, but others could be added as well with little to no effect on the rest of the system. The only concern would be maintaining a cycle time of approximately 10 Hz with more vision processing in the software.

Another possible addition to the current software would be to create a method to handle occluded characteristics. Since the current characteristics are based on particle pairs, a missing particle would cause the software to fail. The particle could fail to be detected due to an incorrect region extraction, an occluding object, or a temporary unfavorable lighting condition. A secondary check could be created after the main check for characteristics that comes in a pair. This check could function similarly to the red car check for tail lights, and would attempt to determine if single particles could be the characteristic in question.

The first major recommended modification to the module would be the ability to completely remove objects through the use of vision. At times, such as when the lane markings caused Odin to stop during its first NQE course A run, objects are detected that do not actually exist in the world. This is often a result of ground-based features, such as hills or road markings, not being filtered out by the road detection routine built into the IBEO software. A visual method could be created to determine if an object of significant height does exist for each static object detected by the range-finders. Any software created for this purpose would have to be tested extensively to ensure that no valid objects are incorrectly removed. A similar addition would be the ability for vision processing to detect its own objects. A method such as stereo vision or optical flow could be used to create a set of visually detected objects. The major concern for this addition would be that a method to compare inconsistencies between the range-finding and visually detected objects would also have to be created.

Another option to improve the accuracy of object detection would be to create a method to create objects from raw scan data. While the IBEO objects are sufficient in most situations, many times the tracking software fails when one objects moves close to another. In this situation the object ID’s merge and all history of each individual object is lost. A custom object creation routine using point clustering or another statistical method could handle these cases. This would also allow more than the IBEO maximum of 64 objects per iteration to be detected and tracked. However since the IBEO is a closed system, certain data, such as the strength of each laser return, is unavailable to the user, making this task more difficult.

Finally, a possible extension of this software could be to other objects of interest for autonomous road vehicles. If used on commercial streets certain entities such as bicycles and pedestrians would also need to be handled differently than most objects. While a similar version of the range-finding software could still be used for these entities, a different method of visual verification would need to be created.

7.3 Conclusions

In conclusion, a novel method of object detection and classification was created for use in the DARPA Urban Challenge. This method fuses the accurate object detection and location of laser range-finders with the visual classification capabilities available through vision processing to detect traffic vehicles from multiple observation angles. The range-finding portion of the module alone enabled Odin to successfully complete the Urban Challenge, earning team *V*ictor *T*ango third place and $500,000. The few faults the module encountered during the competition could have been rectified by including the vision processing software. However, due to the short timeline for the event, there was not time for sufficient testing of the entire module to ensure one hundred percent a success rate for correctly classifying vehicles while using vision.

The software is robust enough to detect any objects of interest around the platform. While some false positive object detections still exist, some small additions and modifications to the module could even further improve its accuracy. The software is also modular enough that a different method of object detection or object classification could be created and inserted in place of the current methods. Work on this software module could easily be continued to allow it to be used on actual commercial roads in the near future.

References

Allen, Chuck, "How Digital Cameras Work," http://www.csusm.edu/iits/trc/training/lessons/digitalCameras/digicams101.htm (San Marcos, CA: California State University San Marcos, May 2007).

Castellanos, J.A., Niera, J., Strauss, O., Tardos, J.D., "Detecting High Level Features for Mobile Robot Localization," in *Proceedings of the IEEE/SICE/RSJ International Conference on Multisensor Fusion and Integration for Intelligent Systems* (December 1996), pp. 611-618.

Defense Advanced Research Projects Agency, "Urban Challenge homepage," *http://www.darpa.mil/grandchallenge* (Washington DC: *DARPA*, October 2007).

Futoshi, K., Tanabe, Y., Fukuda, T., and Kojima, F., "Acquisition of Sensor Fusion Rule Based on Environmental Condition in Sensor Fusion System," in *Proceedings of the IFSA World Congress and 20th NAFIPS International Conference*, vol. 4 (July 2001), pp. 2096-2101.

Graefe, V., "Vision for Intelligent Road Vehicles," in *Intelligent Vehicle Symposium* (July 1993), pp. 135-140.

Hall, D.L., and Llinas, J, "An Introduction to Multisensor Data Fusion," in *Proceedings of the IEEE*, vol. 85, no. 1 (January 1997), pp. 6-23.

IBEO, "ALASCA User Manual," manual (Hamburg GER: Ibeo Automobile Sensor GmbH, 2006).

Jasiobedzki, P., "Fusing and Guiding Range Measurements with Colour Video Images," in *Proceedings of the International Conference on Recent Advances in 3-D Digital Imaging and Modeling* (May 1997), pp. 339-344.

Kato, T., Ninomiya, Y., and Masaki, I., "Preceding Vehicle Recognition Based on Learning from Sample Images," in *IEEE Transactions on*

Intelligent Transportation Systems, vol. 3, no. 4 (December 2002), pp. 252-260.

Koller, D., Weber, J., and Malik, J., “Robust Multiple Car Tracking with Occlusion Reasoning,” in *Proceedings of the EECV* (May 1994), pp. 189-196.

Litwiller, D., “CCD vs CMOS: Facts and Fiction,” reprint from *Photonics Spectra*, vol. 35, no. 1 (Laurin Publishing Co. Inc., 2001).

National Instruments, “Acquiring from Firewire Cameras with National Instruments IMAQdx and Legacy NI_IMAQ for IEEE 1394,” http://zone.ni.com/devzone/cda/tut/p/id/2977 (Austin, TX: *NI Developer Zone*, February 2006).

Naughton, Russell, “Remote Pioleted Aerial Vehicles: An Anthology,” *http://www.ctie.monash.edu/hargrave/rpav_home.html* (Victoria AUS: *Hargrave Military Academy*, August 2007).

Sotelo, M.A., Nuevo, J., Bergasa, L.M., Ocana, M., Parra, I., and Fernandez, D., “Road Vehicle Recognition in Monocular Images,” in *Proceedings of the IEEE International Symposium on Industrial Electronics*, vol. 4 (June 2005), pp. 1471-1476.

Talukder, A., Goldberg, S., Matthies, L., and Ansar, A., “Real-time Detection of Moving Objects in a Dynamic Scene from Robotic Vehicles,” in *Proceedings of the International Conference on Intelligent Robots and Systems*, vol. 2 (Oct 2003), pp. 1308-1313.

Appendix A – ASD Vehicle Scan Examples

This appendix illustrates some typical vehicles returned from the IBEO laser range-finders. The object identification number, as well as its relative velocity, is displayed next to each observed object. Below each screenshot is a short description of the actual state of the vehicle.

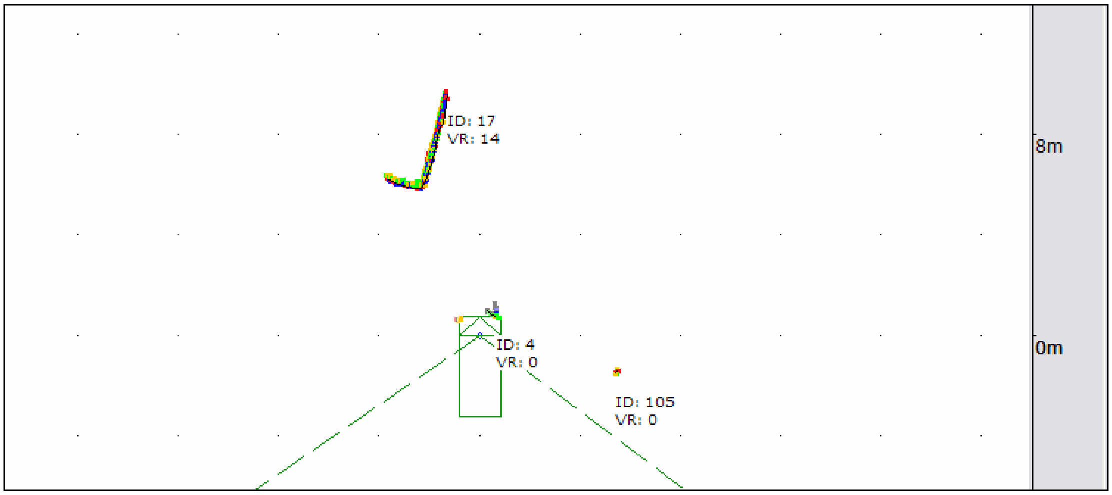


Figure A-1. Odin is stationary while Vehicle 17 passes on the left. Both sides of the object are seen.

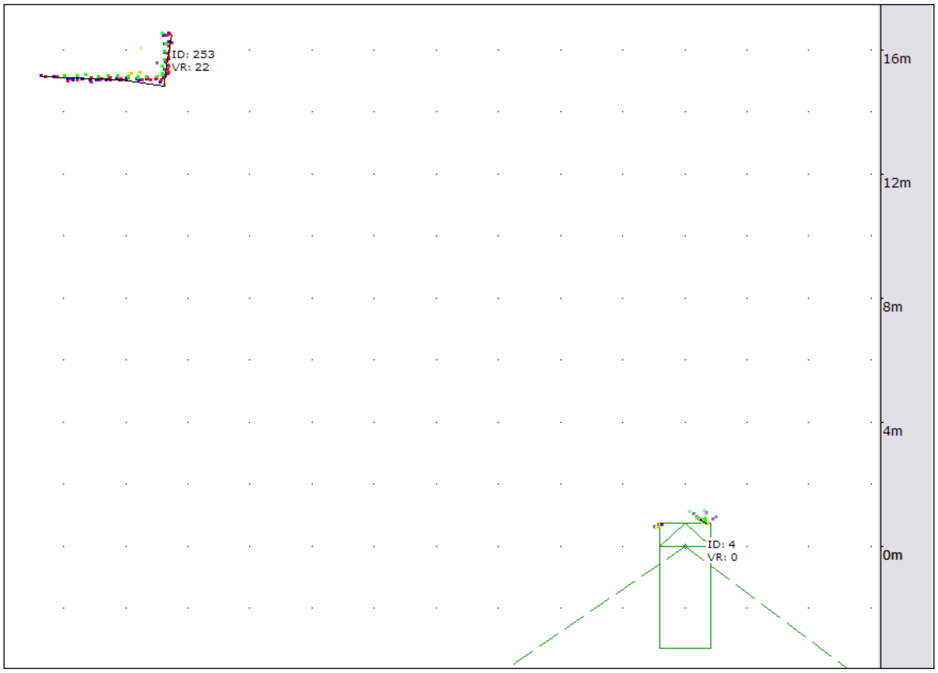


Figure A-2. Odin is stationary while Vehicle 253 travels perpendicular to Odin's lane of travel. Both sides of the vehicle are still seen, although not as sharply as in the previous example.

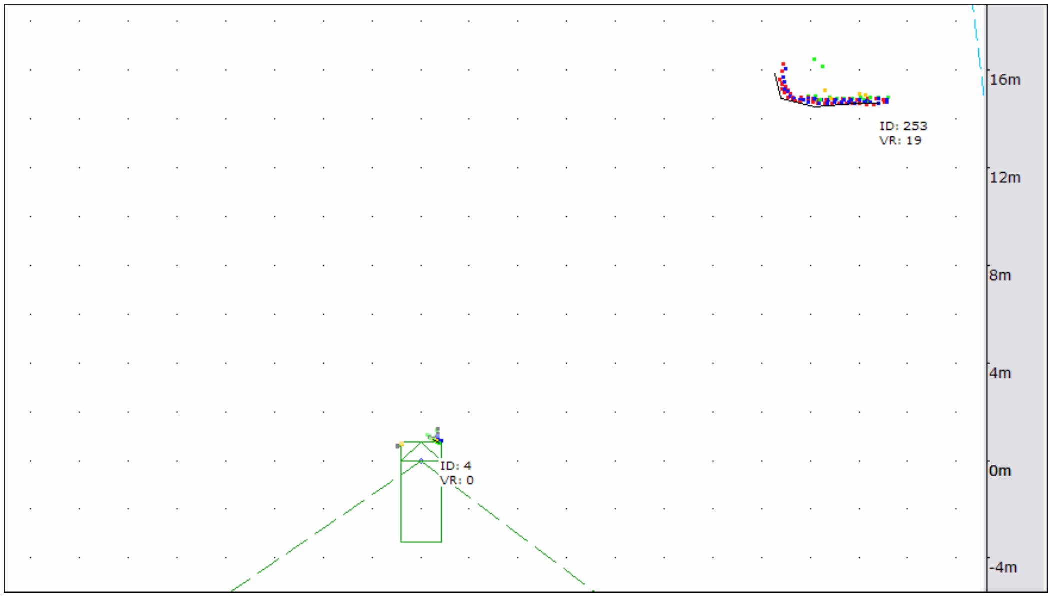


Figure A-3. Odin is stationary while Vehicle 253 travels perpendicular to Odin's lane of travel. Some missing scan returns on the front of the vehicle slightly distort its profile.

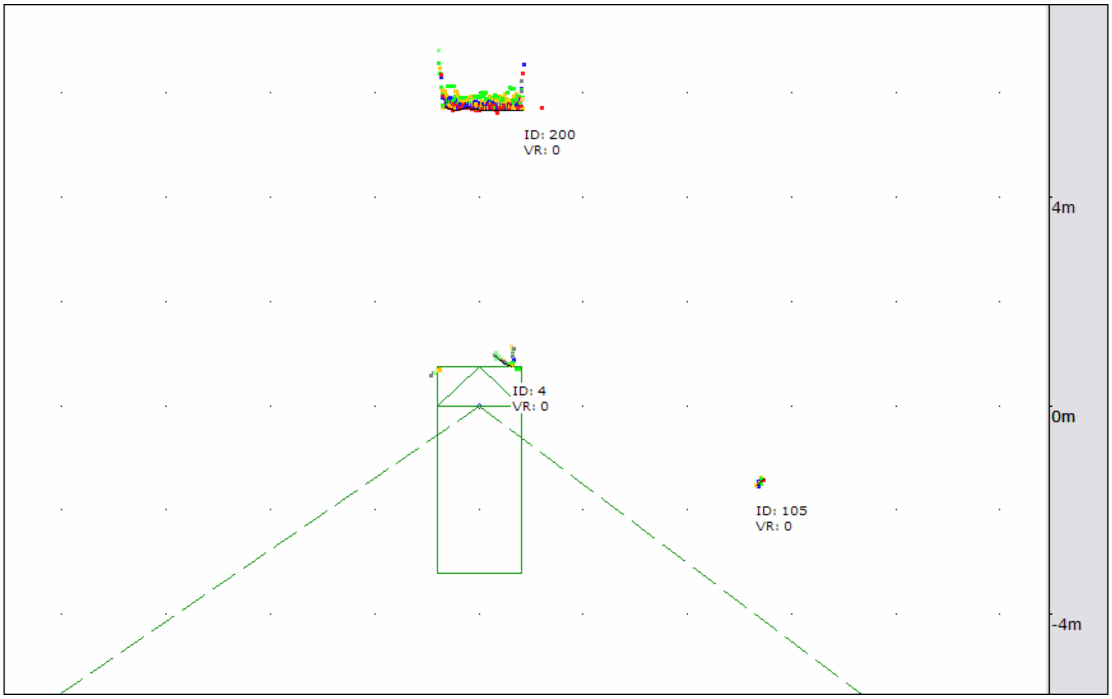


Figure A-4. Odin is stationary while Vehicle 200 sits in its lane of travel. Only the back of the vehicle can be seen.

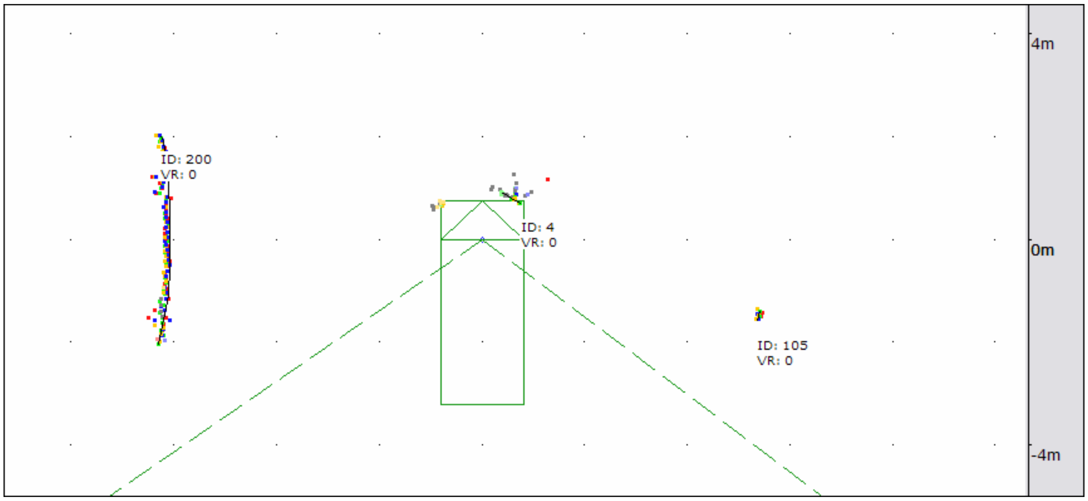


Figure A-5. Odin is stationary while Vehicle 200 sits to its side. Only the side of the vehicle can be seen.

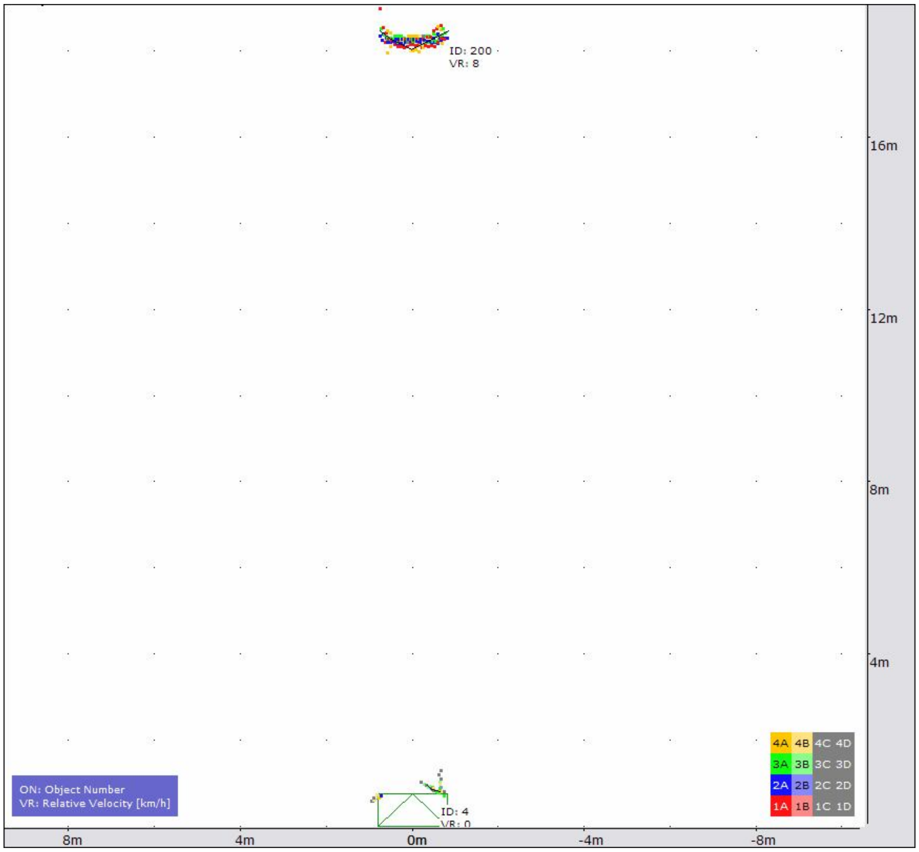


Figure A-6. Odin is stationary while Vehicle 200 sits in its lane 20 meters away. Only the back side of the vehicle is seen, but the profile points are in the shape of a slight “V”.

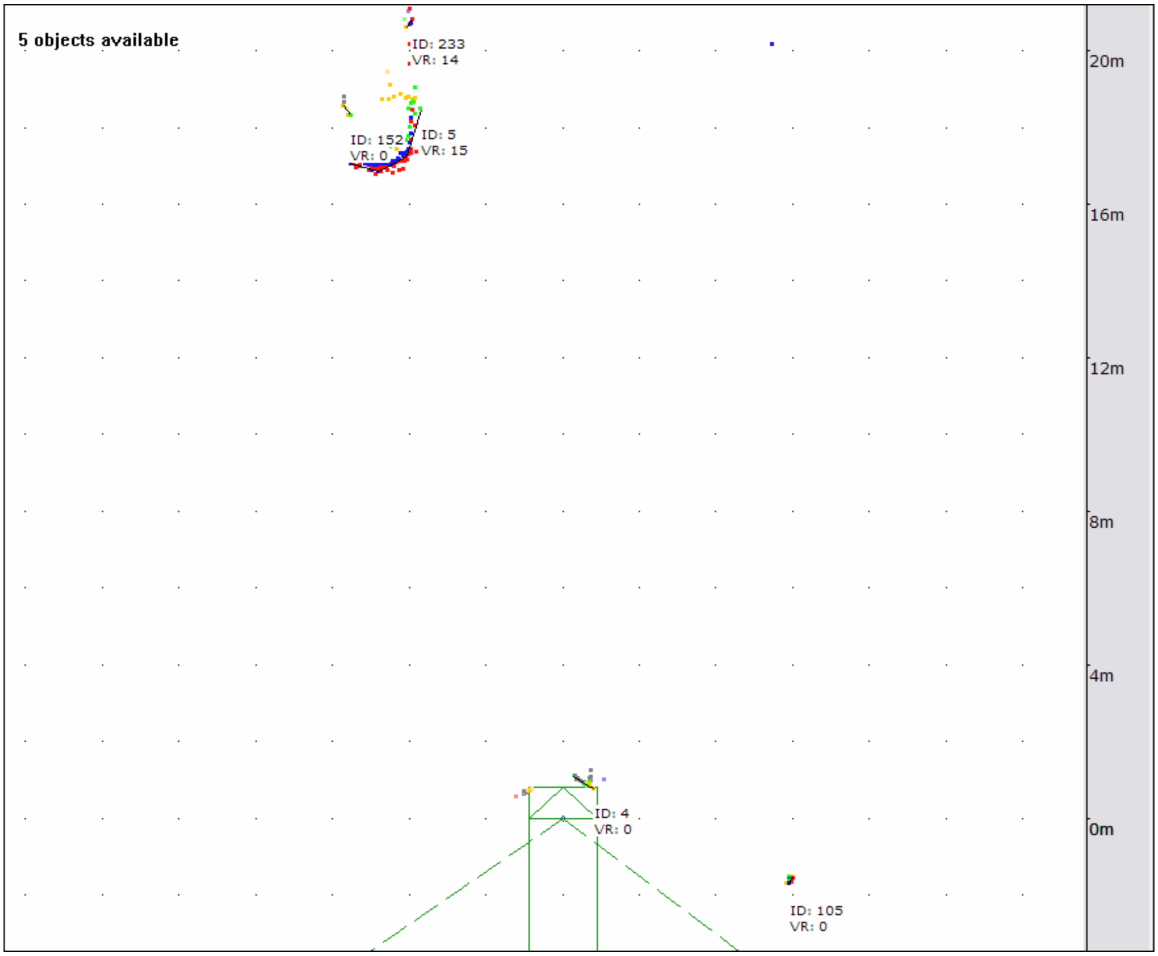


Figure A-7. Odin is stationary while Vehicle 5 travels past in the oncoming lane. Scan points hitting the side of the vehicle are completely reflected, leaving a gap in returns. The result is that the single vehicle is actually displayed as multiple unique objects (5, 152, and 233).

Appendix B –Vehicle Characteristic Examples

This appendix illustrates some results from the vision processing routines for tail lights and tires. Tail light detection figures have three panels: the initial image, the result after the threshold operation, and the final image before particle analysis. The images panels, the analysis result, and any details relevant to the process for that object are described under each figure.

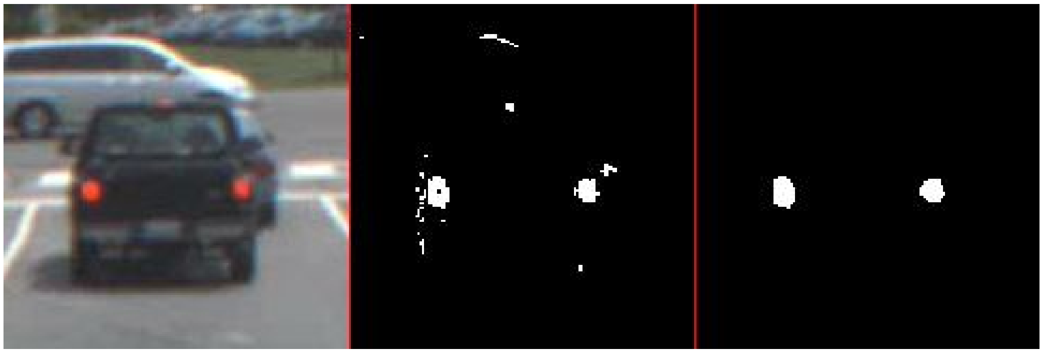


Figure B-1. A black truck with the brake lights illuminated. The software successfully found the tail lights.

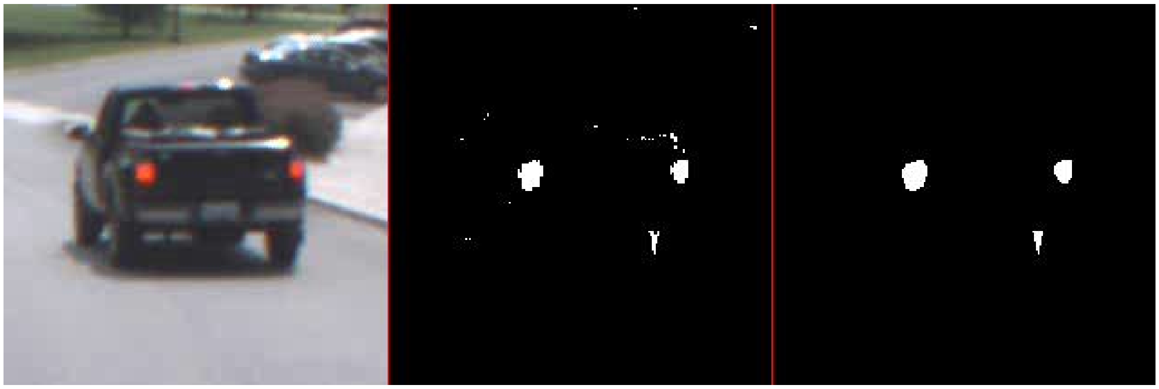


Figure B-2. The same black truck with the brake lights illuminated at an angle. The software again successfully finds the tail lights.

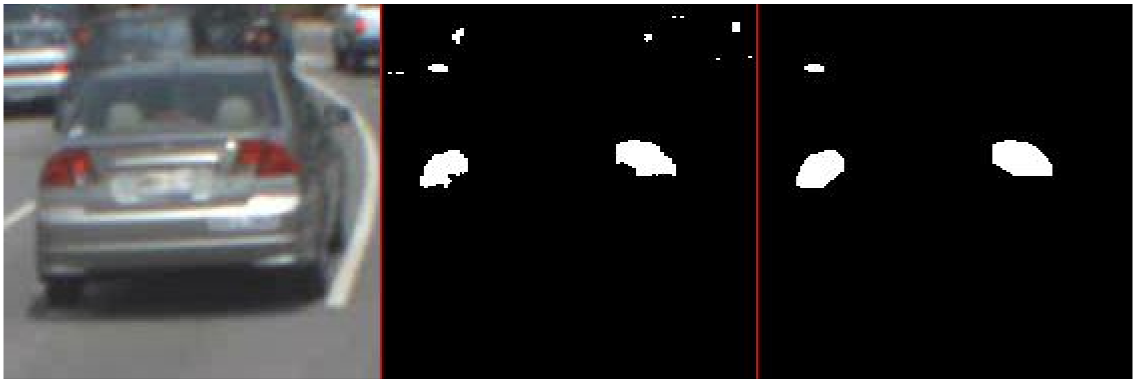


Figure B-3. A silver car in heavy traffic slightly at an angle. The software finds only the tail lights of the correct vehicle even with the others in the background.

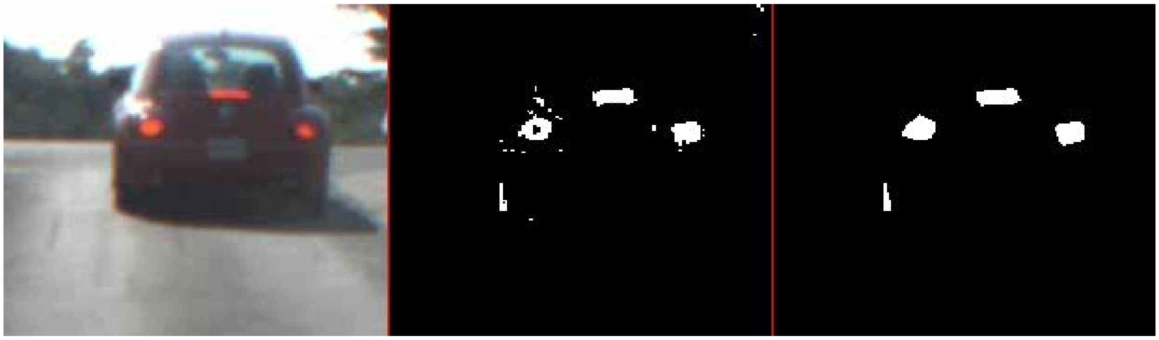


Figure B-4. A red beetle with tail lights illuminated. Although the vehicle is red as well, the shadows on the vehicle body allow the tail lights to be segmented and successfully detected.

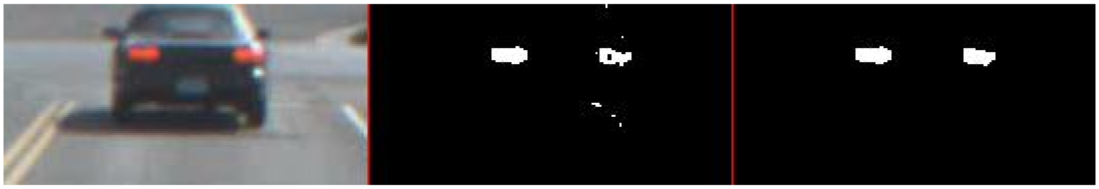


Figure B-5. A dark green car with tail lights illuminated. This car is further than any of the previous examples, but because the lights are illuminated the software is successful.

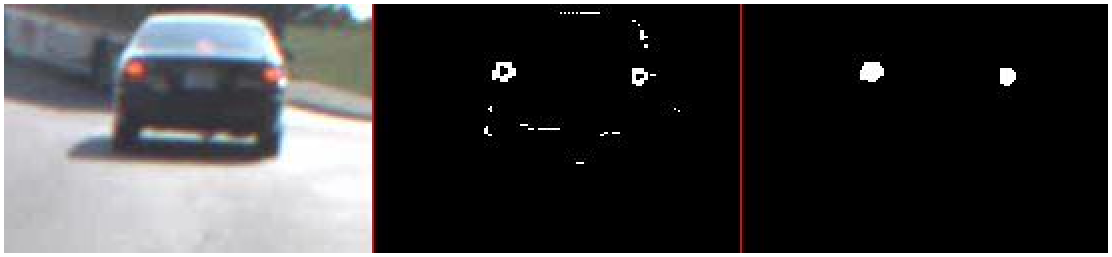


Figure B-6. A dark car with small tail lights illuminated. The software is successful once again.

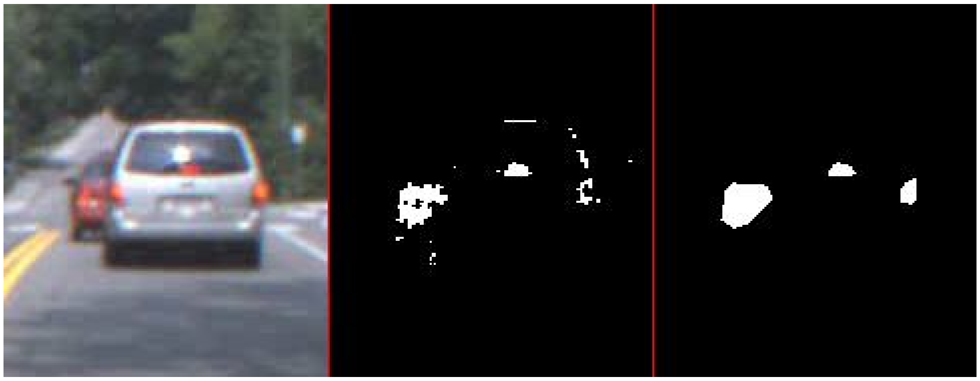


Figure B-7. A white van with small tail lights illuminated. The software fails to detect the tail lights based on the area, height difference, and ratio criteria. This failure is due to the fact that the red car in front of the van can be seen, and is combined with the side of the left tail light.

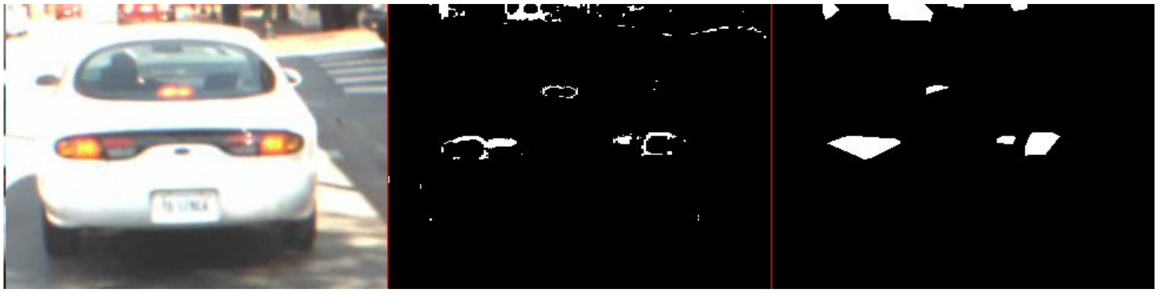


Figure B-8. A white car with tail lights illuminated. The software fails to detect the tail lights based o the area and ratio criteria. This failure is due to the image being extremely bright after coming out of a shaded area, causing the hue of the tail lights to be shifted outside of the threshold range.

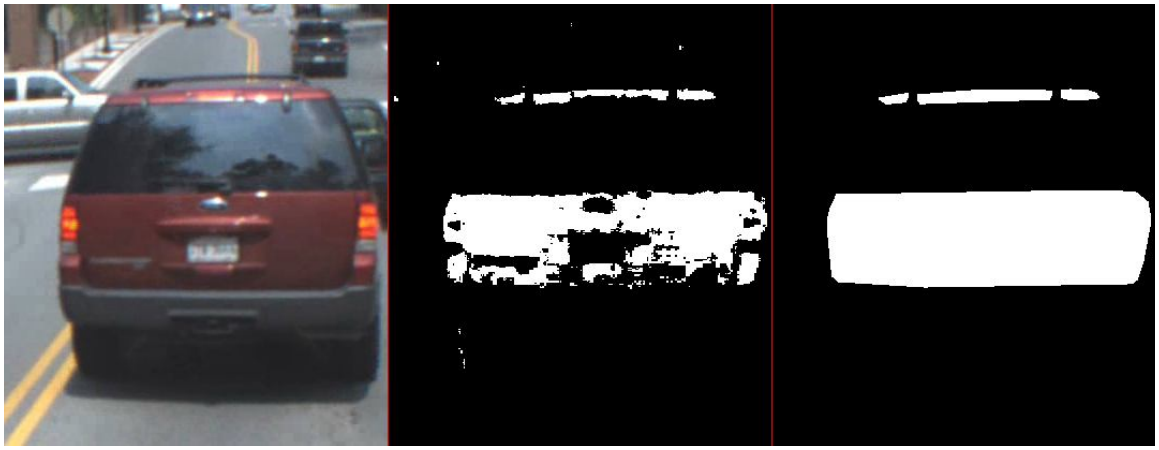


Figure B-9. A red van with tail lights illuminated. The software fails to detect the tail lights due to the red body color. However, the red car detector is able to determine that there is a single large particle in the image that is most likely a red car.

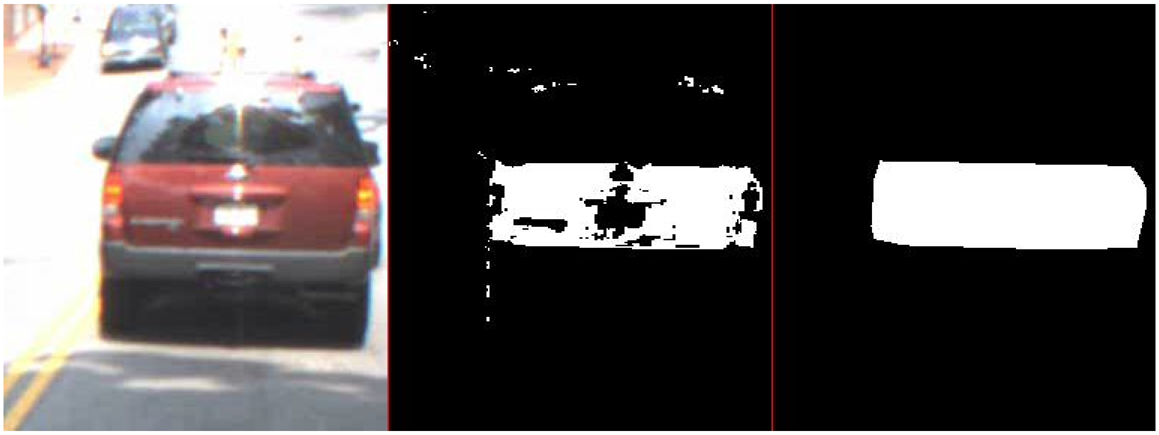


Figure B-10. The same red van with tail lights illuminated. Even with the whiteout condition caused by Odin leaving a shaded area the red car detector succeeds again. If the car was not red the software would probably fail to detect the lights for the same reasons as Figure B-8.

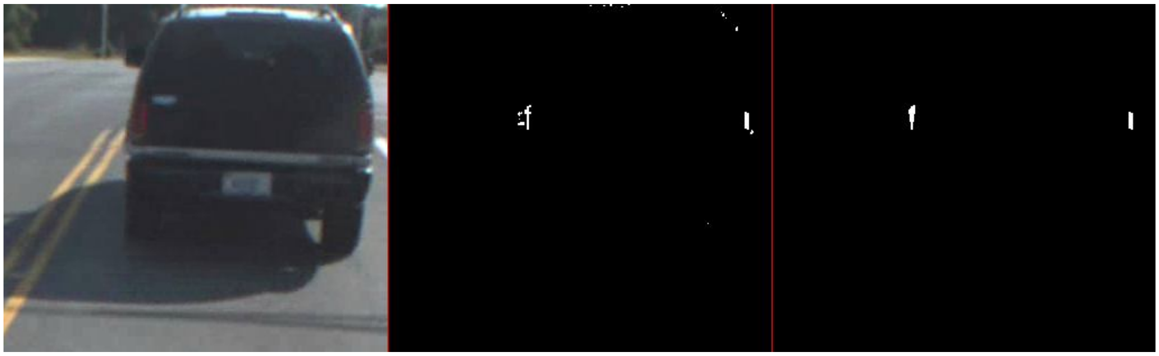


Figure B-11. A dark van with the tail lights not illuminated. Since the rear of the vehicle is not illuminated by the sun, the software is barely able to detect the tail lights. If the vehicle was any further away the tail light detector would probably fail in this condition.

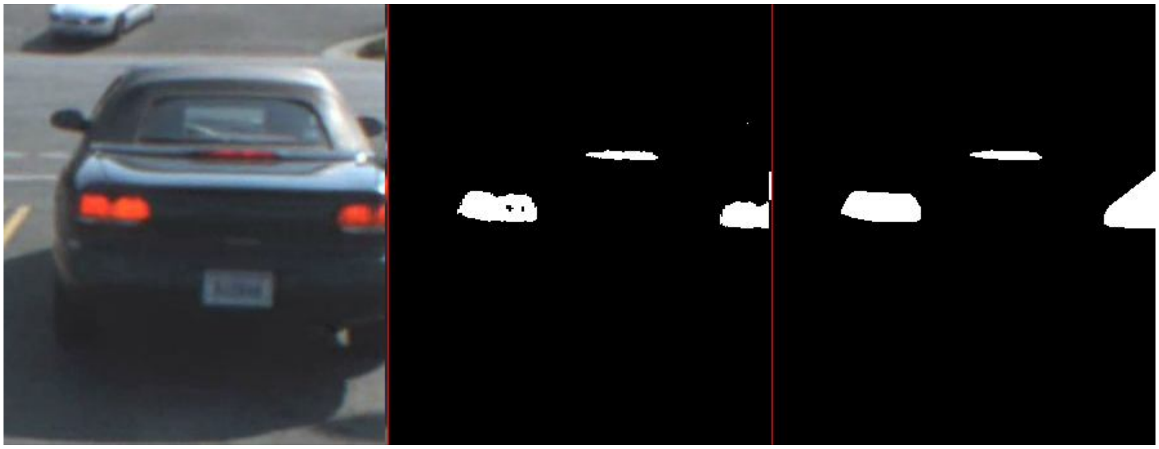


Figure B-12. A blue car with tail lights illuminated. Even though the vehicle is close to Odin the detector fails due to a bad region being selected. This illustrates the importance of accurate region extraction, and is support for the addition of some form of occlusion reasoning for characteristic pairs.

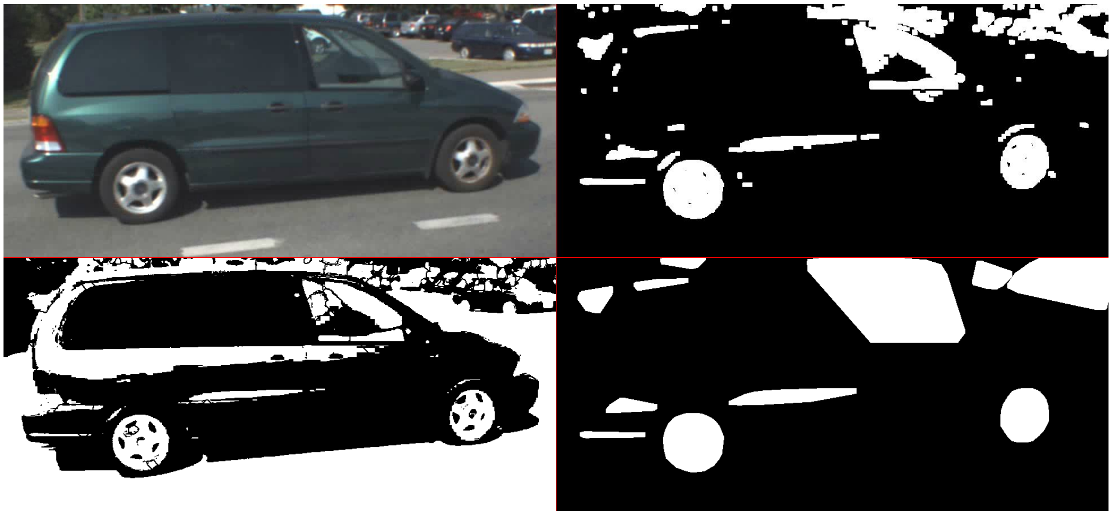


Figure B-13. A blue van observed at a short distance. The tire detector is able to segment out the tires and return the correct result.

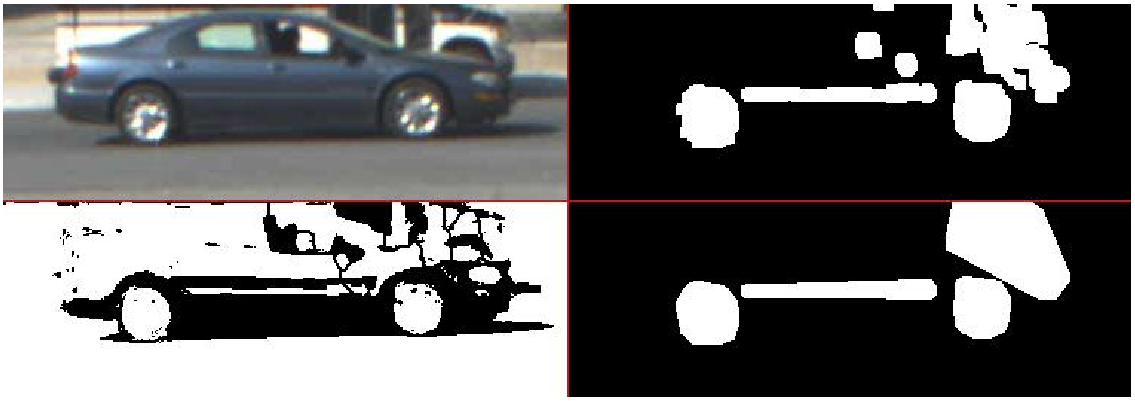


Figure B-14. A blue car observed at a medium distance. The tire detector is again able to successfully segment out the tires.

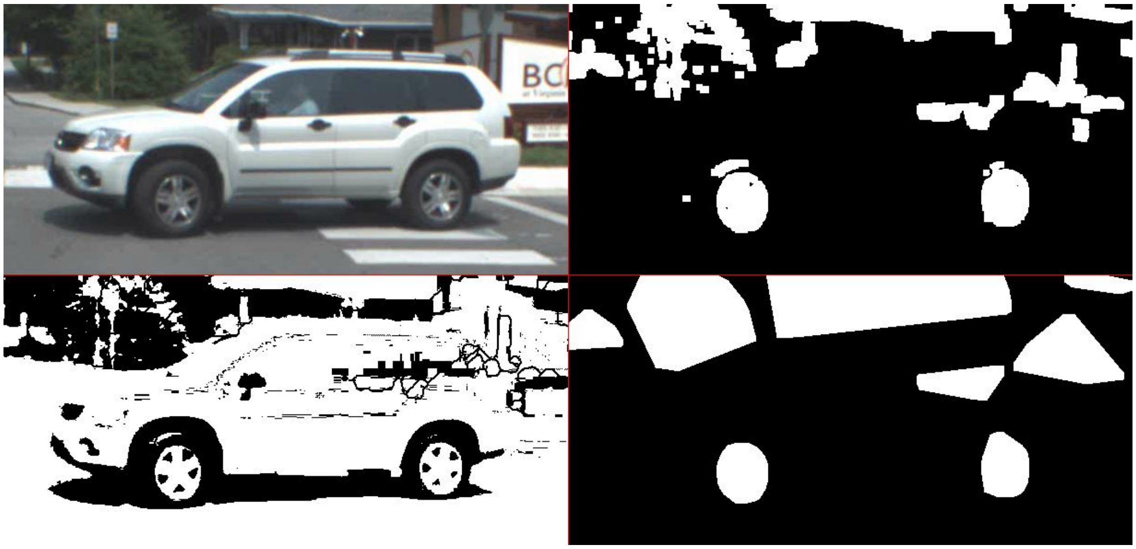


Figure B-15. A white SUV observed at a short distance. The detector is able to find the tires even with the slight deformation to the rear tire.

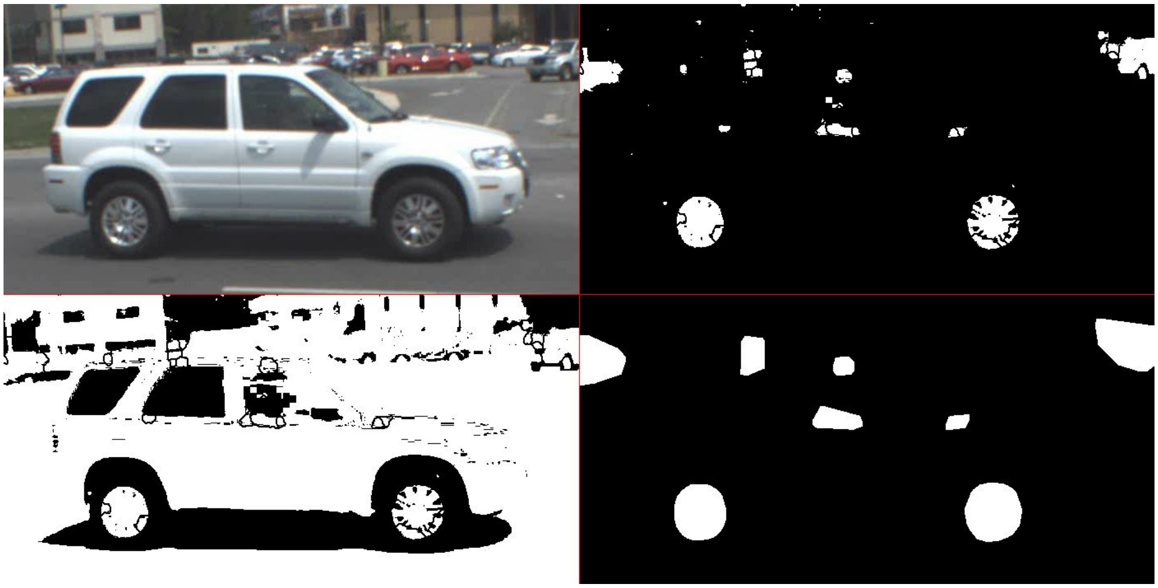


Figure B-16. Another successful result of a white SUV observed at a short distance.

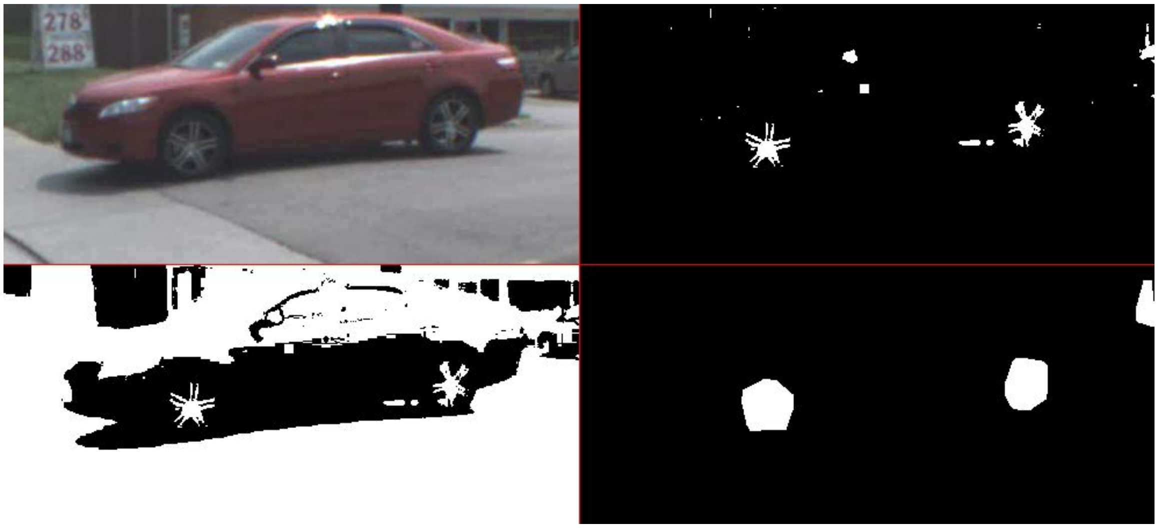


Figure B-17. A red car observed on a slight hill. Even though the hubcaps are not solid, the detector is able to round them out and determine the two particles are tires.

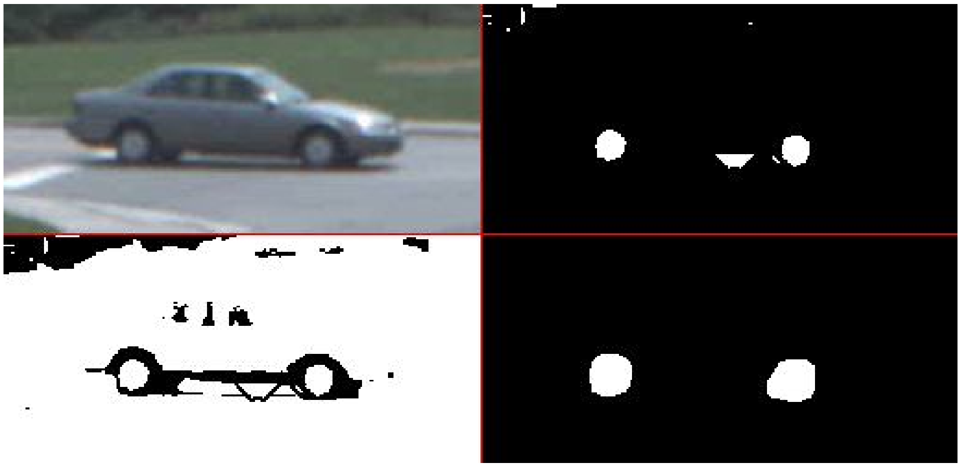


Figure B-18. A grey car observed at a relatively far distance. Even at approximately 20 meters the black tires give the center sections enough of a buffer that they can be segmented out.

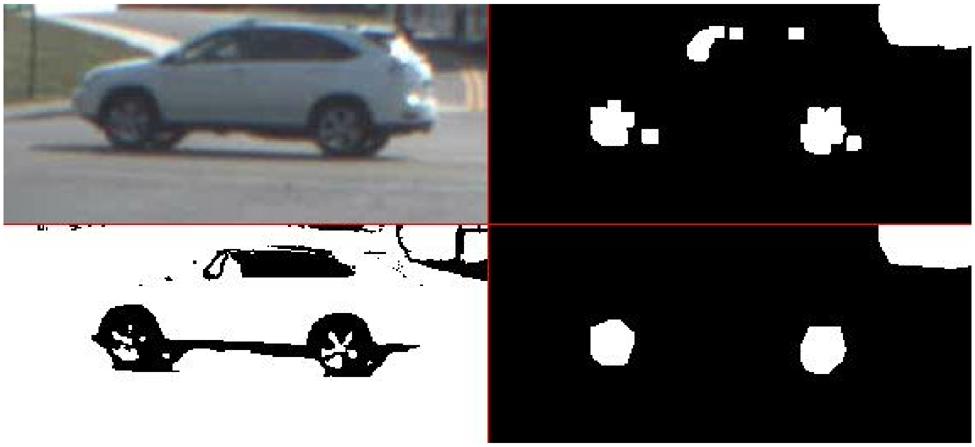


Figure B-19. A white SUV observed at a far distance. Once again the software is successful with the segmentation even at a good distance if the lighting is favorable.

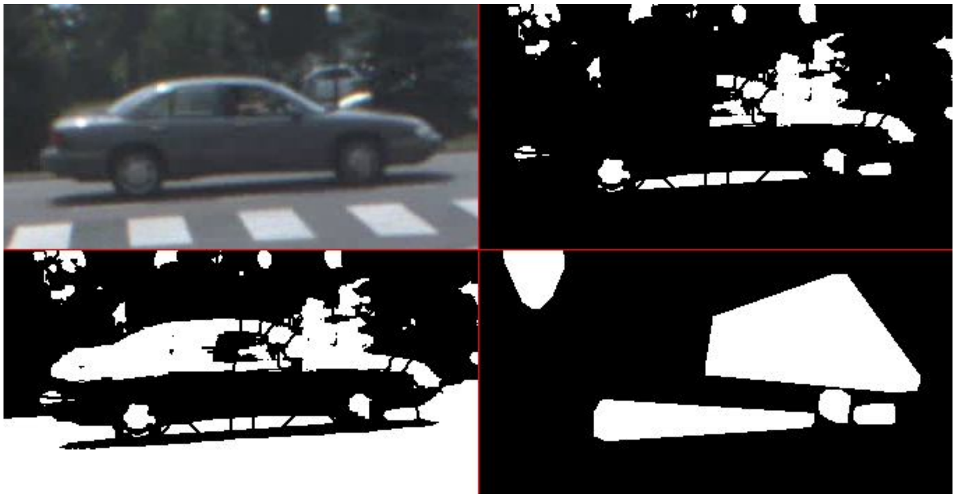


Figure B-20. A grey car observed at a medium distance in poor lighting. In this case even though the hubcaps are able to be segmented, the light area seen under the body is merged with one of the tires. The result is that the software is unable to find a particle match.

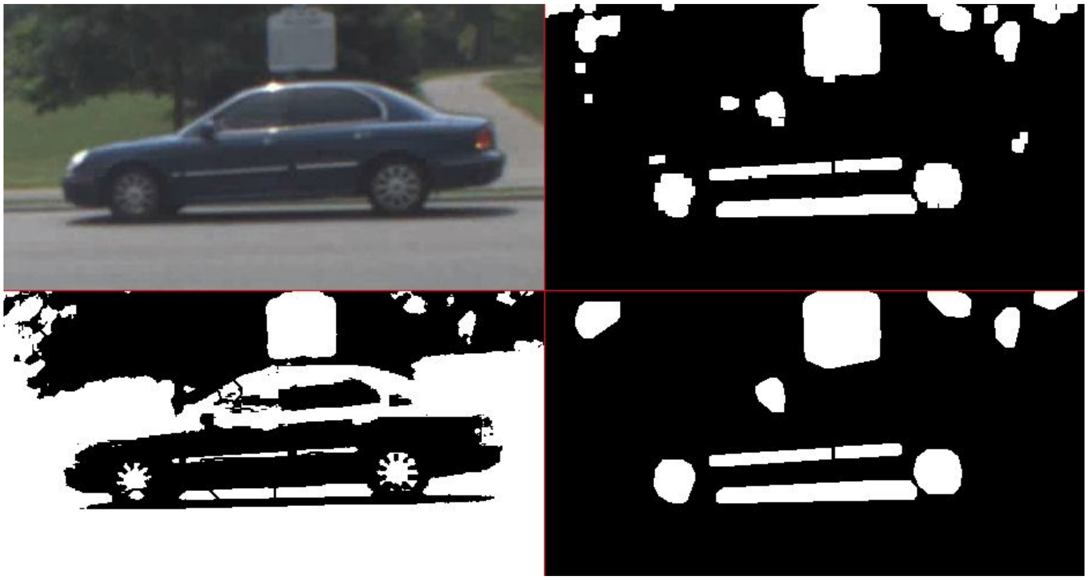


Figure B-21. A blue car observed at a medium distance. This view is similar to the one seen in Figure B-19, but because the lighting is more favorable the detector is able to correctly segment out the tires.

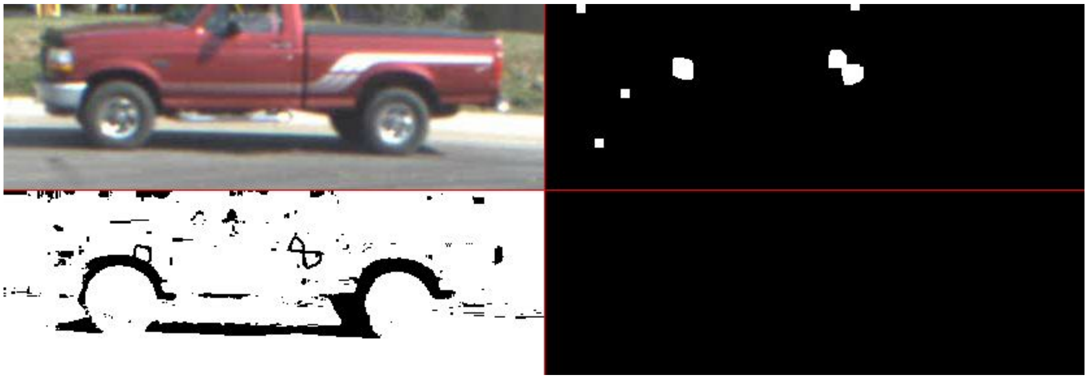


Figure B-22. A red truck observed at a medium distance. Here the lighting conditions are unfavorable again, this time due to the large amount of light. The detector is unable to segment out the hubcap from the rest of the tire.

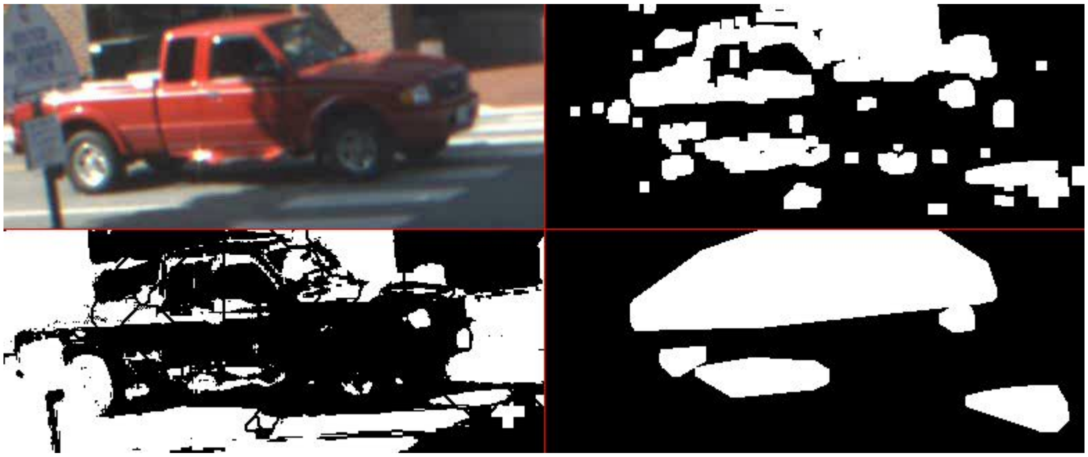


Figure B-23. A red car half in the shadow of a building. The rear tire can not be segmented due to how bright the surrounding tire is, while the forward tire fails due to the low light in the shadow area.

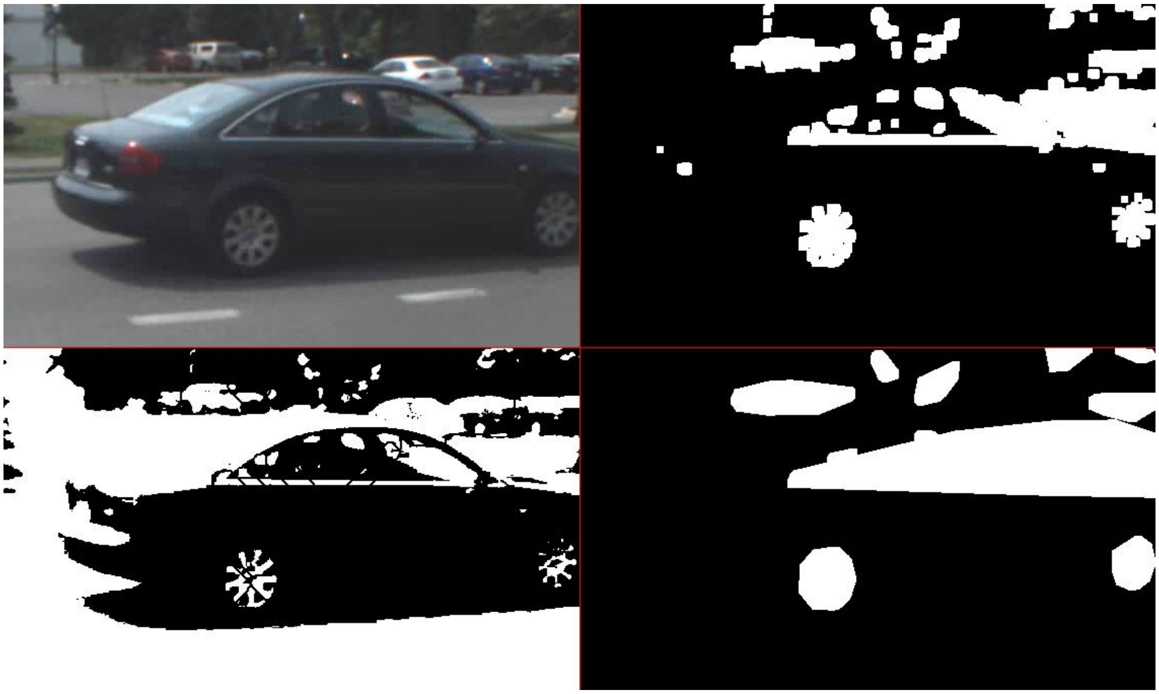


Figure B-24. A blue car observed at a short distance. Even though the entire car is not in the extracted region, the segmentation and subsequent detection succeeds. This figure illustrates how the tire detection software is not as sensitive to the region extraction as the tail light software is due to the location of the tires relative to the edges of the vehicle.

Vita

Stephen Cacciola was born on October 31st 1982 in South Kingstown, Rhode Island. After graduating from South Kingstown High School in 2000, he enrolled in the mechanical engineering program at Virginia Tech. As his senior design project, Stephen worked with the DARPA Grand Challenge team. His involvement with the project led to his continued education by enrolling in Virginia Tech's mechanical engineering graduate program. During his master studies he participated in the DARPA Urban Challenge competition, focusing on object detection and classification for autonomous vehicles. Stephen's term of graduation is fall of 2007, after which he will work for TORC Technologies in Blacksburg, VA.