TRACKING DYNAMIC OBSTACLES IN A STRUCTURED URBAN ENVIRONMENT AND SUBSEQUENT DECISION MAKING FOR AN AUTONOMOUS GROUND VEHICLE

by

DANIEL F PAWLOWSKI

Submitted in partial fulfillment of the requirements

For the degree of Master of Science

Thesis Advisor: Dr. Roger Quinn

August 2008
CASE WESTERN RESERVE UNIVERSITY
SCHOOL OF GRADUATE STUDIES

We hereby approve the thesis/dissertation of

___Daniel Pawlowski____________________________________

candidate for the _____Master of Science_____________degree *.

(signed)___Roger Quin____________________________________

(chair of the committee)

___Michael Branicky____________________________________

___Christopher Hernandez__________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

(date) ___5/30/08_____________

*We also certify that written approval has been obtained for any
proprietary material contained therein.
Contents

List of Figures vi
List of Algorithms vii
Abstract viii

1 Introduction and Motivation 1
   1.1 DARPA Urban Challenge ........................................... 2
   1.2 Dexter’s AI .............................................................. 3
   1.3 Thesis Outline .......................................................... 11

2 Background 13
   2.1 Past Research .......................................................... 13
      2.1.1 Obstacle Tracking .............................................. 13
      2.1.2 Obstacle Avoidance ........................................... 16
      2.1.3 Lead Vehicle Following ...................................... 21
      2.1.4 Observing Intersections .................................... 21

3 Obstacle Tracker 23
   3.1 Filter Detected Objects ............................................. 24
   3.2 Match Detected Objects to Tracked Obstacles .................. 25
   3.3 Estimate Obstacle Movement ..................................... 29
   3.4 Obstacle Memory ...................................................... 36
3.5 Results ................................................................. 38

4 Lead Object Observer ........................................ 40
   4.1 Passability Check ................................................. 40
   4.2 Jam Lane Description ............................................. 44
   4.3 Results ............................................................. 45

5 Stay in Lane – Maintain Speed Behavior .......... 47
   5.1 Return to Lane Algorithm .................................... 48
   5.2 Modifying Tight Turns ......................................... 49
   5.3 In-Lane Obstacle Avoidance ................................... 50
   5.4 Jam Behavior ..................................................... 52
   5.5 Results ............................................................. 55

6 Lead Vehicle Observer ...................................... 56
   6.1 Lead Vehicle Observer .......................................... 56
   6.2 Results ............................................................. 58

7 Intersection Observer ..................................... 60
   7.1 Intersection Geometry .......................................... 60
   7.2 Four-way Stop Intersections ............................... 62
   7.3 No-Stop Intersections ............................................ 64
   7.4 Results ............................................................. 67

8 Conclusions ..................................................... 68
   8.1 Results Summary ................................................ 68
      8.1.1 Area A .......................................................... 69
      8.1.2 Area B .......................................................... 74
      8.1.3 Area C .......................................................... 75
   8.2 Future Work ...................................................... 77
8.2.1 Obstacle Tracking ............................................. 77
8.2.2 Obstacle Avoidance ........................................ 79
8.2.3 Lead Vehicle Following .................................... 79
8.2.4 Observing Intersections .................................... 80

Bibliography ......................................................... 81
## List Of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Dexter, Team Case’s entrant in the 2007 Urban Challenge</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>Overview of Dexter’s AI Structure</td>
<td>4</td>
</tr>
<tr>
<td>1.3</td>
<td>SICK LIDAR unit</td>
<td>6</td>
</tr>
<tr>
<td>1.4</td>
<td>Dexter’s LIDAR positioning</td>
<td>7</td>
</tr>
<tr>
<td>1.5</td>
<td>View of the Mood Selector VI</td>
<td>8</td>
</tr>
<tr>
<td>1.6</td>
<td>Example of Obstacle Delay in Dexsim</td>
<td>10</td>
</tr>
<tr>
<td>2.1</td>
<td>LIDAR-Based Obstacle Recognition. Reproduced from [4]</td>
<td>14</td>
</tr>
<tr>
<td>2.2</td>
<td>Example of use of potential fields to represent obstacles. Reproduced from [6]</td>
<td>17</td>
</tr>
<tr>
<td>2.3</td>
<td>Path modeling using elastic bands as inspiration. Reproduced from [6]</td>
<td>18</td>
</tr>
<tr>
<td>2.4</td>
<td>C-space representation of obstacles. Reproduced from [9]</td>
<td>19</td>
</tr>
<tr>
<td>2.5</td>
<td>C-space representation using a rotating object. Reproduced from [9]</td>
<td>19</td>
</tr>
<tr>
<td>2.6</td>
<td>Holonomic path conversion for nonholonomic use. Reproduced from [18]</td>
<td>20</td>
</tr>
<tr>
<td>3.1</td>
<td>Output of the Obstacle Tracker</td>
<td>24</td>
</tr>
<tr>
<td>3.2</td>
<td>Filtering detected objects</td>
<td>25</td>
</tr>
<tr>
<td>3.3</td>
<td>LIDAR noise causing heading estimates to wander</td>
<td>31</td>
</tr>
</tbody>
</table>
3.4 Example of obstacle jitter following bug fix
3.5 The path of a tracked vehicle testing detection of stop points
3.6 Tracker estimates of a starting and stopping vehicle
3.7 Obstacle memory at intersections
3.8 Obstacle memory in parking lots
4.1 Impassible obstacles
4.2 The obstacle merging process
4.3 Example of a simple Jam condition
5.1 Return to lane behavior
5.2 Simple in-lane obstacle avoidance
5.3 Multiple obstacle in-lane avoidance
5.4 Example of Jam behavior
5.5 Example of complicated jam behavior and performance
6.1 Speed estimates of a stopping vehicle
6.2 Speed estimates of a vehicle accelerating
7.1 Example of basic intersection geometry
7.2 Tracking estimates of a moving vehicle
8.1 RNDF of Area A
8.2 Late obstacle detection
8.3 Picture of a problem-causing cement barrier
8.4 Raw LIDAR points of cement barrier
8.5 Erroneous LIDAR fits of cement barrier
8.6 More problem cement barriers
8.7 RNDF of Section B
8.8 RNDF of Section C
8.9 Obstacle occlusion at intersections ................................................................. 76
8.10 Intersection Observer output of occlusion ...................................................... 76
8.11 Picture of a roadblock .................................................................................. 77
8.12 Occlusion detection failure ......................................................................... 78
# List Of Algorithms

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Matching detected objects to tracked obstacles</td>
<td>26</td>
</tr>
<tr>
<td>3.2</td>
<td>Analyzing all linked obstacles as a group</td>
<td>27</td>
</tr>
<tr>
<td>3.3</td>
<td>Prospective algorithm for creating new obstacles</td>
<td>28</td>
</tr>
<tr>
<td>3.4</td>
<td>Updating an obstacle’s speed and heading</td>
<td>29, 30</td>
</tr>
<tr>
<td>3.5</td>
<td>Method for determining stationary vehicles</td>
<td>32</td>
</tr>
<tr>
<td>3.6</td>
<td>Occlusion rules</td>
<td>36</td>
</tr>
<tr>
<td>4.1</td>
<td>Analyzing obstacles within the travel lane</td>
<td>42</td>
</tr>
<tr>
<td>4.2</td>
<td>Handling impassible obstacles</td>
<td>44</td>
</tr>
<tr>
<td>5.1</td>
<td>Shift breadcrumbs algorithm</td>
<td>48</td>
</tr>
<tr>
<td>6.1</td>
<td>Estimating a lead vehicle’s velocity</td>
<td>57</td>
</tr>
<tr>
<td>7.1</td>
<td>Intersection precedence at 4-way stops</td>
<td>63</td>
</tr>
<tr>
<td>7.2</td>
<td>Tracking vehicles at no stop intersections</td>
<td>65</td>
</tr>
</tbody>
</table>
The 2007 DARPA Urban Challenge prompted innovations in the field of automated ground vehicles. One of the most complicated new tasks was interaction with other moving vehicles. Previously, the operating environment was static, and therefore limited tracking capabilities were needed. Using a Kalman filter-based approach, dynamic obstacles are tracked with accurate estimations of each obstacle’s speed and direction of travel. This information is used in determining intersection precedence for both stopped and moving traffic. Vehicles were required to stay within their respective lane markings at all times. If possible, static obstacles are avoided within the lane boundaries; dynamic obstacles are tracked for the purpose of allowing a safe following distance.
Chapter 1

Introduction and Motivation

The purpose of this thesis was to provide an automated ground vehicle, Dexter, Case Western Reserve University’s entrant in the 2007 DARPA Urban Challenge, with reliable obstacle tracking, vehicle following, obstacle avoidance, and decision making at intersections. Dexter, as pictured in Figure 1.1, was built from the ground up for a previous Urban Challenge and donated by Ensco, Inc. The following five modules were developed for this thesis; the Obstacle Tracker, the Lane Object Observer, the Stay in Lane – Maintain Speed behavior, the Lead Vehicle Observer, and the Intersection Observer.
1.1 DARPA Urban Challenge

In response to a mandate from Congress that one-third of all military vehicles be automated by the year 2015, the Defense Advanced Research Projects Agency (DARPA) sponsored an autonomous ground vehicle race called the Grand Challenge in 2004 [4]. The race consisted of a 125+ mile course through desert terrain. After no vehicles successfully completed the course in 2004, the event was held again in 2005, during which five teams completed the event. In 2006 DARPA announced that they would be hosting another race in 2007, which would take place in an urban environment, comprised of regular road surfaces with the presence of traffic vehicles. Race entrants were required to be able to follow all California traffic laws, while navigating through complex intersections (with or without the presence of stop signs), partial or full lane blockages, and complex parking lots. This new setting forced major innovations in the field of automated ground vehicles. Particularly, previous work done regarding obstacle
detection and tracking was unsuited for a structured environment with the presence of other dynamic vehicles. Intersections presented an entirely new problem, as no substantial work had previously been performed regarding intersection precedence and safety requirements.

Prior to the final race, each team first submitted a video of their robot performing basic tasks. Upon approval of a team’s video entry, officials from DARPA travelled to each teams’ site to witness a more complicated on-site performance. Finally, 35 teams were invited to travel to Victorville, California to take part in the National Qualification Event, composed of three separate courses that required various skills to complete. Area A required several passes through intersections, in the presence of oncoming traffic that did not stop. Area B tested basic lane following skills, traversing a course with several bends and basic intersections without the presence of traffic, as well as testing the ability to park and exit parking spots. Finally, Area C tested the vehicle’s ability to determine intersection precedence at four-way stop intersections and the vehicle’s ability to replan its desired route and perform u-turns in the face of road blockages. More information about the rules regarding the competition can be found in [4].

1.2 Dexter

Dexter’s artificial intelligence (AI) control system architecture is shown in Figure 1.2. The vehicle operates given an RNDF (Route Network Definition File) and MDF (Mission Data File). The RNDF provides a sparse GPS map of the course which the robot will traverse. This map consists of points referred to as waypoints, a number of which are designated as checkpoints, which are used in specifying the mission for the vehicle. The RNDF is augmented
with mapper beads, which is a more detailed GPS description of the road, with beads spaced approximately 1 meter apart. These beads can be created automatically by parsing waypoints in the RNDF or manually by analyzing satellite imagery. The MDF (Mission Data File) is a list of checkpoints in the order in which Dexter must navigate them.

The Global Mapper module converts the information contained in the RNDF into the data types other modules require. This ranges from basic lane descriptions, to parking lot definitions, and intersection geometry. This information is available to other modules via a UDP (User Datagram Protocol) request, which is the mode of all inter-module communications within Dexter’s computer architecture. UDP messaging is a form of network communications which provide rapid transfer of information, reliable for time-sensitive operations.

The Route Planner analyzes the various lane connections at intersections in
order to generate a path for the robot based on what checkpoint sequence the MDF specifies as the current mission. The Physical State Observer is used to determine Dexter’s starting position for this path, as well as knowing when a checkpoint goal has been realized. The route plan is also used to determine when Dexter is approaching intersections or zones.

There are numerous observers which play a part in deciding Dexter’s actions. Some are self-explanatory, such as the Physical State Observer, which monitors the vehicle’s global position, as well as speed, heading, steering angle, etc. Dexter has two GPS receivers that can be accurate to 10 cm. It also has a six-axis inertial measurement unit (IMU), which measures linear accelerations and angular rates of the vehicle. The GPS and IMU outputs are integrated as to provide accurate tracking of Dexter’s movements. The Lane Observer performs a UDP request to the Global Mapper to receive the current lane description. This data is merged with the output from visual and LIDAR based estimates of the lane edges (if enabled), if this sensor data is relatively believable, based on the expected lane geometry. The Obstacle Tracker Observer tracks and predicts the movement of obstacles from data generated by the six LIDAR units, and, although ultimately unused during the competition, radar and stereo cameras.

This data is used for intersection awareness, obstacle avoidance, and vehicle following. The Lead Vehicle Observer tracks the distance to and speed of impassible objects. Originally, this was done with a Kalman filter, but due to difficulties with the algorithm and the graduation of the original author, this was altered to a simpler relative differential system that will be explained in detail later. The VBIL uses the calculated speed and distance ahead of lead vehicles to modulate the speed of Dexter as to provide safe following. This observer uses information gleaned from the Lane Object Observer, which analyzes objects in the current travel lane and determines whether or not they are avoidable without changing lanes. Finally, the Intersection
Observer analyzes intersections to determine precedence regardless of intersection geometry and the presence or absence of stop signs.

The sensor package that Dexter carries includes SICK LMS-291 laser rangefinders (LIDAR, for Light Detection and Ranging), with a resolution of 0.5° over a 180° scan plane at 40 Hz, and accurate to 80 m (seen in Figure 1.3). As illustrated in Figure 1.4, there were two units mounted on the front of Dexter, one on each of its sides and one on the rear of the vehicle. These sensors are reliable, depending on the surface material and the angle of the object that has been scanned. Specifically, materials such as reflective tape on construction barrels and specific cars’ hoods/windshields proved difficult to detect.

*Figure 1.3: SICK LIDAR unit*
Dexter also carries an Eaton Vorad EVT-300 radar package, which is capable of tracking moving objects in a 12º band out to 100 m. Additionally, obstacle detection was done via a Videre stereo camera unit. Road tracking was done with eight 640*480 resolution cameras (apparent in Figure 1.3), as well as a downward-facing hybrid LIDAR unit, capable of detecting reflectivity properties of the road bed and lane markings (this is the unit shown in detail in Figure 1.3).

Decisions to act are done in the Mood Selector module. Based on the current route plan and observer information, the Mood Selector decides which legal action should be performed.
There are a finite set of driving maneuvers that Dexter can complete called moods. This finite-state decision-making process allows constant analysis of goal achievement and ease of choosing the correct mood to accomplish these goals. Figure 1.5 presents the various moods that the Mood Selector can call, such as navigating parking lots, changing lanes, moving through intersections, negotiating a traffic jam environment, and performing a u-turn. These moods control execution of lower level modules called behaviors.

Behaviors are responsible for creating the path for Dexter to traverse. This path will be referred to as the breadcrumb path. Breadcrumbs are attractor points that Dexter follows. The breadcrumb data type holds all information needed for Dexter to follow each crumb, including speed limits, travel direction (forward/reverse), and aggressive/passive driving behavior. The breadcrumb path, upon creation, is sent to the VBIL (Vehicle Behavior Interface Layer). The VBIL analyzes the breadcrumb trail and imposes safety limits based on Dexter’s physical constraints, such as turning radius and global speed limit as well as an obstacle collision fail-safe. The VBIL also analyzes lead vehicle information and alters the breadcrumb trail’s speed.
limits as to provide a safe following distance behind the vehicle. The Vehicle Controller follows this finalized path.

The Vehicle Controller follows the breadcrumb path using a wagon-handle steering algorithm. The wagon handle is a calculated path to return to the prescribed path dependent on the robot’s speed. The speed determines the distance along the path along which Dexter aims to return to the trail. Based on the trail as well as Dexter’s position and heading, this wagon handle path is calculated. Further detail on Dexter’s software and hardware can be found in [19].

The artificial intelligence code can also be run on a simulator called Dexsim, developed by Christian Miller [3]. This enabled easy debugging, especially for moods and behaviors. Since Dexter was fully autonomous (no steering wheel or seat), testing code on the robot involved shipping it to a test track for the day, and thus the simulator was an invaluable tool. Unfortunately, in simulation, there is a noticeable delay in the LIDAR driver that causes the raw LIDAR scans to lag behind when Dexter is moving or turning, which causes a pronounced effect as illustrated in Figure 1.6.
Figure 1.6: Example of Obstacle Delay in Dhexsim. This screenshot was taken while Dexter was traveling approximately 15 mph. The green block is a computer generated obstacle, while the orange line represents Dexter’s view of the obstacle.

The five particular modules that were developed or modified for this thesis shall be discussed in more detail. The lowest level module is the Obstacle Tracker, responsible for filtering detected objects into tracked obstacles of concern, predicting tracked obstacle movement, and providing rudimentary obstacle memory. In order to provide information necessary for observing intersections, it is necessary for the tracker to accurately and quickly reflect vehicles’ speeds as they come to a stop. Also, it is possible that obstacles can occlude each other and thus the tracker must have a reliable ability to recognize, remember, and predict movement of obstacles that are fully or partially blocked.

The Lead Object Observer analyzes Dexter’s current lane of travel to determine if any obstacles are present. It calculates whether or not an obstacle can be passed without changing lanes. Impassible objects are routed to be analyzed by the Lead Vehicle Observer, while passable
obstacles are evaluated to determine the route that must be taken to perform in-lane avoidance. The Lead Vehicle Observer tracks the rate of travel of impassible objects as well as their distance in front of Dexter. Originally, this was done with a Kalman filter that used the entire lane length to determine a vehicle’s position and subsequent rate of change in position. This was altered to a much simpler, but still effective version that uses the object’s position relative to Dexter.

The Stay in Lane – Maintain Speed behavior is what performs nearly all of Dexter’s driving maneuvers. Even other behaviors, such as changing lanes and negotiating intersections, use this behavior for the breadcrumb path generation. Stay in Lane simply follows a given lane description, with the ability to negotiate passable obstacles identified by the Lead Vehicle Observer.

Finally, the Intersection Observer was also developed for this thesis. This module has the ability to determine lawful intersection precedence based on what vehicle arrived and stopped at the intersection first. Intersections can contain roads which have no stop sign. In this case, the approaching vehicle’s speed is accurately determined and the estimated time until that vehicle enters the intersection is calculated. With this information, Dexter decides if he has enough time to safely clear the intersection.

1.3 Thesis Outline

Chapter 1 introduces the thesis and provides a brief summary of the impetus for the research, the 2007 Urban Challenge. Dexter’s system architecture is discussed, presenting the modules developed for the thesis. The organization of the thesis is also presented.
**Chapter 2** details background research that has previously been performed in the pertinent areas.

**Chapter 3** presents the Obstacle Tracker module, detailing the process in which raw observed data is transformed into obstacles that are tracked with an estimate of their dynamic behaviors. Additional requirements, such as rudimentary obstacle memory, are also discussed.

**Chapter 4** details the Lane Object Observer which analyzes the current travel lane for the presence of obstacles and determines the required avoidance if the obstacle is passable within the lane limits. If not, information about the obstacle is sent to the Lead Vehicle Observer.

**Chapter 5** discusses the Stay in Lane – Maintain Speed behavior, which creates a path for the vehicle to follow. Various features are detailed, in particular the alteration of the original path in order to avoid obstacles while staying within lane boundaries.

**Chapter 6** details the process in which impassible obstacles are analyzed to determine their speed and position, which is performed by the Lead Vehicle Observer. This information is passed to other algorithms, which modulate the vehicle’s speed in order to provide a safe following distance.

**Chapter 7** presents the Intersection Observer. This observer determines intersection precedence in addition to analyzing oncoming traffic for safety concerns.

**Chapter 8** summarizes the work and results as well as presenting possible future work to be performed.
Chapter 2

Background

2.1 Past Research

Automated vehicles have been studied for over a decade, but this research has mostly dealt with vehicles in stationary structural environments, absent dynamic obstacles. As a result, obstacle detection was the primary concern, whereas tracking objects was of lesser importance. Also, the real-time vehicle control, steering, and tracking available were limited and unsuitable for high-speed performance. As progress was made in vehicle control as well as the introduction of dynamic traffic, obstacle tracking capabilities became needed.

2.1.1 Obstacle Tracking

Obstacle detection has been performed by laser rangefinders for about two decades, with the majority of work being on the grouping and cohesion of the range data collected by LIDAR units. Generally, the LIDAR range scans are broken down into suspected obstacles. Some
algorithms merely create a bounding box around the groupings and designate the box as an obstacle [22]. More advanced algorithms fit line segments to the points in a grouping and use these lines to represent the obstacle edges, as seen in Figure 2.1 [5].

Figure 2.1: (a) raw LIDAR range data points, (b) suspected obstacle groupings, (c) breakdown of fitting line segments within a grouping, and (d) final obstacle line segments. Reproduced from [5]

In Dexter’s AI, the LIDAR Module did not perform the initial grouping of suspected obstacles as this would be too demanding on computing resources with several LIDAR units in a dense obstacle field.

Although Dexter did not use downward facing LIDAR for obstacle detection, the capability certainly exists and could have been added to Dexter’s AI. This process involves analyzing LIDAR scans of road beds for the presence of bumps. Dexter wasn’t given this capability because the effectiveness breaks down at high speeds. The unit would be required to scan the road at a point very far along the road and any object that lay across the entire width of the road (such as a fallen tree trunk) would merely appear as a momentary bump across the road bed. The AI would be very suspicious of such a temporary bump; otherwise it would be prone to numerous false positives. More information on downward facing LIDAR can be found in [19].
The next step in obstacle tracking is the ability to match detected objects with obstacles tracked from previous cycles. The main limitation in this process is calculation time. One solution is what is called an anytime algorithm, defined as an “algorithm that returns the best answer possible even if it is not allowed to run to completion, and may improve on the answer if it is allowed to run longer” [20]. This solution performs a rudimentary match and then, as time is allowed, proceeds to perform more robust matches. The algorithm provides a mediocre solution quickly; then as time elapses the solution improves. For obstacle matching, this is done through initially performing restricted matching, essentially doing a m:1 match merely finding one tracked obstacle that matches each new detected object. The optimal matching sequence is then started, matching all detected objects with all tracked obstacles [20]. The restricted matching will rarely provide an adequate solution, particularly for this project’s needs. Dexter uses entirely optimal matching, but this did cause severe latency issues in dense obstacle fields. But the majority of obstacles fall outside of travel lanes and are thus irrelevant. Therefore, the solution was to filter out obstacles that lie entirely outside relevant areas, and then perform an optimal matching algorithm with a greatly reduced number of detected objects.

The final step in obstacle tracking is prediction. Accurately estimating an obstacle’s motion is key to the efficiency and accuracy of tracking detected objects, particularly in the cases of partial and full occlusion. Prediction is predominantly done by applying a Kalman filter to moving obstacles [22]. The Kalman filter is comprised of two steps: prediction and correction. Initially, the current location of the obstacle is predicted based on the previous iteration’s estimated speed, heading, and acceleration. After the matching process, the actual location of the obstacle is compared with the predicted position and the speed, heading, and acceleration estimates are altered based on the previous iteration’s estimate’s accuracy. This process is
explained in more detail in [7]. The process used in the Obstacle Tracker is a rudimentary Kalman filter and will be explained rigorously later in Section 3.3.

2.1.2 Obstacle Avoidance

There are numerous methods for obstacle avoidance developed over the past two decades. Figure 2.2 illustrates the preeminent idea for avoidance which is to model the driving surface as a virtual potential field. Detected obstacles are represented by a rise in potential. Dynamic obstacles can be represented with this method, simply by expanding the potential to include where the obstacle will be and lowering the potential where the robot just was or will move away from in the future. Path modification is influenced by the gradient of the potential field [7].
Other methods that have been used include treating avoidance as a fluid dynamics problem, using dynamic equations to find the resulting flow field through obstacles. Models that treat obstacles as forces that repel the vehicle’s path have also been researched. This model requires solving dynamic differential equations and thus consumes a large amount of computing resources [6].

A slight modification of finding the gradient of the potential field method is to model the path as an elastic band, upon which the potential field acts as a repulsive force.
Figure 2.3: Path modeled as an elastic band which can be treated as a collection of points attached by springs for the purpose of calculations. Reproduced from [5]

Apparently however, this method has only been proven to work on holonomic vehicles, thus making it ineffective for automotive use. Also, this approach is not suited for high speed operation due to the complex computing required nor is it desirable for dynamic obstacles [6].

These methods are concerned with creating smooth, continuous paths for the robot, which was not fully necessary for Dexter, due to his wagon-handle steering algorithm. As long as the path was reasonably continuous, the vehicle was capable of following the path fairly precisely. This slight inaccuracy in path following was taken into account in the path generation by enforcing overly strict safety requirements. Dexter wasn’t concerned with squeezing through extremely tight spaces (although it was demonstrated to handle such situations) and therefore a reasonable buffer was given for avoiding obstacles and the exact path is less important.

Another method of path planning in an obstacle field is via use of configuration space representation (Figures 2.4 and 2.5). Configuration space representation is a technique in which areas are analyzed based on where the object can be placed without colliding with the obstacles,
based on the object’s configuration. In essence, the object is replaced by a point and the obstacles are expanded to represent where that point would be allowed to go without the object hitting an obstacle [9].

Figure 2.4: (a) C-space representation of obstacle B for object A in a fixed orientation; (b) path for object A through an obstacle field with C-space representation. Reproduced from [9]

The calculations for determining C-space for objects with numerous degrees of freedom are complex. Simply adding one-axis of rotation to a block complicates the problem tremendously, as is the case in Figure 2.5.
Figure 2.5: C-space representation of an obstacle given three different ranges of rotation for object A. Reproduced from [9]

The C-space representation for the obstacle is quite different based on the orientation of the object. However, configuration space representation is used regularly in robotic manipulator arms, with several degrees of freedom, so Dexter could certainly utilize this technique.

Dexter presents an added challenge, as it is a nonholonomic vehicle. There are methods to convert a path for a holonomic vehicle into one suitable for nonholonomic use. First, paths are generated based on a collision-free requirement. These paths are then approximated by generating collision-free, but viable paths. The path chosen is selected based on the quality of the solution, dependent on the length and number of changes in direction [18]. Figure 2.6 illustrates two differing paths for a nonholonomic vehicle to go from point A to B. The first path involves an extreme number of changes in direction, wiggling along the entire path. The alternative is to back out from in between the block and drive normally through the narrow opening. The second option is far superior as it involves only a few changes in direction.

Figure 2.6: A holonomic path is generated (left), while a path feasible for nonholonomic vehicles is generated from that path (right). Reproduced from [18]

C-space techniques were considered for Dexter, but ultimately, obstacle avoidance requirements were not complicated enough to justify such a method. Far simpler avoidance techniques were sufficient.
2.1.3 Lead Vehicle Following

Vehicle following algorithms have been researched extensively for their prospective use in convoys, heavy traffic, and impending collision awareness. Solutions consist of vehicles communicating their GPS positions amongst each other, active sensors, predominantly LIDAR units, and passive sensors such as vision, all of which possess advantages over the others but also have shortcomings. The GPS solution, for example, requires a reliable GPS signal, which cannot be guaranteed, as well as consistent vehicle-to-vehicle communication. The majority of this previous work is done under the assumption of no a priori road information, requiring the robot to follow the exact path of the lead vehicle, not just matching speeds [20]. However, for this project, the road lanes are given in the RNDF, simplifying the problem tremendously. For vehicle following, all Dexter needs to do is provide a safe following distance. This merely requires knowing the vehicle’s speed and location, as well as Dexter’s. With these limitations, the sensor and tracking requirements are quite simple.

2.1.4 Observing Intersections

Prior to the 2007 Grand Challenge, analyzing intersection precedence remained relatively unnecessary. The research that has been done in the subject of road intersections includes recognition of roads at an intersection and impending collision awareness [21]. It wasn’t until the recent Urban Challenge that there was an impetus to explore proper vehicle etiquette at intersections. [21] explores software that can detect when a vehicle is violating intersection safety. This study analyzes the failure to stop at a stop sign or red light and explored the potential
for using radar units to detect oncoming traffic for potential collision awareness. The vehicle is equipped with a GPS system and a map data file. With this information, the software can monitor the vehicle as it approaches a stop point and if a failure to decelerate is detected, a HUD (heads up display) message is displayed to the driver or an automated braking assist is activated. Also, as a vehicle approaches an intersection, the incident angle to other roadways is determined, which would allow radar units to dynamically reposition to track oncoming traffic. However, this software merely analyzes the control vehicle for intersection violations. The driver must still make the cognitive decisions for precedence or safety to proceed in the face of oncoming traffic.
Chapter 3

Obstacle Tracking

The main objective of the obstacle tracker is to filter all detected objects into tracked obstacles. This is done through a number of processes. Raw range data collected from laser rangefinders (LIDAR), radar, and stereo cameras are preprocessed to distinct range points and into coherent objects. This work is performed in lower-level modules. All objects are created as distinct line segments and are treated as such throughout the AI. (Note: Sensor-generated obstacles will be referred to as detected objects and the tracker-based obstacles as tracked obstacles.) LIDAR was the primary means of detection, though radar was effective in some cases. Obstacles generated by stereo camera systems were ultimately unused, due to lack of development of the stereo camera module. The obstacle tracker possesses the ability to easily integrate stereo camera information, as well as any additional data that may be available in the future.
3.1 Filter Detected Objects

First, all detected objects are filtered based on location. If the object’s entirety lies outside of drivable regions of interest, it is ignored, illustrated by Figure 3.2. These regions of interest include the current travel lane, lanes adjacent to the current lane, intersection areas, and parking lots. Also, regardless of regions of interest, objects directly in front of Dexter are included, to ensure that if the robot is somehow off the road, it will still see obstacles that it can potentially hit. The purpose of this filtering process is to retain latency. In high obstacle areas such as tall grass, there can be as many as 80 detected objects passed to the obstacle tracker. Trying to track 80 objects is beyond the ability of the module. If the latency is slowed
significantly due to a multitude of objects, the performance of the tracker is reduced to near-uselessness. As the time-step between iteration steps increases, it causes the current detected objects to not match up with the tracked obstacles, which in turn creates more obstacles to be tracked, and thus further slowing down the tracker.

![Image](image_url)

*Figure 3.2: (a) the detected objects returned by the LIDAR Module with the road edges displayed; (b) the tracked obstacles after filtering objects outside drivable area*

### 3.2 Match Detected Objects to Tracked Obstacles

The next step is to match the new detected objects with the tracked obstacles from the previous iteration. This is done by comparing the locations of each line object and their angles with respect to the vehicle. If the lines segments are deemed similar enough, the detected object becomes linked to the tracked obstacle. Detected objects are compared with every tracked
obstacle, with a similarity value generated from an amalgam of the distance the lines are apart as well as the relative angles between the lines. The lowest value is chosen as the obstacle that matches the line segment. If the number is smaller than a certain cutoff value, the detected object is deemed to match the tracked obstacle. The obstacle’s new position is then updated based on the location of the matched detected objects.

for each detected object, calculate similarity values for each tracked obstacle

- similarity value equals a distance value plus an angle difference value
- distance value is the sum of distances from both endpoints to tracked obstacle
- angle difference value is the smallest angle formed by both line segments times a constant of .08

choose the tracked obstacle with the smallest similarity value
if that number is less than a certain cutoff, in this case 12, the detected object is linked to the tracked obstacle

Algorithm 3.1: Matching Detected Objects to closest Tracked Obstacle

A flaw with this method was detected. Assume two vehicles, facing opposite directions, are parked bumper to bumper. Viewed from the side, the obstacle tracker will think this as one obstacle, which is the correct decision to make. If the cars then begin driving forwards, the obstacle would not split into two separate obstacle, but remain as one. Both detected objects still match the tracked obstacle, so it will simply grow. So a step was added that checks all the detected objects that match a given tracked obstacle against each other. This way, if there are two objects that clearly don’t match each other, one is no longer matched with the obstacle. This way the tracked obstacle shrinks to the correct size and a new obstacle is made for the unmatched line. This is not accomplished easily however. An obstacle should not be split if it is comprised of a long line of small objects that are similar enough to match each other. The two furthest line segments most likely do not match each other, but working inward, they match other segments
that will eventually match each other, and therefore can be considered one obstacle. If only two line segments match each other, but do not match any other lines, they should be separated into a new obstacle.

If only two line segments match each other, but do not match any other lines, they should be separated into a new obstacle.

linked detected objects are compared with each other and grouped
if there are clearly separate objects that both match the current tracked obstacle,
the obstacle is split into two line segments and tracked separately
all remaining linked objects are merged and used to update the obstacle’s position and length

Algorithm 3.2: Analyzing all linked objects as a group

Detected objects that do not match any tracked obstacles must be compared with each other in order to create new tracked obstacles. The presence of numerous LIDAR units complicates this process. The LIDAR Module does not combine the output of each separate unit. Therefore, for each real-world obstacle, there can be as many as four resulting detected lines sent to the Obstacle Tracker. Currently, matching unmatched detected objects uses the same algorithm as matching objects with obstacles. This only finds the most similar line segment. As a result, if three or more LIDAR units see an obstacle, there are at least two tracked obstacles created in the first cycle. This can cause unreliable tracking. Steps were taken to minimize the effects of this, but a complete solution was not perfected.

An algorithm was written to solve this problem, but unfortunately could not be fully tested prior to the competition. This method creates a 2-D array that contains the similarity values for all detected objects to each other. Similarity values greater than the cutoff are voided and remaining values are used to process the detected objects into groups based on what lines are similar with each other, similar to the process above that determines if detected objects linked to a tracked object match each other. These groups are then merged into one line segment and
converted into a tracked obstacle. This was briefly tested to solve the problems presented by the original algorithm, but the code has a tendency to behave improperly. Further work in this area would likely be able to eliminate the problems, but there was no time before the competition to investigate further.

Algorithm 3.3: Prospective algorithm for creating new tracked obstacles

Tracked obstacles that were not matched to any detected objects begin decaying both in their estimated speed and their detection score. The detection score is a measure of how confident the Tracker is that there is actually an obstacle there. The detection score is limited to a high of 1000, if a score is lower than 100, it is still tracked but not reported as an obstacle. These values were chosen by Scott McMichael prior to this writer’s involvement with the Urban Challenge, but work adequately and were thus never checked for their validity nor changed. Obstacles with scores that decay below a value of three are deleted. Outside of parking lots, both the speed and detection scores are decayed by 5% per cycle, while within lots the speed is reduced by 25% since most obstacles within the parking area will be static obstacles. Thus, within lots, any slight velocity that might have been estimated for the obstacle is reduced extremely rapidly.

Finally, tracked obstacles were compared with each other to ensure that if the tracker assigns two obstacles to only one real-world obstacle, they will be merged into one. The
similarity check is far more stringent than previous checks, so the algorithm does not incorrectly merge obstacles as they may pass each other.

### 3.3 Estimate Obstacle Movement

The tracker predicts obstacle movements by estimating the speed and heading of each tracked obstacle. Each iteration calculates a speed and heading for each tracked obstacle, based on the distance traveled and acceleration at that time step and direction traveled, respectively. This is done via a rudimentary Kalman filter. There are two phases to this process, predict and update. Each tracked object’s current position is predicted from the previously calculated speeds and headings, based on iteration latency. Then, after the new detected objects are matched (or not) the speed and heading estimates are updated, based on the error of the predicted position from the actual recorded position. The equations for estimating the speed, acceleration, and heading, respectively, are as follows:

\[
E_D = \text{actual distance traveled} - \text{predicted distance traveled}
\]

\[
E_H = \text{actual heading} - \text{predicted heading}
\]

\[
S_i = \text{speed estimate for current iteration}
\]

\[
S_{i-1} = \text{speed estimate for previous iteration}
\]

\[
A_i = \text{acceleration estimate for current iteration}
\]

\[
A_{i-1} = \text{acceleration estimate for previous iteration}
\]

\[
H_i = \text{heading estimate for current iteration}
\]

\[
H_{i-1} = \text{heading estimate for previous iteration}
\]

\[
C_1 \text{ through } C_3 = \text{numeric constants}
\]
\begin{align*}
S_i &= (E_D)C_1 + A_{i-1}(t_i - t_{i-1}) + S_{i-1} \\
A_i &= (E_D)C_2 + A_{i-1} \\
H_i &= (E_H)C_3 + H_{i-1}
\end{align*}

Algorithm 3.4: Updating an obstacle’s speed and heading

In a true Kalman filter, the three constants would be self-adjusting so as to provide the most accurate blend of using the actual data or the predicted model. But this was unnecessarily complex, as using constants worked adequately. For this use, the following constants were chosen: C_1=1.5, C_2=3, C_3=0.05. These constants were chosen to maximize the responsiveness of the model without grossly impacting the smoothness of the estimate. If the constants are set to make the model extremely responsive, the tracking of a steady speed obstacle will be noisy. The error terms for distance traveled was limited to within ±.75 m so as to limit the amount that false data impacts the speed and acceleration estimates. The acceleration term was also limited to between -5 and 2 m/s². Otherwise the acceleration value would occasionally be infeasibly large.

Originally, the obstacles were tracked using accelerations and velocities in the global x, y frame. However, since the module is tracking nonholonomic automobiles, it proved advantageous to model obstacles using speed and heading instead.

One issue of import is to be able to accurately reduce obstacles’ speeds as they come to a stop. This proved difficult due to the need to filter jittery LIDAR readings. It was noticed that even completely motionless obstacles have a good amount of noise. During the competition, there was a programming bug that caused this noise to be offset as much as .5 m/s. Figure 3.3 shows a smaller offset, along with the noise of the estimate. Various alterations to the tracking algorithm were attempted, but if too much filtering and associated delay was introduced, response times grew unacceptably long. Converely, with high responsiveness, obstacles never
completely came to rest due to the LIDAR jitter. This noise also caused obstacle heading estimates to wander as the vehicle was at rest. The degree of wander can be seen in Figure 3.3. Therefore another solution was required.

![Figure 3.3: Speed and heading estimates of a row of barrels as Dexter is stationary. The horizontal axis is measured in module iterations (20 iterations = 1 second). The speed remains approximately 0.2 m/s while the heading estimate wanders greatly due to LIDAR noise.](image)

The solution was to analyze whether or not the iteration’s move was physically possible given the vehicle’s heading. Obviously, this can only be performed when the tracker has a good estimate of the heading. One of the difficulties with speed/heading models is that the initial heading of obstacles is unknown, particularly if first seen when stationary. Once the obstacle begins moving, the heading can change by up 180 degrees. Therefore it was impossible to estimate the heading of very slow or stationary obstacles. Slow moving obstacles contain enough
noise that the heading estimate will jitter, making a confident estimate not possible. Upon reaching a reasonable speed or distance traveled, obstacles were assessed to have a fairly confident estimate of their heading. With an accurate heading estimate, it was possible to filter the jitter of stopped vehicles. Due to the nonholonomic nature of cars, at slow speeds, any movement that deviated from that heading by a large enough amount can be disregarded as noise. If there is an abundance of noise relative to the number of legal movements, it can be assumed that the vehicle has stopped. This process was able to rapidly detect the stopping point of other robots, as well as quickly respond once the vehicle resumed motion.

obstacles are initially classified as having a moderate confidence
the heading estimates of obstacles with moderate confidences is allowed to react
directly with the current iterations heading estimate
once an obstacle reaches an estimated speed of 1 m/s (effectively begins moving),
the obstacle is assigned a “high” confidence
the heading for high confidence obstacles are modeled using the Kalman-filter
method above
illegal moves are defined as any time an obstacle moves in such a way that the
calculated heading change divided by the estimated speed is larger than a
specified value
an illegal move cause a counter to be incremented
the counter resets upon a legal move
if the counter reaches 3 while the speed is less than 2 m/s, the obstacle is assumed
to be stationary
the heading is locked to the previous value
the speed is set to zero
upon resuming movement, the first legal move releases the obstacle
obstacles with moderate confidence are not treated in such a way

Algorithm 3.5: Method for determining stationary vehicles

Upon writing this thesis, the bug was discovered. While tracking a stationary obstacle, any acceleration below zero caused the speed estimate to decrease below zero. This should not be possible and the speed estimate should be limited to greater than or equal to zero, but due to the incorrect placement of an absolute value operator this caused
the speed to be assigned not as zero but rather greater than zero. Upon fixing this error, the speed estimate tremendously, greatly reducing the noise as well as centering around zero. The noise magnitude was improved but still seen to be as large as 0.1 m/s for relatively good data. Also, the heading wander still persisted. It is not clear whether the removal of this bug would provide an adequate solution for detecting the stop point of dynamic vehicles and further testing of this fix would be necessary. Figure 3.4 shows the magnitude of the speed estimate noise.

![Figure 3.4: Example of obstacle jitter following bug fix](image)

Figure 3.4 illustrates the path of an observed vehicle as it performs two full stops. The vehicle makes a 90 degree right-hand turn in between stops, which is accurately tracked in Figure 3.5.
Figure 3.5: The path that the vehicle traveled, stopping at two stop signs and performing a 90 degree turn between stop signs. The vehicle was programmed to travel as fast as 8 m/s. Stop lines are denoted by a white line with a red dot. The blue line is Dexter’s route plan and green dots are GPS waypoints.
Figure 3.6: Tracker estimates of vehicle speed and heading as it decelerates, accelerates, and performs turns (20 cycles = 1 second). The highlighted areas are where the vehicle was stopped at each stop sign, respectively. The tracker responds very quickly to the vehicle as it comes to a stop. The heading estimate is excellent. The speed estimate is reasonable, reliably within 2 m/s.

As Figure 3.6 shows, the speed estimate has a tendency to accelerate and decelerate too rapidly, a problem that is quite noticeable approaching and leaving the first stop sign. As it approaches and slows, the vehicle’s estimated speed drops rapidly and then ramps back up slightly. Then, upon accelerating through the intersection, the speed clearly is estimated to be too high, followed by the speed estimate returning to a good approximation. However, this is acceptable behavior, due the rapid recognition of the point at which the vehicle has come to a complete stop and the point the vehicle resumes driving. Tuning the constants for more accurate
acceleration and deceleration would result in the tracker taking much longer to assess the stopping and starting point of vehicles at intersections.

3.4 Obstacle Memory

One of the hardest problems to solve was the issue of obstacles occluding other objects. This occurs frequently at intersections and in parking lots and can cause obstacles to shrink as they are eclipsed by others. Every tracked obstacle that either shrinks in length or isn’t matched with a detected object must be checked against every detected object for an occlusion. If such an occlusion is present, a partial occlusion must be treated differently than a full occlusion. In the case of partial occlusions, the tracked obstacle appears to shrink in length. The one endpoint of the obstacle that is visible must be matched with the appropriate endpoint of the detected object. The hidden endpoint must move so as to maintain the original length of the obstacle, since it is doubtful the obstacle has actually shrunk.

\[
\text{if updated length > 95\% of existing length,} \\
\text{check both endpoints and midpoint for occlusion by other objects} \\
\text{if one of these points are occluded, maintain original length but update position} \\
\text{if all points are occluded, decay speed estimate slightly} \\
\text{if all points are not occluded, limit length decrease to 95\% of original} \\
\text{in parking lots, rules regarding speed decay cause speed to decay rapidly} \\
\text{occluded obstacles’ detection score is not decayed} \\
\text{else update position to currently recorded}
\]

Algorithm 3.6: Occlusion rules
Figure 3.7: Example of obstacle memory at intersections. The lower left obstacle is occluded but the entire length is remembered

Obstacles that are fully blocked must continue moving at their predicted speed and heading. Also, the detection score tracked obstacles that are unseen usually decays until it reaches a threshold value at which point it is no longer reported as an obstacle. This detection value doesn’t decay for fully occluded obstacles. In parking lots, it is extremely important for the tracker to have a robust memory, which is illustrated in Figure 3.8. If an obstacle it can’t see is forgotten, it’s possible the robot could get stuck in a loop of exploring that area, seeing the obstacles, retreating, and then re-exploring the area again if the obstacle disappears. At intersections this is important because if an obstacle is forgotten when another vehicle passes in front of it, intersection precedence based on order of stopping would be forgotten with the obstacle and could cause Dexter to enter the intersection out of turn.
3.5 Results

The Obstacle Tracker performed quite well throughout testing and during the competition. Obstacle tracking was tested in the three NQE areas. Tracking moving vehicles at intersections was tested in Area A. Out of 17 turns, each with numerous vehicles passing by, the obstacle tracker successfully tracked dynamic vehicles every time. There was difficulty at one specific intersection (described in detail in Section 8.1) with distinguishing between a lengthy concrete barrier and vehicle’s passing directly in front of it. The detected object matching algorithm was inaccurately matching the vehicle’s LIDAR line segment to the barrier’s tracked
obstacle. The vehicle position of the vehicle was still safely seen within the lane, thus making it acceptable for the Intersection Observer, but it did cause some obstacle confusion at times.

Area B did not contain any dynamic obstacles, but Dexter was able to drive approximately ½ mile at 30 mph with a cement barrier directly abutting the right edge of the lane. Dexter was able to accurate track the barrier as a static obstacle slightly outside the lane even while traveling at a relatively high speed.

In Area C (detailed in Section 8.3), the Tracker had to maintain obstacles occluded during intersection monitoring. Out of 10 intersections, no obstacles were forgotten at any point when occluded. However, at one intersection, one vehicle’s position estimate was particularly noisy, which caused the obstacle to drift out of the lane when occluded, which reset the vehicle’s estimate stop time (more information in Section 7.2). Since Dexter never observed the vehicle moving, there was no accurate heading estimate, and obstacle jitter could not be filtered. Other than that one glitch, the Obstacle Tracker perform perfectly in Section C.

Overall, out of a scale from 1 to 5, with 1 being completely useless and 5 being perfect, the Obstacle Tracker module performed at approximately level 4, being satisfactory but prone to a few scattered errors.
Chapter 4

Lead Object Observer

The Lead Object Observer determines the viability of the current travel lane. It ascertains the drivable area to the left and right of each obstacle, and if passable, calculates the offset from the center of the lane that the robot must travel in order to safely pass each obstacle. If the obstacle is impassable, it returns the distance to the front of the obstacle. The module also returns obstacle information relevant to lead vehicle following and in-lane obstacle avoidance.

4.1 Passability Check

The observer is passed in a lane description generated by the Lane Observer, which in most circumstances is the current travel lane. The obstacle list is then sorted by proximity in front of Dexter and information needed for the passability check is generated for each obstacle. This includes finding the distance from the lane center for each endpoint, the nearest lane point
to each endpoint, and which endpoint comprised the front and rear edges of the obstacle, respectively. Obstacles spaced too close together for Dexter to snake through are grouped. Even though both obstacles separately may be passable, if the obstacles are too close together, there may be not enough room for Dexter to pass safely. Therefore grouping them treats them as one set of obstacles to pass through, rather than two separate obstacles and deviations from the center line. Also grouping them together ensures that Dexter will stop prior to the first obstacle and not entrap himself if the first obstacle is passable.

![Image showing obstacles and Dexter's path]

**Figure 4.1:** These obstacles are avoidable individually, but they are too close together along the lane for Dexter to safely pass through and therefore are considered a lead vehicle at 7.6 m with a speed of 0 m/s.

After grouping obstacles, the drivable areas to the left and right and in between obstacles are calculated. Slots smaller than 2.5 meters are ignored due to safety requirements. If passable,
the slot nearest to Dexter’s current center line is chosen as the drivable gap. The observer returns a deviation value, which is the necessary movement from the center of the lane that Dexter must travel in order to pass the obstacle. For each obstacle, this deviation is filtered over 50 iterations in order to eliminate noise as the robot approaches and passes the obstacle. Otherwise, as the vehicle passes the obstacle, due to noise in raw LIDAR data, as the obstacle jitters ever so slightly, the required deviation will be noisy and the steering angle of Dexter will wobble as a result.

**Algorithm 4.1**: Analyzing obstacles within the travel lane and performing the required calculations

Figure 4.2 illustrates how Dexter sees obstacles following the merging process described in Algorithm 4.1.
Passable objects along with their deviation data are now made available to other modules, published via datasocket. Impassible objects however are not immediately deemed impassible. Due to obstacle noise, particularly as Dexter is approaching obstacles, passible objects are somewhat frequently classified as impassible and vice versa. Therefore there is a counter that keeps track of how frequently a particular obstacle is deemed impassible. Every iteration as such caused the counter to be incremented, while if it is seen as passible, the counter is decremented. Once the counter reaches five, the obstacle is published to the Lead Vehicle Observer as an impassible object, along with data that the observer needs to do it calculations. The counter has upper and lower limits of ten and zero respectively, which allows the observer to accurately maintain obstacle types once they have been identified regardless of obstacle vibration.
for each obstacle, a counter is incremented each iteration the obstacle is deemed impassible and decremented otherwise
once the counter reaches 5, the obstacle is classified as impassible
the counter has a maximum value of ten and a minimum of zero
the information regarding impassible objects is sent to the lead vehicle observer

Algorithm 4.2: Handling impassible obstacles

4.2 Jam Lane Description

There are potential situations on a two-lane road where both lanes are not passable by themselves, but Dexter can still pass such obstacles. Figure 4.3 shows one such environment. This situation is resolved through a number of steps. Upon approaching a stationary object that is impassable in-lane, the Mood Selector provides this Observer with the lane description for the opposing lane. If the adjoining lane is also impassible, the mood Navigate Jam is started. This mood generates a lane description that encompasses the entire lane width. The Observer determines the road’s viability based on this extra-wide lane description. There were simpler approaches for the previous passability check, but due to the requirement to handle multiple (three or greater) lane descriptions at once, the algorithm was made more complicated.
Figure 4.3: Example of Jam condition and resulting generated path.

4.3 Results

During the competition and test runs, the Lane Object Observer accurately analyzed obstacles within the travel lane. This module only tested in Area C, when Dexter had to twice queue behind a vehicle at an intersection. The Observer correctly detected the vehicle as an
impassible obstacle within the lane, and Dexter safely queued behind it. During testing, Dexter
never incorrectly assessed an impassible obstacle as passable or vice versa, and passable
obstacles were successfully analyzed consistently. Overall, using the same performance scale as
in Section 3.5, the Lane Object Observer operated a level of 5, with no errors ever occurring
during testing or operation.
Chapter 5

Stay in Lane – Maintain Speed Behavior

The Stay in Lane – Maintain Speed module is a behavior which does nearly all of the path generation for Dexter. As the behavior’s name suggests, the main purpose of this module is to simply drive down the middle of a given lane description. However, this algorithm is extremely robust and is used for the final path generation in numerous other behaviors. The main features of the behavior is the ability to generate a path for a smooth return if Dexter is over two meters from the center of the desired lane, altering the path around tight corners to ensure Dexter stays within lane boundaries, and most importantly, in-lane obstacle avoidance.

To handle the shifting for the 2+ meter return to lane, tight corners, and obstacle avoidance, a universal shift breadcrumb trail algorithm was written. This allowed for easy shifting of the path. Based on which shift was occurring and on the distance required to shift, a length of shift integer is calculated, in addition to specifying which breadcrumb the shift should be centered around. With this information, the algorithm performs the shift on the specified
breadcrumbs as well as any subsequent breadcrumbs. This allowed for simple control of path shifting. Prior to this algorithm, the breadcrumb path was simply immediately shifted, which at certain speeds caused Dexter to oscillate around the trail as it tried to return to the path.

5.1 Return to Lane Algorithm

The Change Lanes behavior uses this module; changing lanes by slowly shifting the lane description, the rate of which is speed dependent. Navigating intersections is also done with this code. One advantage of this code, and the main reason Change Lanes utilizes it, is that if Dexter is more than 2 meters from the desired path, it generates a smooth transition back to the lane description, calculating a longer transition the further away Dexter is from the lane center. Without this, it was possible for the wagon-handle steering algorithm to cause oscillations upon returning to the lane description.

Algorithm 5.1: Shift Breadcrumbs algorithm

the distance from Dexter’s center to the desired path is calculated
the first breadcrumb in the trail begins at Dexter’s position
subsequent breadcrumbs are shifted towards the desired path until it is reached
Dexter returns to the path at a rate of 1 m towards the path for every 4 meters along the path
the return path is limited to between 4 and 20 m in length to ensure that Dexter can safely and successfully return to the path in a reasonable distance
5.2 Modifying Tight Turns

The behavior also has the ability to alter the path around tight turns. With the wagon-handle steering, the robot tends to cut tight corners. Upcoming turns are detected based on the change of heading between breadcrumbs over a short segment of the path. The headings between the first two crumbs in the segment and the last two crumbs are compared. If the difference in headings is large enough, the upcoming turn is recognized as tight enough to warrant modifying the path. This check is performed along the length of the breadcrumb trail. Upon recognizing that a tight corner is imminent, the breadcrumb trail preceding the corner is shifted towards the outside of the corner. As the breadcrumb trail exits the corner, the path is shifted back.
5.3 In-Lane Obstacle Avoidance

In-lane obstacle avoidance is also handled by this behavior. After obstacles in the travel lane are analyzed by the Lane Object Observer, they are read by this module. The required deviation, direction of shift, and the breadcrumbs that relate to the front and back of the obstacle are all required for obstacle avoidance. The Lane Object Observer provides this information. For each obstacle, the shifting of breadcrumbs begins four meters prior to the front edge and the breadcrumbs are shifted back to center two meters after the back edge. Each shift is performed over a distance of two meters along the lane.

Figure 5.2: Basic in-lane obstacle avoidance.
The behavior imposes a speed limit of 3 m/s for breadcrumbs six meters prior to and following the obstacle. The algorithm can handle having numerous obstacles in the travel lane, assuming the Lane Object Observer deems them passable.

Figure 5.3: Multiple in-lane obstacle avoidance

Figure 5.3 shows a path generated for avoiding three successive obstacle in Dexter’s travel lane. It is apparent, however, in the first image, that for the second obstacle in particular, it appears that Dexter plans to cut back too soon after passing the obstacle. This happens because the
LIDAR cannot see the side of the obstacle to ascertain its length along the lane. This cutback distance expands as Dexter performs the avoidance maneuver and the side of the obstacle comes into view, as can be seen in the second image.

5.4 Jam Behavior

Avoidance of obstacles while in the Navigate Jam mood uses the same algorithm as in-lane avoidance, just with an altered lane description. The major difference is the potential magnitudes of the required deviation from the normal path. In Navigate Jam, Dexter usually has to negotiate much further from the lane center. To ensure that Dexter is generally driving straight when he reaches the obstacle, rather than in the midst of performing turns, which would increase the area Dexter sweeps with his body, the shifting of breadcrumbs is done well in advance of the obstacle. This distance is dependent on how far Dexter has to deviate from the original path. The larger the deviation, the further from the obstacle the path is shifted, seen in Figure 5.4.
Figure 5.4: The path shifting here is done a large distance prior to and following the obstacle

In the NQE, there was a segment of road where there were vehicles parallel parked on both sides of the road. Unfortunately, Dexter never got a chance to navigate this segment, but this scenario was shown to work in simulation.
In this case (Figure 5.5), there was an added wrinkle. An obstacle was placed a short distance from the final parked vehicles, forcing Dexter to quickly cut back into his lane. Once Dexter’s desired travel lane is passable, the Navigate Jam mood is terminated, restarting the Follow Lane mood and resuming normal travel. The proximity of the final parked vehicle and the last obstacle makes this an extremely difficult scenario to navigate, but Dexter is able to reliably pass without collision.
5.5 Results

In-lane obstacle avoidance was never testing during the competition, so the usefulness of the behavior must be determined solely on test data, both simulated and genuine. For single obstacle avoidance, Dexter will never come into contact with the obstacle, but in the case of multiple avoidance, it sometimes occurred that Dexter, as he passed one obstacle, would prepare to pass the subsequent obstacle too early. As a result, occasionally the side of Dexter or the rear tire would bump the obstacle. However, snaking through a dense multiple obstacle area was outside the scope of the competition. Therefore, using the same scale described in Section 3.5, the Stay in Lane – Maintain Speed behavior operated at a level 4, as it was fully operable and effective, but there were occasional specific situations that could cause failure.
Chapter 6

Lead Vehicle Observer

The Lead Vehicle Observer is responsible for determining the speed and location of impassible objects in Dexter’s lane. This is done by finding the rate of change of the distance between the object and Dexter. The slope of the previous ten relative distances is analyzed in conjunction with the time elapsed to determine the vehicle’s speed. This number is filtered by a simple mean value filter in order to eliminate oscillations.

The algorithm simply keeps track of the distance between the vehicle and Dexter. The slope of the distances for the previous second is calculated, which provides the differential speed of the vehicle. Adding Dexter’s speed to this value gives an accurate estimate of the lead vehicle’s speed. This technique does have a drawback though. If Dexter is travelling much faster than a vehicle, it approaches rapidly and subsequently slows down behind the vehicle. Initially, the estimated speed will initially be too high, but as Dexter slows the estimation will climb. This
leads to oscillation of the estimate, which sometimes is unrecoverable. Adding a simple median filter smoothed the estimation adequately.

Knowing the distances between the vehicle and Dexter for the past 10 iterations, find the slope of these readings. This slope is the speed of the vehicle relative to Dexter. Dexter’s speed is added to this to give the vehicle’s absolute speed. To protect against oscillations and jumpy data, the speed of the vehicle is averaged with the previous ten estimates.

**Algorithm 6.1: Estimating a lead vehicle’s velocity**

![Graph](image)

**Figure 6.1:** A lead vehicle travelling at 4 m/s slowing to a stop. The white represents the distance between the vehicle and Dexter. The green represents the estimate for the vehicle’s speed. (Ten iterations equal one second)

It is apparent in Figure 6.1 that output from the LIDAR module and obstacle tracker is extremely noisy, which has a noticeable effect on the speed estimate once the vehicle has stopped. Near the end of the graph, the white line oscillates 2-3 meters. Given the range data, the observer does an excellent job estimating speed.
Figure 6.2: A vehicle accelerating from a stop to 6 m/s.

Figure 6.2 illustrates the results for a vehicle accelerating from a stop. The top speed of the vehicle of 6 m/s was slightly faster than Dexter’s top speed given in the MDF. Therefore, the vehicle slowly pulls away from Dexter, but the speed estimate remains accurate. The response time is quite small, as the speed estimate reaches the accurate value of 6 m/s quickly, with little overshoot or oscillation.

6.2 Results

The Lead Vehicle Observer was seldom used during the national qualifying event. Only during queuing did the Observer track an obstacle, and both times, they were static obstacles, requiring no actual tracking. However, there were no bugs of any sort during this, and Dexter safely queued behind the vehicles. During simulation and testing, Dexter was proven to exhibit
safe behavior for following vehicles. The Observer estimated the lead vehicle’s speed with a
degree of accuracy suitable for safe following behavior, and operated at a level of 5, using the
scale set forth in Section 3.5, as it no situation was encountered where it failed.
Chapter 7

Intersection Observer

The Intersection Observer sends the Mood Selector the “green light” for Dexter to begin negotiation of an intersection once it has deemed the intersection safe to traverse. Vehicles stopped at stop signs are analyzed for right of way and vehicles approaching without stop signs are analyzed to determine their rate of speed and whether Dexter can safely proceed. The Observer also needed to be able to deal with complex intersection geometries, such as adjoining intersections.

7.1 Intersection Geometry

Intersection geometry is retrieved from the Global Mapper, which identifies intersections based on data contained in the RNDF. The intersection observer makes a UDP request to the Global Mapper for the upcoming intersection geometry. Each intersection is broken up into
various regions, which are used to identify the state of traffic in each lane. Regions are generated based on intersection geometry. The region for approaching lanes extends 50 meters along the lane, using lane points every 5 meters to generate an enclosure that does a good job estimating the lane geometry. The same is done for lanes that exit from the intersection, except the region only extends 5 meters along the lane since distant vehicles exiting the intersection are irrelevant. The regions for approaching lanes are much longer in order to be able to track approaching vehicles if there is no stop sign for that lane. If there is a stop sign present for a given approaching lane, a box is generated at the point it reaches the intersection. This box extends in 2 meters in either direction along the lane and is used for determining when a vehicle has stopped at the intersection. Finally, a box is created that encompasses the center area of the intersection. This process is illustrated in Figure 7.1.
Figure 7.1: The various regions that the Intersection Observer creates the intersection in order to analyze precedence.

### 7.2 Four-way Stop Intersections

Intersection precedence where all lanes have stop signs is determined by order of arrival at each stop sign, respectively. The time of each vehicle’s stop is determined by when they come to rest while inside the small box near the stop line. This time of their right of way (r.o.w.) stop is stored in memory and compared with the time Dexter came to a stop at the intersection. When
Dexter’s recorded time is older than all other vehicles at the intersection it is his turn to proceed. The Observer is aware of and keeps track of vehicles that are queued in the approaching region behind other vehicles, but does not consider them having achieved a stop for determining right of way until they enter the small stop region.

*as Dexter approaches the intersection, all other lanes entering the intersection are watched*

*when vehicles come to a stop within their respective stop region, the current time is assigned to that vehicle as the time of its stop for right of way purposes*

*upon completing his own right of way stop, Dexter checks each intersection entrance for vehicles that have made a prior right of way stop*

*if there are no vehicles with earlier r.o.w. stops or in the intersection, Dexter is allowed to proceed through the intersection*

*else Dexter waits for the vehicle with precedence to proceed and exit the intersection before performing another precedence check*

*if a vehicle given precedence does not move for 30 seconds, that vehicle’s precedence is ignored*

**Algorithm 7.1: Intersection Precedence at 4-way stops**

Unfortunately, this method is extremely reliant on having correct GPS coordinates for the intersection geometry. If the stop boxes are in the incorrect location, Dexter may ignore vehicles that have precedence, or give precedence to obstacles that lie off of the road. This observer does have the ability to correct intersection GPS based on the stop location of Dexter. If the AI is running vision-based road and/or stop-line detection, Dexter will be in the correct position either laterally within the lane or along the lane, respectively. If the intersection position is incorrect, the observer can shift all other lanes and the intersection interior box to match where Dexter believes the correct intersection lies.

If the observer does give precedence incorrectly, there is a timeout of 30 seconds, after which Dexter will ignore the precedence for that given vehicle. For instance, if, at a four-way
intersection, there are stationary obstacles at the three other stop signs when Dexter arrives at the intersection, Dexter will wait 30 seconds for the vehicle he detected first to proceed. If at that point, the obstacle hasn’t moved, he will wait 30 seconds for the next obstacle, and so on. This must be done sequentially because if one of the obstacles is a vehicle, it might also wait thirty seconds for one of the stationary obstacle to move and then begin moving.

### 7.3 No-Stop Intersections

The most challenging decision at intersections is determining safe driving at intersections where approaching traffic does not have a stop sign. An accurate estimate of the speed has to be made quickly upon detecting the oncoming vehicle. Due to range limitations on the effectiveness of the LIDAR units, objects, particularly passenger cars, are only reliably seen within 40 meters. The proximity in which they are first detected makes it unlikely that Dexter should enter the intersection at any point that he sees an oncoming vehicle, unless that vehicle is travelling extremely slowly. Because of this limitation, Dexter’s driving behavior through intersections is far more aggressive than normal, increasing the rate at which he accelerates. The Challenge rules state that the robot must proceed when the intersection will be clear for at least ten seconds. As such, Dexter only enters the intersection if there are no approaching vehicles or the calculated time to reach the intersection for all vehicles is greater than ten seconds.

The speed of oncoming traffic is used to estimate the time until a vehicle will enter the intersection. If this time is large enough, Dexter can safely pass through the intersection. The speed estimate is generated by analyzing the rate at which the distance the vehicle is from the intersection decreases. The time to reach the intersection is then calculated using the speed
estimate and the distance the vehicle is currently from the intersection. The previous 30 measured distances to the intersection are fitted with a line, of which the slope is an estimate of the vehicle’s speed in meters/iteration. Multiplying by the number of iterations per second which is continually tracked based on the module’s latency provides the speed in meter per second. The observer runs at 20 Hz, at which rate 30 iterations is a delay of 1.5 seconds. This is an acceptable delay in order to get an accurate estimate of the speed. If this delay is shortened, the error in the initial speed estimate increases tremendously. If the speed estimate is too low, Dexter may think that there is plenty of time to proceed, when in actuality he might be cutting the vehicle off.

as Dexter approaches the intersection, all other lanes entering the intersection are watched
as soon as Dexter sees an approaching vehicle, the distance of the vehicle from the entrance to the intersection is monitored
once 30 measurements have been taken, the slope of these values is taken
this slope is multiplied by the number of iterations per second providing the initial speed estimate
the vehicle’s current distance to the intersection is divided by this speed to give the estimated time for the car to reach the intersection
if all approaching vehicles are estimated to take longer than 10 seconds to reach the intersection, Dexter is told to proceed through the intersection

Algorithm 7.2: Tracking vehicles at no-stop intersections
Figure 7.2: The calculations of a vehicle approaching an intersection travelling at approximately 5 m/s. The x-axis is in module iterations at 20 Hz.

It is apparent in Figure 7.2 that the estimation of the vehicle movement is excellent. The vehicle was likely travelling somewhat slower than 5 m/s, since in simulation the desired speed is merely what the vehicle attempts to obtain with no guarantee that it actually reaches that speed. Slightly noisy range data still results in an accurate speed estimation. Importantly, this estimation is accurate at the beginning of tracking. As a result of a good estimate of the speed, the calculated time to reach the intersection is also accurate, as it decreases at the same rate as the distance to
reach the intersection assuming constant speed. In the above example, upon seeing the obstacle first at 50 m, Dexter would wait to proceed until the vehicle has passed through the intersection. When the observer first estimates the vehicle’s time to reach the intersection, it is less than ten seconds, which proves to be an accurate estimation.

7.4 Results

Determining when it is safe to perform behaviors for navigating intersections is successfully accomplished by the Intersection Observer module. In the NQE, the Observer performed well at both four-way stop and moving traffic intersections. In Area A, Dexter negotiated through 17 intersections while the Intersection Observer behaved incorrectly at 2 intersections. At the first, it missed identifying an open slot as safe to proceed. This happened the second problem intersection, but then Dexter tried to enter the intersection when it was unsafe to go. Once he started moving though, safety precautions immediately stopped him prior to any collisions. It is not clear in the logged data why the Observer thought it was ok for Dexter to enter the intersection. The Intersection Observer handled 10 four-way stop intersections in Area C without a single failure. Using the scale set forth in Section 3.5, the Intersection Observer performed a level 4, with only sparse errors. In this case 2 out of 27 intersections were incorrectly handled.
Chapter 8

Conclusions

8.1 Results Summary

The best method to assess the work done in this thesis is to analyze its real-world success. Performance in the Nation Qualifying Event reveals whether the modules perform as intended. The three areas of the NQE taxed the various components and abilities of the software, and the modules in this thesis performed adequately.

Figure 8.1: RNDF of Area A
8.1.1 Area A

This test consisted predominantly of performing turns across one or two lanes of moving traffic. This is a test of sensor capabilities as well as decision making. Unfortunately, the sensors used by Dexter proved to be limited in their effectiveness in this situation. The LIDAR units had difficulty seeing the front of the traffic vehicles as they approached intersections. The LIDAR scan plane met the vehicle along its hood or windshield, both of which resulted in poor detection. Traffic was seen from the side up to 60 meters away, from the front, traffic regularly came as close as 20 meters before being seen by the LIDAR units. This made it extremely difficult to perform any analysis of the oncoming vehicles. This reduced the effectiveness of the observer to merely knowing that if it could see something coming then Dexter should not enter the intersection. Fortunately, this distance was large enough as to provide Dexter enough time to clear the intersection because he was in aggressive driving mode, greatly increasing his rate of acceleration through the intersection.

Figure 8.2: This is the point where the lower left obstacle, which is an oncoming vehicle, is detected, approximately 20 meters from Dexter. The orange line segments are obstacles while the green dots are GPS waypoints.

Figure 8.2 presents a problem with the track layout. The top obstacle is generated by a concrete barrier that is angled and abuts the road. Along the entire course, K-Rail barriers were
placed directly at the edge of the lane description. The majority of robots had difficulties with the proximity of these barriers. Dexter actually performed quite well in spite of these barriers. Figure 8.3 shows a barrier that is angled.

![Figure 8.3: The concrete barrier that is angled toward the intersection created a problem for Dexter’s intersection observer and decision making.](image)

This angle caused the Obstacle Tracker to have difficulties keeping the obstacle constant. The LIDAR Module creates line segments for each LIDAR unit separately. Due to the line of sight for the right side LIDAR, the line fit for that unit was incorrect.
Figure 8.4: The raw LIDAR range points for the environment.

The light blue points in Figure 8.4 are the data points for the right side LIDAR unit. The barrier past the angled barrier is only partially seen. This caused the line fit for that segment to be incorrect.

Figure 8.5 shows all the line fits for each LIDAR unit. The two objects to the left of Dexter are vehicles. The angled barrier is somewhat misrepresented. The purple and white lines are poor estimates of the environment. These fits result in the Obstacle Tracker having difficulty knowing the true position of the barrier. The end effect is that the obstacle jitters, a result of the purple and white lines occasionally being matched to the barrier obstacle. This motion caused the Intersection Observer to think that there was something moving in the intersection and therefore
unsafe to proceed until a timeout condition was reached. Dexter still did navigate the intersection, but after a lengthy delay.

Figure 8.5: The LIDAR fits for the data in Figure 8.4. Axis values are in GPS coordinates (latitude and longitude)

The other intersection in Area A had a concrete barrier running lengthwise along the lane the robot was performing a left turn into. Numerous vehicles had difficulty performing this turn smoothly, often stopping in the middle of the intersection, wary of a collision with the barrier. Others actually hit the barrier. Dexter handled this situation quite well, only once slowing slightly as it perceived the barrier as in the travel lane. The robot slowed as it prepared to pass the obstacle within the lane boundaries. The Stay in Lane – Maintain Speed behavior was written to decrease the vehicle’s speed as it approaches passable in-lane obstacles. As soon as Dexter
accurately realized that the barrier was not of concern, he correctly resumed normal travel speeds.

Figure 8.6: This intersection has K-rail barriers along the far lane which Dexter is supposed to turn left and enter.

In addition to intersection awareness, this section somewhat tested the Lane Object and Lead Vehicle Observers. Occasionally, Dexter was close enough behind a traffic vehicle to assess it as a lead vehicle, at which point it successfully began tracking its position and speed.

All in all, Dexter negotiated 17 turns across traffic, and there were issues in only two intersections. The main difficulty was the distance at which Dexter first saw oncoming vehicles. In conclusion, considering the environment, the Intersection Observer performed adequately, as did obstacle tracking. Obstacle detection was limited by the technology used by the robot, as well as a flaw in the LIDAR Module, which was not developed for this thesis. It was a major testament to the code that the intersection awareness was successful, having never been tested with real world data prior to the qualifying event.
8.1.2 Area B

Area B did not present nearly as large of a challenge to these modules as the other areas. This test consisted of lane following and parking lot tests. Lane following was complicated by the presence of the same concrete barriers abutting the lanes as in Section A. Dexter was able to accurately track and realize that the barriers were not in his travel lane while travelling at speeds up to 30 mph. Many teams had a great deal of trouble simply driving down a two-lane road with
barriers. This was a testament to the robustness of the Obstacle Tracker and Lane Object Observer.

8.1.3 Area C

Figure 8.8: RNDF of Area C

This area consisted of four-way stop intersections, testing the robot’s ability to determine intersection precedence. There was also a roadblock that required the robot to replan its route and perform a u-turn. Dexter handled intersection precedence well, going out of turn only once in 10 trips through the intersection, which was the result of an inaccuracy in the Obstacle Tracker. Again, the presence of obstacles near the road edges added to the difficulty of keeping track of vehicle precedence, which caused the one misjudgment. Other than that, every component worked as desired. The Obstacle Tracker retained vehicles as other cars blocked the LIDAR units’ view of the vehicle, illustrated in Figures 8.9 and 8.10.
Figure 8.9: The vehicle in the intersection occludes the stopped vehicle, but the obstacle remains tracked, allowing accurate precedence monitoring.

Figure 8.10: Close up of the Intersection Observer regions in the above situation. The stopped vehicle is accurately seen as in the stopped region as the other vehicle passes in front. The boxes are those used in intersection precedence awareness explained in Section 7.1. The line segments are the obstacles.

Although Dexter eventually drove through the roadblock, it was handled correctly from an observer standpoint. The Lane Object Observer accurately saw that the travel lane was impassible. The Mood Selector then requested the adjacent lane to be checked for passability.
The observer again correctly returned that that lane was impassible. With both lanes separately impassible, the Navigate Jam mood requested that the road as a single lane be analyzed. Again, the Lane Object Observer returned that the situation was impassible. Subsequently, due to a bug in the Mood Selector, Dexter chose to perform an incorrect maneuver.

![Figure 8.11: The roadblock present in Area C](image)

8.2 Future Work

Although the work done for this thesis performed adequately during the NQE, there is certainly further research that could be done for this subject. In fact, advances in technology might render this work useless.

8.2.1 Obstacle Tracking
Some other robot vehicles used more advanced sensors than Dexter’s. The most common advanced sensor was the Velodyne laser scanner that scans all three dimensions rather than a single plane. This sensor could create a point cloud of all obstacles around Dexter. Naturally, if this sensor was used, the requirement of the tracker would have increased substantially. There would have been a larger number of obstacles to track, ground data would have to be filtered out, and obstacles couldn’t be simplified into simple two dimensional lines. The entire module would be much more complicated and advanced.

As for improvements that could be made to the current software, the line of sight calculations for determining full and partial occlusions is a lengthy process and only checks certain points along the obstacle, such as the endpoints and midpoint. If there is an extremely long obstacle, such as the cement barriers in the NQE, the tracker does not realize that a vehicle is occluding just a small part of the long obstacle behind it. Figure 8.12 shows this situation.

![Figure 8.12: Failure of occlusion detection of a small part of a long obstacle. The long obstacle is a K-rail occluded by a vehicle in the travel lane. Since the vehicle does not occlude the midpoint or either endpoint, the K-rail is broken into two segments.](image)

Also, the algorithm used checks for line of sight against every other obstacle. This consumes a large amount of computational effort and a better solution should be investigated.
As stated earlier, the method used to match and merge detected objects with tracked obstacles should be enhanced. The current solution works well for most situations, but it consumes a large amount of resources and fails occasionally. Possibly an algorithm that looks at the relative speeds of the sensor data as it is seen could prevent incorrect matching of dynamic vehicles with static obstacles or each other.

8.2.2 Obstacle Avoidance

The algorithms used for obstacle avoidance were adequate for the requirements of this project. That being said, the techniques utilized were not particularly advanced or fool-proof. A Configuration-space representation could do a far superior job of determining passability and generating a path through an obstacle field of increased difficulty. For this project, the demands of obstacle avoidance are limited in scope, not needing to navigate complex obstacle arrangements within the travel lane. However, for this technology to perform in an unmonitored environment, more advanced obstacle avoidance algorithms would be required. C-space representation would be a possible solution to handle advanced avoidance requirements.

8.2.3 Lead Vehicle Following

Following a lead vehicle was a rudimentary problem, simply requiring an estimate of the vehicle’s position and speed. In this regard, no further work would need to be done. The current software does an adequate job, providing information accurate enough to safely follow a vehicle.
8.2.4 Observing Intersections

The Intersection Observer worked, but is reliant on accurate *a priori* intersection geometry. If there is a slight error in the positions of lanes, the module can fail. Research to develop methods of determining location of intersection and lane boundaries based on visual information or sweeps of the area by downward facing LIDAR would be helpful in ensuring an accurate assumption of lane positioning. If Dexter’s stopping position was reliably accurate even with incorrect *a priori* geometry, the module could accurately reposition the geometry, but if the lanes are inaccurate and not just the location, the current software would not perform well.
Bibliography


