Building an Autonomous Ground Traffic System

Tarun Nimmagadda

April 20th, 2007

Abstract

Ground traffic systems are the combination of ground vehicles, roads, highways and intersection infrastructure that make it possible for people to commute on the road. Recent advances in the field of robotics allow us to increase the autonomy of currently human-operated ground traffic systems. This document presents the key software components developed as a part of this thesis research to create an autonomous vehicle platform; specifically, a lane-based obstacle tracking system which enabled our autonomous vehicle, Marvin, to successfully handle dynamic traffic. Furthermore, the Autonomous Intersection Management (AIM) protocol designed by Dresner and Stone [7] shows that intersection efficiency can be significantly increased if the intersection can communicate wirelessly with autonomous vehicles. Towards this end, this document presents the extensions made to Marvin's capabilities so that it would communicate with the AIM system thereby bringing us closer to an Autonomous Ground Traffic System.

Figure 1:
Marvin, the autonomous vehicle in California during the 2007 DARPA Urban Grand Challenge.

1 Introduction

It is obvious from industry and government investments and even entertainment media, that we envision autonomous vehicles to have a substantial role in the future of transportation. What is less obvious is what needs to happen to convert our existing technologies to be autonomous and the intermediate steps along the way.

The creation of large scale automobile manufacturing by Henry Ford in 1914 and the standardization of the National Highway System in 1920 are the origins of today's Ground Traffic Systems. American highways which were once the source of quick and reliable travel are now a source of frustration due to congestion, and loss of life [1]. A report from the National Highway Traffic Safety Administration says that nearly 6 million police-reported motor vehicle crashes occurred in the United States in 2006 [5]. A total of 42,642 people lost their lives and 2.6 million people were injured. The 2007 Urban Mobility Report [10] shows that congestion in urban areas caused Americans to travel 4.2 billion hours more and to purchase an extra 2.9 billion gallons of fuel costing them over 78 billion. There is good reason to believe that autonomous vehicles combined with intelligent traffic infrastructure could save thousands of lives and also provide large savings in time and money.

1.1 History

Autonomous vehicle research has been an active field since 1995, when CMU's Navlab project achieved 98.2% semi-autonomous driving on a 3000 mile trip across America. There has been a surge since 2002, when DARPA announced the Grand Challenge. The first DARPA Grand Challenge started in 2004 as a 150 mile autonomous race through the desert. DARPA's goal was to develop technology that will keep soldiers off the battlefield and out of harm's way. The challenge is conducted in support of the congressional mandate to transition to one-third of the United States' ground military vehicles being autonomous by 2015. The 2004 and 2005 Challenges featured a 150 mile course in the Mojave Desert. While none of the teams made it past 8 miles in 2004, the 2005 Grand Challenge course was completed by five teams with Stanford University's team finishing first [15]. Austin Robot Technology (A.R.T.) also participated in the 2005 race and reached the semi-finals.

The DARPA Urban Grand Challenge was held in 2007 in an attempt to have vehicles race against each other in a mock "urban" city. This challenge required cars to obey traffic laws while navigating traffic circles and intersections containing professional drivers and other robot cars. Unlike in the previous challenges, where the robots did not encounter other vehicles on the course, the Urban Challenge required robots to interact closely with other vehicles. Carnegie Mellon's Tartan racing team won the race and five other teams completed the course with four of these teams finishing within the 6 hour deadline. This goes to show that cars possessing the capability to navigate traffic autonomously is no longer just a dream.

1.2 Background

The University of Texas did not participate in the two previous Grand Challenges, but partnered with A.R.T for the 2007 Grand Challenge. Since our collaboration began in August 2006, we have acquired and installed the Velodyne High Definition Lidar (HDL) Sensor and the Applanix GPS System on the vehicle. Dr Peter Stone's course at the University of Texas focused on introducing undergraduate students to the challenge of autonomous driving. This partnership provided an great platform to create unique undergraduate research opportunities. Our team made it to the National Qualifying Event (semi-finals) again in 2007, but was not among the top eleven teams chosen for the final race. This was in a large part because we did not have enough time to make our system robust by the competition deadline. Our car, Marvin, performed quite well until a self-inflicted accident put us out of the race. In particular, the lane-based obstacle tracking approach presented in this article showed very promising results in real-world tests. In one test that required robots to merge into moving traffic, Marvin made 7 laps of the course and was one of the few top performers on that test. We now have a stable platform for future development and improvements.

This article describes the software architecture we built for the grand challenge, while focusing on my contributions to the project. The second half of the article is a description of our ongoing effort to use the AIM system to create a complete Autonomous Ground Traffic System.The Autonomous Intersection project is an ongoing project at the University of Texas for over 3 years. Dresner and Stone's research on implementing the proposed Intersection Management Protocol on a custom simulator has been widely published [7]. The success of this approach in the simulator suggests that the next logical step is to implement the protocol on real hardware. Because we have only one autonomous car, we are creating a 'mixed-reality' demonstration where one of the cars approaching the intersection is Marvin and the the rest of the cars are virtual (simulated). If the project is succesful, it will be the starting point for a research effort to implement the AIM system at a General Motors (GM) facility in Detroit with 10 autonomous cars.

The contributions of this research are the following: A Lane-based obstacle tracking approach, MapLanes Manager to process and filter large amounts of laser data, Hermite splines to closely approximate roads, the Mission and Graph modules, and visualVelodyne to display range data in a easily navigable 3D scene.

2 Vehicle Hardware

The Austin Robot Technology (ART) vehicle, Marvin, is a 1999 Isuzu VehiCross that was selected for its ability to negotiate off-road terrain in the 2005 Grand Challenge. It has been outfitted with a custom-built reinforced front and rear bumper brackets to protect it against collisions and facilitate mounting of front and rear facing Lidar devices. It has been upgraded by the ART team to achieve drive-by-wire and now has alternators to power the onboard systems. They added shift-by-wire, steering, and braking actuators to the vehicle. Control of the throttle was achieved by interfacing with the vehicle's existing cruise control system. The car had a red "Manual Override" button on the custom center console that allowed the operator to engage and disengage the car controls for use by a human driver. All of the hardware work done on the car including the installation of our sensors was done by the ART Hardware team.

The computing power on the car consists of 3 machines containing AMD Opteron processors and 6600 GT NVIDIA graphics cards. Each machine is shock-mounted in a custom-designed computer rack. The two on-board alternators provide power to the computers, actuators, the siren, and the various sensors. The vehicle is equipped with a variety of sensors. The Applanix POS-LV device provides sub-meter odometry information by combining information gathered using Differential GPS, an inertial measurement unit (IMU), and the Distance Measurement Indicator (wheel rotation information). The information from the various devices is combined internally using sophisticated filters [12], in order to provide accurate position, orientation, and velocity of the car. This unit is shock-mounted alongside the computers on the rack in the back of the car. The ART team built a custom roof rack for the Velodyne Lidar and the GPS receiver. The roof rack being the highest point of the vehicle provided the best visibility of the terrain and clear reception of GPS signals.

2.1 Lidar Devices

A laser range finder works by measuring the time of flight of laser pulses. It transmits a narrow laser beam in a given direction, and records the time taken for the pulse to return back to it. This information is then used to find the distance to the closest object in that direction. Lidar systems (Laser Radar) deflect the laser beam using an internal rotating mirror in order to create a fan-shaped scan of the surroundings. On Marvin, we use SICK LMS Lidars which are know for their ruggedness and ability to work in adverse weather conditions. These devices are also used in hundreds of mobile robot frameworks around the world. We use two Sick lidars for precise, accurate sensing in front of and behind the vehicle.

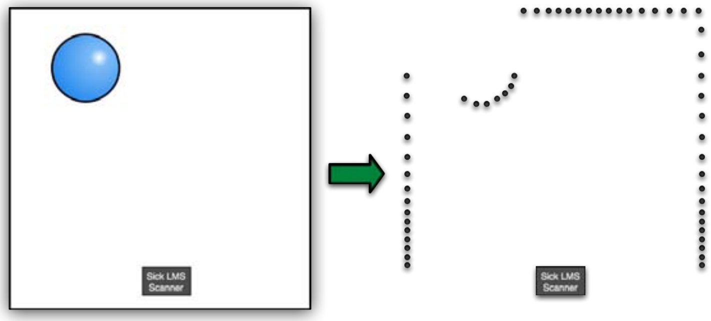
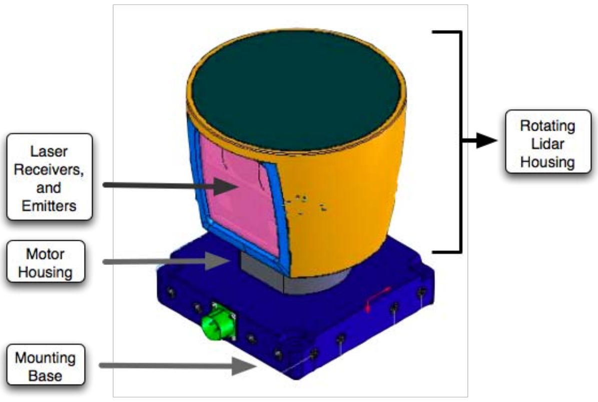


Figure 2: *This figure explains how a simple laser range finder like the Sick LMS perceives the world. The scanner provides the distance, and the corresponding angle to each of the points in the picture above.*

Lidar devices can also return intensity information (color) of the object that the laser beam hit. However, Lidar devices are a different class of devices than cameras. Cameras are considered passive sensors because they are designed to detect naturally occurring energy. Unlike passive sensors that can only work when naturally occurring energy is available, Active sensor technologies such as Lidar provide their own source of energy to illuminate their target. This allows them to work at all times of the day independent of lighting conditions created by the sun. Shadows and low-light conditions are known problems with using video cameras for sensing which do not affect the operation of Lidar devices.

2.2 Velodyne HD Lidar



The Velodyne HDL is a unique device that has 360 Horizontal Field of View (FOV) and a 26.8* Vertical FOV. It produces range information to one million points a second, which is three orders of magnitude more information than the SICK Lidars. The device internally utilizes 64 fixed-mounted lasers each mounted at a specific vertical angle. The entire unit spins at a configurable rate between 5-15Hz while each laser fires thousands of times per revolution, producing range information with very impressive density [4]. The device scans objects that are up to 120 meters away. Five out of six finishing teams at the Urban Challenge had purchased Velodyne units which they used extensively.

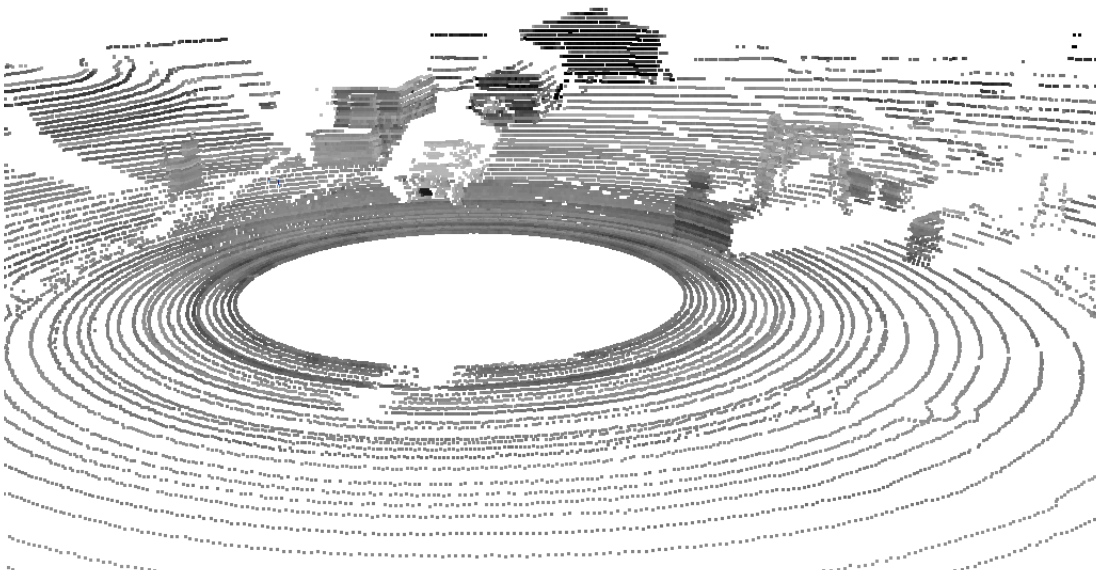


Figure 3: *The above image was produced using the VisualVelodyne tool developed for visualizing the 3D range data produced by the Velodyne HD Lidar*

3 Vehicle Software Architecture

The first and perhaps the most important step to creating a complete robotic system is the choice of the architecture. Any architecture must impose beneficial and reasonable constraints on the design, and interfaces without being overly restrictive. The design considerations for a robot system are quite different from other software applications. The environment around the robot is very dynamic, and the sensors are always noisy and uncertain. A good architecture should provide reasonable frameworks for the following:

- Designing complex high-level behaviors. In order to manage the complexity, an architecture should provide mechanisms for task decomposition to design modular components.

- Faciliate concurrent execution of behaviors, drivers.
- Creating a real-time system that is capable of processing large amounts of sensor data, and acting on it within fractions of a second.
- Testing and valdiation of individual modules independent of the entire system.

Our architecture is very similar to the 3T architecture proposed by the Jet Propulsion Laboratory (JPL) [6]. However, it is hard to determine which specific architectural paradigm was used in our system, as we also incorporate principles of Brooks' Subsumption architecture within the Navigator module. [cite Intelligence without Representation] The actual implementation of our architecture uses a open-source robotics software framework called Player/Stage.

3.1 Player/Stage

Player is a robot device server that facilitates communication between sensors, client programs / drivers, and actuators. Player allows you to write device drivers for each sensor and actuator that is part of the system. Player uses IPC internally to allow drivers to communicate with each other. Message passing is handled using incoming and outgoing message queues for each driver. Player uses a publish/subscribe system that allows each driver to publish information that multiple other devices can subscribe to. This mechanism allows for the publisher to keep sending the data even if one of the non-critical subscribers (such as a visualization utility) crashes. This paradigm also allows for the creation of 'abstract' drivers that decompose complex higher level behaviors into more manageable modules. We have created several of the modules that make up our world model using abstract player drivers. As described above, Player provides reasonable and beneficial constraints on our design without being overly restrictive. It benefits us by allowing us to build a highly concurrent and modular program. It allows us to easily test our large complex system by turning on and off drivers, and creating simpling placeholder drivers.

Perhaps the biggest reason to choose player as a starting point was because ART's existing system from the previous Grand Challenge attempt contained player device-drivers for all the existing actuators, and some of the sensors that were installed on Marvin. Beyond a small amount of 'tuning' conducted by some Team Members, we used the existing drivers unchanged.

Stage is a simulator of a 2D environment that is built to work with Player. It simulates devices like 2D laser scanners fairly well. It has allowed us to rapidly develop and test higher level behaviors that work without any changes on the real vehicle. Testing our software on Stage was a vital part of our development cycle. Testing software on the car is a very time consuming process, and Stage allowed us to develop and test large portions of our code without having to access the real hardware. Stage allows you to create configurable devices that emulate real hardware quite closely. In order to get the most leverage out of stage, we created an odometry device that simulated the dynamics of the car using the real wheel-base, acceleration and steering capabilities of Marvin. We also added artificial noise to the output of the laser scanner device, thus forcing us to design our controllers to have a higher degree of robustness.

3.2 Objective

The DARPA Urban Grand Challenge required cars to use the provided Route Network Definition File (RNDF), and then execute the Mission as specified in the Mission Data File (MDF) while following all the driving rules specified in the California Drivers Handbook for human drivers. The RNDF specified a relatively description of all the accessible roads where the vehicle is allowed to travel, including information such as the latitude and longitude of waypoints, the locations of stop signs, parking lots, lane widths, and lane markings. It also specifies the locations and the names of all the checkpoints. The RNDF does not specify a starting and ending point, or the locations of road blockages. DARPA also provided a MDF file that specifies an ordered list of checkpoints to reach, and the speed limits for various sections of the course. [2] [3]

3.3 World Model

3.3.1 MissionGraph

The MissionGraph module consists of two separate pieces: A rich Graph data structure and a Mission data structure. The very first step in the execution of our software is to parse the provided RNDF file, and create a graph to represent it. Waypoints are represented by nodes, and the connections between these waypoints are represented by edges. Nodes and edges also store additional information provided in the RNDF such as the type of lane markings to their left and right and whether or not a node is at a stop line. Other information, such as the speed limits, is parsed from the MDF and associated with the corresponding edges. The optimal graph search algorithm A* (A Star) was used to develop a path planning algorithm upon this data structure[1]. This planner produces the quickest path from the current location of Marvin through all the required checkpoints.

The Mission data structure is created from the MDF file and represented using a queue of the checkpoints to be reached. In order to be robust to a massive system failure that shuts down the entire software system, the Mission module logs the progress made to a file on disk at regular intervals. If and when the program is restarted, the Mission module automatically recovers the progress made and allows Commander to plan a path through the remaining checkpoints instead of starting over from the beginning.

Building the Mission planning in a modular fashion allowed us to test it independently from the rest of the software system. RNDF Parser, the MDF

[1]A team member, Mickey Ristroph developed this module

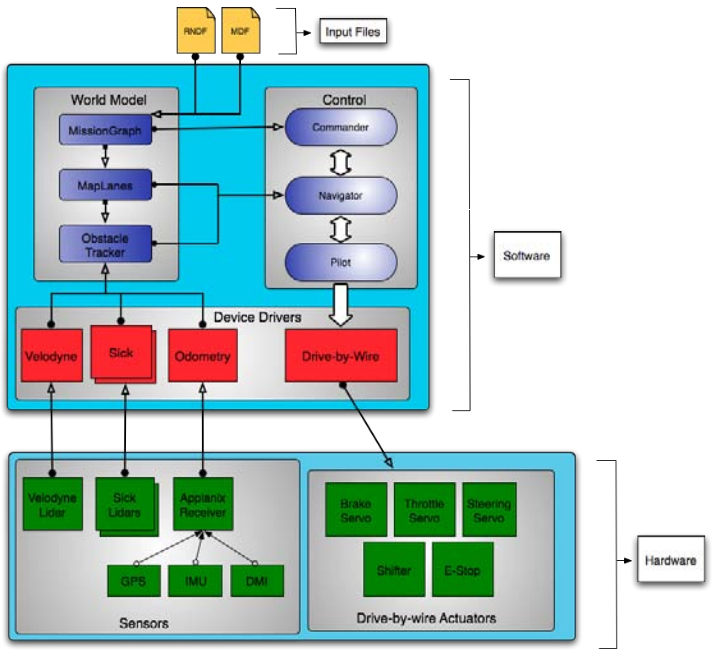


Figure 4: *Software Architecture: We use the Player [9] robot server as our interface infrastructure. Hardware devices are shown in green at the bottom. Device drivers are shown in red in the middle of the diagram. Commander (planning), Navigator(behaviors), and Pilot(low-level actuator control)—are shown on the right side.*

Parser were tested rigorously on several hundred positive and negative samples of RNDF and MDF files.

3.3.2 MapLanes

A core module of our World Model that allowed us to build our navigation behaviors, and our obstacle tracking module is a new map data representation that we call MapLanes. The MapLanes representation partitions space using a set of convex quadrilaterals. Each quadrilateral is constructed such that its left and right edges form the boundaries of the lane. A lane is partitioned into several such quadrilaterals that share their top and bottom edges with their neighbors. MapLanes uses the Graph created by the MissionGraph module and creates a lane map in the global Cartesian coordinate system.

The MapLanes data structure was originally created[2] to represent the loca-

[2]A student on the team, David Li, was responsible for this concept and the original implementation of MapLanes. Patrick Beeson is responsible for the current implementation of MapLanes

tions of the lanes in a manner that is useful for building navigation behaviors, such as 'Lane Following'. It was then extended to make it possible to use this data structure to filter laser range data, and build Obstacle tracking algorithms.

To summarize, the purpose of MapLanes is to

1. Represent lane information in a form that is useful for vehicle navigation.
2. Provide a way of classifying range data as being in the current lane, in an adjacent lane, or off the road entirely.
3. Provide a data structure that is more suited for building obstacle tracking algortihms than the OccupancyGrid.

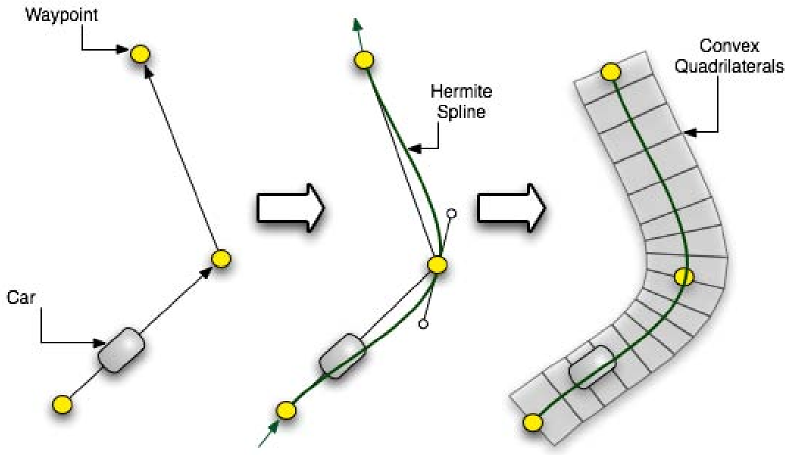


Figure 5: *Guessing the road shape: Given waypoints that define a lane, a cubic spline is used to generate a rough approximation of the road. We utilize a few non-standard heuristics to detect straight portions of roadway which improve the spline tangents at each point. The curves are used to generate the quadrilaterals, the collection of which are called MapLanes*

The MapLanes road generation algorithm[3] uses standard cubic splines [1], augmented with a few heuristics about roadways, to connect the RNDF waypoints (Figure 4). We first create a C1 continuous Hermite spline from the discrete series of waypoints that define a lane in the RNDF. We chose the Hermite form because its representation allows us to control the tangents at the curve end points. We can then specify the derivatives at the waypoints in such a way that the spline that we create from these curves has the continuity properties we desire.

[3]Justin Hilburn and I worked together to create an implementation of smoothly interpolating Hermite splines

We then convert the spline from a Hermite basis to the Bézier basis. This step allows us to use any of a large number of algorithms available to evaluate Bézier curves. At this time, we express the curve in terms of nth degree Bernstein polynomials which are defined explicitly by:

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i} \qquad i = 0, \ldots, n.$$

Any point on the curve can be evaluated by:

$$b^n(t) = \sum_{j=0}^{n} b_j B_j^n(t).$$

We set $n = 3$. The coefficients b_j are the Bézier control points.

This spline, along with the lane widths defined in the RNDF, gives the vehicle an initial guess at the shape of the roadway (see Figures 5, 12). Each lane is then broken into adjacent quadrilaterals that together tile the road. This system was designed originally to allow the vision sub-system to refine the shapes of these quadrilaterals and provide a more accurate depiction of the road, but we did not manage to accomplish this in time for the Urban Grand Challenge.

MapLanes is an alternative data structure to the OccupancyGrid for tasks such as Urban Driving. The OccupancyGrid is commonly used in robotics applications to provide spatial and temporal filtering of range information obtained by a laser range finder. The OccupancyGrid allows you to fuse the range data gathered by several different lasers into a single coherent map with very little computational overhead. A disadvantage of using the OccupancyGrid to store and process this information is that it partitions space into grid-aligned square cells which do not allow you to represent lanes that are used for driving closely (see fig) especially when the lane has a curve. MapLanes provides a very different alternative to partitioning space by using a partitioning that closely reflects/resembles the structure of the road.

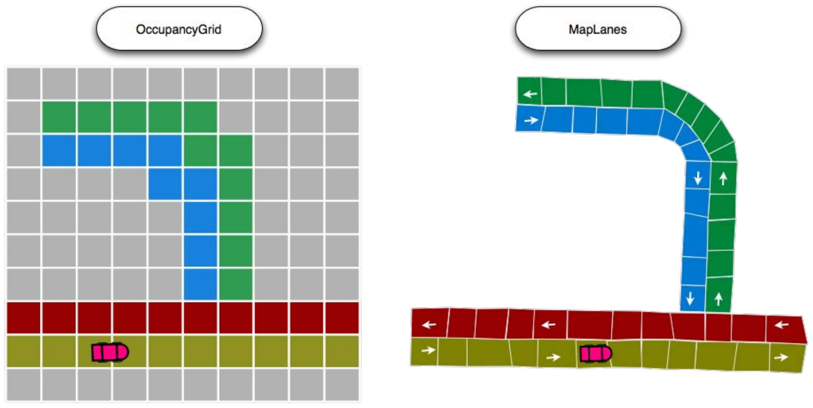


Figure 6: *Pictured on the left is the OccupancyGrid representation and on the right is the MapLanes representation*

3.3.3 Polygon Operations

Each quadrilateral within MapLanes is placed into a data structure that contains some important information such as the type of lane markings on the boundaries of the polygon, and also cached information about the quadrilateral such as the midpoint, length, width, and heading of the polygon. We created a polygon library that provides numerous methods for extracting information from the ordered list of polygons. This library performs most of the computation pertaining to the current state of the world surrounding the vehicle. Examples include: filtering out range data not on the road, determining distances along curvy lanes, and determining which lanes will be crossed when passing through an intersection.

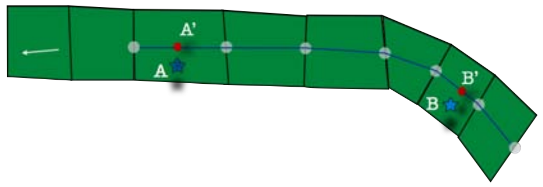


Figure 7: *The distance function in this space is not Euclidean but rather an approximation of distance along the lane. The distance computation between two points first projects each point onto the mid-line of the lane being tracked. In this figure, A', B' are the closest points on the midline to A, B respectively. The lane distance is computed by using the summation of piecewise line segments between A' and B' that form the midline of lane.*

3.3.4 MapLanes Manager

Any world model that does not consume and interpret the sensor data quickly enough is not feasible for use in Urban Driving. One of the biggest advantages of the OccupancyGrid representation is that it is a very efficient data-structure to store and retrieve data. Some commonly used functions (defined on the OccupancyGrid) are computationally efficient and easy to write. For example, consider the 'Containment Operator' that calculates which cell contains a particular Point (x, y). This operator is often used to determine the cell that the end point of a laser return is located in. This operator is used extensively in the filtering of Velodyne data and in the Obstacle Tracker. This function is trivially simple, and O(1) for an OccupancyGrid with a fixed grid resolution. The naive approach, for MapLanes requires you to iterate through all quadrilaterals in the region. This is a O(n) function, which is a disaster considering that the Velodyne produces one million data points in a second. The initial implementation proved to be too slow to meet the required real-time constraints.

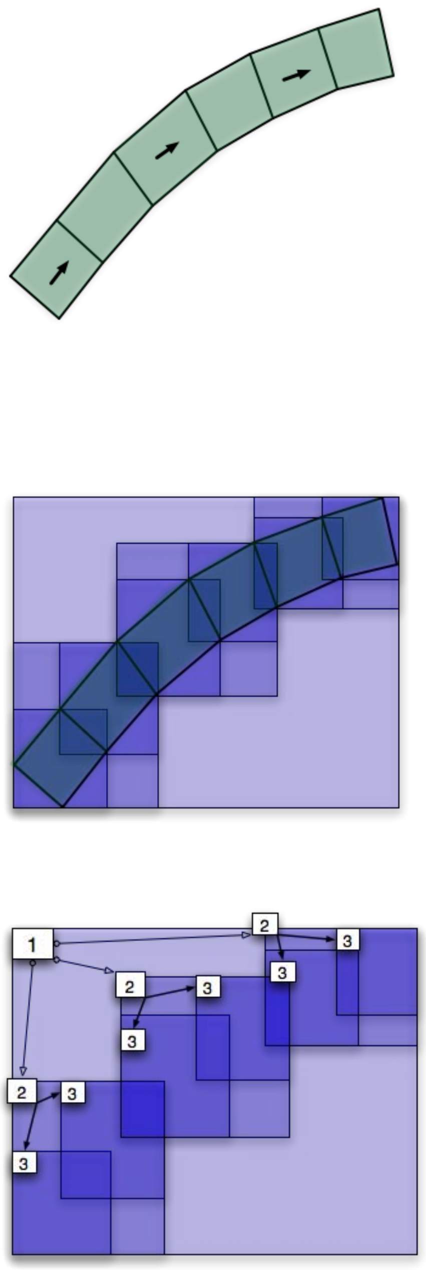


Figure 8: *This sequence of images illustrates the MapLanes structure of a road, the hierarchical bounding boxes that can be constructed corresponding to them, and finally the tree of bounding boxes.*

In order to increase the speed of this operator, we used a data structure that consisted of hierarchical tree of bounding boxes for the MapLanes quadrilaterals. This data structure was inspired by KD trees and resembles a simplified form of them. The MapLanes quadrilaterals change infrequently (1 Hz) in sharp contrast to the high data-rate of the Velodyne. So, we created a hierarchical structure that pre-computes the axis-aligned bounding boxes for each quadrilateral, and all the quadrilaterals in a lane, and so forth. This change allowed us to improve the complexity of the containment operator to $O(log(n))$ and gave us an 6.5x speedup in practice. As most of the data produced by the Velodyne was outside the lanes, this allowed us to greatly reduce the amount of data that we processed.

MapLanes manager also caches several computations that are common to multiple users of the MapLanes data structure. All of these optimizations were driven by the very real need to process all our sensor data in real-time.

3.3.5 Velodyne Processing / Filtering

The Velodyne High Definition Lidar (HDL) provides around one million points of data every second. Most robotics platforms across the world that were created prior to the availability of the Velodyne used lidar devices such as the Sick LMS. The Velodyne produces 3 orders more magnitude more data than the Sick, and so, the data produced by it cannot be processed with the same techniques. Instead of using computationally intensive 3D modeling techniques [16], we use simple "height-difference" maps to identify vertical surfaces in the environment. With every complete set of 360° data, we create a 2D (x, y) grid map from the 3D point cloud, recording the maximum and minimum z (vertical) values seen in each grid cell [4].

Next, a simple ray-casting algorithm casts rays from the sensor origin to calculate the closest obstacle in each direction. A cell in the 2D grid is said to have an obstacle is the difference between the maximum and minimum heights is above a certain threshold. This data is used to create a simulated 2D lidar scan which looks very similar to the data output by the Sick lidar devices (see Figure 9);

[4]Patrick Beeson was responsible for the design and implementation of this module.

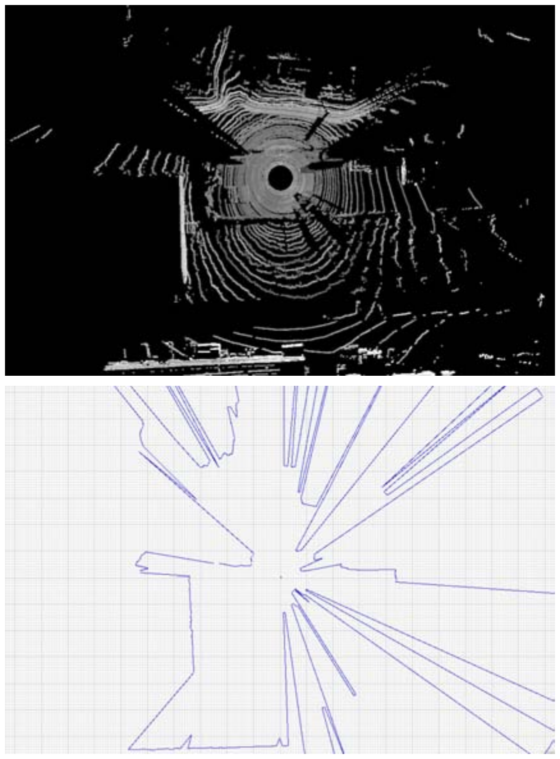


Figure 9: *Raw Velodyne HDL point cloud (bird's eye view of Figure 3 is shown) gets processed into a 2D scan. Notice corresponding features between the two data formulations. This method creates occlusions, but allows for fast processing of the million points per second that the Velodyne HDL transmits.*

Even with the very basic processing technique describe above, we found that the design of this module was not meeting its real-time constraints, and was introducing delays that propagated through the entire system. We experimented with various alternatives to speed-up the processing such as decreasing the 'resolution' of the grid and creating a multi-resolution grid to trade off accuracy for speed. But we found that we were still not processing the Velodyne data as fast as we wanted. Finally, We were able to use MapLanes representation to speed-up our processing. The MapLanes Manager component that I developed used the pre-computed bounding-box tree to provide a quick lookup of whether a point was outside of all lanes. By not considering the points thats were outside the lanes, we greatly reduced the delays in processing laser data.

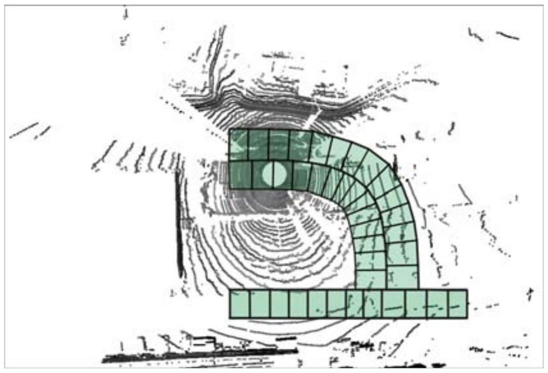


Figure 10: *MapLanes are overlayed on an overhead view of the point cloud data provided by the Velodyne. The car is located on one of the lanes, at the center of the several concentric range-scans.*

3.3.6 Obstacle Tracking

For autonomous driving, a robot needs good approximations of the locations and velocities of surrounding traffic. Recent approaches to obstacle tracking have often utilized a Cartesian-based occupancy grid [8] for spatial and temporal filtering of obstacles. This occupancy grid is used to estimate the state of the obstacle

$$S = \left(X, \dot{X}\right), X = (x, y, \theta), \dot{X} = \left(\dot{x}, \dot{y}, \dot{\theta}\right)$$

S is the state of the obstacle, $X = (x, y, \theta)$ represents the position and orientation of the obstacle, and $\dot{X} = (\dot{x}, \dot{y}, \dot{\theta})$ represents the rate of change in the state of the obstacle.

Many of these approaches also attempt sometimes the shape or extent of surrounding obstacles [13]. These approaches typically tend to be difficult to implement, and get slower as more vehicles need to be tracked.

Our design differs from omni-directional tracking in that we utilize the MapLanes model of the roadway to solve the obstacle tracking problem. The key insight in simplifying the problem of obstacle tracking in the urban driving domain is that the vehicle only needs to track obstacles that are within lanes.[5] We further reduce the dimensionality of the problem by observing that it is sufficient to track the velocity of each obstacle only along the obstacle's lane.

By partitioning the world into lanes and defining an order on the quadrilaterals comprising each lane, we impose a linearization on the space. Thus we can easily track the distance to the closest obstacle in each direction of a lane and the velocity of the obstacle along the lane. This approach is inspired by the way some humans make decisions about driving. When driving 80 mph along a highway it might easily appear that an oncoming car in the neighboring lane is

[5]The 2007 Urban Challenge specifically ruled out pedestrians or any other obstacles that might move into traffic from off the roadway.

about to collide with you with even a minor change in its heading. Despite the presence of imminent danger, a human driver, having mentally tracked cars in neighboring lanes, will continue to drive calmly; he will reason that the other car will stay in its lane. This assumption, based on a human driver's own obstacle tracking, enables that human driver to drive at great speeds without having to be extremely cautious and watchful.

The innovation in the Lane-Based Obstacle Tracking approach is that we are tracking a simplified aspect of each obstacle's state:

$$S = \left(d, \dot{d}\right)$$

d and $\dot{d}$ represent the distance and velocity of the obstacle along the lane. The basic operator in the lane-based obstacle tracking system is:

$$f\left(lane, point, direction\right) \rightarrow S$$

For each lane, a point on the lane and direction along the lane, we build an obstacle tracker that track the distance to the closest obstacle in the specified lane and direction. The obstacle tracker for each lane receives a laser scan which specifies the positions of all obstacles that are within its lane. It then iterates through all these obstacles to find the closest obstacle in the specified direction. It maintains a history of these observations and the time at which they were recorded using a queue of fixed size. We then filter out noise using known acceleration and velocity bounds for cars and estimate the relative velocity from the queue of recent observations. A tracker can then combine the velocity and distance information to calculate 't' the time to collision. In our system, we choose the closest point along the midline of each lane and create a lane-based obstacle tracker in each direction (forwards and backwards) along that lane. We then compare the time to collision reported by each tracker and provide the details of the more imminent collision in each lane.

Figure 13 illustrates an experiment where our vehicle was sitting still and tracking another vehicle driving in an adjacent lane.

One of the advantages of this obstacle tracking approach is that it is lane-centric instead of being obstacle-centric. This means that it scales linearly with respect to the number of lanes that our vehicle attends to, not the number of obstacles. This is a desirable property in Urban driving where we are often surrounded by several fast moving cars that are within the range of our sensors. It also allows us to easily configure the amount of historical data we take into consideration by changing the size of the distance queue. We found that even though it had a simplified model of the world, it was powerful enough to complete the various requirements that DARPA outlined in the Technical Evaluation Criteria, and therefore sufficient for most urban driving tasks. During the NQE event we used a 10 frame queue of distances which reduced the risk of inaccurate measurements from sensor noise, but introduced a lag of about 1 second in the measurements given the 10Hz lidar updates (see Figure 14). We accepted this lag as a trade-off for the Urban Grand Challenge in favor of robust, safe driving over aggressive behavior. If we find and use more robust approaches to estimate velocity the queue of distances, we can decrease the size of the queue and decrease the latency in our decision making.

3.3.7 Lane Observers

The second portion of the perception necessary for multi-agent interaction is a set of *Lane Observers*, each of which instantiates an obstacle tracker on the appropriate set of nearby lanes, and reports the situation to the Navigator control module. A Lane Observer focuses on a set of lanes and lidar range data to track the most imminent collision in those lanes. Think of an observer as a back-seat driver in charge of reporting whether a specific behavior is safe or unsafe, based on the presence of dynamic obstacles. Based on its current plan/state, Navigator chooses which observers to pay attention to. The primary information each observer provides is a single bit, which represents whether the behavior is is safe or unsafe. The observer also provides useful quantitative data (such as estimated time to the most imminent collision) if the behavior is executed.

Our system uses six key observers: Nearest Forward, Nearest Backward, Adjacent Left, Adjacent Right, Merging, and Intersection Precedence. Some of the important ones are described below:

Nearest Forward, Nearest Backward Observers These observers are concerned with the lane that Marvin is currently on [6]. They report to Navigator the most imminent collision in front of, and behind the vehicle respectively. If NearestForward reports that there is a car in front of Marvin in the current lane moving towards it, Navigator will respond by pulling over (Evading) in accordance to the CA Defensive Driving rules.

Adjacent Right Observer The Adjacent Right observer reports whether the lane immediately to the right of the vehicle's current lane is safe to enter, for example to pass a stopped vehicle. If there is a vehicle in the right lane but the time to collision is larger than 10 seconds[7], the lane is considered to be clear. Unlike the Nearest Forward and Nearest Backward observers, this observer incorporates two trackers, one directed front and one behind. It compares the time to collision backwards and forwards in the right lane and reports the most imminent threat to Navigator.

Merging Observer The Merging observer is used to check when it is safe for the vehicle to driving through an intersection or turning into/ merging across traffic. Like the Adjacent Left observer, it combines information from several Obstacle Trackers. In merging scenarios, the observer checks whether all lanes

[6]These observers track the *nearest* lane if Marvin is is not on any lane (off the road)

[7]This is a configurable value in our system. Many such values were specified in the DARPA Rule book for the event

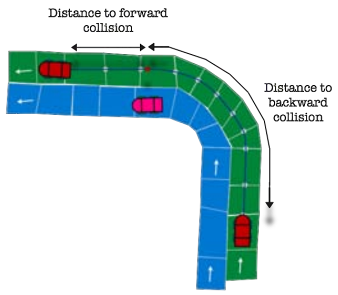


Figure 11: *This diagram shows Marvin in pink, with two cars in the lane to the right of Marvin*

the vehicle *can possibly* traverse are safe[8]. For example, at an intersection, if it is unsafe to turn left the vehicle will wait, even if it plans to turn right. The reason for this is that the Observers are not aware of Navigator's plan and which direction the car is about to turn. Our architecture can be extended to make this possible, but we decided against this because the net effect of the current design is to make Marvin *more* conservative than required. As expected, in the context of the Grand Challenge, this decision did not impact us negatively.

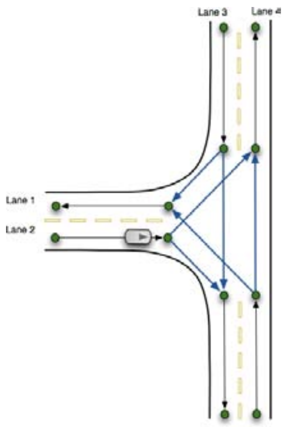


Figure 12: *This diagram illustrates how the Merging observer determines which lanes to track at an intersection. All the lanes that the outgoing transitions overlap are considered.*

[8]Bartley Gillian, a student on the team was responsible for developing the function to determine which lanes to track

3.3.8 Obstacle Tracking Results

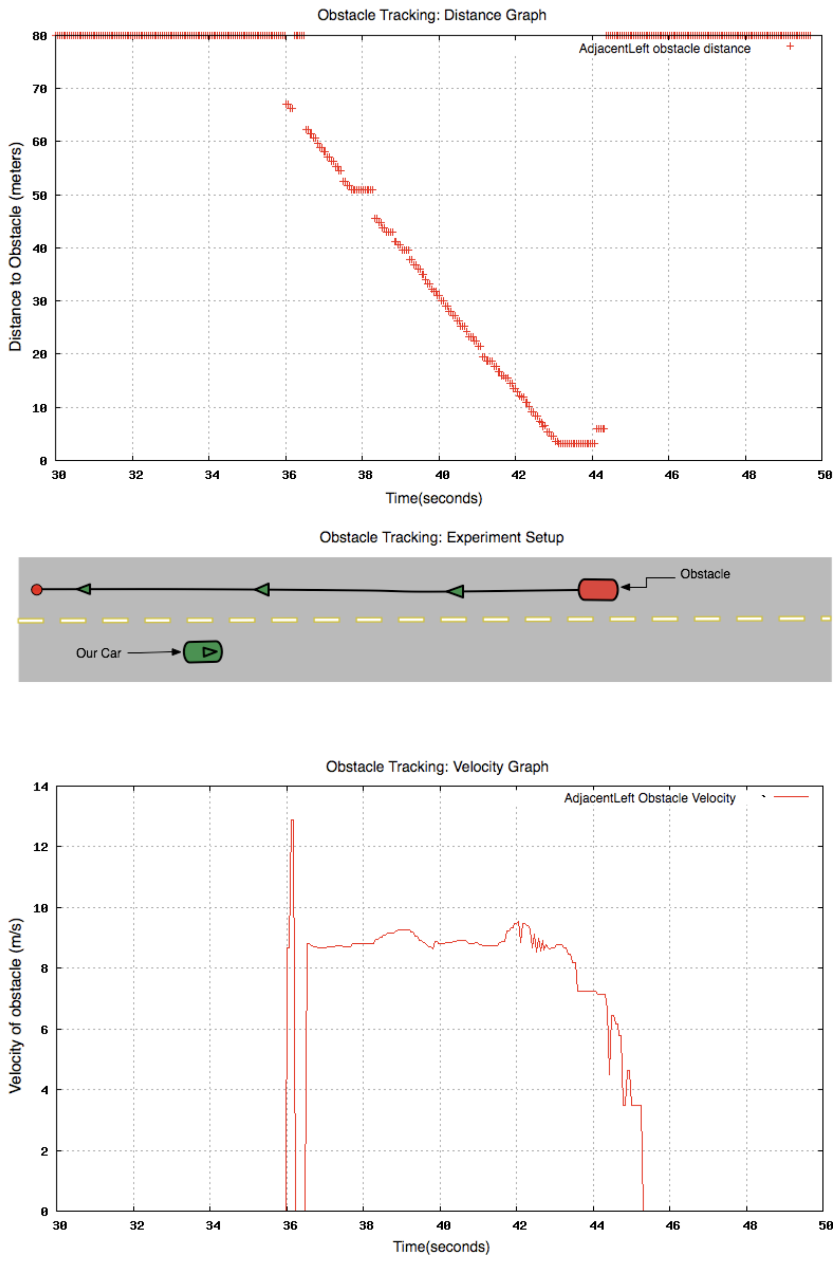


Figure 13: *In this experiment, our vehicle was stopped on a road and tracking another vehicle in the lane to the left. The driver of the tracked vehicle reported an estimated speed of 9 m/s. Other than a brief initial transient, the obstacle tracker accurately models the oncoming vehicle starting from about 60 meters away.*

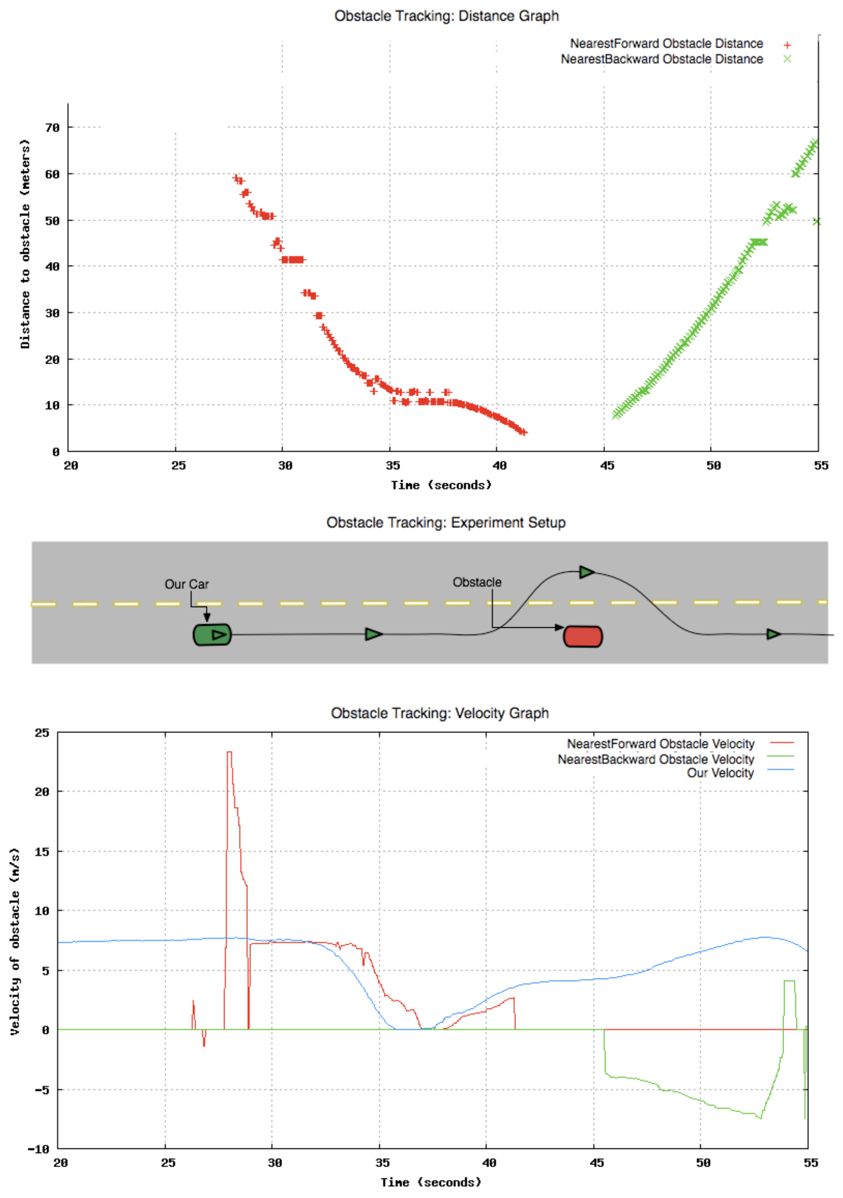


Figure 14: *In this setup, our vehicle drove up to another vehicle that was stopped in the same lane. Our vehicle then passed before returning to the original lane of travel. The above graphs show the relative state of the obstacle estimated by two trackers: one forward and one behind in the current lane.*

4 Control

Our strategy was to decompose the levels of reasoning involved in controlling Marvin into High-level Planning (Commander), A state-machine of various behaviors (Navigator) and low level control execution (Pilot). This structure largely resembles the 3T architecture.

4.0.9 Commander

Commander uses the Graph and the Mission components of the WorldModel, and A* search to find the shortest path through the various checkpoints that the car is required to reach. Commander provides navigator with a route through the next few waypoints that the car should follow, and frequently updates this route based on the progress that Navigator has made, and the feedback it receives. For instance, it is responsible for planning a new route when Navigator reports that the road ahead is blocked.

4.0.10 Navigator

The Navigator module is a hierarchical state machine of various behaviors. such as Lane following, U-turn, Change Lane, etc. It runs the appropriate behavior based on the route provided by Commander.

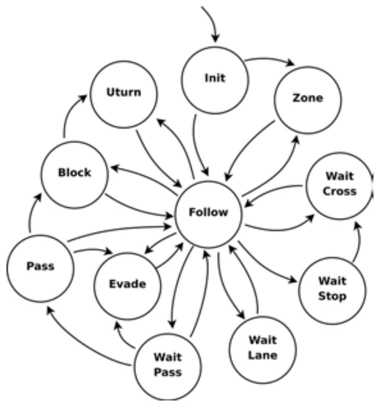


Figure 15: *A simplified illustration of the vehicle's Run state machine. It uses the information provided by the Lane observers to make decisions regarding traffic. For instance, the Adjacent Right observer communicates to Navigator whether it is safe to merge into the right lane*

Each behavior computes a desired travel and turning velocity based on the information provided by the world model. The *Pilot* module transforms the requested velocity and heading commands into the low-level throttle, brake pressure, and steering angle commands to the respective actuators.

The description of the Autonomous Vehicle system above makes many references to the rules specified by DARPA for the Urban Grand Challenge. However, the provided rules were largely derived from the California Drivers Handbook, and reflect most of the capabilities necessary for any urban autonomous vehicle platform.

5 Autonomous Intersection Management

The Autonomous Intersection Management system developed by Dresner and Stone provides a protocol for autonomous vehicles to communicate with the intersection.To improve the throughput and effeciency of the system, vehicles 'call ahead' to the intersection and request a reservation. The intersection manager determines whether or not these requests can be met. Depending on the decision that the intersection manager makes the vehicle must either record the parameters of the *reservation* and attempts to meet them, or make another request at a later time.

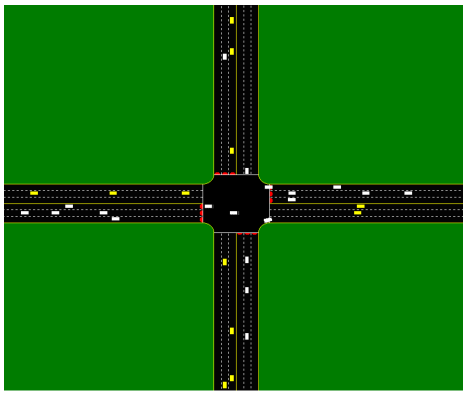


Figure 16: *This is a screenshot of the AIM Simulator developed by Dresner and Stone. Multiple autonomous vehicles communicate with the intersection and move through it without colliding with each other*

To determine whether or not a request can be met, the reservation manager simulates the journey of the vehicle across the intersection, which it divides into a grid of n × n tiles. The parameter n is called the granularity of the reservation manager. At each time step of the simulation, it determines which tiles the vehicle occupies. If throughout this simulation, no required tile is occupied by another vehicle (from a previous reservation), the manager reserves the tiles for this vehicle.

Dresner and Stone evaluated the performance of the reservation system against alternatives such as an overpass, or an traffic light. Using the simulator, they showed that using the reservation-based policy, vehicles crossing an intersection experience much lower delay than the traffic light. Furthermore, they showed that the reservation-based policy also drastically increases the throughput of the intersection. The details of the protocol are explained in their paper . I will focus here on the driver agent, as this is what I have implemented on the actual vehicle.

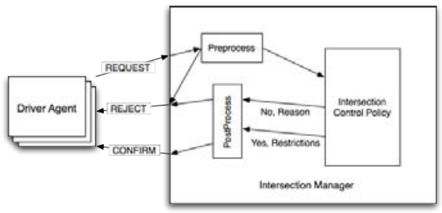


Figure 17: *This diagram explains the communication between the car (driver agent) and the Autonomous Intersection Manager*

The original AIM Simulator worked as a single monolithic program that allowed the cars and the intersection to communicate through function calls. To extend this system to work in the real world, the cars need to be able to work in a decentralized system where they communicate with the intersection over network communication. I extended the capabilities of the original communication system by creating a framework to serialize and de-serialize messages and communicate with the intersection using UDP Communication. The AIM protocol is designed to handle packet loss, but is sensitive to delays in communication. I chose UDP over TCP because it has lower overhead, and has the right characteristics to enable us to scale this system to many more cars in the future.

Digital Short Range Communications (DSRC) [11] is a wireless protocol designed for Vehicle to Vehicle (V2V) and Vehicle to Infrastructure communications. It is designed to provide high data transfer rates over medium range distances. DSRC is a UDP based communication protocol, and using UDP now allows us to transition to using the DSRC protocol in the future.

5.1 Driver Agent

When a vehicle approaches the intersection, the vehicle's driver agent transmits a reservation request, which includes parameters such as time of arrival, velocity of arrival, as well as vehicle characteristics like size and acceleration/deceleration capabilities, to the intersection manager. If the requested reservation is deemed safe, the intersection manager responds to the driver agent with a message indicating the reservation has been accepted. Otherwise, the intersection manager sends a message indicating that the reservation request has been rejected, possibly including the grounds for rejection. The driver agent may not pilot the vehicle into the intersection without a reservation.

In order to be able to effectively use the Intersection Manager, a driver agent must be accurately capable of estimating the time and velocity at which it will reach the intersection accurately. As discussed previously in this paper, the output of the obstacle tracking module and in particular the 'Nearest Forward Obstacle Tracker' that I developed for Marvin is sufficient to reasonably estimate the time to arrival.

5.2 Mixed Reality Demonstration

Testing the Autonomous Intersection Management system on real hardware obviously requires more than one car. Because we are currently limited to only one autonomous vehicle platform, we created a Mixed Reality demonstration that allows us to test the system using one autonomous car, and several virtual (or simulated) cars. In our setup, we ran the Autonomous Intersection Manager software on a wireless-enabled laptop that was placed inside Marvin for convenience. As Marvin drove up to an intersection with a stop line, he sent "Request Messages" over a specified UDP port to the AIM Laptop. These messages specify the estimated time of arrival, and the lane into which Marvin is going to turn. The AIM Laptop also simulates other vehicles that communicate with the AIM, which grants a reservation to Marvin based on this 'virtual traffic'. Marvin obtains a reservation and then drives through the intersection based on the "Confirmation Message" it receives from the AIM Laptop. We repeated this test successfully a few times. Videos of the demonstration are available at the author's website [14]

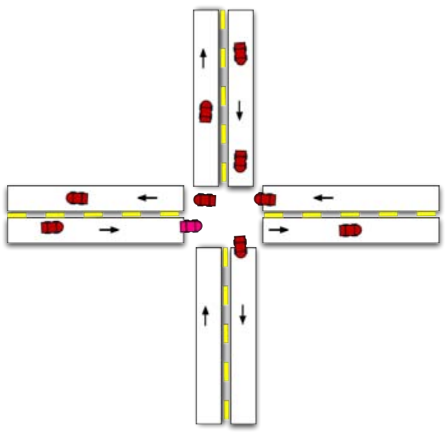


Figure 18: *This illustrates the 'Mixed Reality' Demonstration, where the cars in red are simulated and the car in pink is Marvin*

Future work on this system will analyze the performance of the AIM system in a real-world system through further experiments. Some important issues that are yet to be simulated relate to actual performance of commodity wireless hardware when we transition to having the AIM system physically located at the intersection.

6 Summary and Future Work

In summary, this thesis represents almost 2 years of research conducted in collaboration with A.R.T, and several students at the University of Texas. As presented throughout the document, the research resulted in many contributions to the overall project. The 3 main contributions were:

- Lane-Based Obstacle Tracking (Section 3.3.6)
- MapLanes Manager (Section 3.3.4)
- Mixed-Reality for AIM (Section 5.2)

In addition, several other significant contributions were made along the way:

- Hermite Curves to provide a smooth representation of lanes.
- RNDF, MDF Parser
- Mission, Graph components of the World Model.
- VisualVelodyne: A tool to playback the point cloud produced by the Velodyne in a easily navigable 3d scene.
- VisualCommander: A tool to visually inspect RNDF files.

Despite the significant progress made over the course of this research, there are still many fruitful directions for future work. In particular, the migration of the AIM system of real hardware will involve transitioning from the Mixed-Reality demonstration to incorporate multiple autonomous vehicles. In order to do this successfully, we need to evaluate the performance of the AIM system in the presence of networking issues like low data rates, packet-loss, etc. Another interesting direction for further research is to explore the possibility of equipping Marvin with the ability to transition from human control, to autonomous control while traveling through an intersection.

6.1 Acknowledgements

Many people were involved the creation of the hardware and software platform for Marvin. Several students from the Spring 2007 Autonomous Driving Class contributed substantially to the overall code base. Several members of ART contributed their time towards keeping the hardware components running smoothly. Jack O' Quinn, a volunteer from ART maintained the Automake based build system, the SVN repository, and also wrote a large part of Navigator. Patrick Beeson, the only Graduate level member of the team, contributed significantly to almost all aspects of the overall architecture.

References

[1] 2010 and Beyond: A Vision of America's Transportation Future, 2004.

[2] DARPA Urban Challenge technical evaluation criteria, 2007.

[3] Route Network Definition File (RNDF) and Mission Data File (MDF) formats, 2007.

[4] Velodyne HDL-64E Datasheet, 2007.

[5] The National Highway Traffic Safety Administration. Traffic safety facts, 2006.

[6] R. Peter Bonasso, James Firby, Erann Gat, David Kortenkamp, David P. Miller, and Marc G. Slack. Experiences with an architecture for intelligent, reactive agents. volume 9, pages 237–256, April 1997.

[7] Kurt Dresner and Peter Stone. Multiagent traffic management: A reservation-based intersection control mechanism.

[8] A. Elfes. *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*. PhD thesis, Carnegie Mellon University, 1989.

[9] Brian Gerkey, Richard T. Vaughan, and Andrew Howard. The Player/Stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the International Conference on Advanced Robotics (ICAR)*, 2003.

[10] Texas Transportation Institute. The 2007 Urban Mobility Report, 2007.

[11] Daniel Jiang. What is DSRC?, 2002.

[12] Mohamed Mostafa Joe. GPS/IMU products – the Applanix approach.

[13] J. Modayil and B. Kuipers. Bootstrap learning for object discovery. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[14] Tarun Nimmagadda. AIM Videos, 2008.

[15] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L. E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. Winning the DARPA Grand Challenge. *Journal of Field Robotics*, 23(9):661–692, 2006.

[16] D. Wolf, A. Howard, and G. S. Sukhatme. Towards geometric 3D mapping of outdoor environments using mobile robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2005.