

The Ohio State University

Autonomous City Transport (OSU-ACT)

Developed for the 2007 DARPA Urban Challenge

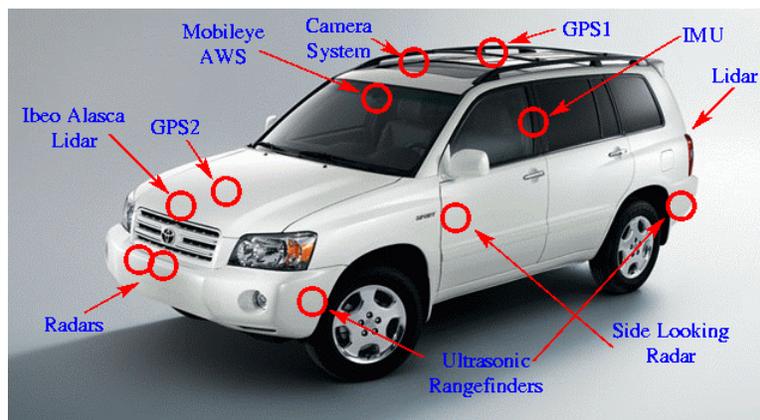
Submission date: June 1, 2007

Corresponding Author:

Prof. Umit Ozguner, OSU Dept. of ECE, [ozguner.1@osu.edu]

Contributors to Report:

Umit Ozguner
Keith Redmill
John Martin
Roberto Mati
Ahmet Yazici
Charles Toth
Arda Kurt
Alex Hsieh



“DISCLAIMER: The information contained in this paper does not represent the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA) or the Department of Defense. DARPA does not guarantee the accuracy or reliability of the information in this paper.”

Executive Summary

The OSU-ACT is an autonomous vehicle developed by a team of faculty, staff and students at OSU, with support from individuals and groups from Italy and Turkey. It utilizes technologies and experience gained through participation in the first two Grand Challenges and over a decade of autonomous vehicle research, development and testing.

The Hybrid SUV selected provided a simplified path for drive-by-wire attainment and enough electrical power for the computers and sensing devices.

Sensors were chosen based on our Grand Challenge experience, evaluation of urban driving needs and budgetary constraints. Primary surround sensing relies on lidars, with vision and radars playing a secondary role.

The autonomous drive through the urban area is based on evaluation of the mission over a map database. As each checkpoint is reached, the next sequence of anticipated situations are pre-planned. These situations are the “meta-states” of a hierarchical state-machine, where the sub-states rely on sensing and analysis of the environment.

The five distinct portions of “intelligence” are distinct software modules. They are:

- Plan Generation
- High Level Control
- Low Level Control
- Sensor Fusion
- Situation Analysis

Tests and selection decisions were undertaken through the development and a sophisticated simulation environment was created and used for verifying all aspects of “intelligence” and software implementation.

1. Introduction and Overview

The Ohio State University Autonomous City Transport (OSU-ACT) is the 7th generation autonomous vehicle produced in 12 years by a core group of researchers at OSU. OSU-ACT was developed by a large team, and many of its sub-systems, hardware selections and the algorithms coded rely on our experience, especially in the first two DARPA Grand Challenges. There are only a limited number of areas where legacy software and some older hardware was used. But a number of our decisions, selections and tests have relied on our previous experience.

In this report we outline the hardware, the over-all intelligent structure for solving the autonomous urban driving problem, and the algorithms developed and used. We also outline tests used, criteria for comparisons and the development process.

Figure 1 shows the logic structure of ACT. The five distinct portions of “intelligence” will be explained in the sequel and tests and design choices will also be outlined.

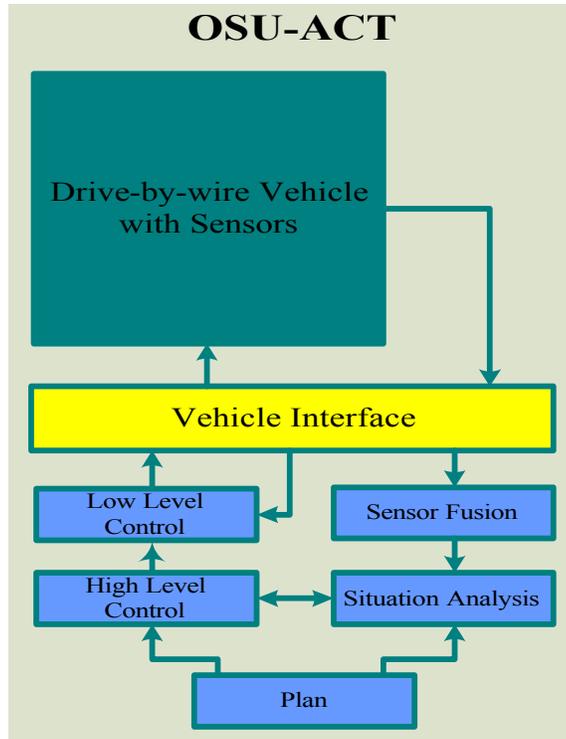


Figure 1. Structure of the OSU-ACT.

The five distinct portions of “intelligence” are labeled:

- Plan Generation
- High Level Control
- Low Level Control
- Sensor Fusion
- Situation Analysis

2. Hardware

2.1. Vehicle Drive By Wire

For the 2007 DARPA Urban Challenge, a new 2006 Toyota Highlander Hybrid SUV was selected for automation. The interior space available in the vehicle provides significant flexibility for the mounting of processing, power, and control hardware, at the cost of only two additional feet of curb-to-curb turning circle diameter and 4 inches of additional width relative to an ordinary passenger sedan. The exterior has convenient hard points for mounting sensors or attaching mounting brackets. The hybrid vehicle provides a number of advantages for automated vehicle purposes. DC power for computers, electronics, and sensors can be derived directly from the vehicle's 280 volt battery pack using DC-DC converters and DC-AC inverters, and a drain of 1800-2500 watts will be insignificant compared to the 45KW maximum output of the battery pack.

The task of actuation is also significantly simplified since the vehicle employs electric actuators for its stock systems. The vehicle is inherently throttle-by-wire, as the division of electric motor vs. ICE power is dynamically controlled. Under normal operating conditions the vehicle is essentially brake-by-wire as well: the hydraulic master cylinder is isolated from the actual hydraulic brake system and driver requested braking torque, which is sensed using a brake pedal position sensor and a pressure sensor in the master cylinder-stroke simulator hydraulic circuit, is measured and then generated automatically through ECU control of regenerative braking and the electrically-operated hydraulic brake system. The transmission is controlled electronically, with the exception of the parking gear. Controlling the throttle, brake, and transmission is accomplished by emulating the existing vehicle driver intention sensors with custom electronic interfaces.

The vehicle has electric power steering: implemented with a 42-volt brushless DC motor integrated into the steering rack. Originally we intended to control the stock EPS motor by connecting it to a brushless DC motor amplifier, mounting a rack position sensor, and implementing position control. However, disabling the vehicle's electric power steering ECU generated error conditions that affected other vehicle systems. Therefore, an Animatics SM2337D brushless servomotor with an integrated amplifier and position control system along with a Carson 23EP055 planetary gearhead was mounted on a hinged bracket that allows it to be locked against a gear mounted on the steering column for autonomous steering.

2.2. Position Sensing

The current position, orientation, and velocity of the vehicle is estimated using a software module filtering data from a number of sensors. Two Novatel Propak-LB-L1L2 GPS receivers using Omnistar HP differential correction technology provide direct position measurements. A Crossbow VG700A three-axis fiber optic vertical gyroscope provides angular rate and linear acceleration measurements. These can be augmented with measurements from the vehicle's own stability system gyroscopes and accelerometers which are obtained from the vehicle CAN bus. Independent wheel speed measurements for each wheel, along with overall vehicle velocity, transmission, and steering wheel angle are also obtained from the vehicle CAN bus. If available, a magnetic compass can also be integrated into the sensing system.

2.3. Environment Sensing

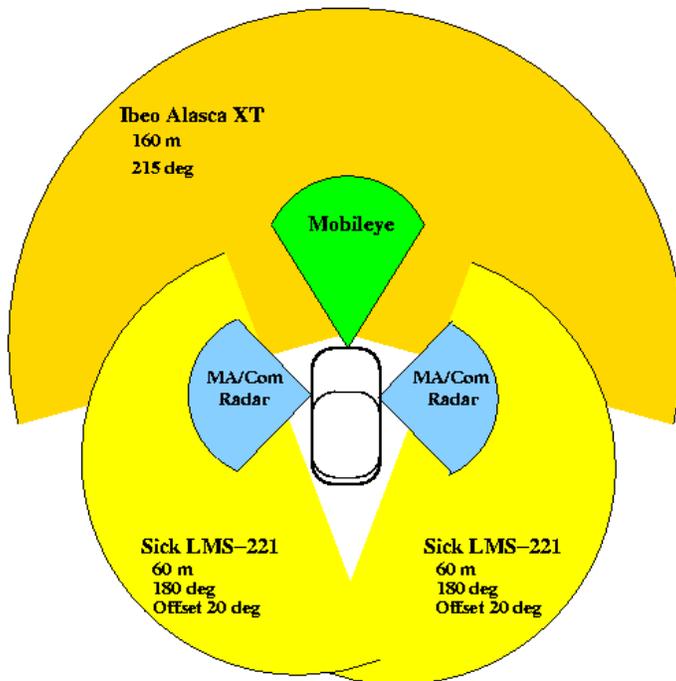


Figure 2. Sketch of sensor coverage.
Another forward-looking camera may yet be added.

Figure not to scale.

The vehicle has been equipped with a sensor system that completely covers the area around the vehicle, as shown in Figure 2. A forward looking scanning laser rangefinder,

the Ibeo Alasca XT, provides 4 vertical scanning planes, a range of up to 160 meters, an azimuth resolutions under 0.25 degrees, and a horizontal field of view (as mounted in front of the grill) of 220 degrees. Two Sick LMS 221-30206 scanning laser rangefinders are mounted behind the rear bumper and angled at 20 degrees from the lateral axis of the vehicle to provide rear and side sensing with a range up to approximately 60 meters and azimuth resolutions of 0.5 degrees. Two side mounted MaCom Short Range Sensor radar systems, with a range of approximately 20 meters and a horizontal field of view up to 70 degrees, provide side sensing of vehicles in adjacent lanes and help to fill the gaps in coverage of the lidar sensors close to the vehicle. In addition, a Mobileye AWS image processing sensor system is installed to both detect forward vehicles and to provide lane marker extraction. OSU developed lane marker tracking may also be installed if needed, as well as image processing algorithms developed by our partners.

2.4. The System Architecture

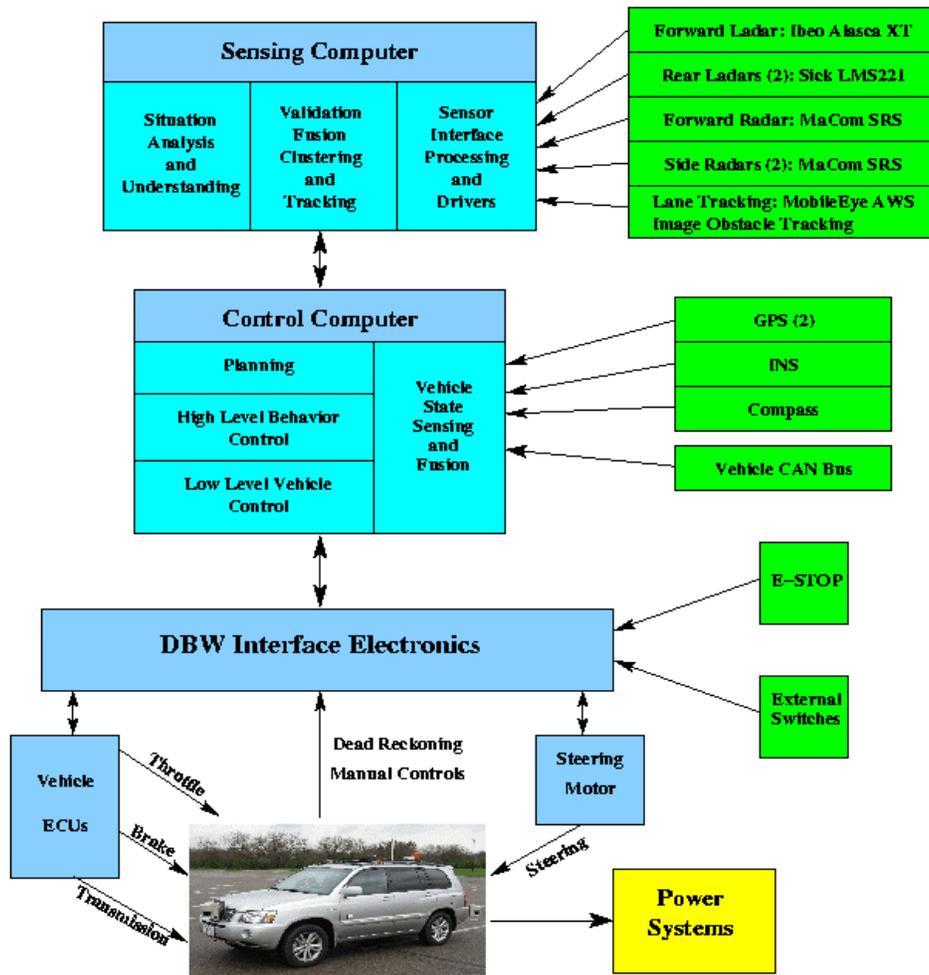


Figure 3. System Architecture (Version 1).

The architecture of the vehicle systems and computers is shown in Figure 3. The sensing computer is a dual processor Pentium system running the Linux operating system and equipped with Arcnet interfaces for the Alasca XT, high speed RS422 serial ports for the

Sick lidars, and a CAN bus interface for the Mobileye and MaCom radar systems. The control computer is a dual core Pentium Duo running the QNX real-time operating system and equipped with serial ports, analog and digital I/O and a CAN bus interface.

3. Algorithmic Approach

3.1. Scenario Generation and Route Planning

We follow a scenario-based hierarchical hybrid system design approach in controlling ACT. This is a significantly extended version of the approach we used in the two DARPA Grand Challenges [1-2].

The finite-state machine (FSM) that will run the car is comprised of two levels. (The two-level FSM is somewhat similar to the approach we used in NAHSC Demo'97 [3] in accomplishing a scenario.) The higher-level states are called the Meta-States. The Meta-States are those that can be discerned from the map, as selected by the Route Plan. Thus, as one traces the path along the Route Plan, one can identify the Meta-States (lane, intersection, zone, zone-with parking lot, etc.)

We define a scenario as a mapping of the mission definition, provided by the MDF, to an ordered set of tuples (road/lane segment ID, appropriate state (and sub-states), entrance and exit criteria). The mapping will depend on selection of an optimal Route. The ACT high-level controller (HCL) will execute this set of tuples until either the scenario is complete or an external event requires the scenario to be replanned. In other words, a scenario defines a specific route through the road network that will accomplish the mission and the expected situation and required vehicle behavior at each stage along the route. Although it seems possible to have the full scenario at the outset, this is not necessary. Each mission will have multiple checkpoints and we plan a scenario only from one checkpoint to the next, with possibly a two-checkpoint horizon.

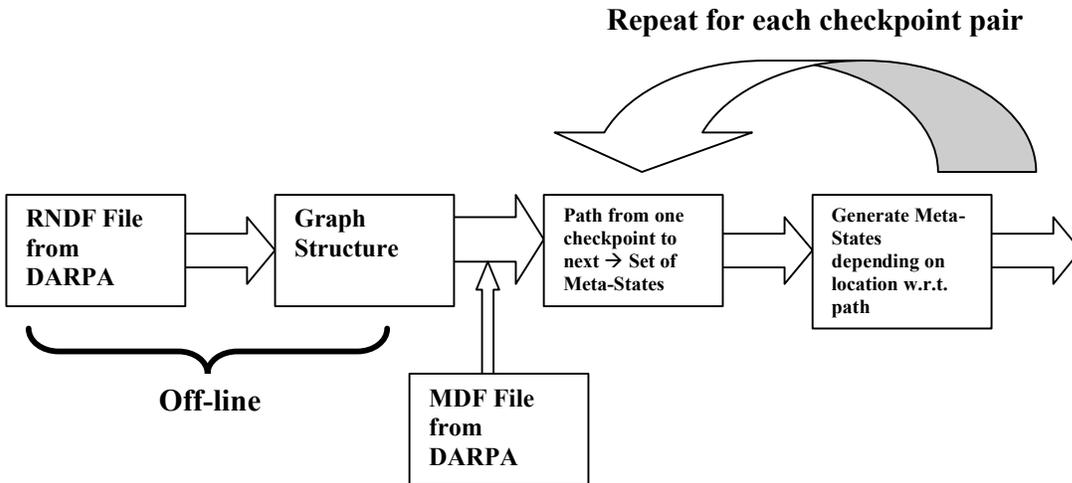


Figure 4. Establishment and running of the scenario

3.2. Route Planning stages

We shall use the term “route planning”, “path planning” or simply “planning” to indicate the sequence of streets, intersections and “zones” that we plan to traverse. Route planning is essentially what you would do with a street map. Route planning will provide us with a list of meta-states (and associated tasks) the car will encounter.

We shall use the term “trajectory planning” to indicate the curve (or sequence of points) the vehicle will follow as it goes down a street, avoids an obstacle, changes lanes, turns a corner, moves into a parking spot, etc.

There are four stages in Route Planning:

1. Establishment of a data structure based on map before race.
2. Route planning before Mission.
3. Route planning during Mission.
4. Route re-planning during Mission due to street blocking.

(1) Establishment of data structure: Almost all path planning algorithms start with an oriented graph. Our route planning is initialized by transforming the known data into a usable graph model. In this stage the given Route Network Definition File (RNDF) is transformed into a Network Graph Structure (NGS).

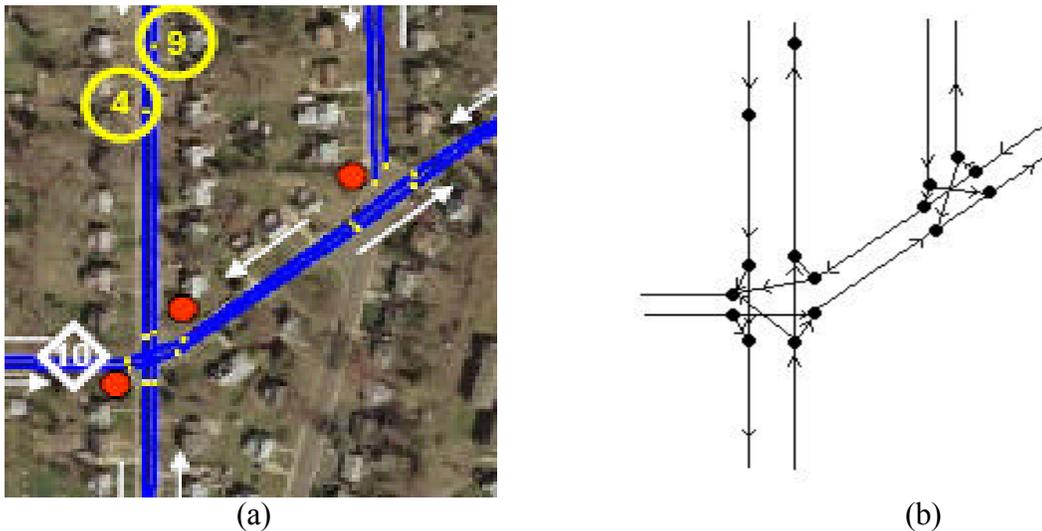


Figure 5. a) A section of sample RNDF map. b) Corresponding NGS.

NGS specifies an oriented graph whose “nodes” are

- Entry/Exit points
- Entry/Exit of Intersections
- Parking locations within zones

We identify different types of “links” in the NGS:

- Lane links
- Intersection traversal links
- Zone links

- Roundabout links

Each link has associated with it data that identifies its meta-states.

(2) Route planning before Mission and (3) Route planning during Mission: It is assumed that route planning need only be done from checkpoint to checkpoint. When the Mission Data File (MDF) is provided, the car need only establish the first path (from start to first checkpoint). As the checkpoint is reached, the next one is picked from the file and the optimal path between the two is generated at that time. Thus a full route from beginning of the Mission to the end is not needed.

(4) Route replanning: After a “U-turn” on a lane, a new route has to be established to the current goal checkpoint and the blocked lane has to be removed from the NGS. Route replanning can also be undertaken after an e-stop “pause”.

Establishment of a path specifies a sequence of meta-states that the car will traverse in going from one checkpoint to the next. Thus the path selected creates a list of Meta-Tasks. (As mentioned before, the Sub Tasks will be situation dependent.)

3.3. Planning between defined node pairs: The A* algorithm

For any given start node/checkpoint and goal checkpoint in this environment, the A* search algorithm is used to plan a route using the graph structure. The A* algorithm is a well-known path planning algorithm [18]. Estimated cost of a path through node n is calculated as $f(n)=g(n)+h(n)$, where $g(n)$ is the cost (distance/time) taken to reach the node n , and $h(n)$ is the estimated cost of the cheapest path from node n to the goal node. The A* algorithm guarantees that the solution is always optimal [19] if $h(n)$ is an admissible heuristic, that is $h(n)$ never overestimates the cost to reach the goal. Our design for A* includes minimum distance/time path for a given initial and final points. In the case of distance minimization, straight line Euclidean distance is used for $h(n)$. To maintain the optimal solution for the time minimal path case, $h(n)$ is calculated as Euclidean distance divided by maximum allowed velocity on each roadway, estimated traverse times for intersections and zones, and estimated completion time for U-turns. (There are no traffic estimates.)

Figure 6 shows a sample route from checkpoint 11 (WID.3.1.2) to checkpoint 5 (WID.1.1.18) for our Site visit path.

3.4. Output of the Route Planning Module and Performance tests

The Situational Analysis” and “High Level Control” modules use the output of the Route planner module. The Route Planner provides the output as an array of link structures. These structures consist of all waypoints describing the link, minimum/maximum velocity values, right and left boundary classifications, and Meta-states. The route planner module currently finds the route in less than 1ms for sample RNDP. Considering four or five time larger graph and $O(n^2)$ nature of the A*, for the final event, the planner can produce solution in a very short time.

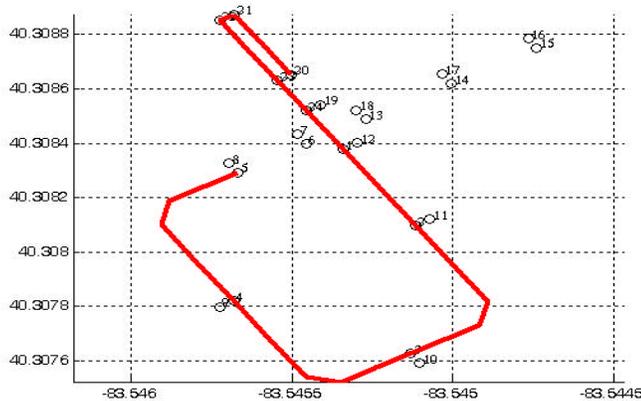


Figure 6. A sample route from checkpoint 11 to checkpoint 5

4. High Level Control

4.1. High Level Control Tasks

The HLC module runs a Finite State Machine by sequencing the Meta-States according to the Route Plan. Transition among the Meta-States is location dependent, except for U-Turn, which is initiated by external sensing.

Transitions among sub-states are triggered by the Situation Analysis Module.

The output of HLC is provided to the Low Level Control (LLC), usually as a set of waypoints and speed, and also as special instructions.

4.2. The Meta-States

Each Meta-State has a number of sub-states and a FSM underneath it. The lower level states will depend on the situations encountered in real-time. We list below a partial, descriptive explanation of a few Meta-States (see Figure 7):

One and two lane roads: We have previously demonstrated car following and passing a stopped or slower car [3]. For two lane roads, one must observe the next lane for traffic while merging into traffic or passing. Lane change behavior is a separate state or a sub-state of standard road following. The reference path that is passed to LLC is generated by fitting a Catmull-Rom spline to the following waypoints and sampling the spline at three-meter intervals for a total of ten points. The reference speed passed down to the LLC is based on the speed limits obtained from the mission definition file altered by road curvature concerns, obstacle or road termination detection, and car following. This reference speed and the 30m-reference path that consists of ten regularly spaced points form the bulk of the HLC-LLC communication. In addition to this main functionality, additional commands that can be passed using the same interface include turn-signal utilization, specialized motions and emergency braking.

T Junction and 4-way Intersection: Intersection behavior is a new and complex problem, with numerous permutations of potential behavior depending on which direction the vehicle approaches the junction and which roads have stop signs.

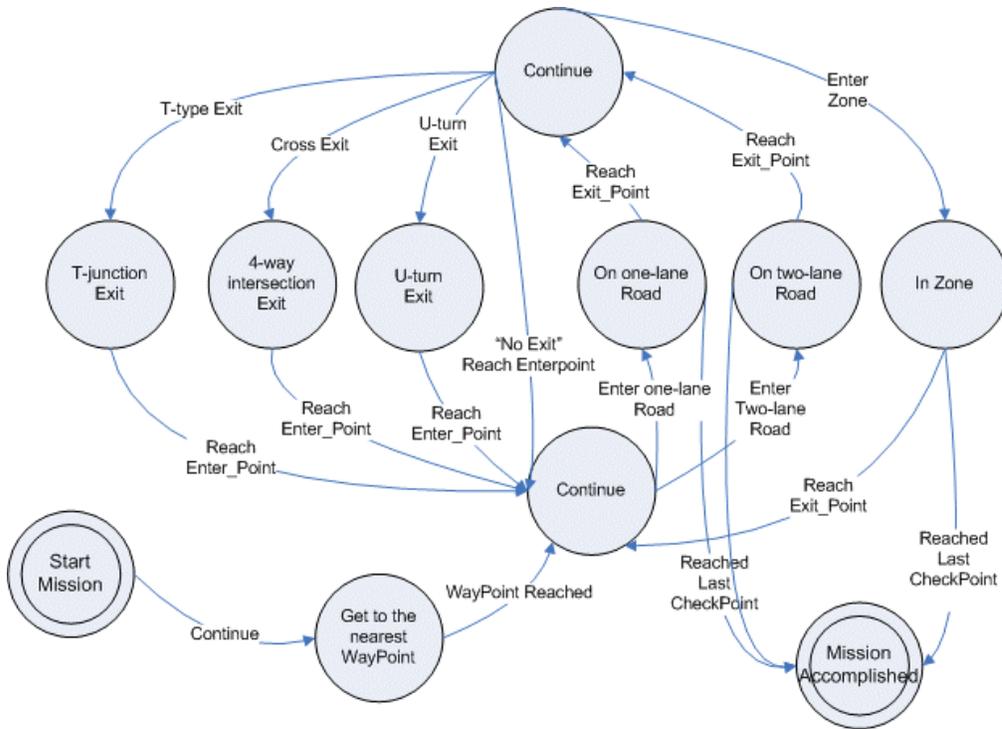


Figure 7: The Meta-States (Version 1)

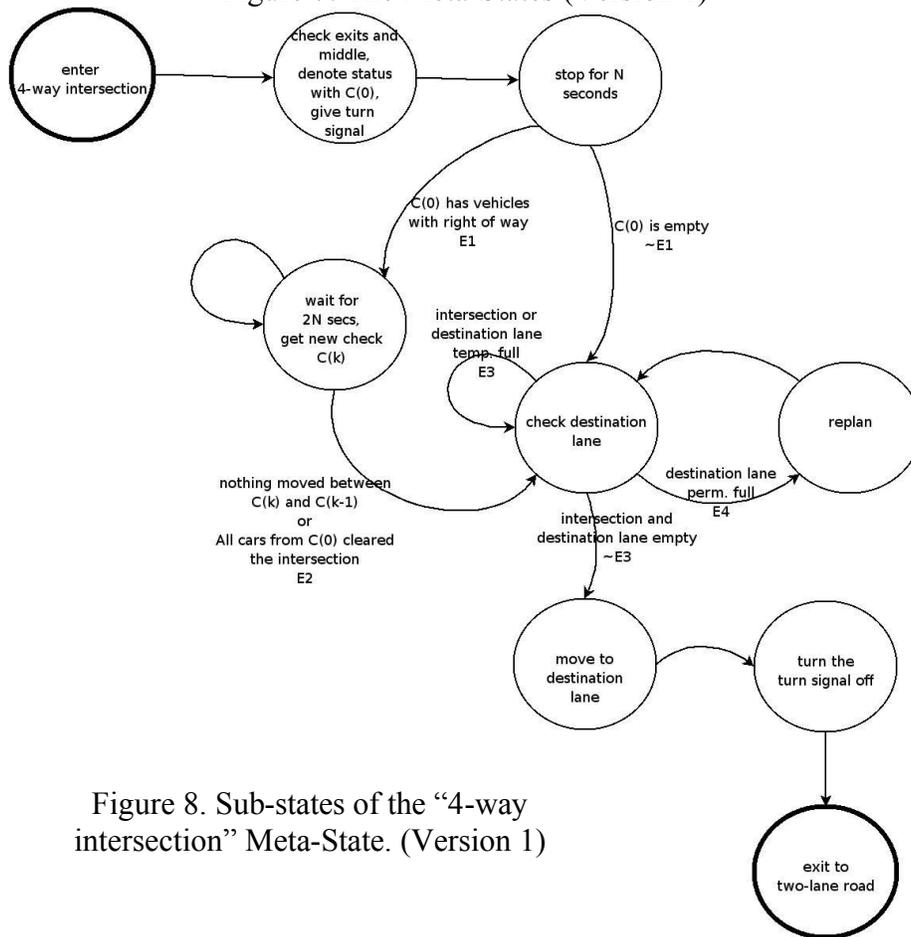


Figure 8. Sub-states of the “4-way intersection” Meta-State. (Version 1)

Each potential behavior will be enumerated as a different sub-state machine. For a given scenario at a specific intersection and specific approach direction only one state will be valid.

An example set of sub-states from the Intersection Meta-State is given in Figure 8.

The zone: This is somewhat similar to the behavior for GC'05 [4]. It has been extended to directly handle moving obstacles. Our sensing and sensor fusion systems have been extended to provide specific identification and tracking of moving objects. One possibility in this state is parking. We have developed both a trajectory establishment procedure and precision motion controllers to handle both pulling in and backing out of a parking space.

5. Sensor Fusion

5.1. Clustering and Tracking

The sensor fusion algorithm implemented on OSU's 2005 DARPA Grand Challenge vehicle used a grid occupancy approach [4]. Due to the traffic situation where an environment is highly dynamic ACT has moved towards a clustering and tracking approach.

The sensor fusion computer is responsible for clustering and tracking all objects which are seen by the sensors. Vehicle detections which are returned by the Mobileye and Macom radars are matched to a lidar generated cluster by looking for a lidar cluster within some distance threshold. If no suitable matching cluster is found, the Mobileye and Macom detections may update or initialize a track without a corresponding lidar cluster. Figure 9 shows returns from all three lidars combined into a single vehicle centered display. The solid blue rectangle in the middle of the display is the ego vehicle. The front of the vehicle is towards the top of the display. The red outline of the laser returns shows other vehicles in a parking lot, and a building to the rear of the vehicle.

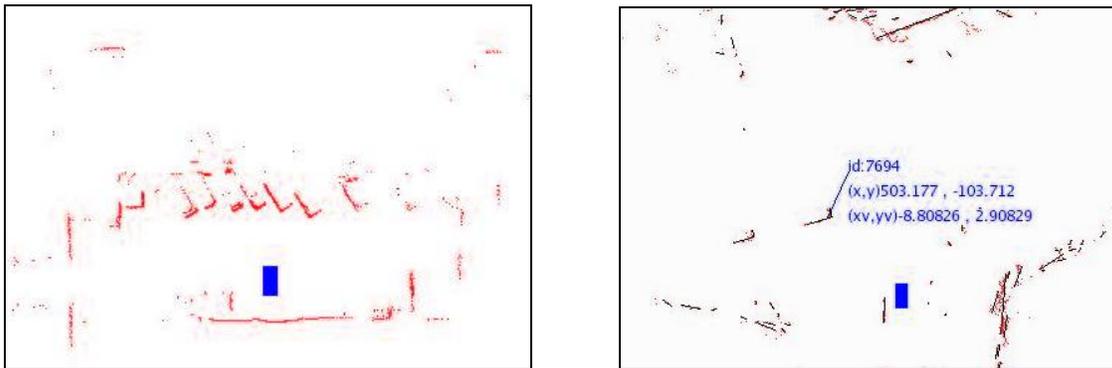


Figure 9. Lidar returns in a parking lot and at an intersection.

The sensor fusion computer first uses information about the position and orientation of the lasers with respect to the vehicle to transform the returns into a vehicle centered coordinate system. Once the returns from the lidars are in vehicle centered coordinates,

the position and orientation of the vehicle with respect to the world are used to transform the lidar returns into a rectangular coordinate system that is fixed with respect to the world.

After the lidar returns have been transformed into world coordinates, they are clustered into groups of points. The clustering algorithm simply places the laser returns into a disjoint set data structure using a union-find algorithm [5]. Ultimately, clusters of laser returns are found whose members are not further than some maximum distance from each other. Although the maximum cluster distance is still subject to some experimentation, it tends to be around 1.5 meters. Once the lidar returns have been clustered, the resulting clusters must be tracked. The tracker keeps a list of active state information for each existing track. The tracks are then matched to the incoming list of clusters. Clusters that cannot be matched to an existing track are then used to initialize new tracks. Clusters are matched to tracks by minimizing the distance between the centroids of the clusters and the predicted location of the tracks. The track predictions are generated with a variable gain gh filter [6]. The data association minimization problem is solved as a linear assignment problem using [7]. The output of the sensor fusion algorithm is a list of tracks. Each of the resulting tracks has a position and velocity. Also, the general size and shape of the point cluster supporting the track is abstracted as a list of linear features.

5.2. Sensor Fusion Results and Performance

The clustering and tracking software has been tested on busy city streets in Columbus, Ohio surrounding OSU's Center for Automotive Research. Figure 9b shows two screen shots of sensor monitoring application showing the sensor output while starting and driving manually in heavy traffic. One of the tracks is showing the track id as well as the position and velocity of a vehicle crossing the intersection immediately in front of the ego vehicle that is represented by a solid blue rectangle.

Generally, the clustering and tracking algorithm shows reasonable performance. However, the use of the cluster centroids to represent a cluster's position presents problems. There are several instances where the centroid of a cluster may move independently of the object responsible for the cluster [8] when an object is partially occluded and moving into occlusion, when an object casts a "shadow" over another more distant cluster, when an object is large enough to extend past the range of the laser (hedge or barrier on the side of a road) and when an object's aspect changes with respect to the lidar. In the future it may become necessary for the OSU clustering and tracking algorithm to attempt to find corner points within clusters and use these points to estimate the position and velocity of a cluster's track.

6. Situation Analysis

6.1. Introduction

For an urban scenario, we are interested in all the clusters in our path, or on a road intersecting our lane. While an autonomous car is navigating through the city, many different situations may arise. The situations may vary if the car is on a one-lane road, a two-lane road, an intersection, and so on. Particularly critical for an autonomous car are

those situations related to intersections. When a car is approaching an intersection, it must give precedence to other vehicles already stopped. If the vehicles are stationary for a long time, the car must decide whether those vehicles are showing indecisive behaviour. Other situations may involve road blockage (the vehicle might carefully perform a U-turn), parking in parking lots, and dealing with dangerous behaviour from other vehicles. All the situations must be evaluated, and sent to the High Level Controller (HLC) in order to drive the car.

The term *situation* is defined to be knowledge concerning the vehicle and/or the prevailing scenario and surroundings. We call *atomic situation* all those situations that cannot be subdivided into other situations. From a practical viewpoint, situations are the switching conditions among meta-states and all the sub-states inside the state-machines. Thus, the aim of the Situation Analyzer Module is to provide the High Level Controller with all the switching conditions.

6.2. Input and Outputs

Inputs and outputs for the Situation Analyzer system are illustrated in Figure 1. The Sensor Fusion and Tracking Module provides information about the set of tracked objects. The information about position and velocity of the centroid, combined with the location of the lines defining the boundary, are given in a local navigation reference frame. The Path Planning Module provides the information related to the optimal path. The path is a sequence of links defining a traversable road. Starting from the path, the Situation Analyzer can identify the location of the road. The road model is structured as a set of polygons generated from a spline curve fitting the waypoints in the optimal path. Such a road model design is particularly suitable for both accuracy and implementation purposes. In order to let the computational cost be as low as possible, only the situations related to the current meta-state or sub-states are checked.

The current sub-state is provided by the HLC. Obviously, there are a number of situations that may arise at the same time. Another input comes from GPS-INS system. The location of the vehicle is fundamental, because both the clusters and the road are provided in a geographical reference frame.

The whole Situation Analyzer output is a vector whose elements are boolean values. Each element contains value 1 if the corresponding situation occurs, 0 otherwise. All the situations are reconstructed from atomic situations simply using logic operations. An example can be shown from Figure 10. If we take into account the state “Check passing lane”, we can realize that there are only three situations to be checked. E6 switching condition “Passing lane full temporarily” is an atomic situation, while E7 is just a situation made from the two atomic situations “Passing lane full permanently” and “Not enough space before intersection”, linked with an OR logic gate.

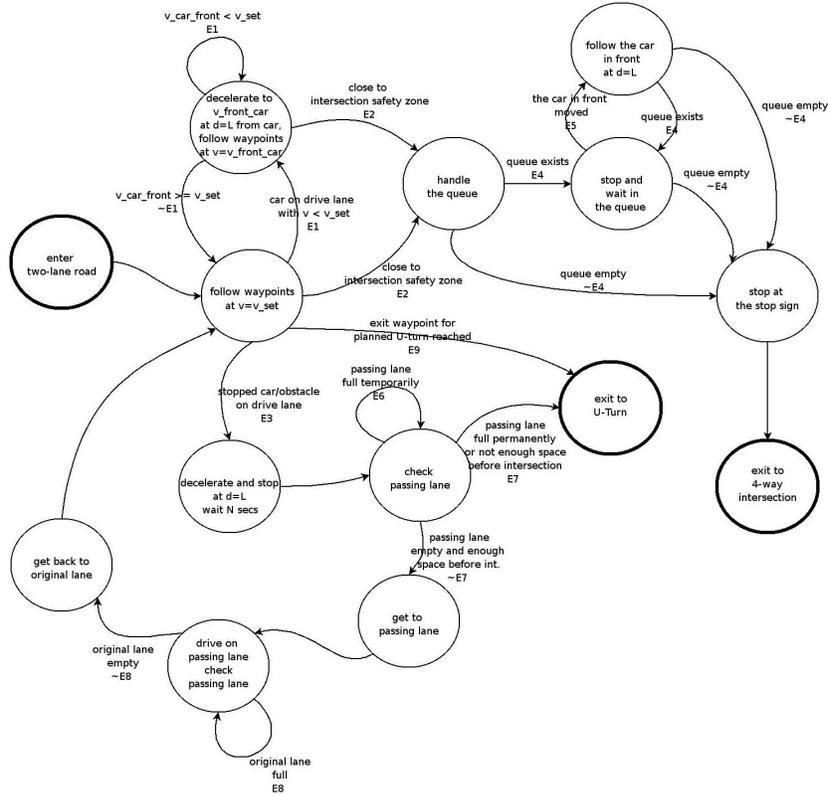


Figure 10. Two-Lane Road meta-state

6.3. Road model generation

One of the most important issues in the Situation Analyzer is road model generation. A good road model, very close to the real one, is fundamental to safely driving in an urban area, avoiding entering passing lanes with opposite driving direction, and remaining at a safe distance from the boundary of the lane. The road is generating starting from the optimal path. All the links inside the optimal path are drivable, and the width of the lane in correspondence of a certain waypoint is provided. From the links in the optimal path, the list of the consecutive optimal waypoints is extracted. For an accurate road generation we need close points, while the waypoints may be very far from each other. For that reason, a Catmull Rom [9, 10, 11] spline function passing through the waypoints is generated. Catmull Rom splines are unique from other mathematically computed arcs in that they pass through all of their control points. Moreover the spline is C^1 continuous, meaning that there are no discontinuities in the tangent direction and magnitude. However, the spline is not C^2 continuous. The second derivative is linearly interpolated within each segment, causing the curvature to vary linearly over the length of the segment. The spline function is then sampled and all the samples are chosen to be equally spaced. In order to have a very accurate road model, we choose to keep a distance of 2 meters between every pair of consecutive samples. Subsequently, all the samples are linked with segments, as shown in Figure 11 with the red straight lines. In this way, the set of segments is an approximation of the spline curve.

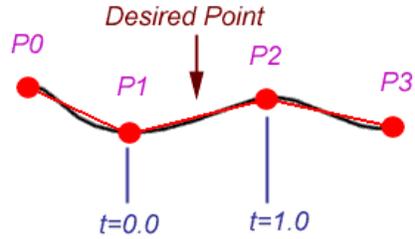


Figure 11. Example of Catmull Rom spline function

We considered a number of different road model designs. One possible approach is to consider a road map with fixed memory occupancy. The map can be divided into cells. Depending on the desired accuracy, the number of the cells may be very large. Since the number of the cells is fixed, the map should be very frequently updated during the mission. Moreover, in that design there might be many empty cells in memory, carrying no information about roads.

Instead we designed an accurate and computationally simple method to represent a road by a set of polygons. From each pair of consecutive samples, a rectangle is constructed. One side of the rectangle has the same length as the distance from the two samples. The length of the other side is the lane width. All the rectangles are then fused together, as shown in Figure 12. The intersection point between two consecutive rectangles, and its opposite point are then computed, and the resulting polygon is a trapezoid. For every trapezoid, the vertices are ordered. All the trapezoids, taken in sequence, will generate the current road. Generating a road as a set of polygons also allows extension of the current implementation with information coming from different sources.

This compares favourably with many algorithms proposed in the literature to extract the edges of lanes [12,13,14] with a prescribed accuracy.

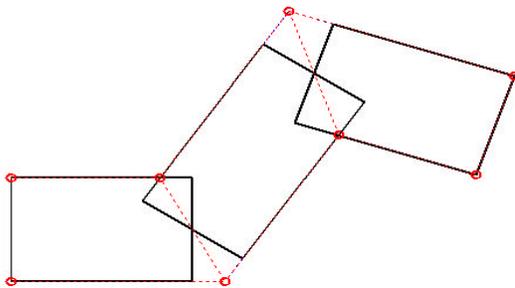


Figure 12. The road model generation process. The red dots are the final road vertices

6.4. Primitives

Some primitives have been defined to easily handle the situations. To deal with many situations, we must be able to figure out if an obstacle is inside a lane. To handle this case, algorithms for finding points inside polygons have been carefully taken into account. Many of these algorithms can be found in the computer graphics literature [15, 16, 17]. Most of them are very simple to implement, and have a low computational order, typically $O(n \log n)$ or $O(n^2)$, with n the number of the vertices in the polygon.

As we define the obstacle with a set of lines, we want to check if at least one line is inside a polygon. We face this problem by sampling all the lines of the cluster, obtaining equally spaced samples, and by applying the “find point in polygon” algorithm to all the samples in the cluster.

6.5. Design of the situations

In this technical report we give details about the design of the situations involved in the Site Visit. However, all the situations are designed and easily extended to more complex scenarios. In particular, many atomic situations are designed to be augmented with information coming from a sensor system. This is especially useful when the vehicle cannot completely rely on the waypoints.

6.6. Intersection meta-state

In real cases, when approaching an intersection, a human driver must take care of a number of possible situations. A T-junction can be thought to be a particular case of a 4-way intersection, where an exit is completely blocked. All the situations that can be encountered are described in the 4-Way Intersection meta-state. Three common situations are here shown as an example: Is it my right of way? Is the intersection free? Are there cars showing indecisive behavior?

When the vehicle arrives at the intersection, it checks the other stop areas. The vehicle will wait until all the previous clusters have moved. To check out whether the intersection is free, a polygon with the same shape as the intersection is used. Using the primitives, it is simple to realize whether an object is occupying that area. In real urban scenarios, the exact dimension of the whole intersection might not be known from stored information. In order to improve the knowledge about the exact geometry of the intersection area with visual information, our current approach is as follows: Let us suppose that we arrive at the intersection with the correct heading angle. From the knowledge concerning the width of the crossing lanes, it is simple to create a box to be superimposed onto the intersection area. With visual information, the vertices of the polygonal intersection area could be stretched, providing a more precise geometry. A particular situation can happen when one or more cars are showing an indecisive behavior, that is, when they remain stopped inside the intersection for a long time, or they are moving too slowly inside the intersection. This situation may be understood by keeping track of the unique ID of the cluster inside the intersection, and associating a timer with each of them. After a fixed time has expired, the cluster is classified as indecisive.

6.7. Double lane meta-state

Situations inside Two-Lane Road meta-state are easily handled starting from the simple structure of the road. When we enter a two-lane road, the opposite lane is automatically created. The opposite lane is constructed from the samples in the original lane. In this way, we obtain an opposite lane whose right boundary perfectly matches the left boundary of the original lane. We investigated other choices for making the opposite lane. One could take the waypoints in the opposite lane from RNDF file as provided by DARPA. But we are not guaranteed they will be coupled with the waypoints in the

original lane. If that is the case, fitting the waypoints in the opposite lane with a spline function might conduct to a right boundary for the opposite lane that does not perfectly matches the left boundary of the original lane; in other words there might be “holes” between the two lanes.

Typically, the situations inside the Two-Lane meta-state involve distance and velocity of objects. Situation like slow traffic, stop-and go behavior, lane blockage, are easily treated by evaluating the position and velocity of the cluster inside the specific lane.

6.8. U-turn meta-state

Before performing a U-turn in a real urban scenario, the driver must pay attention to incoming vehicles from the opposite lane. A typical U-turn scenario is shown in Figure 11. While the car drives in lane A, all the information about the opposite lane is known. When the car is approaching the last point in the lane A, the Situation Analyzer checks lane B and informs the HLC about incoming vehicles.

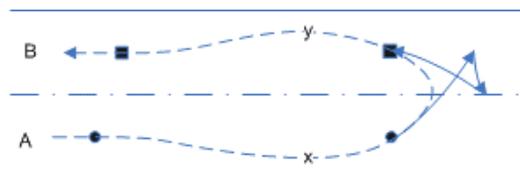


Figure 13. U-Turn meta-state

7. Low Level Control:

7.1. Command Interface

The low-level control receives operational instructions from the high-level control module. These instructions take the form of

1. A path to be followed, defined by 10 approximately evenly spaced control points
2. A desired speed
3. Commands to indicate starting and several forms of stopping
4. A command to indicate a precision stop at a given stopping line
5. Commands to enable specialized ("robotic") motions, for example moving a specified distance along a constant radius arc

The low level control will execute a given command set until either the command is completed and the vehicle is in a stationary state, or until the vehicle has driven off the end of the path provided, at which point the vehicle will be stopped, or until it receives a new command set.

7.2. Longitudinal Control:

The interface and control of vehicle throttle and brake was previously described. A PID controlled is used to generate a virtual torque command to achieve the commanded speed, and a state machine is used to select between the use of throttle, active braking, or engine idle braking. Speed commands are modified to constrain the acceleration and jerk of the

vehicle to a preset comfortable limit. There are also "emergency" deceleration modes that are less comfortable.

To execute a precise stop, the low level control determines the distance from the vehicle's current position to a line drawn through the specified stopping point and perpendicular to the vehicle's path of travel, taking into consideration the distance from the front bumper of the vehicle to its centroid. The speed of the vehicle is controlled to follow a specified deceleration trajectory which results in a speed of 1.0 m/s when the front bumper of the vehicle reaches a distance of 3.0 meters from the stopping line, at which point the vehicle travels at that constant speed of 0.6 m/s until it is brought to a stop over the stopping line.

7.3. Lateral Control:

The path that the vehicle is to follow is specified as a set of 10 control points. The lateral controller identifies both the current location of the vehicle, denoted P_s in Figure 14, and the look ahead point 7.0 meters ahead of the vehicle along its lateral axis, denoted P_o , and extracts a subset of path control points closest to each location. Constant radius circles are fitted to the points in each subset, as shown in Figure 14. These circles are used to compute the vehicle offset distance from the path at P_s and P_o , denoted as O and S respectively in Figure 14, and can also be used to estimate a desired yaw rate. The subset of points also defines a desired yaw angle, from which a yaw angle error measurement can be computed. Situations in which the points define a circle of very large radius are treated as lines in special cases. The offset O and S drive squared-integral-derivative control laws, and the yaw errors drive proportional-integral-derivative control laws, the outputs of which are weighted and combined to generate the overall steering command.

Steering angle commands are limited by a speed-dependent maximum value to mitigate the chance of rollover.

7.4. Testing

The control results, as tested on the Video Submission Course, are shown in Figures 15 a-c. The path traveled is shown in Figure 15a. Note that after the first corner turn, the vehicle stops briefly, transitions into the left lane to pass a stopped vehicle, and then transitions back to the normal lane of travel. The vehicle also stops for the stop sign at the 3rd corner turn.

Lateral control, shown in Figure 15b, shows that offset errors are within 0.55 meters of the commanded trajectory, except when the trajectory is discontinuously changed to generate the lane change maneuver. Longitudinal control, shown in Figure 15c, demonstrates that velocity errors are within 0.35 meters per second, with the significant errors occurring when the vehicle is traveling on a downhill slope, except during deceleration to a stop when a comfortable deceleration rate is imposed for the first two stops, and an "emergency" deceleration is demonstrated for the final stop. Further improvements in longitudinal control are ongoing.

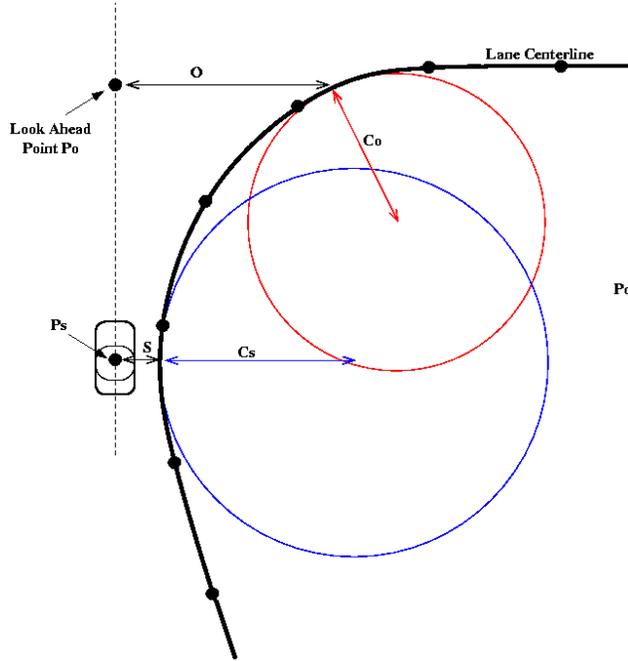


Figure 14. Trajectory following control

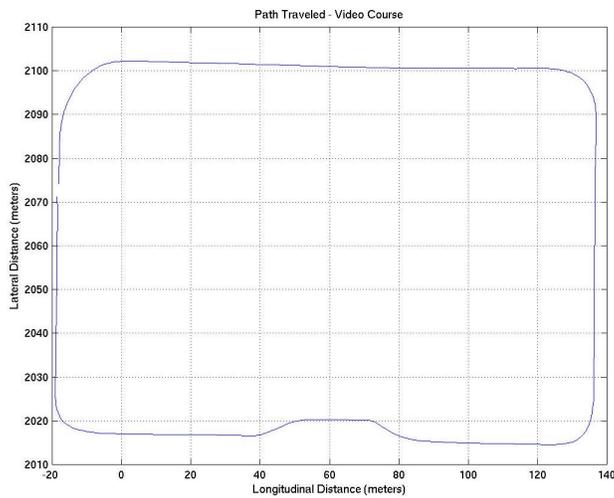
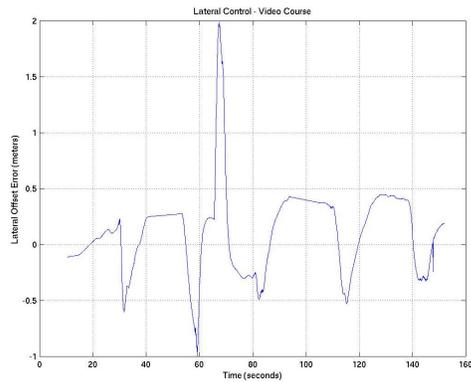
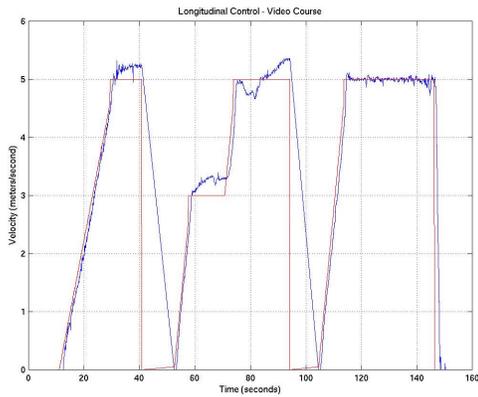


Figure 15: Test results for the video submission. (a) Full route, (b) Lateral control, (c) Longitudinal control.



8. Conclusion

In this report we have summarized the approach in developing OSU-ACT, and the design decisions and tests performed through the process. The description has stressed the new elements of our design, especially in light of the Urban Challenge requirements. Issues particularly relevant to the Site Visit have also been stressed.

Two sets of activities and developments were not mentioned in detail due to page restrictions:

- (1) Aspects of autonomous vehicle control we have had experience with and we already documented in the literature. Lane change is given in [20], car following is given in [21] and basic DGPS/INS use in [22].
- (2) Ongoing development is in
 - Estimation of multiple moving obstacles trajectories and collision estimation
 - Algorithmic speedup of some sensing and registration on a blade architecture computer
 - Vision based additional capabilities
 - Trajectory planning. (See [2] for our Grand Challenge version.)
 - Fault/failure handling

Our simulation studies indicate that problems provided by the Urban Challenge are being solved by the OSU-ACT.

APPENDIX A: Simulation and Visualization Tools

A number of simulation and visualization tools have been developed as summarized below.

Data Visualization Tools

Extensive data visualization tools were developed for the Grand Challenges, where we appreciated their contribution to the design process. Figure 9 is an example of one used for lidar data analysis.

The Gazebo Based Simulator

The computer simulator used for OSU-ACT scenario and architecture testing is based on Gazebo, which is part of the popular open-source software package, the Player Project¹. The primary function of the Gazebo simulator is to provide vehicle, robot or object models in a customizable, user-defined environment. These vehicle models are controllable through the well-documented Player interface just like a real mobile robot. The simulations required in OSU-ACT design and testing cycles can be broken into two major components. The first one is a realistic representation of the vehicle model. Both the physical properties of the OSU-ACT, such as the general size, shape and weight, as

¹ <http://playerstage.sourceforge.net/>

well as the dynamic characteristics such as acceleration, braking and steering behavior are included in the vehicle model used in the simulations. A number of sensors such as laser range finders and GPS receivers are also included in the model in order to simulate sensor fusion and decision-making processes of OSU-ACT.

The second component of the simulation is the environment definition, or the world-file. The world-file contains environmental constants such as gravity and friction and also provides a means to define a simulation course such as a simple ring or an intersection. The terrain and the objects can be customized, one or more vehicles or mobile robots can be placed in the environment and observation point-of-view can be defined through world-file manipulation.

For the traffic problems, more than one car can be simulated in Gazebo. With the multi-car simulation, intersection, road traffic, and other possible scenarios that are possible for the real Urban Challenge can be modeled in Gazebo.

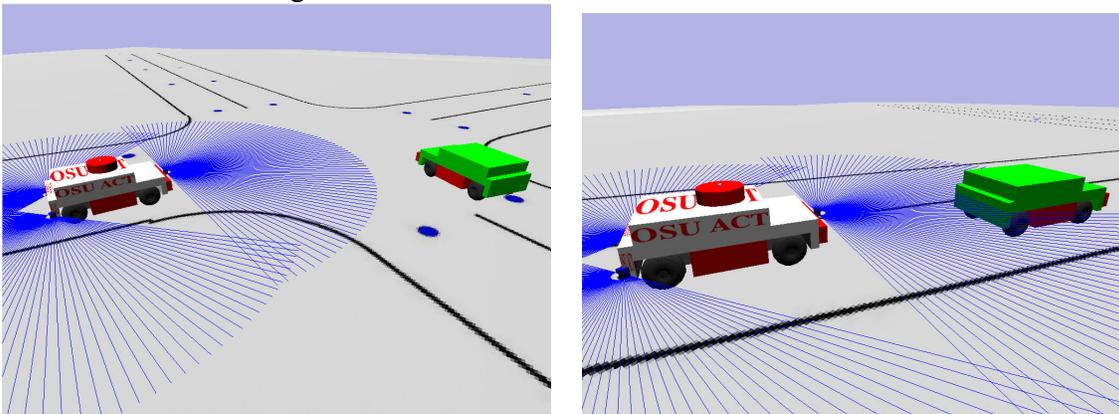


Figure-16: Intersection and vehicle following simulation.

The architecture of the Gazebo simulator is shown in Figure-17. (Note the match with Figure 1.) A simulation interface is created that matches the interface to the real vehicle.

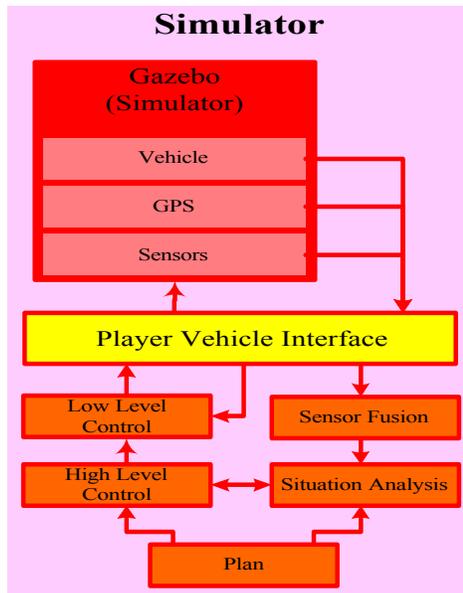


Figure-17. Gazebo Simulation Architecture

After all the control algorithms are successfully simulated in Gazebo environment, the same programs can be directly applied on the OSU-ACT.

Network Simulator

A Network Simulator, based on the VaTSim software that we developed some years ago [23], is being extended. This is to be used for testing state machine algorithms with different scenarios and traffic in complex urban environments.

Robot Based Emulation

Intersection and short routes are being emulated in the lab with wheeled robots. In fact an undergraduate Design Lab has run based on this lab setup.

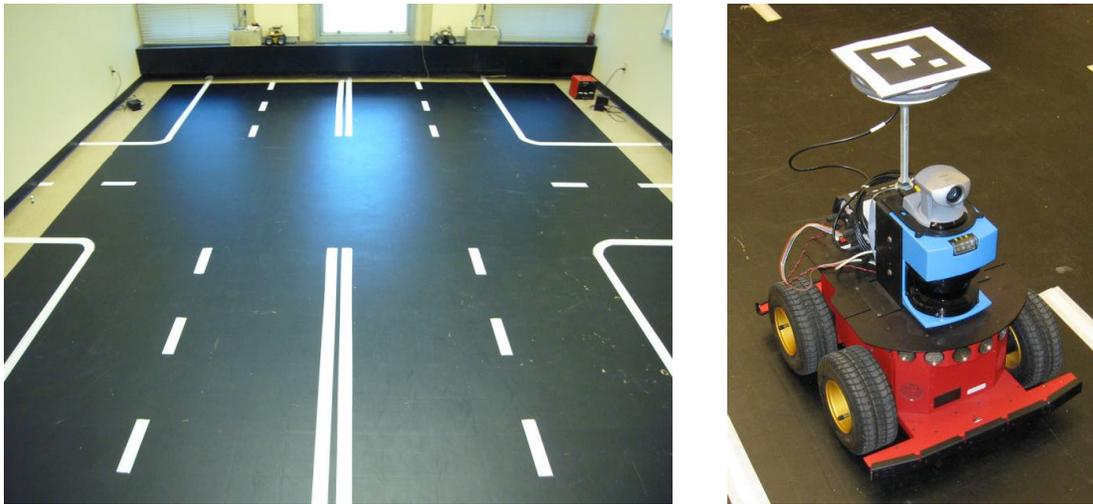


Figure 18. An intersection layout in the lab and the car-emulating robot.

See: http://www.ece.osu.edu/osuact/ece683_2007.html

Appendix B: Mapping Tools

The mapping support for autonomous vehicle navigation includes two essential tasks: 1) to provide terrain, natural- and man-made object/feature information for global path planning, as well as for vehicle local navigation (staying on course in off-road situations), and 2) to sense/map the vehicle vicinity, and thus provide for local vehicle control, such as staying on the road (within a lane), avoiding obstacles, etc. Since, the second objective falls into one of the goals of robotics, mapping is usually only referred to the first task.

Mapping or geospatial data can come in a variety of formats, including mass surface points such as DEMs; vector description of objects, such as road, building, contour lines, natural features; classification information, such as vegetation type, population density, traffic flow; imagery, such as airborne and satellite, unprocessed or rectified, monochrome or color; topology, such as describing the relationships between various data element; and so on. Map data are usually organized in a GIS database, which uses a multilayer approach to store the different data entity, and provides tools to visualize, access and analyze the geospatial data. In the evolution of the DARPA Grand Challenges,

there has been a shift from the relatively basic data formats toward the more abstract geospatial data entities. In the first two DGC races, the main objective of the mapping support for every team was to provide reliable geospatial data in the waypoint-defined corridors for the path planning during and prior to the race. In theory, an accurate terrain model combined with thematic information and the description of all natural and man-made objects would be sufficient for the mapping requirements. In contrast, the DARPA Urban Challenge requires practically no terrain data information, as the vehicles are confined to a road network, which is well defined by the race organizers. In the light of these new requirements, the mapping efforts should support the following tasks:

- Waypoint generation based on the RNDF data,
- Providing tools to manipulate RNDF files, including editing/creating/visualization,
- Create a GIS database for field deployment that can provide all publicly available data for analysis, including orthoimage backdrop, slope analysis, etc.

Until now most of the R&D efforts have been devoted to the first two tasks. The waypoint generation module that provides the driving information for the low-level vehicle control based on the RNDF file has been completed and tested. A toolbox to manipulate RNDF files has been developed, including standalone editing and visualization components. The visualization is currently accomplished by Google Earth technology, using the kml protocol; points with long/lat information can be directly visualized in Google Earth.

The development of the GIS database is based on the earlier system design and implementation used in the first two DGCs [25]. The core component of the system is the ENVI Remote Sensing Exploitation Platform, an excellent environment for the visualization, analysis, and presentation of all types of digital imagery, including advanced yet easy-to-use tools, geometric correction, terrain analysis, raster and vector GIS capabilities, extensive support for images from a wide variety of sources. The previous RDDF interface has been updated to the new RNDF, the import is completed the export is in the works. The editing capabilities are still under investigations and will be developed as decisions are made.



Figure 18. Waypoint and corridor visualization using USGS DOQQ imagery in ENVI.

References:

- [1] Q. Chen, U. Ozguner; K. Redmill; "Ohio State University at the 2004 DARPA Grand Challenge: Developing a Completely Autonomous Vehicle", *Intelligent Systems, IEEE*, Volume: 19, Issue: 5, Sept.-Oct. 2004, Pages:8 – 1.
- [2] Q. Chen and U. Ozguner, "Intelligent Off-road Navigation Algorithms and Strategies of Team Desert Buckeyes in the DARPA Grand Challenge '05," in *Journal of Field Robotics*, Volume 23, Issue 9, pp. 729-743, Sept. 2006.
- [3] K. Redmill and Ü. Özgüner, "The Ohio State University automated highway system demonstration vehicle," (*SAE Transactions 1997*): *Journal of Passenger Cars*, sp-1332, Society of Automotive Engrs., 1999.
- [4] K. Redmill, J. I. Martin, U. Ozguner, "Sensing and Sensor Fusion for the 2005 Desert Buckeyes DARPA Grand Challenge Offroad Autonomous Vehicle", *Intelligent Vehicles Symposium*, 2006 IEEE, 13-15 June 2006 Page(s):528 - 533
- [5] Bernard A. Galler and Michael J. Fisher. "An Improved Equivalence Algorithm." *Communications of the ACM*, Volume 7, Issue 5 (May 1964), pages 301-303.
- [6] T.R. Benedict and G.W. Bordner "Synthesis of an optimal set of radar track-while-scan smoothing equations," *IRE Trans Automatic Control* vol AC-7 pp 27-32 July 1962
- [7] R. Jonker, A. Volgenant "A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems," *Computing* 38, 325-340, 1987
- [8] Robert MacLachlan "Tracking Moving Objects from a Moving Vehicle Using a Laser Scanner" Carnegie Mellon Technical Report CMU-RI_TR-0507 2005
- [9] Catmull, E. E., Rom, R.J., "A class of local interpolating splines", *Computer Aided Geometric Design*, R. E. Barnhill and R. F. Riesenfeld, Eds. Academic Press, Orlando, 1974, pp. 317-326.
- [10] Barry P., Goldsman, R., "A Recursive Evaluation Algorithm for a Class of Catmull-Rom Splines", *Computer Graphics*, (SIGGRAPH '88), vol. 22, No. 4, 1988, pp. 199-204.
- [11] Wang Y., Shen D., Eam Khwang Teoh, "Lane Detection Using Catmull-Rom Spline", 1998 IEEE International Conference on Intelligent Vehicles.
- [12] Schreiber, D., Alefs, B., Clabian, M., "Single camera lane detection and tracking", *Proceedings. 2005 IEEE Intelligent Transportation Systems*, 2005.

- [13] Sunghoon Kim, S. Park, and K.-H. Choi, “Extracting Road Boundary for Autonomous Vehicles via Edge Analysis”, Signal and Image Processing SIP 2006, July 2006, Honolulu, Hawaii, USA
- [14] Keith A. Redmill, Srinivasa Upadhyaya, Ashok Krishnamurthy, Umit Ozguner, “A Lane Tracking System for Intelligent Vehicle Applications,” IEEE ITSC 2001, Oakland, CA, 25-29 August 2001, p. 273-279.
- [15] Haines, E., “Point in Polygon Strategies”, Graphics Gems IV, ed. Paul Heckbert, Academic Press, p. 24-46, 1994.
- [16] Shamos, M.I., Preparata, F.P., “The standard but technically complex work on geometric algorithms”, Computational Geometry, Springer-Verlag, Berlin., 1985.
- [17] O'Rourke, J., “Computational Geometry in C (2nd Edition)”, Section 7.4, “Point in Polygon”, 1998
- [18] Russell S. and Norving P., *Artificial Intelligence: A Modern Approach*, 2nd Edition, Prentice Hall, 2003.
- [19] Dechter R., Pearl J., Generalized best first search strategies and optimality of A*, Journal of the association for computing Machinery, 32(3): 505-536, 1985.
- [20] Hatipoglu, C.; Ozguner, U.; Redmill, “Automated lane change controller design,” K.A.; Intelligent Transportation Systems, IEEE Transactions on Volume 4, Issue 1, March 2003 Page(s):13 – 22.
- [21] C. Hatipoglu, U.Ozguner and M. Sommerville, "Longitudinal Headway Control of Autonomous Vehicles," Proceedings of the 1996 IEEE International Conference on Control Applications, September 1996, Dearborn MI, pp 721-726.
- [22] Keith A. Redmill, Takeshi Kitajima, Umit Ozguner, “DGSP/INS Integrated Positioning for Control of Automated Vehicles”, IEEE ITSC 2001, Oakland, CA, 25-29 August 2001, p. 172-178.
- [23] Jia Lei, Keith Redmill and Umit Ozguner, "VATSIM, a simulator for vehicles and traffic," Proc. IEEE ITSC 2001, Oakland, CA, August 2001, p. 686-691.
- [24] C. Toth, E. Paska, Qi Chen; Yongjie Zhu; Redmill, K.; Ozguner, U.; “Mapping Support for the OSU DARPA Grand Challenge Vehicle” Intelligent Transportation Systems, 2006. Proceedings. 2006 IEEE, Sept. 17-20, 2006 Page(s):1580 - 1585

■