Creating a High-Level Control Module for an Autonomous Mobile Robot Operating in an Urban Environment

Ryan Madigan
Advisor: Peter Stone
The University of Texas at Austin
April 2007

Abstract

The DARPA Urban Challenge is a government-sponsored competition scheduled for November 2007. The challenge is to create a fully autonomous vehicle built upon a street-legal frame that can navigate an urban driving environment. Naturally, urban driving environments require the vehicle to obey traffic laws, interact properly with other vehicles on the road, and generally drive much like a human would. This functionality requires a good amount of sophisticated decision making that was not necessary to complete the 2005 DARPA Grand Challenge. This paper outlines our strategy for implementing a control module to effectively maintain awareness of our vehicle's high-level situation and to ensure that the vehicle is working toward completing its mission while following the rules of the road.

Introduction

The DARPA Grand Challenge, a government-sponsored competition to navigate an extensive desert course without any human intervention, urged computer scientists worldwide to solve the problem of autonomous off-road driving. This necessitated the creation of new control techniques to handle associated sub-problems including road navigation and high-speed obstacle avoidance. A typical approach to robot control for this competition involved the separation of overall functionality into special-purpose control modules overseeing individual levels of the robot's interaction with the world. For the Grand Challenge, the DARPA-provided Route Data Definition File (RDDF) mandated a specific series of waypoints defined as latitude/longitude coordinates that were to be traversed in the specified order. This meant that the entire course could be thought of as one long line that needed to be followed while staying on the road.

The follow-up to the Grand Challenge, the 2007 DARPA Urban Challenge (DUC 07,) moves the competition out of the desert and into an urban environment. Thus, participating vehicles must now obey traffic laws, interact with other vehicles on the road, and navigate a network of interconnected roads and parking lots. These additional requirements for success in the competition create a need for high-level situational awareness, route planning, and decision making. Instead of a single long road segment, the provided Route Network Definition File (RNDF) defines an area in terms of segments, lanes, and zones, and also provides more detailed information such as the location of stop signs and the lane markings alongside the road [1]. This a priori knowledge of the world must be integrated with the robot's local interpretation of the world in order to successfully make high-level control decisions.

For the DUC 07, The University of Texas at Austin has formed a partnership with Austin Robot Technology (ART) to enter the competition. Our vehicle, named Marvin, is pictured in figure 1. Marvin's control software, utilized previously for the 2005 Grand Challenge, is now being reconfigured to handle the additional complexity of the DUC.

Marvin's high-level control, path planning, and decision-making module has been named Commander. We detail the design and implementation of Commander as well as its interaction with other control modules. Additionally, we perform an initial evaluation of the effectiveness and practicality of our Commander module and its potential for success in the actual DUC 07 competition. Is Commander sufficiently fast, expandable, and sensible to effectively compete in the DUC 07?

To answer this question, we evaluate each of the three criteria individually. To assess Commander's sensibility, we have tested Marvin's (and thus Commander's) ability to demonstrate the required evaluation factors for the DUC 07 Track B Video Demonstration [2]. We analyze Commander's future expandability by evaluating the internal construction of Commander itself. Finally, we evaluate Commander's speed by ensuring that it is fast enough to keep the system updated on the high-level situation of the vehicle and that it is always meeting its 20 Hz real-time deadline.

While conducting this research, I have been a part of a larger team with the overall goal of creating a complete system with the potential to win the DUC 07 by implementing a never-before-achieved set of autonomous behaviors. Thus, much of the work for the project is not my own and has been a collaborative group effort. Regardless, I have made significant contributions of my own, including the creation of Commander, the primary focus of this paper. I further elaborate on my own contributions later in this paper.

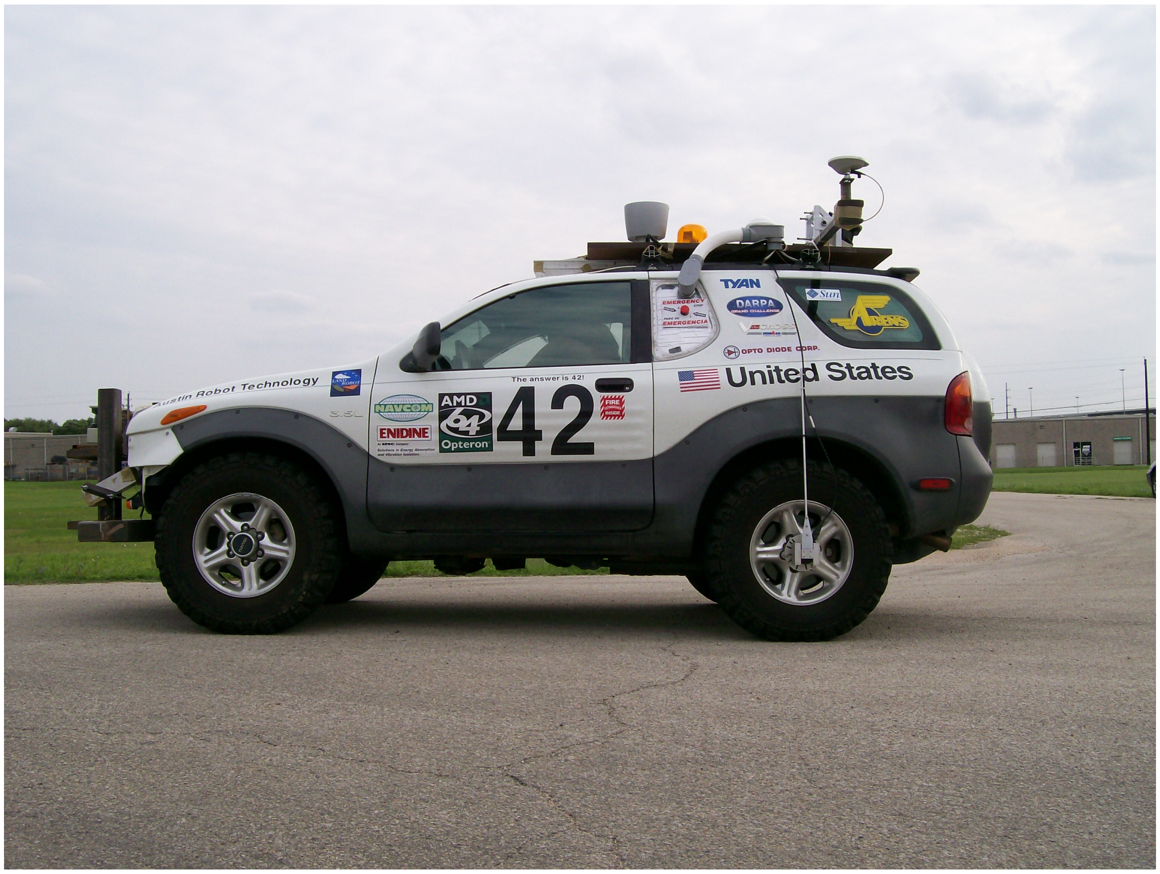

Figure 1. Marvin. Austin Robot Technology entered Marvin in the 2005 Grand Challenge and made it to the semi-finals. ART and the University of Texas together hope to make Marvin a serious contender in 2007.

Marvin's Control Modules

Marvin's software package consists of three primary control modules: Commander, Navigator, and Pilot. Figure 2 outlines a high-level picture of Marvin's overall software architecture. The modules are defined by their relative level of reasoning about the world and the car's position in it.

The Pilot is the closest module to the physical hardware of the car. Pilot is responsible for converting a desired speed and heading into appropriate throttle, brake, and steering corrections through adjustments of the car's actuators. As such, Pilot essentially acts as a single interface between our software controllers and our hardware.

The Navigator fills a crucial middle ground between high-level reasoning and low-level control. The Navigator accepts orders from Commander and combines this with a local sensory understanding of the vehicle's surroundings to decide on an appropriate speed and heading for a given situation. Navigator implements a collection of behaviors such as "Follow Road," "Stop at Intersection," and "Merge Left," that Commander selects from when sending orders. For example, if Commander gives a general "Follow Road" instruction, Navigator will use information from the world model to maintain the current lane of travel while simultaneously performing any minor obstacle avoidance within the current lane. Alternatively, if we see that the lane ahead is blocked, Navigator will recognize that the vehicle cannot effectively dodge the upcoming obstacle while maintaining its current lane of travel and update Commander on our newfound situation. Commander will then solve the problem at a higher level and issue updated orders to Navigator.

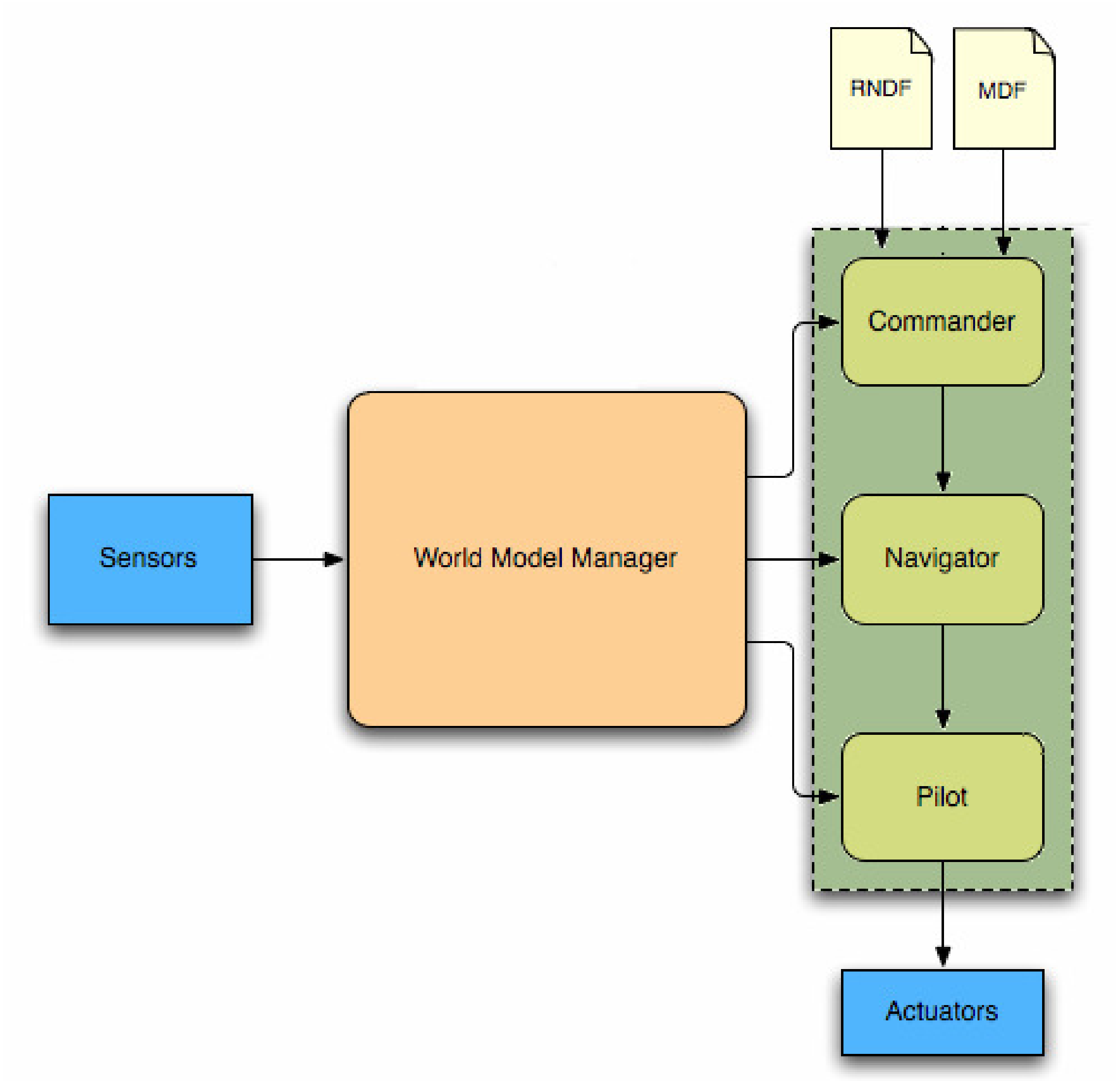

Figure 2. High-level diagram of Marvin's software architecture.

The Commander module operates at the highest level of reasoning in the robot's world. Commander is the only module that deals directly with the RNDF and Mission Data Files (MDF) describing the current mission. As a result, it is the only module that analyzes its surroundings beyond the vehicle's local sensory view. Navigator and Pilot, then, act as implementers of Commander's plan. Commander's primary tasks include planning an optimal route from the robot's current location to the desired goal, maintaining an awareness of the robot's current high-level situation, and sending the resulting desired short-term actions to Navigator as instructions.

RNDF and MDF in the Urban Challenge

An RNDF describes an area as a collection of Segments and Zones, where Segments can be thought of as individual roads and Zones are essentially equivalent to parking lots. Any Segment can be broken down into one or more Lanes, which each have an implicitly defined direction of travel (i.e. there are no two-way lanes). Each Lane is made up of one or more Waypoints, which are defined by their latitude/longitude coordinates. Any Waypoint can also be defined as a Checkpoint, which gives an associated MDF the ability to mandate that the vehicle pass over it at some point in the mission. Transitions from one Segment or Zone to another are defined as explicit one-way Exits between two individual Waypoints. An example of this would be a four-way intersection at which one could continue along the same Segment by driving straight through the intersection or "exit" to another Segment by turning right or left.

As a result, the RNDF creates a high-level topological map of a particular area. Commander can then use this topological map as a resource for path planning and decision making. Additional information found in the RNDF is detailed in [1].

An MDF specifies a mission as a series of Checkpoints that must be physically crossed over by the vehicle in a particular order. An MDF also defines minimum and maximum speed limits for every Segment and Zone in the RNDF. Thus, a mission is a pairing of an RNDF with an associated MDF. Any number of MDF's can be coupled with a single RNDF.

Commander in More Detail

Path Planning: Commander's first priority is to plan a route from the robot's current position to the goal position. This is currently implemented as an A* search in which every Waypoint in the RNDF acts as a node and the exits between them act as edges. The start position for the search is defined as the last Waypoint that the vehicle was near and the goal position is defined as the next Checkpoint specified in the MDF. The search heuristic is Euclidean distance between the current position and the goal position. Since this heuristic never overestimates the true cost of expanding an edge, we are guaranteed to plan an optimal path. For efficiency, the route is only re-planned when the vehicle has visited one of the nodes along the planned path or when the vehicle's situation has changed. Situation changes can be thought of as events that significantly alter the desired behavior of the vehicle, such as the realization that the vehicle's lane is blocked ahead.

Since our real world has the potential to differ from the RNDF, Commander must be able to make adjustments to the RNDF structure as the robot explores a path so as to not make the same mistake multiple times. This can be handled within Commander by removing the Exit connecting the two waypoints between which there is a road blockage and re-planning with the resulting topological map. Additionally, if we can decide that the blockage is temporary (such as a stalled vehicle,) Commander can re-add the blocked Exit to our map, leaving the potential for us to try to navigate the road again in the future.

Behavior Selection: Another of Commander's responsibilities is the selection of a behavior for Navigator to follow. Example behaviors include Follow Road, Turn Around, Stop At Line, and Park. A more complete list of behaviors is presented in Figure 3.

The behavior selected is a function of Commander's state and the high-level situation of the vehicle. Commander's states currently include the following:

1. Road – Normal operation. Road following.
2. Lane Blocked – Our current travel lane is blocked ahead.
3. Dodging Blocked Lane – We are in the process of avoiding a blocked lane.
4. Road Blocked – The entire road is blocked.

The number of states will continue to increase as functionality is added to the vehicle. As mentioned in [3], a finite state machine approach to a high-level control module has advantages and disadvantages. Advantages include high predictability and reliability, as well as high performance since state transitions are well defined and generally easy to compute. The primary disadvantage is an inability to handle situations that were not defined as part of the state machine.

As an example of Commander's internal state shifting and behavior selection, consider the car driving down the road under normal circumstances. Commander's state is Road. It is issuing orders to Navigator that contain the Follow Road behavior. As the vehicle approaches a four-way intersection with no other cars present, Commander recognizes that our imminent waypoint has a stop sign denoted in the RNDF and issues a Stop At Line order to Navigator, which then interprets the information from the camera inputs to effectively stop the vehicle near the stop line. Recognizing that the stop has been performed, Commander then issues orders with the Follow Road behavior to clear the intersection and continue down the road.

At some point, let us say there is a stalled car blocking the vehicle's lane of travel. Upon recognizing the lane blockage, Commander changes its own state to Lane Blocked and begins issuing orders with the Stop Now behavior to stop the vehicle as fast as possible. Once the vehicle has stopped for an appropriate amount of time, Commander will change its own state to Dodging Blocked Lane and plan an alternate route through another available lane. Commander will then begin issuing orders with the Follow Road behavior and specifying waypoints in the alternate lane. Once Commander determines that the obstacle blocking our original lane is safely behind us, it will plan a new route through the original lane of travel and order the Follow Road behavior. Once our vehicle has returned to its original lane of travel, Commander resets its own state back to Road and continues on.

Speed Control: The final responsibility of Commander is to provide Navigator with parameters controlling minimum and maximum speeds. In typical lane-following situations, Commander will pass along the values defined explicitly in the MDF for the current Segment or Zone. For some situations, however, Commander will lower the maximum speed limit to ensure reliable control during higher-precision maneuvers. For example, when the vehicle is performing a maneuver to dodge a stalled vehicle blocking our lane, Commander will ensure that we are not moving faster than 5mph.

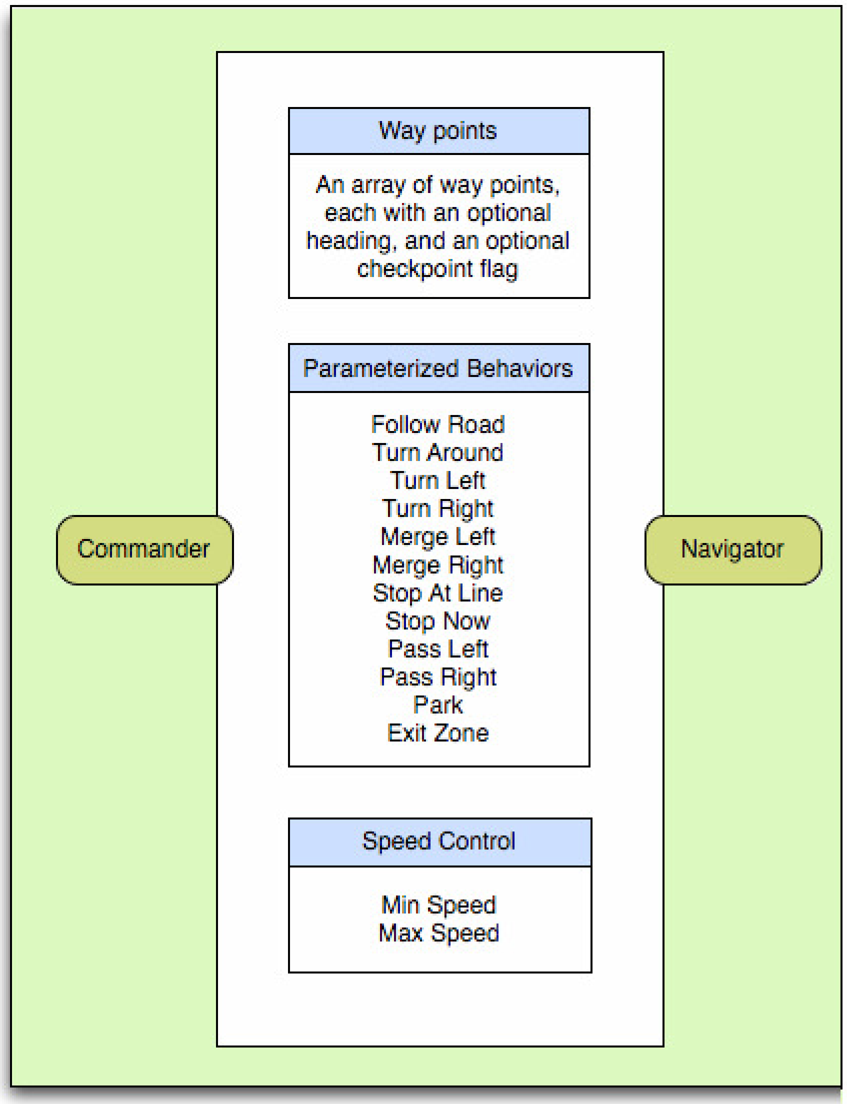

Figure 3. Interface between Commander and Navigator.

Track B Video Demonstration

The video demonstration is a required part of the qualification process for Track B teams participating in the DUC 07 competition. Track B teams include the 78 participating teams who are not receiving funding from DARPA and thus are entirely self-supported. The purpose of the video is to allow DARPA to assess the capability of the vehicle. DARPA then determines whether or not to grant a site visit to the team, where DARPA officials visit to physically watch the vehicle in action. Denial of a site visit marks the end of the team's participation in the DUC 07.

The general setup of the video demonstration is as follows: The vehicle autonomously navigates a circular course composed of a travel lane and a passing lane while staying within the travel lane. At some point along the course, an obstacle vehicle is placed within the travel lane, blocking the autonomous vehicle's path. At this point, the autonomous vehicle must stop, transition into the passing lane to avoid the obstacle vehicle, and then transition back into the travel lane. Figure 4 provides an illustration of this setup.

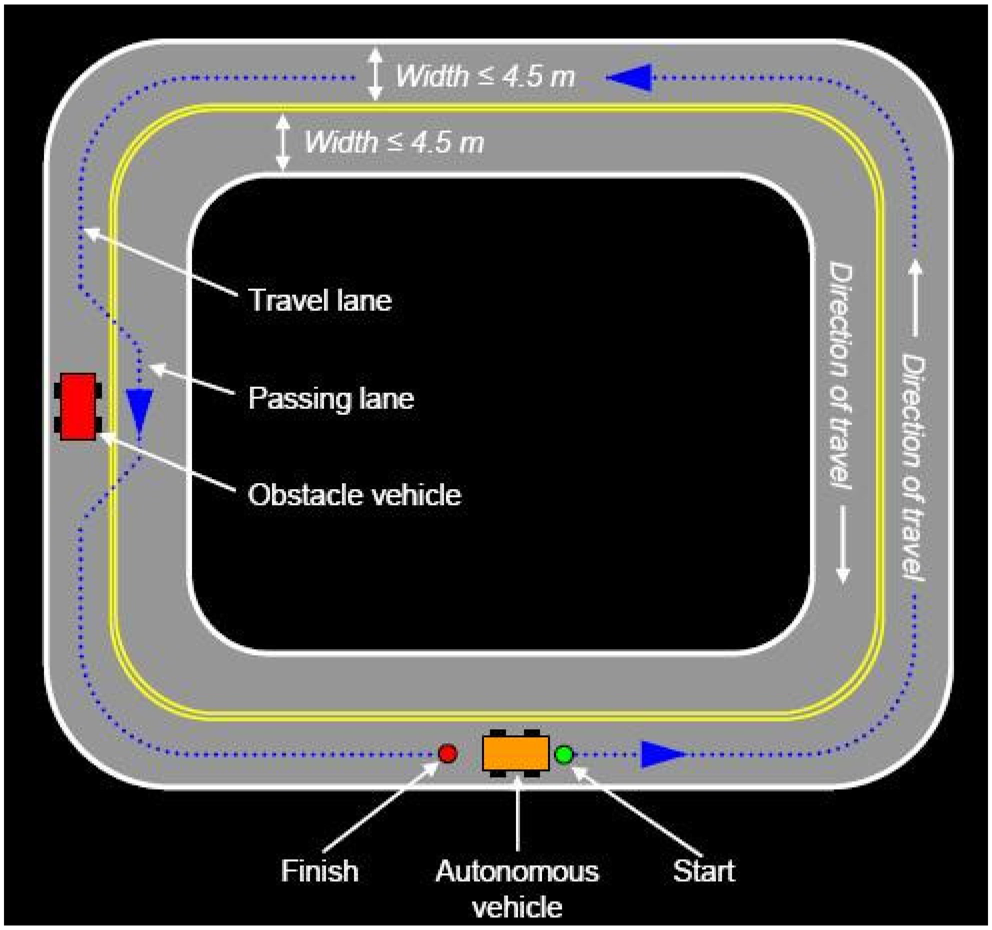

Figure 4. Required video demonstration course as provided by [2].

Of the thirteen evaluation factors for the video demonstration, six of them apply in some way to Commander. Those six evaluation factors, as described in [2], are as follows:

1. Vehicle navigates all course waypoints autonomously.
2. Vehicle is able to stay within travel lane, with wheels of vehicle remaining completely within marked boundaries at all times while traversing the course.
3. Vehicle completes the course at an average speed within the 5-15 mph target speed range. Vehicle progresses smoothly during straight sections of the course and controls speed by slowing during turns, if necessary.
4. Vehicle detects obstacle vehicle at a safe distance and comes to a full stop.

5. Vehicle pulls completely into adjacent passing lane, and maintains safe distance from obstacle during maneuver.
6. Vehicle returns safely to travel lane a safe distance from obstacle and continues without leaving travel lane for the remainder of the course.

Evaluation

We completed filming our video demonstration on Sunday April 1st, 2007 at Southwest Research Institute in San Antonio, Texas. All required evaluation criteria were demonstrated. Given the difficult nature of testing extensively on the vehicle itself, we consider the completion of these tasks to be our main indication of a successful approach to the creation of a high-level control module for the DUC 07 event. Granted, it would be possible to design a specialized system from the outset to complete these specific tasks, but the intent is for Commander to continue to be augmented with additional decision-making and planning capabilities in the months leading up to the DUC 07.

The three outputs of Commander (path planning, behavior selection, and speed control) are easily expandable to produce increasing levels of competence within the world. Like the layered approach detailed by [4], each expected level of competence is thoroughly tested and debugged before other levels are added. Unlike the layered approach, however, we expand our new functionality into the same Commander module, not an entirely new layer on top of Commander.

The advantage of this approach is that the Commander interface remains the same even as behaviors are implemented below it in Navigator. This also allows for a much more intuitive distinction between the implementation of high-level decision making in Commander, behavior implementation and interpretation in Navigator, and low-level control in Pilot. With a strictly layered approach, it would be unclear which capabilities are "higher-level" than others. For instance, the actions of passing a stalled vehicle and proceeding through a four-way intersection are both high-level problems, but it is unclear which might be "higher" than the other, and it makes more intuitive sense to keep them on the same logical level of control.

Commander's expandability stems from the fact that its interface to Navigator can remain the same as competency is added over time. High-level control logic is implemented in Commander that reasons about the vehicle's current state and situation and selects an appropriate behavior for Navigator to follow. These specific behaviors are then implemented in Navigator as the logic is written for Commander. This allows for a gradually expanding robot competency without the accumulation of interfaces every time new abilities are added to the system.

Commander's fast execution is a result of its underlying finite state machine implementation. State transitions are triggered by simple Boolean expressions representing situations encountered by the vehicle. The route adjustment itself is inexpensive computationally since it only requires small modifications to the RNDF structure and an execution of the A* path planner. A performance graph of Commander's execution time is presented in Figure 5.

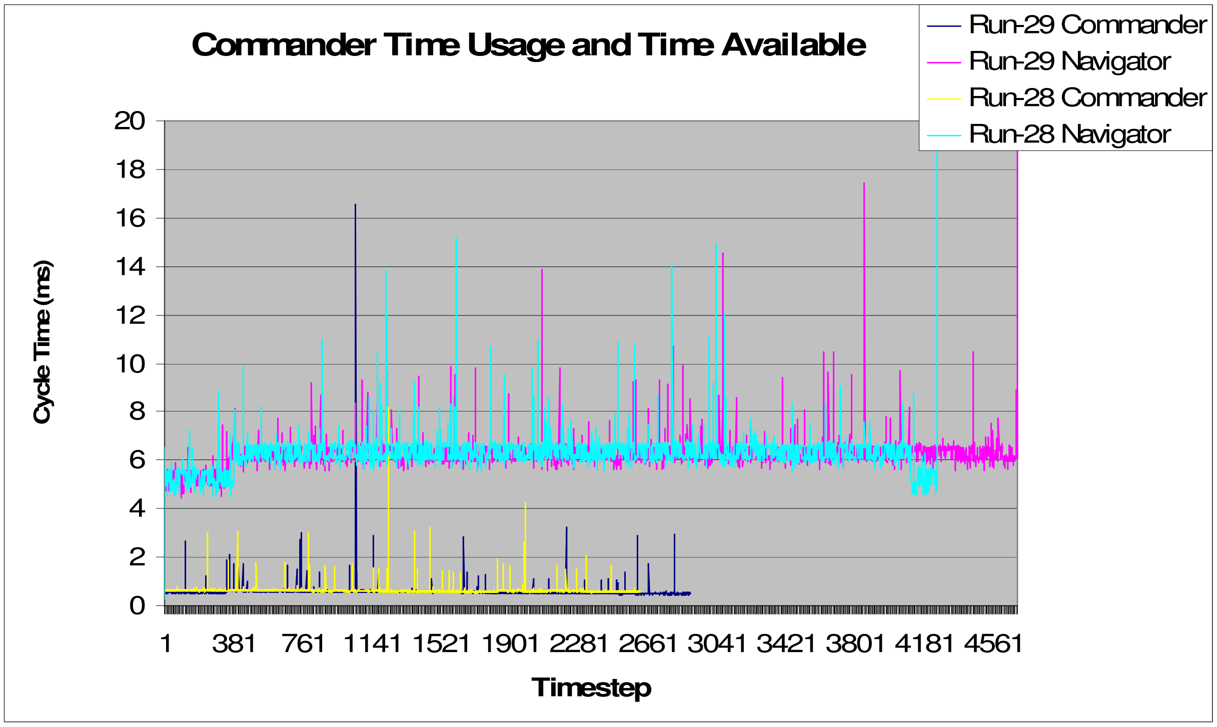

Figure 5. Performance graph comparing Commander to Navigator.

Analyzing Figure 5, we can confirm several required evaluation criteria. To clarify the graph, the dark blue and yellow lines represent the execution time of Commander for two consecutive test runs of our vehicle. The turquoise and pink lines represent the execution time of Navigator for the same two test runs. We can initially see that Commander's state machine implementation allows it to operate very quickly under the common case, since most control cycles will not involve a state change and thus will require very little processing. Most Commander control cycles were completed in under 1 ms of execution.

First, we want to ensure that Commander is almost always executing faster than Navigator. This is an important property to satisfy because we want to ensure that Navigator is always operating with up-to-date situational information from Commander, lest the vehicle might be left performing a behavior that does not make sense given its high-level situation. Each test run contains one timestep for which the Commander execution time shoots above the corresponding Navigator execution time, but this is likely due to random processor scheduling. Given its rare and temporary nature, the spike is not a problem, and therefore this evaluation criteria is satisfied.

Second, we need to verify that Commander is not spiking its execution time above 50 ms. This property is a result of Commander's real-time operation at 20 Hz. If its execution time were ever to spike above 50 ms, it would not be meeting its real-time requirement. Evaluation of this graph shows that even the most extreme spike in execution time was below 18 ms, meaning that Commander is consistently operating well within its 50 ms requirement.

As a result, we can conclude that Commander does provide a fast, expandable, sensible base to expand upon for the creation of a high-level control module for use in the actual DUC 07 event.

My Contributions

My involvement with the project has spanned from overall architecture design to tool development to writing control code for the robot itself. My specific contributions are detailed here.

<u>RNDF and MDF Parser</u>: A parser is required to read input from specified RNDF and MDF files and store the information in an easily accessible data structure. I constructed this data structure and designed a collection of supporting functions to make the information more directly useful to control modules on the car and external visualization tools. The parser is entirely composed of my own code.

<u>Software Architecture Design</u>: I participated heavily in the discussions surrounding Marvin's overall software architecture. The architecture described in this paper was chosen because of its separation of logical functionality into separate control modules as well as its compatibility with the previous architecture that ART had laid out. Other primary participants included Tarun Nimmagadda and Mickey Ristroph.

<u>Commander</u>: The Commander module is composed entirely of my own code, with the exception of the message-passing code used to communicate back and forth with Navigator. The messaging code was written by Jack O'Quin of ART.

<u>Navigator</u>: The Navigator module is also entirely composed of my own code with the exception of the message-passing code and the state machine written within Navigator to allow for pause/run functionality. The messaging code was again written by Jack O'Quin of ART, as was the pause/run state machine.

<u>VisualCommander</u>: It became obvious that the visualization of RNDF files would be useful for the verification and creation of said files. Using a specialized OpenGL interface written by Tarun Nimmagadda, I created a tool that draws the segments, zones, stop signs, and checkpoints of an RNDF to the screen. This tool has been useful for visually examining an RNDF before actually running it on the car, as well as for comparing multiple RNDF's together to visualize differences such as GPS drift over time. A screenshot of VisualCommander is presented in Figure 6.

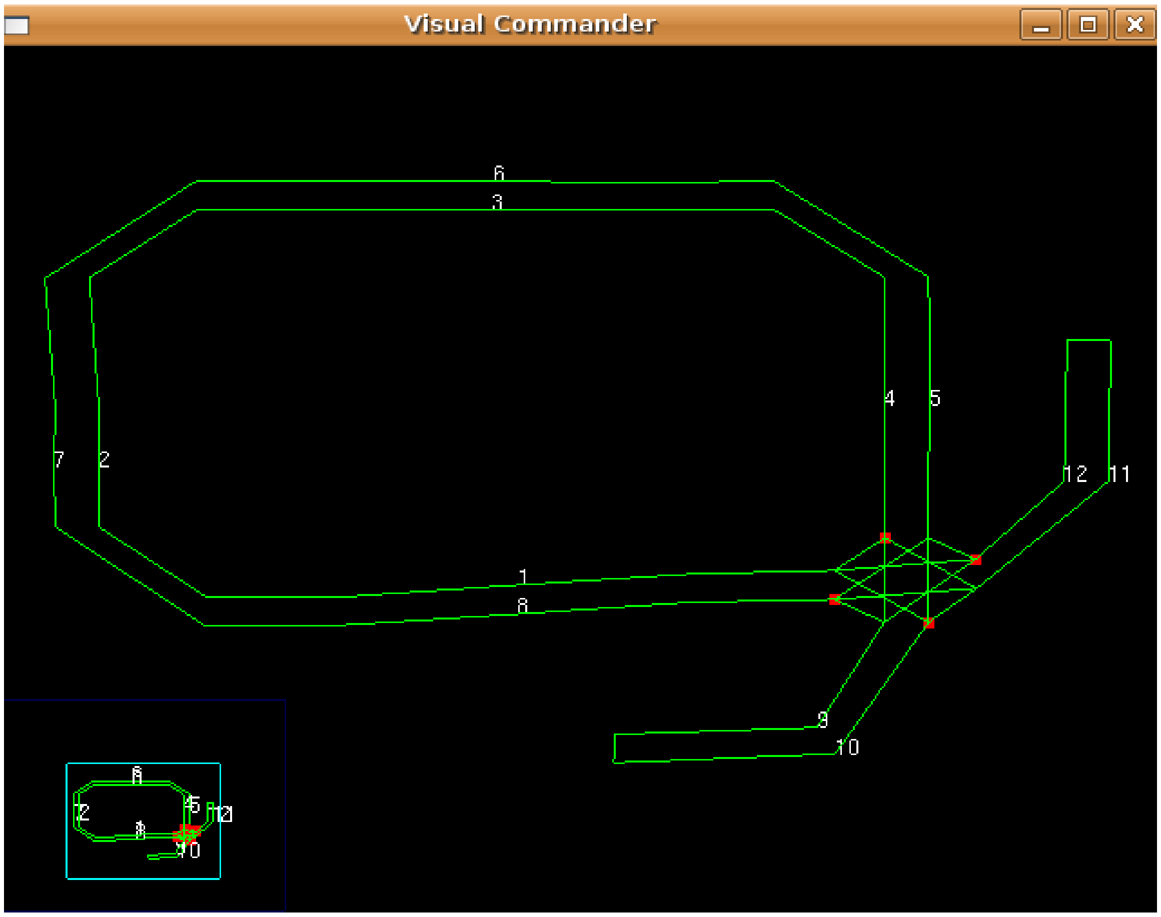

Figure 6. VisualCommander displaying an RNDF. Green lines represent edges between nodes within the route network. Red dots denote stop signs. White numbers represent checkpoints within the route network.

Related Work

Another approach to a mission management module has been implemented in Caltech's Supervisory Controller, or SuperCon. SuperCon acts as an overbearing control module that also doubles as a sanity-checking messenger between other modules. All inter-module communication seems to take place through SuperCon, which monitors the messages to make sure that the robot's goals will eventually be attained [3]. It seems similar to our Commander in that it must have a clear understanding of the high-level situation of the vehicle in order to properly sanity-check the state of individual modules. The similarities seem to end there, however, as our Commander does not act as a messenger between modules, and instead focuses only on high-level path planning and decision making.

Our architecture bears some resemblance to Stanford's software architecture for Stanley, the winner of the 2005 DARPA Grand Challenge. Like Stanford, we use a modified three-layer architecture, except that our planning layer is essentially broken down into three sub-layers of its own in Commander, Navigator, and Pilot. Also similar to Stanford, we implement the majority of our communication between modules with a publish-subscribe system so that direct synchronization between modules is not necessary [5].

We have also found the testing methods developed by Carnegie Mellon University's Red Team to be an effective way of consistently and incrementally testing the vehicle's abilities [6]. The blind path-tracking test has been useful for initially testing pose estimation as well as throttle, brake, and steering controllers before moving on to more advanced tests. The perception-tracking test provides a way to test sensor inputs, occupancy grid construction, and basic obstacle avoidance. Finally, the perception-planning test can be executed to assess planning, decision making, and high-level control capabilities.

Future Work

There is clearly much work to be done before the DUC event in November of this year. Commander needs to be able to distinguish between car and non-car objects in order to reason about the actions other cars are likely to take. This distinction is also necessary in order to establish precedence at a four-way intersection and proceed at the appropriate time. Another key addition along the same lines is the ability to pass a slow-moving car in the vehicle's lane of travel. The vehicle needs to be able to identify the slow-moving object as a car, determine that an adjacent lane is clear, and perform the passing maneuver as safely and efficiently as possible.

Commander also needs to be able to determine if the road the vehicle is traveling on becomes entirely blocked. When this happens, the vehicle needs to be able to dynamically adjust the RNDF structure and re-plan a route without using the segment of road that is blocked. Commander is currently capable of performing this maneuver if the vehicle's lane of travel becomes blocked, but not necessarily if the entire road is untraversable.

On a shorter timeframe, I would like to enhance Commander to be able to lower the maximum speed communicated to Navigator to suit the situation. For example, it would be desirable for Commander to be able to distinguish between situations where the vehicle is driving down a straight segment of road and where the vehicle is approaching a sharp turn. After making this distinction, Commander could begin to lower the maximum speed as the vehicle approaches the turn, resulting in a smooth speed transition. This kind of speed control will be necessary for the DUC event as it will be necessary to achieve higher speeds on straight segments of road in order to reasonably expect to finish the race on time.

Conclusion

Successful completion of the Urban Challenge in 2007 would represent a major milestone in the development of robotic systems capable of interacting with humans in a real-world environment. The introduction of fully autonomous passenger vehicles would likely lead to the reduction of accidents caused by driver error and easier transportation for disabled people, among many other benefits.

The Urban Challenge brings to the forefront a variety of high-level problems that were not present for the previous Grand Challenge. Autonomous vehicles must now have a way of maintaining a constant situational awareness in order to properly interact with other vehicles and deal with typical urban driving problems. Decisions must be made on the fly that could drastically influence the driving behavior of the car – imagine the difference between a passing behavior and a parking behavior. A plan must be established and constantly updated to ensure that the vehicle is working toward mission completion.

We have outlined an approach to the creation of Commander, a control module capable of handling these high-level needs of an autonomous vehicle in an urban environment. We believe Commander provides an acceptable base from which to expand functionality progressively and procedurally. More generally, the separation of functionality between Commander, Navigator, and Pilot allows for a modular control structure divided upon logical decision distinctions. We expect this approach to our software control architecture to allow for more natural division of tasks between modules and easier development as competency is added to the vehicle.

Acknowledgments

I want to thank my advisor Dr. Peter Stone for his guidance and support of the Urban Challenge project here at UT. I would also like to thank Jack O'Quin for teaching me the existing software systems of Marvin and the specific needs of a real-time system, as well as Austin Robot Technology as a whole for actually building Marvin. Finally, without the dedication of Richard Edwards and several others in the CS 378 class this work would not have been possible.

References

[1] DARPA, “Route Network Definition File (RNDF) and Mission Data File (MDF) Formats,” 2007, Available: http://www.darpa.mil/grandchallenge/docs/RNDF_MDF_Formats_031407.pdf

[2] DARPA, “Video Demonstration Guidelines,” 2007, Available: http://www.darpa.mil/grandchallenge/docs/Video_Demonstration_Guidelines_022007.pdf

[3] J. Feingold and R. Murray, “Contingency Management of Autonomous Vehicles in Urban Environments: Supervisory Control as an Approach to Goal Management,” 2006, Available: http://grandchallenge.caltech.edu/media/papers/feingold-surf06.pdf

[4] Rodney A. Brooks, “A Robust Layered Control System for a Mobile Robot,” 1986, Available: http://ieeexplore.ieee.org/iel5/8149/23631/01087032.pdf?tp=&arnumber=1087032&isnumber=23631

[5] Sebastian Thrun et al., “Stanley: The Robot that Won the DARPA Grand Challenge,” 2006, Available: http://robots.stanford.edu/papers/thrun.stanley05.pdf

[6] Chris Urmson et al., “Testing Driver Skill for High-Speed Autonomous Vehicles,” 2006, Available: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?isnumber=4039225&arnumber=4039245&count=41&index=16