Robust Autonomous Vehicles

DARPA Urban Challenge

1 June 2007

University of Utah

Thomas C. Henderson, Mark Minor, Sam Drake, Andy Hetrick, Jacob Quist, Jared Roberts, Hamid Sani, Ramya Bandaru, Marques Rasmussen, Adam Collins, Yu Sun, Suraj, Xiuyi Fan, Ed St. Louis, Shigenori Mikuriya, Ken Dean

University of Utah, Salt Lake City, UT 84112

Corresponding Author: tch@cs.utah.edu

Abstract

Autonomous vehicles are complex systems with many interacting hardware and software components operating in an uncertain and dynamic environment. Organizational principles and procedures are described which help assure reliable and intelligent actions on the part of the vehicle. This includes both high-level system models, as well as process level monitoring and testing to verify and validate the system components on the fly. We propose a high-level model based on a probabilistic characterization of the inputs and outputs (or other observable elements) of the modules, and for individual components, we propose to exploit Instrumented Logical Sensors. These methodologies are to be demonstrated in the context of the autonomous vehicle.

1.0 Introduction and Overview

The 2007 DARPA Urban Challenge (DUC) is a competition that requires driverless cars to navigate through an urban environment. The 2007 DUC is a continuation of the 2005 DARPA Grand Challenge in which autonomous vehicles navigated through an off-road environment.

Autonomous vehicle research has two major directions: (1) intelligent highway and transportation systems, and (2) unmanned vehicles for dangerous environments (e.g., underground mines, battlefield, planet exploration). Here we are mainly concerned with the latter, although basic results may be relevant to both.

DARPA has been the major funding source for the research in this area with the Unmanned Ground Vehicle Program in the 1990's. Results fell far short of expectations due mainly to two difficulties: (1) incorporating human-like performance, and (2) robustness. Attempts to address these issues have been made:

- *subsumption architecture*: a reactive system of hardwired responses to sensor values resulted in very robust and even natural looking operation, but lacked the ability to demonstrate human-level performance or even basic predictability of behavior. (See [3].)
- *probabilistic robotics*: sampling methods and Bayesian methods have been used to reduce the complexity of the high-level representations and full joint probability distributions; this approach has demonstrated good performance in a number of highly structured environments (e.g., museum tours, etc.). (See [16].)

However, the major problems with autonomous unmanned vehicles remain unsolved.

DARPA created a Grand Challenge cross country race competition in 2004-2005 to push researchers and industry to solve this problem. A 140-mile race from California to Nevada was held, and several teams accomplished the trip, while the Stanford team won the prize ($2M). DARPA is now running a second competition - the Urban Challenge - in which vehicles must navigate through an urban environment (see [17]). Note that the U.S. Congress has mandated that 33% of all military land vehicles be autonomously

operated by 2015, and 66% by 2025. Since there are over 360,000 U.S. military vehicles with huge markets both in retro-fit packages and new vehicles, a market of $8-10B has been estimated. Thus, there is great pressure to put energy and support into solving the remaining scientific and technological issues to make such vehicles viable.

We propose to initiate new research and development in the area of intelligent autonomous vehicles. The last few years have seen a tremendous increase in interest in this area in terms of scientific approaches as well as implementations for use in real-world scenarios. The underlying scientific issues are:

- **What constitutes intelligence**? Matching human performance in situation understanding and behavior selection is a very difficult problem.
  ➔ We propose to explore the encapsulation of human perception-action modules in terms of multi-agent systems.
- **What is essential for autonomy**? The basic elements include self-selected goals, some form of knowledge representation, some means of communication and some persistence of self state.
  ➔ We propose to develop a rudimentary theory of software/hardware artifact feedback control based on *Software Kalman Filters*. This involves choosing optimal control actions based on system measurements in order to track a desired execution model.
- **What hardware embodiments lend themselves to support intelligent autonomous behavior?** Software and sensing requirements can be minimized with the proper selection of hardware mechanisms.
  ➔ We propose to ultimately explore these issues in various implementations (aerial, ground and marine), but in the context of this work we focus on unmanned ground vehicles.

The general system architecture requires modules for: perception (sensing and information extraction), localization and map building (knowledge database, environment model and local map), cognition and path planning (position, global map and mission commands), and motion control (path execution and obstacle avoidance). For many of these, traditional methods will be exploited; however, the most significant barriers are:
(1) human-like behavior - here we propose to explore a perception/action loop based on a multi-agent framework, and
(2) overall system robustness - for this we will develop the *Software Kalman Filter*.

For the first item, we propose to study a set of cooperating software agents (micro-experts), A_i, each of which produces various outputs using a set of parameters and thresholds, T_i, and each having an associated model (or set of models), $P_i(X_i \mid T_i)$, describing the agent's variance from the ideal in terms of some appropriate measure. Knowledge of three sorts (physical, image analysis, and structural interpretation) is available and informs the agents' actions and understanding of each other's results. Higher-level control processes may exploit this in several ways:

- Explore the threshold space for global optima.

- Control acquisition of new data (e.g., view token generation as state estimation and select agent action that optimizes information gain).
- Incorporate knowledge in abstract form and communicate abstractions between agents and users.
- Inform the software engineering and system development with deep knowledge of the relationships between modules and their parameters (at least in a statistical sense).

The objective of the second item, the *Software Kalman Filter* work, is to develop better models of software systems which allow designers to make crucial knowledge explicit, accessible and persistent, as well as to use such knowledge as the basis for a rigorous, mathematically well-defined operational framework. Stochastic optimization techniques are proposed as the computational framework in which to model and execute complex software systems (see [9] for details). Software design will be based on firm engineering principles with well-identified models, including those for error and noise processes.

1.1 System Architecture

Figure 1 gives a simplified view of the super architecture and how it relates to the Cognition System. The entire system can be viewed as a loop of messages:

(i) The perception module receives data from the sensors and attempts to construct an accurate view of the state of the world and the state of the vehicle.

(ii) The cognition system uses the *World State* and *Vehicle State* to execute its behaviors.

(iii) The cognition systems issues commands to a *Vehicle Control Interface*, which translates those signals to physical actions in the vehicle.

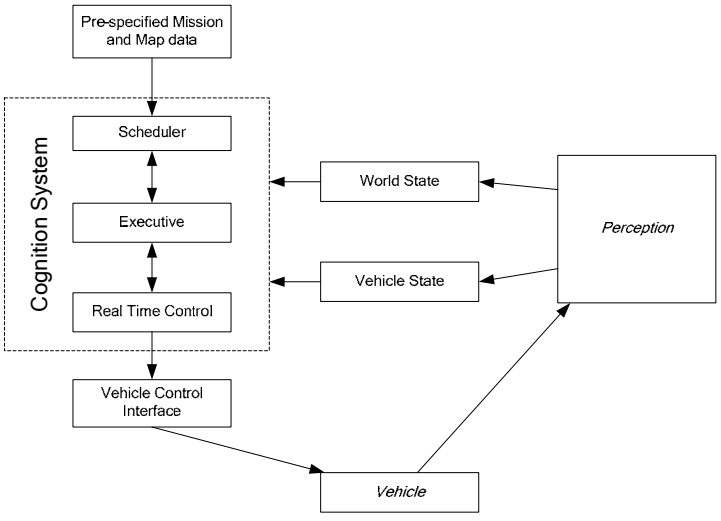

Figure 1. The super architecture and how it relates to the Cognition System.

2.0 Cognition Architecture

The overarching purpose of the cognition system is to navigate the vehicle safely through a pre-specified number of checkpoints. This entails a number of sub-features and behaviors. For example, the system must perceive the road and any obstacles; it must understand how to traverse the stretch of road while avoiding obstacles; it must know how to handle turns and how to pass other vehicles; and it must use its limited perception to figure out its surroundings. These tasks may seem trivial to experienced human drivers, but it requires a great amount of research and mathematical generalization to properly encode into software.

Our system is built to handle urban environments. There is no one-size-fits-all solution to autonomous vehicles. A system that performs well in an off-road environment (such as the 2005 Challenge) will not necessarily perform well in an urban environment. There are different sets of behaviors and rules that the vehicle must follow. We aim to encode these behaviors and build a system that performs well in an urban setting.

The Cognition System is comprised of three critical sub-modules: (1) the *Scheduler*, (2) the *Executive*, and (3) the *Real Time Controller*. These modules run concurrently in a master/slave manner. This means that the upper modules have ultimate control over the lower modules.

The upper modules are responsible for entire scope of the mission, while the lower modules are responsible for immediate tasks. This helps break the project into more manageable pieces.

In addition to the three critical components, we introduce the *Simulator* and *Visualization* modules. These supplementary modules will not be present while the physical vehicle is in operation, but they are used to build, debug, and demonstrate the inner workings of the cognition system.

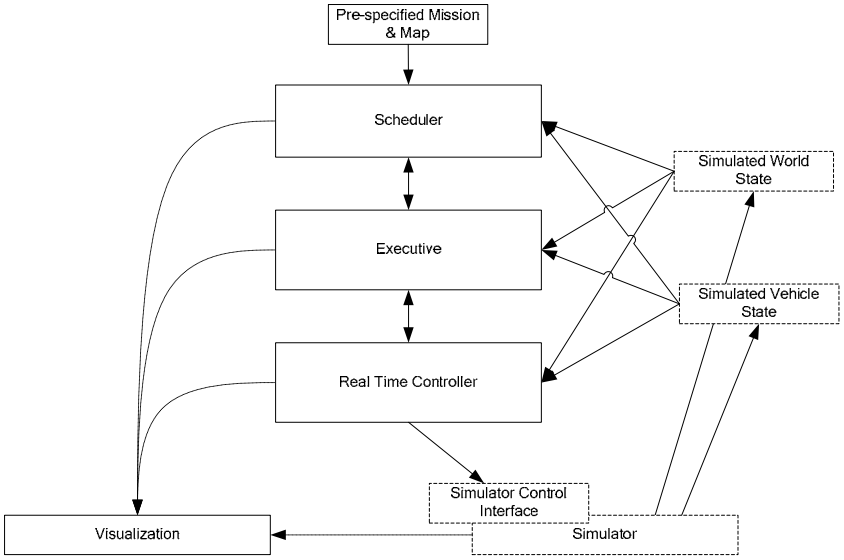

Figure 2. The major components of the Cognition System. The simulator components (in dashed lines) represent the harness in which the Cognition System executes. A Visualization module is also present, the gray lines indicate that the Visualization module is not a critical component.

2.1 Scheduler

The *Scheduler* is the topmost module and is responsible for tasks related to the entire mission. It receives a pre-specified map file and computes an entire-path plan that optimally (shortest path) traverses the pre-specified checkpoints.

Additionally, the *Scheduler* is responsible for breaking down the map into shorter segments called *links*. (Our approach is based on that of Gowdy; see [15].) A *link* is simply a set of two points, usually a straight line, which is given to the *Executive* for further processing. As explained below, there are different types of links, so the *Scheduler* must infer the proper link type from the map.

The Scheduler receives signals from the *Executive*. The *Executive* signals the *Scheduler* when it has finished traversing its delegated link, or when an impassable obstacle is detected. In such a scenario, the *Scheduler* replans the entire mission around the obstacle.

The *Scheduler* is comprised of four major components.

The *Progress Manager* is responsible for keeping track of the progress of the *plan* (see below). It keeps track of which links have been traversed, as well as issuing new links when signaled.

Once the *Progress Manager* decides to issue a new link, it sends a message to the *Link Manager*. The *Link Manager* is responsible for inferring the type of link to be produced. The following table gives a brief description of major link types.

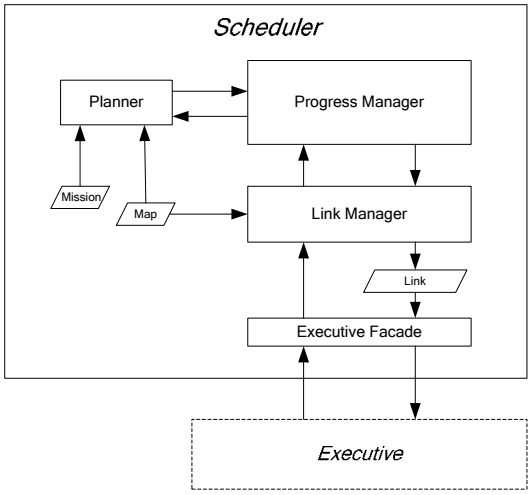

Figure 3. The major components of the Scheduler.

Link Type	Description
Straight-Line	Travel in a straight line
Right-Hand-Turn	Make a right hand turn (at an intersection)
Left-Hand-Turn	Make a left hand turn (at an intersection)
Zone-Exit	Travel to a specified position without regard to lanes in the road (i.e. a blacktop surface without traffic indicators)
U-Turn	Turn around so that the vehicle is in the opposing lane.
Park	Position the vehicle in a parking spot.

Once the appropriate link object has been created, it is sent to the *Executive* through the *Executive Façade*. The *Executive Façade* is simply an interface wrapper that translates messages between the *Executive* and *Scheduler*. A list of major messages is given below:

Message	Direction	Description
New-Link	Scheduler → Executive	Sends a new link object (as described above).
Link-Complete	Scheduler ← Executive	Issued by the Executive when a link has been traversed and the Executive is ready for the next link.
Link-Impassable	Scheduler ← Executive	Issued by the Executive when a link is perceived impassible (i.e. roadblocks)

2.2 Executive

The *Executive* is the middle module and is responsible for navigating the vehicle through a link. It receives links from the *Scheduler* and then computes a route that optimally traverses that link.

The *Executive* will break down the links into smaller segments called *sub-links*. A sub-link is a target point and orientation that is given to the *Real Time Controller* for further processing. The *Executive* must decompose the sub-link well enough so that the *Real Time Controller* does not need to infer anything special about it.

Based on the type of link received from the *Scheduler*, the *Executive* must execute a certain behavior. The simplest case is the *Straight-Line* link. In this scenario, the *Executive* must simply create sub-links in a straight line towards the end point. A more complex link is the *U-Turn* link. In this scenario, the link still has a two-point tuple and the *Executive* must infer the appropriate sequence of sub-links that will take the vehicle to the specified *End Point*.

The *Executive* receives signals from the *Real Time Controller* indicating that it has completed or is unable to complete the assigned path. In the latter case, the *Executive* must deduce whether the obstacle is passable in the current link segment (e.g., if there is an obstacle in the current lane, it decide if it can be passed on the left). If it is passable, then the *Executive* must load a behavior to pass the obstacle. If it is not passable, then the *Executive* must signal the *Scheduler*, whereupon the *Scheduler* will recompute an entire mission path. Otherwise, the *Executive* signals the *Scheduler* when the specified link is traversed and wait for further commands.

The *Executive* is comprised of four major components. These components are described in Figure 4.

As stated above, different types of behaviors are tied to different types of links. These behaviors are loaded into the Progress Manager, which acts as a harness for the current behavior. Each behavior handles links differently. For example, the *Straight-line* behavior will progress differently than the *U-Turn* behavior. Therefore, many of the sub-modules in the Progress Manager act as tools and utilities to help the currently executing behavior.

The Sub-Link Decomposer is specific to the behavior type. But its general purpose is to create sub-links from the parent link constrained to the specific rules of the behavior.

The Executive is the most complex module of the three components. Thus, this design of abstract behaviors will allow other team members to work on behaviors in parallel.

Once the Progress Manager decides to issue a new sub-link, it sends the sub-link object to the Real-Time Controller Interface. The Real-Time Controller interface will in turn send the message to the Real Time Controller. A list of major messages is given below.

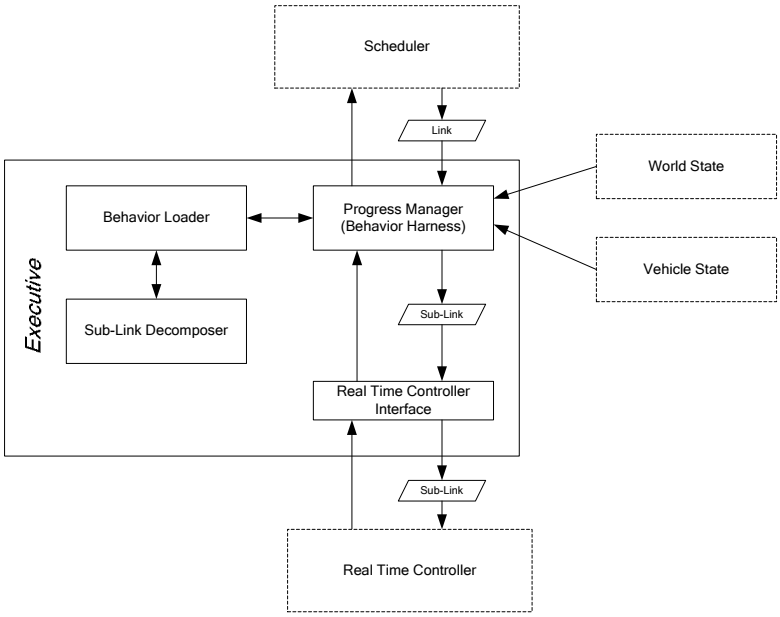

Figure 4. Major components of the Executive

Message	Direction	Description
New Sub-Link	Executive → Real Time Controller	Sends a new sub-link object (as described above).
Sub-Link Complete	Executive ← Real Time Controller	Issued by the Real Time Controller when a path has been traversed and the Real Time Controller is ready for the next path.
Sub-Link Impassable (Current Lane)	Executive ← Real Time Controller	Issued by the Real Time Controller when the current lane is impassible (but not necessarily an adjacent lane.)
Emergency Stop	Executive ← Real Time Controller	Sent by the Real Time Controller when an immediate obstacle was detected and the Vehicle has been stopped.

2.3 Real Time Controller

The *Real Time Controller* (RTC) is the lowest-level processor in the system. It links directly to the vehicle's control system and is the ultimate arbiter of control. Hence, the RTC's primary responsibility is to ensure the safety of the vehicle.

The *RTC* module receives sub-links from the *Executive*. Sub-links are defined as target vectors and target orientations for the vehicle. Additionally, the *RTC* is responsible for notifying the *Executive* when a sub-link has been traversed, or when it detects an obstacle that cannot be avoided.

The *RTC* must continually monitor the perception inputs. If an immediate obstacle is detected, then the *RTC* must stop the vehicle and inform the *Executive*. Furthermore, it must ensure that the vehicle is center of the proper lane.

The *RTC* may receive new sub-links from the *Executive* at any time. When a new sub-link is received, then the *RTC* must plan a smooth path to arrive at the target with the correct orientation. The path must be planned such that the vehicle stays in the proper lane, and other minor obstacles are avoided.

The *RTC* does not need to infer any special information about the sub-link. All inferences have been pre-computed by the *Executive* and *Scheduler*. Thus, the only intelligence the *RTC* must exhibit is (1) how to navigate to the target, and (2) how to handle obstacles. The *RTC* devotes resources to ensuring a safe and successful traversal of the sub-link.

The major components of the *RTC* are given below.

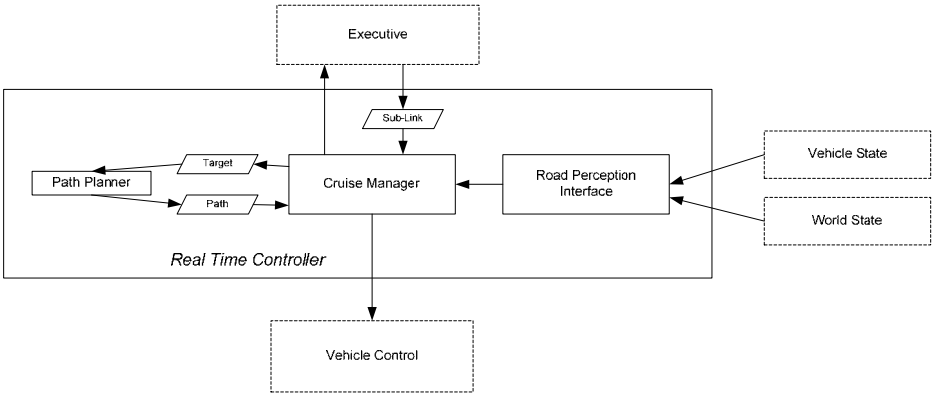

Figure 5. Major Components of the Real Time Controller

The *Cruise Manager* is responsible for keeping track of the progress of the given sub-link. It continually monitors the immediate perception field (as generated by the *Road Perception Interface*) and ensures the vehicle is in the proper lane and that it will not hit any obstacles. When an obstacle is detected, or when it perceives the vehicle is not on center, it will use the *Path Planner* to create an updated plan. If the planner cannot create a plan, then the *Cruise Manager* signals the *Executive* that the sub-link is not traversable.

Additionally, the *RTC* sends signals to the *Vehicle Controller*. The *Vehicle Controller* is an interface that can be interchanged with the simulator, or the actual vehicle. The major control outputs from the *RTC* are defined below:

Control Output	Description
Set Velocity	Sets the vehicles velocity or a specified time Δt. The Δt is used by the lower modules to compute the appropriate acceleration that will bring the vehicle to the desired velocity. (In other words, we can't specify an instant velocity, so we use the Δt to control how fast we want to accelerate).
Set Wheel Turn	Sets the orientation of the wheels. For example, 0˚ moves straight ahead, 15˚ moves to the right, -15˚ moves to the left. etc. Note: This parameter does not define the vehicle orientation, just the wheel orientation. Additionally, this function has a Δt parameter similar to above.

2.4 Simulator

The *Simulator* module is a supplementary system that is responsible for simulating the physical world in which the vehicle may operate. Although this is a supplementary system, it is vitally important to the development of the entire system.

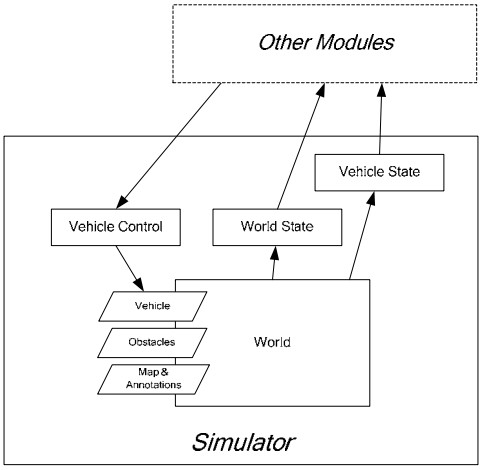

Figure 6. The internal structure of the Simulator

The simulator is responsible for receiving control commands and then creating corresponding perception outputs. The outputs are generalized in the table below.

Perception Output	Description
Coordinates	The world coordinates of the vehicle.
Orientation	The direction that the vehicle is facing
Wheel Orientation	The orientation of the wheels in the vehicle's reference frame.

Velocity	The current velocity of the vehicle
Odometer	The total distance traveled by the vehicle from the start of the mission.
Lane Position	When the vehicle is traversing a lane, this perception output will indicate the degree of alignment with the road (i.e. dead center, too far left, or too far right)
Immediate Obstacles	Boolean values that indicate if a specified region in the vehicles immediate range is populated with an obstacle.

In addition to the core perception outputs, advanced versions of the simulator may generate the following items:

Perception Output	Description
Traversable terrain grid	A data structure generated by a laser-range finder. It finds spikes in the immediate area that may be too steep for the vehicle. This data is used to populate a dense grid of traversable terrain of the surrounding area.
Stop sign line vector	The DARPA specifications require that the vehicle come to a complete stop before a stop sign's white line. This perception output would give a relative vector to that line.
Long range obstacles	Long range items will be perceived with computer vision in the physical implementation. Since computer vision often generates probabilistic results, this output will give a relative vector with a probability of certainty.
Dynamic objects	The DARPA specifications require that the vehicle pass and yield to other vehicles. Since this is a complex task, these outputs will also assign probabilistic certainties to moving items.

Obviously these latter perception outputs add complexity to the simulator.

2.5 Visualization

The *Visualization* module is responsible for generating human interpretable data of the inner workings of the entire system. The *Visualization* module is tightly coupled with all the other components. In this regard, the *Visualization* module can be viewed as library used by each of the main components to render interesting information.

Regardless of the component to be visualized, this module follows a basic pattern: (1) create a wrapper that references the module in question. (2) allow the module to operate without any awareness of the wrapper, (3) the wrapper will periodically query the state of the module and then render a meaningful visualization.

Notice that this design pattern eliminates dependencies from the other components to the visualization module. This is vitally important to the final implementation because the system must operate smoothly without the visualization present.

2.6 Perception

Perception is provided by analysis of the data provided by various sensors external to the vehicle: cameras, SICK laser range finder, sonars, GPS, compass, etc., as well as sensors embedded in the vehicle (e.g., velocity, odometry, on/off status, etc.).

2.6.1 Internal Perception

Access to sensor data internal to the vehicle is provided by the Pronto4 Retrofit Kit from Kairos Autonomi. This data is integrated into control loops (see Section 1.4).

2.6.2 External Perception

The major external sensory modes are vision and range. Vision provides continuous updates on the external world status, e.g., the existence of on-coming vehicles and the detection of the stop line in the lane. The SICK range finder is used to detect obstacles in the path of the vehicle and to the sides and rear. Finally, sonars provide a simple obstacle check in the side and rear directions of the vehicle.

3.0 Hardware Architecture

3.1 Pronto4 Retrofit Kit

The Pronto4 Retrofit kit is a complete implementation of actuators and hardware level interfaces for converting any vehicle into a drive-by-wire vehicle for purposes of teleoperation or autonomous operation. The Pronto4 is an off-the-shelf item that has successfully converted a variety of vehicles to robotic vehicles. The kit has 3 main components power, actuation, and computing.

Power: The Pronto4 power is pulled from the vehicle 12 volt bus and is diverted to actuators and also conditioned for use with computers, and sensors. The Pronto4 power system also contains a backup battery which is isolated from the vehicle power so it can supply the Pronto4 power for approximately 30 minutes in the event of vehicle power failure.

Actuation: The Pronto4 includes actuators for throttle, brake, steering and transmission. Servo motors are used to actuate the throttle, brake and transmission. The steering is actuated by a geared motor with an accompanying encoder. The brake actuators attaches to the brake petal via a pull cable such that the servo motor applies the brake the same way a human driver does. Our installation of the Pronto4 takes advantage of the native vehicle cruise control to tie in the throttle actuator. The transmission actuator ties into the

vehicle via a push-pull cable the same way the column shifter does. The steering actuator attaches to the steering wheel and braces to the floor to supply torque to the steering wheel. A digital signal processor (DSP) is used to issue servo commands to all actuators and provides an interface between the actuator hardware and PC based software.

Computing: The Pronto4 includes a Windows based platform that resides a variety of software modules. These modules issue actuator commands to the DSP, manage data being returned from the DSP and pull data from sensors. This computer provides the entry point for higher level software to interface with the Pronto4 system.

3.2 Power System

The design for the power system was based around the idea that given worse case scenario loads, the system would adequately provide enough current for the 12 volt and 120 volt power rails. During the operation of our vehicle, the sole power generation is done through our 12 volt alternator. This alternator can provide 120 amps of continuous current which equates to just over 1400 watts of useful power. The vehicle uses 20 to 60 amps of continuous current to operate all the essential equipment built in. This gives our equipment about 60 to 100 amps to work with. When designing the system, care was taken to make sure the alternator is not overloaded, so that at steady state, the van could continuously run a course.

Two 2 kilowatt inverters are used in the system. The inverters allow 120 volt computer power supplies and other electronic equipment such as networking routers and switches to be operated. The inverters are modified sine inverters which allowed for sensitive electronic equipment to be operated. The 12 volt battery's are in place as a backup in case the alternator ceases to operate during the competition or out in the field testing the vehicle. The charger supplies power to the entire system, including the vehicles main engine battery, when the vehicle is parked and the engine isn't running.. In addition, to the power system, the pronto 4 system has its own 12 volt battery, isolator and other electronics for overload safety.

Below is a diagram (Figure 7) of the power system components. It details which components were used and in what way they are connected.

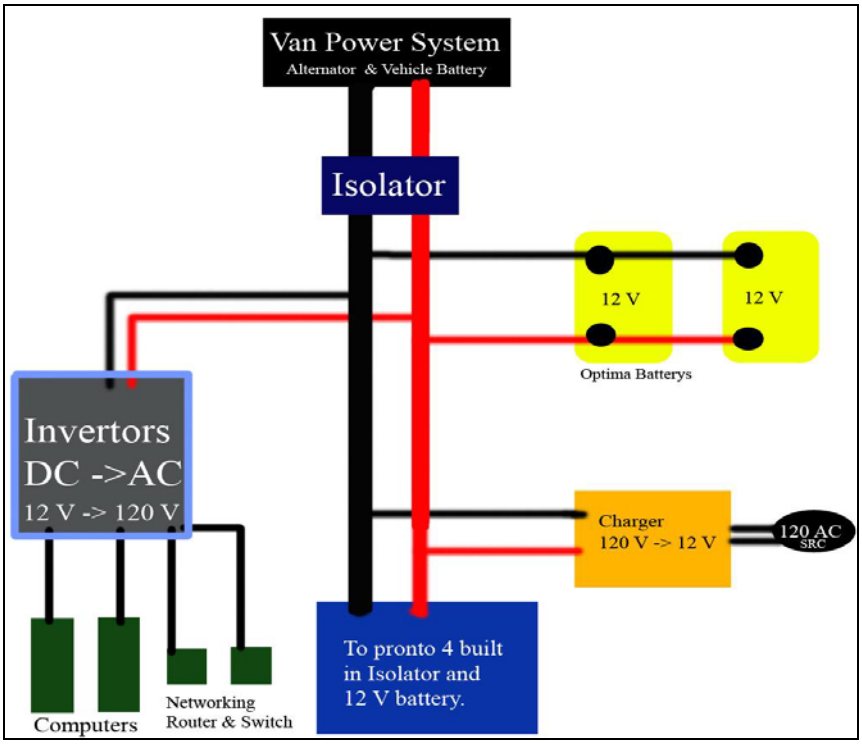

Figure 7 - Diagram of Power System.

3.3 Network

The network, or backbone, between all the computers in the vehicle, was designed to allow a high traffic rate given budget limits of the project. One gigabit cat 6 cabling and Netgear gigabit switch is utilized between all the computers in the vehicle. The switch was chosen over a passive hub since it allows each connection full bandwidth. Theoretically, this allows for one computer to send video data to another, while a separate computer can send data to yet another computer with little interference. In addition to the fast backbone between nodes, another connection was added using a Linksys wireless router. The router allows for an interface to be made with the computers from remote locations throughout the lab building making it simple to remotely configure the computers. In addition, data may be remotely monitored during testing. The wireless router is a developmental device only and can be easily removed for competition events such as the site visit, NQE and UFE.

3.4 Computers

Dual core Athlon 64 bit 3800+ CPU's with 2 gigabytes of system memory and loaded with Windows XP are used in this vehicle. Currently two computers were custom built and run the intelligence software and vision software. With the current level of CPU usage, two computers adequately run the code and capture data. Each computer utilizes about 100 watts. With the design of the network, power and other hardware in the vehicle, two more computers may be added with relative ease in the future as demand

dictates. Figure 8 shows the computers, Pronto 4 system, network, invertors and charger mounted in the vehicle.

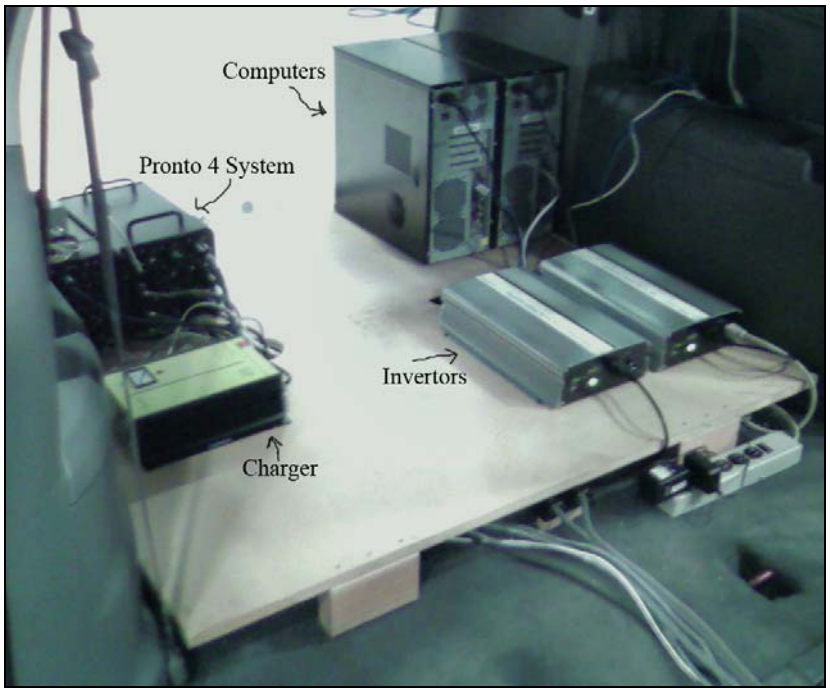

Figure 8 - Computers, Network, Inverters and Charger.

3.5 E-Stop

The design of the E-Stop was based on the idea of simplicity and feasibility. During the initial design stages of the E-Stop system, several solutions presented themselves. The solutions varied from $1000 to $4000 in price. The final design kept cost at a minimum by utilizing digital RC components. A DX6 2.4 GHz spread spectrum digital receiver and transmitter are the core components of the wireless E-Stop system. It was significantly cheaper then the alternatives and provided adequate distance between transmitter and receiver for our purposes (within .25 mile). The design of the system includes a built-in fail safe that if the transmitter power dies or the receiver went out of range, the receiver would go to a preprogrammed state. In our case this pre-programmed state is set to disable the vehicle.

The receiver interfaces to a set of two relay that actuate the ‘disable’ and ‘pause’ functionality built into the Pronto4 system. The activating the ‘pause’ throws a flag in software and the specific behavior is governed in software. On the other hand, the ‘disable’ functionality is hardware based and its behavior is dictated at the hardware level. To the best of our knowledge this configuration meets the requirements of the E-Stop and is capable of interfacing with the DARPA supplied wireless E-Stop for NQE. Figure 9 shows the E-Stop system components and connections.

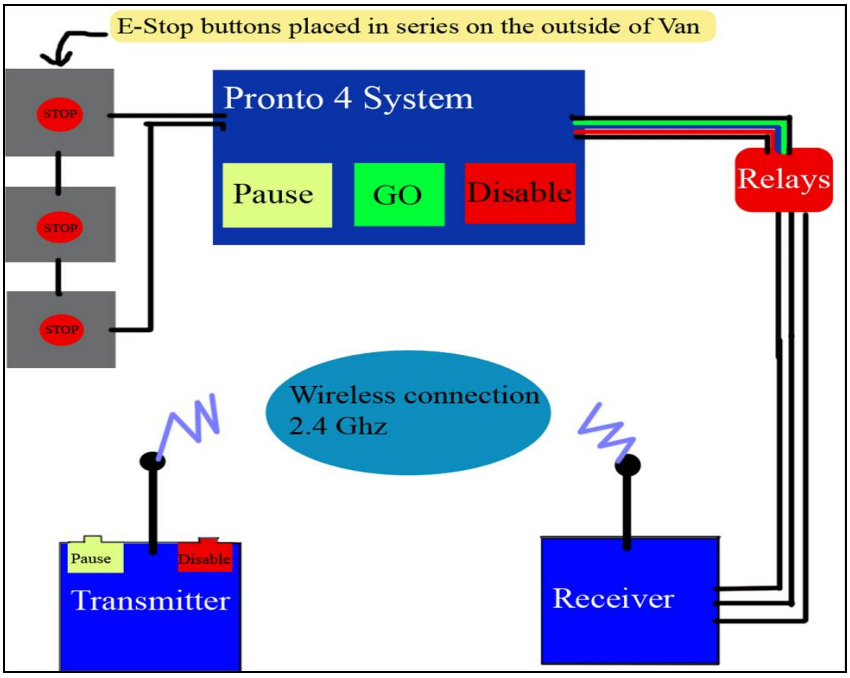

Figure 9. E-STOP Design and Connections.

4.0 Control

The purpose of this control system is to follow a designated path at a given speed on all types of road surfaces, e.g., flat roads, up-hill and down-hill. We report here the design of the speed controller and tracking controller as implemented on the autonomous vehicle.

4.1 Speed Controller

The challenge in designing a speed controller for an autonomous vehicle is the controller must consider both throttle and brake and decide what the appropriate action is for each. For example if the vehicle is currently traveling at 30mph and it is desired to reduce the speed to 28mph, is the appropriate action to apply the brake or let off the throttle. Also, when the vehicle is hovering around the desired speed with little to no throttle applied, it is important to avoid controller chatter where the controller rapidly switches between throttle and brake.

In determining what type of controller to use for speed control, two types were considered PI and PID. Simulations were used for an initial comparison of the two techniques, but in the end simple implementation and vehicle testing concluded, albeit somewhat qualitatively, the PID had better performance.

The implementation of the speed controller has two note worthy attributes beyond a standard PID loop. The both address said issues with switching between throttle and brake. First is the mode selector, which makes the decision between actuating the throttle or brake. The standard PID would dictate that if the error between the actual speed and

desired speed is positive apply the brake and if negative apply the throttle. Our mode selector applies a window around the desired speed. The effect is the controller does not switch modes when the actual speed falls within the window, instead it continues operating in its current mode. This helps prevent the controller from applying the brakes if it overshoots the desired velocity by 1 or 2 mph. The second addition was a rate limiter on the mode selector. So not only does the actual vehicle speed have to fall outside the window of the desired speed, it has to remain outside the window for some specified period of time (typically 0.5 to 1 seconds). Again this modification prevents the controller chattering between brake and throttle modes.

4.2 Tracking Controller

Our approach to vehicle control is to specify a desired path presented in the form of bread crumbs in front of the vehicle. A trajectory tracking controller is then used to follow these bread crumbs as closely as possible. The trajectory tracking controller is an implementation of a control algorithm presented by Kanayama *et al*. [19]. This controller defines a vehicle posture as

$$\mathbf{p} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$$

with respect to a global reference frame. An error is calculated based on a reference posture, $\mathbf{p_r}$, and the current posture, $\mathbf{p_c}$. An error posture is illustrated in Figure 12. The error posture, $\mathbf{p_e}$, is calculated in the vehicle reference frame by

$$\mathbf{p_e} = \begin{pmatrix} x_e \\ y_e \\ \theta_e \end{pmatrix} = \begin{pmatrix} \cos\theta_c & \sin\theta_c & 0 \\ -\sin\theta_c & \cos\theta_c & 0 \\ 0 & 0 & 1 \end{pmatrix} (\mathbf{p_r} - \mathbf{p_c}) .$$

The control law is then defined as

$$\mathbf{q} = \begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} v(\mathbf{p_e}, \mathbf{q_r}) \\ \omega(\mathbf{p_e}, \mathbf{q_r}) \end{pmatrix} = \begin{pmatrix} v_r \cos\theta_e + K_x x_e \\ \omega_r + v_r (K_y y_e + K_\theta \sin\theta_e) \end{pmatrix}$$

where v_r and ω_r are feed forward terms being the reference velocity and angular velocity, respectively, and must be specified as part of the bread crumbs by the path planner. Here $\mathbf{q}$ is the output vector of the controller and represents the required vehicle velocity and angular velocity to maintain the reference path. This velocity is then sent down the line to the speed controller that determines, via a PID loop, throttle and/or brake commands to achieve the velocity. The angular velocity is directly converted to a steering angle using the vehicle kinematic model.

Some deviations from the method presented by Kanayama et al. were necessary in our practical implementation of the controller. Most notably, the time dependence of the desired path was eliminated converting the trajectory tracker into a path follower and a steering saturation was imposed as a function of velocity. Both these changes were made to help improve stability of the control scheme.

Our initial implementation performed well at speeds up to 5 miles per hour. The inclusion of the steering saturation improved performance and speeds up to 10 mph were achieved. Additional improvements will focus on adding constraints to the path planner that take into account the rate of the steering actuator. The addition of constraints to the path planning will guarantee that the vehicle is capable of following the desired path.

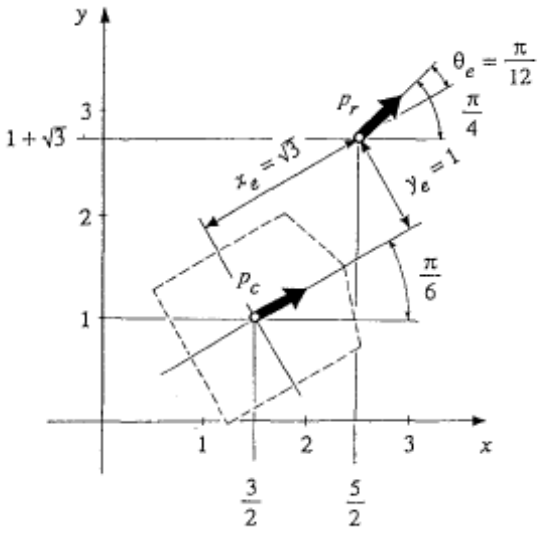

Fig. 12. Illustration of error posture where $\mathbf{p}_e = (\sqrt{3}, 1, \pi/12)$

5.0 Localization

It is essential for an autonomous vehicle to maintain some knowledge of its state in the world. This state consists of its location, heading, speed, etc. Fortunately, there are a variety of sensors that can be employed to help determined the vehicle state, i.e. GPS, IMU, odometers, compass, etc. In addition the Extended Kalman Filter (EKF) is one data fusion tool that can be used to incorporate all sensors into one vehicle state estimate.

Our approach utilizes at minimum a GPS, odometers, and a compass in conjunction with an EKF to provide a vehicle state estimate. The advantage of this approach is not in greater accuracy but improved robustness. The GPS alone can provide sufficient accuracy but temporary GPS outages must be tolerated. The EKF can continue to provide an estimated state from other sensors and its internal model, albeit an estimate with greater uncertainty, until GPS service is re-acquired.

GPS outages do not pose a great problem in the early phases of the project, and therefore extensive efforts have not been allocated to the EKF at this time. Thus currently, our localization is entirely dependent on our GPS sensor. However, it is expected that the Urban Challenge Final Event and the National Qualifying Event will test our vehicles ability to deal with temporary GPS outages and therefore, the full EFK approach will be implemented by the NQE.

6.0 Results and Performance

This section describes the major features of the system as well as the system requirements.

6.1 System Requirements

Our system comes equipped with a simulator that allows the cognition system to run without a physical vehicle present. Both the simulator and the cognition system are based on Sun's Java™ technology, and are therefore platform independent.

The simulator and cognition system can execute in a distributed manner. The different components of the system may require fairly expensive computations, and therefore the preferred hardware environment requires a dedicated machine to run each component. Each of the machines should have the following configurations:

- Recommended Hardware Requirements
  - 2.0 GHZ processor
  - 1 GB RAM
  - 3D Accelerated Video Card
  - Network Connection
- Software Requirements
  - Java 1.5
  - Java OpenGL (JOGL)

The final implementation of the system (i.e., the physical vehicle's implementation) requires minimally this hardware. However, the final implementation will not utilize any of the visualization components, so it is unnecessary for those machines to have graphics capabilities. Figure 13 shows various aspects of the instrumented vehicle.

Figure 13. Implementation: drive-by-wire (left); SICK & controls (ctr); actuation (right)

6.2 Behaviors

Parse and Interpret Mission & Map files. The Mission and Map files are the pre-specified inputs into the entire system. The Mission represents the checkpoints that the

vehicle must traverse, while the Map is a vector map specifying certain characteristics of the load. Our system loads these files and provides a graphical representation of them.

Rudimentary Simulator. The Cognition System is used in the physical vehicle. However, it is impractical to develop such a complex system without a way to test and tweak it offline. Thus, we have created a simulator to guide development. The Rudimentary Simulator encodes the basic physics of the vehicle. It exposes controls to other modules and allows them to interact with simulator. The Rudimentary Simulator also provides very basic perception capabilities. It includes a very basic graphical representation of the vehicle and map.

Navigate the Vehicle in a Straight Line. The system navigates the vehicle in a straight line towards a target checkpoint. Although this may seem trivial to human drivers, it requires that all components of the system interact effectively.

Stop the Vehicle for Perceived Obstacles. The system uses its perception capabilities to recognize obstacles. When an obstacle is perceived, the vehicle will come to a complete stop and end the mission. This behavior, although trivial, ensures that the basic perception system is functioning

Navigate through a Sequence of Straight Lines. This feature expands upon the core feature. The vehicle navigates through a sequence of straight lines checkpoints. This may seem trivial, but it ensures that the system is properly decomposing the map into manageable chunks. It also demonstrates the vehicle can distinguish when it has passed a checkpoint. This feature implicitly demonstrates that the entire system is working properly.

Pass a Stationary Object on the Left. Given a stationary obstacle in the current lane (such as a parked car), the system must plan a path around the obstacle by navigating through the left lane. This feature is somewhat complex because it requires use of the vehicle's limited sensor capabilities. This feature demonstrates that the system is truly

Right Hand Turns: When a vehicle comes to an intersection, it must be able to make a right hand turn. This is actually a somewhat complex operation because the perception range of the vehicle will not always be able to see the right-turn lane.

Left Hand Turns: Even more complex will a left hand turn. This is complex because it requires the vehicle to navigate through a space in which there are no defined lines or markers (e.g., the middle of the intersection).

Zone/Parking Lot Behavior: Part of the DARPA specifications require the vehicle to navigate through an area without any defining traffic lines (such as a parking lot). In this scenario, the vehicle must avoid any obstacles en route to the target checkpoint. Furthermore, the DARPA specifications require the vehicle to navigate to a specific parking spot. This is challenging, especially if there are parked cars on either side.

U-Turns/N-Point Turns: When an impassible obstacle is perceived, the system must be able to navigate to the other lane and choose a different path. This means the vehicle must execute a U-Turn. This implies that the vehicle turns 180˚ in the limited space of the road segment. If the road is narrow, then the vehicle must make a sequence of turns (such as a three point turn) until it has turned completely around.

Dynamic Obstacles: Dynamic obstacles (such a moving cars) introduce a fairly complex set of behaviors. The system must deduce the speed and trajectory of these obstacles and infer whether a specific action is safe or not. For example, making a left hand turn with oncoming traffic is difficult enough as a human, it will be much more complex for a computer.

6.3 Some Example Results

The vehicle has performed well in preliminary trials (see video submitted to DARPA [18]):

- following lanes in the RNDF defined road network,
- detecting stationary vehicles in the lane, and
- leaving the lane and passing stalled vehicles.

A sample image from that run is shown in Figure 14.

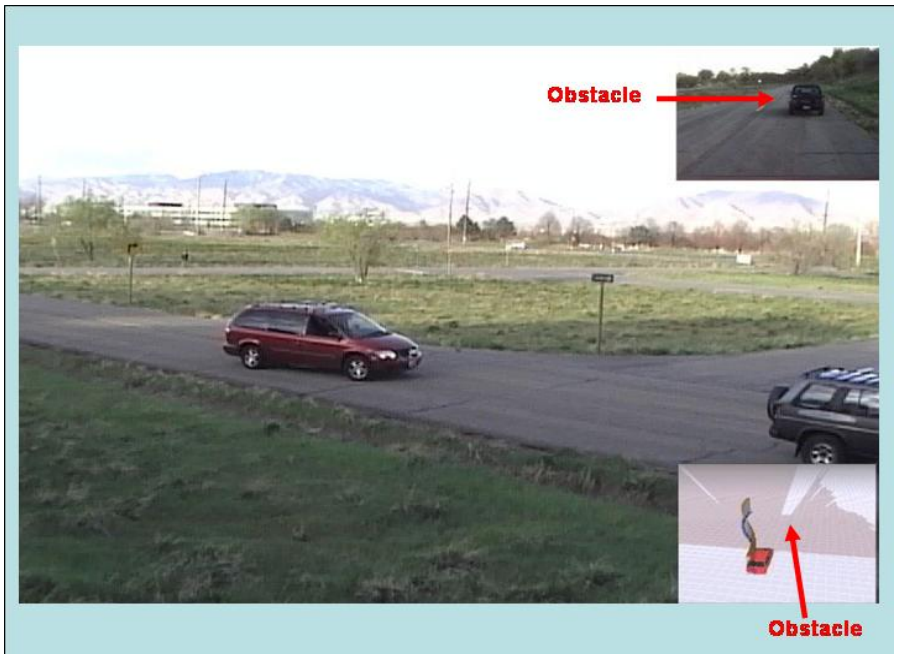

Figure 14. Sample scene from video demonstration (stalled vehicle detection).

As an example of obstacle and vehicle detection, Figure 15 shows vehicles located in a SICK image (in a parking lot). This data is representative as well of that which will be available at intersections and used for precedence determination.

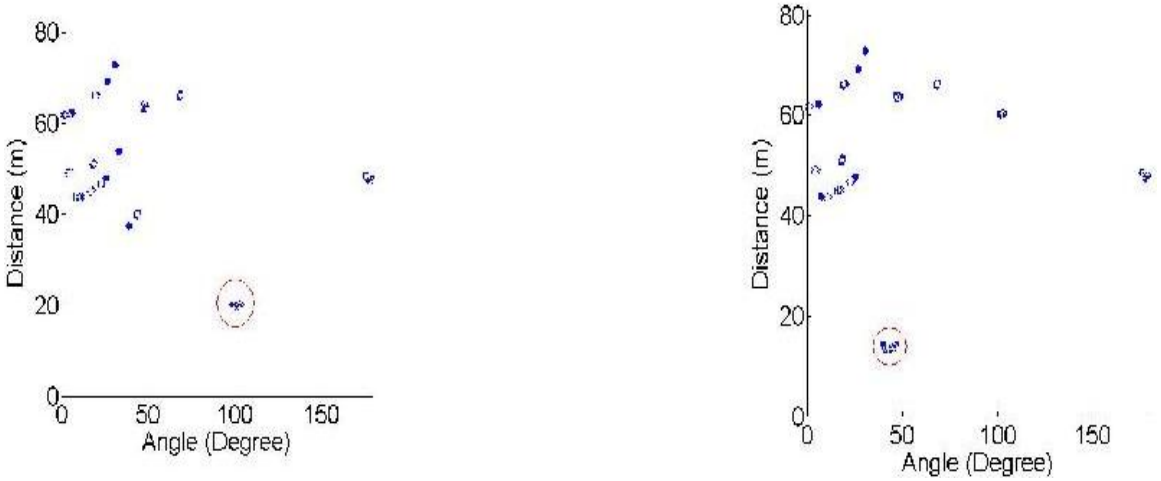

Figure 15. SICK data in Zone area (parking lot) with a vehicle passing from right to left.

Acknowledgments

We would like to thank CAVATI, Inc. for the donation of the vehicle, a Pronto4 system, and cash support, the University of Utah for seed grant support funds, and Troy Takach for his unbounded enthusiasm for the development of autonomous vehicles.

7.0 References

[1] Arkin, R.C., Behavior-Based Robotics, MIT Press, Cambridge, MA, 2000.

[2] Bekey, G.A., Autonomous Robots, MIT Press, Cambridge, MA, 2005.

[3] Brooks, R., ``A Robust Layered Control System for a Mobile Robot," IEEE-T Robotics and Automation, Vol. RA-2, No. 1, 1986.

[4] Durrant-Whyte, H., S. Majumder, S. Thrun, M. de Battista, and S. Scheding, ``A Bayesian Algorithm for Simultaneous Localization and Map Building," Proc. of the 10th International Symposium of Robotics Research, Lorne, Australia, 2001.

[5] Durrant-Whyte, H.F., ``Autonomous Guided Vehicle for Cargo Handling Applications," International Jnl of Robotics Research, Vol. 15, No. 5, pp. 407-440, October, 1996.

[6] Forsyth, D.A. and J. Ponce, Computer Vision: A Modern Approach, Prentice Hall, Upper Saddle River, NJ, 2003.

[7] Hebert, M., C. Thorpe, and A. Stentz, Intelligent Unmanned Ground Vehicles, Kluwer Academic Publishers, Boston, 1997.

[8] Henderson, T.C. and C. Xu, ``Robot Navigation Techniques for Engineering Drawing Analysis," Symposium on Document Image Understanding Technology, College Park, MD, Nov., pp. 157-166, 2005.

[9] Henderson, T.C. ``Explicit and Persistent Knowledge in Engineering Drawing Analysis," UUCS-03-018, School of Computing, University of Utah, October, 2003.

[10] Henderson, T. C. and Lavanya Swaminathan, ``NDAS: The Nondeterministic Agent System for Engineering Drawing Analysis," International Conference on Integration of Knowledge Intensive Multi-agent Systems, Boston, MA, pp. 512-516, Oct 2003.

[11] Nelson, A.L., E. Grant and T. Henderson, ``Evolution of Neural Controllers for Competitive Game Playing with Teams of Mobile Robots," Robotics and Autonomous Systems, Vol. 46, No. 3, pp. 135-150, March 2004.

[12] Nwagboso, C. (ed.), Advanced Vehicle and Infrastructure Systems, John Wiley and Sons, Chichester, 1997.

[13] Russell, S. and P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, Englewood Cliffs, NJ, 2002.

[14] Siegwart, R. and I.R. Nourbakhsh, Autonomous Mobile Robots, MIT Press, Cambridge, MA, 2004.

[15] Thorpe, C.E., Vision and Navigation: the Carnegie Mellon Navlab, Kluwer Academic Publishing, Boston, 1990.

[16] Thrun, S., W. Burgard and D. Fox, Probabilistic Robotics, MIT Press, Cambridge, MA, 2005.

[17] www.darpa.mil/grandchallenge/index.asp

[18] www.cs.utah.edu/urban

[19] Kanayama, Y., Y. Kimura, F. Miyazaki and T. Noguchi, "A Stable Tracking Control Method for an Autonomous Robot," *Proceedings IEEE International Conference on Robotics and Automation*, Vol. 1, pp.384-389, 1990.