UMASS University of Massachusetts Lowell

University of Massachusetts Lowell

Autonomous Vehicle Design Report

for submission to the

Intelligent Ground Vehicle Competition (IGVC)

Oakland University

May 2009

Professor Fred Martin
Department of Computer Science
University of Massachusetts Lowell
1 University Ave
Lowell, Massachusetts
fredm@cs.uml.edu

Compiled by Mark Sherman
msherman@cs.uml.edu

Table of Contents

Robot Overview

The MCP III.5 is a four-wheel drive autonomous vehicle equipped with camera, laser range scanner, GPS, and on-board processing. This vehicle represents four years of design by the Engaging Computing Group, part of the Department of Computer Science, at the University of Massachusetts Lowell.

The robot drivetrain is based on two 24v wheelchair motors with one motor per side of the robot. Each motor drives both wheels on its respective side, provided four wheel drive and differential steering. This drivetrain is very powerful, easily capable of driving a load in excess of 300lbs up a 15% grade incline.

Safety shut-off is implemented in hardware, triggered by either the E-stop buttons on the robot or by remote control. When in E-stop mode, the motor controller immediately ceases delivering any and all power to the motors. The E-stop buttons are active-low failsafe circuits, requiring continuous current flow to maintain Run mode. If the circuit or power is interrupted the system will default to E-stop.

This vehicle has two independent power systems, one at 12 volts and another at 24 volts. The 24v system is used purely for motor power. The 12v system powers all of the control systems, including electronics, sensors, and displays. The independence of these systems protects the control electronics from under-voltage when the motors are under heavy load.

Four sensors comprise vehicle input: laser range scanner, camera, GPS, and compass. The laser range scanner sweeps 180° in front of the vehicle and reports the distance to the closest object for every half-degree. This data is used for obstacle identification and avoidance. The camera is mounted above the robot body on a mast, providing a higher vantage point for vision processing. Computer vision systems are used to interpret the camera data as field lines and obstacles. The GPS system uses two global positioning receivers to determine the absolute location of the vehicle. The digital compass generates immediate heading data.

An on-board computer provides the processing resources to generate motor control signals based on sensor data. The computer is capable of sensor interpretation, vision processing, and area mapping in real-time.

Control software was written in collaboration by a team of graduate and undergraduate students. Significant collaboration was necessarily to develop sophisticated algorithms and systems to smoothly and reliably operate the vehicle under varied conditions.

For clarity, this report will specify the "Navigation Challenge" as the "GPS Challenge."

Robot Hardware

This version of the MCP is powered by a completely new processing system, based around a new on-board computer. These systems have been upgraded to provide more performance and reliability. In upgrading, the layout and wiring of all electronic systems has been redesigned to be more physically robust, allow easier maintenance in the field, and provide improved aesthetics.

On-Board Computer

During development this year the existing computer motherboard failed. A replacement was quickly selected. There were several constraints in the selection of a new motherboard and processor. The first constraint was the new computer should not drain power any faster than previous computers used. In the past, mobile and embedded processors were used, allowing several hours of battery life without recharging. Second, the successor computer should provide a higher level of performance. The third constraint was that the computer must not generate too much heat. This computer, as part of a vehicle, must perform reliably outdoors in potentially extreme heat conditions. The last of the major constraints was the presence of specific I/O ports to interface with the sensors. The camera used in the vision system needs a Firewire port. The laser range scanner needs a hardware serial port and is incompatible with USB/Serial converters.

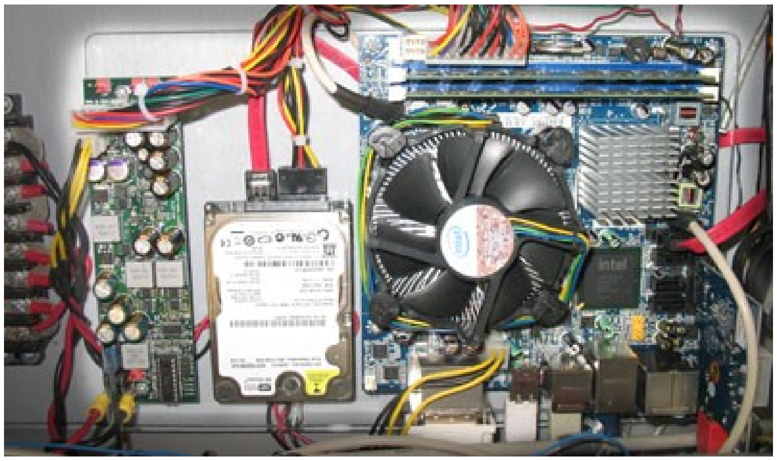

Figure 1: Computer layout.

The result of our research was the Intel DG45LF motherboard paired with a 3.0GHz dual core processor. This combination provides four hours of computation on a single battery charge, and produces acceptably low amounts of heat. During a four-hour run in the field, the ambient heat of the control systems remained the same, and well within limits. A Firewire port was not built into the motherboard, so several PCI-e Firewire adapter cards were tested. The selected card worked natively in Linux and with the existing vision software pipeline.

A new power supply was also introduced, providing an ample improvement. The previous embedded motherboard took power directly from the 12v battery. With all the other equipment running on the system, the raw 12v line tends to vary and can interrupt the operation of the motherboard. The new M2-ATX power supply was designed for automotive applications so it will survive large power fluctuations and safely power down the computer when the battery gets too low. The new supply is visible on the left side of Figure 1.

Control System Electronics

All electronics were rewired and hardware was remounted within the chassis. All hardware is mounted on a single board in the chassis. This board can be easily removed for field maintenance. The board also hides a channel beneath it where wires can be run. In this channel wires are both hidden from view and protected. The 12v and 24v systems were run on opposite sides of the vehicle for safety and ease of identification. There is little risk that a component could be accidentally powered incorrectly.

Safety System and Emergency Stop

The core of the safety system is the Emergency Stop controller. E-stop mode can be activated by pressing any of the three red E-stop buttons on the robot or by activating the remote control. The buttons are wired in series and are normally closed. Any interruption in the circuit, including a button press or a break in the wire, will cause E-stop mode to be activated. This is greatly improved over the previous iteration, where a loose wire would disable the safety systems. As another safety precaution, if power to the board is interrupted, E-stop mode will automatically engage. This situation can occur because the 12v control systems are on a different power source than the 24v motor controller. If the 12v system is interrupted, the 24v system could remain powered, but would automatically shut down as a result of this feature. The control circuit is based on reed switch relays. The output of the circuit is a relay connection. When power is removed from the relay coil the motor controller goes into E-stop mode. The diagram can be seen in Figure 2.

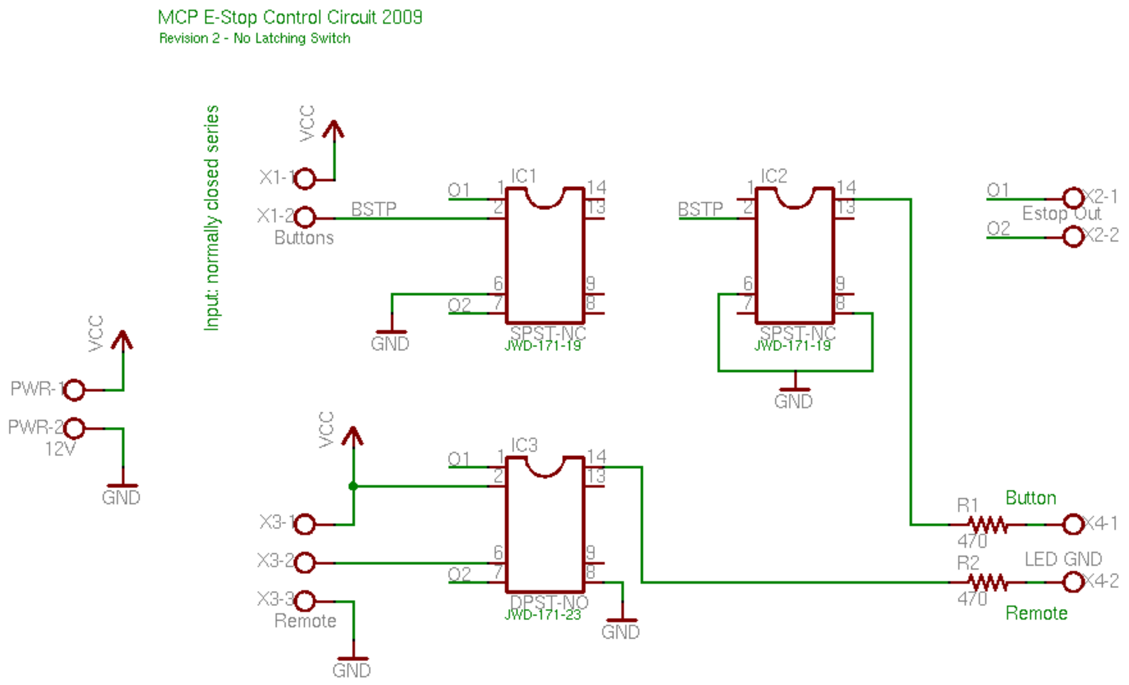

Figure 2: Emergency Stop control circuit.

Two E-stop buttons are on the top of the robot at its front, pointed upwards. The third button is on the mast facing backwards at shoulder level. If the robot unpredictably moves backwards, it is likely a person would hit the E-stop button accidentally, stopping the robot before any bodily harm is incurred.

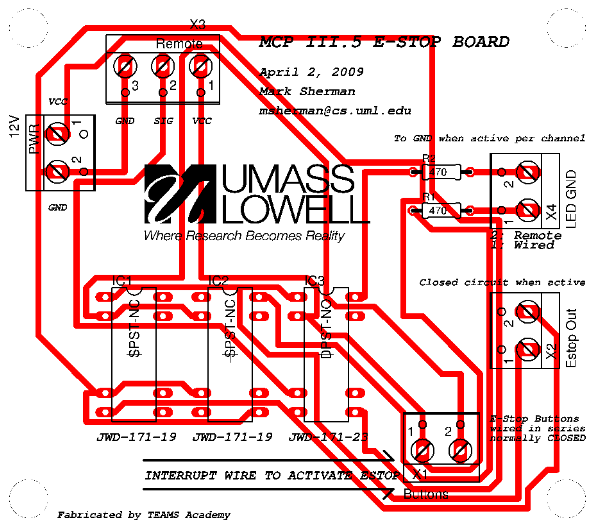

Figure 3: Emergency Stop control board layout. Red lines are underside traces.

The motor controller latches when it receives the E-stop signal from the control board. This safety feature requires an operator to approach the robot to manually restore the robot to Run mode. E-Stop mode cannot be accidentally or remotely canceled.

The remote control uses an automotive after-market remote lock system. When the "lock" button on the remote control is pressed, the controller pulls a signal pin to ground. As long as that signal pin remains grounded the robot cannot be taken out of E-stop mode. Pressing the "unlock" button on the remote control releases the lock, but E-stop must still be deactivated manually on the robot.

This control circuit was implemented as a printed circuit board and was custom fabricated using acid etching. The design goals of this board were to be robust and self-documenting. Low-level control boards like this one tend to stay with the project for many years, beyond the time when their designer may leave the team. Future developers will have an easy time working with this safety board because it is clearly laid out and labeled, as can be seen in Figure 3.

Drivetrain and Chassis

On the previous version of the MCP, each side of the drivetrain had a single sprocket and double link chain to drive two wheels each. This drivetrain had multiple problems. The sprocket on the motor drove one side of the double link chain, and the other side of the chain engaged with the wheel sprockets. This usage is not the intent of double link chain and results in instability as well as inefficient power transfer. The layout of the drivetrain was also problematic. The placement of the motor and wheels did not allow the chain to wrap far enough around each sprocket for proper contact, resulting in gears slipping and the chain falling off. Additionally, if either chain broke or fell off, the entire robot could no longer drive.

This system was redesigned in favor of dual sprockets on the motor, one for each wheel that it drives (Figure 4). Two chains are employed on each side of the vehicle, one to power the front wheel and one to power the back wheel. With this new system a chain break does not completely disable the robot, as only one out of four wheels would be disconnected. There is less of a chance of the chain skipping because the chains wrap at least 180° around each gear. In the process of rebuilding the drive system, we also raised the chassis, creating an additional 3 inches of ground clearance. Clearance was an issue in previous years when the frame would get caught on steep inclines. For additional torque, drive sprockets were selected at half of their previous size, creating an additional 2:1 reduction after the stock transmission.

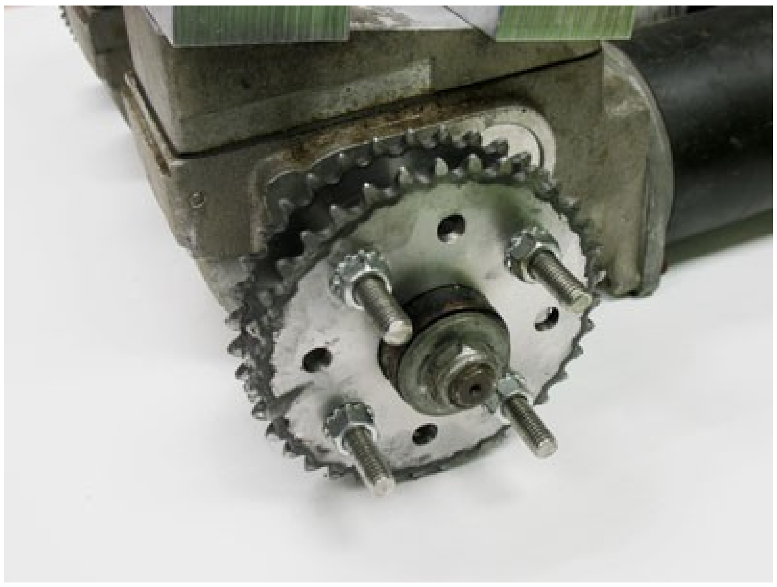
Figure 4: Double drive sprocket on the motor.

The mast was reworked into one piece with a quick disconnect wire harness allowing easy setup and breakdown of the robot for transportation. The mast now supports a compass, two GPS units, a camera, monitor, and emergency stop button. The mast also has a bright LED strip to let operators know when the robot is in autonomous mode.

At the front of the robot is the status and control panel. The top row contains all of the status LEDs and the bottom all the control switches and fuses. Left and right are broken down into 24v and 12v subsystems. Flanking either side are the Emergency motor stop buttons. The panel design can be seen in Figure 5. The only hardware metric that is not on this panel is the voltage readout of the 12v system, which is located on the rear panel near the user terminal.

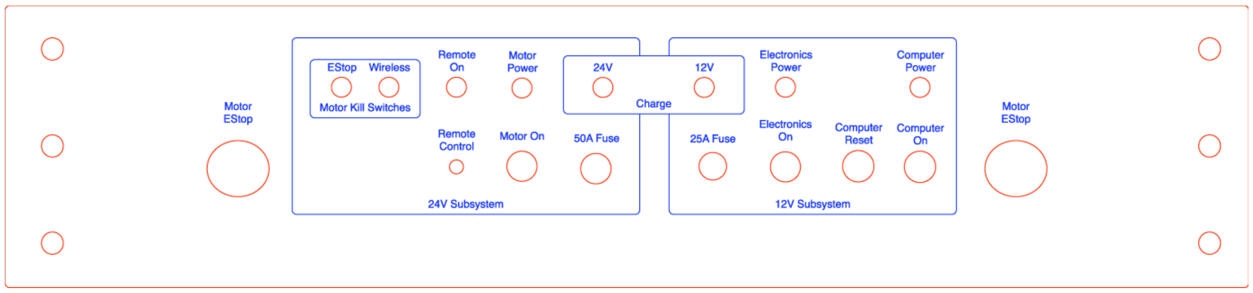

Figure 5: Status and control panel design.

Software Platform

Player/Stage Architecture

Simulation was used heavily during development. The vehicle software is based on the Player/Stage platform, where a robot control program (Player) connects to a simulation environment (Stage). Without any code modifications, the Player can be connected to a real robot instead of the Stage. This modularly allows for development in simulation with minimal changes to test the code on the actual vehicle. Player/Stage is an open source project, and is freely available.[1] Player/Stage includes a significant algorithm library, and use of these libraries is cited throughout this report.

Driver Interface

The robot driver interface is the critical bridge between the simulated and real operating environments. While Player's "position2d" interface provides a powerful and intuitive motion control scheme, no drivers existed that would allow Player to communicate with our robot's motor control system. As a result algorithms could be tested in simulated environments, but the same code could not be used in the physical robot without time-consuming modifications. To resolve this issue, a custom position2d driver was built to meet our specific needs. This driver utilizes Player's support for user-created plug-ins.

The Player control code can provide position2d commands in several formats: "car-like", motor positioning, absolute heading, and forward and angular velocities. The format most compatible with our configuration is a pair of values: forward velocity in meters per second, and angular velocity in radians per second. Our driver accepts this input and computes the rotational speeds of the robot's left and right wheels to match the desired course. These rotational speeds are then translated into power values that represent percentages of the robot's top speed and are transmitted to the motor controller.

The driver also enforces speed limitations. If for any reason the driver receives a position command that would require one of the motors to perform beyond a threshold level, the driver will scale down both forward and angular the velocities by the closest possible ratio that allows the robot to remain on course without exceeding maximum safe speed.

1 The Player Project: http://playerstage.sourceforge.net/

Another compatibility concern was position reporting. The position2d interface supports localization via wheel odometry; however, the power-train design did not allow room for encoders, so odometry support was not included in initial builds of the position2d driver. The intent was that localization would be accomplished through the GPS interface, obviating the need for odometry. The Stage simulation environment does not support simulation of a GPS interface, so programs working in the test environment would use the position2d interface for localization, while programs working on the robot would use the GPS interface. To resolve this conflict, the position2d driver was reconfigured to report GPS data (converted into UTM easting and northing values) in place of odometry data. Code from the GPS driver was streamlined and integrated into the position2d driver, along with support for multiple source of GPS data to reduce position error.

When two GPS units are present, the driver uses the mean of the two reported positions; if more GPS units are present, more elaborate algorithms are capable of detecting outliers and further refining sensor precision. Interface functions that set or reset odometry were modified to offsets applied to the reported GPS positions. The driver also transparently allows access to interfaces for the attached GPS units, removing the need for a separate GPS driver.

GPS Navigation Challenge

For this challenge multiple solutions were developed by different students. Two of these solutions are purely reactive, keeping no memory of their environment. The third solution, of which there are two variations, utilizes mapping and planning. The final contest code will be selected in the field. This section presents each of the solutions developed.

Reactive Solutions

This solution uses a simple GPS Navigation software module to provide a way point heading to the obstacle avoidance routine. It does not use a map, planner, or an explicit finite state machine. The control program navigates to the waypoints without any persistent information about the operating environment. With this solution it will admittedly be more difficult to achieve the same level of success as a control program using persistent world information with an intelligent planner. The obstacle avoidance of a reactive solution will need to be far more robust, implementing an “instinct” for the environment. Although a purely reactive system will likely be outperformed by a well designed planning system, a reactive solution can be used to replace the simple navigation component of an that intelligent planner. This combined system will be more robust, especially in unforeseen circumstances, outperforming other solutions.

Two reactive solutions were designed. The first utilizes the off-the-shelf Vector Field Histogram (VFH) algorithm. The second is custom in both design an implementation, using vector representations of path safety. Both of these were designed as fall back systems, with the off-the-shelf solution being most reliable but least efficient.

VFH Utilization

Desired waypoints are stored in a list. First the current location of the robot is recorded and added to the back of the list. This ensures the robot will return to its “home” when all other waypoints have been reached. Once a waypoint is visited it gets removed from the list. If there are more waypoints remaining the closest waypoint is selected and the robot travels to it. During the search for the closest waypoint the last one (home) is ignored.

The algorithm that moves the robot to the waypoints is simple. The obstacle avoidance and navigation to the waypoint are done by the vector field histogram (VFH) algorithm that comes as part of Player. VFH was first developed in 1991 for use with sonar range sensors.[2] When the VFH algorithm is supplied with coordinates, it starts to move towards that goal. VFH does not need any information about the objects present on the map as it does no planning. Obtaining the behavior best suited for open outdoor environments is simply a matter of adjusting the parameters passed to VFH when the Player server is started. VFH does not generate the most efficient paths, but it does work reliably.

One of the problems with using VFH is that it has no concept of course boundaries. These edges must be artificially introduced into the laser range scanner values. In Player this can be done by creating a custom driver for the laser device. The custom driver reads values from the default laser driver and a custom position driver. The position driver provides the orientation and the current coordinates of the robot, which are referenced against the known course boundaries. From this data the laser values are manipulated to see an imaginary wall where the boundary is, preventing VFH from going outside the bounds.

This approach is the lowest level of failsafe, intended to be used in the field only if the more sophisticated solutions fail.

Vectorized Approach

In the Vectorized Approach, all laser range scanner data is collapsed into a number of vectors that originate from the robot. These vectors either guide the robot towards open spaces or away from obstacles. An approximate representation is shown in Figure 6. Vectors are classified under two types: guiding vectors (which tell the robot where to go), and pushing vectors which describe areas containing open space. Readings in the laser scan within a certain distance threshold produce pushing vectors, which are larger for close ranged readings and readings from the front of the robot. Note that pushing vectors do not necessarily represent a single physical object. Walls and large convex objects may appear as multiple pushing vectors with small angles between them. It also presents the opportunity for certain large concave surfaces to appear as a single high-magnitude pushing vector, which would help prevent the robot from driving into traps. In addition to guiding and pushing vectors, space-type arcs are also generated, to quickly ascertain if a certain bearing in front of the robot is open space or an obstacle.

2 Borenstein, J.: Koren, Y. (1991). “The vector field histogram- fast obstacle avoidance for mobile robots. *Robotics and Automation, IEEE Transactions on* **7** (3): 278-288.

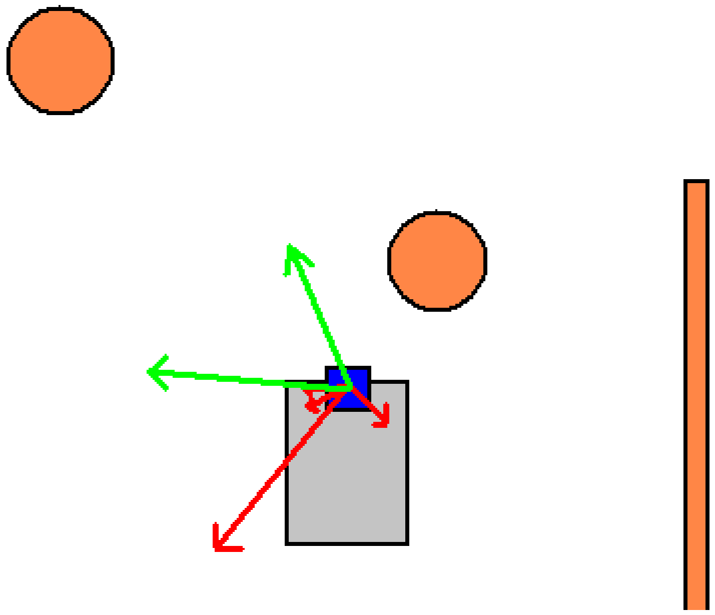

Figure 6: An approximate representation of the vector guides used by obstacle avoidance. The 'pushing' vectors are red, indicating how to move to avoid obstacles. The 'guide' vectors are green, indicating how to move towards free space.

The obstacle avoidance routine uses the vector guides in addition to the distance and bearing of the next waypoint. The obstacle avoidance first uses the space-type arcs to determine if the bearing to the waypoint is in an obstacle region or a open space region. If it finds an obstacle region, it checks to see if the distance readings on that bearing are closer than the distance to waypoint.

If the avoidance routine finds that the bearing is in an open space region, it will perform as follows. The closest guiding vector towards the waypoint will be selected. The select guide will be vector-summed with the bearing to the way point, which is given an equivalent magnitude. The most significant pushing vector is then selected and is vector-summed into the guide. Note that although smaller pushing vectors are not directly added into the final vector, their existence prevents the creation of guiding vectors, thus they have an impact without risking over-summed or null results.

If the avoidance routine finds that the bearing is in an obstacle-filled region, it will perform a wall-following behavior until it finds that the bearing is in a clear area, minding course boundaries. The wall-follower does not use guiding or pushing vectors, as the use of trigonometry with the original scan data produces information that is much easier to interpret for the purposes of aligning to a wall.

Planning Solution

Most of the complexity of the GPS Navigation Challenge is in negotiating traps, switchbacks and other potentially confusing obstacles in the field. Traps most often take the form of a circular obstacle where the goal point is inside the circle and only one break in the barrier exists to enter and exit through. Reactive solutions often fall victim to these traps, caught repeating the same incorrect behavior over and over. If the robot manages to escape under reactive control it has no mechanism to prevent itself from entering that trap again. To overcome this problem, a planning solution is necessary. A planning solution creates maps of the environment as it is explored and designs efficient routes through that environment.

Data from the laser range scanner are used to detect obstacles and open spaces relative to the robot. Data from the GPS and compass are used to determine how those locations correspond to a larger world-view, such as the competition field. As data are accumulated, a map is created that represents how certain we are that any given location contains an obstacle or is safe to travel through. Utilizing this map, the path-finding algorithm is able to build efficient routes to any point within the environment.

In accordance with regulations, all map data are destroyed at the conclusion of a run.

Two different mapping solutions were explored: a hex grid and a square grid. A hex grid was chosen because it allowed for more flexibility in path creation and smoother cornering. A 60 degree turn can be traversed more quickly and easily than a 90 degree turn. The square grid implementation was developed as an alternative. Having an alternative raises the likelihood one system will be successful. The hex grid can be seen in Figure 7.

Surroundings are mapped using a two-pass scan of laser data. In the first pass, all laser contacts are flagged as obstacles and the map cell that contains the coordinates of the laser contact has its Obstacle Certainty counter incremented. Cells surrounding the contact are also flagged for caution to allow for localization error. If the grid resolution is high enough that the cells are smaller than the dimensions of the robot, this caution zone will also prevent the robot from attempting to pass through cells that would bring it into contact with an adjacent obstacle. Contacts from beyond a certain range are discarded as being unreliable (e.g. a steep incline may appear to be an obstacle). In the second pass, an algorithm scans for open space: cells are checked starting directly in front of the robot. If there are no laser contacts within or close to a cell, that cell's Safe Certainty counter is incremented and the algorithm moves on to check neighboring cells. Cells that are beyond reliable range or at the edge of the laser's field of vision are ignored.

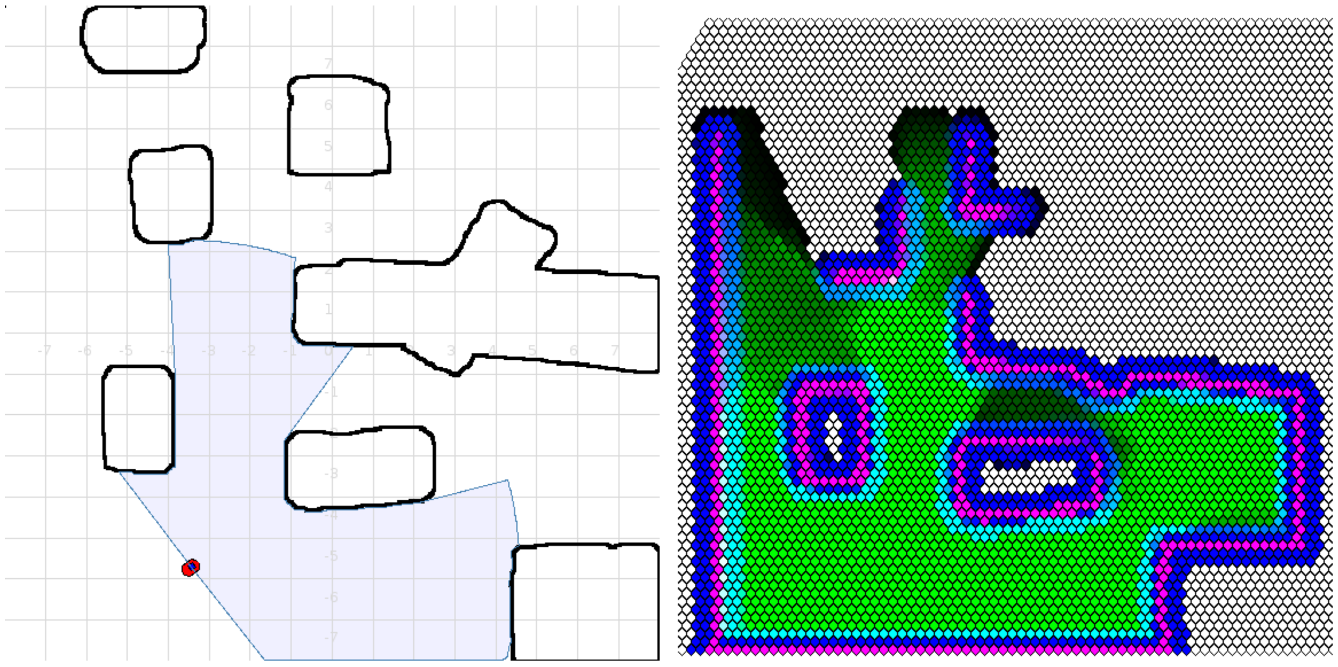

Figure 7: Hex Cell map next to the simulated environment being mapped. Brighter green indicates a higher confidence of empty space. Pink is a known obstruction. Blue is a safety barrier. The robot is represented by the red vehicle on the left panel.

The mapping algorithm is also capable of correcting for transient obstacles, such as a person walking. When a cell contains contradictory information, it is flagged as low confidence. If further scans provide consistent evidence for one state over the other, the appropriate certainty counter will reach its maximum value; readings that would increment the maxed-out counter instead decrement the opposing counter. If scans consistently report contradictory readings (an unlikely but theoretically possible situation), both counters will remain at or near their maximum values. In such a case, the danger of a potential obstacle is considered too great and the cell will be avoided unless the robot can find no other path.

The A* search algorithm in used to determine the best path from the current location to a destination. A* is a best-first graph search that find the least-cost path based on a weighted heuristic function. The higher a cell's obstacle certainty, the higher its path weight; the higher its safe certainty, the lower its weight. There is also a buffer zone of caution around obstacle readings to prevent the robot from brushing against an obstacle while trying to traverse an adjacent cell. The values assigned to these weights are calibrated to encourage approaching unexplored areas if they could potentially reveal a shorter path. Path nodes that pass through cells that have little or contradictory data are marked as being low confidence, letting the robot know that it should recalculate the path after gathering more data.

Autonomous Challenge

To discover the lines on the field the vision system uses a growing-regions blob detection algorithm. To do this the images must first be transformed from the Red, Green, and Blue color space to the Hue, Saturation, and Lightness color space. The HSL color space is used because it is easy to detect white due to the Lightness factor. In the RGB color space, white, or shades of white, is anything where the values of Red, Green, and Blue are the same. This seems easy enough to use to detect white, but many objects do not meet that criteria exactly, resulting in signal noise. With the HSL color space, lightness values can be examined directly making it much easier and faster to scan an image. This method is prone to noise from glare off of other objects in the field. However, this can be easily reduced by looking at the shape and size of blobs.

Blob Detection

Blob detection is done by scanning every pixel of the image. With the Growing Region algorithm the scans occur in rows starting from the top of the image. Each pixel in the lightness layer of the image is examined and compared to a variable threshold value which can be set according to the conditions of the environment. If the pixel meets the threshold value, it is marked as the starting pixel for a line blob. The row continues to be scanned until a pixel does not meet the threshold value, this pixel marks the end of a line blob. Once a line blob is detected it is compared to the line blobs from the pixel row above its row. If any line blobs from the previous row are found to overlap the current line blob, the two line blobs are attached to one another and given the same blob ID. This process continues for each row of the image. The result of the algorithm is a collection of line blobs which are combined by their blob ID to make single blob objects. These blob objects contain the location of each pixel that outlines the blob, along with the minimum and maximum bounds of each blob. Line blobs are found, connected to each other, and formed into blob objects all in one pass through the image. The execution time of this algorithm grows as the square of the image size; at our 640 x 480 resolution, it executes in approximately 10 milliseconds. Once all the blobs have been detected, blobs with an irregular shape as determined by examining their bounds are discarded as noise from glare. All accepted blobs are drawn onto a black image to create a binary image for further processing (as seen is Figure 9).

Hough Transform

Once the blob detection is complete, the blobs are used to create a binary image. Each blob, provided it meets the filter requirements, such as shape and size, are outlined onto a black image. This image represents the blobs with the highest probability of being a line. The Hough transform is then used to find straight lines in an image making it ideal for finding the path; blobs with straight edges which meet a minimum length criteria are detected. Using Hough, we are also able to connect lines that are spaced apart, like dashed lines or lines with objects covering part of them. This prevents the robot from running off track when a white line painted on the course is not continuous. Once all the lines in the binary image are detected they are filtered one more time. This final filtering step is based on the compass heading. By tracking the rotation of the robot, the vision system can determine whether or not it should be able to see both lines, just the right, or just the left line. If both lines should be visible, the lines are examined to find one line of the left side of the robot which has a corresponding parallel line on the right side. If only one line should be seen, the closest line to the robot is taken as the path. The vision control program then looks at the distance of each line from the center of the image to determine how far it is from running over the edge of the path. This distance can be provided to lower level control programs to have the lines portrayed as obstacles, or a turn speed and radius can be provided to get directly back to the center of the path.

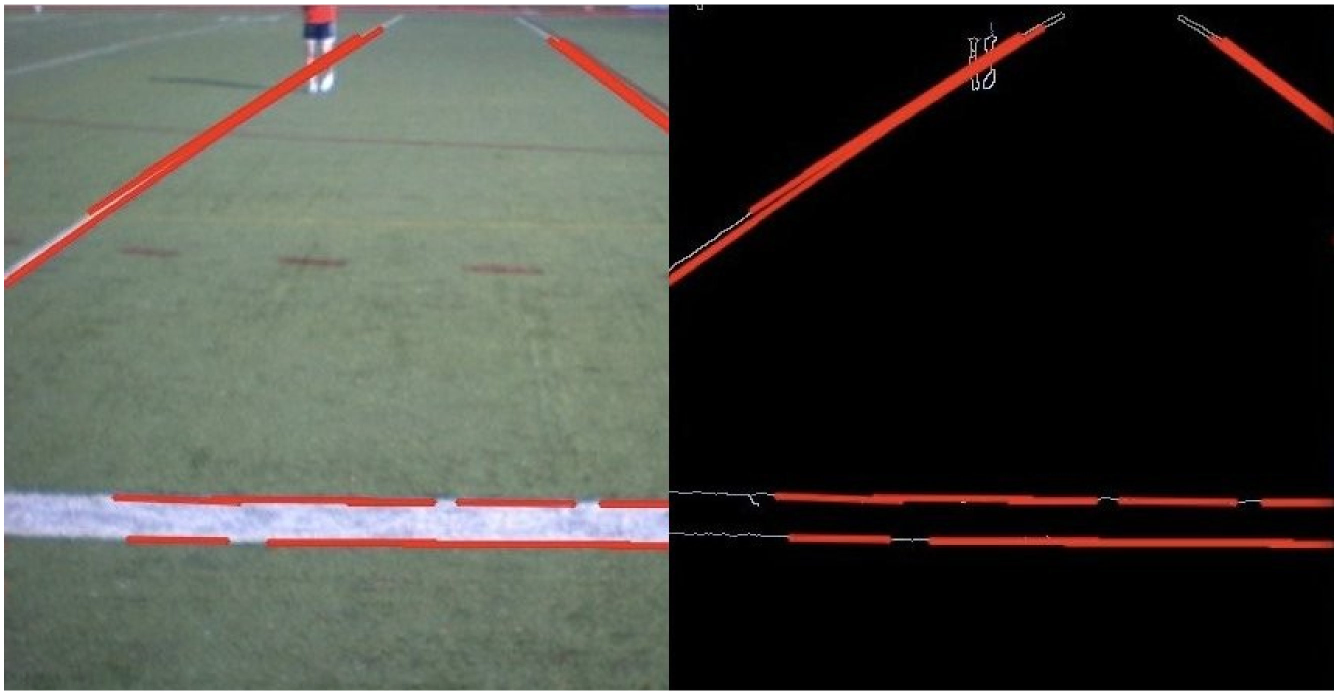
Figure 8: Hough-detected lines drawn on the source image (left) and the Canny edge-detection resultant image (right). Note the presence of the person breaking the line did not affect the result.

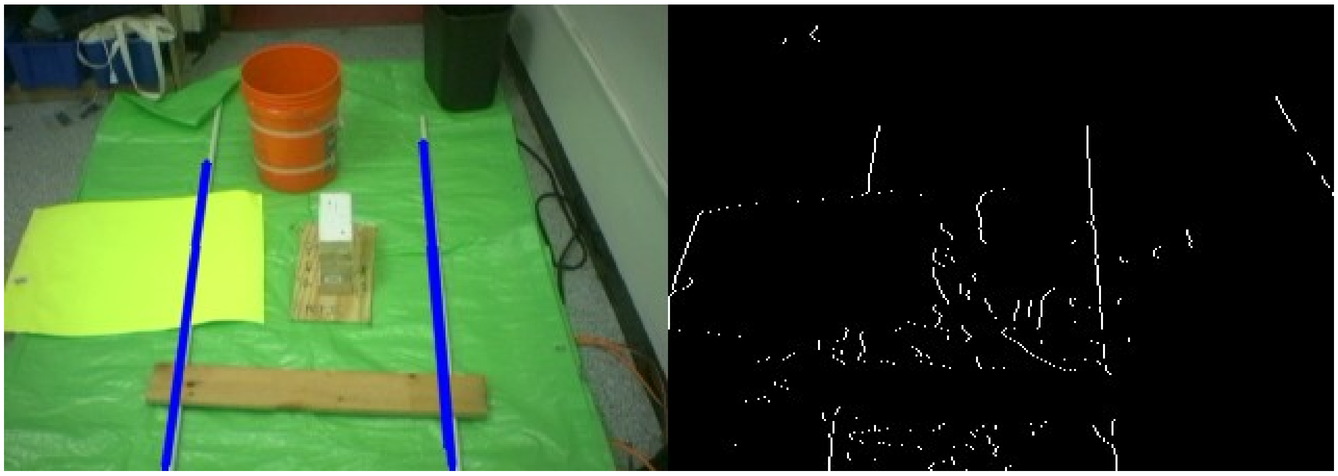
Figure 9: Left: image with Hough-detected lines drawn in blue. Right: binary image created by blob detection.

Team Organization

- Ryan Buckley, Computer Science, Blob detection.
- Michael Court, Computer Science graduate student, Path following algorithms.
- Jim Dalphond, Computer Science graduate student, Drivetrain, Computer, Electronics.
- Munjal Desai, Computer Science graduate student, Reactive navigation, Player/Stage/Gazebo.
- Katherine Dufault, Mechanical Engineering, Drivetrain, Chassis.
- John Fertitta, Computer Science, Vision systems and autonomous navigation.
- Seung Wook Kim, Computer Science graduate student, Line detection.
- Michael McGuinness, Computer Science, Mapping, Pathfinding, Obstacle avoidance.
- Gregory Pilla, Computer Science graduate student, Reactive navigation, Software integration.
- Megan Reichlen, Computer Science, Blob detection, Data fusion.
- Mark Sherman, Computer Science graduate student, Safety systems, Electronics, Integration
- James Shimer, CS graduate student, Process communication, Intelligent Laser algorithms.
- Chad Sweeney, Mechanical Engineering, Drivetrain machining.
- Nat Tuck, Computer Science graduate student, Pathfinding, Obstacle avoidance.
- Ryan Tucker, Computer Science graduate student, Hardware, Chassis plating
- Tor Valeur, Computer Science, Driver code, Mapping, Pathfinding, Obstacle avoidance

Team overseen by Professor Fred Martin, UMass Lowell Computer Science

Appendix: Equipment Specifications

Laser Range Scanner: SICK LMS200 - $5,800

- 180° field of view
- 0–80 meter operating range
- 75 Hz scanning frequency

Computer:

- Intel DG45LF Mini-ITX Motherboard - $147.26
- Intel Core 2 Duo 3.0Ghz dual-core processor - $167.66
- 2GB RAM - $25
- Western Digital 160 GB SATA hard drive - $47.99
- M2-ATX power supply unit - $79.50
- 2-port PCI-e Firewire card - $59.00
- Apple iSight Camera - $200
- LCD Monitor - $100

Motor Controller: Roboteq AX3500 - $395

- 40A sustained max current (1 hour)
- 250A surge current
- Automatic current and voltage limiting
- Digital and analog control inputs

Drivetrain:

- 4-wheel drive, differential steering.
- Invacare 1085952 motors with gearbox assembly - $1390
- 2:1 reduction outside of manufacturer gearbox
- Chain, sprockets, idlers – 173.80
- Chassis materials ~ $500

Custom E-Stop Circuit Board

- Electronic components - $40
- Remote Control system - $40
- Manufacture: donated, professional process ~ $65

Additional Electronics

- SEC12-15 Samplex 12V 15A Battery Charger - $149
- Batteries - $300

Base vehicle cost: $9600