University of Massachusetts Lowell
Design Report of the MCP Intelligent Ground Vehicle

Compiled by Mark Sherman
Advised by Fred Martin & Holly Yanco

May 14, 2010

Contents

1 Introduction

1.1 Team Members

Software

Nat Tuck, Tom Kiley, David Hunt, Robert Thew, Christopher Deveau

Electrical Engineering

Mark Sherman, Jeff Rousseau, Randy Kwende, Nikolaos Eleftheriadis, Shiv Sharma, Jason Tarantino, Garo Yessayan

Mechanical

Andrew Sutton

1.2 Robot Overview

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

The MCP started this development season in a working state, so the goal of development was to minimize hardware downtime and develop and all-new software base.

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, providing 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 primary 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. Secondary sensors report robot health, including battery voltage, temperature, and current draw.

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.

The software platform is based entirely on a modular simulation environment known as Player/Stage, allowing for software to be developed in simulation and nearly transparently ported to the actual robot and back.

2 Robot Hardware

The MCP is powered by a modern dual-core processing system, upgraded from the 2009 entry. The new system includes IEEE1394 and true UARTS on-board, which allows the robot to operate without daughter cards. With all necessary ports built-in the computer gains additional robustness and reliability. In upgrading, the layout and wiring of all electronic systems has been redesigned to be more physically robust and allow easier maintenance in the field.

2.1 Electronics

The heart of the electorincs system is a Pentium Core II Duo for processing and a Roboteq AX3500 motor controller. The major constraint in choosing the motherboard was the presence of specific I/O ports to interface with the sensors. The camera used in the vision system needs an IEEE1394 Firewire port. Of all the firewire chipsets in existence, only one is proved to work reliably in linux with full bandwidth, so the selected mothorboard must inclue that Texas Instruments chipset. The laser range scanner needs a hardware serial port and is incompatible with USB/Serial converters.

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.

2.1.1 Emergency Stop

The core of the safety system is the Emergency Stop (E-Stop) controller. E-stop mode can be activated by pressing any of the 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. 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 solid state relays as they provide additional reliability over the vibration-sensitive reed switch relays used in the past.

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.

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. This circuit was integrated into a new board and will be replaced on the MCP. This board is described in section 4.1.

2.1.2 Power Switcher Board

As described above, the MCP employs two independant power systems: one at 12v and another at 24v. It is a firm requirement that both systems be able to be charged without powering them down. The operator should be able to simply plug and unplug the charger at their convenience and have the system "just work." A problem was discovered with the previous implementation: the battery was charged in parallel with the load, creating additional current draw. This problem resulted in destroying the chemical composition of a new lead-acid battery within one year, reducing its runtime from three hours to fifteen minutes. This problem is described in detail below.

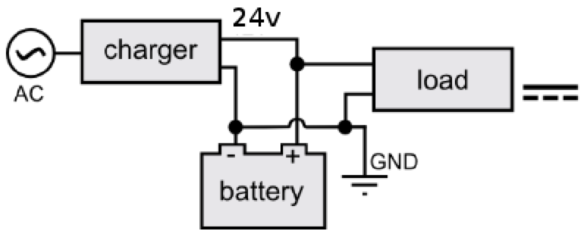


Figure 1: The original, problematic MCP power configuration.

Charging a lead-acid battery is done in three stages, which are automatically detected and executed by most smart chargers today. The first two stages provide ample current to allow the battery to regain the majority of its charge, and then the third stage provides a low current floating voltage as maintenance. The amount of current that the battery draws is what triggers the changes in phase. If a load is connected in parallel with the battery, as shown in Figure 2.1.2, and that load is drawing current, the charger never detects a reduction is current demand and continues to pump the early-phase high current levels, even if the battery is long done calling for it. This overcurrent is damaging to the battery.

The solution to this problem was to design a power switching circuit that would sit between the computer and the two sources of power. To do this a Linear Technologies LTC4414 IC was used to sense when the charger was plugged in. When the charger is sensed an external power MOSFET is triggered, cutting the batteries off from the load. With the battery cut off from the load, it is possible to charge the batteries with one bank of the charger, and run the computer with the second bank. The charging circuit designed to solve these problems is depicted in Figure 2.1.2.

With the addition of the power switcher, a "cold off" switch has been added that completely isolates the battery from the system. When the battery is connected, the intelligent power switcher always uses a faint amount of current to monitor the input lines. For long-term storage and transportation this is unadvisable, and the "cold off" provides a complete, full off-state of the robot. When the robot is on powered by the charger, the robot will continue to run off of charger power even if the batteries are in "cold off."

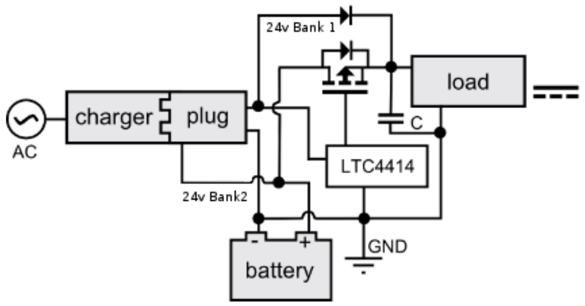


Figure 2: The improved MCP power configuration.

2.2 Mechanics

The MCP uses differential drive, where the wheels on either side of the robot can be controlled independantly. 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 design a chain break does not completely disable the robot, as only one out of four wheels would be disconnected. This design also provides power and avoids chain skipping as the chains wrap 180° around each gear. Ground clearance has been inproved, allowing for travel on inclines up to 15%. For additional torque, drive sprockets were selected to create an additional 2:1 reduction after the stock transmission.

The mast holds the camera, GPS units, compass, and computer monitor. With all that componentry, the mast is generally difficult to remove for travel. All connectors have been equipt with quick disconnect wire harnesses to allow 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.

3 Control Software

3.1 Software Development Tools & Environment

While the mechanical and electrical teams were designing and building the robot, it was necessary to be able to test software. Although the previous year's robot, the MCP, was working, it can be difficult for multiple people to test code on one robot. For the past two years, our robotics team has used an open source development platform to build and test code in a virtual world. This approach allows many people to develop software at the same time as well as keeps people from testing unsafe code on the physical robot. The development environment is explained in the following sections.

3.1.1 "Player" Interface Environment

Player is a network interface that is language- and platform-independent allowing software engineers to develop code in any way that is comfortable for them. Code for previous robots have been written in anything from C/C++ to Scheme. All sensors on the physical robot are networked through Player, allowing control software access to both physical sensors and simulated sensors. For each robot, a single driver is developed that provides an interface to each sensor. This driver is accessed through the Player platform which creates a layer between the control code and the driver. The software team is then able to switch between running in a simulated environment to running on the actual robot without making any changes to code, simply by loading a different configuration file.

3.1.2 “Stage” 2D Simulator

Stage simulates the physical robot in a 2-dimensional world which contains various sensor models that the Player server can connect to. A virtual version of the MCP was built in software. The simulated robot contained all of the sensors the real robot has and performs very close to how the real robot does in the real world. Stage is able to provide simulated position data to the GPS sensors on the simulated robot, making it very useful for the navigation team. Test courses were created and the navigation team was able to run the control code quickly and safely without ever having to touch the physical robot. Stage was most widely used by the navigation team working on the GPS challenge. Depictions of simulation can be found in the appendix to this paper.

3.1.3 “Gazebo” 3D Simulator

Although Stage provides a great environment for simulation, it only exists in a 2-dimensional world. Obstacles are defined simply as black lines on a white canvas and can only be detected with sensors like SICK laser rangers and sonars. While this was useful for the navigation challenge, it was not useful for simulating the autonomous challenge, where vision is required. Gazebo provides a three dimensional simulation environment. In this environment, the autonomous team created a course based on the IGVC autonomous course. White lines were drawn on simulated grass, and 3-dimensional barrel obstacles were placed in the path. A model of the robot was created that provided a camera along with the other sensors on Stark. The camera provided images of the simulated environment to the software and the control code could be tested. Depictions of simulation can be found in the appendix to this paper.

3.1.4 “Dwight” Logging System

A logging system has been developed that allows the passing of various types of messages; like errors, warning, general information, and debugging outputs. The platform will send the messages to both the console and over a network to a server so the robot can be monitored remotely. The network component is capable of presenting monitoring data to anywhere from the competition pit area to anywhere over the internet. This will be helpful in determining what goes wrong during a run, since time on the course during and after a run is limited.

Our logging system is comprised of two components, a client and server. The logging client exposes a single function that a developer can access from their control program, this function is called logMsg(). The function parameters are; robot name, control program, message severity, and a message. The function sends a request to the server which stores the message. The logs are updated to a web service, allowing the team to diagnose problems in real time. All sensor information, including diagnostics sensors, could be accessed by low level controls on the robot and logged as well.

3.2 Control Driver

The Player/Stage architecture requires a driver to be written that provides the abstractions to the low-level interfaces of the robot hardware. At the beginning of the development season the previous season's driver was rewritten. A peer code review was used to familiarize additional team members with this critical code segment. The driver was calibrated so that speeds and turn rates specified would be as accurate as possible. Over the last few weeks all of the systems on the MCP have been verified and are functioning correctly.

3.3 Autonomous Challenge

This semester brought a lot of changes to the MCP control code for the autonomous challenge. Originally it was thought that the existing vision code from the previous year was good and needed little work. It quickly became clear that in order to be highly competitive this system would need to be almost entirely rewritten. The driving force behind the rewrite was a need for competitive robustness, which was attainable only when multiple people contribute, check, and combine their work. The new code base is not monolithic like the previous version, where components can be assigned to different people and recombined in a pipeline architecture. With this modulartiy, several new ideas were explored, some of which were incorporated at this time.

3.3.1 Stereovision

Stereo vision was investigated, which has the possibility of providing vision-based distance metrics. There are some robot design that depend entirely on stereo vision for obstacle detection, but that was not the goal of this investigation. Through testing it was found that stereo vision, if used, would be best for identifying landmarks and corresponding them with the laser data, allowing for an auto-calibration and data overlay between the two sensor systems.

3.3.2 Color-based Path Detection

A new approach to the vision problem was chosen. Previously blob detection was at the heart of detecting lines on the field. While this detection worked very well, the data gathered was not sufficient to drive the robot. The robot would easily find itself misinterpreting a line on the left side of the robot as line that belongs on the right side, causing it to steer out of the course. The solution Therrien developed was to not look for the lines but instead to look for a safe path. Color samples of the image are taken from the area directly in front of the robot and then used to find areas farther away that are safe to drive. This is similar to what the Stanford team had done in the Darpa Grand Challenge. [3] By doing this a polygon is generated that defines a safe area. The goal now is to drive towards the center of the polygon. This avoided the problems of confusing the lines, as navigation no longer depended on the lines.

After testing in the simulator, additional features were added to this algorithm, resulting in drastic improvements. Instead of looking at a single point in front of the robot, the sample is taken from a sample area. This area is taken from an image in HSV color space, which prevents shadows from being an issue. A histogram is then created and used to build a back projection image. The back projection image uses the histogram to draw areas on the image in grey scale, areas that match the color in the sample zone are white, areas that are completely off are black, with shades of grey in between. This image is then tresholded to eliminate noise and areas that are unclear. As a final step, the blob detection from last year is used to find just the lines in the image, these lines are then drawn onto the thresholded back projection as an unsafe area (in black). Once all the filtering is done contours of the light areas are found, these contours represent the safe zones for the robot. A zone is then chosen from the list based on a few parameters, first a specific point in front of the robot must be in the contour, this ensures that the robot stays between lines, instead of jumping over them when it's next to one. Of the contours that pass the first test the largest is used to guide the robot. As before the robot just aims to drive towards the center of the contour. This process is shown in Figure 3.3.2. An example with images is shown in Appendix A.

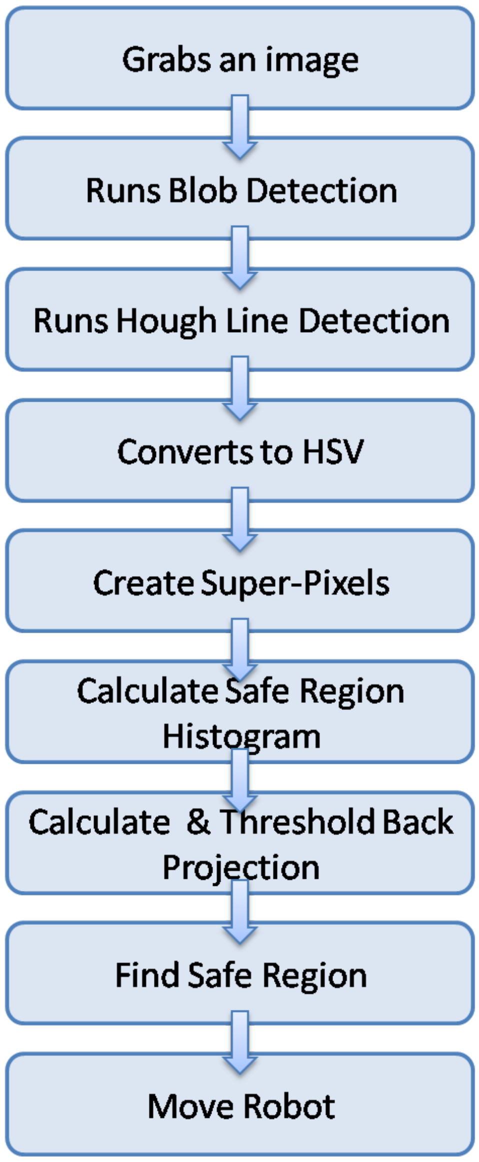

Grabs an image
Runs Blob Detection
Runs Hough Line Detection
Converts to HSV
Create Super-Pixels
Calculate Safe Region Histogram
Calculate & Threshold Back Projection
Find Safe Region
Move Robot

Figure 3: The vision pipeline.

Obstacle avoidance using the laser was a problem that was thought to be solved early on. An algorithm was implmented that used laser data to find a safe area to drive. It would read the laser data and create "arcs" of free space, these arcs were then filtered on size and chosen based on which one could drive us the furthest before hitting an obstacle. In simulation, on a course where the lines were draw as objects the laser could see, the robot drove through the track very efficiently. The goal was to take the safe region found in the vision code and translate it to 3D space, the bounds of this region would be used as obstacles and added to the laser data as if they were there.

Unfortunately, a good way to translate the 2D pixel space to 3D has not yet been found. One possibility is a "birds eye view" transform but it is not mature enough to be in the mainstream code. The idea now is provide a two layer approach. The vision code is actually pretty good at detection obstacles so when driving on it's own it can provide minimal obstacle avoidance. Conjoining this ability with the laser data should be effective. The idea is that when the laser sees and obstacle that is to close to the robot it will take control and perform a wall follow around the object. The direction of the wall follow will be dependent on where the vision code was trying to drive the robot at the time, the thought being that it will be trying to drive away from the closest line and the obstacle. Performing a tight enough wall follow will prevent the robot from going out of the course.

Overall the vision team has made substantial progress in controlling the robot. Currently the code is at a stage where it requires testing and tweaking udner real condtion.

3.4 Navigation Challenge

For the 2010 IGVC challenge the MCP team started with a supposedly stable robot platform to build on. As such, we decided to build a completely new control system with some aggressive design choices. First, we selected an appropriate platform for development, then we developed a variety of algorithms to be tested in the standard two-level mobile robot control architecture of high level mapping and low level reactive (or near-reactive) obstacle avoidance.

3.4.1 An Erlang-based control system

Erlang is programming language and runtime system initially developed for telecommunications applications. Many of the design properties that were necessary for telecommunications are similarly valuable for robotics, including native concurrency, extreme robustness, and a focus on soft real-time performance. The Erlang language is a high level dynamic functional language, which has allowed us to develop software quickly that can be easily modified and maintained.

The only potential drawback to our use of Erlang is computational performance. Erlang does not compile to native code, which results in a performance hit compared to a language like C or FORTRAN. Erlang provides an interfacing system to allow performance-critical portions of a program to be written in a lower level language without compromising the overall robustness of the system. At the time of this writing, the MCP navigation team is still doing performance testing to determine if we need to use this mechanism to speed up portions of our control code. The

3.4.2 Virtual Force Field (VFF) Method for Obstacle Avoidance

The VFF method works by adding vectors. The robot's target exerts an attractive vector and any obstacles the robot spots exert repulsive vectors. The sum of these vectors is used to direct the robot. A diagram of this method is pictured in Figure 3.4.2.

To build these vectors, sensor data needs to be organized and examined. A two-dimensional grid called a histogram grid is built in memory. The robot is situated in the center of an active window of the grid, which extends as far as the robots sensors can reach. The histogram grid represents the robot's model of the world around it.

Each cell of the grid contains a certainty value. A cell that the robot determines is occupied by an obstacle exerts a repulsive force on the robot inversely proportional to the cell's distance from the robot. The sum result of all the repulsive vectors is added to the attractive vector that tries to pull the robot towards its target. The final result is one vector that can be used to turn and move the robot.

Our implementation of VFF was developed in Erlang, and made use of the languages list processing abilities. The grid was represented as a list of tuples, with the first tuple containing the X and Y index values of the grid, with {0,0} being the location of the robot and the other grid locations extending out in positive or negative directions: {0,1},{0,-1},{-1,-1}, {1, -1}, etc... Each grid location represents an area defined by a constant value representing the length of each grid side. Initially the grid side length is 0.2, but this can be changed at any time based on testing. As the laser sweeps, it may hit the same obstacle twice or more within a grid. The larger the grid side length is used, the fewer cells there are and more hits will end up in the same cell. When multiple hits occur in a cell, we use only the largest certainty value for that cell. The 0.2 cell size does result in few duplicates being thrown out.

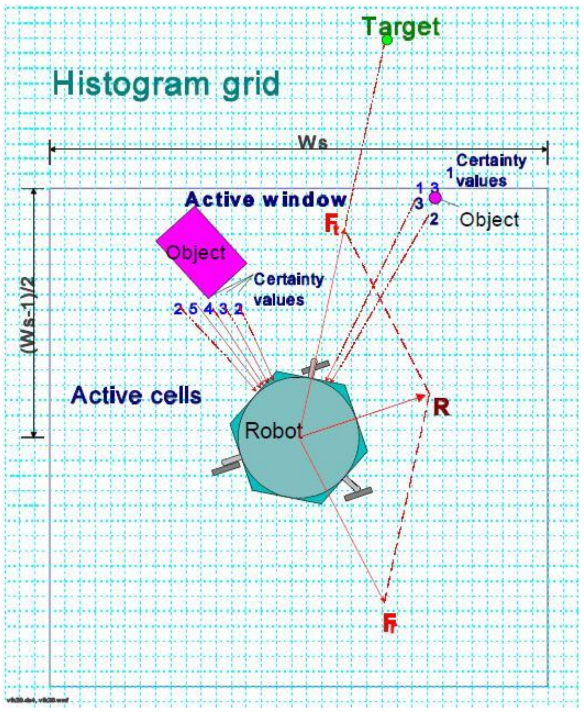


Figure 4: The VFF histogram grid.

During each read/analyze/move loop, we get a list of ranges from the robot's laser. We map the values from these ranges to a list of tuples representing the histogram grid. First, we find the grid position from the distance and angle of the range value. These values are then mapped to a grid position by dividing the distance by the grid constant 0.2. This mapping gives us the X and Y values for each grid cell as an integer. We then filter out all the empty certainty values - the ranges where the laser hit nothing. This leaves us with a list of grid cells that have certainty values, but there may be multiple certainty values for any one cell. Next we filter out all the duplicate hits, leaving a short list of tuples with only the cells that have an obstacle and the maximum certainty value for that tuple.

Once we have this final list of cells with certainty values, we map the values to vectors and the use a fold list method to sum up all the vectors using an add_vector method. This gives us one resulting vector for the repulsive forces. This vector is summed with the attractive vector to produce a final vector that directs the robot.

The VFF implementation was tested using player/stage and performed well at avoiding obstacles in relatively open fields, but had problems getting close to its target when the target was surrounded by close obstacles. The robot would swerve away and go in circles. It would usually reach its target, but would take a long time proceeding over the final few meters.[1]

3.4.3 VFH+

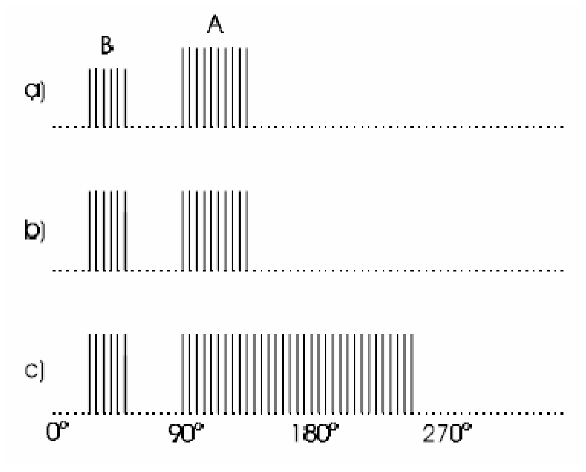


Figure 5: Stages of histogram processing: a) Primary polar histogram, b) binary polar histogram, c) masked polar histogram.[4]

The VFH+ method uses a four-stage data reduction process to find the best direction of motion. Speeds can then manually be calculated based on the turn rate or goal distance. In the first data reduction stage, the grid map from the VFF algorithm is taken and reduced down into a polar histogram. This histogram is a one-dimensional array of values representing 180 degrees of laser readings. Each laser reading is given a single value for "polar obstacle density" which is based on the certainty of an obstacle in that direction. The obstacle density is also raised depending on proximity and width. The second reduction stage takes this histogram and changes each value to a 1 or a 0 based on high and low thresholds. This is called the binary histogram. The third data reduction stage takes the robot's turn radius into account. For example, if the algorithm thinks there is an opening on the extreme left but in reality the robot could not make the turn, then that opening is marked as blocked. This reduces shaky turns and indecisive behavior. I simplified this third step significantly since our robot has the capability of differential turning. I slow down our speed as the angle needed to turn becomes larger. As the direction to turn begins to go behind the robot, our speed approaches zero.

The fourth and final data reduction stage involves selecting the best steering direction. This is the section that took by far the most work to tweak. For each value, if the opening is narrow then the center angle is added to the list of possible directions. If the opening is wide then a direction on each side is chosen. The target direction is also added to the list if it exists in an opening. Each direction in the list is then given an overall cost based on the absolute difference between the direction and the target direction, the wheel orientation, and the previously selected direction of motion. Directions closer to the target goal are almost always given the best cost to maintain a goal based behavior. In the end, VFH+ worked smoothly and efficiently. It can go between waypoints quicker than VFF and its motion is much less wobbly.

3.4.4 Path planning: A journey of its own.

Our first attempt at global path planning for our new control system was to simply implement A* search. We expected this to run slowly but just barely usable, and that's exactly what happened.

Once we had working A*, we tried a variant algorithm called D* which saves state information between searches to speed things up. This algorithm is very popular at the moment; for example, it is is used on the Mars rovers for path planning. Surprisingly, for the navigation challenge, this provided worse performance than A*. The problem was that D* does extra bookkeeping all the time in order to speed up repeated searches, while the main performance issue in the navigation challenge is a pause for an initial search at a new way point; with the extra bookkeeping done by D* this initial search is actually slower than an A* search.

In order to combat this issue, the team is now working on an incremental topological mapping technique based on Poncela et al. This system may or may not be ready in time for the competition; if not, good old A* will be finding our paths.[2]

4 Future Work

Alongside the software development for this year's MCP entry, UMass Lowell has also constructed a new hardware robot platform. The development of this new robot brought many new concepts that can also be applied to the MCP. This section outlines a few key technologies that will someday be included on the MCP.

4.1 Motor Controller Interface Board

The Motor Controller Interface Board (MCI) was designed to replace three separate existing systems: the motor controller communication breakout, the emergency stop, and the remote/computer input switcher. One unit is currently in service in UMass Lowell's new robot, but was designed to be installable into the MCP as well. This new board acts as a single fail-safe point of contact to the Roboteq Motor Controller, both fulfilling safety requirements and maintaining a flexible interface for controlling the robot. Having all of these systems combined into one point of failure allows us to make sure that the vehicle will fail-safe should anything happen to the radio controller or computer. If a wire were to come loose from the emergency stop system, or if the MCI board loses power for any reason, the motor controller will immediately be e-stopped and the robot will stop moving.

For the wireless e-stop, a 433Mhz 4 channel transmitter and receiver were chosen. The OEM wireless transmitter is capable of 300 feet, far exceeding the requirements of the IGVC.

With this upgraded board, the improvements to the MCP include better E-stop transmission range, simpler diagnosis and maintenance in the field, and a much smaller (one-third size) device footprint.

4.2 Power System simplification

The two, independant power systems described in Section 2.1.2 have created a long history of design challenges. That architecture was inhereted from previous versions of the MCP, where it solved an issue of the time: heavy motor draw would dim the voltage to the electronics, causing a premature shutdown. Today that is no longer an issue, as it has been shown that single large battery system coupled with high-efficiency DC-DC converters can provide power to all parts of the robot. A single battery system allows the robot to be considerably lighter, and the batteries may be relocated within the MCP's frame to improve the center of balance.

References

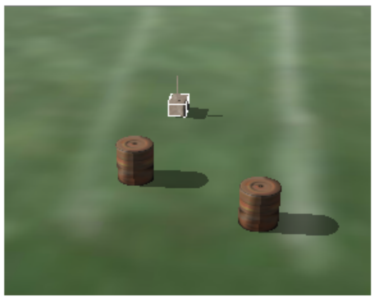
[1] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots. 1989.

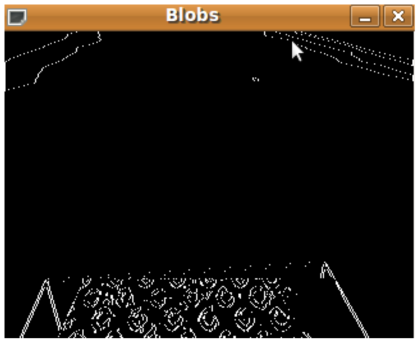
[2] A. Poncela et al. Efficient integration of metric and topological maps for directed exploration of unknown environments. *Robotics and Autonomous Systems*, 41:21, 2002.

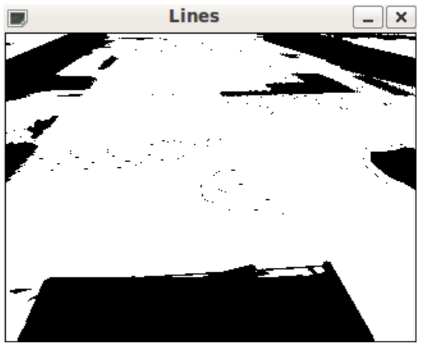
[3] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. Winning the darpa grand challenge. *Journal of Field Robotics*, 36:1–43, 2006.

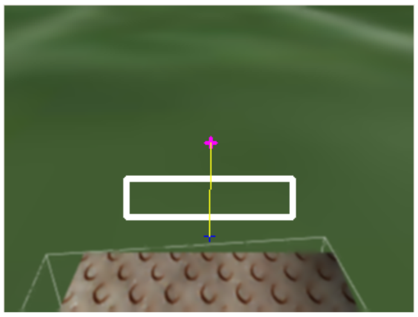
[4] Iwan Ulrich and Johann Borenstein. VFH+: Reliable obstacle avoidance for fast mobile robots. *IEEE Conference on Robotics and Automation*, 1572-577, 1998.

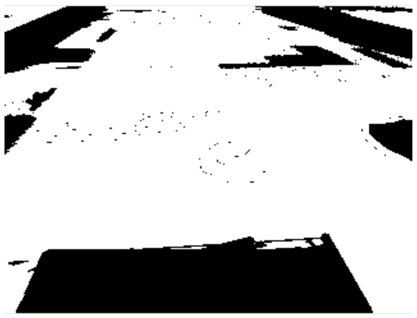
A Vision Pipeline Example

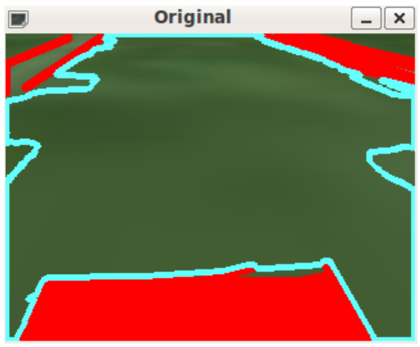












The upper left image depicts the robot on a simulated field in Gazebo. For more information on Gazebo, see Section 3.1.3. The next image, in the upper-right, shows the result of the blob detection algorithm to detect lines on the course. Those lines are then showed in the back projection in the next image. The fourth image shows the input from the simulated camera after a blur is applied. The rectangle shows the area that is sampled as "safe" colors, and is projected with the lines, as shown in the bottom-right image. The final image shows the original camera image with the safe and unsafe zones superimposed.

B Component Costs

SICK LMS200	Laser Rangefinder	$5,800
Intel Core 2 Duo	3.0Ghz dual-core processor	$168
Computer components	Motherboard, RAM, etc	$292
Apple iSight	FireWire camera	$200
LCD Monitor		$100
Roboteq AX3500	2-channel, 60-amp motor controller	$395
Invacare 1085952	Motors with gearbox assemblies	$1390
Drivetrain components	Chain, sprockets, idlers	$173
Chassis materials	80/20 extruded aluminum	$500
Custom circuit boards	Parts and manufacture	$95
Remote E-Stop system		$40
Samlex SEC12-15	15A 12V Battery charger	$149
Batteries	AGM Lead-Acid	$300
Vehicle base cost		**$9602**