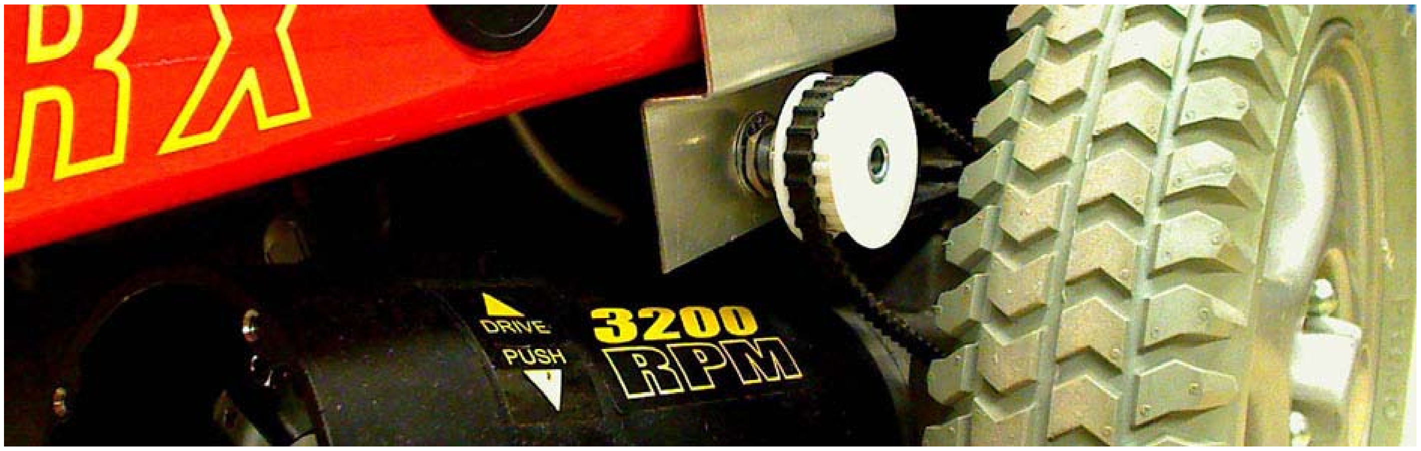


HARLIE

HARLIE Autonomous Real-time Linux-enabled IGVC Engine

CASE WESTERN RESERVE UNIVERSITY EST. 1826

I, certify that the engineering design in the vehicle (original or changes) by the current student team has been significant and equivalent to what might be awarded credit in a senior design course. There have been significant changes to both hardware and software on the robot.

[illegible] 5/14/2010

Signature Date

I. Mechanical Design

The base is from the Ranger X model wheel chair from Invacare's Storm series wheel chairs. The wheelchair base provided our team with a rugged base equipped with suspension and motors. The suspension system will help to stabilize the upper frame unit to reduce noise in sensor values, which is a desirable quality for an outdoor robot. Also, since it is a wheelchair base designed for patient care, the base and motors are manufactured so that there is little if any room for failure. The wheelchair base also proves capable of climbing over or through most obstacles such as ramps or sandpits, as disabled users require the power chairs to do such tasks daily without fail.

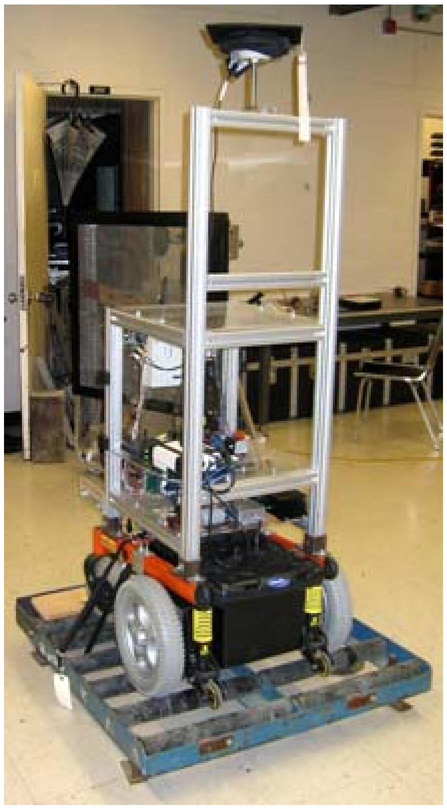


Figure 1 - Partially constructed Harlie used for testing mechanical structure and wiring against vibration damage.

The frame was designed and built using 4 cm square aluminum framing. Designing a method for mounting the frame to our base was one challenge previous teams had to overcome. As manufactured, the base was sloped down towards the casters. Using a simple mount would cause our frame to also lean at this angle. To overcome this problem our team developed custom mounts that would allow us to level the frame on top of the sloped base. Having a level frame made mounting the LIDARs easier since they were required to be mounted parallel with the ground.

Previous teams had also attempted a vertical plane approach but wire routing between the two bisections of the robot was difficult and components were able to become loose and fall from the central vertical Plexiglas wall, causing damage to several components. This year, we returned to the more compact horizontal shelves which are more difficult to access but yield more reliable robots that need to be serviced less often.

II. Electrical System

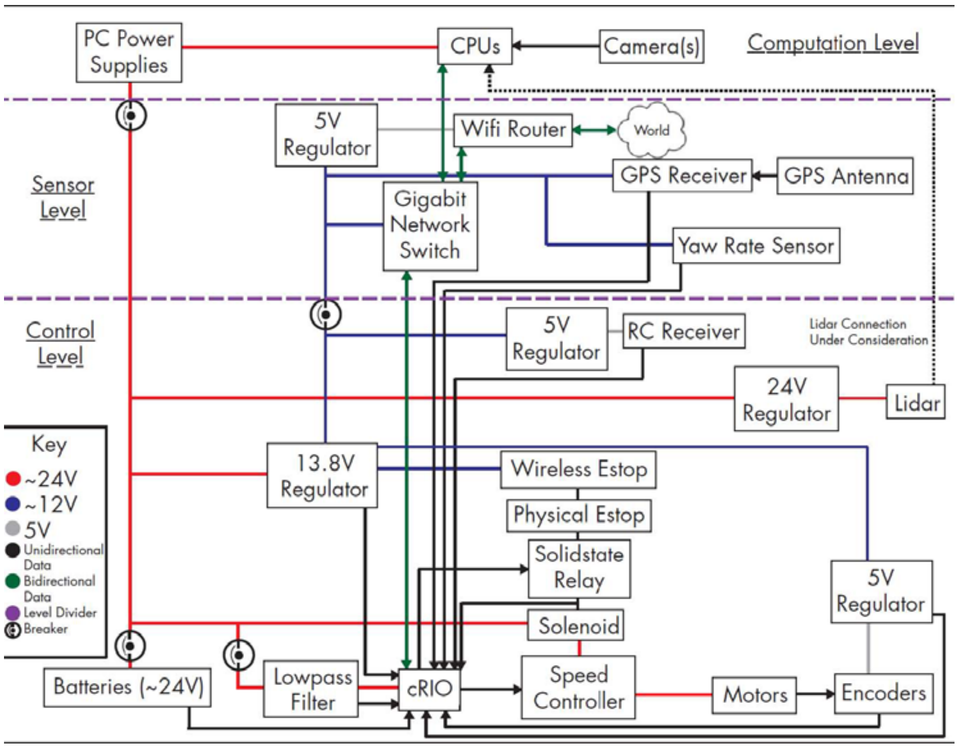


Figure 2 - Electrical System Block Diagram

The power distribution for Harlie was completely redesigned this year. After experiencing failures with the Carnetix supplies through using them as a source of centralized 5V power generation, it was decided that 5V generation, when needed, will be regulated at the destination from the 13.8V power. Our power redesign focused on distributing the unregulated 24V from the batteries as well as a regulated 13.8V level. We then used 7805 5V regulators on a custom designed pcb to regulate 5V on site and used a special 24V regulated power supply for the LIDAR. A low pass filter was constructed using a 1 ohm power resistor, a diode, and a 10mF capacitor to protect the cRIO from sudden drops in voltage from the motors.

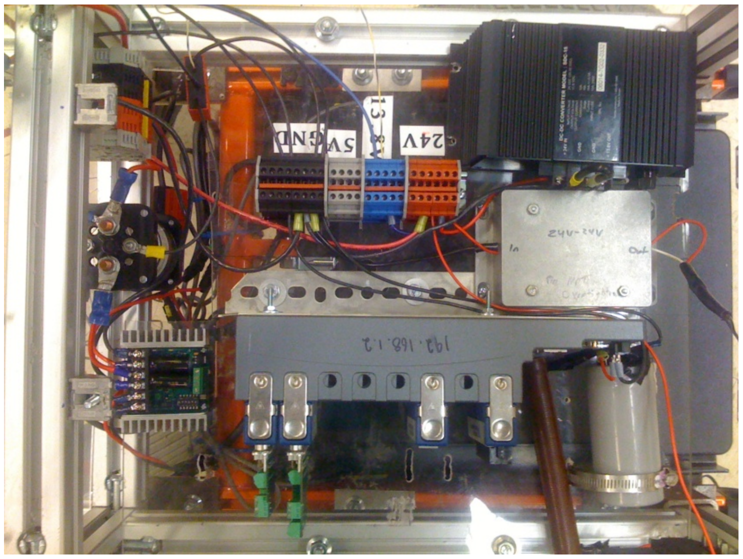


Figure 3 - New main shelf and color system

The format for the power also set and followed a strict color safety code to avoid the confusion and hazards of previous designs. This method was simple but has proven to be the single most effective safety and reliability change this year. We simply mapped voltages or respective levels to a key, such that 24V was red, 13.8V was blue, 5V was grey, and ground was black. Signal wires were unique wiring patterns based on the spare wire at our disposal, but were not allowed to be solid color unless representing a signal ground.

We also added some maintenance improvements this year. This includes the use of colored, one way power connectors to connect each shelf or component to the central power generation on the lowest shelf. These connectors add a safe, color coded, and removable way to easily debug power or signal issues between shelves by quickly and reliably isolating problems. We have also added larger power distribution blocks to feed the more power hungry components individually. These blocks can handle significantly more amps than the smaller blocks which makes them more suitable for splitting power off to the motors and the computer.

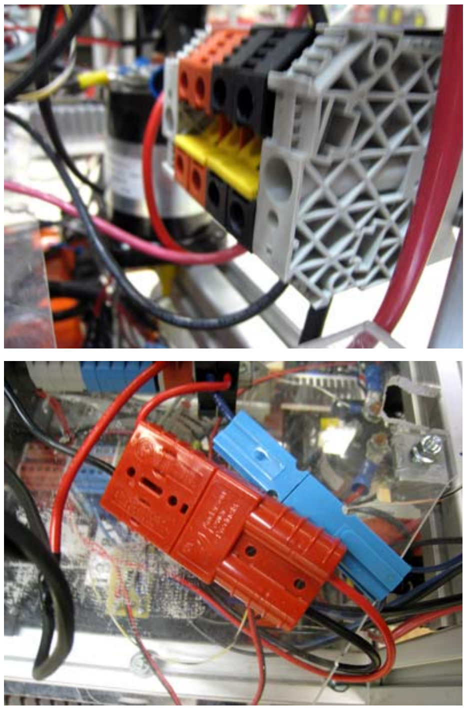


Figure 4 - Safety and Reliability Additions to Power Distribution

We also put our breakers to better use. We kept the 120A thermal breaker as a main power switch and safety, but also added 10A breakers to the secondary sensor shelf. This allowed for easy powering of components as well as safety in case something were to short, but not trip the main robot breaker. We also added brightly colored indicator LEDs to show the current status of each shelf. If 24V and 13.8V are not present at the shelve, then these LEDs will be disabled, quickly letting us know we've forgotten to switch something back on after maintenance or testing.

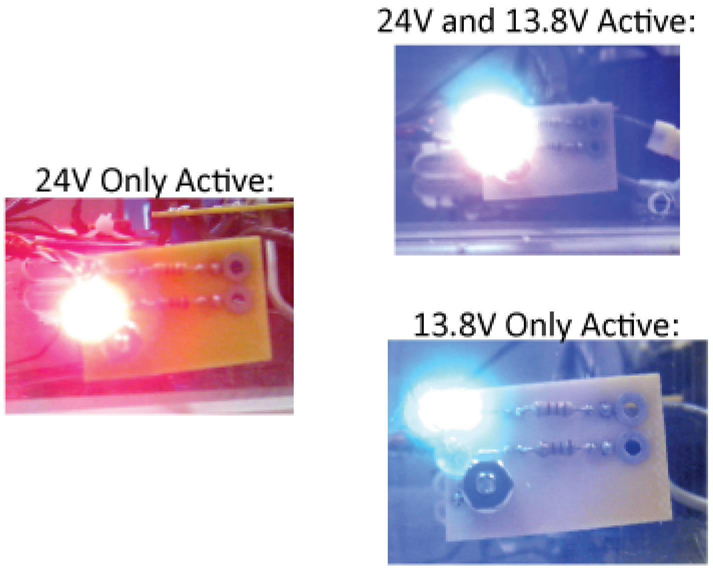


Figure 5 - Shelf Power Indicators

While we did add a large inverter and modern computer to Harlie, we still achieve roughly 3 hours of runtime from our set of 2 year old Optima Blue Top batteries.

III. Safety

A number of additional safety features were refined or added this year. Harlie features three emergency stop actuators: a push button, a wireless toggle, and a software controlled estop. These are interfaced to a high current solenoid that provides power to the motor controller only when all three are active and if a single one determines unsafe behavior or is activated, then the motors will be completely powerless.

The most important new safety feature is the addition of a bright strobe indicator on the robot. This light serves as a beacon to let both the operators and passers-by that the robot is active and running. The very bright flashing is also a constant reminder to disable the motors for both the safety and sanity of programmers.

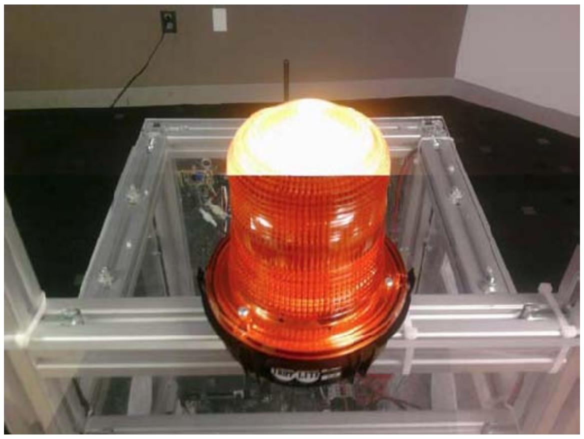


Figure 6 - Safety Light in both off (lower) and mid-strobe (upper).

IV. Sensors

We use quite a number of sensors on our robot since it was designed for both IGVC and general mobile robotics research. These sensors are as follow:

Quadrature Encoders

Encoders form the basis for our velocity estimation. We use encoders placed near the wheels to determine our robot's velocity for position estimation and we use encoders placed on the motor shafts for PID speed control. Because these encoders are quadrature, we get high resolution and directional information from the two square wave signal outputs. These two signals yield better results than using single direction encoders and needing to guess or have another sensor to determine direction.

Analog Devices Gyro

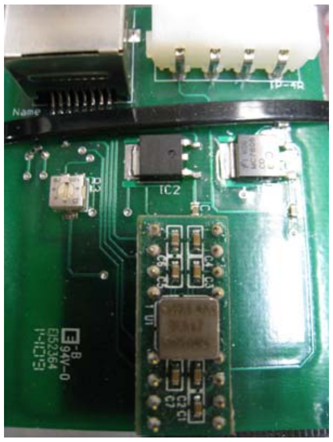


Figure 7 - ADXRS150 on Custom PCB

We use the Analog Devices 150 degree/s gyro to determine our current angular velocity. This sensor was tricky to calibrate as the zero reference is dependent on temperature. Thankfully, the sensor outputs its current temperature, and we were able to model the bias versus temperature. This is an improvement over past years, as before we had to wait 10 seconds while booting to calibrate the bias. This is no longer needed, as the sensor is ready to go instantaneously and can now also correct for a changing bias.

GPS

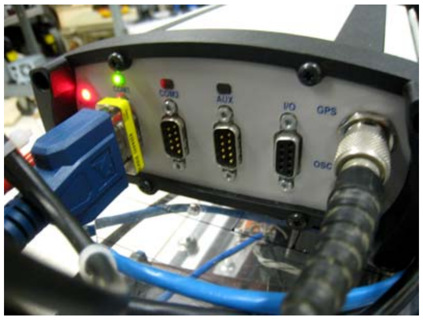


Figure 8 - Novatel Propak V3 HP

We use the Novatel Propak V3 HP as our GPS receiver. This receiver yields very accurate GPS information in open spaces, but can have issues with some obstacles such as tall buildings or trees. Omnistar HP correction allows a theoretical precision of < .1m, but we typically see precisions of roughly .4 m RMS.

Compass

Figure 9 - SICK Laser Range Finder (LIDAR)

New this year is the addition of a PNI CompassPoint Prime. This compass was generously donated to us by PNI Corporation. The main use for the compass is to determine our orientation in three dimensional space. It uses three axis compass and accelerometers to provide us with an accuracy of 1 degree RMS in roll, pitch, and yaw. This is very useful for determining when we are on unlevel surfaces as well as providing heading information while GPS is unavailable, inaccurate, or the vehicle is stationary.

LIDAR

Harlie avoids obstacles by the use of SICK Light Detection And Ranging (LIDAR). It uses a rotating mirror to emit an infrared beam within a viewing angle of 180 degrees. With the return of the reflected beam to the LIDAR, the range to obstacles can be detected. SICK has the capability of working in the range of 10 cm to 80 m and can provide data at a rate of 500kbaud. Another positive feature is that the LIDAR is an active device as it operates without an external illumination. The main limitations to the SICK LIDAR are size, power stability requirements, detection of low reflectivity objects, and rough terrain/ramps driving the plane into the ground.

Firewire Cameras

The Firefly MV from Point Grey Research was used to capture the image stream to detect lines and obstacles. This camera uses a 1/3" wide-VGA CMOS sensor from Micron with a 752 x 480 resolution and a 1394a digital interface. A Senko lens was secured to the camera and adjusted to maximize the depth-of-field and provided for the widest angle. This wide angle lens results in image distortion that must be corrected, but is easily compensated for using the OpenCV computer vision library.

Figure 10 - Firefly MV Camera

V. Lower Level Software - National Instruments Compact RIO (cRIO)

This year, we split the software into two main sections: a real-time sensor based section to be run on the National Instruments cRIO's FPGA and PowerPC processors and a higher level planning and image manipulation section to be run on any networked computer.

The cRIO is responsible for gathering data from sensors, issuing commands to the motors, and then filtering this data to gain an estimate of current position.

FPGA Sensor Readings

The most useful feature of the cRIO is the very large and powerful 40MHz FPGA onboard. This interfaces through the cRIO's logic modules to digital inputs, digital outputs, analog inputs, and a serial interface. The largest change here from last year is the use of the FPGA for as much as possible. This not only frees up the real-time PowerPC for computations, but also runs reliably and safely since there is dedicated hardware for each sensor. We programmed the FPGA in LabVIEW and currently read in all of the low level sensors through the FPGA, including all four encoders, the analog gyro, voltage monitoring of the central power generation, and even parse the serial bytes from the GPS and the compass into hexadecimal representation of floating point numbers for transport to the real-time target.

Motor Commands, PID Generation, Speed Control, and Radio Control

Our FPGA is also responsible for physically moving the robot. The current configuration receives commands from the trajectory planner as a left wheel speed and a right wheel speed. These are then sent through a PI feedback loop using the motor controllers. We chose to not use the programmed Derivative feedback, as an overdamped system performs more safely and with more predictable motion than a critically damped system. This results in the robot quickly reaching its desired speeds but without very rapid acceleration and possible oscillation.

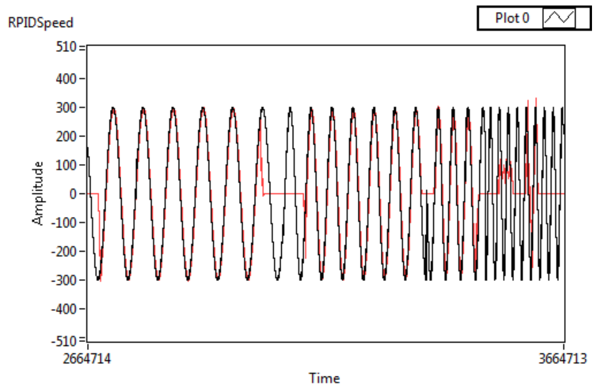


Figure 11 - PID Command (Black) and Motor Output Speed (Red) - Note failure at very high frequency on far right.

For safety, the FPGA continuously monitors the change in motor encoders from one PID iteration to the next. If this value goes over a set speed limit: usually 1.2m/s, but will change to 2m/s for IGVC, then the value will be limited. If anything were to happen such that the FPGA loses control of the motors or power is lost to the FPGA but not the motor controller, the FPGA has an output that controls a solid state relay connected in series with the other emergency stops. This prevents software or hardware crashes from becoming dangerous as the robot will simply stop and not continue forward.

The FPGA also handles interpreting the pulse width modulated signals from an RC receiver. This can be used to drive the robot by hand without the need for any other software or computer to be running or operational.

Localization and the Physical State Observer (PSO)

The last task for the cRIO and the only task for the PowerPC is to filter the sensor readings reported by the FPGA and report the current best guess plus Gaussian distributed standard deviations of error on that position. For this task, we are currently using an Extended Kalman Filter.

A Kalman Filter is an optimal linear filter and applies for any signals and estimated values that are linear and have zero-mean Gaussian distributed noise in the system. Applying this filter to non-linear systems yields poor results, so Extended Kalman Filters can be applied to approximate a non-linear function as a linear model using the Jacobian matrix (first derivatives) to estimate the covariance matrix of the nonlinear function. Our equations are as follow:

$Predicted\ state\ \boldsymbol{xp} = nonlinear\ function \rightarrow\ \boldsymbol{f}(\boldsymbol{previous\ state})$

$Our\ state\ consists\ of\ [x\ position, y\ position, heading, velocity, angular\ velocity]$

$Predicted\ covariance\ matrix\ \boldsymbol{Pp} = \boldsymbol{F} * \boldsymbol{P} * \boldsymbol{F'} +\ \boldsymbol{Q}$

$Measurement\ residual\ \boldsymbol{y} = \boldsymbol{zk} - \boldsymbol{h}(\boldsymbol{previous\ state})$

$Our\ sensors\ include\ [gpsX, gpsY, compass\ heading, left\ encoder\ change, right\ encoder\ change,$
$\ and\ current\ angular\ velocity\ from\ the\ gyro$

$Residual\ Covariance\ \boldsymbol{S} = \boldsymbol{H} * \boldsymbol{Pp} * \boldsymbol{H'} + \boldsymbol{R}$

$Kalman\ Gain\ \boldsymbol{K} = \boldsymbol{P} * \boldsymbol{H'} * \boldsymbol{S^{-1}}$

$Final\ State\ Estimate\ \boldsymbol{x} =\ \boldsymbol{xp} + \boldsymbol{K} * \boldsymbol{y}$

$Final\ Covariance\ Estimate\ \boldsymbol{P} = \boldsymbol{Pp} - \boldsymbol{P} * \boldsymbol{K} * \boldsymbol{H}$

$f\ is\ a\ nonlinear\ model\ of\ the\ dynamics\ of\ the\ system$
$F\ is\ the\ Jacobian\ matrix\ of\ the\ update\ model\ f$
$h\ is\ a\ nonlinear\ model\ of\ the\ predicted\ sensor\ values\ according\ to\ the\ predicted\ state$
$H\ is\ the\ Jacobian\ matrix\ of\ the\ sensor\ model\ h$
$zk\ is\ the\ actual\ measurement\ taken\ by\ the\ sensor$

The most obvious problem with the Extended Kalman filter is that it is a non-optimal estimator of state. We often see cases where the highly non-linear nature of rotation makes the covariance matrix break down for a few iterations and underestimate the true variance in values until the rotation ends and a new linear approximation is converged. Despite these shortcomings, our localization is very precise (within .1m) as long as the GPS receiver's variances that are reported are correct. Omnistar HP tends to underestimate the true error in its position if obscured by obstacles.

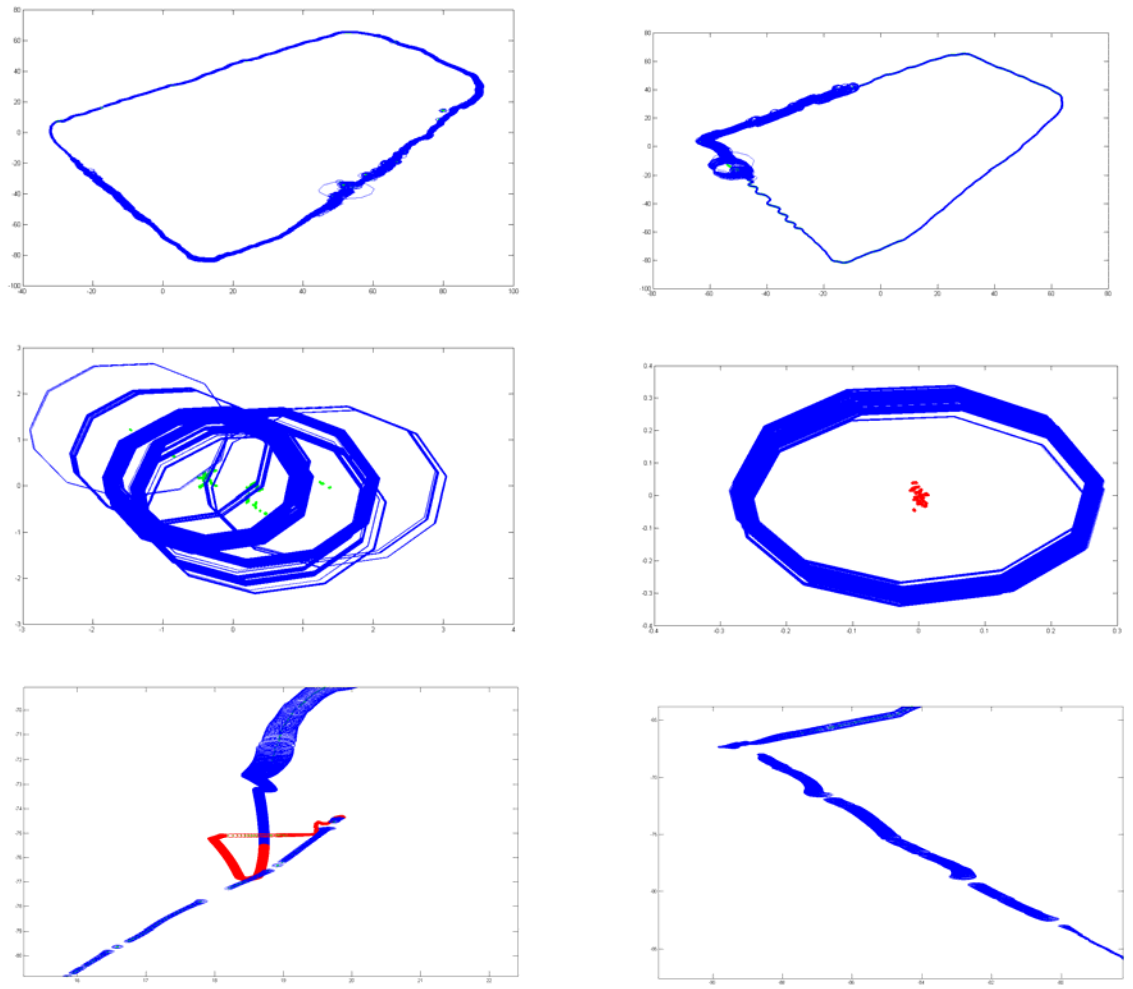


Figure 12 - Test Run Variances Without (Above Left) and With Omnistar (Above Right)
- Stationary GPS points and variance (Center Left) and Omnistar points and reported variance (R)
- Estimated Values while obscured by building (Lower Left) and Obscured by large tree (Lower R.)

VI. Upper Level Software

The higher level computation is preformed on an Intel i7 based computer over the network. The Physical State Observer reports the position of the robot over the network to ROS (Robot Operating System), which then also takes input from the LIDAR and the cameras to make decisions about how and where to travel.

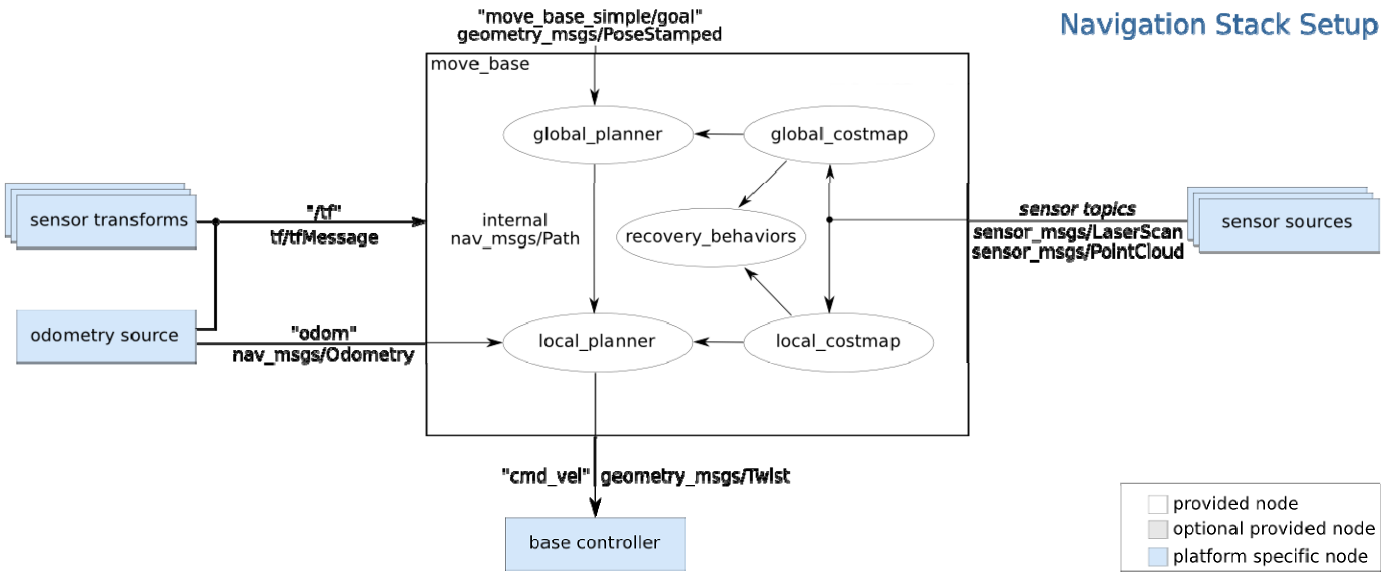


Figure 13 - Upper Level Software Architecture

Plan for Path Following

In order to follow a path defined by an ordered list of waypoints, we have a three step process. First, we get the first waypoint off the list and give it to our Global_planner. In the second step, the Global_planner uses that desired destination, our current position and the obstacle map to compute the single-source shortest path through the map. See Figure 12 for a visualization of the shortest path in an indoor environment. In order to do that, we use Dijkstra's algorithm. We take this shortest path and pass that off to the

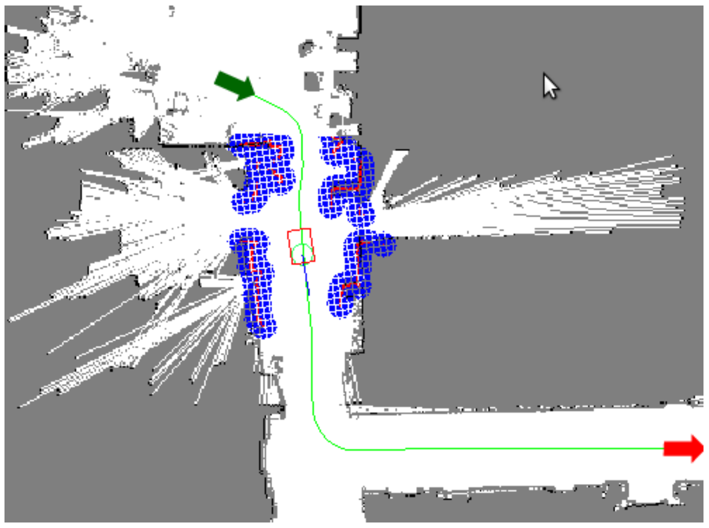


Figure 14 - Global_planner Visualization

Local_planner (see Plan for Control Decisions section) for further processing. If the obstacle map or our current position deviates too much from their values when the shortest path was calculated, that path is invalidated and recalculated. Once we have reached our goal within tolerances (0.5 meters), the third step in path following is to remove the top waypoint that we just completed from the ordered list and return to step one until the ordered list is empty.

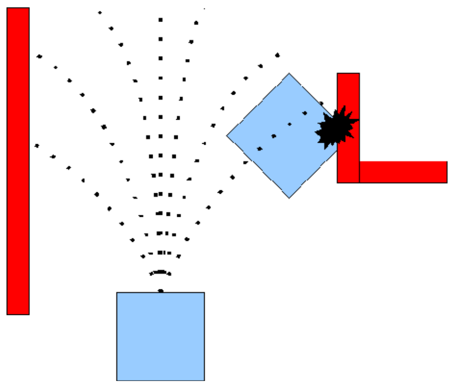


Figure 15 - Trajectory (local_planner) Illustration

Plan for Control Decisions

In our system, control decisions are handled by the Local_planner. This subsystem receives a shortest path from the Global_planner and obstacle map information and is designed to follow that path as closely as possible, while minimizing the amount of time it takes to safely traverse that path. See Figure 13 for a graphic of the results of the forward simulation done by the Local_planner for a number of possible controls. Our Local_planner uses the Dynamic Window Approach (DWA), originally described in *The dynamic window approach to collision avoidance* by D. Fox, W. Burgard, and S. Thrun. In short, DWA uses the following 5 step process:

1. Discretely sample in the robot's control space (v, omega) where v is the translational speed of the robot and omega is the angular speed.
2. For each sampled velocity, perform forward simulation from the robot's current state to predict what would happen if the sampled controls were applied for some short period of time.
3. Evaluate (score) each trajectory resulting from the forward simulation, where a high score is awarded to the set of controls that maximizes proximity to the goal, proximity to the global path and speed, minimizes proximity to obstacles, and does not result in a collision with an obstacle.
4. Pick the highest-scoring trajectory and send the associated controls to the steering subsystem.
5. Go back to step one and repeat

How the vehicle deals with complex obstacles including switchbacks and center islands, traps and potholes

In order to deal with complex obstacle arrangement and environments, we retain obstacle information from system initialization. In essence, we do not forget what we have previously seen. In order to do this efficiently, we maintain two obstacle maps. One is a 6 meter by 6 meter square rolling window centered on the robot and is used by the Local_planner when forward simulating possible control vectors. In order for the Local_planner to update quickly, we need to keep this map reasonably small. In contrast, we use a 100 meter by 100 meter map that has a fixed origin at our initialization position; this map is not a rolling window and is meant to accumulate knowledge about the environment as we run a given course. By building off our previous obstacle maps, each time the Global_planner is run it has more and more information available to it. Because of this, we are able to remember the complex obstacle, such as a switchback and, after sensing the impassable end, plan back around the switchback in order to continue on to our goal. The same logic applies for other complex obstacles that are detected and placed in the obstacle map.

Accuracy of arrival at navigation waypoints

By using the Omnistar HP high-precision DGPS service, we are able to achieve very good accuracy when reaching a navigation waypoint. Under optimal Omnistar HP, we can reach one of these points within 0.05 meters. Under normal GPS conditions, we can reach a navigation waypoint within 0.5 meters.

Distance at which obstacles are detected

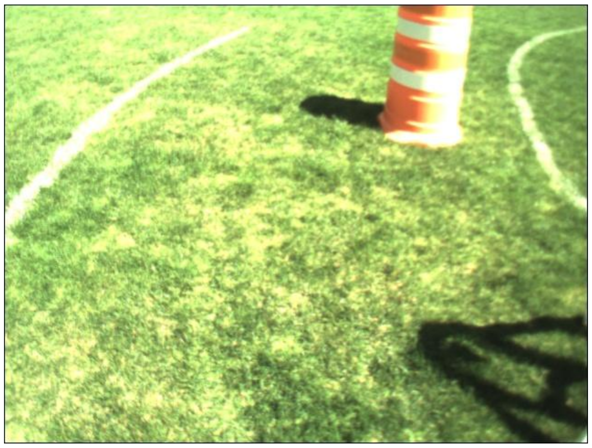
Using our SICK LIDAR, we are able to detect obstacles at a range of up to 80 meters with centimeter accuracy. In order to reduce the number of false detections that could occur from very distant pings bouncing off a hill or other change in ground elevation, for example, we limit the distance to approximately 25 meters. This is still plenty of distance to allow the navigation subsystems time to plan around obstacles and maintain a high rate of speed along the entire path.

Reaction times

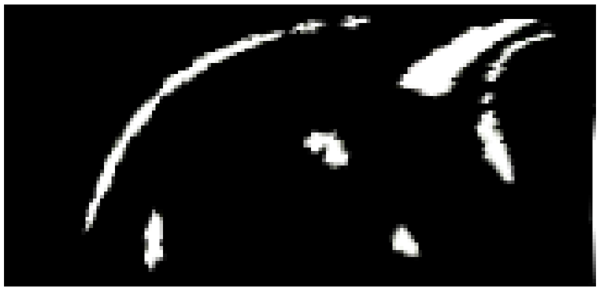
The Local_planner, responsible for generating the control vectors for the robot, runs at a rate of 20 Hz. This allows us to react to an obstacle within 50 milliseconds of detection, which, at our upper speed limit of 5 miles per hour, is plenty of time to either avoid the obstacle smoothly while following the existing planned path or stop and allow the Global_planner to regenerate shortest path information. In addition to this, the steering subsystem, running at a rate of 50 Hz, has access to a sonar sensor for ranging information. While a single sonar is too inaccurate for mapping effectively, it is very useful for collision detection; the steering subsystem uses its sonar to halt forward movement of the robot if an obstacle is within a half meter while still allowing rotation. By allowing rotation, but not translation, we are able to steer away from the obstacle while guaranteeing that we do not hit it.

Vision

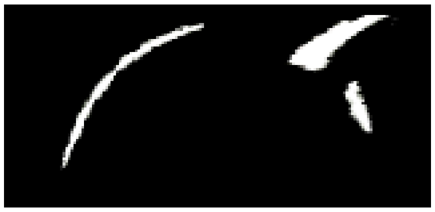
Our vision algorithm in development currently uses the following steps:



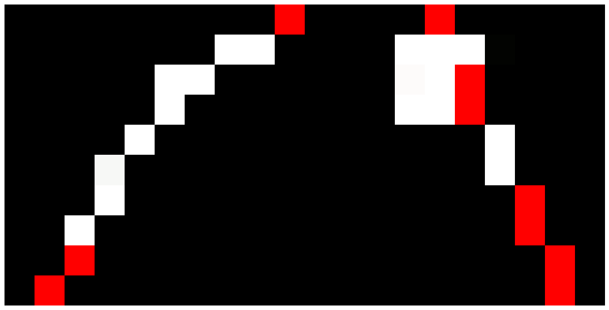
1. Import raw image from camera.



2. Transform into Top Down View, Crop, and Threshold based on texture -> Uniformity of local area. This allows us to properly isolate the white lines and any other unnatural objects which are likely to be obstacles.



3. Remove blobs that are of a size less than a certain threshold. This allows us to ignore patches of dead grass or other small areas while generally preserving the structure of the lines.



4. Extend directional blobs based on elongation factor and resample to import to obstacle map. This final step allows us to plan through both solid and dashed lines. The resampling allows us to easily export the white lines as x and y pixel locations offset from the robot. The increase in size of the lines is purposeful in that it expands the location of lines and barrels such that the original width is roughly preserved.

VII. Cost List

Part Description	Retail Price	Cost to team
Wheelchair base	$3,000	$0
Batteries	$400	$400
Bosch Aluminum Framing	$1,000	$1,000
Power Converters	$430	$430
Electrical Components	$850	$850
Wireless EStop	$490	$490
Sabertooth Speed Controller 2x25	$125	$125
SICK LIDAR	$6,000	$6,000
i7 Server	$700	$700
Network Equipment	$40	$40
Yaw Rate Gyro	$80	$80
Quadrature Encoders	$320	$320
CompassPoint Prime	$400	$0
Sonar	$80	$80
RS422-USB Converter	$80	$80
FireFly Camera	$350	$350
Camera Lens	$150	$150
RC Controls	$500	$500
NovaTel ProPak GPS	$5,490	$2,700
cRIO and Modules	$4,300	$4,300
Total	$19,115.00	$15,925.00

VIII. Team Harlie

Chad Rockey - Electrical Engineering - Team Lead

540 Hours

Eric Perko - Computer Science - Software Lead

500 Hours

Ben Ballard - Computer Engineering

400 Hours