University of Minnesota IGVC Design Report 2014

University of Minnesota – Twin Cities
Bryan Stadick, Max Veit, Adam Juelfs, Alex Dahl, Terry Furey, Frans Elliot
Dr. William Durfee – wkdurfee@umn.edu

INTRODUCTION

This year, GOFIRST and the University of Minnesota are proud to enter the 22nd annual Intelligent Ground Vehicle Competition with their entry Ground Squirrel. Ground Squirrel was designed and built by a self-motivated, self-run conglomerate of students at the University of Minnesota - Twin Cities. The team, under the student organization header GOFIRST, consists of students from several different majors across the University. This is the group's second year entering IGVC. The previous year's robot is being used for the 2014 competition. While no major hardware changes have been made, a few electrical changes and sensors have been added along with a completely new software base.

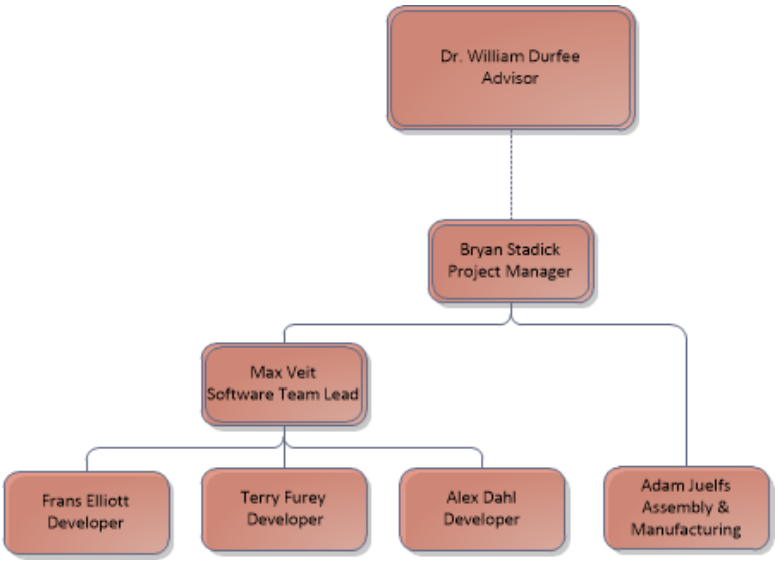

Figure 1. Structure of team.

TEAM ORGANIZATION AND DESIGN PROCESS

Team Organization

The core team consists of six members from varying majors. The team organization consists of a project leader and one sub-team leader covering software. The project leader's responsibilities consist of keeping the project schedule, organizing meetings and members, tracking the team's financial status and doing any other administrative tasks. The software leader is then responsible for assigning tasks to members, providing guidance and assistance in their respective specialty area and ensuring the progress is being made toward the final product.

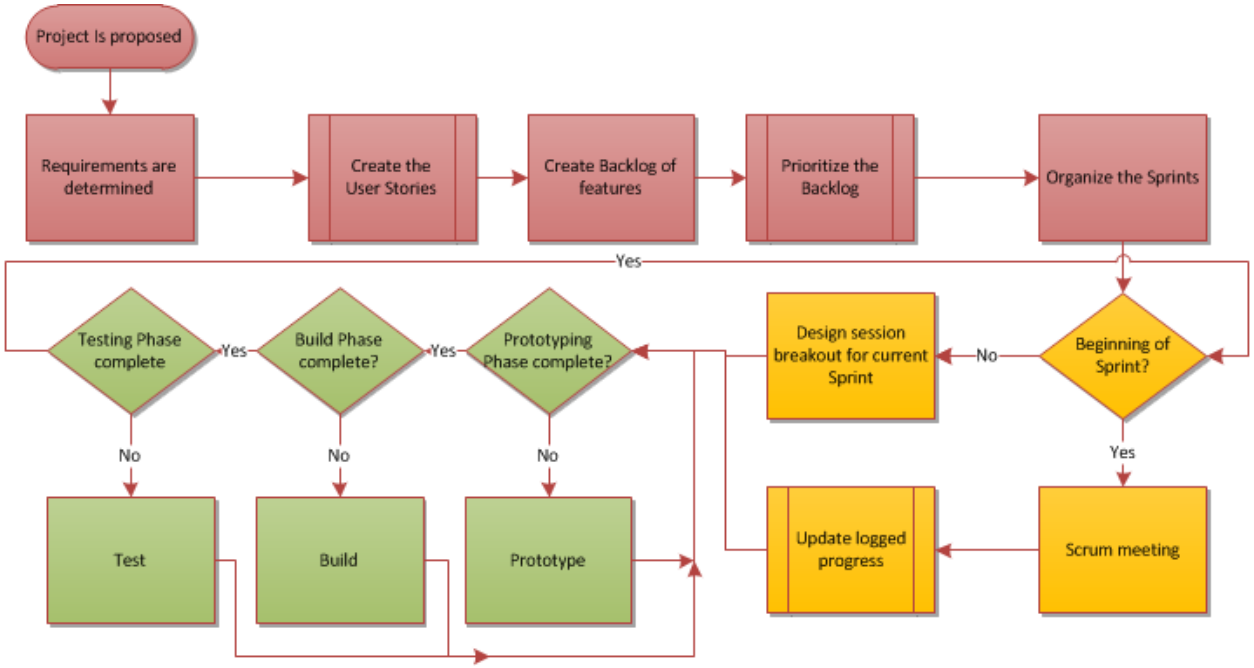

Figure 2. Structure of a SCRUM sprint.

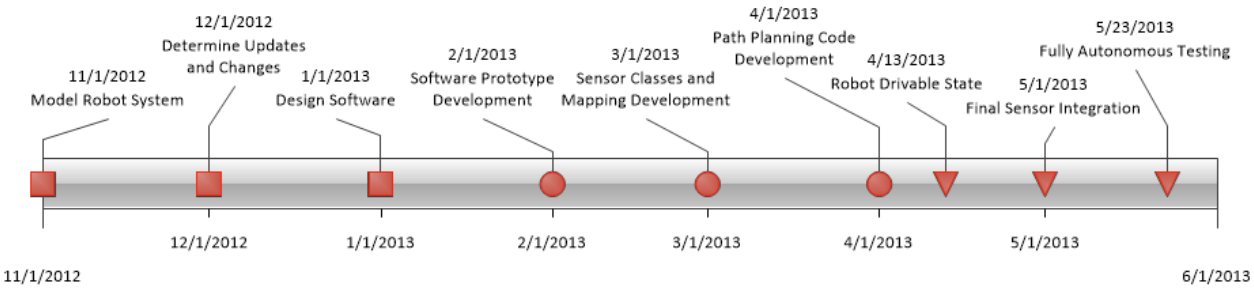

Figure 3. Timeline of sprints.

Design Process

In the design and construction of Ground Squirrel, an agile development methodology was followed. Usually used in software development, we adapted the agile development process to work just as well for hardware as it does for software. Of the various agile development models, we utilized the SCRUM model by running a series of "development sprints." Each sprint consisted of a goal/schedule stage, design and review stage, development and production stage and finally a testing and review stage. A condensed structure of a sprint is shown in Figure 2. In total, nine sprints were carried out over the development of Ground Squirrel, and the content of each sprint is displayed in Figure 3.

Figure 4. Picture of completed Ground Squirrel.

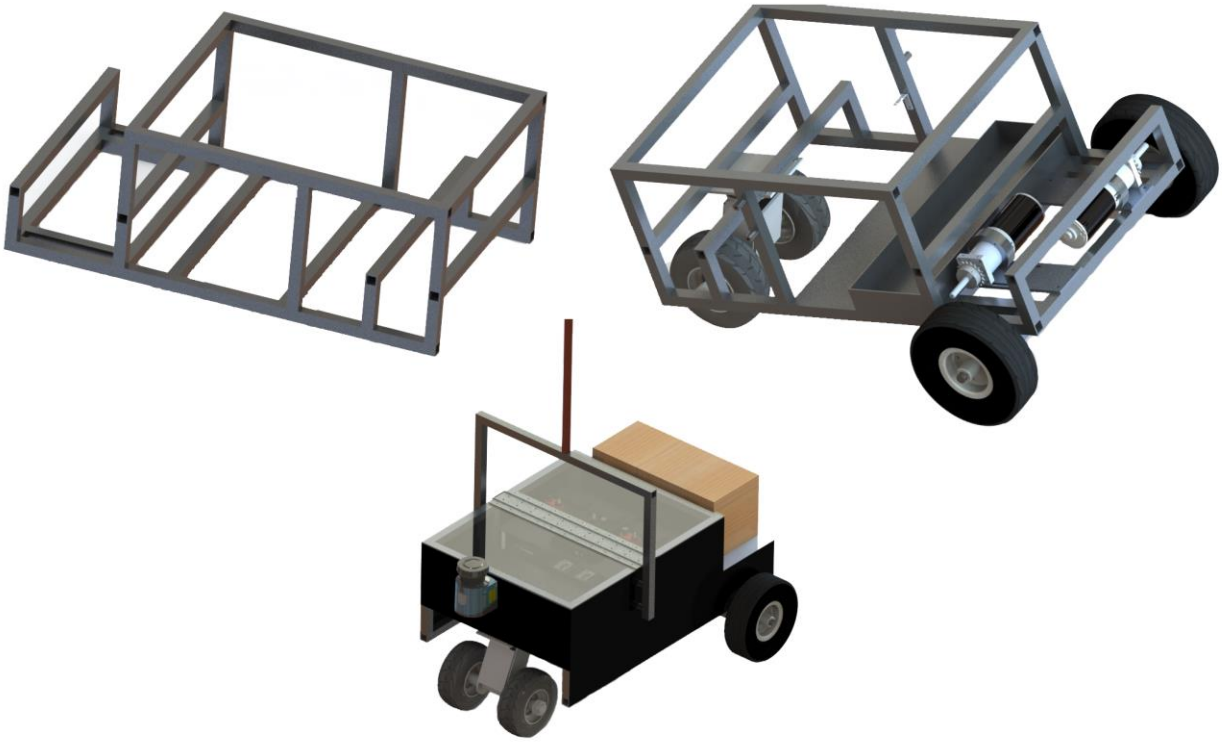

Figure 5. CAD models of the robot chassis, drive system, and fully assembled.

DRIVE AND CHASSIS

The chassis consists of two levels, an upper non-load bearing level and lower load bearing level. This leads to one of the innovations of Ground Squirrel:

- Upper chassis is completely reconfigurable and bears no structural significance (Chassis**Error! Not a valid bookmark self-reference.** section)

Chassis

The chassis of Ground Squirrel was designed for manufacturing simplicity, as well as reusability. Aluminum extrusion is used wherever possible due to its high strength-to-weight ratio. The chassis is composed of two pieces: the load-bearing base, and the nonstructural top. A ladder geometry is used for the base, allowing for easy construction while maintaining rigidity. All joints on the base are welded to maximize the chassis strength. The remaining frame members are all non-loadbearing, and are bolted on to form the frame.

The advantage of this design is that it is reconfigurable. Since the base layer includes all structural components, a new top layer can be bolted on for a different robot configuration. Further, this minimizes the amount of welding required, reducing the cost of manufacturing.

Drive

A differential drive was chosen due to its simplicity. The two drive wheels each are powered by a brushed DC motor. Turning is accomplished by driving one wheel faster than the other, allowing for a simple control algorithm.

The third wheel is a dual wheel industrial caster. Pneumatic tires are used for all three wheels to allow Ground Squirrel to traverse rough terrain. Due to the inertia of the caster, the robot drives in a leading caster configuration to prevent fishtailing.

To determine the ideal drive configurations, an empirical linear model of the system was derived. This model aided in control development, as discussed in Path Execution and Motor Control.

Weather Resistance

To keep everything dry the robot body is fully enclosed. Thin ABS sheeting is used for the skins due to its low weight and low cost. Weather stripping is used between all connections to keep the enclosure watertight. The side access panels are attached with Velcro, and the top panel is hinged to provide for easy access. All other panels are bolted on.

ELECTRONICS, CONTROLS, AND SENSORS

Ground Squirrel consists of three layers of electronics: data communications, sensors, and power distribution. Each subsystem is broken down further in the following subsections. Some key points of innovation include:

- Modular control board and connectors for easy removal from robot and for use outside of robot (Control Board Structure section)
- Multi-level control board system for wire runs and isolating subsections (Control Board Structure section)

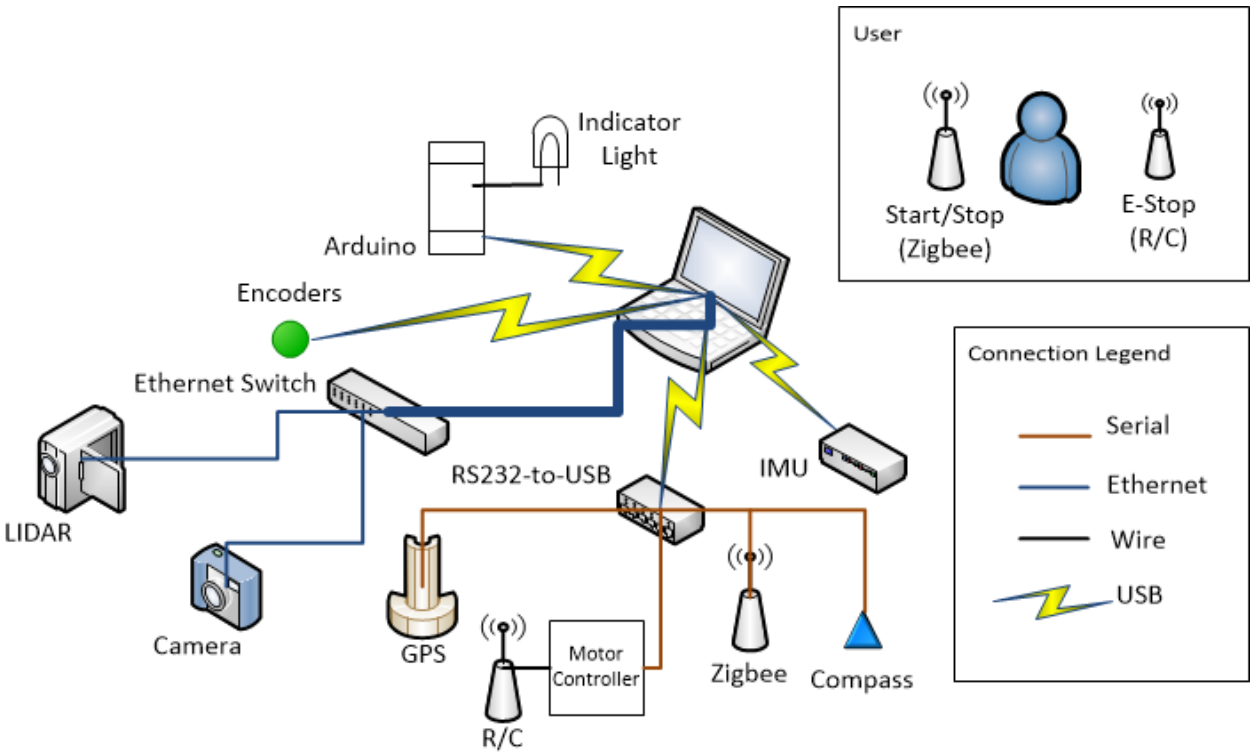

Figure 6. Data, sensor, and controls interconnections.

Controls and Interconnections

The brains of the robot consist of a Lenovo IdeaPad Y580 laptop running an Intel i7-3630GM at 2.4GHz, an NVIDIA Geforce GTX 660M GPU and Ubuntu 13.04 LTS. It has been upgraded with 16GB of DDR3 RAM and a 128GB Samsung SSD. The main decision for using this particular laptop was the price point and the ability to parallelize our algorithms and code on the GPU. The choice of upgrading RAM was so that we gain a large amount of caching space in faster volatile memory. The upgrade to an SSD was to have a solid-state system that is capable of handling large shocks on rough terrain.

For interfacing to the laptop, a combination of Ethernet, USB and RS232 is used. A Cisco 200 series gigabit Ethernet switch is used to connect multiple Ethernet-enabled devices together. Several USB to TTL level serial and RS232 adapters are used to breakout USB ports into serial ports for direct communication with most sensors on the robot. USB is then used to connect the devices capable of USB communication such as Arduino.

An Arduino Mega2560 is used to breakout to lower level sensors and indicators that the laptop is unable to interface with directly. The Arduino controls the indicator light, on board decorative LEDs, sensing the hardware E-stop, and controlling the cooling fans

All data and control signals are ultimately collected, processed, and generated on the laptop. The control signals are then distributed to the various other parts of the system. Figure 6 shows the controls and communications layout.

Motor Control

Ground Squirrel uses a Dimension Engineering Sabertooth 2x60 motor controller. This motor controller provides many different features and contains two channels capable of running two separate motors at 60A on each channel with a peak voltage of 120A at 30V. It is internally fused via a thermal breaker and

diode protected for overcurrent and reverse current protection. A 5V supply is also present on the controller. More about how this is utilized is described in Safety Features section below.

This motor controller allows for several forms of control including PWM, analog voltage, R/C and packetized serial data. We are utilizing the packetized serial control, which allows for the most customization and direct communication from the laptop. This method of control enables the Sabertooth to use a built-in E-stop as described in Safety Features section below. The Sabertooth has built in regenerative drive as well, meaning that when the motors are commanded to stop or slow down, the Sabertooth will instead redirect current stored in the motors back into the battery, thus saving power and extending the runtime of our batteries. Other features of the motor controller include built in speed ramping, differential turning and stopping.

As an add-on to the motor controller, we are using a Dimension Engineering Kangaroo x2. This small device plugs directly into the Sabertooth motor controller and provides a dedicated device to run a PID loop for controlling the wheel position or velocity. The PID loop uses feedback from quadrature encoders to control the wheel position or velocity. Communication to the Kangaroo x2 is via packetized serial. More on how the Kangaroo x2 works and fits in with our system is described in the Path Execution and Motor Control section.

Safety Features

Safety is a main priority in the construction and functioning of Ground Squirrel. The Sabertooth motor controller when in packetized serial mode allows for a built-in E-stop. By simply grounding the second signal input (S2) on the motor controller, the motors will come to a complete stop. Two forms of locking E-stops are present on the robot. The first is the on-board E-stop which, when pressed, grounds S2 bringing the robot to a full stop in hardware. The second E-stop is the wireless E-stop that connects to the robot via an R/C radio. By simply having the wireless E-stop turned on and pressing the button, a relay is flipped on the robot and S2 is grounded. If the wireless E-stop is not on or loses signal, S2 will be grounded by default. While the stop is in hardware, the software will also pick-up on the changed state and go into shutdown mode, stopping any further program execution.

Ground Squirrel's software also contains a watchdog timer. If it goes for some time and does not receive a signal from the start/stop switch or teleoperated control, depending on the mode of operation, it will go into a stand-by mode until communication is reestablished or the robot is shutdown.

The entire robot power system is also protected by a 150A thermal circuit breaker, which is accessible from the outside and fully weatherproof. More about this is described in Power Distribution. All electronics are also insulated from the chassis and each other, or they are properly grounded.

Sensors

Most of Ground Squirrel's sensors are located on the exterior of the robot or separate from the control board and are connected to the control board via a single connector. This allows for easy disconnection and removal of the control board from the robot. The most crucial sensors are described in more detail below.

GPS. The robot uses an Ag Leader GPS 1500 integrated antenna/receiver WAAS DGPS. This GPS provides sub-meter accuracy at an affordable price point. The data is output at 10Hz and is formatted in the NMEA standard for positional data over a serial connection. Along with positional data, the GPS outputs a simulated radar speed-reading. The unit is fully weatherproofed and easily mountable to any flat surface.

Compass. Heading data is acquired from a Digital Yacht HSC100 Fluxgate Compass. The heading is output at 10Hz via a serial connection and is formatted in the NMEA standard for heading data. The sensor is capable of 0.5° resolution and can output accurate headings with up to 45° of tilt on two axes. The compass also includes built in compensation and calibration for handling magnetic interference.

LIDAR. The primary obstacle detection used by Ground Squirrel is a SICK LMS100-10000 Laser Range Finder. The LMS100 provides 270° degrees of measurement with 0.25° of resolution, a range of 20m, and accuracy of ±30mm. Data is generated at 50Hz and can be queried from the device over serial or Ethernet.

Camera. Vision data for line detection and flag identification is gathered using an Axis M1013 Network Camera. This camera provides H.264 of MJPEG streams over an Ethernet connection for higher bandwidth. Images are streamed with a resolution of 800x600 at a rate of 30fps with a horizontal viewing angle of 67°. The camera has the ability for color, brightness, sharpness, contrast, white balance and backlight compensation built in. Multiple streams can also be broadcasted simultaneously from the camera.

Encoders. Attached directly to the output shafts on the drive wheels are two US Digital E4P, 360 count optical quadrature encoders. The encoders are used for distance tracking along with velocity estimations in conjunction with the accelerometer and gyroscope combination. A dedicated Phidgets quadrature encoder counter board is used to ensure all ticks and directions of the encoder are caught. The board interfaces to the laptop via USB and provides a high level set of commands to communicate with it.

IMU. The ArduIMU is a low cost, Arduino based inertial measurement unit (IMU) used as supplementary data for localization. The IMU attaches to a dedicated 3DR uBlox GPS and compass for localization measurements. The ArduIMU features an integrated Kalman filter which combines the GPS, compass, accelerometer, and gyroscopic data to give roll, pitch, yaw, latitude, longitude, and altitude.

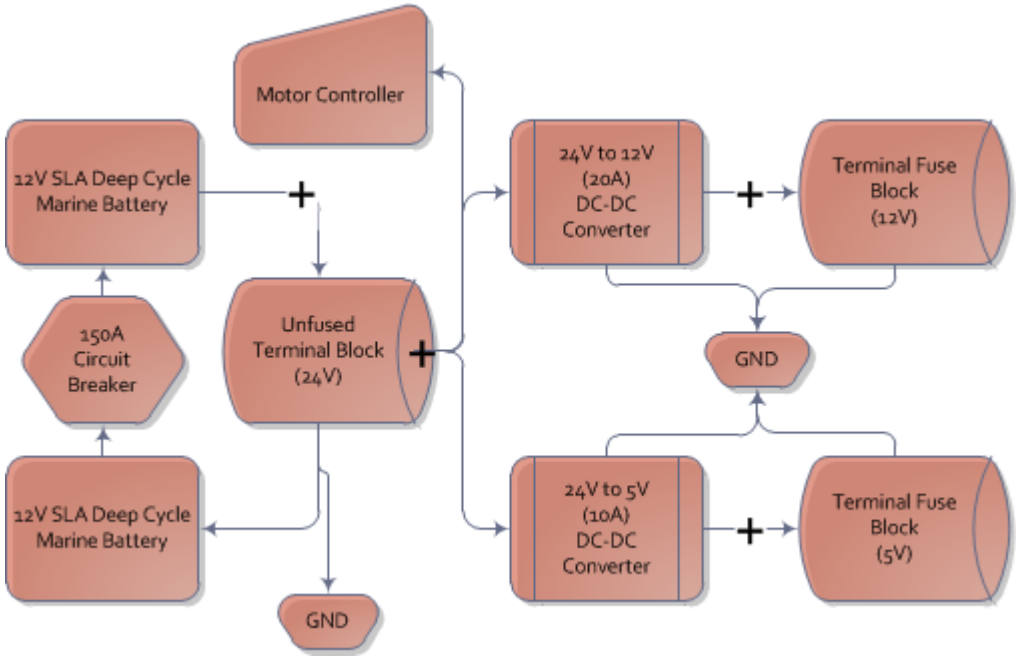

Figure 7. Power distribution block diagram.

Power Distribution

Ground Squirrel's power supply consists of three levels of voltages all supplied by two 150Ah deep cycle marine batteries connected in series to provide 24V. The 24V main line connects to a marine rated, unfused terminal block. From there the 24V line is stepped down to 12V and 5V via a 20A and 10A DC-DC step down converters respectively. The power from the 12V and 5V lines is then distributed via marine rated fuse blocks with ground planes. The fuse blocks accept standard ATC automotive fuses, which are available in sizes from 1A to 30A.

The entire power system is then protected via a 150A waterproof, marine rated thermal circuit breaker. The breaker will trip under a worst-case scenario motor stall where both motors are pulling 80A, and it will trip during a short. The protections built into the motor controller provide enough protection to protect against any transience that may occur in other situations.

In the power circuit, the breaker is positioned between the batteries to allow for charging of the batteries in parallel with the breaker is open or in series with the breaker closed, as shown in Figure 7. The battery charger used allows for the charging of the batteries in series and while they are in use, which enables for the robot to perform low power operations while the batteries are being charged. The battery also has a charging/testing port attached and a quick-disconnect connector for easy removal from the robot.

Figure 8. Control board structure.

Control Board Structure

The control board containing the laptop, power distribution, motor controller, Arduino's and serial hub is modular in that it is easily removed from the rest of the robot. All connections on the board that run to other parts of the robot pass through quick-disconnect connectors and a single connector connects all sensors that do not use a simple-to-remove connector (e.g. Ethernet, USB, and DB9). The board itself is an aluminum frame with an ABS plastic top, bottom and shelf. All the power distribution blocks are located on the bottom of the board leaving the top for the laptop and other digital electronics. The layer between the top and bottom is for routing wires. This arrangement provides a less cluttered area onto which the electronics are mounted.

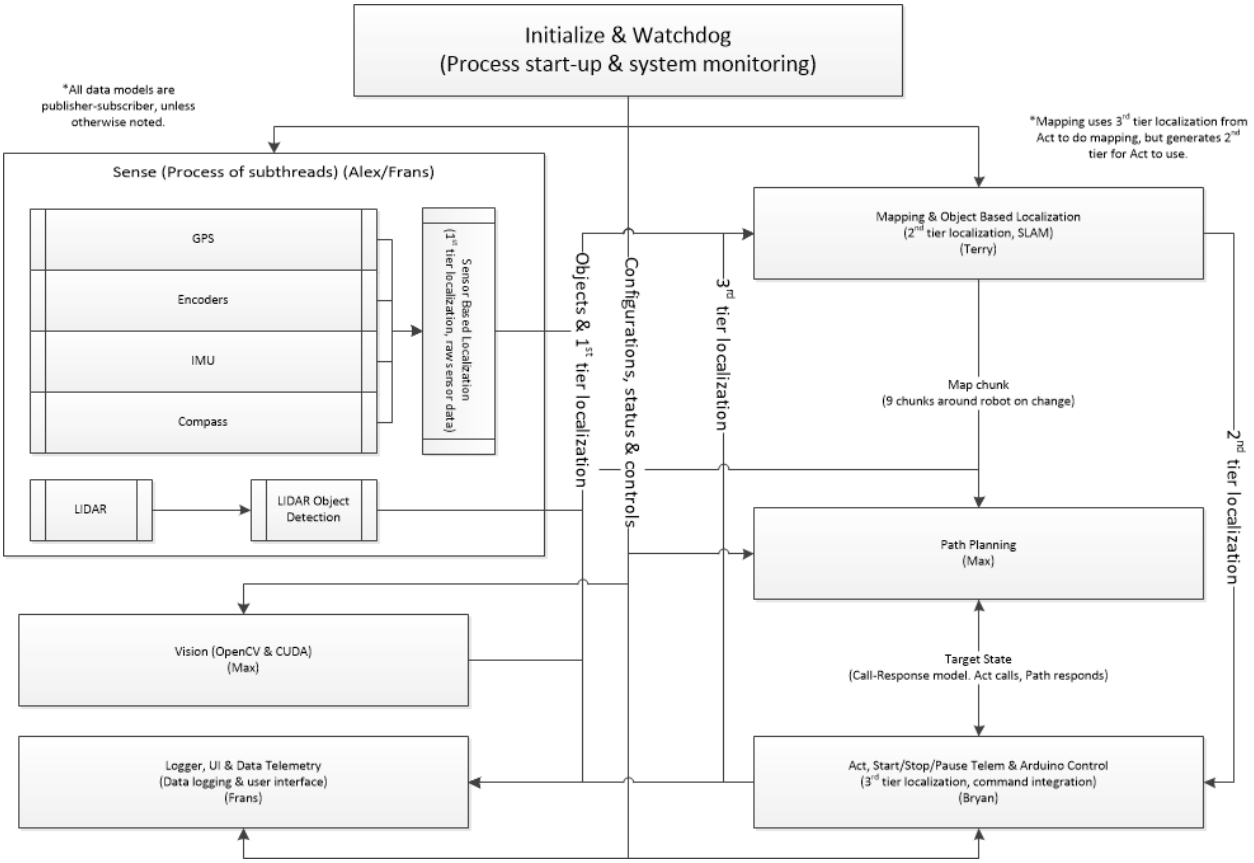

Figure 9. Software process layout and interconnections.

SOFTWARE HIERARCHY AND CONTROL

The software for Ground Squirrel is written in a combination of C++ and Python. The code contains mostly custom classes with custom and pre-defined libraries. The entire software hierarchy is based around distributed networking and computing. Each major component is split into a separate process, as shown in Figure 9. Each process then breaks its tasks down into classes and follows an object-orientated approach. Several shared components are built into libraries and provide API's for both C++ and Python. A master process controls the execution of sub-processes and regulates the system at a high level. This control is described in more detail in the next section.

Communications, Controls, and Interconnections

Communication between processes is handled using the 0MQ cross platform messaging library. The message queues provided by the 0MQ library communicate over TCP sockets and provide for publisher-subscriber, request-reply, and push-pull topologies. The ability to communicate over sockets provides Ground Squirrels software with the capability to be distributed across several systems in the future. Each process is given a configuration file location at start-up so that all processes know which port and addresses to communicate to.

All data is communicated using a publisher-subscriber topology. Each process runs asynchronously from all other processes, and can publish or read data at its own pace. Timestamps are provided in all data packets so the processes can determine if the data should be used or discarded because it is too old. Processes internally are split into sub-threads that each runs different tasks asynchronously.

A master process controls the start-up, execution, and shutdown of all sub-processes. The master process uses a request-reply topology to communicate commands and to send a heartbeat signal. This allows the master process to determine if a sub-processes needs to be killed and restarted if it does not respond.

Firmware (embedded systems)

Custom firmware was written for use on the Arduino Mega2560. This firmware provides a request-reply communication channel to query data from the Arduino's status and attached sensors and to issue control commands for controlling systems attached to the Arduino.

DATA PROCESSING AND MAPPING

Mapping

Ground Squirrel uses a sparse map to store obstacles' locations on the field in an absolute space. The map consists of "chunks" that are about 7 meters square. Each chunk stores all of the obstacles, lines, and flags found in that area. This allows a means to pass only relevant information to the path planning (i.e. an area of 3x3 chunks centered on the chunk containing the robot).

Adding obstacles to the map is simple for flags and barrels. These obstacles are simply put into the chunk in which their center is located. Lines are joined to any lines they overlap, and then are cut just outside of their chunk so that there is overlap with neighboring lines (which are the same line, just cut to fit into their own respective chunks). The overlap prevents the path planning from recognizing a gap as a valid path.

The map also finds the location of the robot based on how the objects' positions change between iterations. This Simultaneous Localization and Mapping (SLAM) algorithm locates a small number of the nearest objects, preferably barrels, and it compares their locations (relative to the robot) to the locations of the same number of objects that were selected in the previous iteration. By comparing locations, it can determine how far and in what direction the robot has moved.

Sensor Data Processing

Ground Squirrel has location data coming in from four different sensors; GPS, compass, encoders, and IMU. These need to be combined into one location to pass on to the map. To do this, the data is combined with a weighted sum. The weights are passed in from the configuration file so they can be adjusted easily.

Distances are derived from the GPS, encoders, and IMU. Since the distance between any two points on the field small relative the diameter of the Earth, the distance can be calculated as if it were a flat surface. The change in latitude and longitude between the current and previous iteration is found and that change is converted into meters. The change in the number of ticks of the encoders from the current iteration and the previous iteration is found and is converted to meters. The encoder and GPS distances are combined in a weighted sum to produce a new distance. This new distance is then converted back to a change in degrees of latitude and longitude and is added on to the previous iterations latitude and longitude to become the current position.

The current orientation of the robot is derived from the compass, GPS, and IMU data. The compass has zero as the value for directly north and it increases going clockwise. In order to compare it to the orientation derived from the GPS and IMU and to better conform to the standard axis, the compass data is transformed so that zero points directly east and increases going counterclockwise. From the change in latitude and longitude calculated above, an angle can be found by taking the inverse tangent of the latitude divided by the longitude. These two angles can be combined in the same way as the location data to produce the current robot orientation.

VISION AND LANE DETECTION

The robot uses a video camera and a computer vision algorithm to detect the locations of lines painted on the field. The algorithm to extract lines from the image uses a combination of thresholding and mathematical morphology approaches. In order to find the location of the lines in physical space, we precomputed a perspective transformation matrix that the robot uses to transform lines from image space to physical space.

Line Extraction

The vision algorithm begins with the raw image fed from the video camera. The most straightforward way to extract lines from this image is to threshold on luminance, that is, find all areas of the image whose luminance is above a certain value. Since the white lines contrast with the darker grass, the lighter areas of the image can be interpreted as lines.

There are, however, two types of objects with luminance's very similar to that of the lines, the sky and the white stripes on the barrels. It would be problematic for the mapping and navigation functions if these features were to be counted as lines, so they must be excluded. The sky can easily be excluded with a horizontal horizon-like cutoff. The white stripes on the barrels, however, are more challenging.

To exclude the barrel stripes, we decided additionally to find all barrels in the image. The areas corresponding to the barrels can then be excluded from the line finding. The barrels are detected using their distinctive orange color and shape; all areas of the image whose hue is within a certain range are counted as barrels. Of course, this procedure does not return a contiguous region for each barrel, as the white stripes split each barrel into discontiguous regions. These regions are connected by means of a morphological "close" operation (following an "open" operation to remove noise and other spuriously included pixels).

Finally, with all the excluded regions of the image well defined, the raw output from the luminance thresholding can be trimmed so it only contains the lines painted on the ground. A contour-tracing algorithm turns the final binary image into a series of closed contours describing the lines in image space.

This algorithm has one disadvantage in that it depends on many adjustable parameters. We developed a graphical interface that allows us interactively to tune the parameters to suit the current lighting conditions.

Camera Calibration

Once the lines are located in image space, their locations must still be found in physical space so that they can be used by the mapping and navigation functions. If we assume that all lines lie in the same plane (the ground plane), we can find a one-to-one mapping between pixels, or points in the image plane, and points in the physical ground plane. Finding the correct parameterization of this mapping is simply a matter of calibration.

Ideally, this mapping would be a simple perspective transformation. The camera's lens, however, introduces distortions, which we had to take into account. To this end, we employed the automatic distortion calibration facility provided by the OpenCV library. By capturing images of a flat chessboard-patterned surface held in front of the camera at various orientations, the software could infer the best distortion corrections to apply.

We then undertook a second calibration procedure to fit the parameters for the perspective transformation matrix itself. We placed a chessboard pattern or other distinctive marker on the ground in front of the camera, measured its distance to the point on the ground directly under the camera, and captured an

image of the marker with the distortion corrections applied. Once we had collected a reasonably large number of these data points, we used an OpenCV convenience function to find the best perspective mapping (in a least-squares sense, given our measurements) between the image plane and the physical ground plane in a coordinate system relative to the robot.

With these mappings known, the robot can then transform the locations of lines found by the extraction algorithm into physical robot-relative coordinates: It applies the distortion correction and then the perspective transformation to each point on an image-space contour to obtain the corresponding contour in physical space. The physical-space contour is then passed on to the mapping subsystem.

Other Applications: Flag Detection

A variant of the above hue thresholding algorithm could also be used to detect colored flags. However, as the flags are not situated on the physical ground plane, their physical locations cannot be determined using the same transformation employed for the lines. One option is to use only the horizontal location of a flag in an image to determine in which direction the flag is located. If distance information is needed, it could be inferred from the apparent size of the flag and a $\frac{1}{R}$ scaling law.

Performance Considerations

We expect our implementation of the algorithm described here to be relatively efficient, as we made use of optimized OpenCV methods for the most complex operations (most notably the morphology operations).

However, should our available computing power prove insufficient to run the vision algorithm alongside the other computationally expensive components of our software (e.g. path planning and mapping), there are several ways we could reduce the computational cost of the vision component.

The first and most obvious way is to reduce the framerate of the camera so fewer images are processed in the same timespan. Ideally, we would set the framerate to the lowest possible value that still allows the robot to detect significant course changes in time to respond to them. Another, less desirable option is to reduce the size of the individual images being processed. The reduction in cost that this option offers is accompanied by a loss in resolution and accuracy of the returned line locations. Because of this drawback, we intend to use this option only as a last resort.

One final option we have not yet explored is to exploit the processing capability of the control laptop's graphics processing unit (GPU). However, morphology operations do not yet appear to be implemented for the GPU in the OpenCV library. We have not investigated whether the operations could be accomplished using the other GPU-accelerated operations in OpenCV.

PATH PLANNING, NAVIGATION, AND ROBOT CONTROL

Path Planning

The purpose of the path planning module of the robot software is to use the map of obstacles in the robot's vicinity (generated as described in Mapping), along with a "target point" such as a GPS waypoint, to approximate the optimal path to that target.

The path planning module works in robot-relative coordinates. It constrains its search to the local bounding box given by the mapping module, since no obstacles are defined outside that box. To ensure the path is always up to date with the latest obstacle information, the path needs to be recomputed on every "significant" map update. The definition of "significant" is left to the mapping module; the path module simply recomputes its path every time it receives a new map from the mapping module.

The first step in finding a path is to turn the obstacle map into a graph describing the allowed locations of the robot and the ways it can go between those locations. First, the algorithm requires a description of configuration space, which defines the positions of the robot that are allowed and do not bring it in contact with any obstacles or lines. For this purpose, we assigned the robot a radius, which is the largest distance of any part of the robot from the center of the robot's coordinate system. To generate a map of configuration space, the path-planning program expands all obstacles by the robot's radius. For barrel obstacles, it adds the robot radius on to the obstacle radius. For line obstacles, it computes a path that is offset from the line by a distance equal to the robot radius. Finally, it computes the configuration space as the region within the local bounding box that is not occupied by any expanded obstacles.

To compute the graph describing the configuration space, we decided to apply the probabilistic roadmap algorithm (PRM) because it is relatively efficient and generates good coverage of the configuration space compared to grid-based methods. It proceeds by picking a random point somewhere in the allowed region of configuration space and testing whether it can be linked to each of its *k* nearest neighbors with a straight line that only goes through the allowed region. The new point is connected by a graph edge to each neighbor that is accessible in this way. The procedure continues until a sufficient density of points is reached. At some point in the procedure (at the end, in our case) the start point is added and connected with its neighbors.

With a graph defined, it is possible to apply a common graph search algorithm such as A* to find the shortest path from the start point (the robot) to any other node in the graph. It is necessary, however, to select a goal node so that the robot knows which path to take. If the target point is inside the local bounding box, it is added to the graph and connected as described above. If it happens to lie outside of the box, however, the selection of a goal node becomes more difficult.

Our current approach is to take the closest node to the target point to be the goal node. This approach could lead to the robot backtracking through the course at certain points. To prevent this behavior, we decided additionally to use information about the robot's direction, averaged over some large length (e.g. 15m) of the robot's previous path. If the target point is in a sufficiently different direction from the robot's average displacement, a different goal node is chosen in a direction interpolated between the target direction and the robot's average direction.

With a path to the target generated, the path module must send the control module information that the control module will use to generate drive commands that will make the robot follow the path as closely as possible. The current approach used by the control module is a position-based PID algorithm. Under this approach, the path module selects discrete points along the path and communicates them to the control module. The control module then generates appropriate drive commands to move the robot through this sequence of points.

The path module selects this point by discretizing the path at a certain length interval. The selection of this interval is governed by a number of criteria such as the desired accuracy of the robot's path following, computational cost, and desired smoothness of the robot's motion. It is currently implemented as a tunable parameter, although future implementations may be able to discretize adaptively. A fixed number of the discretized points are then communicated to the control module. This procedure is repeated on every path update.

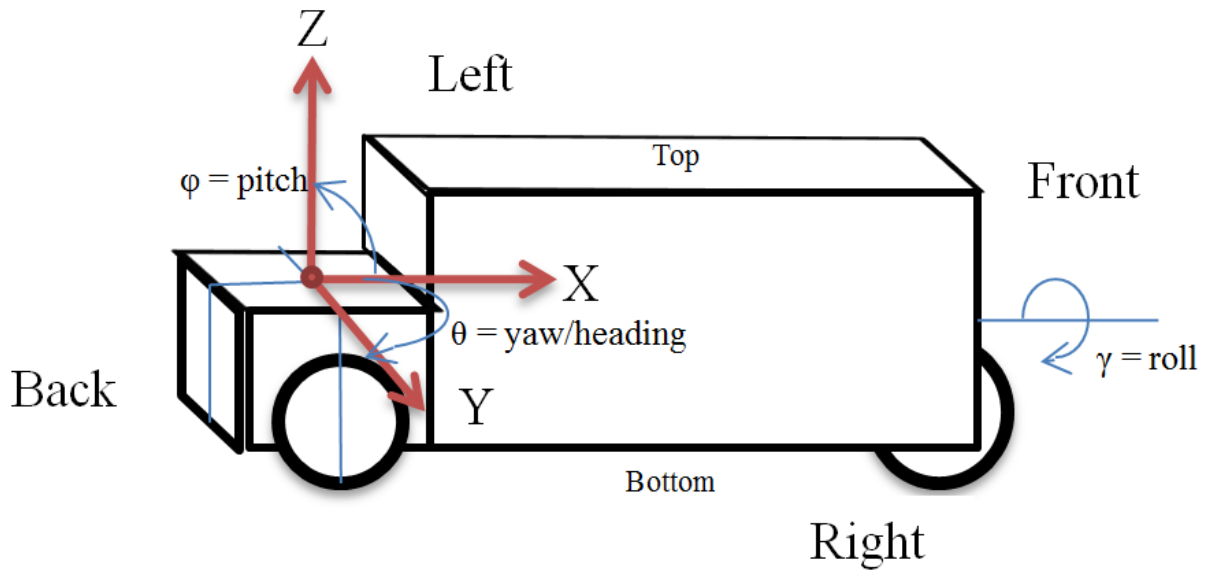

Figure 10. Robot coordinate system.

Path Execution and Motor Control

A key point of innovation in the area of controls for Ground Squirrel includes:

- Dual layered PID on relative and global position, with one level running a dedicated micro-processor.

To understand how to control Ground Squirrel, tests were run and an empirical model of the robot was constructed. This model uses the coordinate system seen in Figure 11. This model related the command given to the motor controller, the input, to the velocity of the robot, the output. The motor controller takes commands as a direction, power between 0 and 128, and a motor channel to control. Equation (1) below is the equation that describes the velocity of the robot, where *S* is the command value given to the motor controller.

$$V = \begin{cases} \frac{0.2333}{8} S - 0.3966 & 16 < |S| \leq 128 \\ 0 & |S| \leq 16 \end{cases} \quad (1)$$

Using simple equations that represent differential drive motions, the velocity of each wheel given by Equation (1) could be mapped to robot position and orientation.[1]

$$x = x_0 + \cos(\theta_0) \int_{\tau_1}^{\tau_2} V dt \quad (2)$$

$$y = y_0 + \sin(\theta_0) \int_{\tau_1}^{\tau_2} V dt \quad (3)$$

$$\theta = \frac{(V_R - V_L)t}{b} + \theta_0 \quad (4)$$

$$x = x_0 + \frac{b(V_R + V_L)}{2(V_R - V_L)} \left(\sin\left(\frac{(V_R - V_L)t}{b} + \theta_0\right) - \sin(\theta_0) \right) \quad (5)$$

$$y = y_0 + \frac{b(V_R + V_L)}{2(V_R - V_L)} \left(\cos\left(\frac{(V_R - V_L)t}{b} + \theta_0\right) - \cos(\theta_0) \right) \quad (6)$$

Equation (2) and Equation (3) describe the relative *x* and *y* position of the robot when moving in a straight line respectively. Equation (4) describes the relative orientation of the robot during a turn. Equation (5) and Equation (6) describe the relative *x* and *y* position of the robot while the robot is also turning. The *b* in Equations (4), (5), and (6) is the wheel base length. Together these equations make a first order model of the robot that describes the kinematics but disregards the dynamics. It also assumes a relatively flat surface or PID velocity control and that the robot does not slip. To account for these inaccuracies and to correct for outside disturbances, a more complicated system is implemented to control the robot.

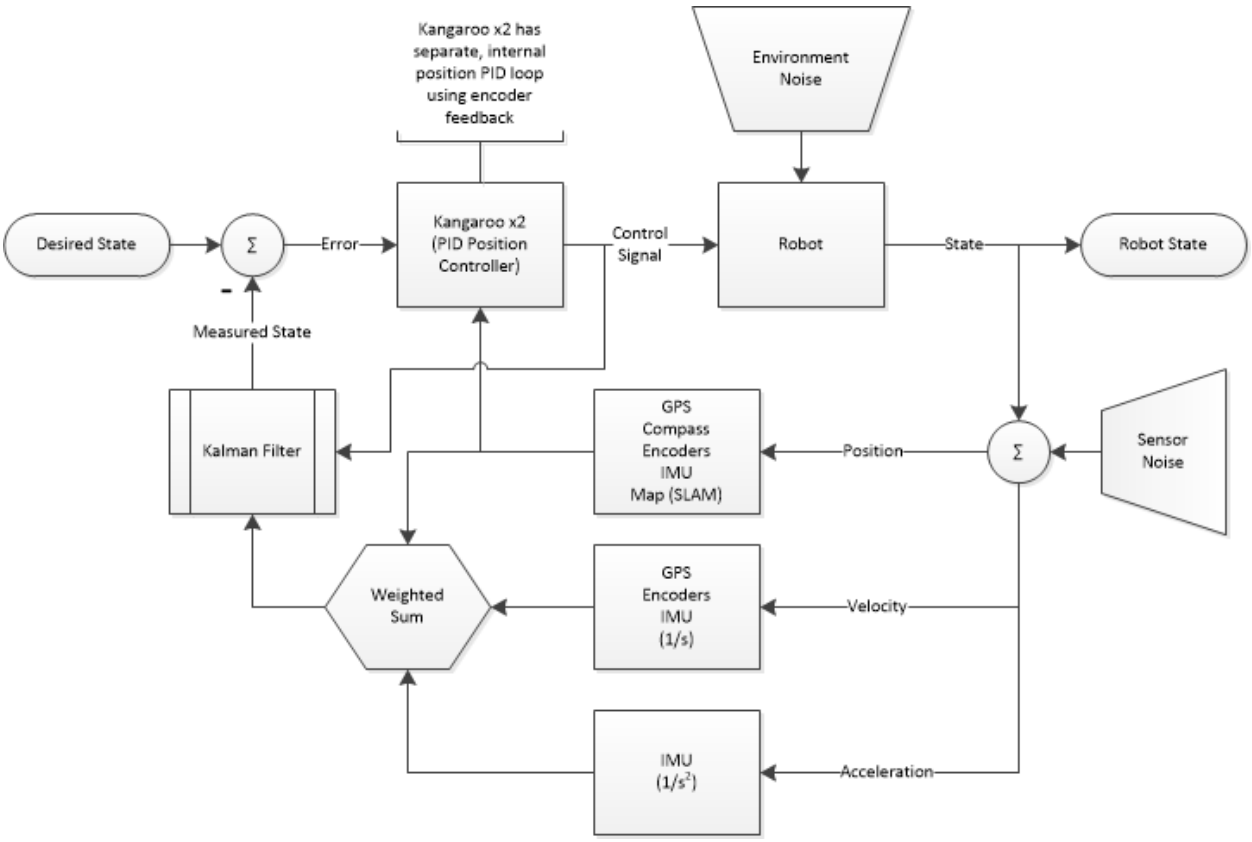

Figure 11. System control loop.

The motor control process receives the path as a series of waypoints, relative to the course starting point. Each of these waypoints is close enough to be broken up into a simple delta changes, consisting of an orientation change and a small linear movement. The commands are then generated to achieve these movements. A level of abstraction between the raw commands for the motor controller and the action the motor takes was implemented via the Kangaroo x2. The Kangaroo x2 takes the quadrature encoders as feedback input and the amount and direction to move or turn the differential drive robot. It then uses a built-in PID loop to move to the given relative position or orientation. This gives the advantage of having a simple interface to control the motors with in addition to a dedicated processor for running a low latency PID loop.

The position that the Kangaroo x2 controls is relative to the robot's current position. A secondary PID loop is run on the laptop to correct and control the robot's global position relative to the field. This PID controller uses the localization from the sensors and the map as feedback to correct for the error from the set point given by the path planning. The secondary PID loop also helps to correct for the dynamics of the caster not directly accounted for by the system model. The dynamics of the caster tend to cause the robot

to have a correction period during large angle, tight turns, where to begin moving in a forward direction again, the robot will veer slightly off course as the caster spins around.

Since control signals are being generated by the motor controller process, a Kalman filter can be implemented to combine and filter the localizations from the sensors and map. This localization is the most accurate of the three generated localizations. This data is fed back to the sense and map process where it is used to correct for error and map the placement of objects respectively.

PERFORMANCE ANALYSIS

Speed

Ground Squirrel has a top speed of 8mph on level ground, and it takes 10.2 seconds to reach this speed from standstill.

Ramp Climbing Ability

Due to Ground Squirrel's high torque motors and low weight, the wheels will slip before they stall on inclines. Assuming a coefficient of friction of 0.8, we calculate the max incline that can be traversed from a standstill to be 35°.

Reaction Times

The limiting factor in reacting to new environments is the path planning. From this, we predict that at map resolutions of ½ meter and ¼ meter we will see performances of

- Predicted Performance: 0.35 Seconds (1/2 meter)
- Predicted Performance: 3.33 Seconds (1/4 meter)

Times predicted are based on running the path-finding algorithm on a simple generated test map. No optimization or parallelization has been implemented at this time.

Battery Life

With the deep cycle marine batteries, the calculated battery life of the robot is a nominal two hours at full power. After testing, this performance spec appears to match the calculated time. The time to reach full charge is about twelve hours.

Range of Obstacle Detection

Using the LIDAR for object detection gives a max distance of detection of 19.5 meters within our set resolution. Line detection is based solely on camera image data and is thus limited by the field of view of the camera and our region of interest, which in turn is dependent on the mount height and angle of the camera. At the time of this writing the FOV and ROI intersect was unknown.

Complex Obstacles

Complex obstacles such as switchbacks and dead ends should be handled naturally by the robot's persistent mapping and path finding systems. The persistence of the mapping combined with the nonlocality of the path finding algorithm ensures the robot does not backtrack or become stuck in any limit cycles.

As the robot approaches a switchback, for example, the LIDAR and camera constantly update the robot with the locations of newly visible obstacles (barrels and lines). The path finding algorithm ensures the robot will take the (approximate) shortest path to the next GPS waypoint given the current (and accu-

mulated past) information the robot has on the obstacles around it. At no point during a switchback would this algorithm lead the robot to attempt a spurious or disallowed path.

Dead ends are handled in a similar way. As soon as the robot detects that there is no way out of an area of the map (again, the map's retention of past obstacles helps here), the path-finding algorithm finds a route out of that area.

Accuracy of Arrival at Navigation Waypoints

The accuracy of arrival at GPS navigation waypoints with our current localization methodology is dependent on the accuracy of our GPS and our coordinates fixation at the beginning of the course. With our GPS, we can get an accuracy of up to one meter. With the use of the IMU, we may be able to get a better accuracy, but currently the accuracy of the IMU coordinates is unknown.

BUDGET AND TIME BREAKDOWN

We were limited this year to a budget of $4000, in addition to $9000 the previous year. An approximate, high-level breakdown of how this money was spent is outlined in Table 1 below. Over the course of two semesters, our team has placed in over 1500 hours of work and countless hours more in training. This estimate is based off five hours of work a day, two days a week over 30 weeks and accounts for absences and breaks.

Table 1. Cost of Ground Squirrel.

Item	MSRP ($)	Cost to Team ($)
Chassis Material	470	470
NPC Motors (x2)	200	200
Other Drive Components	455	455
General Hardware	100	100
Lenovo Y580 Laptop with Upgrades	1200	1200
Sabertooth Motor Controller	190	190
Laptop Interfaces	490	490
E-stop Button (x2)	150	150
Power Distribution and Batteries	650	650
Axis Ethernet Camera	200	0[1]
SICK LMS100 LIDAR	3900	0[2]
Digital Yacht Compass	280	280

[1] Previously owned by team.
[2] Donated by SICK.

Ag Leader 1500 GPS	995	845[3]
IMU and Supplementary GPS	170	170
Other Sensors	330	220[4]
General Electrical and Wiring	150	150
Total	**9930**	**5570**

The team consisted of six members, with varying academic interests and experience.

Table 2. Team members.

Name	Academic Department	Class
Bryan Stadick	Electrical Engineering	Senior
Max Veit	Computer Science and Physics	Senior
Adam Juelfs	Chemical Engineering	Sophomore
Alex Dahl	Computer Science	Sophomore
Terry Furey	Math	Sophomore
Frans Elliott	Electrical Engineering	Freshman

CONCLUSION

We believe that this year we have prepared a robot that is suited for tackling the challenge issued by IGVC. We have taken our experience from the previous year and grown from that. Ground Squirrel this year contains much improved and more sophisticated software for handling the task of autonomous navigation. The team as a whole has a better understanding of the problems faced and has learned much in the areas of robotics, computer vision, software development, AI, and controls in preparation for this year. We look forward to testing our robot out and learning further about where we can improve.

[3] University discount applied.
[4] Some parts donated by Banner Engineering

ACKNOWLEDGEMENTS

At this time, we would like to acknowledge Mr. Bruno Bittner at SICK for helping us acquire a LIDAR and Mr. David Anderson at Banner Engineering for his work in getting us the touch button in addition to several other sensors not used on this year's robot. We would like to thank the University of Minnesota for providing us with the funding we need, and Dr. William Durfee for supporting our group over the past several years.

REFERENCES

1. **Lucas, G.W.** A Tutorial and Elementary Trajectory Model for the Differential Steering System of Robot Wheel Actuators. *Rossum.* [Online] 2000. http://rossum.sourceforge.net/papers/DiffSteer/.