Olin College DRAPER LABORATORY SCOPE Senior Capstone Program in Engineering

Project: Draper 2010-11

Final Report

Prepared for:

Draper Laboratory

Sponsor Liaison:
Troy B. Jones

SCOPE Project Team:
Daniel Grieneisen
Nicholas Hobbs
Jacob Izraelevitz
Arash Ushani
Ann Wu

Faculty Advisor:
David Barrett

April 22, 2011

CONFIDENTIAL

Table of Contents

Table of Figures

Acknowledgements

The 2010-2011 Draper Laboratory SCOPE team would like to thank a number of people and companies for their support. First, we thank the Charles Stark Draper Laboratory for sponsoring us. We thank Troy Jones of Draper Lab for acting as the primary liaison between our team and Draper Laboratory, as well as Dan Strickland and George Sass, both of Draper Laboratory, for their help. We thank Professor David Barrett for serving as our faculty advisor and providing us with guidance in undertaking this project. In addition, we thank the MIT Lincoln Laboratory SCOPE Team, Prof. Siddhartan Govindassamy, and Garland O'Connell for critiquing and offering advice on our design decisions during our design reviews.

We would also like to acknowledge the 2009-2010 Draper Laboratory SCOPE team for the work that they did on this project, completing the mechanical and electrical design to make the vehicle drive-by-wire, as well as developing a software architecture with basic autonomy capabilities. Additionally, we specifically thank 2009-2010 team members Andy Barry and George Sass for quickly answering all of our questions vehicle software and hardware.

Finally, we would like to acknowledge suppliers who have provided equipment to the team at reduced or no cost. First, we thank National Instruments and Lesley Yu for both their hardware and software support, as well as for the technical support that they provide us. Second, we thank Advanced Motion Control for their hardware support of the team.

Executive Summary

Project Description

The 2010-2011 Draper Laboratory SCOPE team is working as part of a collaboration between Draper Laboratory and Olin College of Engineering to develop an autonomous John Deere Gator XUV. The overall goal of this project is to build on the work done by last year's team in order to create a robust, reliable, extendible unmanned ground vehicle (UGV) to be used as a test and research platform for future engineers at both Olin College and Draper. In order to accomplish this, we are ensuring the robustness of the vehicle hardware systems, adding sensors and software to develop reliable autonomy capabilities, and developing an extendible software architecture.

We have defined our deliverables for the project to be two different goal demonstrations: a base mission and a resupply mission. These demonstrations will require us to incrementally achieve greater software and hardware functionality. Our base demonstration is GPS waypoint navigation and avoidance of static obstacles. The resupply demonstration is a mission along a road with intermittent GPS loss.

Because this will be used as a test and research platform in the future, all of our design decisions have been made with the values of dependability, extendibility, and flexibility in mind. At the conclusion of this project, we are delivering a vehicle capable of reliable autonomous navigation and obstacle avoidance with a clean and easily extendible software architecture and a robust mechanical and electrical system.

Introduction

This final report encompasses a summary of all of the work done by 2010-2011 Draper Lab SCOPE team in improving and expanding the capabilities of the unmanned ground vehicle developed by the previous year's Draper SCOPE team.

Our goals were twofold. Firstly, we aimed to achieve certain vehicle capabilities. These are be demonstrated by various missions that the vehicle can accomplish. Secondly, we aimed to create an extendable system so that future teams working on this project can reuse most of the work that we have completed.

Mission

For this project, we have defined two goal demonstrations: a base mission and a resupply mission. Each of these demonstrations requires incremental development of more advanced software functionality. In the base mission, the vehicle drives to given waypoints while avoiding obstacles. In the resupply mission, the vehicle drives along a path or road from a supply depot to an outpost.

Base Mission

For our base mission demonstration, the vehicle autonomously drives on flat, low complexity terrain with static obstacles. The vehicle navigates to manually input GPS waypoints while avoiding obstacles. Alternatively, the user can teach the vehicle a mission by driving the path beforehand as the vehicle records a set of GPS waypoints. In order to achieve this, we upgraded the current state of the system, implemented a new software architecture, developed a set of behaviors to identify and avoid obstacles, and make some hardware changes.

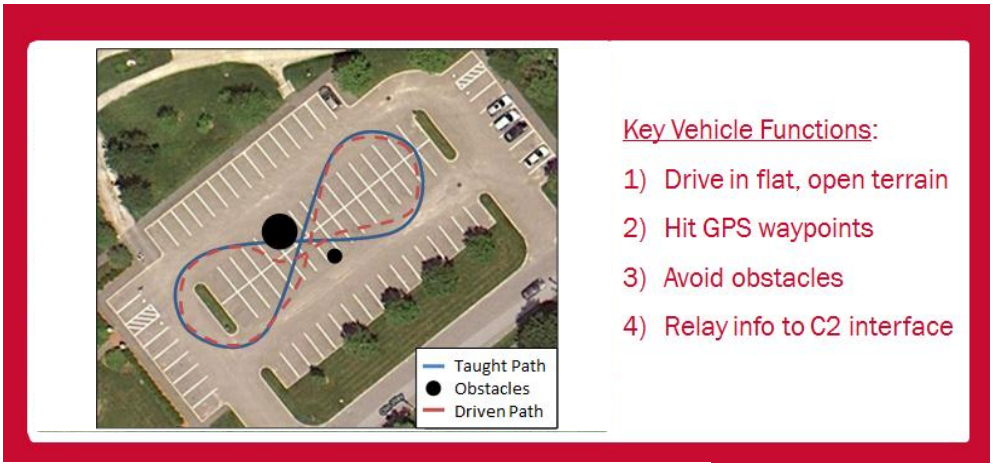


Figure 1: Base Demonstration

Figure 2: Resupply Mission Demonstration

Resupply Mission

For the resupply mission demonstration, the vehicle autonomously drives along a tree-lined access road and avoid obstacles along the way. In order to achieve this, we developed algorithms to navigate in a GPS-lossy environment, as the trees obscure GPS coverage over part of the road. The vehicle also is able to identify the path and remain on it with a minimal number of GPS waypoints.

Design Values

As we worked on this project, we adhered to the following design values:

1. Dependability – Because this vehicle will be used as a test platform, it needs to be highly dependable and well tested so that it does not fail in the field. Additionally, because the Gator's capabilities are going to be extended by future teams, the hardware and software that we develop need to work reliably so that teams do not need to redo work that we have done this year.
2. Extendibility – This vehicle is going to be used by future teams at both Olin College and Draper Lab for a variety of missions which we do not yet anticipate. The software and hardware architectures need to be designed so that they can be easily extended in the future to meet these unanticipated missions.
3. Flexibility – The vehicle needs to be flexible enough to be used for a wide variety of missions and tests by Olin and Draper in the future without major modification.

Initial Vehicle State

Last year's Draper Lab SCOPE team did a significant amount of work to make the vehicle drive-by-wire and give it basic autonomy capabilities. For a full description of what they accomplished, please see their final report. One of our first tasks was to verify the capabilities of the vehicle in its current state, and to try to ensure that there were no major hidden problems within any of the major subsystems that could appear later and jeopardize our project. Here, briefly, was our assessment of the state of different vehicle subsystems at the beginning of the project.

Mechanical System

Most of the previous mechanical system appears to be robust and effective. The gator has a pair of linear actuators which engage the brake and gas pedals. There is a modified power-assistive steering mechanism used to drive the steering column and turn the front wheels. This entire system is designed such that if power is removed from the actuators, the actuators have low holding force, so gator can still be driven normally. This is invaluable in getting to and setting up field testing.

There were two main mechanical failure points that we discovered in assessing the vehicle. The first was the mounting system for the pegboard connected to the brushguard on the front of the vehicle. Because this mount was slightly underconstrained, the pegboard could roll forwards and backwards by 5-10 degrees. This error would become significant once we begin to mount sensors such as scanning LIDARs on this board. The other main mechanical failure point was in the steering endstop mounts. In order to sense the limit of the steering without hitting the mechanical stops, there are two magnets mounted to the steering tie rod which are measured by two Hall Effect sensors mounted to the vehicle chassis. These magnets were poorly mounted, causing them to slide into positions that could not be read by the sensors after repeated outdoor testing.

Both of these problems and our implemented solutions will be discussed in more detail in the "Mechanical Modifications" section of this report.

Electrical System

The electrical system of the gator at the beginning of the semester was well organized and effective, with one noticeable exception (addressed below). There is a Honda 2000W generator which provides 120V AC power to the entire system. This allows the vehicle to be used for extended field testing. Inside of a waterproof electronics box, the AC power is divided into three different power rails by DC regulators: a 12 V rail, a 24 V rail, and a 48 V rail. The 12 V rail powers a Microstrain IMU, a number of fans, the emergency stop system, and some Ethernet hardware. The 24 V rail powers a LIDAR, a NAVCOM GPS, the linear actuators for the gas and brake pedals, and the two on-board computers (a Sea Level PC and an NI CompactRIO) through a UPS. The UPS ensures that the computers remain powered when switching the AC power source (for example, from the Honda generator to the wall power in the lab).

The 48V DC power rail was used solely for powering the steering motor. Unfortunately, there were two problems with this. The steering motor, custom ordered by last year's team from Potomac Electric, is rated to be run at 170V DC. However, it was being powered by 48V DC using a Victor motor controller. The first problem was speed. The motor control scheme from last did not turn the steering quickly enough for the vehicle to reliably follow waypoints using the Dubin's Curve algorithm implemented last year. The second problem was shutoff. If, when the motor is powered, the driver backdrives the steering wheel, it can cause a significant voltage spike (10-20V above the 48V being supplied to the motor). When this happens, the voltage spike propagates back through the Victor motor controller to the 48V power supply. The output protection on the 48V supply causes it to shutoff, killing

all power to the steering motor. This situation can also happen if the gator tries to turn its wheels while not moving or moving very slowly. The friction between the tires and the ground coupled with the elasticity of the tires causes the motor to be backdriven a small amount, but quickly enough to send a significant voltage spike. We fixed this problem by acquiring a new servo amplifier spec'd to run the steering motor at up to 170V DC.

Modifications that we made to the electrical system are described in detail in the Hardware section of this report.

Software System

The software of the vehicle is implemented in National Instruments' LabVIEW. We are currently building upon what was supplied to us by last year's team. For a more detailed description of their code, please see last year's final report.

Essentially, there are two main LabVIEW Vis that need to be run for the vehicle to behave autonomously. One runs on the Sealevel PC and the other runs on the FPGA. An interface on the computer allows the user to input a mission plan, such as specifying GPS coordinates by clicking on a map. The vehicle can then carry out the mission, or it can be put in several different modes. The user can directly control the gas, brake, and steering in LabVIEW. The user can also control the vehicle using the included Xbox controller. Additionally, the user can choose to run a steering test that will also recalibrate the position of the steering wheel.

Hardware

Hardware Additions

Innovative LIDAR Mounting Scheme

Our obstacle avoidance and path planning algorithms are highly LIDAR-dependent. As such, we developed two mounting schemes:

Base Demo Configuration

Two LIDARs are mounted on the front hood facing forward. One is mounted at a tilt of 8 degrees and one at 20 degrees. These two LIDARS therefore effectively detect obstacles scan at two different distance ranges, allowing the Gator to get an early alert on the existence of an obstacle while still being able to avoid it at close range. These LIDARs are mounted using the existing configurable mounts.

Figure 3: LIDAR Scan orientation for Base Demo functionality

Option1&2 Demo Configuration

1) Two LIDARs are mounted on the front hood at 45 degree angles from the forward axis. For the Option 1 and 2 demo functionality, we also oscillate these LIDARS about a horizontal axis perpendicular to vehicle motion, allowing us to obtain 3D information.

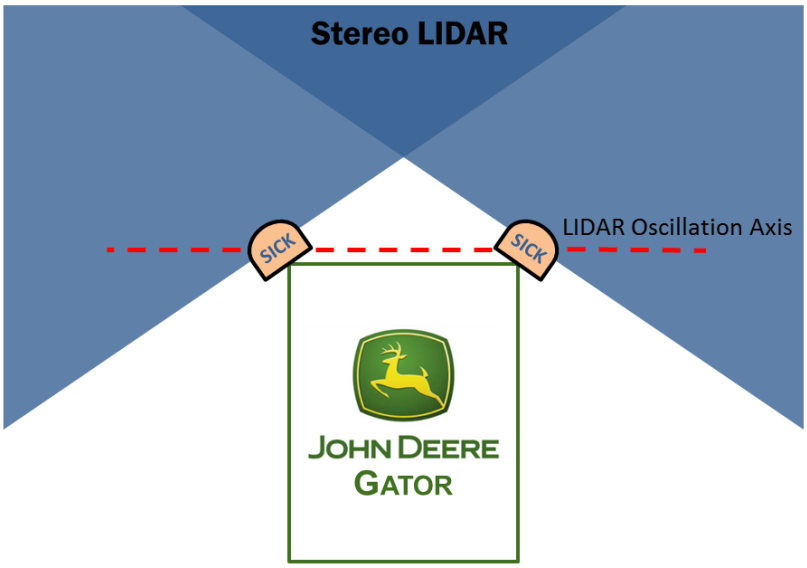


Figure 4: LIDAR Scan overlay, top view

This LIDAR scheme has the following attributes:

1) High Front Coverage: Obstacles directly in front of the vehicle need to be detected soonest because the vehicle approaches them the fastest. This obstacle area is covered by both LIDAR's. Since the two LIDAR's are oscillated out of phase, we are effectively doubling the obstacle detection data rate in front of the vehicle for the 3D point cloud.
2) Wide Side Coverage: Each LIDAR scan sweeps an arc to the side of the vehicle, even seeing obstacles near the cabin doors. This coverage allows us to detect obstacles during sharp turns, when there is a large angle between the vehicle's intended motion and its true yaw.
3) Low Ground Clearance: LIDAR's as a rule are mounted as low as possible on autonomous vehicles to avoid blind spots below the scan. Moving the pegboard lower onto the front bumper would therefore be even better, but could endanger the LIDAR's.
4) Small frontal blind spot: The LIDAR's are blind to obstacles closer than ½ the distance between the two LIDAR's. However, since that distance between the LIDAR's is roughly 3ft, this 1.5ft long blind spot is largely negligible. A typical obstacle in that blind spot could not be avoided by either a braking or a turning maneuver, but it would not be able to enter the spot without being sensed by both LIDARs.
5) Detection of LIDAR-black obstacles: If a LIDAR-black obstacle lies within the overlap region of the two LIDAR's, its location can still be triangulated from the LIDAR angular measurements, even though the range data is lost. The intersection of the two dead spots of the LIDAR scan represents the location of the LIDAR-black obstacle. This capability may not be useful outdoors, as we have yet to encounter a LIDAR-black obstacle outside the lab, so we have no plans to implement it next semester. However, it would be a good enhancement of the vehicle as an Olin research project.

LIDAR Oscillation Mechanism

Oscillating a LIDAR through an axis 45 degrees off a body axis is difficult to implement without interfering with the beam scan. Two designs were considered:

1) Mounting a shaft on the bottom of the LIDAR: This allows the shaft to be supported on both sides of the LIDAR without interfering with the beam. Unfortunately, the motor load is roughly 3 times more than rotating the LIDAR through its center of mass.
2) Mounting a shaft through the LIDAR center: This shaft may only be supported on one side, otherwise the shaft mounts intersect the beam. The shaft is therefore cantilevered off two bearings.

As high-load support members are cheaper than a more powerful motor and electrical system, we selected the second choice.

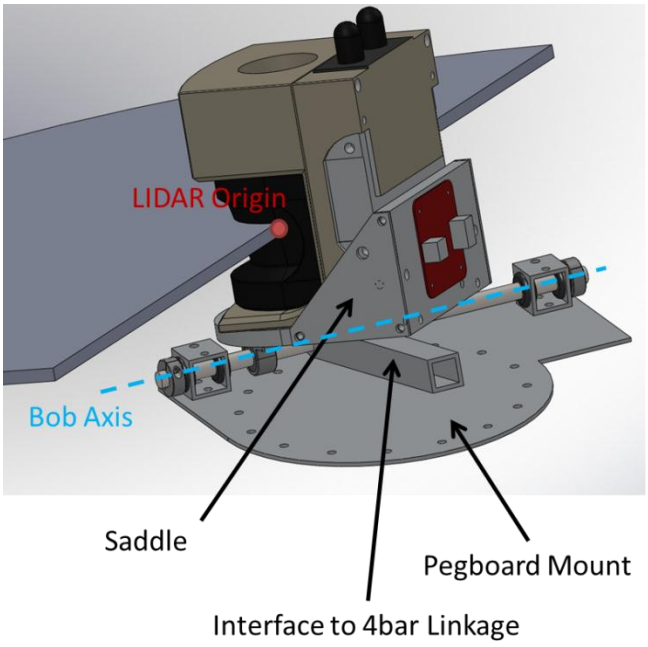


Figure 5: LIDAR Oscillation Mechanism (Design 1)

The LIDAR in Design 1 is oscillated about a shaft mounted on its bottom, supported by bearings on both sides. The LIDAR is actuated up and down by the square aluminum bar stock attached to mounting saddle. This design increases the moment of inertia of the LIDAR by a factor of about three, substantially affecting motor and electrical specifications.

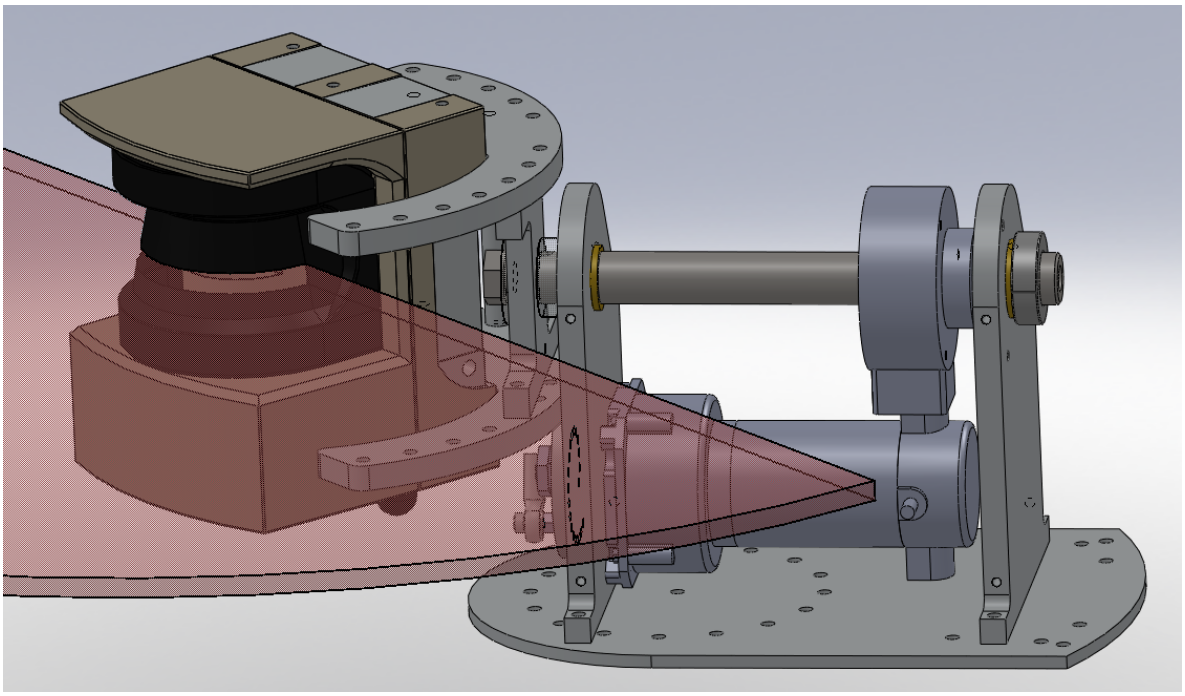


Figure 6- LIDAR Cantilever oscillation Mechanism (Design 2)

In Design 2, the LIDAR is cantilevered off two bearings, mounted to the front pegboard of the vehicle. This mechanism requires a heavier mounting structure to limit vibration than the previous design, but has lower motor loading.

We designed the system to be mountable at any angle in increments of 22.5 degrees to the pegboard, and mount the LIDAR at any 11.25 degree angle increment in its oscillating saddle. The saddle is oscillated with a 4-bar linkage mechanism off a continuously running motor. The supporting shaft does not transmit the oscillation torque, but only supports the weight of the LIDAR. To prevent backlash in the 4-bar linkage mechanism from affecting the LIDAR angle, the LIDAR angle is measured directly by a shaft encoder. These parts were manufactured professionally by the Macdiarmid machine shop.

The shaft encoder is a relative encoder, not an absolute. The zero point is therefore set off the index pulse, which is oriented with respect to the shaft by the encoder shaft clamp. On startup, the LIDAR should oscillate until it detects this index to zero the angle. If you need to take off the encoder and put it back on for some reason, this index location needs to be recalibrated in code. The LIDAR oscillation motor has no encoder, so the oscillation speed is determined from the LIDAR angle encoder.

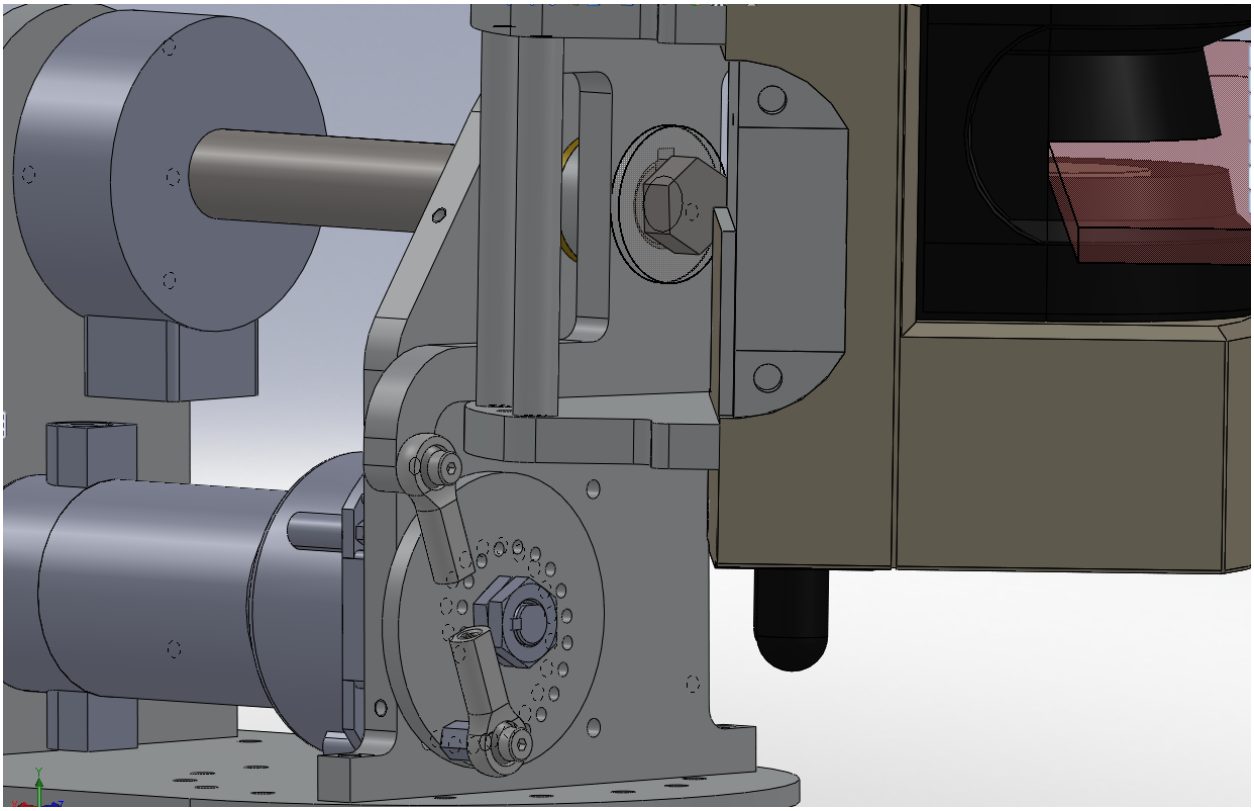


Figure 7: Close Up on LIDAR Actuation

The LIDAR is actuated by a 4-bar linkage mechanism, attached to a continuously running motor. The mean and amplitude of the oscillation can be tuned by two parameters respectively: the length of the threaded rod linkage and the chosen hole in the spinning disc on the motor shaft.

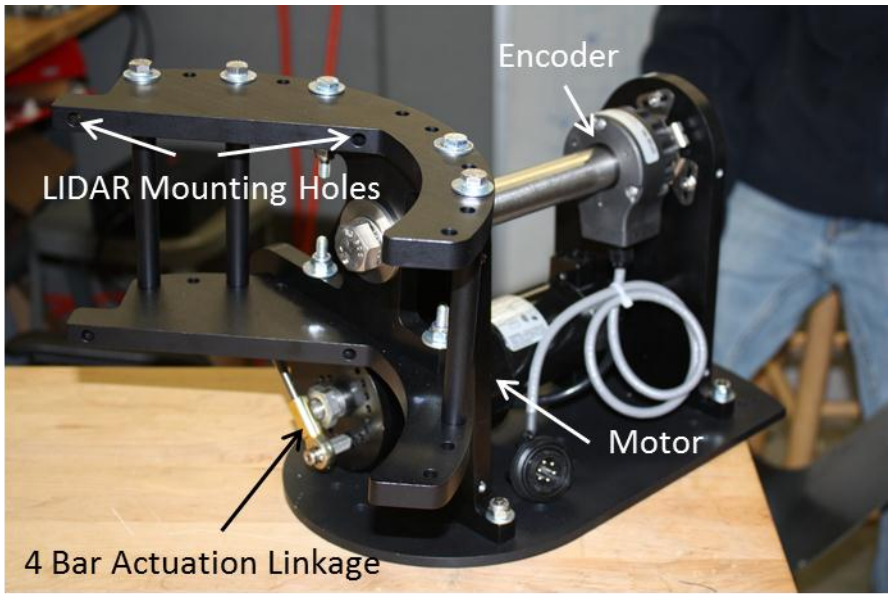


Figure 8: Completed Oscillation Assembly

Dell R5400 Computer

Figure 9: Dell Precision R5400

The SeaLevel Relio S5220 was replaced by a Dell Precision R5400 rack workstation. It has eight cores, as opposed to the two cores in the existing workstation. This is an appropriate number because each complex algorithm running on the workstation will consume approximately two cores, and the base goal of this project requires two to three of these algorithms at any given time. In addition, the separate cores of the machine encourage the use of parallel processing. Using software from National Instruments, this workstation can be converted to a dedicated LabView Real-Time platform.

While the SeaLevel workstation existed on a 24V power supply with UPS backup, this new workstation runs directly off the AC power provided by the generator. To accommodate this, a separate AC UPS, the Back-UPS ES 550 is used to provide brief support during outages, or when switching over to a different power line. The FPGA is integrated with the workstation's power supply, as detailed in the next section.

FPGA

Figure 10: NI PCIe 7852-R

The FPGA has been replaced by an NI PCIe -7852R, which features a Virtex 5 LX50 FPGA.

Figure 11: NI 9151

The drawback of using this is that it does not contain expansion slots for C-series I/O modules. To compensate for this, two NI9151 R-series expansion chassis are included; each as 4 expansion slots and a third chassis may be added for additional slots if desired.

Steering Motor Controller

Figure 12: AMC 30A20AC

The steering controller in the preexisting system is a 48V Victor HV motor controller. There have been some performance issues associated with this motor controller because it is a poor match to the steering motor, which was designed to handle much higher voltage (110V). An issue arose where back-driving the steering motor would cause a voltage spike which the Victor could not handle, and the solution to this is a motor controller that is designed for higher-voltage motors. The upgraded part is the AMC 30A20AC, which can output up to 170VDC at 30A. We have tested this controller manually, and it successfully solves the power supply shutoff problem.

The AMC servo amplifier should be able to handle any back-driven voltage from the motor because it is designed to output a control signal at a higher voltage than the nominal voltage steering motor should operate at.

Switching Interference

The switching frequency of the servo amplifier will create 22kHz interference, causing undesired crosstalk with several encoder inputs in the rest of the system, if it is not properly compensated for. The solution currently being used is a combination of shielding the output lines of the amplifier, shielding the input lines of the encoder, as well as using a filter card made by AMC.

The filter card being used is FC15030, shown Figure 13. This card is placed between the output of the AMC30A20AC and the steering motor.

Figure 13: AMC Filter Card

Mechanical Improvements

Brake Actuator

The gas and brake are controlled by two clutched linear actuators. This type of actuator was chosen by the 2009-2010 SCOPE Team for safety considerations. A typical linear actuator has a large holding force, meaning that if its power is cut, it will keep the actuated pedal in the last commanded position, which can be dangerous for a gas/brake pedal. The clutch eliminates this problem, allowing the actuator to be back-driven by the driver once power is cut.

These actuators are of unfortunately inexpensive manufacture, and have failed occasionally. The 2009-2010 SCOPE Team had to replace one misbehaving actuator. We had a similar problem when the brake actuator output shaft hit a steel mounting beam during an aggressive braking event. We cut out the conflicting beam metal to prevent this problem in the future, but also had to fix the actuator.

We added a small sheet metal shim inside the brake actuator to keep the output shaft from binding under a bending moment. However, the actuator should probably be outright replaced.

Rear Encoder

The vehicle rear wheel encoder, mounted last year, is used for determining vehicle velocity and position estimation. Unfortunately, the chain connecting the encoder with the engine shaft sometimes came off during rough terrain testing. The existed mounting scheme also included no way to tension this chain.

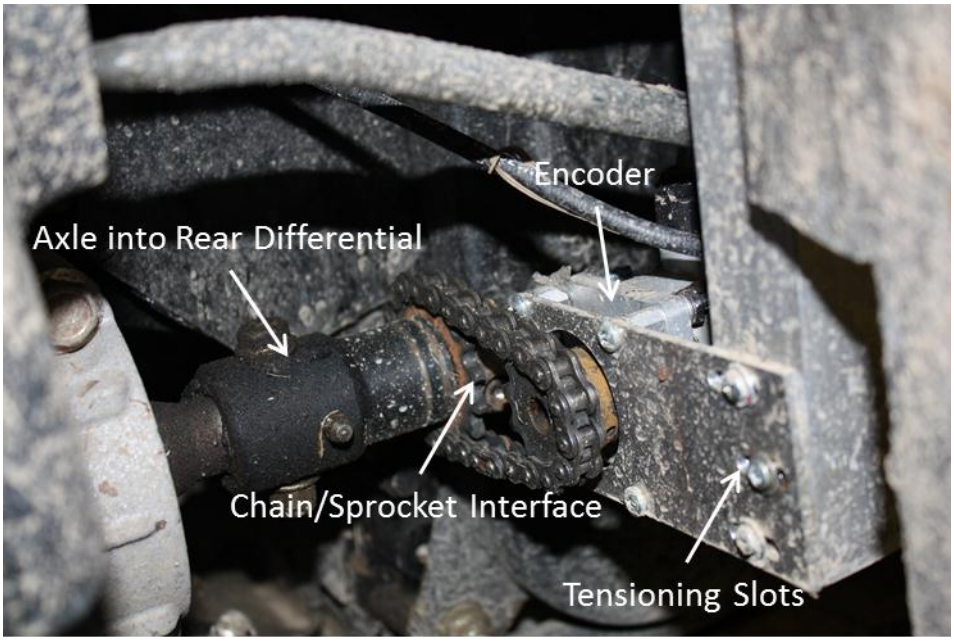


Figure 14: Rear Encoder Mount – The new encoder mount mitigates the chain derailment problem by both shortening and tensioning the chain.

Our improved mount lowers the distance of chain travel and includes slots for properly tensioning the chain. The encoder has never failed since the new system was installed.

Steering Wheel Endstops

The vehicle has two steering maximum angle detectors to warn the system if the steering slews too far in either direction, allowing the computation system to prevent the steering motor from attempting to power through the hard steering endstop. This system doubles as a method for zeroing the steering encoder count. In the steering test, the steering is actuated in each direction until it hits each endstop, then computes the known encoder count of a straight course.

These detectors are two Cherry Hall effect sensors that detect two magnets mounted on the steering tie-rod. The Hall Effect sensors are accessible from the cabin, under the brake pedal.

This steering endstop system was implemented by the Draper 2009-2010 SCOPE team, but failed from wear over the summer. The magnets were mounted on the tie-rod by two hose clamps, which slowly moved as the car vibrated. Additionally, the tie-rod moves vertically as the vehicle suspension takes loads, aggravating the problem to the point that vehicle consistently failed steering tests.

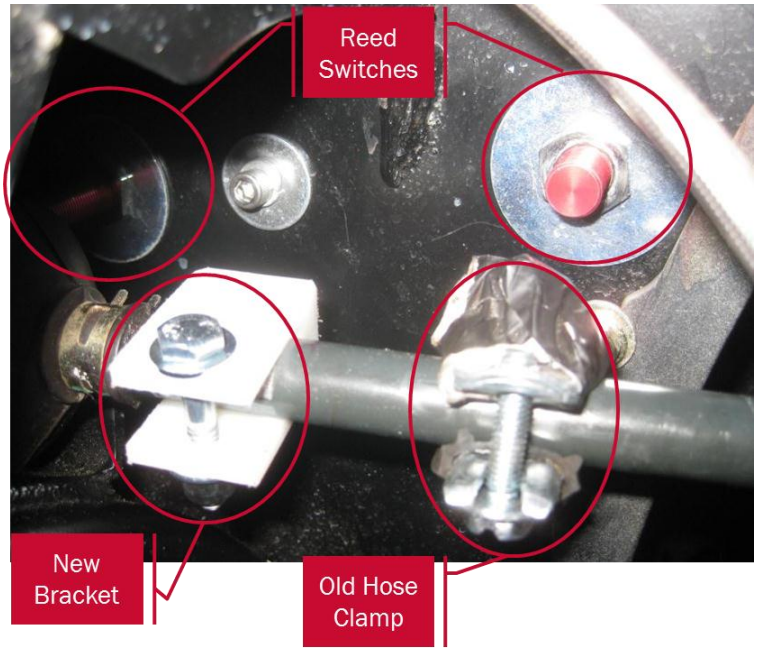


Figure 15: New and Old Magnet Mounting System

The new bracket clamps to the tie rod on a hex interface, while the old system was not well constrained.

We fixed this problem by adding a bracket that clamps onto a hexed portion of the tie rod shaft. This magnet is both axially and rotationally constrained by the hex. The bracket is also mounted substantially nearer to the steering rack, meaning it moves less vertically with the car suspension. If this continues to be a problem, we may replace the magnet with a large vertical bar magnet to always ensure tripping the Hall effect sensor.

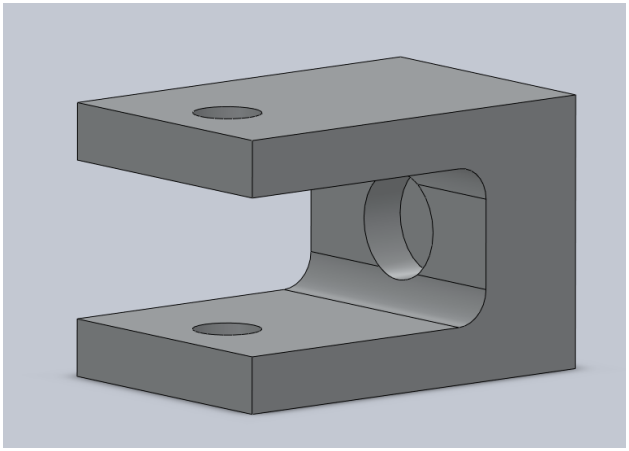


Figure 16: Steering Limit Magnet Bracket

The bracket clamps on the shaft with the large U cavity, and two magnets were press fit into the circular pocket. This part was designed for milling machine fabrication.

INS Mount

The Microstrain INS determines the vehicle acceleration, angular rate, and absolute attitude. The INS uses a combination of accelerometers, gyros, and a three-axis magnetometer to measure these quantities, and is therefore sensitive to both large magnetic fields and vehicle vibration. After experimenting with several locations across the vehicle, we determined that the rear of the electronics box was the most optimum location for mitigating both effects. The rear of the box was also already vibration damped, and is relatively far from steel chassis members.

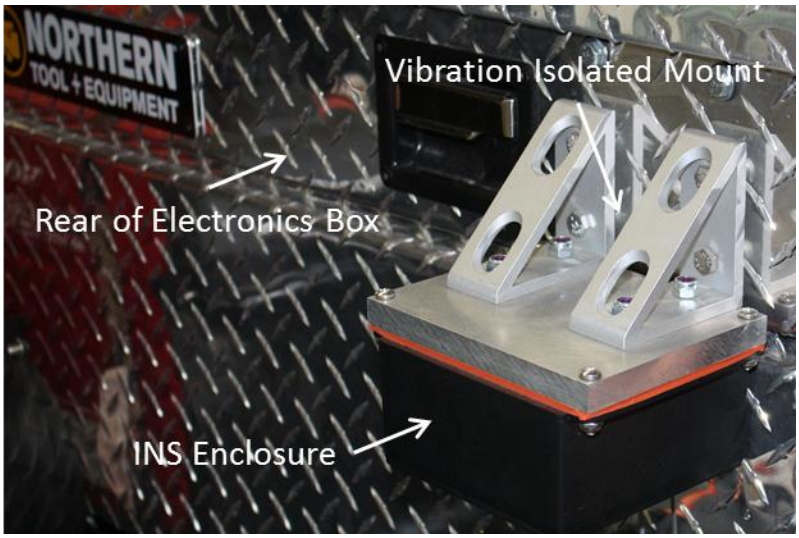


Figure 17: INS Mount

The INS is mounted to the bottom of a small plastic enclosure box, which is bolted upside-down for better rain protection. The entire mount is further vibration isolated by rubber sheeting and washers.

This mounting scheme certainly helped improve the INS data, but the signal was still too noisy. We only found the raw magnetometer readings useful, and usually only used the cleaner GPS heading (while time-delayed) for vehicle missions.

Box Renovation

The major changes to the mounted electronics box components, and the sizes of the larger parts, are listed below:

Original Part	Updated Part
Stealth PC 7.3” x 9.5” x 6.25”	Dell R5400 Workstation 17” x 27” x 3.5”
DC UPS 3.9” x 5.2” x 4.7”	AC UPS 8” x 5” x 14”
48V Victor HV Motor Controller 2.2” x 2.7” x 2.09” 48V Power Supply 9.8” x 5” x 2.08”	30A20AC AMC Servo Drive 4.24” x 7.35” x 2.45”
NI 9074 cRIO	NI PCIe-7852R FPGA 2x NI9151 R-Series Expansion

Many of the updated devices are significant larger than their original counterparts; because of this, adding in new hardware cannot be as simple as replacing each corresponding component in the same space. Instead, most of the layout needs to be reconfigured, while keeping many of the same connections.

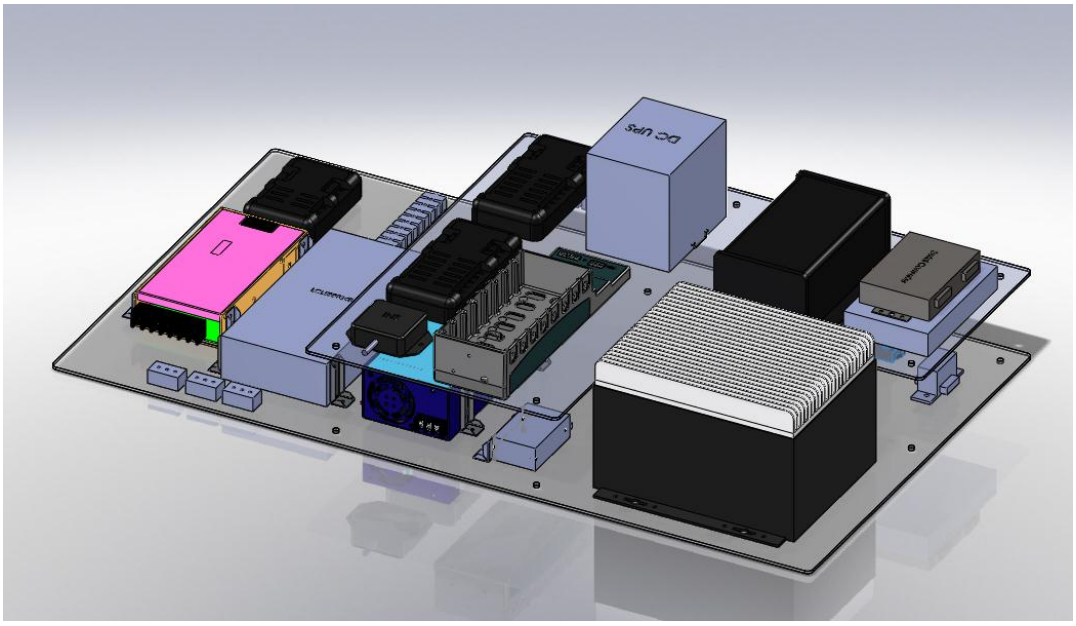


Figure 18: Original Electronics Box Layout

The original layout is shown above.

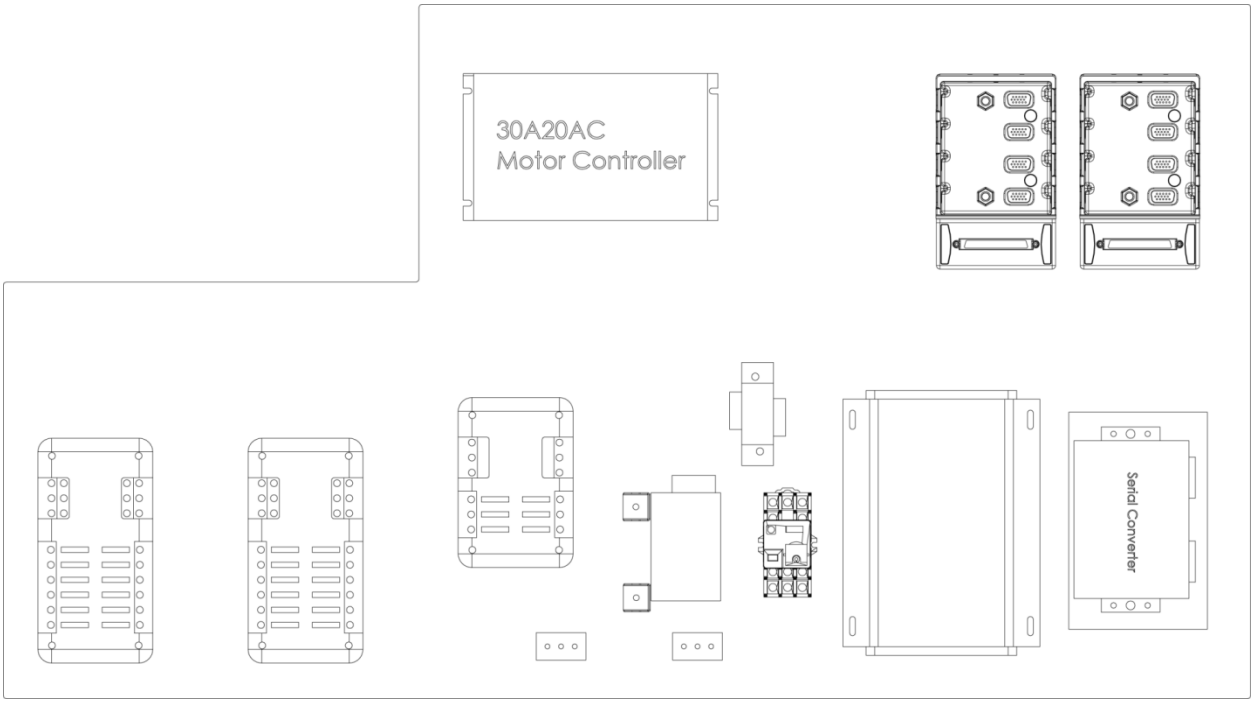


Figure 19: New electronics box layout: top layer

The new layout features a significant larger upper layer; most of the bottom layer is allocated to the workstation, and several components that were on the bottom layer are moved to the top. In addition, the AC UPS, not shown in this diagram, is moved to the side of the container, to allow for more space.

Wiring diagrams for this revision are included in the Appendices.

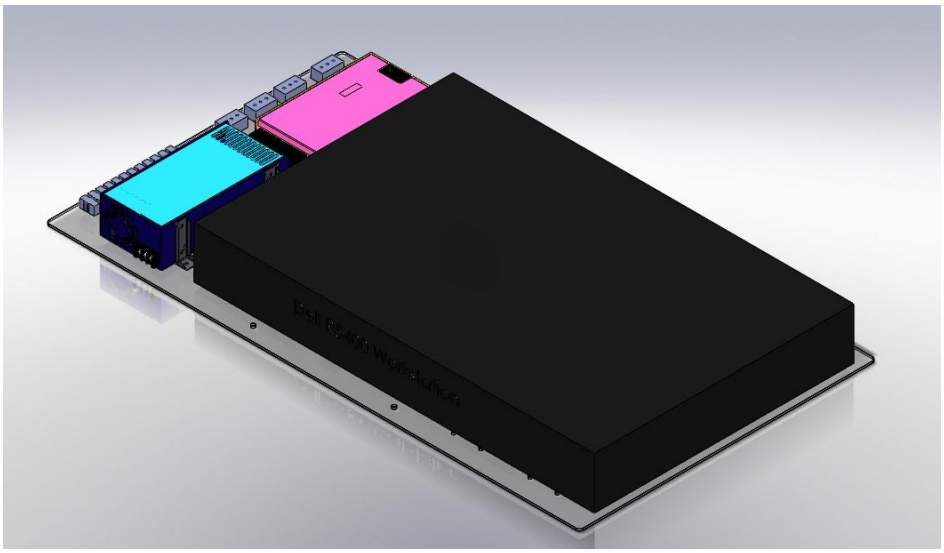


Figure 20: New electronics box layout, bottom layer

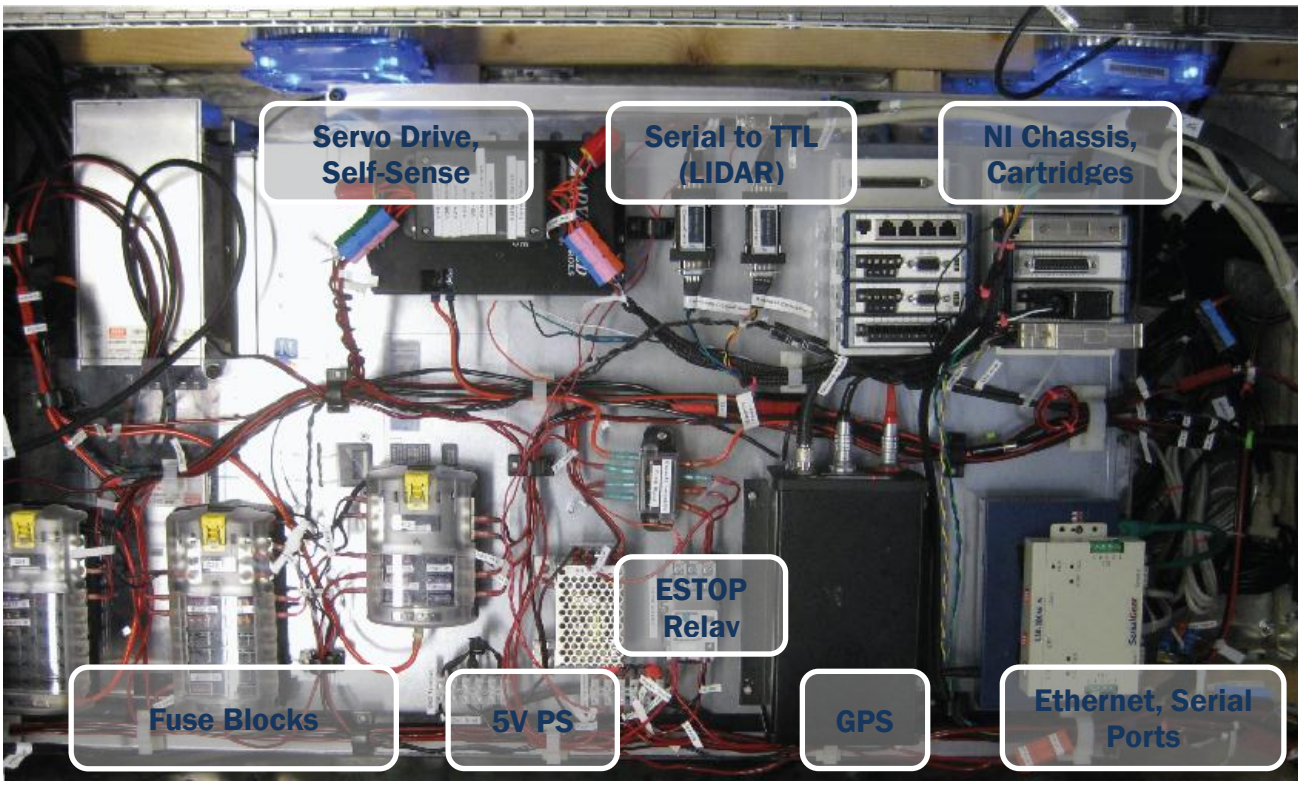


Figure 21: Current state of electronics box

Vehicle Cabin

All lines enter through the back of the cabin, and are routed down the center, between the two seats. This setup allows drivers and programmers in the vehicle more mobility. In addition, all connections that run down the center of the cabin can be easily disconnected for debugging.

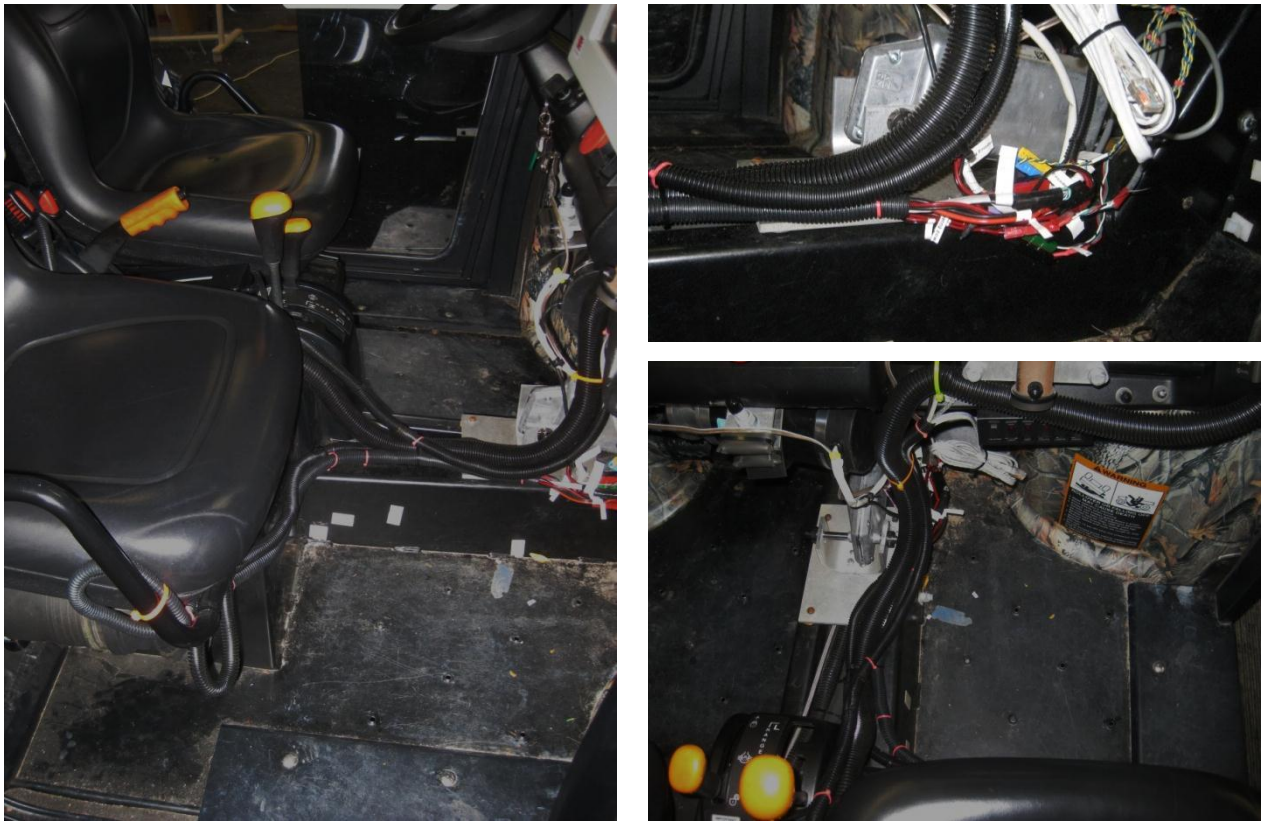


Figure 22: Vehicle Cabin

Software

Development Process

Team Draper developed the entire vehicle codebase in LabVIEW. The team used a waterfall development methodology (shown in Figure 23). This began by identifying vehicle capability requirements as a team. Then, a software sub-team lead by the architect would design an algorithm set to achieve the desired capabilities. This plan was captured by a block diagram created by the architect. These plans would then be reviewed, either internally or for major decisions, by external reviewers.

Following the planning stage, individual developers would be charged with the development of VI's. These Developers would then implement and test their function in isolation – using self-written test cases. Once they believed they had achieved proper function, their code would be integrated into the main codebase and the new software would be tested on vehicle in the field.

One suggested area of future development would be a whole vehicle simulator. While standalone testing is important (and should not be replaced by whole vehicle simulation), being able to make a first pass at removal of integration errors in off line simulation would decrease integration debug time.

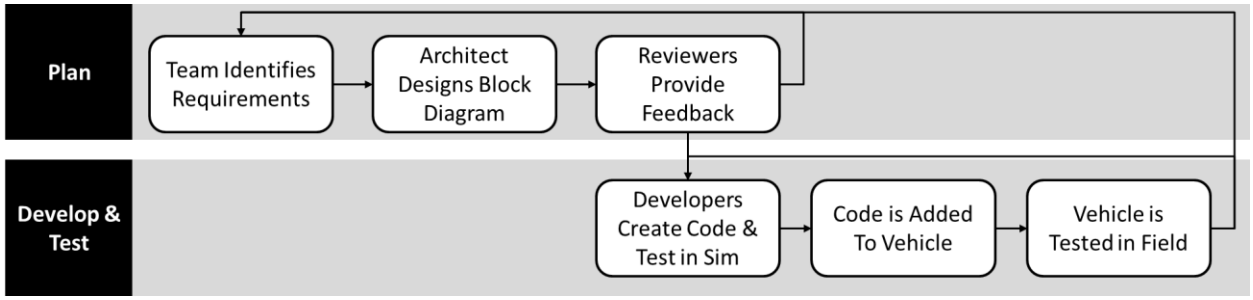


Figure 23: The waterfall development process used by the Draper SCOPE team.

To facilitate this process, the team uses SVN source control from TortoiseSVN. This requires linking the Tortoise tools with LavVIEW merge and diff tools. Instructions for repository environment set up can be found in the appendicles. Additionally, the team uses a few useful conventions to make try to avoid conflicts:

1. Don't use virtual folders. Instead, all folders should be auto-populating folders, which copy the file system's structure. To add an auto-populating folder to the project, make sure it exists in the file system. Then right click in the project tree and select Add>Folder(Auto-populating)… All nested folders and VI's will be added to the project.
2. All VI's should be in an auto-populating folder.
3. If you're currently working on a VI, rename the VI as <YourName><VIName>.vi. Once you think the VI is finished and debugged, remove your name.
4. Don't edit VI's that have a team member's name in them before asking for permission.

5. When committing, always attach useful comments. These should include what changes were made and what state of robustness those changes are in (first draft of code, in debug, debugged).

Faster FPGA Compile Times

Currently, a major challenge of the development process is writing and debugging code for the FPGA. Because the LabVIEW code must be compiled every time it is modified and run, this makes debugging a time intensive process. Potential remedies for this issue include investigating NI's cloud computing compile options.

Olin Robotics Architecture

Team Draper has organized the Ella codebase around the emerging Olin Robotics Architecture. This architecture, described in greater detail below, is a modified three layer architecture with a traditional, process based decomposition of software tasks.

Software Decomposition

The Olin Robotics Architecture, diagramed in Figure 24, separates software modules vertically by abstraction level and memory duration and horizontally by tasked performed. The vertical decomposition has three layers named after the different cognitive levels in a human brain: the hindbrain, the midbrain, and the forebrain. The hindbrain is responsible for processing the majority of sensor input and executing position and velocity control of motors. All of this is done on Ella's FPGA through an NI R-Series chassis with C-Series modules. The midbrain constructs ephemeral representations of the world in order to execute reactive behaviors. Finally, the forebrain constructs long term memory representations, decides vehicle metastate and objectives, and tunes the behaviors generated by the Midbrain.

Within each layer, software tasks are decomposed into generating meaningful sensor representations, using these representations to propose control decisions, and selecting from proposed control decisions. These different processes are summarized as sense, think, and act. Each process runs asynchronously from other processes on its own dedicated core. This asynchronous architecture allows higher throughput through pipelining.

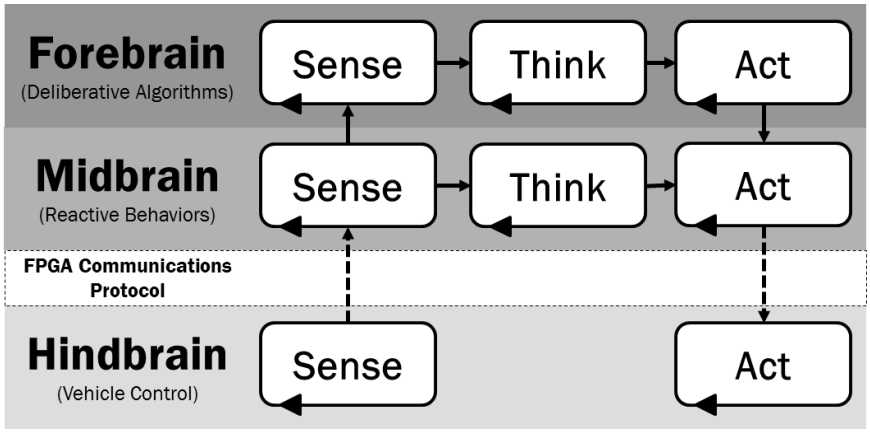


Figure 24: The Olin Robot Architecture as implemented on Ella. Each loop is a truly parallel, asynchronous process with a dedicated core.

Communication

Communication between processes is done in one of two ways. There are two primary types of communication: between layers and within layers. Within a layer, data is passed between loops using local variables. Between layers, information is passed using global variables called Neurons. A neuron is a cluster of related data. For example, Ella has a "Where Neuron" that contains all of the components of the vehicle's current pose. These global variables cannot be passed between the FPGA and PC, however. For this, we have created an FPGA Communications Protocol. This protocol is responsible for transmitting data which is then decoded on the other fabric and passed into a global variable. This wrapper allows data passing between the hindbrain and other layers to be treated like a global variable.

Each of the architecture layers and the software developed on them will be described in greater detail in the next section.

Algorithms

Forebrain

The forebrain is the highest level in our three-layer architecture. It contains higher-level mission planning and execution functionality. In the forebrain, the user specifies a mission, and as the vehicle executes it, the forebrain produces a desired heading and desired velocity which is then incorporated in an arbiter in the midbrain to decide how the vehicle actually moves.

Mission Planning

The Mission Planning interface allows the operator to plan a mission for the vehicle to execute. There are currently four different types of missions that can be run: manual mission, waypoint mission, teach mission, and playback mission.

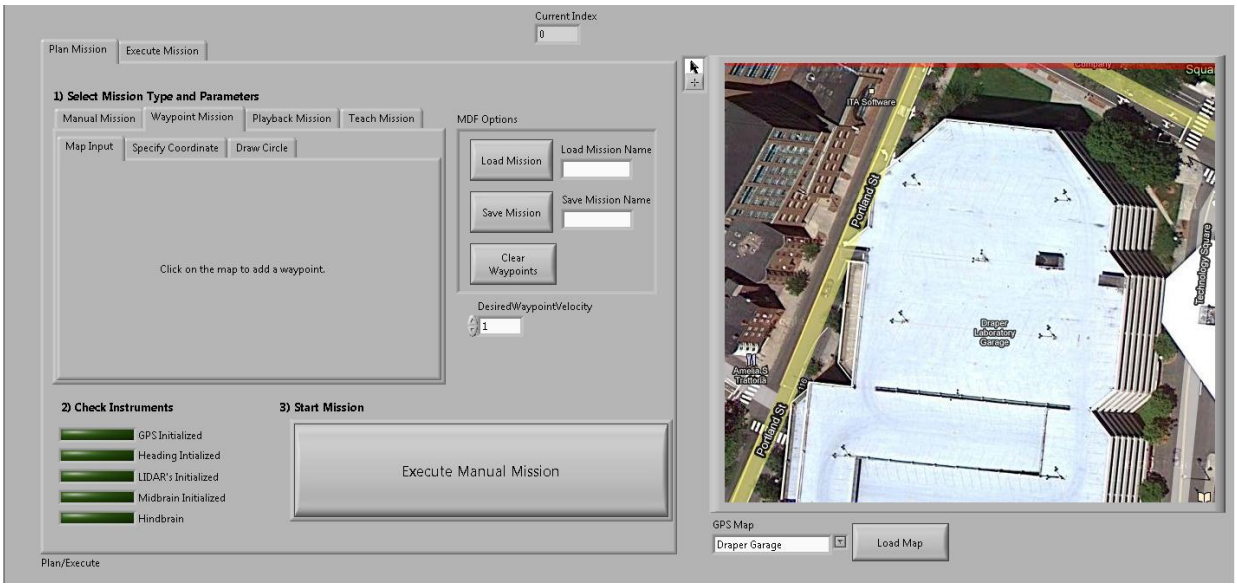


Figure 25 Mission Planning Interface

Manual Mission

Starting a manual mission allows the operator to set the desired velocity and wheel angle manually, using either the GUI or keyboard commands.

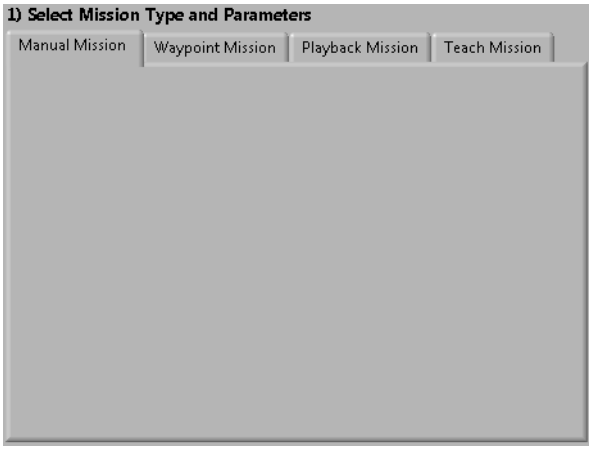


Figure 26 Manual Mission Interface

Waypoint Mission

A waypoint mission allows the user to specify a set of waypoints for vehicle to drive to. These waypoints can either be entered by clicking on the satellite map of the terrain, by manually typing in waypoint coordinates, or by creating concentric circles of waypoints by specifying the center location, radius, number of waypoints, and spacing between concentric circles. Additionally, missions can be saved, and previously created missions can be loaded.

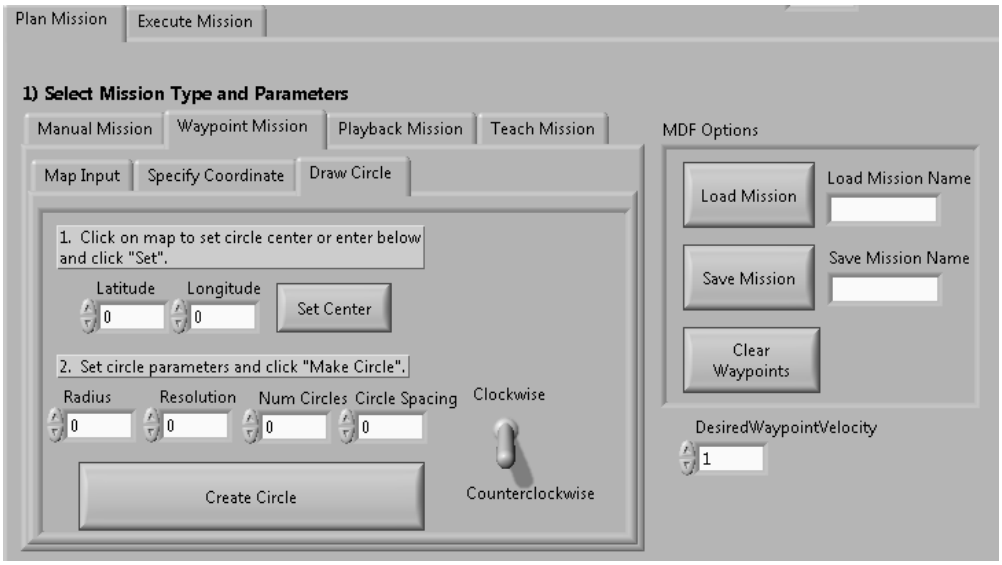


Figure 27 Waypoint Mission Interface

Teach Mission

Teach missions allow the operator to teach the vehicle a path for later playback. A GPS log of the taught mission is recorded and saved to file.

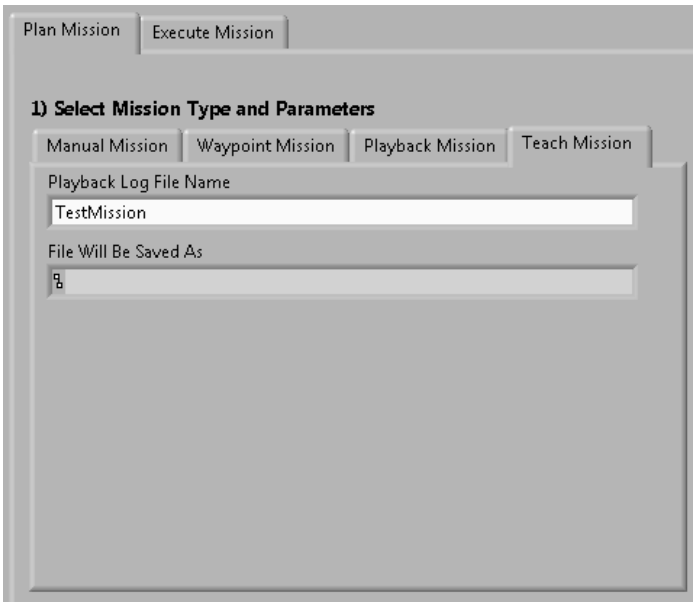


Figure 28 Teach Mission Interface

Playback Mission

In a playback mission, the vehicle will redrive a path taught in a teach mission. The operator selects the desired log file, and then specifies the minimum spacing between waypoints and the number of times that the path should be loaded.

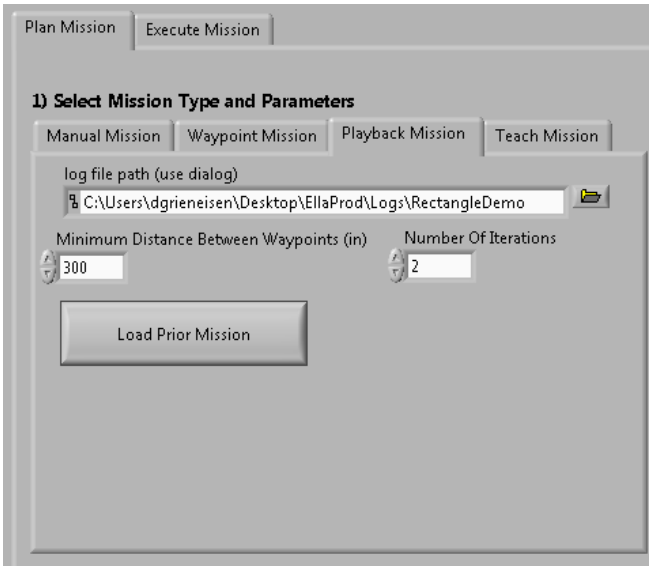


Figure 29 Playback Mission Interface

Mission Definition File

Once a waypoint or playback mission is planned, a mission definition file (MDF) is created. The MDF contains an ordered array of waypoints and a value specifying which waypoint the vehicle is currently executing. Each waypoint consists of a structure containing latitude, longitude, desired velocity to get to the waypoint, and desired heading once the waypoint is reached. This structure is defined using a strict type-def, so it can be easily expanded in the future to include more information, such as desired behavior, commands for the payload, etc.

Mission Execution

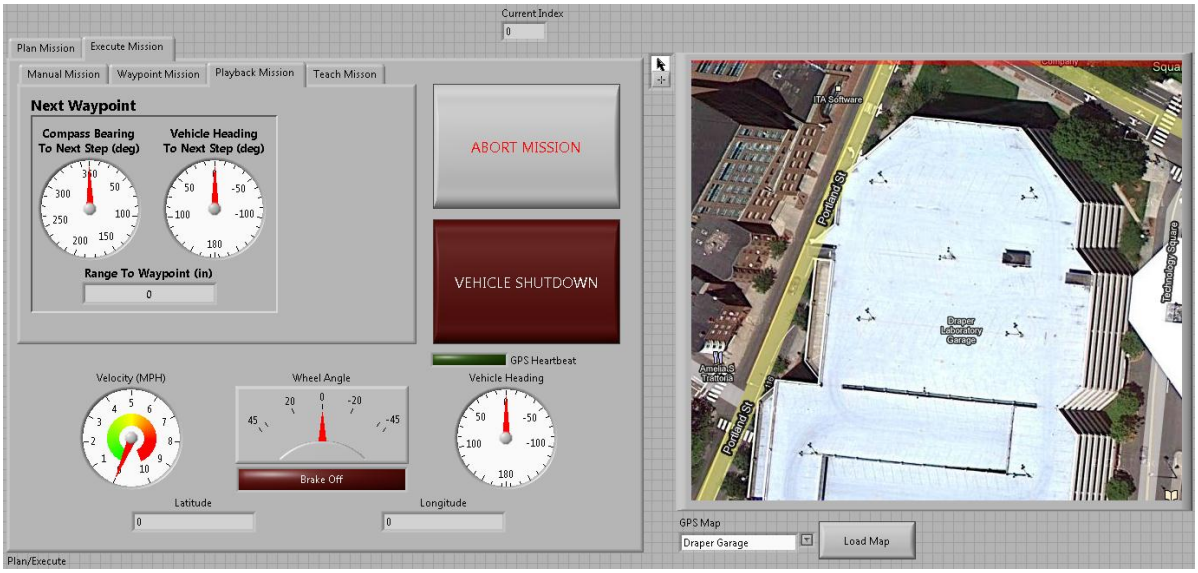


Figure 30 Mission Execution Interface

Once the operator finishes planning the mission, it is then executed by the vehicle. As the mission is executed, the forebrain provides desired heading and velocity commands to the arbiter in the midbrain. The mission execution interface displays information about the vehicles state and relationship to the next waypoint, as well as the vehicle and waypoint locations on a satellite image of the terrain.

Waypoint Reached Metric

When executing an MDF, the vehicle uses two metrics to determine when it should move on to the next waypoint. The first is if the vehicle approaches within 96 inches of the desired waypoint. In this case, the vehicle is sufficiently close to the waypoint that it has successfully reached it, and it can move on.

There are, however, certain instances when this does not suffice. If the vehicle is forced to deviate from its course due to obstacles, then it may not be able to reach the current waypoint in the path. If the vehicle is within 15 feet of the waypoint and the difference between the current angle of the wheels and the angle to the waypoint is greater than 60°, then it is probably not reachable by the vehicle. In this case, the vehicle forgets about the waypoint it is currently pursuing and moves on to the next one. This allows the vehicle to avoid situations in which it has to circle around to try to reach a waypoint that it missed, rather than continuing to follow the desired path. In the future, we would like to implement a more intelligent waypoint-forgetting metric that takes into account an accurate model of vehicle dynamics, but this is sufficient for now.

Midbrain

The midbrain is responsible for executing the robots fast reactive behaviors. As such, the layer enforces two conventions. First, to insure speed, each process must execute in fewer than 50ms.

Additionally, in order to insure ephemeral memories, all representations should last for fewer than 500ms.

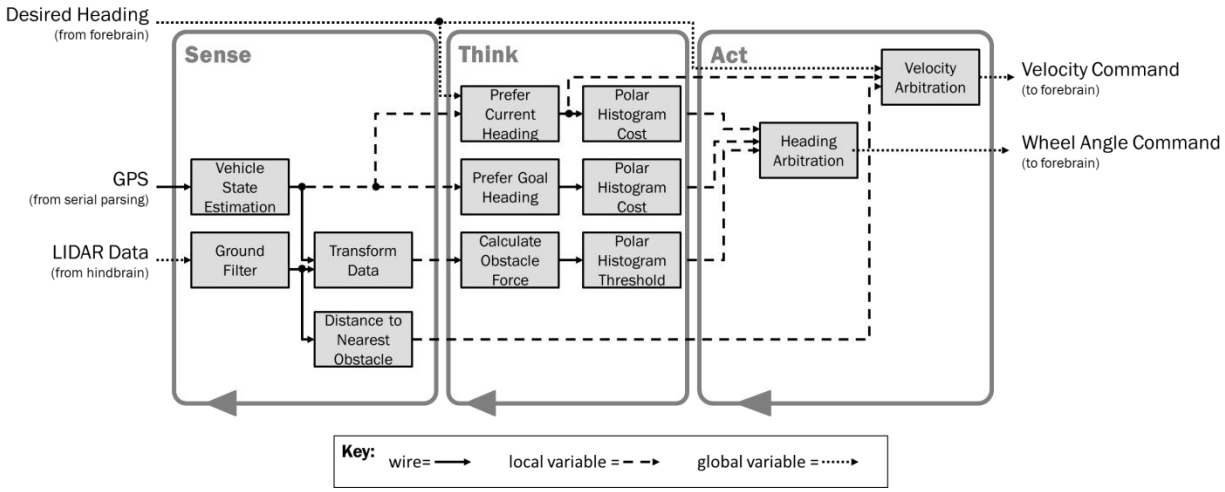


Figure 31: The architecture of the midbrain.

The midbrain has three main processes: sensing, thinking, and acting. In this layer, the sensing process creates ephemeral representations of the world in the moment. The reactive behaviors in the thinking process then use these representations to propose vehicle actions. Finally, the action process selects what vehicle action should be executed, and passes that action to the forebrain for execution. Each of these processes will be discussed in greater detail in the following sections.

Sense

The sensing process takes partially transformed data and creates ephemeral representations that can be used by the thinking processes reactive behaviors. As we can see from Figure 31, this layer processes GPS data and LIDAR data.

LIDAR Filtering

In order to avoid obstacles, they must first be identified. The Draper team did this by identifying points they that were part of the ground plane, and then labeling all other points obstacles. Below is an example LIDAR scan and the altitude as a function of position in the scan. This example will be used to explain the two methods used to filter data:

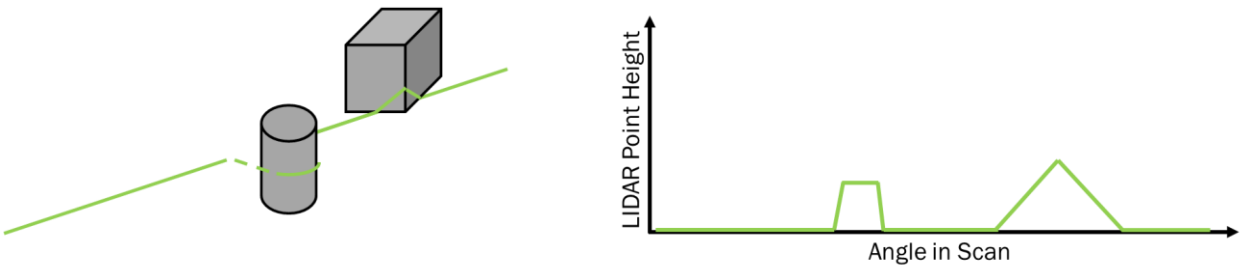


Figure 32: LIDAR scan of obstacles

Figure 32 shows where a LIDAR scan intersects a typical set of obstacles. At right is the height of the resulting scan points.

The two obstacles have a very different LIDAR signature based on where the scan intersects them. In the left obstacle, the scan intersects the obstacle well above the ground plane causing a discontinuity in the height data. However, in the second obstacle, the scan intersects the obstacle on the ground plane, resulting in a gradual increase in height of scan points. While our environments have many different, much less geometric shapes, the beam intersection pattern still almost always follows one of these two patterns. To detect the two different signatures, we use two different filters.

The first of these filters is a probabilistic ground height filter. For this filter, the frequency of the heights of scan points is modeled in a histogram (see Figure 33). The mode of the distribution is take to be the average ground height. Then, some deviation from that average is considered part of the ground plane. All other points are classified as obstacles.

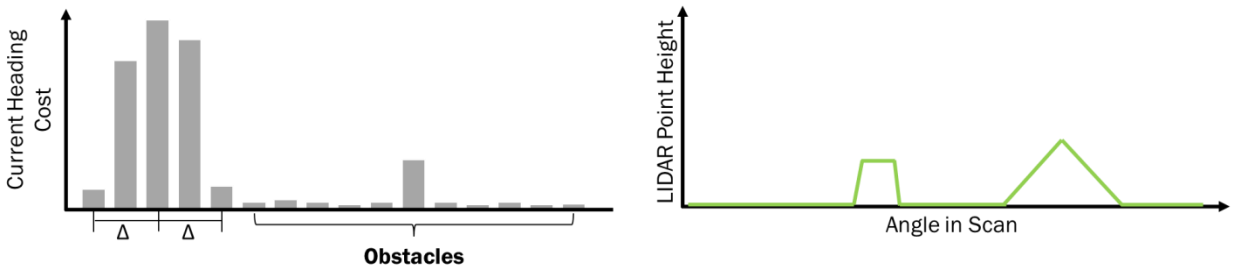


Figure 33: Probabilistic ground plane filter

The probabilistic ground plane filter uses a histogram of height frequencies to find points that lie outside the ground plane, which are then colored as obstacles. This filter provides an estimate of ground height that will be used in our second filter.

The second filter is a modified derivative filter. This filter passes over the scan's height values using a finite state machine shown in Figure 34. The machine has two states the last point was not an obstacle, or the last point was an obstacle. The start state is determined by whether or not the probabilistic ground height filter colors the first point as an obstacle. For this example, we assume the starting point is not an obstacle. The next point is classified by taking the difference in height to the next point. If this difference exceeds some threshold (four inches in our implementation) then it is flagged as an obstacle, otherwise it is not classified as an obstacle. Once a point has been classified as an obstacle, subsequent points are classified differently. Those points' heights' are compared to the height of the last known ground point. If the difference lies outside of some threshold (eight inches in our implementation), then the new points are also classified as obstacles. Otherwise, they are not classified as obstacles, and the machine transitions back to the not an obstacle state.

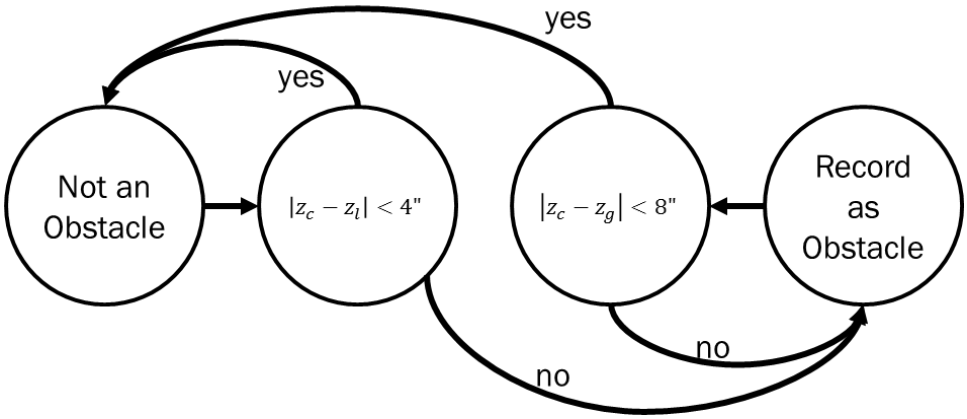


Figure 34: Finite state machine representation

The Finite State Machine processes the height of LIDAR points to determine whether or not the points were obstacles. The start state is defined by whether or not the probabilistic height filter classifies the first point as an obstacle or not.

State Estimation

The global pose of the vehicle is measured using the GPS. It provides position measurements with decimeter accuracy and heading measurements with degree accuracy at 5 Hz. We then use a simple state estimation formula, as shown below, which combines the pose measurement from the GPS with an estimate made from numerically integrating the steering angle and velocity measurements. This allows the vehicle to have a continuous smooth estimate of pose, rather than discrete jumps every 200 milliseconds.

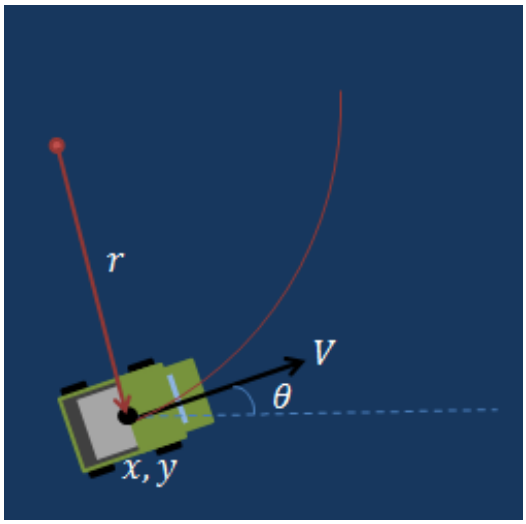


Figure 35: State Estimation

State Sensors

- GPS-Position: x_{meas}, y_{meas}
- GPS-Yaw: θ_{meas}

Model Parameters

- Wheel Encoder – Velocity: V
- Steering Encoder – Radius: r

$$\dot{x}_{est} = Vcos(\theta_{est}) + L_1(x_{meas} - x_{est})$$
$$\dot{y}_{est} = Vsin(\theta_{est}) + L_1(y_{meas} - y_{est})$$
$$\dot{\theta}_{est} = V/r + L_2(\theta_{meas} - \theta_{est})$$

Think

The think process is responsible for generating proposals for vehicle action. It does this with several fast, reactive behaviors. Each of these behaviors generates its own action proposal. All proposals are modeled as a histogram, as shown in Figure 36.

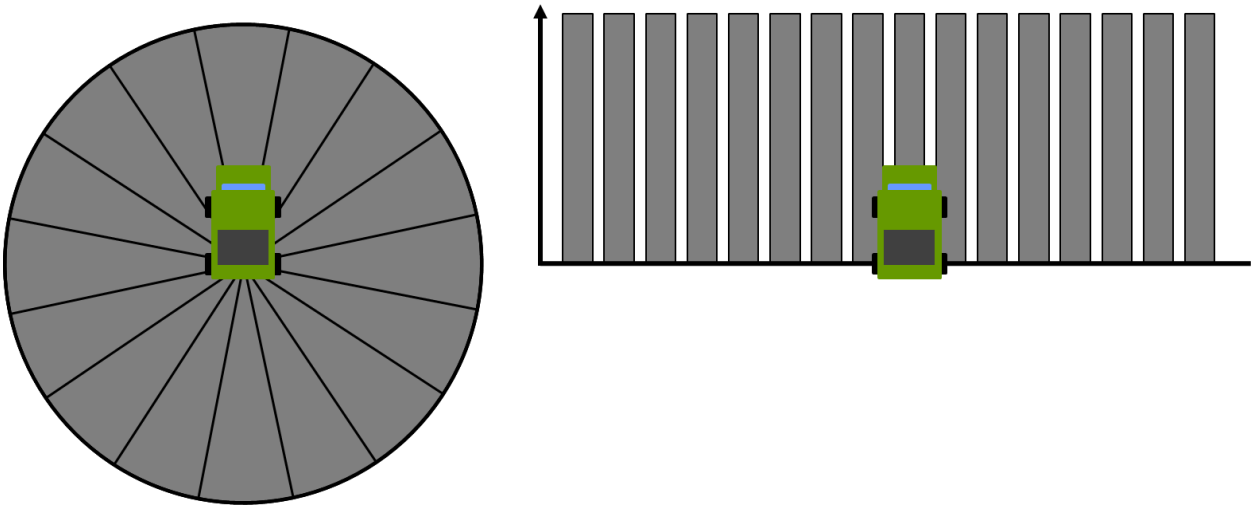


Figure 36: Vehicle action proposals take the function of a polar histogram

Each sector of the polar grid shown above is represented by a bin in the histogram. In truth, our vehicle has 360 bins, one for each degree. However, for ease of visualization, the number of bins has been reduced.

Behaviors can propose that the vehicle move in any polar vector. These vectors are defined as the middle of each polar sectors (shown in Figure 36 left). Proposals are generated through one of two methods, either a threshold proposal or a cost proposal. Cost proposals assign a cost of traveling in each vector. Vectors with higher costs are less desirable than those with lower cost. Threshold proposals mark vectors as either drivable or undrivable. Thresholds are typically made by generating a cost function and then marking all paths above some threshold cost as undrivable. This approach is modeled off of vector field histograms developed by Johann Borenstein and Yoram Koren (Borenstein & Koren, 1991).

The implemented set of behaviors is modeled after Iwan Ulrich and Johann Borenstiens VFH+ Obstacle avoidance (Ulrich & Borenstein, 1998). The original paper should be read for an indepth understanding of this topic. However, a brief overview will be presented here. There are three major behaviors: obstacle avoidance, goal heading preference, and current heading preference.

Obstacle Avoidance

To avoid obstacles, the obstacle avoidance behavior forms a threshold proposal. Figure 37 shows an example obstacle set and how the behavior forms a threshold proposal.

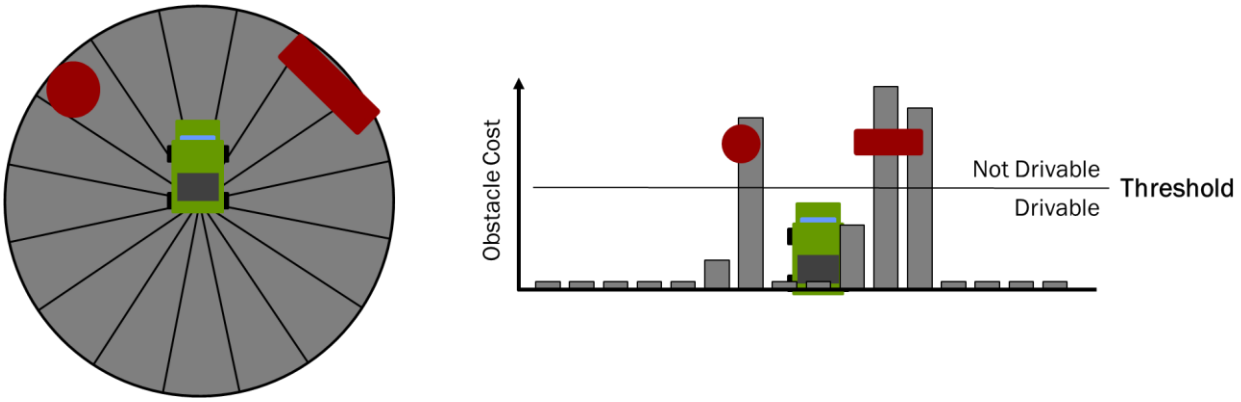


Figure 37: The threshold proposal for an example obstacle set (obstacles shown in red).

In truth, not all vectors below the threshold are chosen as chosen as candidate vectors. Instead, narrow vallies and wide valleys (where a valley is a region of drivable vectors bounded by undrivable vectors) are treated differently. For more information on this, refer to Ulrich & Borenstein, 1998.

Prefer Goal and Current Heading

There are two cost functions that also generate action proposals. These make the vehicle prefer vectors near the goal vector and near the current vector – allowing the vehicle to move towards a goal and prefer straight trajectories respectively. This is done by applying an inverse Gaussian distribution with the mean at the desired or current heading, as shown in Figure 38.

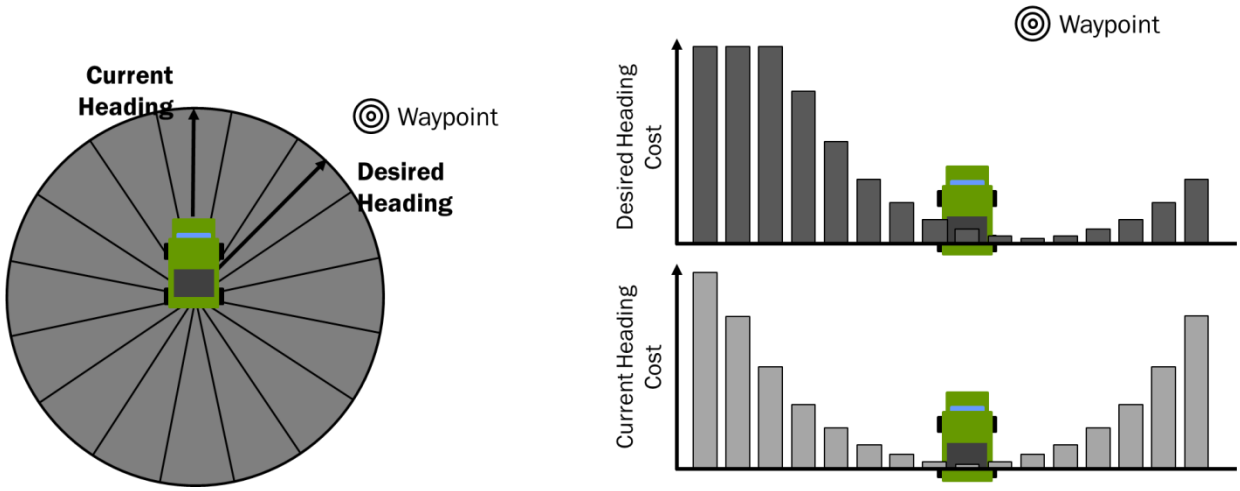


Figure 38: The cost functions for an example current and desired heading.

Act

The act process selects a vehicle action from the generated proposals using a blending arbiter. This involves making two decisions: setting the heading and velocity. The heading arbitration is a two step process. First, all cost proposals are incorporated. Each cost proposal is multiplied times a gain to give it an influence weight and then summed, giving the total cost of each vector.

$$c_{total} = \sum_i g_i \times c_i$$

Then any vector which has been marked as undriveable by a threshold proposal is removed from the set of possible vectors. The arbiter then selects the minimum cost vector and commands.

To determine the velocity of the vehicle, three parameters are considered. These parameters can be seen in seen in Figure 39. The desired wheel angle, θ_d, is the difference between the current wheel angle and the desired heading. The current wheel angle is defined as θ_c. Finally, we consider the distance to the nearest obstacle, d_n.

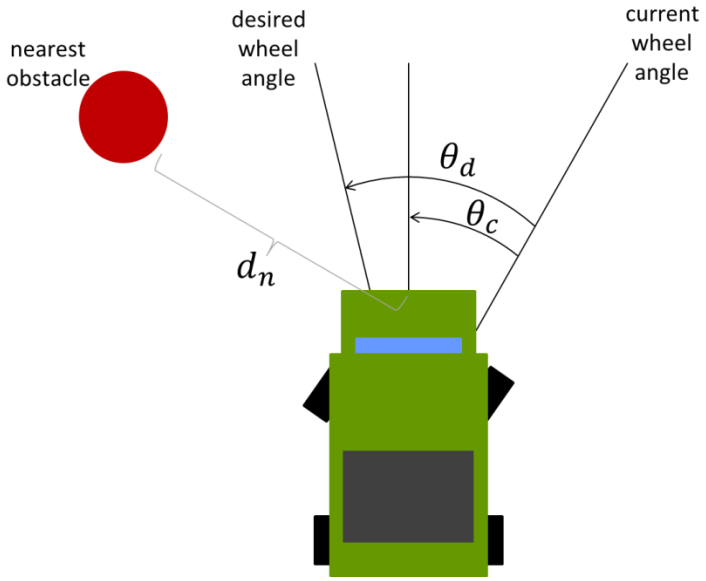


Figure 39: Parameters of the velocity arbiter.

We then define the velocity with the following equation:

$$V = V_{max}\left[1 - max\left(g_1 \times \frac{d_n}{d_{max}}^2, g_2 \times \frac{\theta_d}{\theta_{dmax}}^2, g_3 \times \frac{\theta_c}{\theta_{cmax}}^2\right)\right]$$

Thus, the proportion of each parameter is allowed to slow down the vehicle by some amount, which is governed by the gain term. Whichever parameter slows the vehicle down most is given control of the velocity.

Hindbrain

The Hindbrain runs on the FPGA. It is responsible for the low level acting and sensing, as detailed below.

Act

Steering Controller

The Steering Controller sets the wheels to desired steering wheel angle. It reads in the steering wheel encoder and sends an analog voltage to the steering wheel motor controller. There are two continuously running loops in the VI.

One loop reads the quadrature encoders from the steering wheel column. This is converted to a steering wheel angle and published as a global variable. This angle should be zeroed so that it is referenced from a wheel angle that drives the car straight. This could potentially be done using the steering wheel endstops (further detail below).

The second loop implements a PID controller for the steering motor to drive the steering wheel to any desired angle. Currently, this is being used simply as a proportional controller since there is no noticeable steady state error in our driving conditions; the steering wheel motor has a very good response.

There are several features that work in conjunction with this control loop. A deadband zone is implemented for the motor so that it does not needlessly jitter when the steering wheel angle is sufficiently close to the desired angle. The controller is also aware of the current state of the wheel and will not drive it past a certain angle to ensure that the column is not damaged (assuming that the wheel has been properly zeroed). Additionally, the controller is aware of the current state of the endstops and will not drive the wheel past these endstops.

Throttle Controller

The Throttle Controller is responsible for the vehicle's speed. It reads in the rear wheel encoder and commands the linear actuators for the gas and brake pedals.

One loop reads the quadrature encoders from the rear wheel. Assuming no slip, this is converted to a vehicle speed in miles per hour, which is published as a global variable.

The second loop implements a PID controller for the linear actuator that controls the gas pedal. While a PID controller seems to work reasonably well, there were several issues that had to be dealt with. There is a variable time delay between when a voltage is commanded in LabVIEW to when the linear actuator responds. Also, the relationship between the input voltage and the acceleration is nonlinear. These challenges were dealt with by tuning the gains by trial and error to give a response that was reasonable in terms of overshoot and settling time.

Additionally, the Throttle Controller can brake the vehicle if so commanded. When the brake is activated, the gas pedal is released.

Sense

Endstops

As detailed in the mechanical section, there are two endstops to detect when we are near the ends of the steering wheel range. There is a magnet attached to the steering wheel shaft which is then picked up by two Hall Effect sensors. When the endstop is detected, the controller does not allow the wheel to be further driven in that direction.

Additionally, it is possible to use the endstops to zero the steering wheel (as the team last year did). However, this is subject to accuracy errors as the detection of the endstops is not very precise.

ESTOP

The ESTOP state is sensed through the FPGA in Hindbrain. This is especially important for the linear actuators. The reference voltage for the linear actuators is determined by the voltage on startup. So, 1 V is commanded to the linear actuators whenever the ESTOP is engaged.

Future Development

Software

Hybrid A*

One of the main future improvements that we would like to implement is an intelligent path planner. Currently, the vehicle tries to drive a straight path to the next GPS waypoint, unless there are obstacles in the way. This puts limits on the scarcity of waypoints. For example, there need to be enough waypoints on a path that the vehicle will try to stay on the path, but they cannot be so dense that if the vehicle has to deviate from the path, it has to loop back to reach the missed waypoints. Ideally, it could be given very sparse waypoints and it would be able to plan an intelligent path between them, replanning when necessary.

We believe that the Hybrid A* Search, developed by researchers at Stanford and implemented on their DARPA Urban Challenge Vehicle Junior, is the best path planning algorithm to implement on our vehicle (http://robots.stanford.edu/papers/junior08.pdf). This planner produces a nearly optimal path which takes into account the vehicle's dynamics. It is advantageous over other A* or D* planners because the path that it produces is guaranteed to be drivable by the vehicle. To implement the Hybrid A* path planner, first the world around the robot is divided up into a grid. Grid squares are on the order of 50 cm per side. Next, short paths that the vehicle can drive are drawn into neighboring grid cells. Once each of these paths is in a new cell, its total cost is estimated. The cost is based on the distance traveled and two heuristics: a non-holonomic no obstacle heuristic, and a holonomic heuristic. The planner then chooses the lowest cost cell that has a path drawn to it from the start and draws paths from this cell into the cells beside it. This process is repeated until a path to the goal is achieved. After the path is planned, it can be optimized using various techniques further discussed in the paper.

Non-holonomic no-obstacle heuristic

The first heuristic accounts for the non-holonomic nature of the vehicle, but does not account for obstacles. For a given (x,y,theta), the non-holonomic heuristic returns the estimated length of a path that the vehicle could actually drive to the goal (x,y,theta) assuming that there are no obstacles in the way. This heuristic is admissible because the vehicle cannot get to the goal in a shorter path than that estimated by this heuristic.

Holonomic obstacle heuristic

The second heuristic does not account for the non-holonomic nature of the vehicle, but it does account for obstacles. This heuristic estimates the total path length from the current node to the goal taking into account the cost of obstacles in the way.

Vehicle Testing

Navcom GPS Performance

GPS is a powerful localization tool for any autonomous vehicle, and greatly speeds development of location-based behaviors by removing the need for complex environment-aware localization. The Gator's GPS is a NavCom VueStar, which can provide centimeter-level position accuracy with an update rate of up to 25Hz. It is currently being run at 5 Hertz. The GPS receiver is located inside the main electronics enclosure and is powered with 24VDC at less than 25W, while the GPS antenna is mounted to the top of the Gator's cab. To test the GPS, we drove the vehicle around various parking lots, roads, and wooded paths around Olin College and logged the GPS position and the course over ground.

First, we considered the GPS coverage around Olin. We took our GPS points, converted them to KML (Keyhole Markup Language, an extension of XML used for geographic annotation) files, and displayed them in Google Earth to generate the following graph. Red indicates a GPS point where we had successfully acquired a signal. A blue line indicates a gap between successive GPS points. This is usually, but not always, due to GPS outage. For example, we started the vehicle in the lower left parking lot, but did not start logging data until we had driven towards the wooden path. An initialization error recorded a point at our original starting location before starting to record the actual GPS path we drove along.

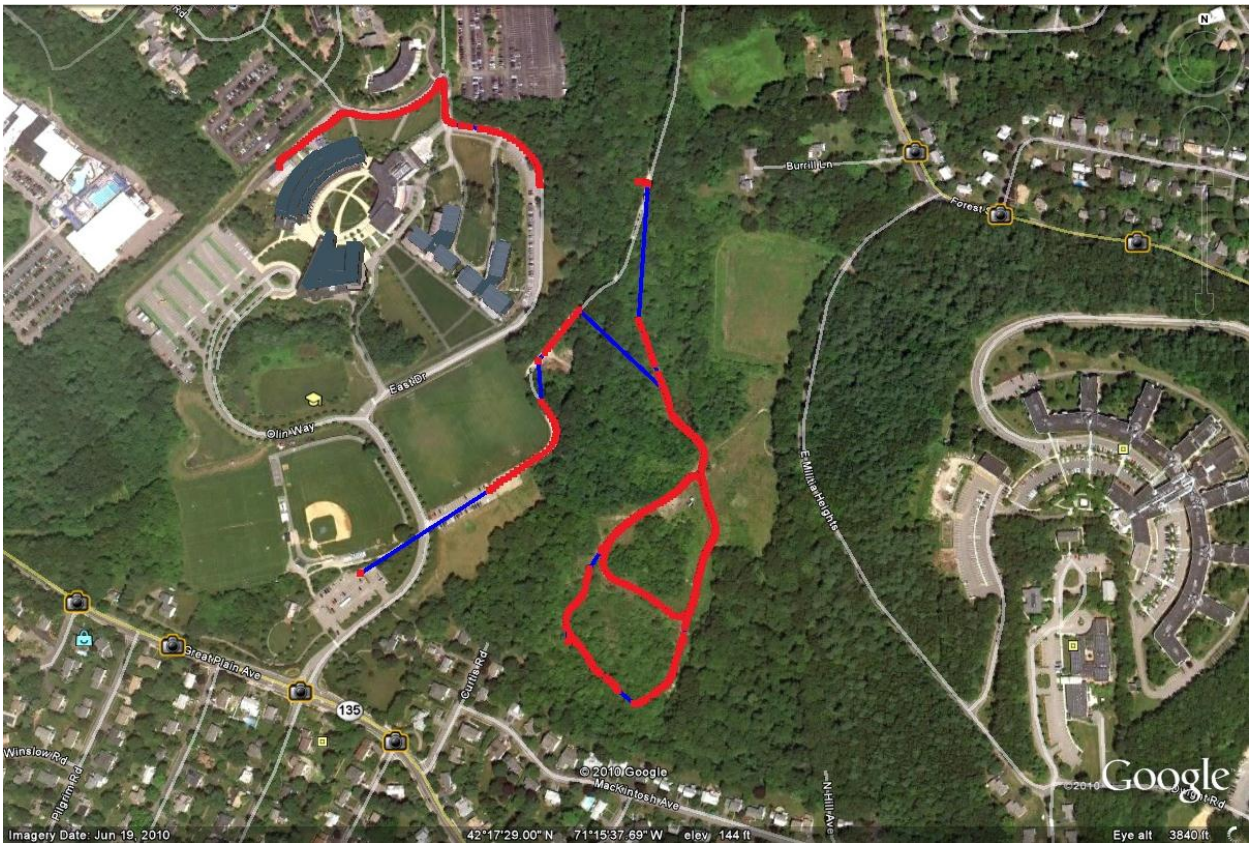


Figure 40: This plot displays GPS coverage around Olin. Red indicates a GPS fix was achieved. A blue line indicates GPS loss.

Looking at his coverage map, we can see that we can expect to lose GPS signal in some wooded areas. However, this outage is normally over fairly small distances and can be dealt with. For the most part, we have a reliable GPS signal.

To consider accuracy of the GPS, we looked at the GPS latitude and longitude coordinates and also the heading given. For this test, we drove the vehicle in a rectangular loop in a parking lot. Additionally, we looked at the heading information. From the graph below, we can determine that there are two minor issues. Firstly, we can see an initialization error when we first start. The GPS reads a heading of zero, which is north, when first recording data. Secondly, the heading is slow to update, which can be seen on the turns. This is due to how the GPS determines heading combined with the fact that we are reading it at 5 Hertz. However, the heading does very well on the straightaways. We believe that, combined with other sensors such as the INS, this will be sufficient for purposes of knowing in which direction we are driving. Additionally, we will try increasing the update rate of the GPS.

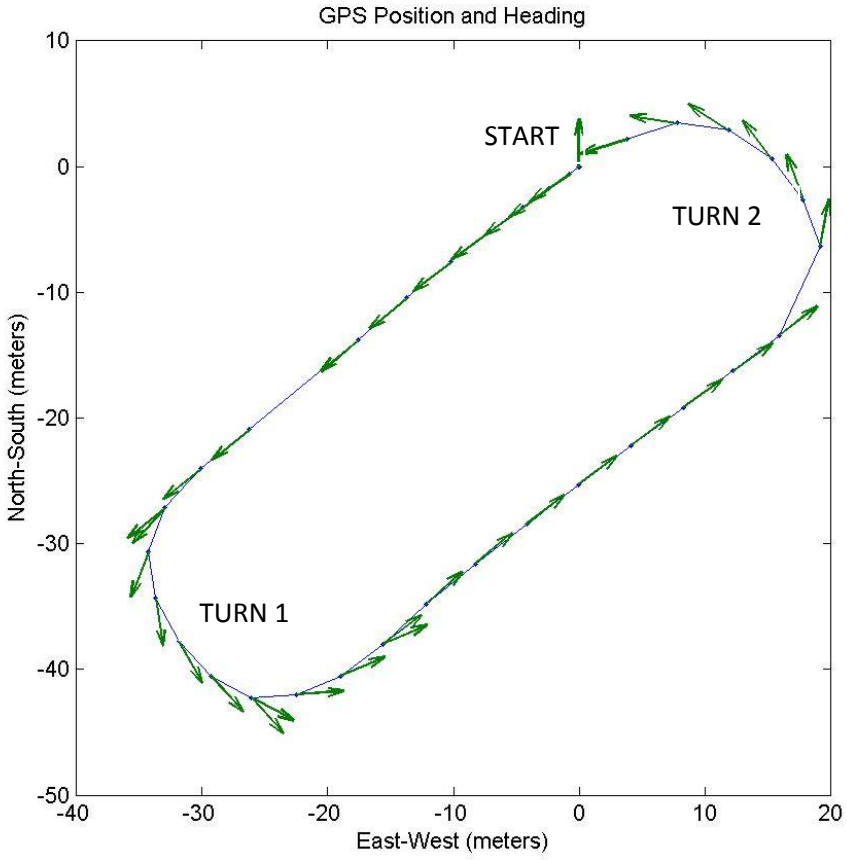


Figure 41: GPS position and heading when driving in a rectangular loop.

MicroStrain INS Performance

A MicroStrain 3GX-I Inertial Navigation System (INS) is incorporated into the sensor suite to complement GPS data. It provides three axis acceleration data and also three axis angular velocity. In addition, the INS will give information on pitch, roll, and yaw. This will allow for better terrain mapping and will increase operational safety by allowing for development of maneuvers aimed at avoiding rolling or other destabilizing the vehicle. The INS is currently mounted on the roof of the cabin.

To test the INS, we drove the vehicle around various parking lots, roads, and wooded paths around Olin College and logged the INS data. This includes roll, pitch, yaw, acceleration in each Cartesian direction, and angular velocity about each Cartesian direction.

To consider the heading given by the INS, we first looked at the yaw measurements. However, the yaw reading is very poor. The following graph displays the INS yaw reading when driving twice around a rectangular loop in a parking lot. We believe the problem arises due to vibrations when driving the car. The yaw value seems to be fairly steady when the car is turned off. However, as soon as the engine is turned on, the yaw value varies greatly, even when the vehicle is stationary. One way to approach this problem is to consider placing the INS at a different location on the vehicle.

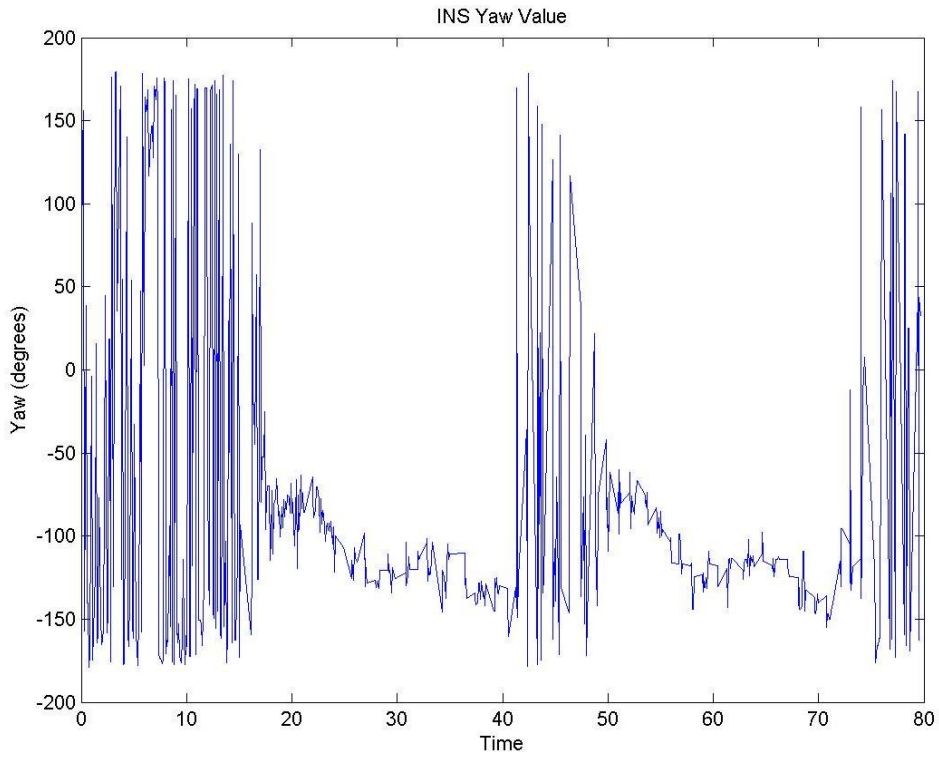


Figure 42: INS Yaw data.

As another way to determine heading, we can integrate the angular velocity about the z-axis. The following graph displays that data over the same trial as compared to the GPS course over ground data. The offset is mainly due to the fact that the INS heading, after being integrated, does not have the correct initial value.

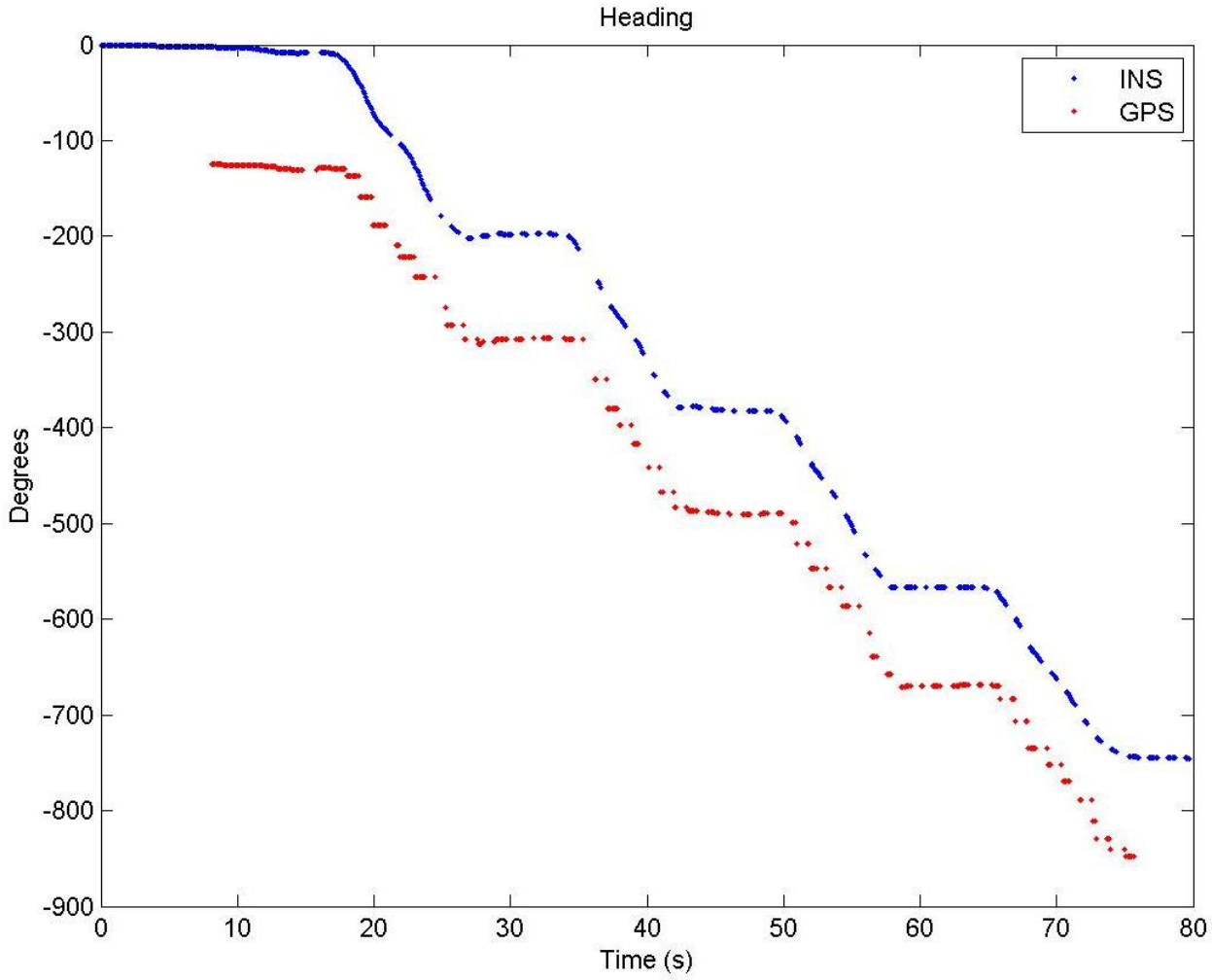


Figure 43: Heading as determined by integrating z-axis angular velocity.

To get a sense of how well integrating z-axis velocity works over time against factors such as accumulating error through drift, we can plot the difference between that heading and the heading given by the GPS. This graph is displayed below.

We expect a constant error due to the fact that the integration of the z-axis angular velocity does not start at the correct initial value. However, we also see spikes of error whenever we are turning. This demonstrates the delay in the GPS heading as discussed above. Because the INS provides an almost immediate response to any changes in heading while the GPS heading needs some time to change, we see large changes during each turn.

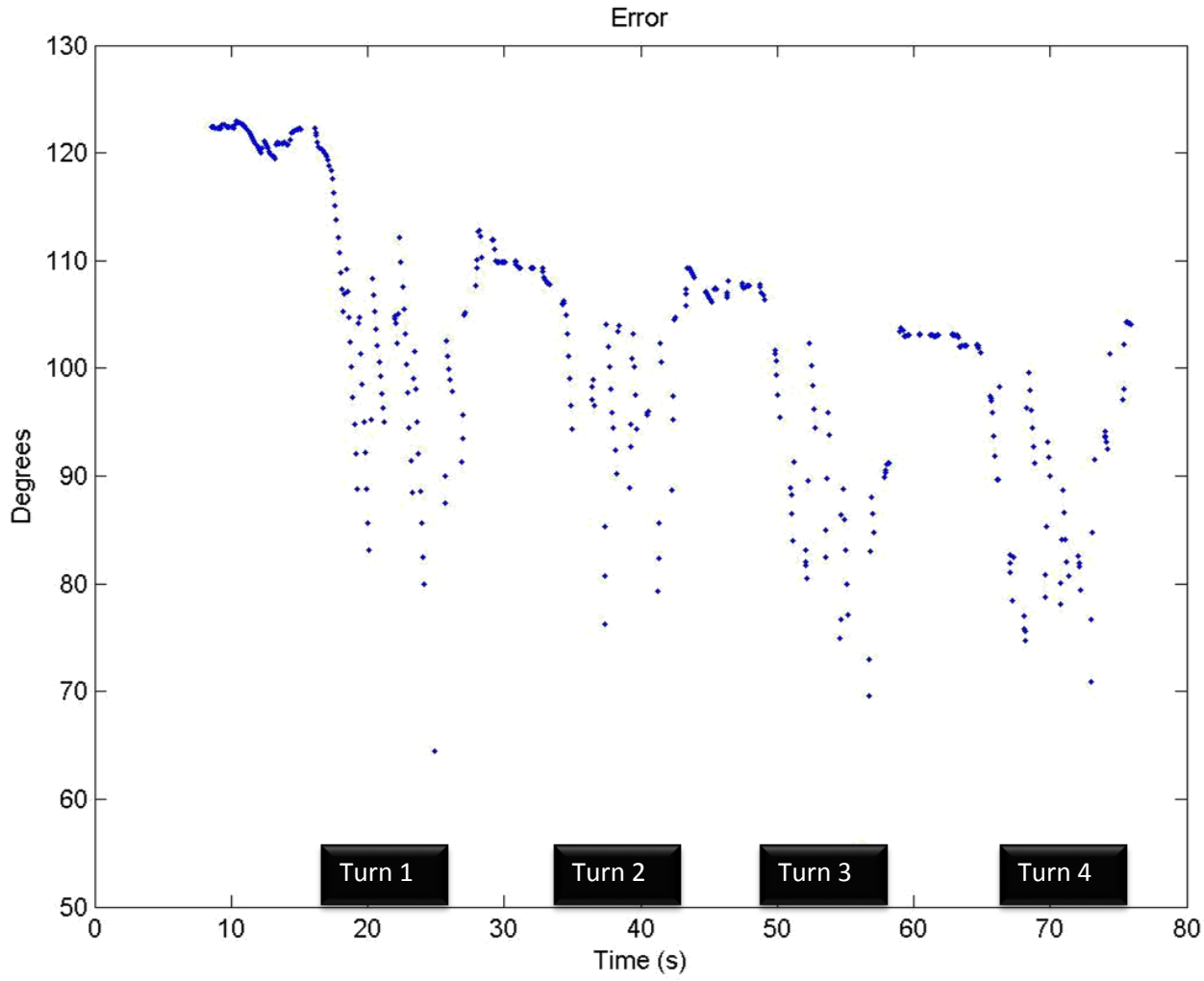


Figure 44: Difference between GPS heading and integral of z-axis angular acceleration.

To account for this, we shift the GPS heading by one second. This rough estimate alleviates some of the problem. While we can still see effects of turning, we can now see that the integration method provides reasonable results over short periods of time, but is subject to long term drift.

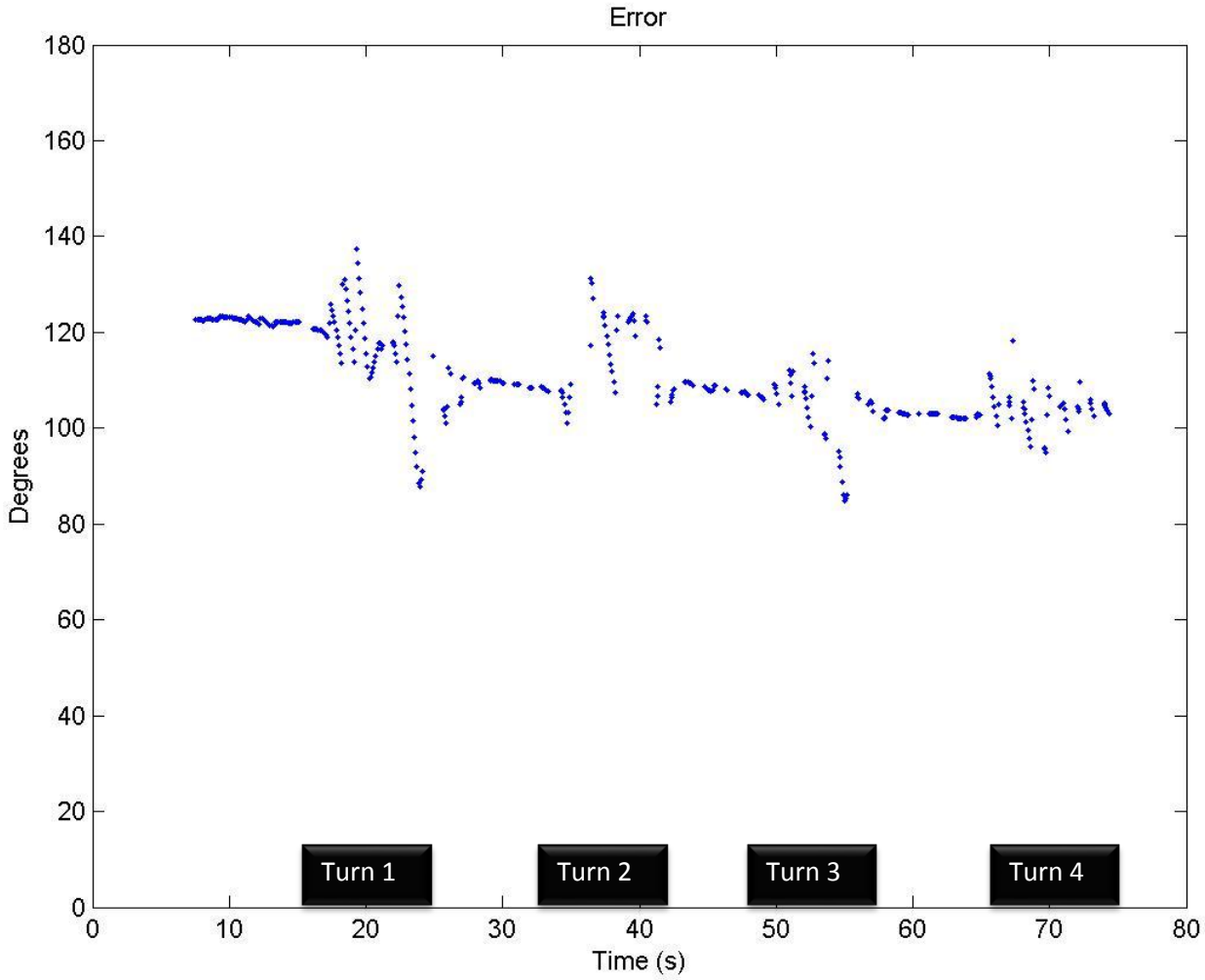


Figure 45: Results with time shifted GPS heading.

Here are the same two graphs for a much longer run time. As can be seen, there is a drift of roughly forty degrees over a period of two and a half minutes. The period of time where the GPS heading is constant at 360 degrees corresponds to GPS loss.

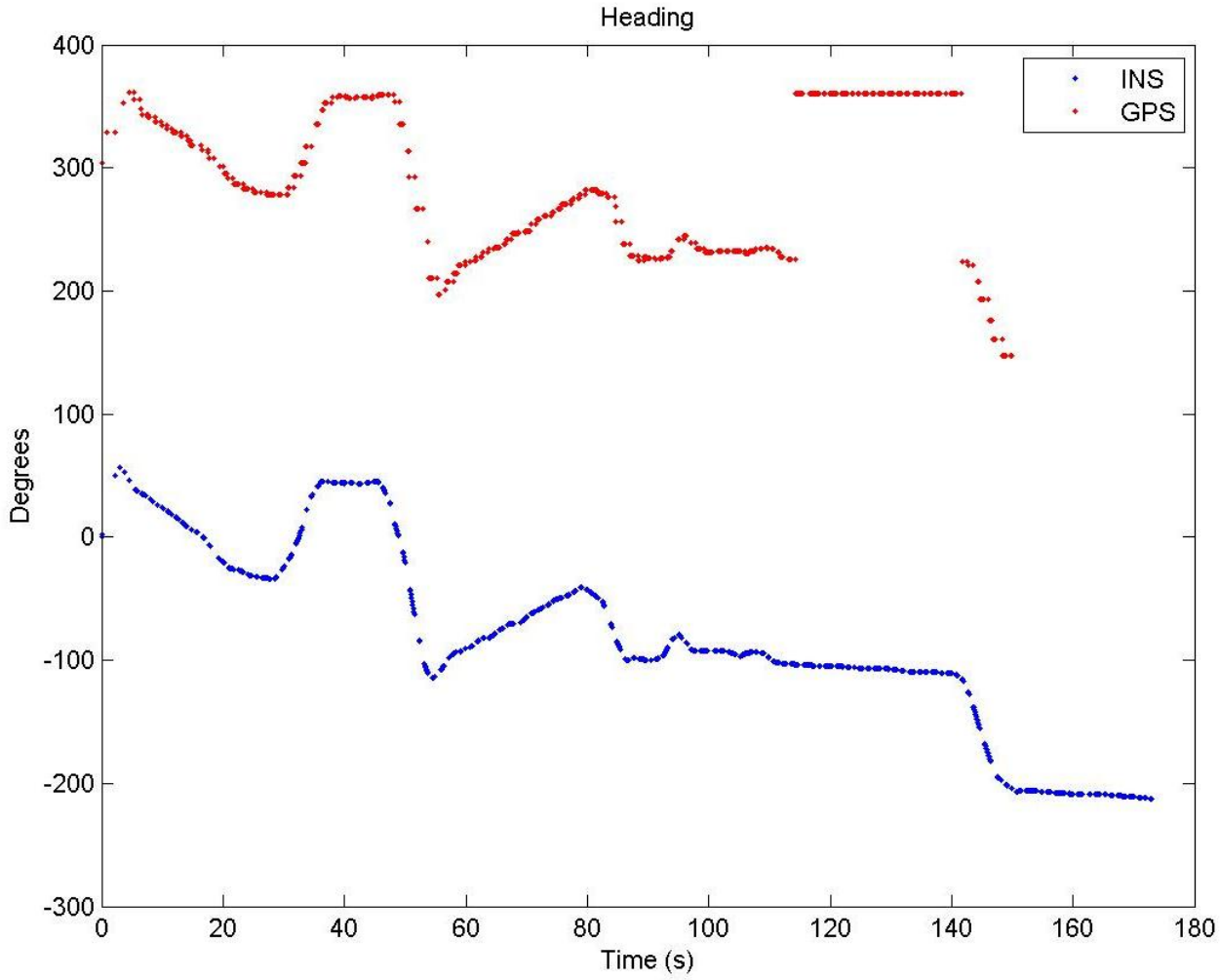


Figure 46: GPS Heading vs. Integration of INS Z-Axis angular acceleration

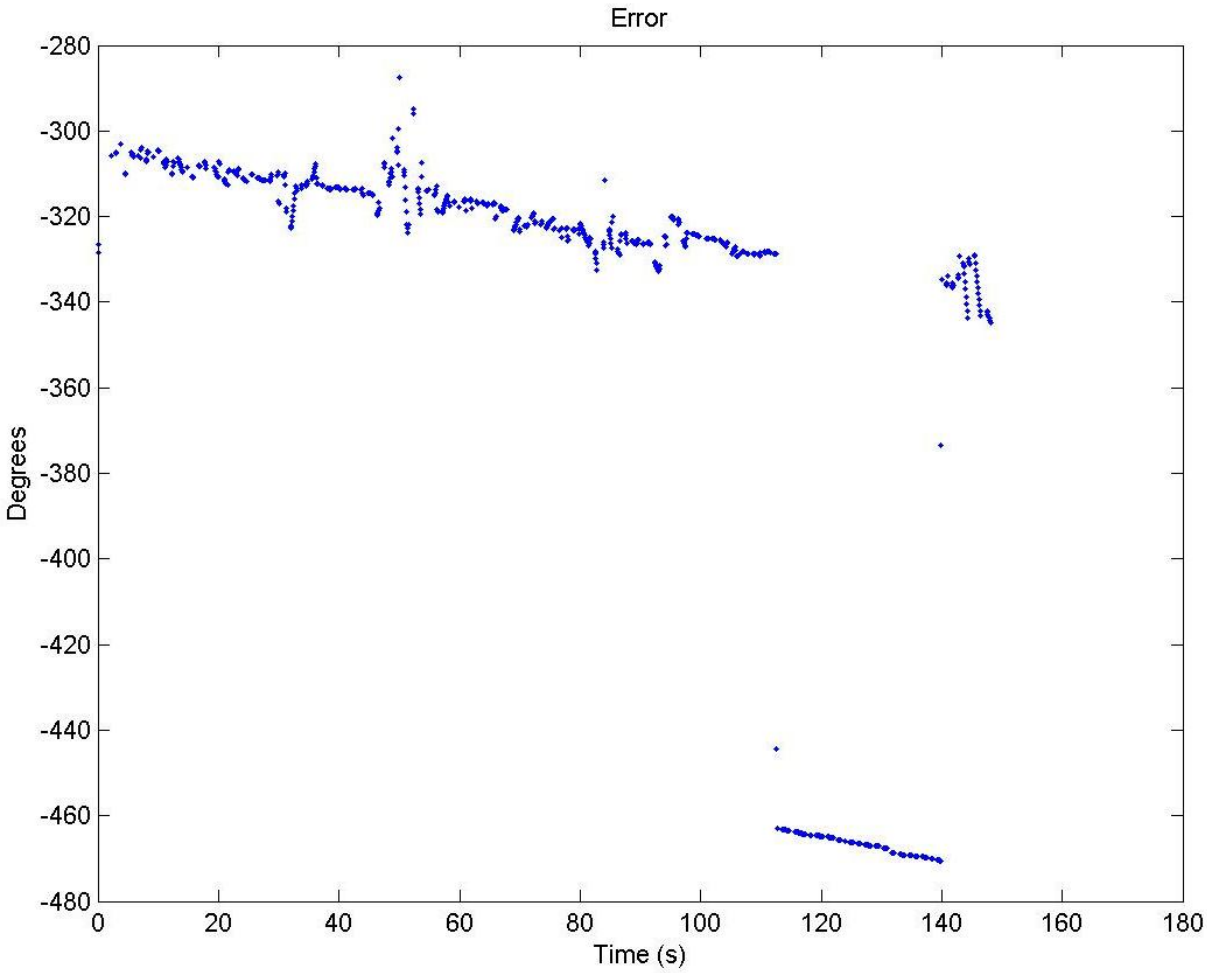


Figure 14: Error between GPS heading (time shifted) and integration of INS z-axis angular acceleration.

Using our INS Cartesian accelerations, we can calculate several measurements, such as forward velocity, side to side acceleration (which would indicate turning). The following results are for the test in which we drove twice around a rectangular loop in a parking lot.

To calculate the scale, we took the average of the z-axis acceleration and set it equal to gravity. However, when we then look at the resulting z velocity, we see speeds of over five meters per second (which is faster than the car is driving forwards). Clearly, this data is too noisy to be useful. Likewise, the forwards velocity of the car (calculated by integrating the y-axis acceleration from the INS) is too large after eighty seconds, as the vehicle is limited to a much lower speed. However, in the side to side acceleration, we can see some useful data. There are four small bumps that correspond to each turn made in the parking lot. All in all, the Cartesian accelerations from the INS have the potential to be useable, but they accumulate significant error when integrated to find velocity.

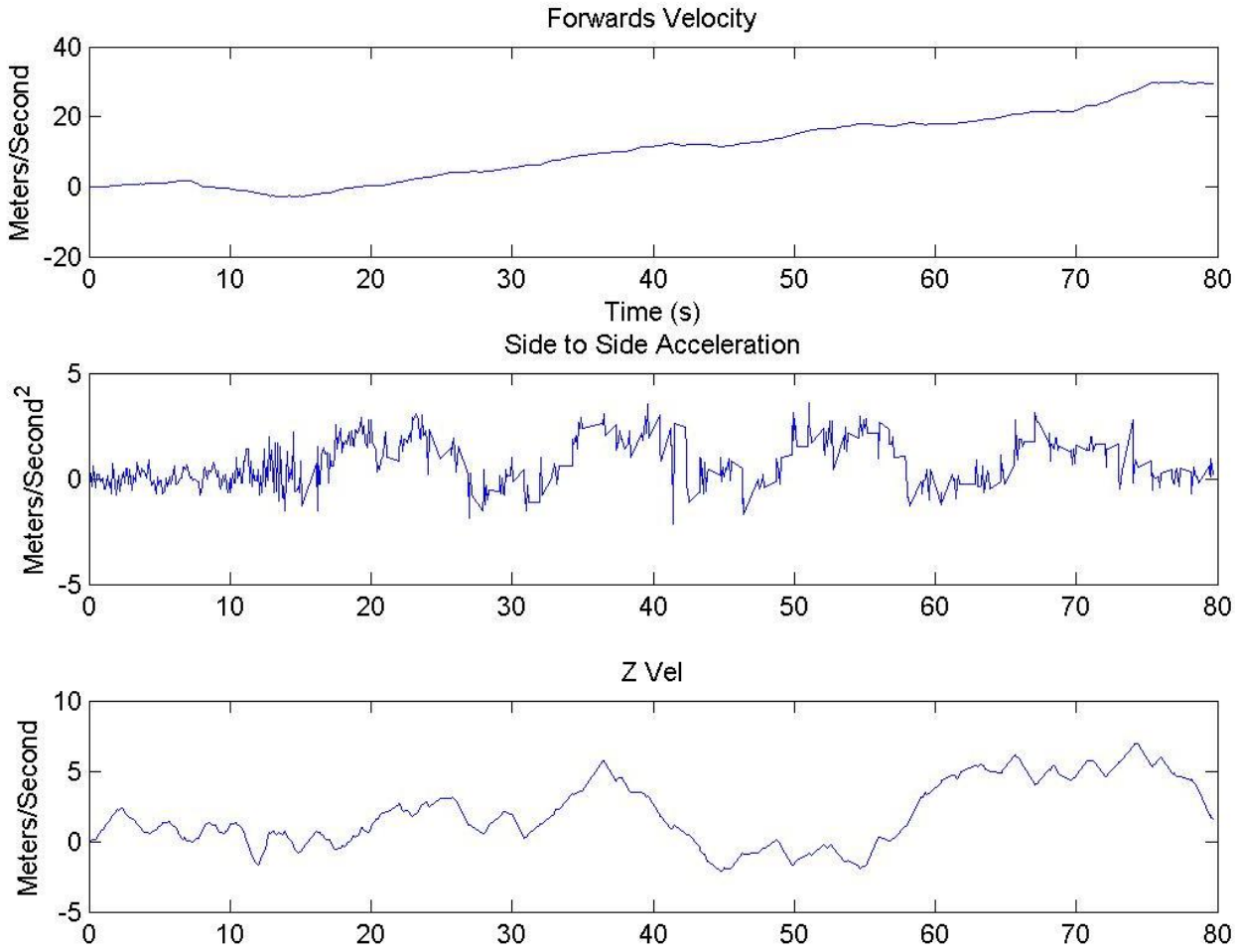


Figure 47: Data obtained using the Cartesian acceleration data from the INS.

Similar bumps appear in the INS roll data. Again, this is indicative to each time we turn. However, the roll angle is extremely noisy as in the case of the yaw angle data. This is likely due to the vehicle vibration as explained above.

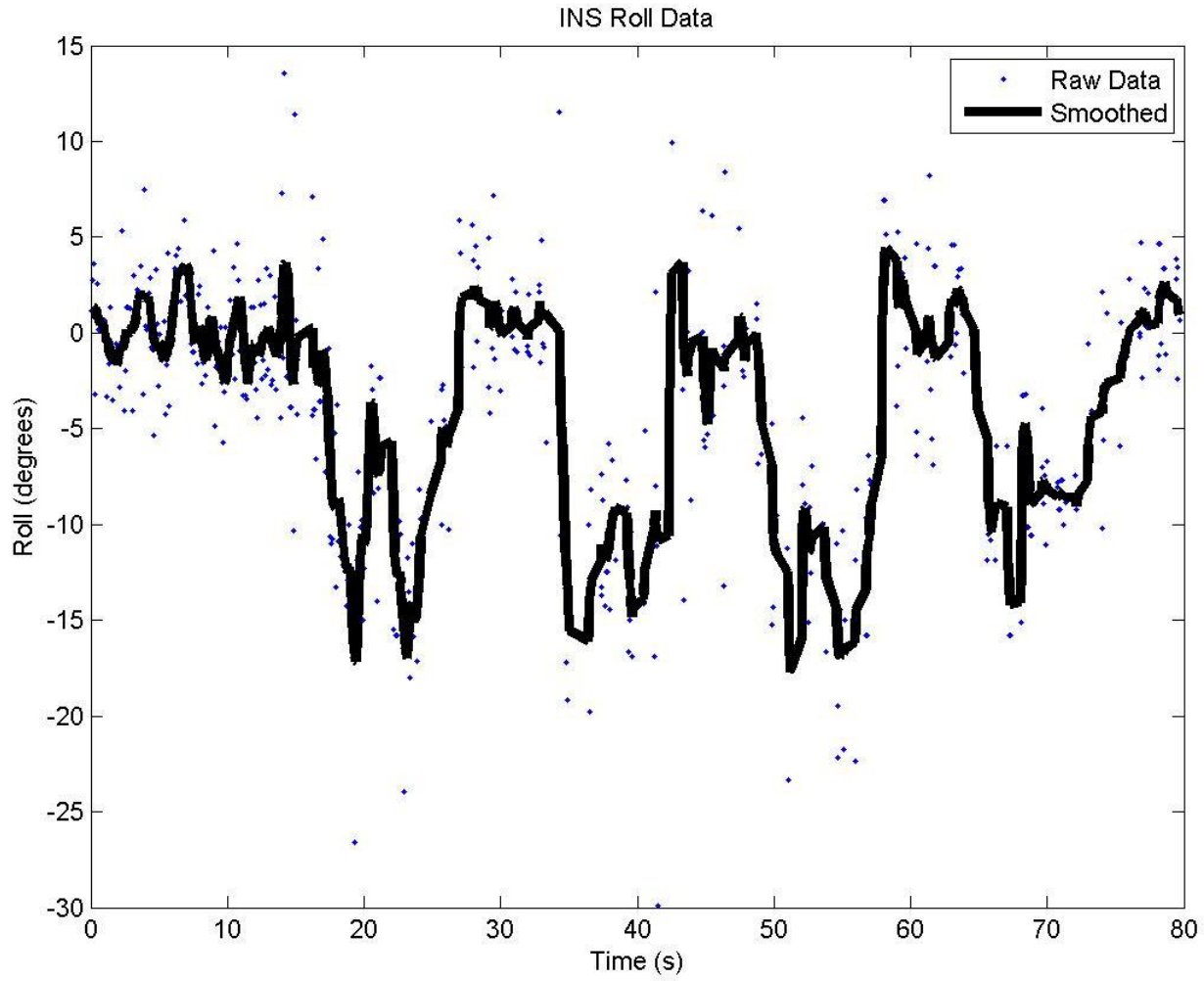


Figure 48: The INS roll data. The thick black line is the same data processed through a low pass filter.

In conclusion, the INS can give us usable data. Integrating accelerations to find velocity will not work, but other methods of using the INS can prove successful.

LIDAR Performance

To test our dual LIDAR scheme, we recorded the LIDAR sensor data in the wooded paths and fields. We mounted a digital camera on the front of the vehicle and took video of the terrain. To evaluate the performance of the dual LIDARs, we use the playback feature while running the recorded video and simply compare the two. The graphs of the LIDAR account for the 45^{O} outward shift of each LIDAR, but not the 10^{O} downwards shift. The graphs are therefore somewhat incorrect, but they do display the right pattern.

The following are from driving in a path marked by trees and bushes on either side. Each LIDAR can clearly make out the tree line on its side, as well as distant tree line on the far side.

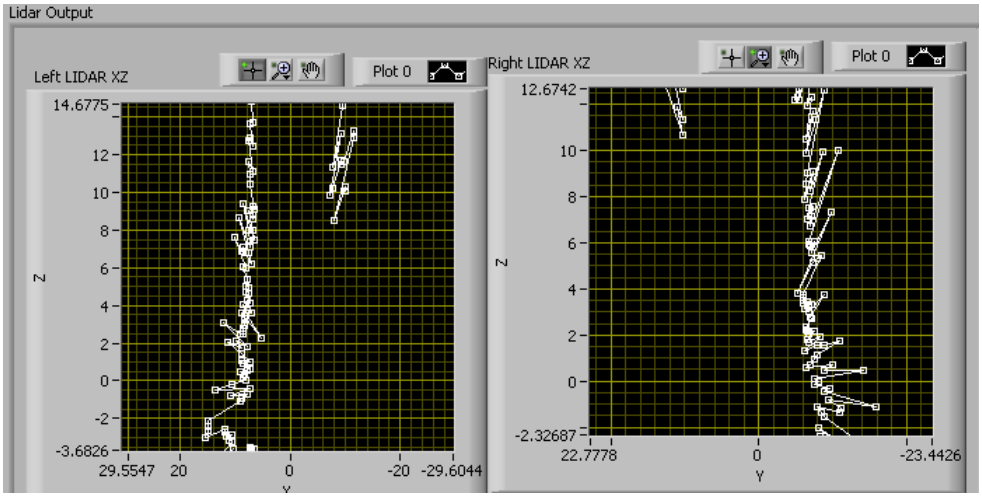


Figure 49: Driving through a wooded path.

The following are from driving in an open field. The left LIDAR can see some of the bushes and hills towards the left. The right LIDAR mostly sees no obstacles.

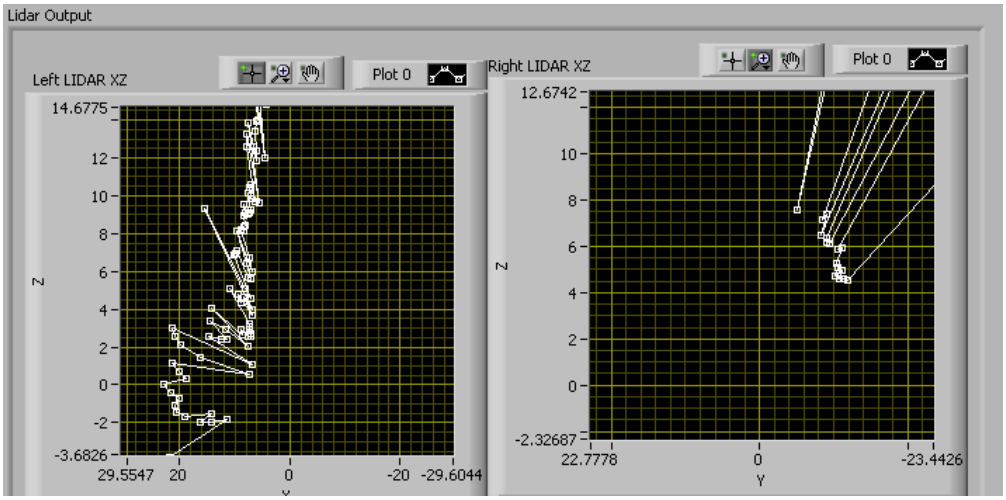


Figure 50: Driving in an open field.

As evidenced by these examples, as well as the rest of the data, the dual LIDAR scheme should work for local navigation. Extensive tests will be done with targets at known distances to quantify LIDAR performance in the next phase of this project.

Appendices

Appendix A: Testing, Startup and Shutdown Procedures

Startup Procedure

1. Always drive with a partner. Do not attempt to drive and use the computer at the same time. Designate one person as the vehicle driver and the other as the software engineer.
2. Make sure the ESTOPs are on.
3. Boot computer and load LabVIEW code, if necessary.
4. Check for sufficient fuel. Switch power from the building supply to the generator. Disconnect all power and network cables attached to the building.
5. Close electronics box and rear hatch.
6. Turn the vehicle on (switch ignition halfway, wait for lights to turn off, start engine).
7. Turn on lights and heating as needed. Note that drawing too much power (i.e. both the heating unit and the floodlights) will blow the fuse.
8. Straighten wheels and run LabVIEW code.

Testing Procedure

1. Turn the back ESTOP off. Make sure the ESTOP in the cabin is on.
2. Load the mission in LabVIEW and put in autonomous mode.
3. Make sure that the vehicle heading is correct. Often if you have backed up to the starting location, the heading will be the opposite of what it should be. This can cause the vehicle to be confused and turn wildly. To fix this problem, simply drive forward before starting.
4. Once the software engineer has indicated the test is ready to be run, make sure the gear is in forward gear and disengage the parking brake. When ready, turn the cabin ESTOP off.
5. As the vehicle driver, your hand should always be by the cabin ESTOP. Do not attempt to look at the computer screen. The vehicle driver's eyes should be on the road. The software engineer may use the computer during this time.
6. When test is completed, turn the cabin ESTOP on. Stop the LabVIEW code, if necessary.

Shutdown Procedure

1. Turn off the engine. Make sure to engage the parking brake and turn off any lights/heating/fans.
2. Turn off the generator and switch the power from the generator to building power. The UPS will allow you to do this without turning off the computer. Connect network cables if necessary.
3. If completely shutting down the system, turn off the computer and turn off the power strip in the electronics box. Make sure to reset the UPS, or otherwise it will try to keep power available.

Appendix B: Vendor Contact Information

The following vendors have been helpful in providing products and support to the team.

National Instruments

National Instruments (NI) has provided us with significant software and hardware support at little to no cost to the team. They are also extremely efficient at providing good technical support. All of the software and some of the hardware onboard Ella comes from NI. Our team's main contact at NI is Lesley Yu (lesley.yu@ni.com).

Olin College has a great relationship with NI, and Prof. David Barrett can provide other contacts there if Lesley cannot be reached.

Advanced Motion Control

Advanced Motion Control (A-M-C) is also providing some hardware sponsorship to the team. They provided us with a brand new servo amplifier for the steering motor at an educational discount. They are very responsive to both phone and e-mail messages.

Karl Meier (e-mail: kmeier@a-m-c.com, phone: 805.389.1935) is the person at A-M-C in charge of university relations. For future purchases at A-M-C, he is the person to contact in regards to an educational discount. Matt Cole (e-mail: mcole@a-m-c.com, phone: 805.389.1935) is in charge of product returns and repairs. He expedited the process on a motor controller that we broke and sent in for repair (though it was ultimately deemed 'broken-beyond-repair').

Potomac Electric

Potomac Electric built the DC servo motor used to actuate the steering wheel. We talked to the VP of Engineering, Leny Chertov (e-mail: lenyc@potomacelectric.com, phone: 617.364.0400) about the electrical specifications of the motor when purchasing a new servo amplifier.

Padula Brothers

Padula Brothers is the closest authorized John Deere dealer. The gator was originally purchased from them, and so they have F. W. Olin College of Engineering on file. We ordered a new light and new wheels from them this year, and they have provided good service to us. Their phone number is (978) 537-3356.

Appendix C: Sensor Settings

GPS

The GPS can be configured using StarUtil, a utility program from NavCom. StarUtil is apparently not available for download from their website, but it is already installed on the Gator, and it can be copied onto other computers for GPS testing purposes.

Currently, the GPS is configured to communicate at 19200 baud and is normally connected to COM1.

StarUtil allows you to configure a number of different GPS settings, including speed and data output format. To increase the rate that new navigation solutions are generated (i.e. new latitude, longitude, altitude information is created), change the LSKJDFLKSDJFLKSJD FLKSDJ FLKSD JF in the SLDFJLSDKJFLSDKF menu. We currently have the GPS configured to return positions updates at 5 Hz.

Because of difficulties with LabVIEW's drivers for the NAVCOM, we are currently returning data using NMEA strings, rather than the NCT binary data format. The NMEA strings can be configured using the SKDFJLSKDJFLSKDJF SLDK F menu. We currently have drivers to parse GGA strings (information about location and number of available satellites) and VTG strings (information about current heading), so ensure that both are being output.

INS

The Microstrain INS is configured to communicate at 38400 baud and is normally connected to COM2.

The Microstrain will return a few different types of data. It can return magnetometer data, accelerometer data, angular rate data, and an estimate of pose that combines all three sensors. The pose estimate can either be in Euler angles or quaternions. Each of these types of data can either be returned as raw measurements (direct readings of the ADC inputs from the sensors), calibrated instantaneous data, or calibrated gyro-stabilized data (data filtered using input from the gyroscope). From our testing, we have discovered that due to the vibrations of the vehicle from the generator and the engine, the instantaneous pose data is useless. The gyro-stabilized data is also useless because of a large time lag in the estimate. This could perhaps be fixed by changing settings on the INS, but we did not have a chance to experiment.

The magnetometer data is pretty stable, even with the vibration of the vehicle. Unfortunately, due to the large amounts of ferrous metal on vehicle, it needs to be calibrated. We produced a rough calibration by recording the heading reading from the magnetometer and the heading reading from the GPS for a number of points. However, this calibration is only accurate to within 5-10 degrees, and so it ideally should to be re-calibrated before use.

LIDAR

The LIDARs are both configured to operate at 500 kbaud on startup. We do not have a hardware RS-422 port in the lab that can communicate at this data rate (since it is non-standard), and so

the LIDARs can only communicate with our FPGA driver. There is a LIDAR Utility in the code which allows arbitrary telegrams to be sent to the LIDAR to change settings. On startup of the FPGA code, a message is sent to put the LIDARs in continuous measurement mode, which means that they constantly stream back data.

The LIDARs are currently configured to scan 180° in 1° increments, returning data with millimeter precision. These values can be adjusted by sending the appropriate telegrams. These can be found in the Telegram Manual. Note that the checksums listed in the manual are sometimes incorrect, and so a new CRC checksum frequently needs to be calculated when sending a new message. There is a program in the LIDAR folder which computes these checksums.

Appendix D: SVN Set Up

Setting Up Your Environment

Draper Team uses TortoiseSVN in conjunction with LabVIEW's Merge and Diff tools for conflict editing. This section details how to install TortiseSVN and then link it to these tools.

Installing TortoiseSVN

Download either 32 or 64 bit Tortoise from their website's download page and follow their instructions.

Linking LabVIEW's Merge and Diff Tools

To link TortoiseSVN to LabVIEW's Merge tool, follow the instructions on Tomi Maila's blog. This blog also shows how to use the Merge too to resolve conflicts. Note: make sure that the path used in the "External Program" field actually points to LVMerge.exe. In particular, if you're running 64 bit LabVIEW, it's likely to be in a different location.

To link TortoiseSVN to LabVIEW's Diff tool, repeat Tomi Maila's blog instructions with two exceptions. Instead of selecting "Merge Tool" from the left tree menu, select "Diff Tool." Then, in the "External Program" field, put in the path "C:\Program Files\National Instruments\Shared\LabVIEW Compare\LVCompare.exe" Again, keep in mind that your path may be slightly different.

Checking Out the Repository

1. Navigate to the folder you want to check code out to.
2. Right click and select "SVN Checkout…" (if this does not appear, you have not properly installed TortoiseSVN).
3. Enter http://acl.olin.edu/svn/scope10-draper/EllaProd into the URL field.
4. After pressing OK, enter your credentials.

SVN Standards

The team uses a few useful conventions to make try to avoid conflicts:

1. Don't use virtual folders. Instead, all folders should be auto-populating folders, which copy the file system's structure. To add an auto-populating folder to the project, make sure it exists in the file system. Then right click in the project tree and select Add>Folder(Auto-populating)… All nested folders and VI's will be added to the project.
2. All VI's should be in an auto-populating folder.
3. If you're currently working on a VI, rename the VI as <YourName><VIName>.vi. Once you think the VI is finished and debugged, remove your name.
4. Don't edit VI's that have a team member's name in them before asking for permission.
5. When committing, always attach useful comments. These should include what changes were made and what state of robustness those changes are in (first draft of code, in debug, debugged).

Appendix E: FPGA Inputs

This appendix outlines the function of each pin of the I/O FPGA modules.

NI9401

Pin	Color	Function
1	Black	Module ground
2		
3	Black	Ground to CommFronts
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17	Orange	Raphael
18		
19	Green	Leonardo
20	Yellow	Raphael
21		
22	Blue	Leonardo
23	Red	Steering end stop +
24		
25	Black	Steering end stop -

NI9411

Pin	Color	Function
1	White	Steering Encoder A+
2	Green	Steering Encoder B+
3	Yellow2	Rear wheel encoder A
4		
5		
6	Orange	Unused
7		
8		
9	YELLOW	Steering Encoder A-
10	PURPLE	Steering Encoder B-
11	Purple2	Rear wheel encoder B
12	Black	Shield GND
13	Red	Steering Encoder VDD
14		

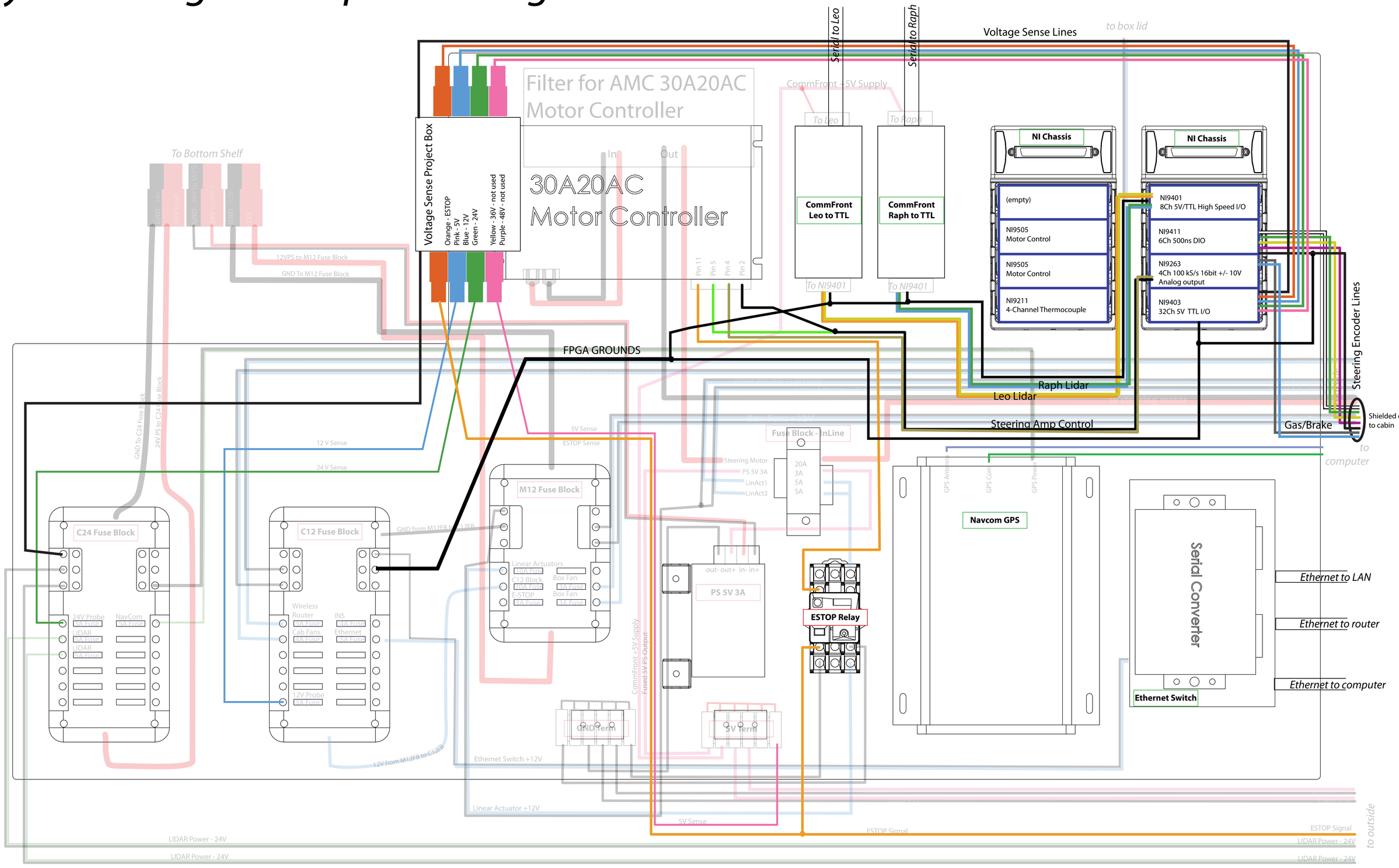
NI9263

Pin	Color	Function
0	Blue	Gas Actuator Signal
1	Black	Ground (same as FPGA ground)
2	Gray	Brake Actuator Signal
3	Black	Ground (same as FPGA ground)
4	White	Steering Servo Drive control signal
5	Black	Ground
6		
7		
8		
9		

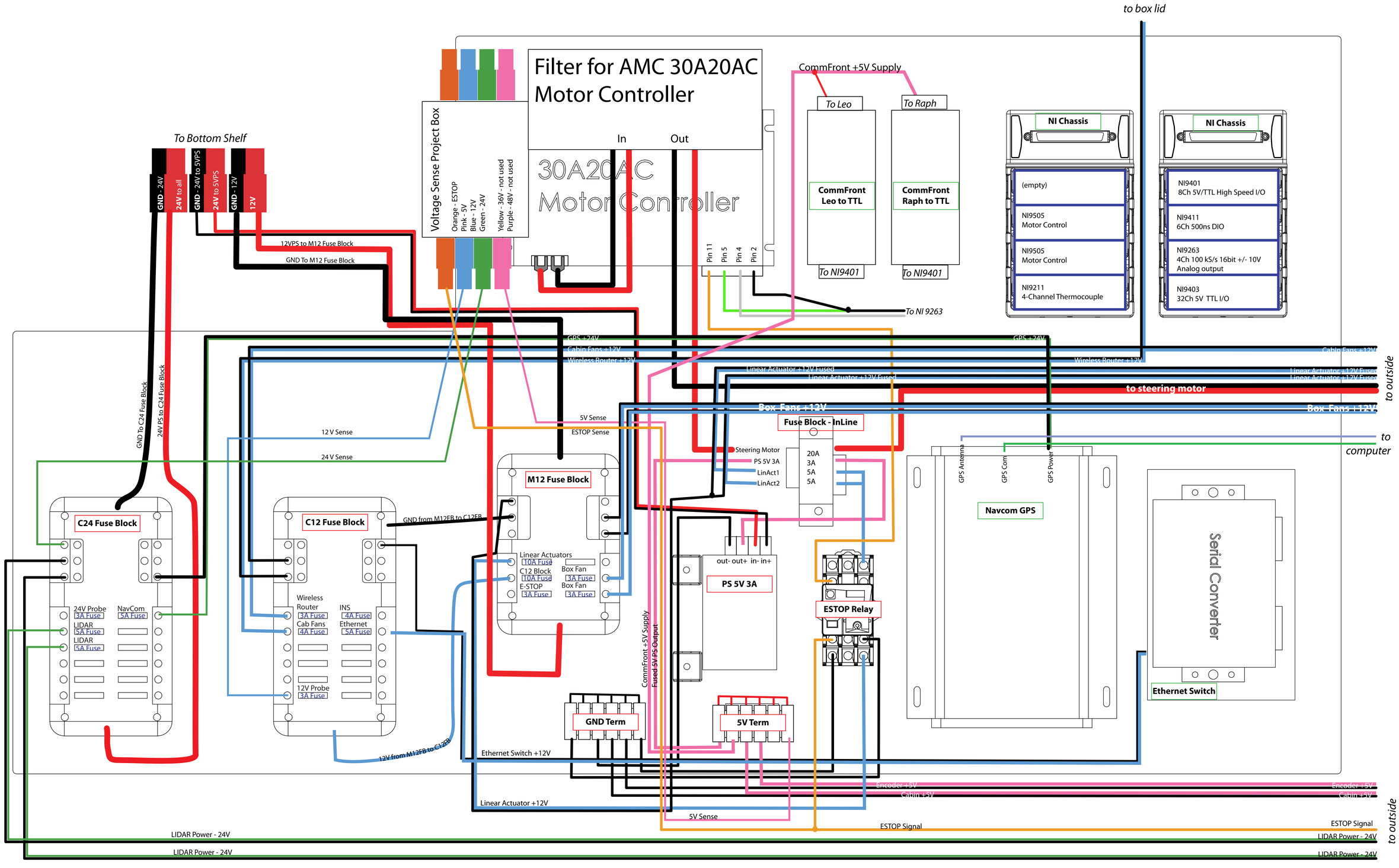
NI9403

Pin	Color	Function
1	Orange	E-STOP Sense
2	White	5V Sense
3	Blue	12V Sense
4	Green	24V Sense
5		
6	Purple	48V Sense
7	Yellow	36V Sense
8		
9		
10	Black	Ground

System Diagram: Top Shelf - Signals and Data



System Diagram: Top Shelf - Power Distribution



Appendix G: Code Walkthrough

PCMain-Rev2.vi

This is the robot's top level VI. Running this VI will boot all code needed to run the robot.

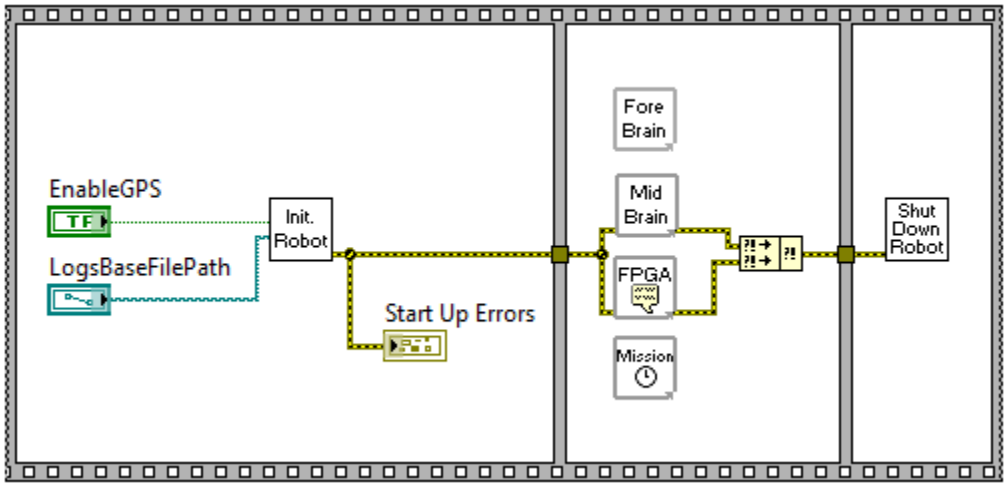


Figure 51: The block diagram of PCMain-Rev2.vi - the robot's top level VI.

Icon	Name	Description
Init. Robot	InitializeRobot.vi	Initializes global variable values, opens required serial ports, and sets FPGA control value before running the FPGA (Hindbrain). Also initializes logging.
Fore Brain	Forebrain.vi	Runs the forebrain code. Allows the definition and execution of the Mission Definition File. Determines the vehicle's desired heading.
Mid Brain	Midbrain.vi	Runs the midbrain code. Executes reactive robot behaviors and arbitrates the vehicle speed and velocity.
FPGA	MaintainFPGACommunication.vi	Updates global variables written on the FPGA. Maintains all DMA transfers and FPGA front panel controls.
Mission	MaintainMissionClock.vi	Updates the global variable that tracks mission duration (in EllaMasterControls).
Shut Down Robot	ShutDownRobot.vi	Turns the vehicle velocity to 0 and shuts down the FPGA and all serial ports. Shuts down and saves all logs.

Forebrain.vi

This VI runs all of the robot's mission planning and execution. It consists of three main sections: mission planning and execution, status sensing and user interface.

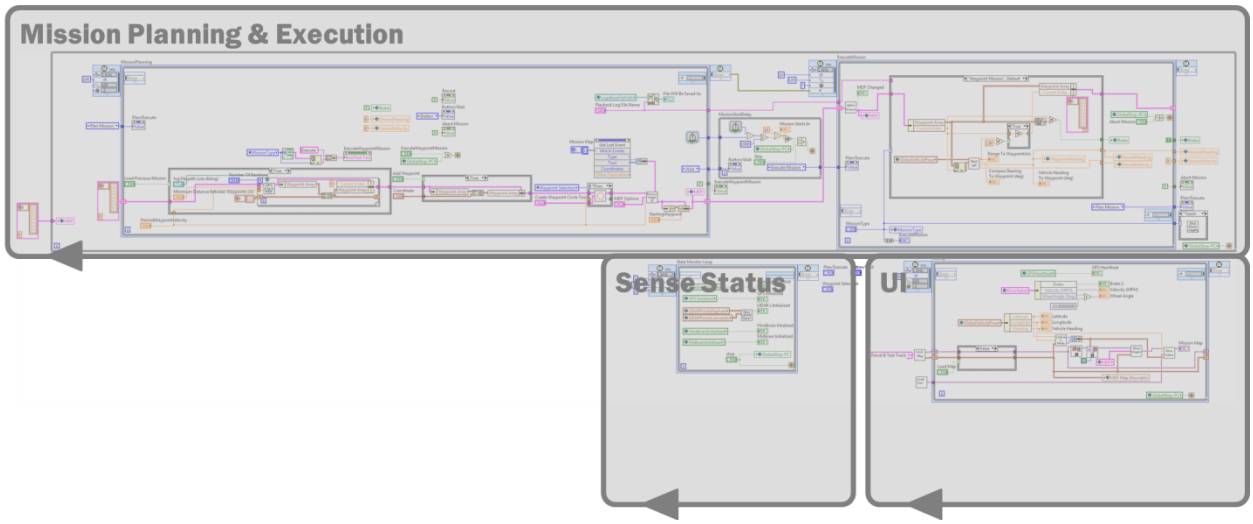


Figure 52: An annotated version of the Forebrain.vi code.

Mission Planning and Execution

This loop is used to plan and execute the mission. It is subdivided into three different loops: mission planning, mission start delay, and mission execution.

Mission Planning

This loop allows the user to plan a mission. The user can plan a waypoint mission by clicking on locations on a map, manually entering waypoints, or using the waypoint circle creation tool. They can also execute a teach mission, which records a series of waypoints for later use in a playback mission, which loads a previously taught mission. Finally, they can run a manual mission in which desired velocity and wheel angle are directly specified by the user

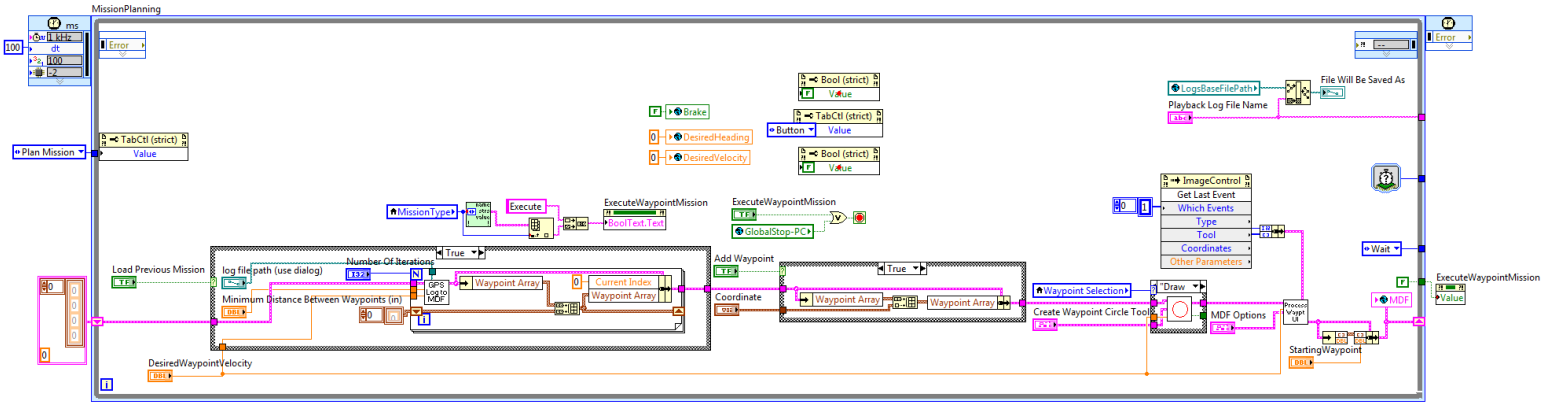


Figure 53: The loop which handles mission planning.

Icon	Name	Description
	CirclePaths.vi	Produces a set of waypoints in concentric circles and adds them to the MDF.
	ConvertGPSLogToMDF.vi	Loads the specified GPS log and converts it into an MDF with a given spacing between consecutive points.
	ProcessWaypointUI.vi	Processes user interaction with the mission map.

Mission Start Delay

This loop introduces a time delay of 10 seconds between the end of mission planning and the start of mission execution.

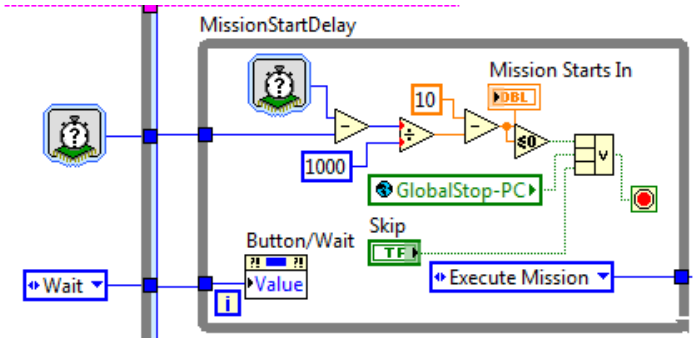


Figure 54: The Loop which handles the mission start delay.

Mission Execution

This loop executes the planned mission. It cycles through waypoints in the specified MDF and generates a desired velocity and heading for the arbiter in the Midbrain.

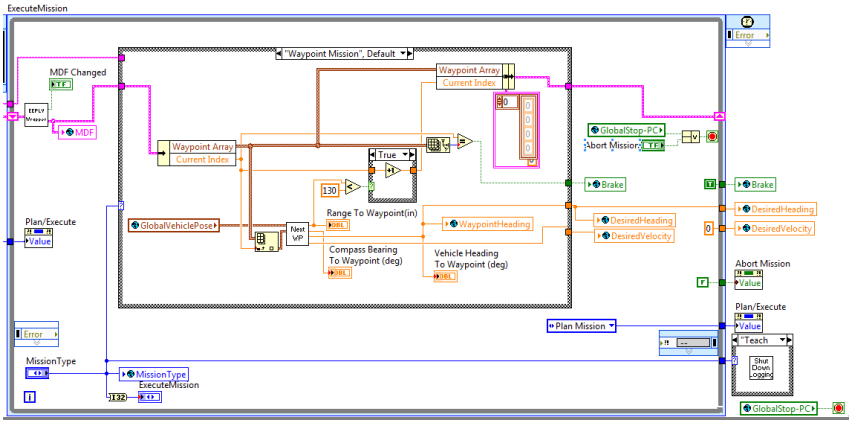


Figure 55: The loop which handles mission execution.

Icon	Name	Description
	Eeplv_wrapper.vi	Interacts with the code developed by SSCI to add voice control capabilities to the vehicle. When a command is received and processed, this VI receives a modified MDF over a network stream and replaces the old MDF.
	NextWaypoint.vi	Calculates the range and absolute and relative headings to the next waypoint.
	PlaybackNextWaypoint.vi	Calculates when to move on to the next waypoint in a playback mission. It looks at the range to the next waypoint and checks to see whether the waypoint has been passed.
	Write To Log.vi	Writes the current vehicle pose to a log file for later playback.
	WriteGPSCoordinatesToCSV.vi	Writes an array of GPS coordinates to a CSV file for portability into other programs.
	ShutDownLogging.vi	Closes the files used to log GPS points in a teach mission.

User Interface

This loop handles the user interface. It displays waypoints on the satellite map.

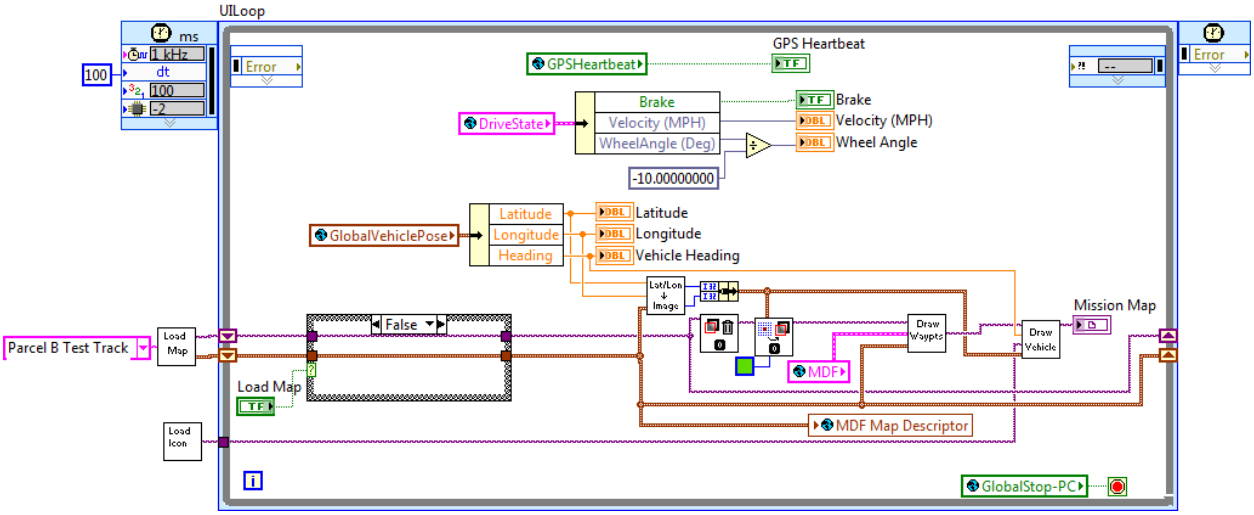


Figure 56: This loop handles the user interface for the Forebrain.vi code.

Icon	Name	Description
	Load Map.vi	Loads an image of the specified map as well as metadata to associate with GPS coordinates.
	Load Vehicle Icon.vi	Loads a car icon to display the vehicle location on the mission map.
	Lat Lon To Image.vi	Converts latitude and longitude to a pixel location on the mission map.
	Draw Waypoints.vi	Draws the waypoints in the MDF onto the mission map.
	Draw Vehicle Icon.vi	Draws the vehicle icon on the mission map.

Status Sensing

This loop displays the vehicles status on the user interface.

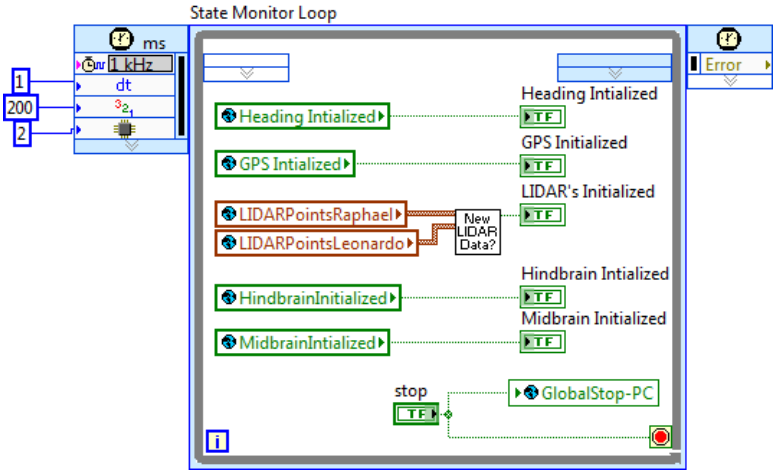


Figure 57: This loop is used to display the vehicle status.

Icon	Name	Description
New LIDAR Data?	NewLidarData.vi	Checks to see whether the LIDARs are outputting new data to ensure that they are functioning properly.

Midbrain.vi

This VI runs all of the robot's reactive behaviors. It is divided into four functions: sense, think, act, and user interface

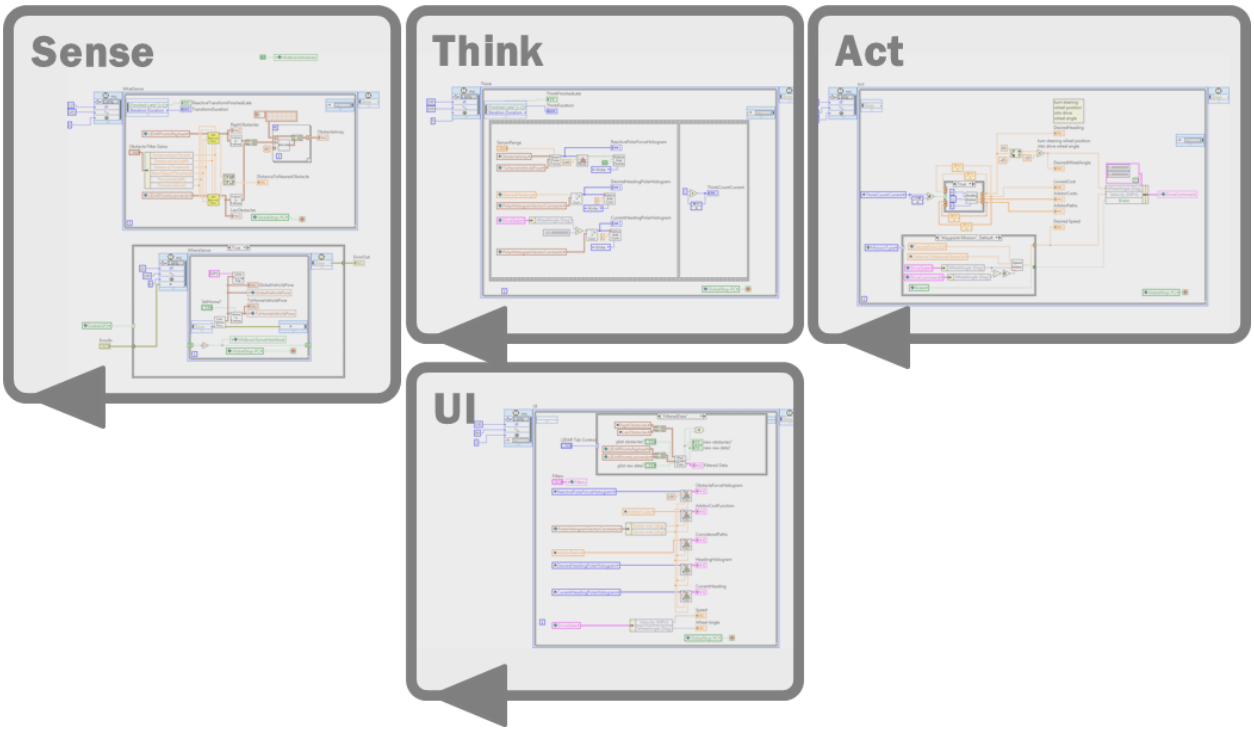


Figure 58: An annotated version of the Midbrain.vi VI. Each loop calls out the function of a section of the Midbrain.vi code.

Sense

There are two primary sensor operations: what the vehicle sees and where the vehicle is.

What Sense

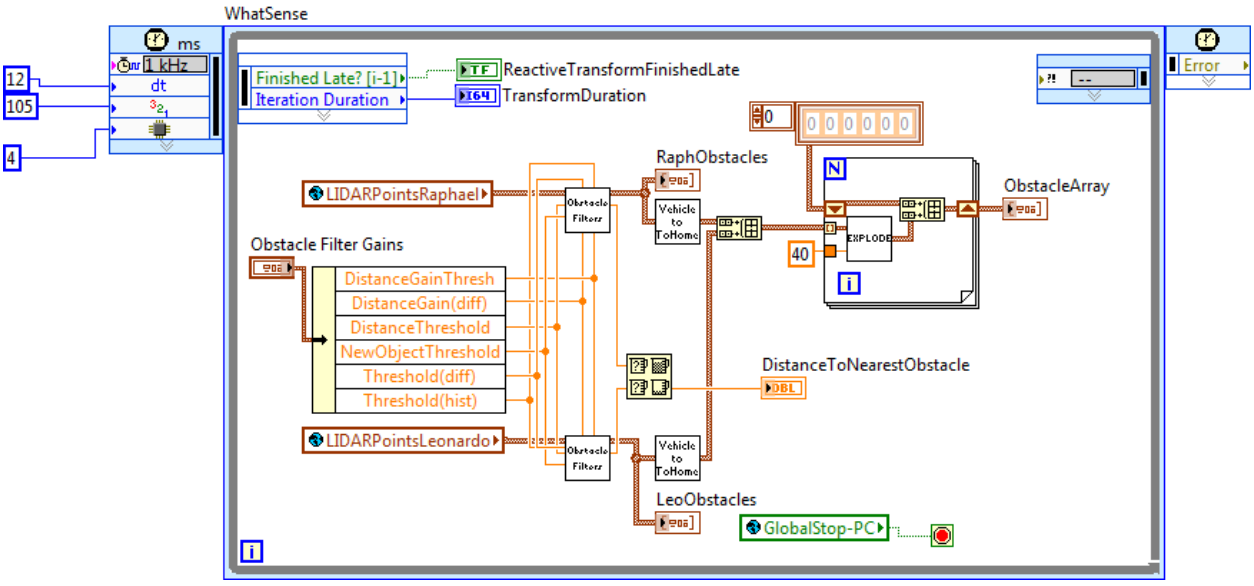


Figure 59: The portion of the block diagram responsible for sensing what is around the robot.

Icon	Name	Description
Obstacle Filterrr	ObstacleFilter.vi	Applies the probabilistic ground filter and the derivative filter to supplied LIDAR points and returns points found to be obstacles by either of the filters.
Vehicle to ToHome	CoordinateTransform--Vehicle-ToHome.vi	Transforms points from a vehicle centered reference frame to the ToHome global reference frame.
EXPLODE	ExplodePoints.vi	For every obstacle point, adds a radius of other obstacle points to account for vehicle diameter in an occupancy grid. This should be replaced with angle enlargement for the VFH+ algorithm. It may still be the proper approach for optimal path planning depending on algorithm.

Where Sense

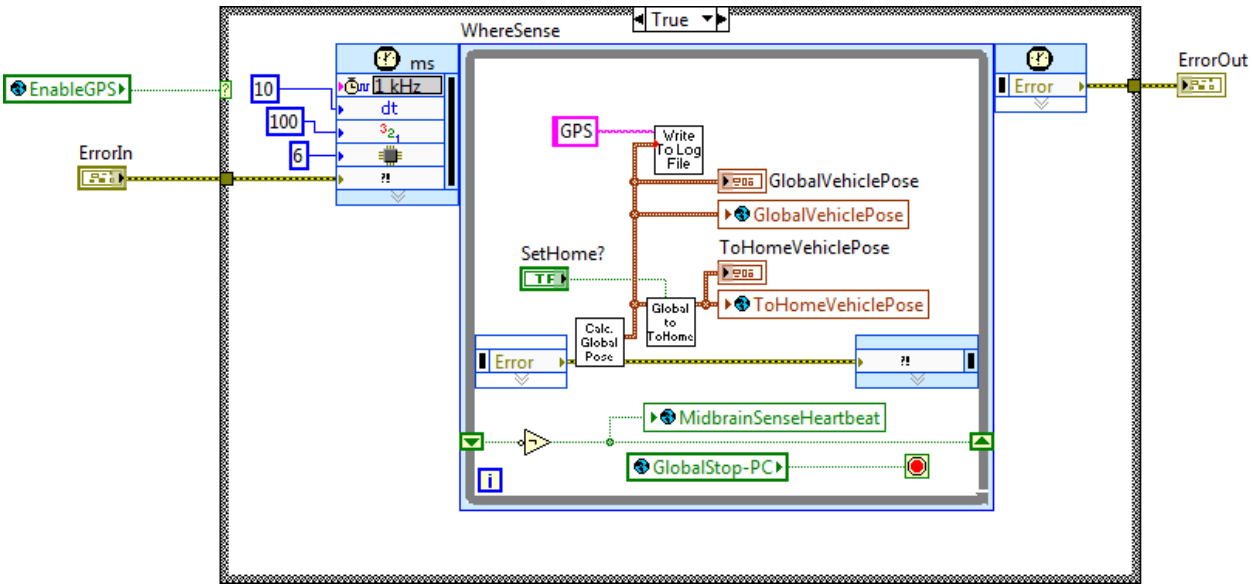


Figure 60: The portion of the Midbrain code responsible for determining robot position.

Icon	Name	Description
Calc. Global Pose	CalculateGlobalPose.vi	Calculates the pose of the vehicle based on GPS and a basic vehicle state estimator.
Global to ToHome	GlobalToToHome.vi	Transforms the global pose to the ToHome coordinate system (defined as inches from mission start).
Write To Log File	WriteToLog.vi	Writes any input to a log file. In this case, logs GPS position over the course of the run.

Think

This loop generates heading proposals that the arbiter selects between.

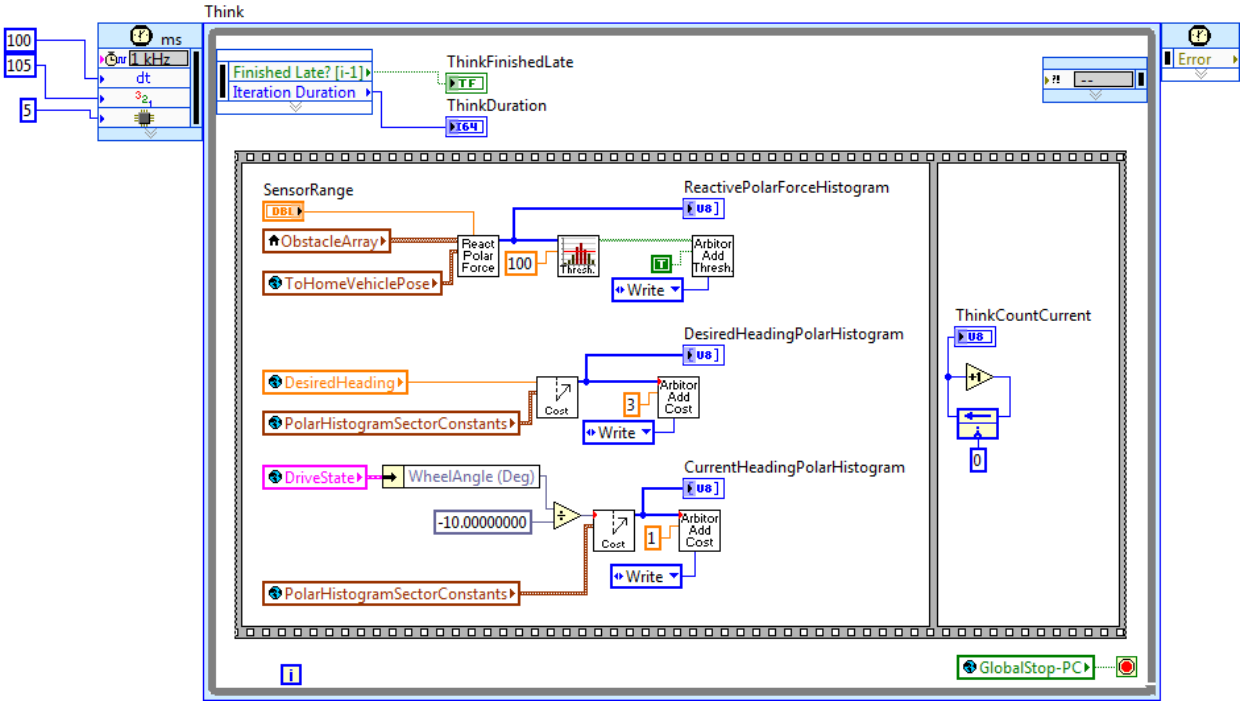


Figure 61: The portion of the Midbrain.vi Code responsible for generating heading proposals.

Icon	Name	Description
React Polar Force	ReactivePolarForce.vi	Calculates the force generated by ever detected LIDAR point in each sector of the polar histogram.
Thresh.	Threshold.vi	Turns a cost polar histogram into a threshold polar histogram. Any sector exceeding the specified threshold cost is marked as undrivable.
Cost	CalculateHeadingCost.vi	Creates a cost polar histogram with a minimum cost at the specified heading.
Arbitor Add Thresh.	AddCostToArbiter.vi	Adds a cost proposal to the set of proposals to be considered by the arbiter.
Arbiter Add Thresh.	AddThresholdToArbiter.vi	Adds a threshold proposal to the set of proposals to be considered by the arbiter.

Act

This loop selects the proper heading from blending heading proposals as well as selecting the proper velocity for the vehicle. These commands are then passed to the hindbrain.

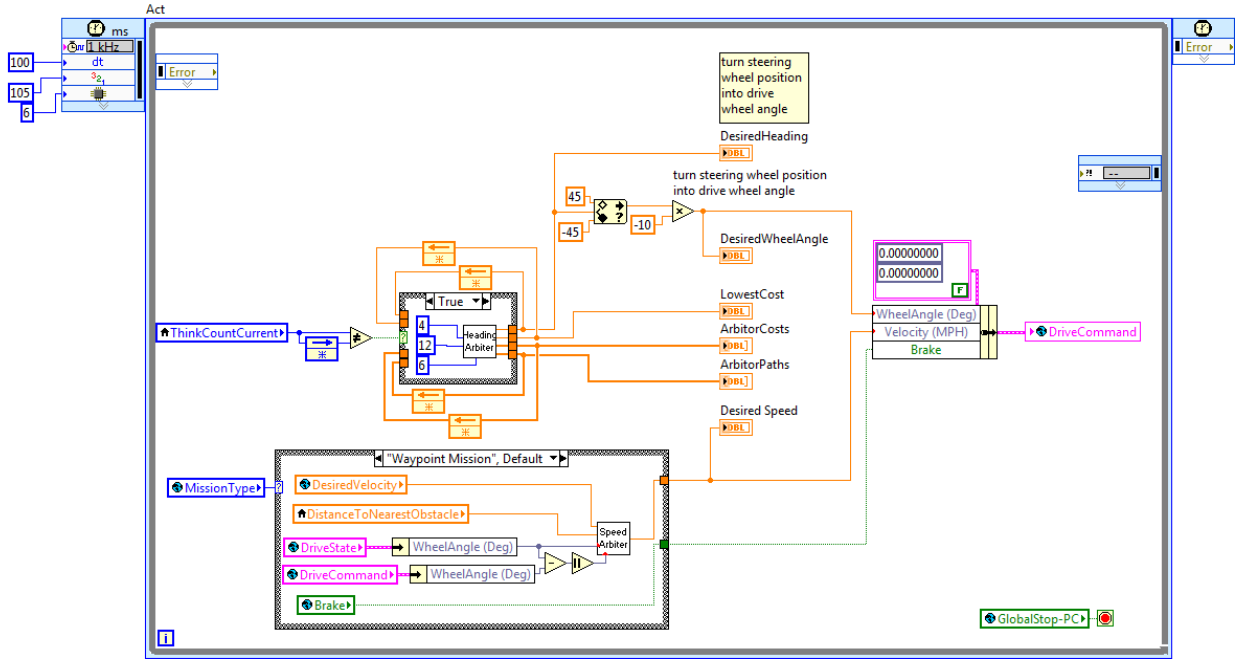


Figure 62: The portion of the Midbrain code responsible for arbitrating the vehicle's heading and velocity.

Icon	Name	Description
Heading Arbiter	HeadingArbiterRev2.vi	Considers all of the cost and threshold proposals and then selects the lost cost permissible heading. This is a blending arbiter.
Speed Arbiter	SpeedArbiterRev3.vi	Determines the velocity of the vehicle. This is a selection (not blending) arbiter that slows the vehicle to the slowest proposed speed.

User Interface

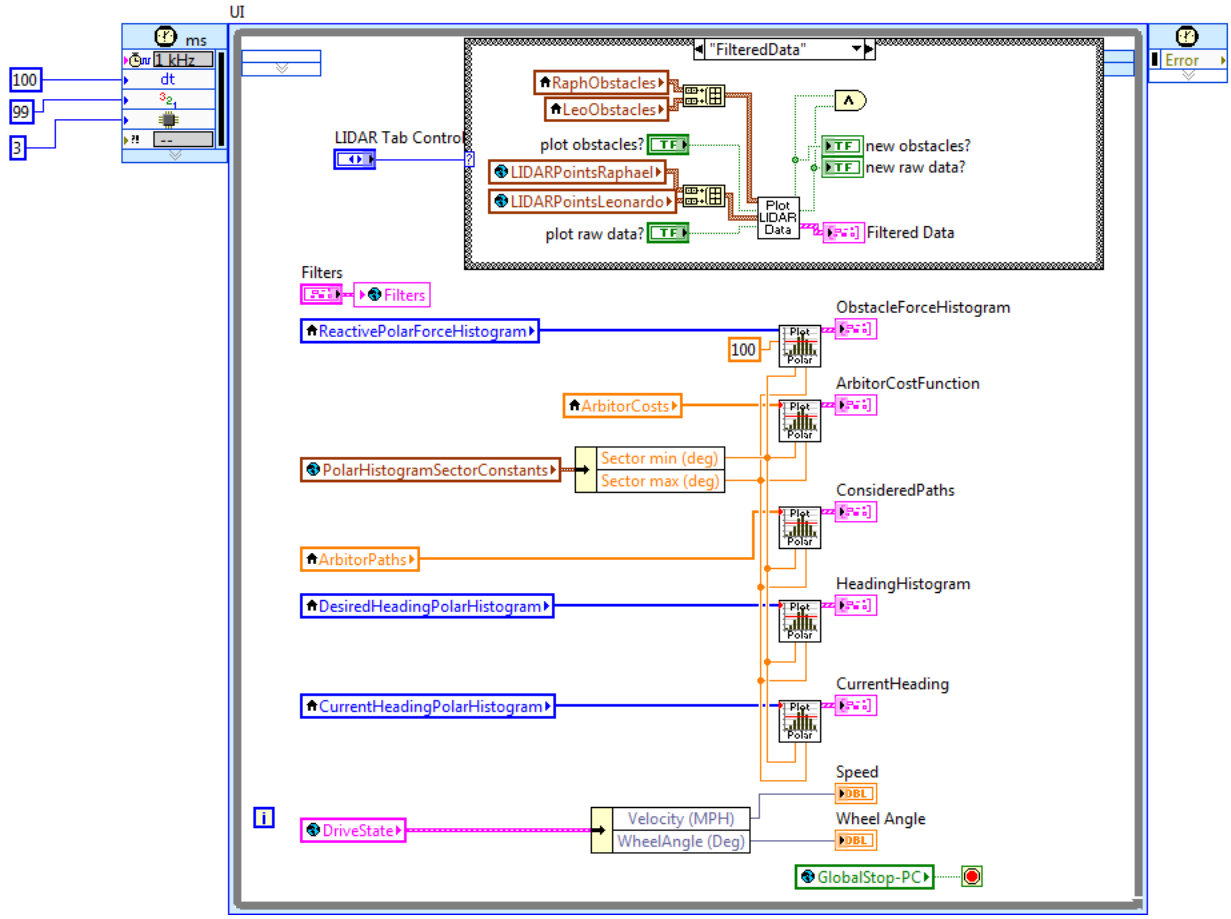


Figure 63: The portion of the Midbrain code responsible for updating the Midbrain's user interface.

Icon	Name	Description
Plot LIDAR Data	PlotLIDARData.vi	Generates an xy plot of two different point clouds. Allows the user to select which set or sets of points to display.
Plot Polar	PolarHistogramPlotter.vi	Generates a histogram plot of the provided polar histogram. Optionally plots a threshold line for threshold proposals.

Hindbrain.vi

This VI contains all of the code that runs on the FPGA. It contains motor and actuator control as well as LIDAR input, output, and coordinate transformations. There are two loops which control the front panel interface and a number of subVIs which do actual I/O and control.

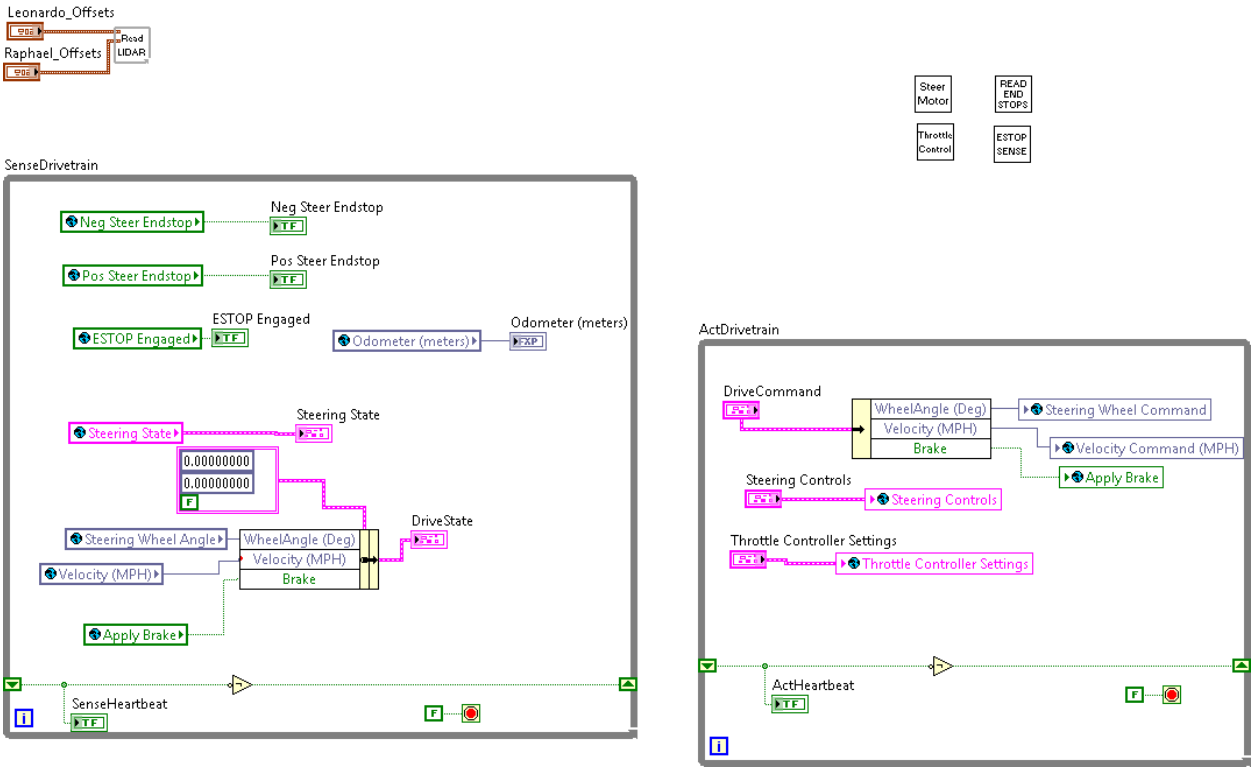


Figure 64: Hindbrain.vi block diagram.

Icon	Name	Description
Read LIDAR	ReadLidar.vi	Sets the LIDARs to continuous measurement, reads in data, transforms data into vehicle-centric Cartesian coordinates, and puts data into a DMA channel to send to the PC.
Steer Motor	DriveSteerMotor.vi	Reads encoder information for the steering motor and performs PID control on the steering wheel position.
Throttle Control	Throttle.vi	Reads the rear wheel encoder and performs PID control on the on the throttle to maintain vehicle speed.
READ END STOPS	ReadEndstops.vi	Reads the value of the steering endstops to ensure that the steering wheel will not hit a mechanical stop.
ESTOP SENSE	EstopSense.vi	Checks to see whether the Estop is currently engaged.

SenseDriveTrain Loop

This loop reads the current state of the drivetrain (speed, wheel angle, etc.) and writes the state to the front panel so that it is readable from the PC.

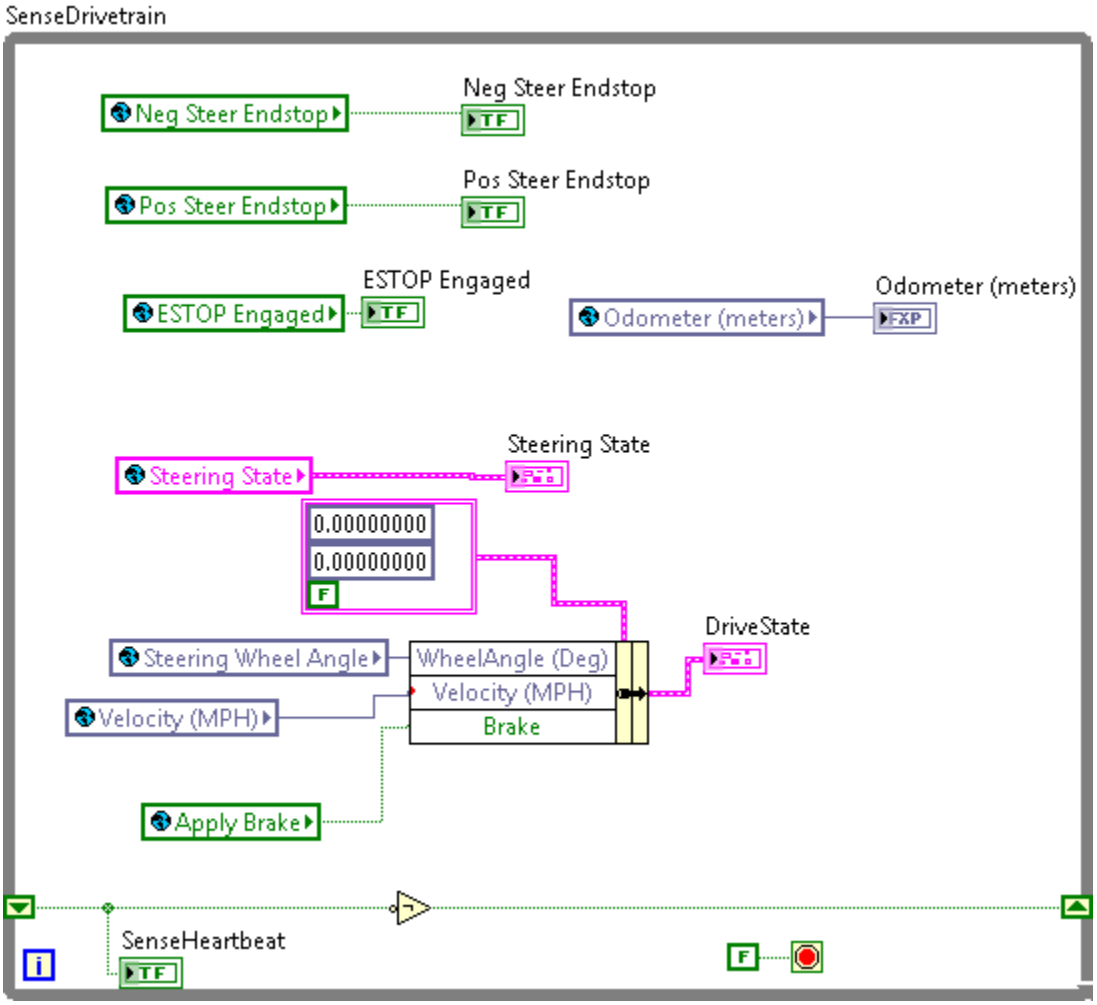


Figure 65: SenseDriveTrain Loop Block Diagram

ActDriveTrain Loop

This loop is used to command the state of the steering and throttle controllers.

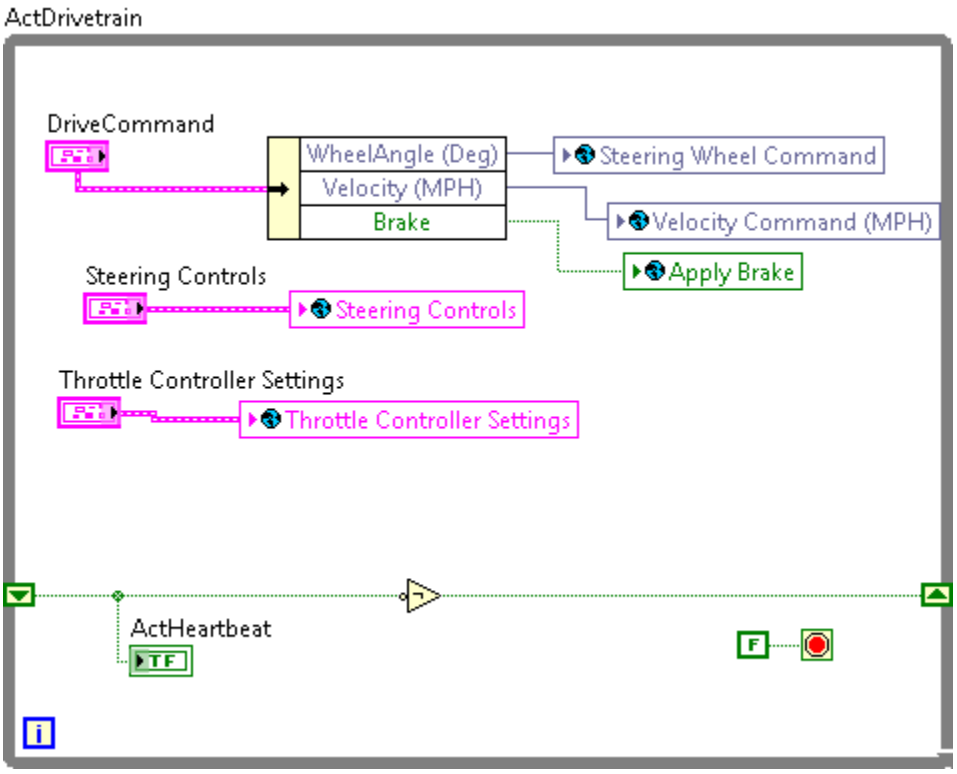


Figure 66: ActDriveTrain Loop Block Diagram