

Autonomous Ground Vehicle



2009-2010 Final Report

Prepared for:
Draper Labs

Sponsor Liaison:
Troy Jones

SCOPE Project Team:

Jared Barrow	Gregory Marra
Andrew Barry	George Sass
Giulia Fanti	Alex Trazkovich
Nicole Lee	

Faculty Advisor:
David Barrett

April 30, 2010

Table of Contents

1	Acknowledgments	6
2	Executive Summary.....	7
2.1	Project Scope	7
2.2	System Design	7
2.3	Project Status.....	7
3	Original Project Proposal.....	7
4	Introduction	9
4.1	Project Goals	9
4.2	Drive-by-Wire Conversion.....	9
4.3	Autonomy	10
5	Electrical System Design	10
5.1	Power Distribution	10
	Electronic Board Mounting	11
	Handling Power Loss	16
	Waterproofing Electrical System	18
5.2	Computational System	19
	Main Computer	19
	Real Time Computer	19
	Network Communication	19
	Wireless Networking	20
5.3	Sensor System.....	20
	NavCom VueStar GPS	20
	Pushbroom LIDAR	20
	INS	20
	Stereocamera	21
5.4	Emergency Stop System	21
	Emergency Stop Electrical Disconnect	22
	Manual Driver Control	23
6	Mechanical Systems.....	23
6.1	PQ Controls Linear Actuators	24
6.2	Brake Actuation System	24
6.3	Diesel actuation system.....	27
	Actuator Mounting	27
	Actuating the Pedal Arm	31
	Performance	32
6.4	Steering Actuation	32
6.5	Wheel Encoder Mount System.....	33
6.6	Generator Mount	36
6.7	Steering Limit Switches.....	38

6.8	Monitor Mount	40
6.9	Cabin Wire Routing	41
6.10	GPS mount	41
6.11	Electronics Enclosure Modifications	41
6.12	Electronics enclosure mounting	42
7	Vehicle Software Architecture	42
7.1	Goals	43
7.2	Parallel Processing	43
7.3	Hierarchy	43
7.4	Example: Adding a Xbox Controller Driver	44
	Architecture of an Xbox Controller Driver	44
	Sense Block	44
	Think Block	45
	Integration	50
	Testing	50
7.5	Hybrid FPGA / Scan Mode Interface	51
7.6	cRIO Cards	51
	NI 9041 Digital I/O	51
	NI 9403 Digital I/O	51
	NI 9263 Analog I/O	52
7.7	LIDAR Safety System	52
8	Logging/Replay System	53
8.1	Logging	53
	Logging Threads	53
	XML Log Format	54
8.2	Replay	54
	Overview	54
	Random Access	54
9	Controls	55
9.1	Acceleration	55
9.2	Steering	56
9.3	Braking	58
10	Waypoint Following and Path Planning	59
10.1	Curve Planning	60
11	Conclusion	61
11.1	First Semester Accomplishments	61
	Actuation	61
	Power Distribution	62
	Emergency Stops	62
	Computation	62
	Sensors	62
	Software Architecture	62

Controls	62
11.2 Second Semester Accomplishments	62
Electrical System	62
Emergency Stops	63
Sensors	63
Controls	63
Autonomy	63
11.3 Future Plans	63
GPS Denied Navigation	63
Rough Terrain Path Planning	63
Dynamic Obstacle Avoidance	64
A Turning On the Robot	65
A.1 Turning On	65
A.2 Initializing Software	65
A.3 Software Settings	66
A.4 Run Self-Test	66
A.5 Following Waypoints	67
B Electrical System Diagrams and CAD	69
C Mechanical System Diagrams and CAD	71
D Datasheets	75
E LvROS Overview	141
F Table of Queues	147
G LvROS Code	153

List of Figures

1 Honda eu2000 Generator	11
2 Electronics Enclosure CAD Render	12
3 Electronics Enclosure Right Side	14
4 Electronics Enclosure Left Side	15
5 Wireless Router	17
6 Generator Mount	18
7 Rear Emergency Stop Button	21
8 E-Stop Button	22
9 PQ Controls Model 751 Linear Actuator	24
10 Brake Pedal Mounting Bracket	25
11 Actuator Mounting Bracket	26
12 Brake Pedal Extender Arm	26
13 Double Clevis Linkage	27
14 Assembled Brake Pedal Actuation Linkage	28
15 Brake Pedal Actuation In Action	29
16 Brake Pedal Actuation System Installed	29

17	Diesel Pedal Actuation CAD Render	30
18	Cross Section of Actuator Mount Adapter Plate	31
19	Accelerator Actuator System	32
20	Power Steering Motor Mount	34
21	Power steering adaptor	34
22	Power steering motor support bracket	35
23	Wheel Encoder Mounts	35
24	Wheel Encoder Mounts	36
25	Generator Mount	37
26	Generator Mount	38
27	Generator Mount	39
28	Monitor Mount	40
29	GPS Mount	42
30	LvROS Software Hierarchy	44
31	LvROS Sensor Hierarchy	45
32	Creating a type-definition	46
33	Read Xbox Controller Values	47
34	Adding a Queue	48
35	Xbox Sense Block	49
36	Xbox Think Block	50
37	Select Drive Command Block	51
38	Replay User Interface	55
39	Velocity Control Block Diagram	56
40	Steady State Vehicle Behavior	57
41	Steering testing	59
42	Examples of Dubins Curves	61
43	Turning on the Generator	65
44	Sea Level PC Power Switch	66
45	Program Select	67
46	GPS Map Tab	68
47	Electronics Top Level	69
48	Electronics Bottom Level	70

List of Tables

1	Electrical Power Buses	11
2	Electrical Bus Usage	13
3	NI 9401 Connections	52
4	NI 9403 Connections	52

1 Acknowledgments

The 2009-2010 Draper-Olin SCOPE Team would like to make a few acknowledgments to parties whose guidance and insight were invaluable to our progress on this project. We thank Draper Labs for sponsoring the project and providing guidance and expertise from their experience with other autonomous platforms. We particularly thank Linda Fuhrman from Draper for arranging funding of the project and Troy Jones from Draper for acting as our team's liaison and offering insight from his experience with the MIT Grand Challenge vehicles. We would like to thank David Barrett for acting as our faculty advisor and providing us with invaluable advice regarding our electrical and mechanical modifications to the John Deere Gator. We would like to thank Olin sophomore Emily Towers for her assistance with the project. We also thank Tom Bottiglieri for offering insight from his experience with the MIT CSAIL / Draper Labs / Lincoln Labs Agile Robotics Autonomous Forklift project.

2 Executive Summary

2.1 Project Scope

This project is a collaboration between Draper Labs and Olin College to create an autonomous ground vehicle (AGV) research platform. Once completed, this vehicle will be capable of serving as a fully functioning research platform for both parties. Draper Labs will have easy access to an intact and fully autonomous vehicle that can be used, among many other things, for testing of novel sensors, calibration of sensory equipment, and demonstration of guidance systems. With this in mind, one of the primary goals for the first year of development is to create a system that is robust, maintainable, and easy to expand. All three of these qualities support the long-term viability of this project, especially given that students working on the project will have a comparably high turnover rate (Olin has no graduate program). As such, it is crucial that all systems be thoughtfully designed and well documented, with an emphasis on balancing platform stability with facilitation of system upgrades.

2.2 System Design

Like all robotics endeavors, this project is centered around the integration of mechanical, electrical, and software systems. The primary goal of the first semester has been to create a stable base platform, in both hardware and software, with a particular focus on the *drive-by-wire* conversion of the vehicle. Both the electrical and mechanical systems of the vehicle have been designed and submitted for review to ensure both safety and durability. After reviewing designs and prototyping concepts where possible, the core electrical and mechanical systems have been fabricated and installed on the vehicle. The team has also spent considerable effort on developing a robotics operating system. In particular, the software architecture is natively asynchronous and all code is compartmentalized according to the *sense-think-act* paradigm of robotics, with the goal of simplifying code integration to allow for easy development and debugging.

2.3 Project Status

With the majority of the drive-by-wire conversion completed by the end of the first semester, the second semester of the project has been focused on expanding the vehicle's autonomy capabilities. This includes mounting additional sensors to expand the vehicle's sensor suite, development of obstacle avoidance and path planning algorithms, and finalization of hardware systems. Future stages of the project will particularly emphasize software development (particularly implementation of additional autonomous behaviors), researching and testing of novel sensors, and integration of wireless communication and emergency stop systems.

3 Original Project Proposal

Statement of Work

1. Build prototype unmanned ground vehicle to support general robotics research at Olin and Draper Labs.
2. Develop a wide range of perception sensor systems (LIDAR radar, stereo vision, etc.) to allow high speed navigation in a complex woodland environment.
3. Design comprehensive, modular, expandable payload bays to support a wide array of future sensor and actuator payloads.
4. Continue to develop a LabVIEW based robotics toolbox and development system to allow the easy generation of complex unmanned mission control code.

4 Introduction

This report is a compilation of documentation for the 2009-2010 Draper-Olin SCOPE project. The final goal of the project is to create an Autonomous Ground Vehicle, based off of the John Deere Gator XUV, that will ultimately serve as a test and research platform for both Draper Labs and Olin College. This report will focus on the first year goals of the project, with discussion of long-term context as appropriate.

4.1 Project Goals

The year-long goal of this project is to create a system that can move autonomously and safely navigate complicated and dense outdoor environments. In addition to its native infrastructure, the vehicle must also be capable of carrying a payload of up to 500W weighing up to 100lbs. that has maximum dimensions of $2' \times 1' \times 1'$.

4.2 Drive-by-Wire Conversion

The first semester of this project centered around the drive-by-wire conversion of the base platform, a John Deere Gator XUV utility vehicle. The primary conversions to this vehicle are installation of robust mechanical actuation systems for the brake, accelerator, and steering, as well as an electrical system to power this actuation system and all related computational components. The design and implementation of a robust emergency stop system has also been of high priority. Installation of necessary sensors (GPS, front LIDAR, encoders) also falls into the first semester, although additional sensors required for more robust autonomous navigation will be added as the project moves forward.

Testing of this system prioritizes robustness, ease of operation, and safety over complex autonomous behavior. All current field operations will be performed in a controlled parking lot environment and include extensive testing of safety and operational procedures. There are eight separate stages of the field test:

1. *Manual Driving*: An operator will drive the vehicle in manual mode, performing maneuvers such as minimum radius turns and multiple stops/starts. Testing will proceed if and only if the operator reports no obstructions to manual driving caused by any systems installed.
2. *Manual Driving to Pause Mode*: The vehicle will be switched from manual to computer control, which will automatically put the vehicle into Pause mode. In this mode, the brakes should be applied and the steering wheel should remain locked in its last commanded position.
3. *Disable During Pause Mode*: While the vehicle is in Pause mode, the operator will activate a hardware emergency stop that will prevent the vehicle from accelerating or steering.
4. *Pause Mode to Run Mode*: While the vehicle is in Pause mode, the operator will switch the vehicle to Run mode with the software manual override activated. The vehicle should not accelerate or steer during this transition.

5. *Disable During Run Mode:* While the vehicle is in Run mode (with the software manual override activated) the hardware emergency stop will be activated. The vehicle should not accelerate or steer during this transition, but all computer systems should remain online.
6. *Autonomous Driving:* All emergency stops and software manual override will be released and the vehicle will be driven under computer control. This can include either pre-programmed maneuvers or driving via a front panel Graphical User Interface.
7. *Manual Override During Autonomous Driving:* While the vehicle is driving in a straight line, with a velocity of 5MPH, the operator will activate the hardware emergency stop system and take manual control. Computer control should be released and the operator should be able to drive without impediments.
8. *Autonomous Driving to Pause Mode:* While the vehicle is driving in a straight line, with a velocity of 5MPH, the operator will activate a software pause. The vehicle should perform a controlled stop and remain static once stopped. After successfully stopping, the vehicle will be put back into Autonomous mode.

4.3 Autonomy

The second semester of the project centered on the development of more complex autonomous behaviors. Specific objectives during this time period include robust waypoint following, simple path planning, and static obstacle detection. Future behaviors to develop may include target identification, GPS-denied navigation, and path planning in three-dimensional environments. The long-term goal is to successfully complete autonomous woodland driving in Olin's robotics test track ("Parcel B").

5 Electrical System Design

The electrical system is a crucial part of autonomous operation. As the Gator's alternator does not provide sufficient power, an independent electrical system is required to convert the Gator to a drive-by-wire platform. This means installation of a power distribution system, addition of a computational system, and inclusion of a high-reliability emergency stop system. The entire electrical system is tightly integrated; all sensors, computers, actuators, and emergency stops are powered from a common source and share a common ground. It is worth noting that this system is also independent of the vehicle and can be installed and removed with minimal modification to the base platform (modifications made in the course of this project are primarily to mount sensors, actuators, and other system components securely to the vehicle).

5.1 Power Distribution

Ultimately, all core systems on the Gator are powered by DC power. Electrical power for non-standard vehicle features is provided by a Honda eu2000 gasoline generator. This generator provides 1600W continuous AC power. Three AC to DC power supplies provide 12V, 24V,

and 48V DC buses. A 24VDC to 5VDC power supply provides a 5V DC bus. Note that while total power capacity of these power supplies exceeds the maximum 1600W continuous power from the generator, the Gator does not reach maximum power draw under normal circumstances.



Fig. 1. A photograph of the Honda eu2000 gasoline generator used to provide electrical power to our modifications to the John Deere Gator XUV. It provides up to 1600W continuous AC power.

Power Bus	Power Supply	Generally Powers	Capacity
12VDC	MeanWell HRP-300-12	Gas and Brake Actuation	300W
24VDC	MeanWell HRP-600-24	Computational Systems and Sensors	648W
48VDC	xppower LCL500PS48	Steering Actuation	500W
5VDC	MeanWell SD-15B-05	Basic Sensor Power	15W
AC	Honda eu2000	Draper Payload 120V AC	500W

Table 1. Electrical power buses on the Gator.

The 12VDC and 24VDC power buses run to fuse panels that serve as their primary distribution points. The 48VDC bus runs directly into the Victor speed controller that drives the steering actuation motor. All of the power buses share the same DC ground, but care has been taken to avoid ground loops in the system.

Electronic Board Mounting The electronics in the rear electronics enclosure are divided between the “top board” and the “bottom board”. The bottom board holds the power sup-

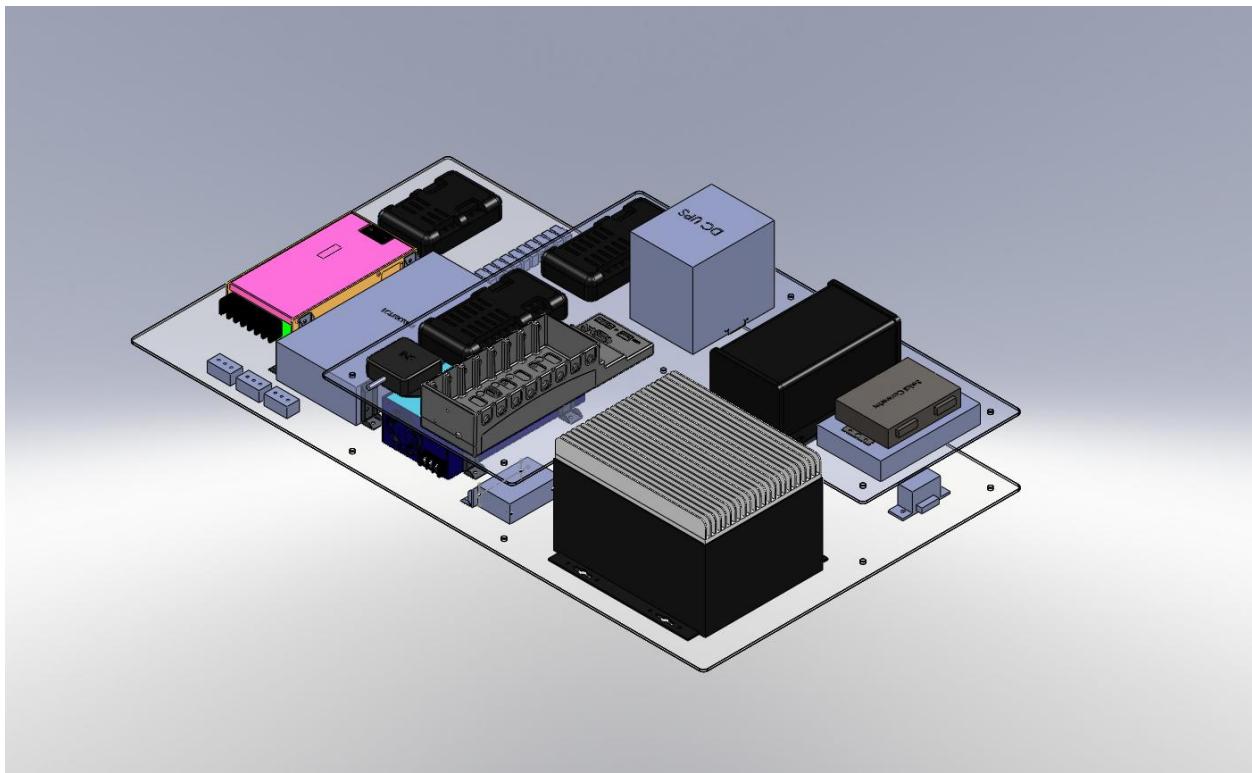


Fig. 2. CAD of the electronics boards mounted inside the electronics enclosure.

Component	Bus	Max Draw
Steering Motor	48VDC	84W
48VDC TOTAL	M48	84W / 500W
cRIO	24VDC	24W
NavCom	24VDC	24W
Roof LIDAR	24VDC	43.2W
Ethernet Switch	24VDC	24W
Wireless Access Point	24VDC	24W
SeaLevel PC	24VDC	90W
INS	24VDC	24W
24VDC TOTAL	24VDC	325W / 648W
Accelerator Linear Actuator	12VDC	96W
Brake Linear Actuator	12VDC	96W
Electronics Cooling Fans	12VDC	6W
Emergency Stop Relay	12VDC	~0W
12VDC TOTAL	12VDC	192W / 300W
Draper Payload	AC	500W
Daylight Visible Monitor	AC	240W
AC TOTAL	AC	740W
12VDC TOTAL	12VDC	192W / 300W
Rear Wheel Encoder	5VDC	~0W
Steering Wheel Encoder	5VDC	~0W
Steering Magnetic Limit Switches	5VDC	~0W
5VDC TOTAL	5VDC	~0W / 150W

Table 2. Electrical power bus usage on the Gator. Note that while total power capacity exceeds the maximum continuous power of 1600W from the generator, the Gator does not reach maximum power draw under normal circumstances.

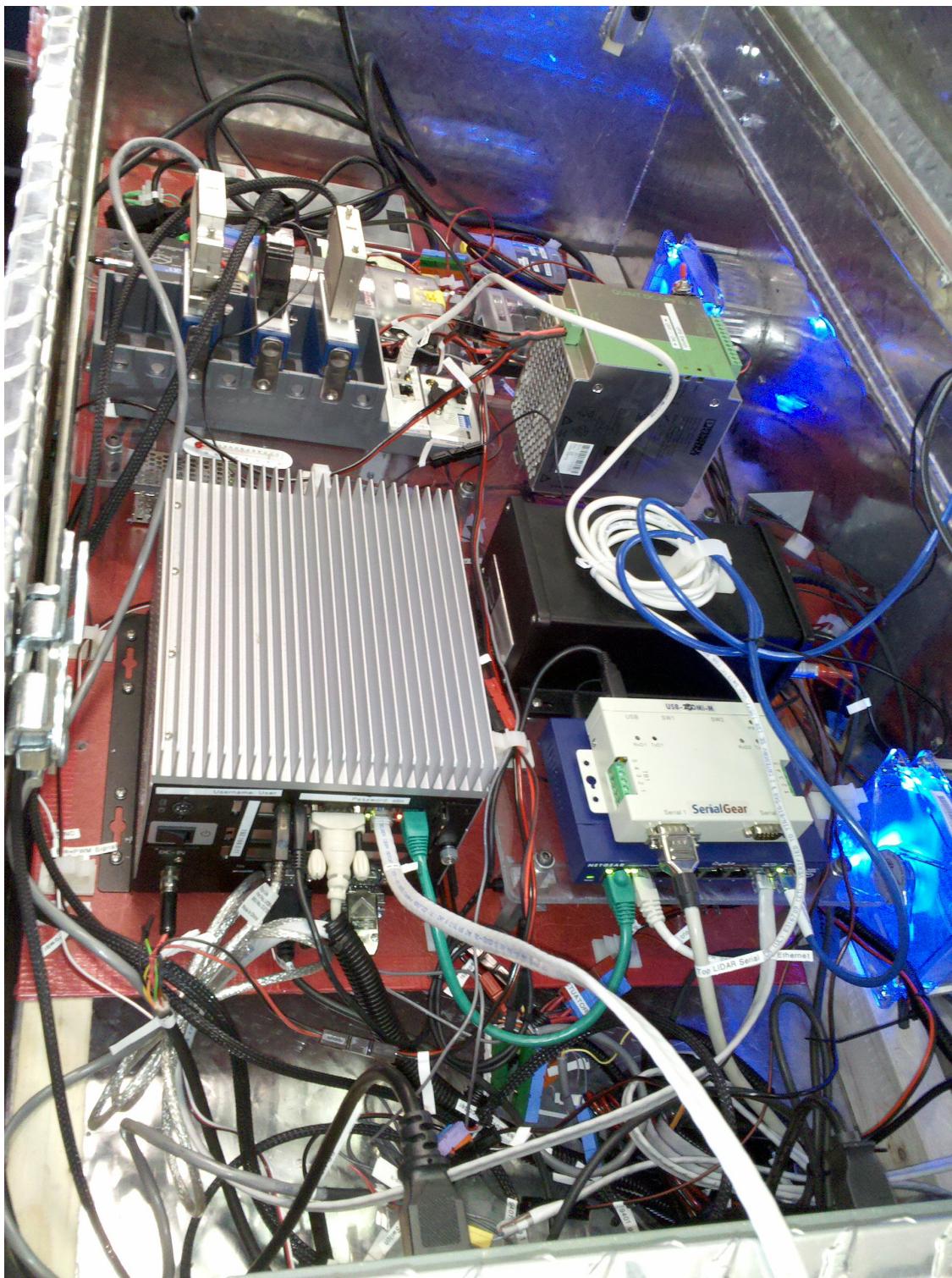


Fig. 3. A view into the electronics enclosure from the passenger side of the Gator. All of the wires enter the enclosure through a hole in the bottom front passenger side of the box.

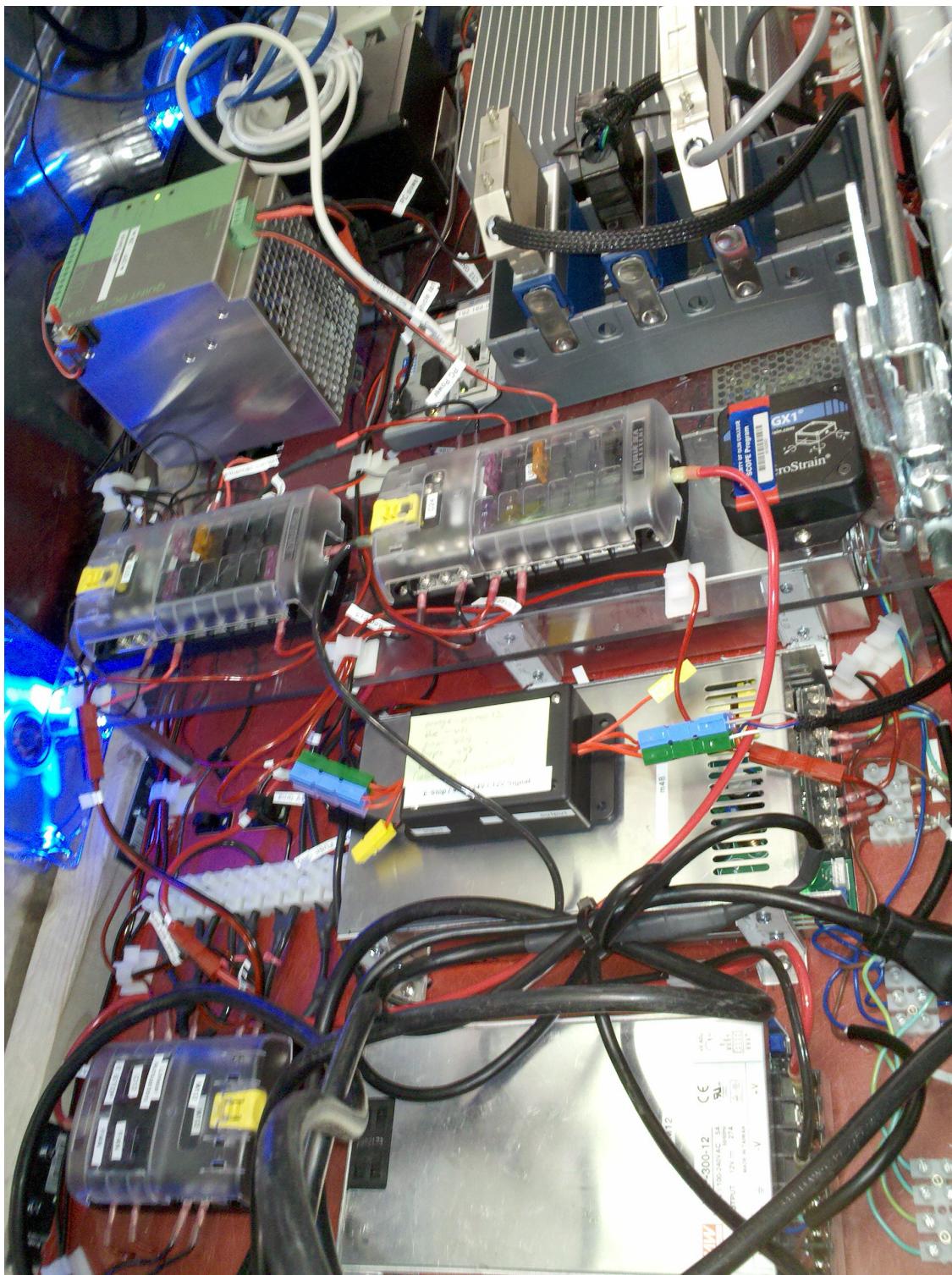


Fig. 4. A view into the electronics enclosure from the driver side of the Gator.

plies, steering motor speed controller, emergency stop system, actuator power distribution, and the x86 PC. The top board holds the GPS, INS, DC UPS, cRIO, networking equipment, and sensor and computational power distribution. Components were placed depending on how frequently we felt they would need to be accessed.

The top board is mounted above the bottom board on 1/4-20 hex standoffs. Almost all connections that span between boards are disconnectable inline, allowing the top board can be easily removed to allow access to the bottom board. Some connections, principally those with proprietary and expensive cables (such as the GPS and INS), must be disconnected from top board components before the top board can be removed. All connections are labeled on both sides and color coded to make reconnection quick and easy.

The bottom board is mounted to a wood frame inside of the aluminum electronics enclosure. 1/4-20 threaded inserts have been added to the wood frame so that the bottom board bolts into it. Due to imprecisions in the mounting of these inserts, only about five of the nine mounting holes can be used. Nonetheless, this provides more than adequate security for the electronics boards.

The aluminum electronics enclosure is mounted on rubber shock absorbers to two steel box tubes in the bed of the Gator. This allows access underneath the enclosure for cable routing. The shock absorbers provide a degree of vibration isolation for the electronics. The two steel box tubes are bolted to the bed of the Gator.

As more electronics have been added to the system, some of the wall space of the aluminum electronics enclosure has been used. An AC power strip is mounted to one wall of the box. It serves as the primary AC distribution point for the vehicle, allowing easy switching between generator AC power and wall AC power. The vehicle's wireless router is mounted to the lid of the electronics enclosure. An emergency stop button is mounted to the rear exterior of the electronics enclosure.

Handling Power Loss The Gator sometimes loses electrical power. The most common scenario is switching from the generator to wall power while transitioning from a field test back to the garage. To mitigate the problems caused by losing system power, a DC Uninterruptible Power Supply is installed on the top board. The 24VDC power to the cRIO and x86 PC pass through this UPS, which is capable of supporting these two computational systems for up to fifteen minutes. This means that the vehicle can lose power without requiring an entire system reboot, although network communication, sensors, and everything else will go dark during the no-power period.

Since the DC UPS is smart and automatically provides power during power loss, it must be specifically told to turn off when you are finished operating the Gator. The DC UPS has remote turnoff functionality, so a toggle switch has been mounted to its side to shut down the UPS when finished with operations. The switch is normally closed during operation, and switching it off and back on disables the UPS and readies the vehicle for the next mission.

To detect power losses, we have constructed a level converting box that uses voltage dividers to bring each power bus (12v, 24v, 48v, and e-stop status) down to 5v to be read by the cRIO's 9403 DIO card. This enables the system to know the state of each power

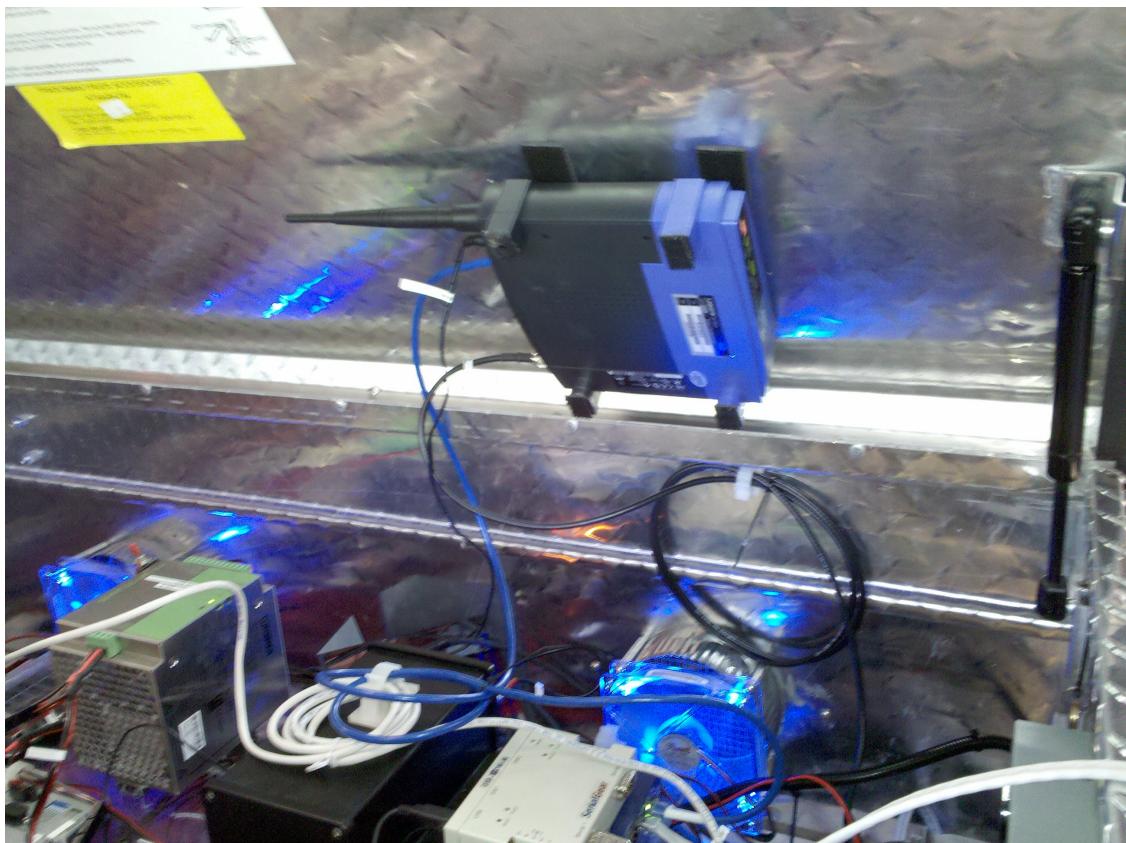


Fig. 5. The wireless router is mounted with high strength velcro to the lid of the electronics enclosure.

supply and the emergency stop system. Due to back-EMF issues with the steering motor, the 48VDC power supply occasionally entered over-voltage protection mode and shut down during field testing, so this system has proven very useful for monitoring vehicle status.

Waterproofing Electrical System The Gator needs to be able to operate during inclement weather conditions and is hose washable for easy maintenance. To meet these requirements, care has been taken to design electronic housings to mitigate water threats.

The primary electronics enclosure is a 4' × 2' × 2' aluminum toolbox. Since this box houses our power electronics and computers, it requires a ventilation system for cooling. Two hooded vents have been installed in the box and sealed with marine silicon sealant. This allows us to circulate air through the box without the risk of water entering. It is possible to spray water into the electronics box from below these vents, but such an event is not expected under normal operating conditions.

For waterproofing of the generator electrical ports, a rubber sheet is affixed to the rear of the generator using silicon caulk as well as modifications to the generator bolt hardware. This waterproofing flap is shown in Figure 6.



Fig. 6. Waterproofing over the generator electrical ports.

Remote emergency stop buttons are mounted on the dashboard of the Gator and on the rear electronics enclosure. The rear emergency stop button is mounted externally, and is inside a NEMA 4X rated enclosure with liquid sealing strain relief passthroughs for the wires.

Bulgin Buccaneer IP68 rated electrical connectors or water sealed automotive spade connectors are used for all outdoor electrical connections, allowing removal and servicing of components while still ensuring that they remain waterproof. The electrical connections to the linear actuators and steering wheel motor are made using Buccaneers. The connections for these actuators in the cab near the operators' feet have proved particularly useful for disabling certain actuators during testing. A 6-pin Buccaneer is used to connect the rear speed encoder, enabling easy disconnection. The emergency stop buttons use sealed automotive spade connectors. Some connections under the Gator's hood, particularly the steering magnetic limit switches, are made using Anderson Powerpoles, but we do not anticipate this area of the vehicle experiencing adverse conditions.

5.2 Computational System

Computational power is an essential part of any autonomous vehicle. This section is meant to serve as a brief overview of the computational systems onboard the Draper-Olin AGV.

Main Computer The Gator's main computer is a SeaLevel Relio R5140. The computer is entirely solid state, incorporates a dual core CPU and 4GB of RAM, and is cooled with an external heat sink - there are no moving parts, which greatly increases durability and suitability for field operations. The I/O capabilities of the computer include six USB 2.0 ports, four serial ports, one VGA port, and two ethernet ports. The computer has an input voltage range of 18-28VDC at 90W, and is powered off of 24VDC in the Gator.

Real Time Computer All real time computing on the Gator, including control of all actuators, is handled by a National Instruments CompactRIO (cRIO). The cRIO consists of a real time processor (the Power PC) and an FPGA with a number of interchangeable I/O modules (cRIO cartridges/cards). Data from the various I/O cartridges can be accessed through the FPGA, and some cards can be connected directly through the Power PC in Scan Mode. The cRIO cards used in the Gator are enumerated below:

1. **NI 9401 Digital I/O Module:** High speed digital I/O. Currently used for reading rear wheel and steering wheel encoders.
2. **NI 9403 Digital I/O Module:** General purpose digital I/O. Used to read steering wheel motor limit switches and power bus status lines.
3. **NI 9263 Analog Output Module:** Multiple, accurate analog outputs for controlling the accelerator and brake linear actuators.

Network Communication Network communications on the Gator rely on the TCP/IP protocol. All computational components are connected to a common ethernet switch via

standard CAT5 cable. The ethernet switch is powered off of 24VDC. There is also a wireless access point providing network access to remote computers. The IP addresses of the x86 PC and the cRIO are written on them. At the time of this report, the x86 PC was 192.168.0.11 and the cRIO was 192.168.0.12.

Wireless Networking The ethernet switch is connected to a Linksys WRT54GL wireless access point running the third party dd-wrt firmware. The wireless router is mounted inside of the electronics enclosure. The wireless network name is “OLIN_GATOR”, and there is no wireless encryption. To configure the wireless access point, visit 192.168.0.1 on the network. The username is “gator” and the password is “olin”.

To connect to the wireless network, your computer will need a static IP address. Use an IP of the form 192.168.0.X with a subnet mask of 255.255.255.0 and 192.168.0.1 as the default gateway. Don’t use an IP address used by the x86 PC or cRIO.

The wireless access point has had one antenna replaced with a Hyperlink Technologies HyperGain Model HGV-2410U antenna to provide greater range and signal strength. The antenna is mounted outside of the electronics enclosure on springs to provide some resilience to shock forces.

5.3 Sensor System

Paired with the computational system is a flexible sensor suite. The first year of the sensor suite is limited for the sake of simplicity, and a number of sensor additions are planned for future development. The sensors currently included on the vehicle are outlined below.

NavCom VueStar GPS 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. 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.

Pushbroom LIDAR In order to avoid obstacles directly in front of the vehicle, the Gator has been outfitted with a pushbroom Sick LMS-29i LIDAR mounted on the roof. This will allow for detection and avoidance of both static and dynamic obstacles in the vehicle’s path, but does not facilitate dodging moving obstacles that may be on a collision course with the vehicle. As all planned tests involve only static obstacle avoidance, this is not presently a problem, but may be an important consideration going forward. The Sick LIDAR is powered off of 24VDC at 30W, has an operating range of up to 80m with 1mm accuracy, captures data in a 90° viewing angle, and updates at a maximum rate of 50MHz.

INS A Microstrain 3GX-i Inertial Navigation System (INS) is incorporated into the sensor suite to complement GPS data. In addition to providing dead-reckoning position data (most

useful in GPS-denied areas), the INS will also give information on pitch, roll, and yaw. This will allow for better terrain mapping and will increase operational safety by allowing for development of pitch/roll/yaw-based maneuvers aimed at avoiding rolling over or otherwise destabilizing the vehicle. The INS is mounted inside of the rear electronics enclosure lined up with the coordinate frame of the vehicle.

Stereocamera Though a stereocamera is currently mounted in the cabin of the vehicle, it is not currently integrated into the vehicles path-planning algorithms.

5.4 Emergency Stop System

Safety is of the utmost priority for the team. Operating an experimental autonomous vehicle as large as the Gator poses a potential risk to life and limb. We have striven to engineer a safe emergency stop mechanism that enables the operating crew to terminate autonomous operations in the event risky situations arise.



Fig. 7. The rear emergency stop button is easily accessed from outside the vehicle.

Emergency Stop Electrical Disconnect The software platform of the Gator supports putting the vehicle into PAUSE mode, in which it will not send commands to any of its actuation systems. However, this software pause is insufficient to prevent undesired vehicle actions arising from programming errors, software crashes, or electrical wiring faults. To provide an override mechanism for these risks, the core of the Gator's emergency stop system is an electrical relay that physically disconnects the entire actuation system (steering, acceleration, and brake actuators) from electrical power in the event of an emergency stop.

Two mushroom-head emergency stop buttons (Figure 8) are mounted on the Gator. One is in the driver's cab on the dashboard and the other is on the rear of the vehicle mounted to the electronics enclosure. These emergency stop buttons latch when pressed, and do not reset until twisted to release. The buttons are in series and configured so that pressing any of the buttons (or any electrical disconnect between them) disrupts the emergency stop button circuit. This circuit is in turn connected to a normally-open dual pole dual throw (DPDT) electrical relay. As long as the emergency stop buttons are electrically connected to each other and not pressed, the relay will connect the gas, brake, and steering actuators to their respective power buses. Any emergency stop or fault causes the relay to revert to its normally-open state, removing power from the actuators.



Fig. 8. A sample E-Stop button.

An emergency stop will not remove power from sensors or computational equipment. The Gator's cRIO can sense the state of the emergency stop system. If the emergency stop has been tripped, the Gator control system moves into PAUSE mode, preventing further autonomous operation until an operator override. The vehicle only operates if both the

hardware and software emergency stops are disengaged. The cRIO software waits one second after removal of a stop condition before sending non-zero commands to actuators.

During the Summer 2010, a wireless emergency stop system will be added to allow unmanned operation.

Manual Driver Control Simply disconnecting electrical power from the actuators would not be enough to prevent an accident if the safety officer driver were unable to then manually take control of the vehicle. All actuation components have been designed to ensure the driver can manually operate the vehicle while the actuation system is unpowered. This enables the vehicle to be manually driven between autonomous missions, and allows the driver to retake control in an emergency.

The linear actuators that control the accelerator and brake pedals of the vehicle are electromagnetically clutched. When power is removed from their clutch line, they slide freely with minimal resistance. In testing, the Gator's factory installed brake pedal return spring is enough to return the brake pedal back to its neutral position when the clutch is disengaged. The factory installed accelerator pedal return spring was not strong enough on its own, so an additional, stronger return spring was added in parallel to the existing system.

The power steering conversion installed on the Gator has been intentionally designed to avoid using a large gearbox. Instead, we have installed a large, easily-backdriven motor to control the steering column during autonomous operation. When this motor is disconnected from power via the emergency relay, the driver can manually operate the steering wheel without an unusual amount of resistance.

Ensuring that all actuation systems are operable after removing power allows the safety officer driver to take control of the vehicle should a dangerous situation arise.

6 Mechanical Systems

The mechanical modification of the vehicle has focused on drive-by-wire actuation, electronics and sensor mounting, and operator interface construction.

The three actuation systems necessary for drive-by-wire operation are the brake pedal, the accelerator pedal, and the steering. Both brake pedal and accelerator pedal actuation are accomplished using linear actuators from PQ Controls. Steering actuation is accomplished using a heavily modified after-market power steering conversion kit available for the Gator XUV.

Sensors, computational power, and power supplies are mounted in various locations throughout the vehicle. The primary electronics box is mounted on two vibration isolating rails in the bed of the vehicle and contains the computers, the CompactRIO, the power supplies, and most of the power distribution elements on the vehicle. The vehicle also includes a roof-mounted optics-quality pegboard for sensor mounting—current roof-mounted sensors include a SICK scanning LIDAR and a NAVCOM GPS antenna with provisions for other sensors. Additionally, a forward-facing stereo-vision camera is mounted inside the cab.

Inside the cab, a daylight-visible monitor and keyboard/mouse tray provide an operator interface for the software programmer. In case of emergency, a dashboard-embedded E-stop

button allows the safety officer, seated in the driver's seat, to reclaim manual control of the vehicle.

The individual mechanical subsystems are outlined in more detail below.

6.1 PQ Controls Linear Actuators

Both the accelerator and the brake actuation systems utilize Model 751 linear actuators available from PQ Controls (Figure 9).



Fig. 9. PQ Controls Model 751 linear actuator

These actuators have a 3 inch throw and supply up to 90 pounds of force. The accelerator pedal requires approximately 20 pounds of force and 1.5 inches of throw, while the brake pedal requires approximately 45 pounds of force and 0.5 inches of throw. This allows for comfortable margins of safety for both force and distance. The actuators run on 12VDC and draw 3.8 Amps of current at stall.

Control is accomplished through a 1-4V command line where voltage maps linearly to position. Note that the end position and linear scaling factor are based on the voltage at the time of power-up, so for simplicity's sake, the actuators should always be powered on at 1V in the fully relaxed position.

A separate 12V clutch line provides the capability for emergency override—when power is cut to the clutch, the actuator arm can slide relatively freely, requiring only 8 pounds of force to move. This is well under the force provided by pedal return springs and the force which a human operator can easily apply. For simplicity, the linear actuators' clutch and power line have been tied together, so cutting power to the clutch also removes power from the actuator as a whole.

The actuators are environmentally sealed and resistant to humidity, water, dirt, dust, and mud.

6.2 Brake Actuation System

The brake pedal actuation system consists of a PQ Controls linear actuator mounted to the vehicle frame and coupled to the brake pedal. The existing brake pedal mounting bracket—

which is located above and behind the top of the brake pedal—provides a convenient attachment point for a linear actuator mount. This bracket is shown in Figure 10.

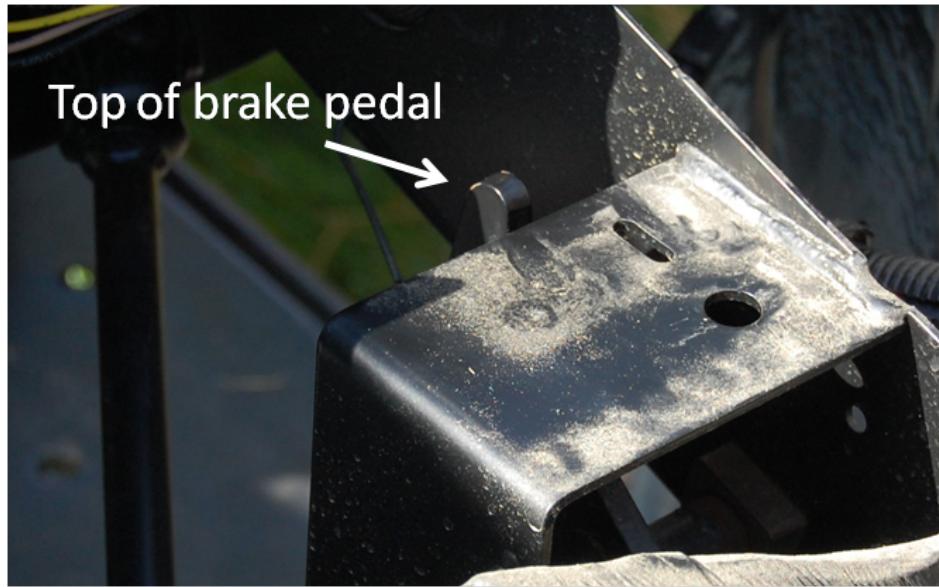


Fig. 10. The brake pedal mounting bracket.

Although this bracket provides a convenient mounting point that keeps the actuator away from the operator's feet and legs, space limitations under the Gator's hood prevent the linear actuator from being mounted in the typical fashion—that is with its mounting axis parallel to that of the axis of rotation of the brake pedal. Instead, the actuator must be turned on its side and mounted on a perpendicular axis in order to fit within the available space profile.

The actuator mounting bracket consists of a bent bar of 0.25" 6061 aluminum bar stock that holds a steel pin on which the actuator is mounted. Shaft collars prevent the pin from sliding free. In the future, the aluminum mounting bracket could be replaced with one made of black anodize steel, although this is not necessary for normal operation of the vehicle. The actuator mounting bracket bolts to the Gator brake pedal mounting bracket via two post-machined holes. Figure 11 shows the assembly before the linear actuator is added.

To allow the linear actuator to actuate the brake pedal, a brake pedal extender arm (shown in Figure 12) is manufactured from 0.25" aluminum plate and bolted onto the brake pedal. It is recommended that the brake pedal be removed from the vehicle prior to drilling the brake pedal extender arm mounting holes in order to ensure accurate hole placement and alignment.

Because the actuator is mounted on an axis which is not parallel to the axis of rotation of the brake pedal, an additional parallel axis of rotation must be introduced in the coupling linkage. This is accomplished through the use of a double clevis link—a device consisting of two parallel, pivoting pins held at a constant distance from one another. These linkages are

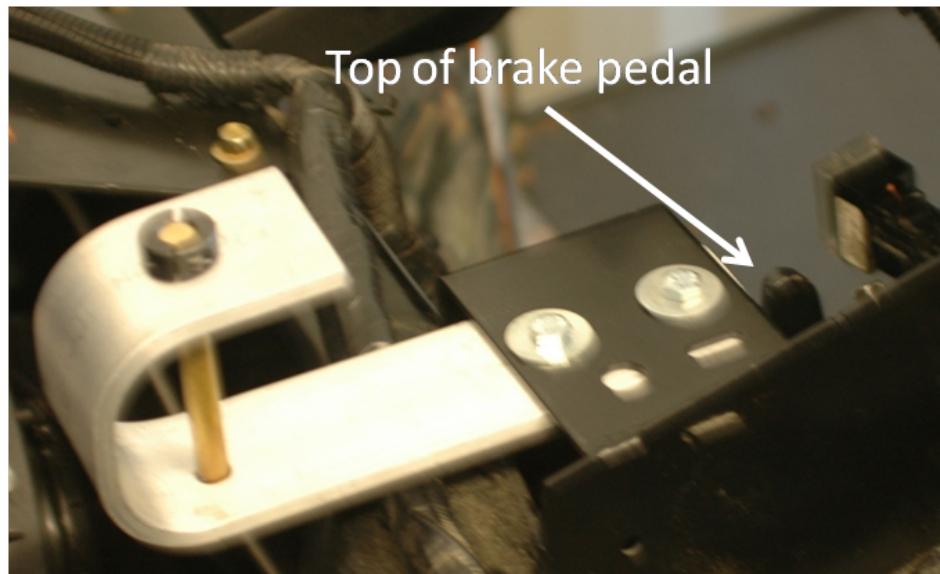


Fig. 11. The actuator mounting bracket mounted to the vehicle frame.

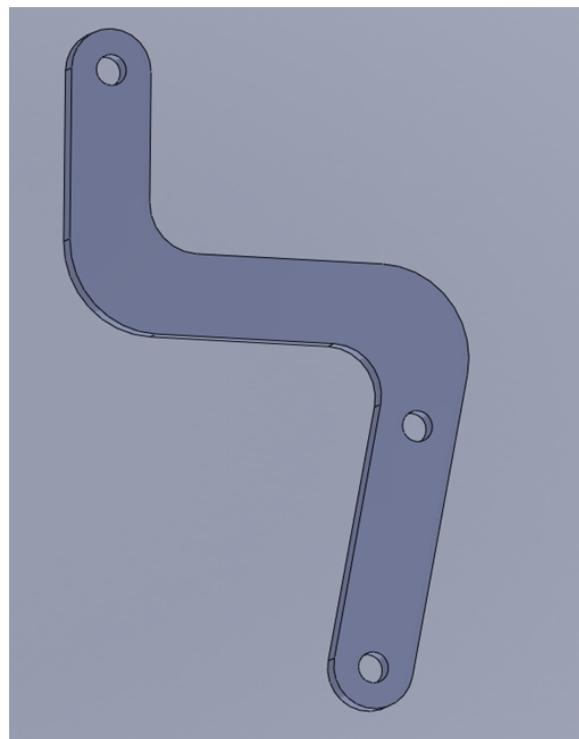


Fig. 12. The brake pedal extender arm.

available off-the-shelf from numerous hardware suppliers—the one currently used in the Gator is pictured in Figure 13. A 5/16th inch clevis link was used because it was the smallest readily available.



Fig. 13. The 5/16th inch double clevis link used in the current design.

The linear actuator rod is coupled with the double clevis link using a 5/16th inch ball-joint rod end. The other pin of the clevis runs through a bronze bushing which is press-fit into the exposed end of the brake pedal extender arm. The assembled linkage is shown in Figure 14.

A completed CAD model of the actuation system is shown in Figure 15. The CAD model is useful for understanding how the linkage converts the linear motion of the actuator to rotary motion of the brake pedal. The assembly is shown at both extremes of its actuation range—note how the double clevis linkage allows power to be transmitted without the actuator binding.

The brake pedal actuation system is fully functional and has not yet experienced a mechanical failure of any kind during testing. The completed system is shown in Figure 16.

6.3 Diesel actuation system

Our diesel actuation system is based on the same PQ Controls linear actuator as the brake actuation system. It takes advantage of the portion of the diesel pedal arm that connects to the return spring (see Figure 17). It consists of a mounting plate, a U-bracket which holds a pivot joint to mount the actuator, the actuator itself, and an arm which connects to the return spring section of the pedal arm. The return spring itself is not removed, but is simply slid to one side so both systems can mount to this section of the pedal arm.

Actuator Mounting The mounting plate for the actuator takes advantage of the center column of the vehicle being pre-drilled for a cup holder. The cup holder is held to the center column with four expanding panel clips and can be removed easily by pulling out the four retaining pins for the clips. This exposes the center column, which is made out of 0.125" steel, and four 0.28" holes drilled to mount the cup holder. An adapter plate to connect the U-bracket was machined out of 1/4" aluminum using the abrasive water jet cutter. Mounting this plate was a challenge, since the center column is welded shut from the bottom, and the

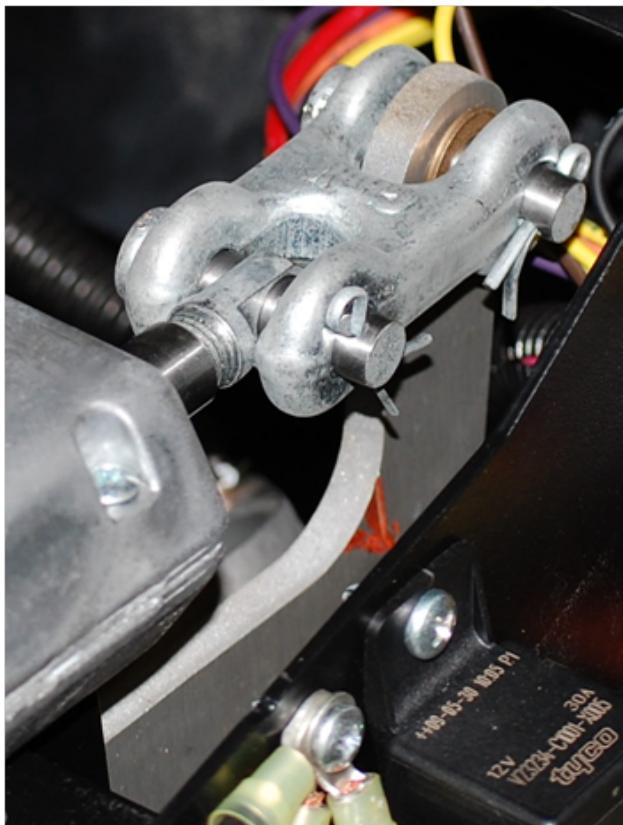


Fig. 14. The assembled brake pedal actuation linkage (top view).

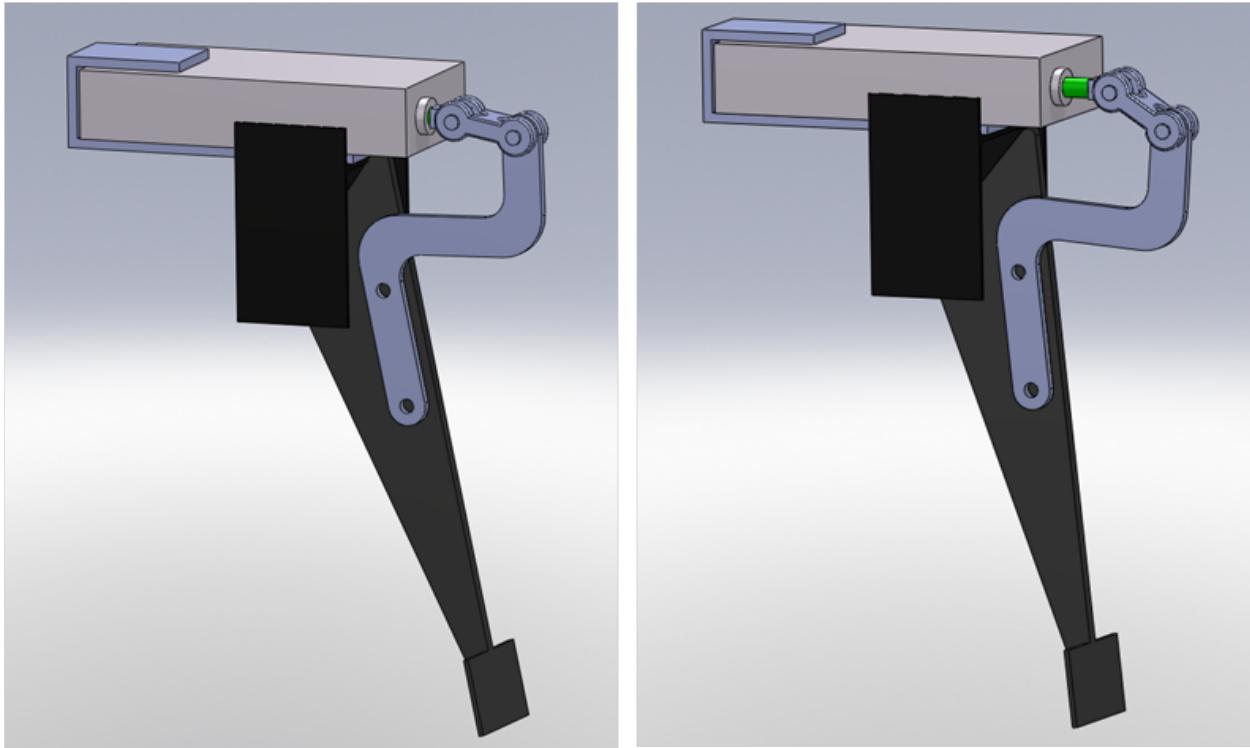


Fig. 15. The brake pedal actuation system, shown with the pedal at rest (left) and depressed (right).

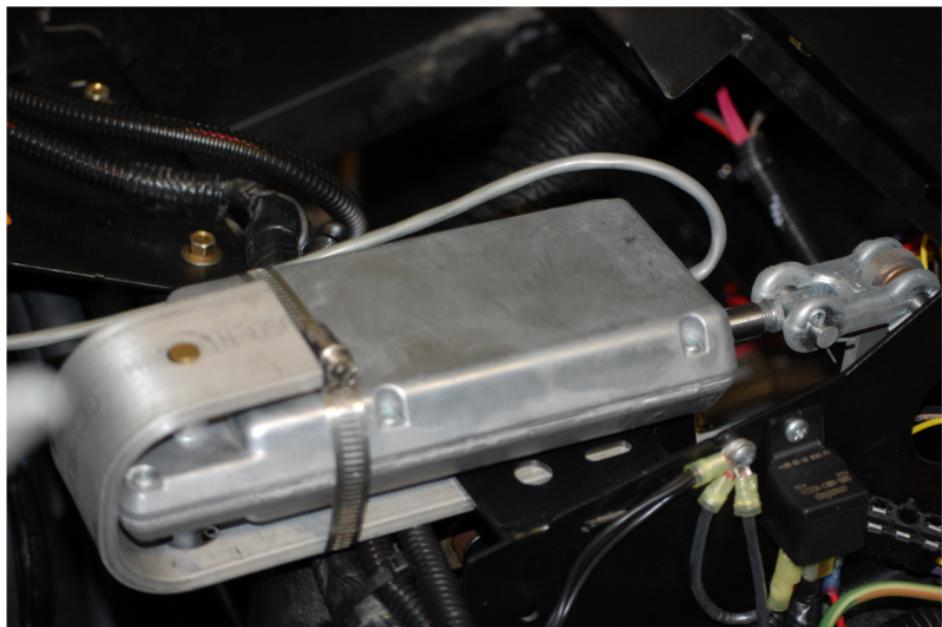


Fig. 16. The brake pedal actuation system fully installed on the Gator.

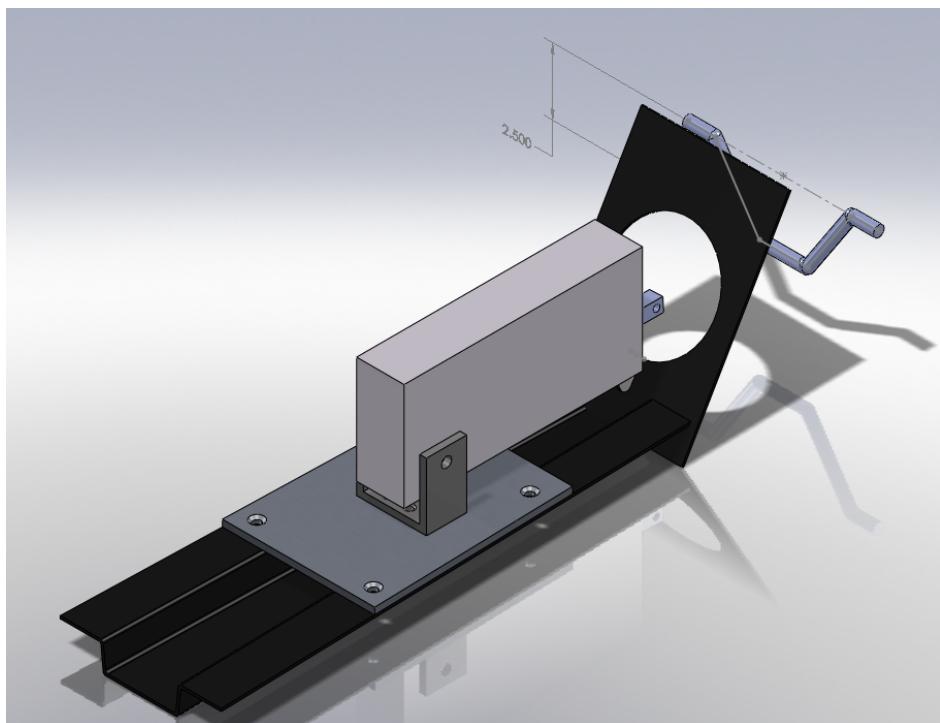


Fig. 17. A CAD rendering of the entire Diesel actuation system. The full pedal arm is not shown, but would extend from the left side of the center column of the vehicle.

power transfer shaft for the front differential is directly below two of the cupholder mounting holes. To minimize vehicle modifications, four open-end knurled rivet nuts were inserted into the four cup holder mounting holes, which must be expanded to a size of 0.4" to fit. These nuts are secured with a rivet gun, and 1/4-20 cap head screws are used to secure the mounting plate.

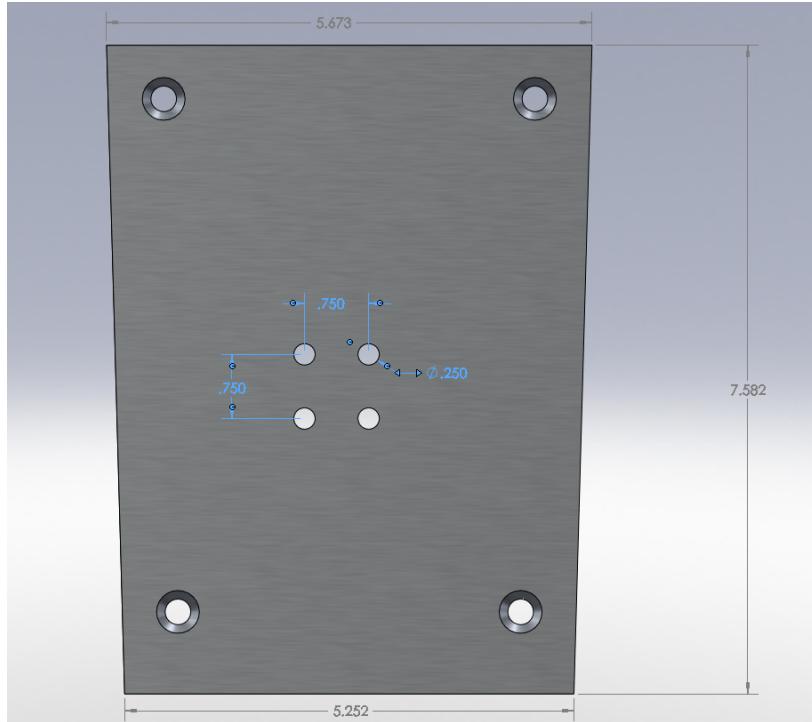


Fig. 18. A cross section of the adapter plate for the actuator mount.

The four holes in the middle of the panel are threaded for 1/4" x20 bolts. These allow for the U-bracket to be attached to the mounting plate. This bracket is cut out of a 1/4" sheet of aluminum using the abrasive water jet cutter, with 4 .255" holes for the 1/4-20 bolts which attach it to the mounting plate, and two .380" holes which will be 3.15" above the mounting plate when the bracket is bent.

A 0.375" hardened steel drill blank is fitted through the U-bracket and the mounting hole in the actuator. The drill blank is restrained using two shaft collars, and the actuator is kept from moving side-to-side (and potentially side-loading) using one additional shaft collar.

Actuating the Pedal Arm To connect to the pedal arm, the actuator uses a 3/8"x16 threaded rod, which connects to a locked-down split hanger collar (McMaster part #3023T23), held in place by a locknut. The collar uses a delrin bushing to reduce friction on the pedal arm, and to account for any misalignment between the hanger and moving shaft. Since the

PQ Actuators are built with a 7/16" x 20 threaded actuator arm, a threading adapter was machined out of a section of brass rod.

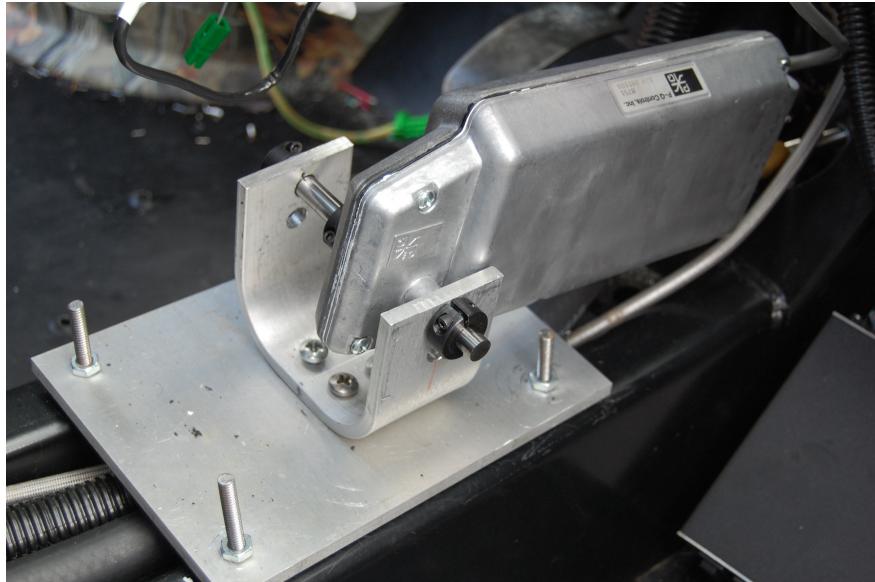


Fig. 19. A view of the entire accelerator actuator system.

Performance Because the return force of the unpowered linear actuator is greater than that of the unmodified diesel pedal return spring, an additional, stronger spring has been added in series, which means the force to fully depress the pedal is 19 now pounds, with an action arm of 6" and a throw of 6", depending on the position of the speed regulator bolt. The return spring mount is only 2.5" long, so the force required to depress the pedal by pushing on the return spring mount is approximately 45.6 pounds. Since our actuator can produce up to 90 pounds of force at a throw of 4", this performance requirement is well within its operating range.

6.4 Steering Actuation

The steering actuation uses a pre-existing power steering assist kit that is modified to meet project requirements. This steering plan is similar to a system prototyped on the summer 2009 research vehicle, the Yamaha Rhino, and is designed by the same manufacturer. The kit required two main modifications: the addition of a more robust controllable motor and the creation of some sort of feedback method from the device. The stock motor from the power steering kit does not provide feedback and lacks sufficient power to control the vehicle in worst case conditions. Both of these issues are addressed by installing a larger motor with a built-in encoder. This solution requires several additional changes to the pre-existing kit:

its position in the gator, the motor attachment to the rest of the power steering unit, and the motor interface with the power steering unit to deliver power.

The motor has to deliver 150 in-lbs at the steering wheel (twice that for a sizable safety margin). The motor is limited also to 5A at 12V to remain compliant with the cRIO 9505 motor controller. The 150 in-lbs requirement was determined through experimental measurement, by turning the steering wheel from left lock to right lock and back while resting on the grass at a standstill. The motor chosen to meet these specifications is provided by Potomac Electric, and has a 4" diameter and a length of 8" with a 5/8" diameter shaft.

In order to rotate the power steering kit into a more favorable position, an adapter plate was designed to fit between the power steering assembly and the brace that attaches it to the chassis. Keeping the motor in the original orientation would have required major modifications to the chassis and dashboard in order to fit the motor.

Inclusion of the larger motor required development of a method for attaching it to the rest of the unit. The original motor attached to the power steering housing through an aluminum heat shrunk collar with through holes for two bolts. The diameter of the new motor made a direct analog unfeasible, requiring design a larger collar which offset the motor enough to allow use of the two existing bolt holes from the previous motor mount. Figure 20 shows this adapter.

The original motor delivered power to the steering kit through a non standard spider coupling, requiring design of a corresponding piece that fits on the new motor. The motor shaft is attached to the new spider coupling with a keyless bushing (Figure 21 shows the new coupling).

When power is cut to the motor the operator has full control of the vehicle and is able to back-drive the motor.

In addition to the various adaptor pieces the motor is secured by a harness on the back of the motor. The harness is shown in Figure 22 and consists of a 4" rubber-coated u-bolt, two machined plates, a threaded rod, and a split clamp hangar which attaches to pole running through the dashboard.

6.5 Wheel Encoder Mount System

To provide velocity feedback for vehicle control, the Gator utilizes an optical wheel encoder. The wheel encoder mounting consists of an ANSI 35 sprocket and chain system pictured in Figure 23.

One sprocket is attached to the optical encoder shaft, and the other is driven by the driveshaft located under the utility bed on the right side of the Gator. The driveshaft sprocket was manually split and fitted with socket head cap-screws in order to mount it around the shaft without requiring major disassembly of the Gator platform.

The encoder itself is bolted to a length of steel L-channel, which is in turn bolted to the frame of the gator using a pair of tap rivets. Using a sprocket chain allows the system to accommodate the slight yet inevitable misalignment that results from this assembly.

Because of the tendency of sprocket chain to stretch over time, all sprocket chain systems require some way to dynamically tension the chain. In this system, tensioning is accomplished

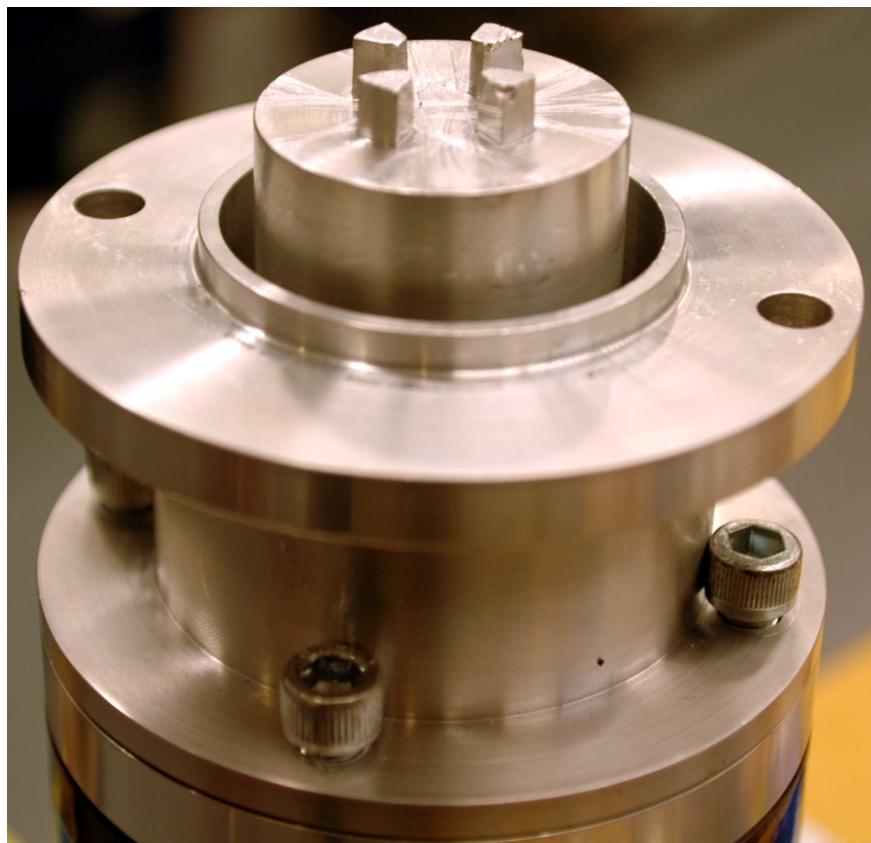


Fig. 20. Power steering motor mount.

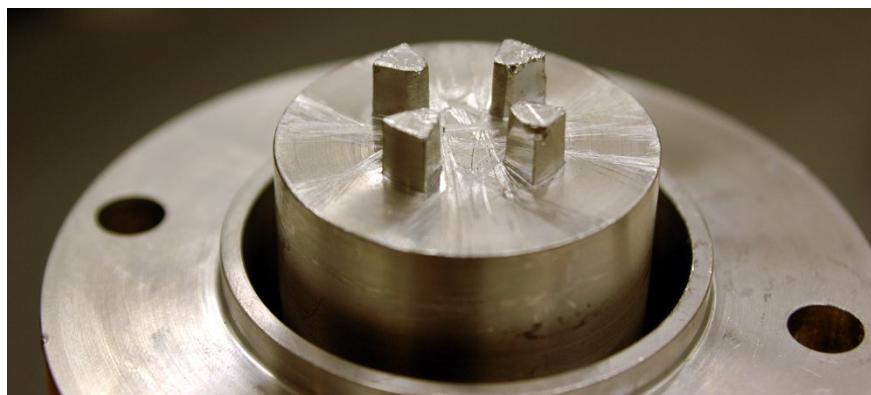


Fig. 21. Our manufactured spider coupling adapter mounted on the motor.



Fig. 22. Rear of the steering motor with additional support shown

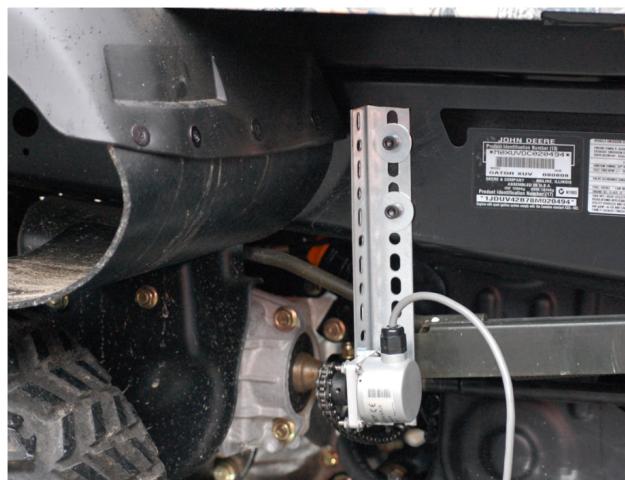


Fig. 23. Wheel encoder sprockets and mounts.

through the use of a flexible polymer sprocket ring designed to provide tension to roller chain systems without extra mounting hardware. Because of its natural elasticity, the polymer sprocket ring exerts a radial force on the sprocket chain loop, keeping the chain in tension as it spins.

Since velocity feedback will be derived from the rotary motion of the encoder, it is important to understand the ratio between encoder ticks and vehicle speed. Since the chosen encoder has 500 ticks per revolution and the sprocket system results in 16:9 gear ratio between the vehicle drive shaft and the encoder shaft, the encoder will register approximately 890 ticks per revolution of the drive shaft. There is a one-to-one correspondence between rotations of this shaft and rotations of the tires, which are approximately 23 inches in diameter. Using this as a baseline and calibrating further using field testing, it was possible to establish a ratio of 3000 encoder ticks per foot of forward vehicle movement.

Since the encoder sprocket chain system is mounted low and on the exterior of the vehicle, it requires additional protection towards the ultimate goal of off-road missions. For this reason, a sheet-metal bash guard was designed and fabricated. The bash guard bolts to both the encoder mount L-channel and the vehicle chassis itself. The fully assembled system is shown in Figure 24

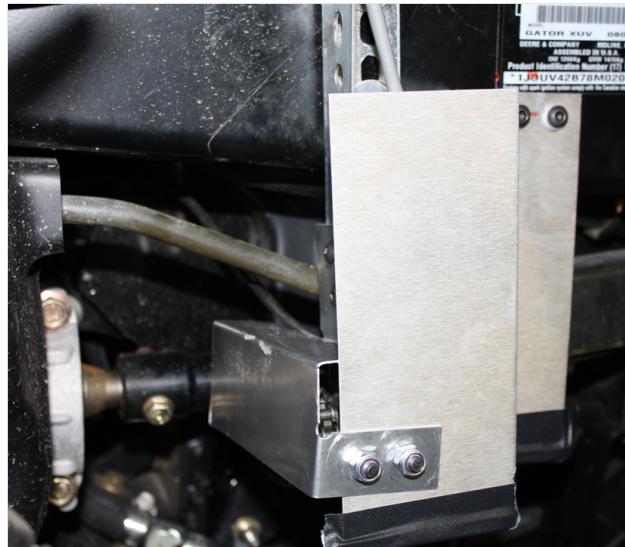


Fig. 24. Wheel encoder bash guard fully installed.

6.6 Generator Mount

The system's gasoline generator is mounted to the rear of the Gator's utility bed by two standard ratcheting tie-down straps. The straps attach to four 800-pound working load tie-down rings which have been installed in the base of the utility bed. The assembled mounting system is shown in Figure 25.



Fig. 25. The generator in its mount.

In preliminary tests on rugged terrain at high speeds, the mount has proven to provide sufficient stability. This mounting system was chosen for its simplicity as well as the ease of install and uninstall should the generator ever need to be replaced or removed for offboard testing. As an added benefit, this mounting scheme allows the preexisting rubber feet on the generator to provide shock absorption and vibration isolation.

6.7 Steering Limit Switches

Since the steering motor encoder provides only relative position feedback, it is important to have an absolute measurement of steering position so that the vehicle can automatically calibrate regardless of the positions of the steering at startup. In the Gator AGV, this is accomplished through the use of a pair of magnetoresistive proximity sensors mounted such that the steering rack-and-pinion will trigger one at each mechanical steering endpoint.

The sensors used are Cherry MP100502 Hall Effect sensors, which output 5 volts normally and drop to 0 volts in the presence of a magnetic field of sufficient strength aligned with the sensor. One such sensor is pictured in Figure 26.



Fig. 26. A Cherry sensor.

The sensors are mounted through two pass holes drilled through the cab behind the pedals in such a way that the sensing ends are aligned with the steering rack shaft. A cab view of the back ends of the two mounted sensors is shown in Figure 27. The mechanical design of these sensors allows their effective length to be easily adjusted using the built-in clamp nuts, which is important since the effective range of the sensors is quite short and therefore precise axial positioning is required.

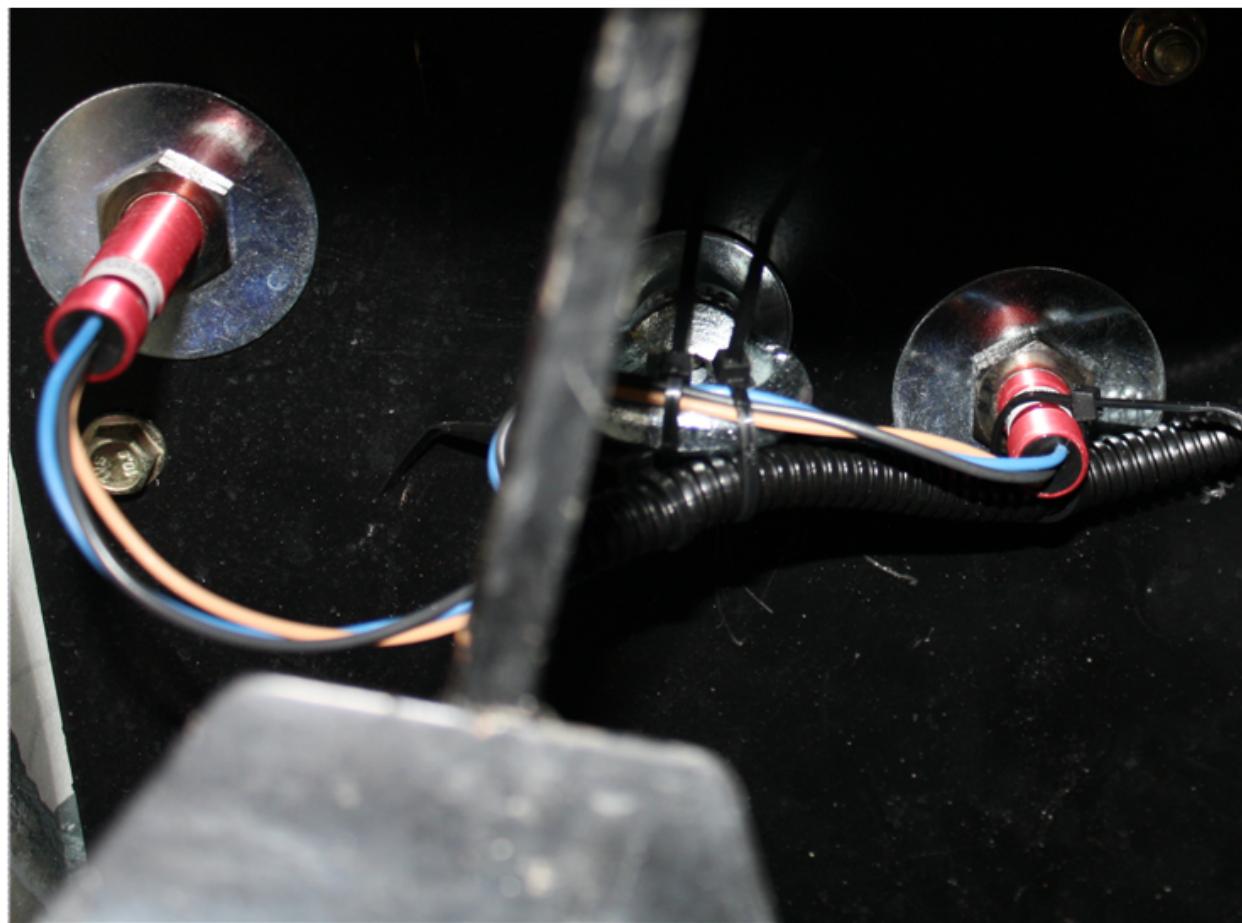


Fig. 27. Cab view of the back ends of the installed limit sensors.

To provide magnetic pickup points along the steering rack, two 3/8th inch ring clamps are attached to the rack itself, and a small circular magnet is affixed to the end of each clamp. This allows the magnetic pickup points to be easily adjusted, which in turn sets the effective position of each limit sensor as well as the steering range that they encompass. Also note that the Hall Effect sensors are only sensitive to one magnetic pole, so proper orientation of the magnets is important.

Ideally, the magnetic pickup points are adjusted so that they will trigger the limit sensors precisely inside the mechanical limit of the steering. This allows the automatic steering

calibration routine to run regardless of the initial position of the steering column, as a turn in a certain direction will always hit the same sensor. Additionally, this allows the limit sensors to double as a shutoff triggers if a problem develops with the steering control software during a run, while at the same time preserving the maximum useable range of steering.

6.8 Monitor Mount

To mount the Nortec SUN-1710-P daylight-visible monitor in the cab, we replaced the original mounting support, a 1.315" steel bar, with an extended 19" section of steel pipe. This way, the monitor could be mounted using its original mounting hardware: two 1/4"-20 dome head bolts – one to hold the bracket for the monitor, and the second to adjust monitor angle. The mounting holes for these bolts were drilled 6" and 6.5" from the top of the pipe section, so that the monitor would clear the windshield, and to allow for the current mounting of the stereo camera. The latter hole was tapped.

To support the monitor and mounting bracket, a triangular section was waterjet cut from 1/4" aluminum. This plate connected to the steel pipe section via three 1/4"-20 screws, and to two vibration damping clamping hangers (McMaster-Carr part #2615T14) via two 3/8"-16 screws. These hangers clamp on to the exposed 7/8" pipe-section supporting the Gator dashboard.

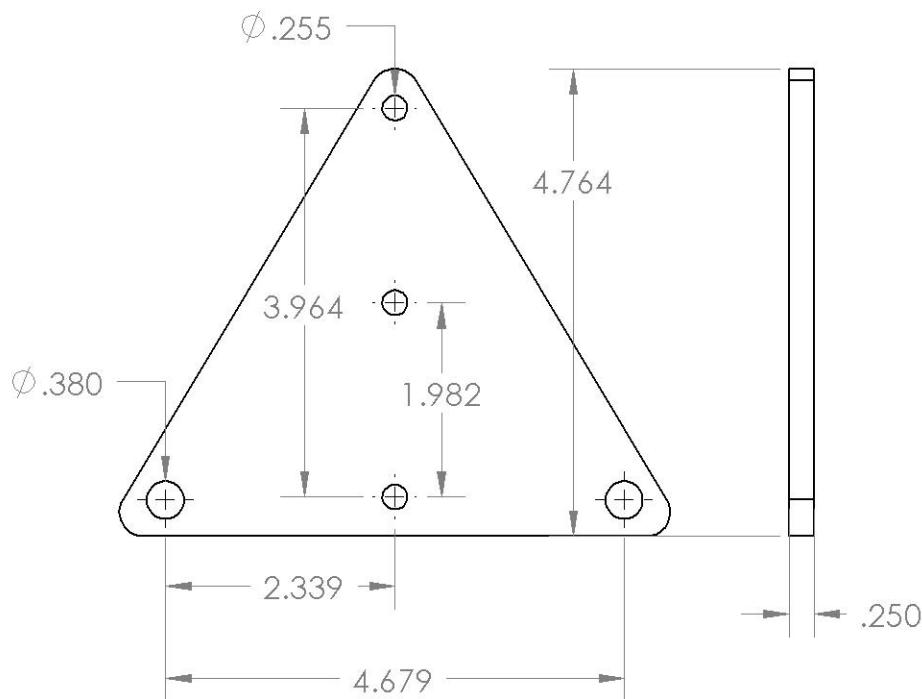


Fig. 28. Monitor mount adapter plate spec. All dimensions are in inches.

Finally, to stabilize the monitor, a 3-11/32" strap hinge was used to connect the pipe to the dashboard. This hinge does not bear any downwards load, but rather stabilizes the monitor against sideways vibrations, and therefore the 1/4" self-tapping screws which connect the hinge to the dashboard are sufficient.

6.9 Cabin Wire Routing

There are four areas where cables are routed into the cabin of the Gator. The first two passthroughs are previously existing holes, located behind the driver-side seat and designed for 3/8" conduit. Conduits for communications with the cabin e-stop button and linear actuator are routed through these holes, and follow pre-existing conduits running along the left roll bar of the Curtis cab, with their last zip-tie fastening in the center of the hood, under the dashboard.

The power cables for actuators and steering motor are passed through another pre-existing hole, this one located behind the battery compartment, behind the driver's side seat and directly above the FWD crankshaft located on the passenger side of the vehicle.

The final pass-through is located between the two seats, a half inch above the seatbelt mounting brackets. It is not pre-existing, and is drilled to be 1.45" wide, to allow for a VGA cable for the monitor to be passed through. Conduits for monitor power and signal, as well as USB and IEEE 1394 connectivity and signal from the Cherry magnetic sensors (used as end-stops for steering and located in the driver side firewall) in the cabin are passed through this hole.

6.10 GPS mount

To fulfill our goal of GPS-guided navigation within a parking lot, we needed to mount a GPS antenna. The mount had to be less than 6" in height, as it would have to clear the top of the parking container with 10" of entryway clearance while mounted on top of a roof-mounted pegboard approximately 2" in height.

The GPS mount consists of a 2"W x 2"L x 1/4"T sheet of aluminum drilled to accommodate 4 3/8" holes spaced to correspond with 4 3/8" holes in the pegboard, and one 5/8" hole in the center. Four 3/8"x16 standoffs are used to elevate the plate, to accommodate for the head of a 5/8"x11 bolt which is attached threads-up to the plate using a nut with locking washer. The antenna is designed to screw onto the large bolt and is then held on using a locknut with built-in serrated washer. Altogether, the unit measures approximately 5.5" in height.

6.11 Electronics Enclosure Modifications

The electronics enclosure holds all of power and computational electronics, and required three modifications: cooling vents, power and signal wire ports, and a system for mounting the electronics inside the box.

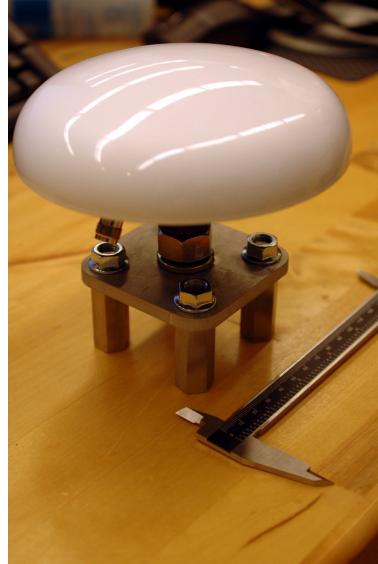


Fig. 29. Our GPS Mount.

The cooling system uses two 120mm computer case fans and two weatherproof vent covers (such as those used for hot air exhaust in houses) mounted on two 5" vent holes. These exhaust vents will be weather-sealed during installation using outdoor caulk.

Signal wires are routed out of a third 5" weatherproofed vent on the bottom of the box, as are all power wires.

The electronics plate is designed to be mounted on a supporting frame using several bolts, to reduce plate flexure and potential damage from shocks from rough terrain. The supporting frame is fastened to the sides of the electronics box using screws that have been weatherproofed with silicone sealant.

6.12 Electronics enclosure mounting

The electronics enclosure is mounted to the vehicle using two pieces of square channel high tensile steel with corrosion resistant coating bolted to the bed of the vehicle. The box is then attached to the steel with four vibration-damping sandwich mounts allowing for a smooth ride. This simple system is robust and requires relatively little assembly.

7 Vehicle Software Architecture

The Gator's software is based on the Olin Intelligent Vehicles Lab's LabVIEW Robotics Operating System (LvROS) developed during the summer of 2009. The architecture is designed for use on a variety of mobile robotic platforms. Substantial resources have been committed to improve the logging/replay and embedded portions of the system. Here a brief overview of the system's goals and code architecture is given, followed by a more detailed examination of

the new replay functionality. The reader is encouraged to peruse Appendix E, which details how individual modules are added to the system.

7.1 Goals

The vehicle's software system needs to support all autonomous vehicle operations, ranging from basic motion to complex algorithms for path planning and obstacle avoidance. This overarching goal, combined with the desire to be able to easily modify and update the software, leads to a number of specific software-level parameters for the system:

- Parallel processing
- Hierarchical code structure interface
- Code reuse
- Not necessary to modify existing, tested, working code

7.2 Parallel Processing

Parallel processing is a requirement of all modern robotic systems. A single-threaded system would have difficulty meeting the timing requirements of the multitude of sensors, thinking processes, and actuators on board. Moreover, in a single-threaded solution, failure of one system may cause loss of the entire computation engine at once. This is addressed by a multi-threaded system that allows different processes to run at different rates, supporting a LIDAR gathering data at 75Hz as well as a slow stereo vision system capable of only a few frames per second.

LabVIEW provides for multi-threaded environments and safe communication between processes, satisfying timing and robustness requirements with little extra complexity. Specifically, LabVIEW while loops, queues, and notifiers (see Appendix A for more details) are used to perform safe multi-threading and interthread communication.

7.3 Hierarchy

To manage the necessary complexity, the system has a hierarchical code structure. In addition to reducing complexity, this structure allow for code reuse and helps prevent the modification of working code. The LvROS system separates code modules first into broad categories and then deeper into sensor drivers, communication components, and user interface systems. The top level includes the following systems:

- Sense
- Think
- Act
- Log
- Networking
- User Interface

Below this level each system is further subdivided. *Sense*, for example, is divided into the different sensors we have available (ex. LIDAR, INS, etc.). Those are optionally further subdivided into drivers and processing systems. For the sake of brevity, the entire system is not detailed here. Figure 30 shows the system layout and Figure 31 shows a more detailed view of the sensing system's layout.

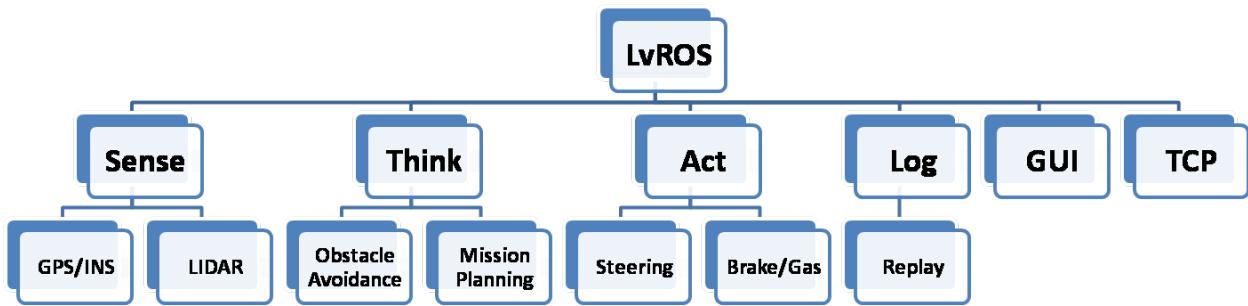


Fig. 30. LvROS software hierarchy.

7.4 Example: Adding a Xbox Controller Driver

Here we go through the procedures necessary to add a new module to the software system. It is recommended that you read Appendix E (LvROS Overview) first for a basic understanding of the architecture and queue system.

Architecture of an Xbox Controller Driver To conform to our architecture style, the Xbox controller needs two main components. It “senses” the inputs (reads the USB game controller) and “thinks” about those inputs (transforms button presses and joystick motions to drive commands.) Thus, we will want two threads for the system: a sensing thread and a thinking thread. To communicate, these threads will use queues. Finally, we must integrate with the main driving system, which requires modification of our “acting” system.

Sense Block The sense block of a Xbox control driver must do two things:

1. Read gamepad state
2. Enqueue gamepad state

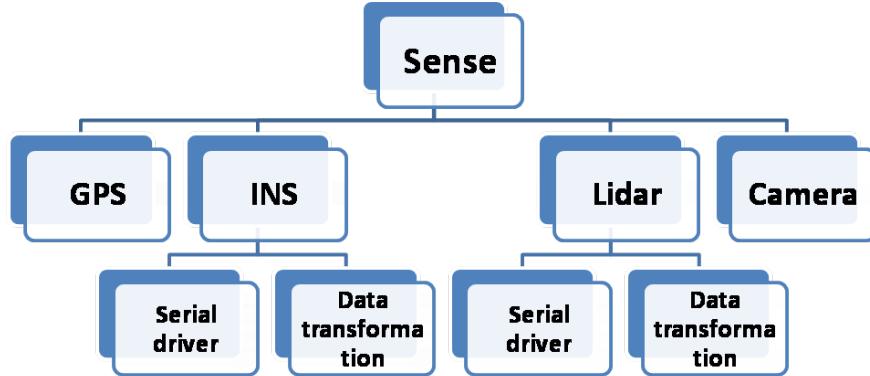


Fig. 31. LvROS sensing hierarchy.

Here we examine these in order. To read the gamepad state, we'll use LabVIEW's built-in joystick control VIs. Open the joystick reference with *Initialize Joystick* (in Connectivity → Initialize Joystick). Then use *Acquire Input Data* (also in Connectivity) to read the game controller's values.

Now we want to translate these values into something we can enqueue. To do this, we must define a type-definition for our new queue. To make a type-def, create a cluster on a front panel, put the appropriate elements in it (such as X button, Z button, Left Stick Y, etc.) and then right click on the cluster's border, select Advanced → Customize... (Figure 32).

With our typedef defined, we need to translate the values coming out of the Read Data VI into our format. A simple unbundle and bundle will do the trick. We put this in its own SubVI to make the code a bit cleaner. Our version is shown in Figure 33.

With SubVI reading the Xbox data, we're ready to enqueue it! But first we need to create the queue we are going to use. Here we call it the "XboxGamepadQueue". To initialize it, we open *InitSense.vi* and add another instance of *InitQueue* (Figure 34). Note that we could use a notifier to make this more efficient, but we don't cover that here.

With our queue and typedef ready to go, we can now finish our sense block. All we need to do now is create a while loop that continuously reads the gamepad's values (using our SubVI) and enqueues them onto the XboxGamepadQueue. Note that it is important to use a *lossy* enqueue, as noted in Appendix E. Our final sense VI is shown in Figure 35.

Think Block Now that we have our data on a queue, we need to interpret it. To do this, we'll add a thinking block that reads our data queue, figures out what to tell the steering wheel and accelerator pedal to do, and outputs that to a queue.

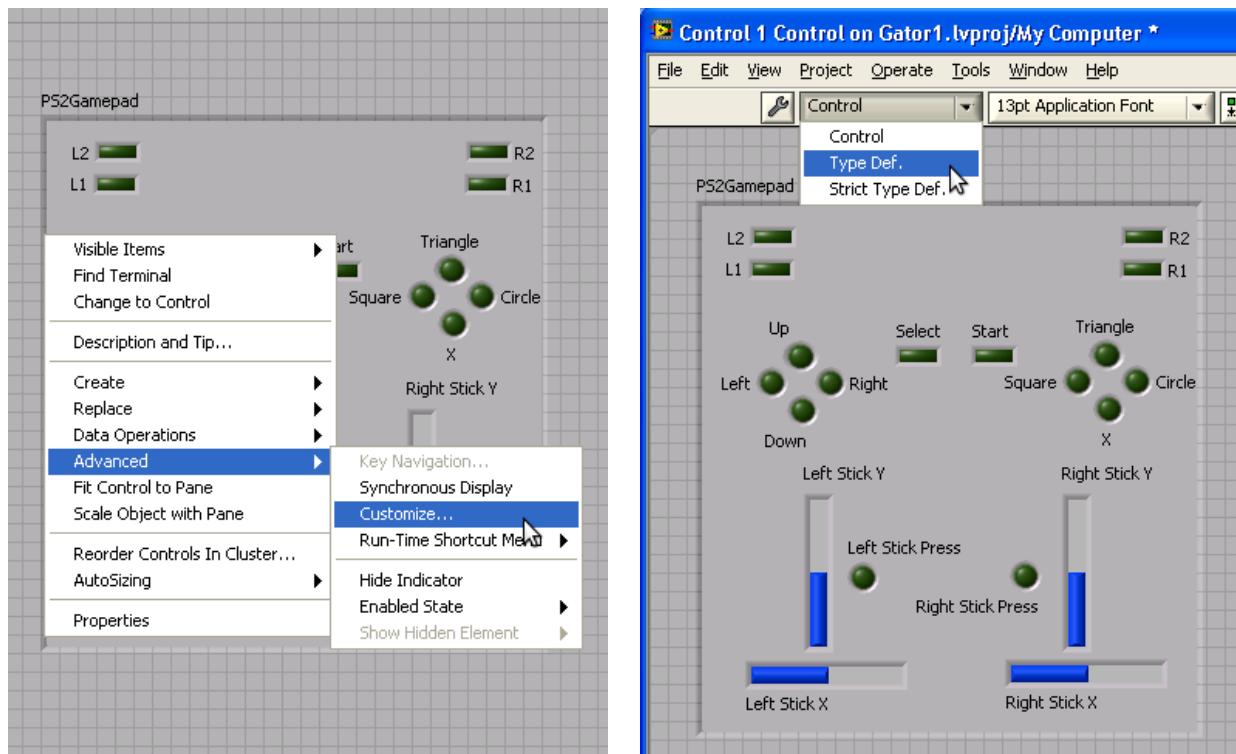


Fig. 32. Creating a type-definition.

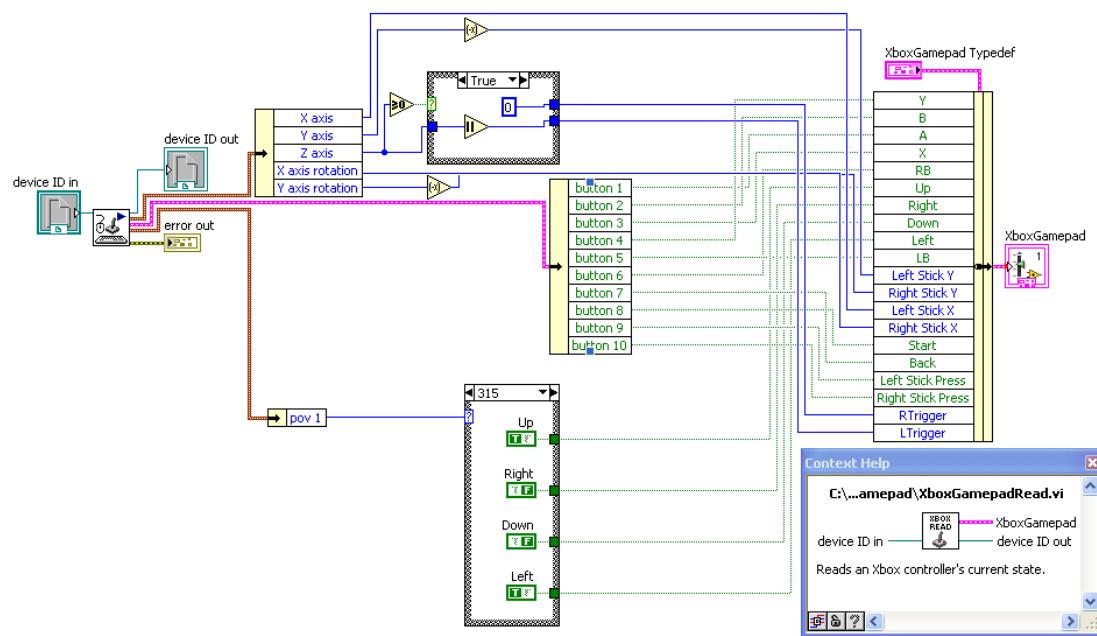


Fig. 33. `XboxGamepadRead.vi` function. Here we read the values coming out of LabVIEW's joystick functions and put them into our typedef. The context help box shows this VI's inputs and outputs.

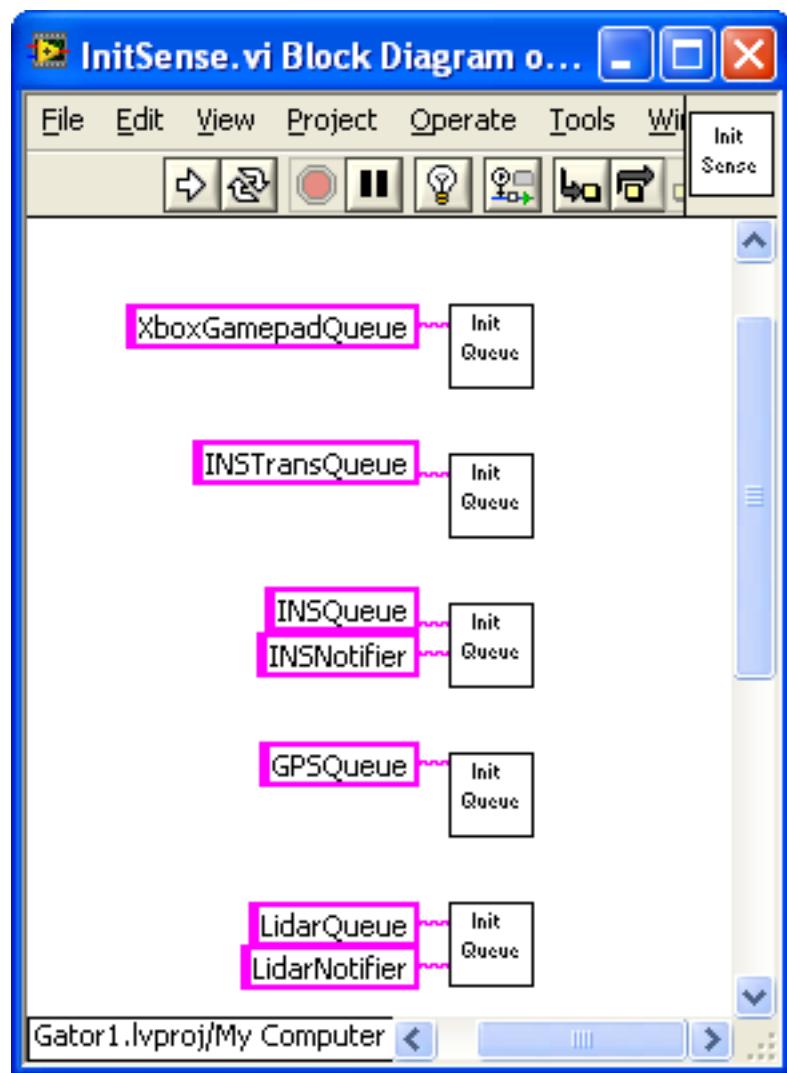


Fig. 34. Adding the Xbox Gamepad Queue for initialization in the *InitSense.vi* block.

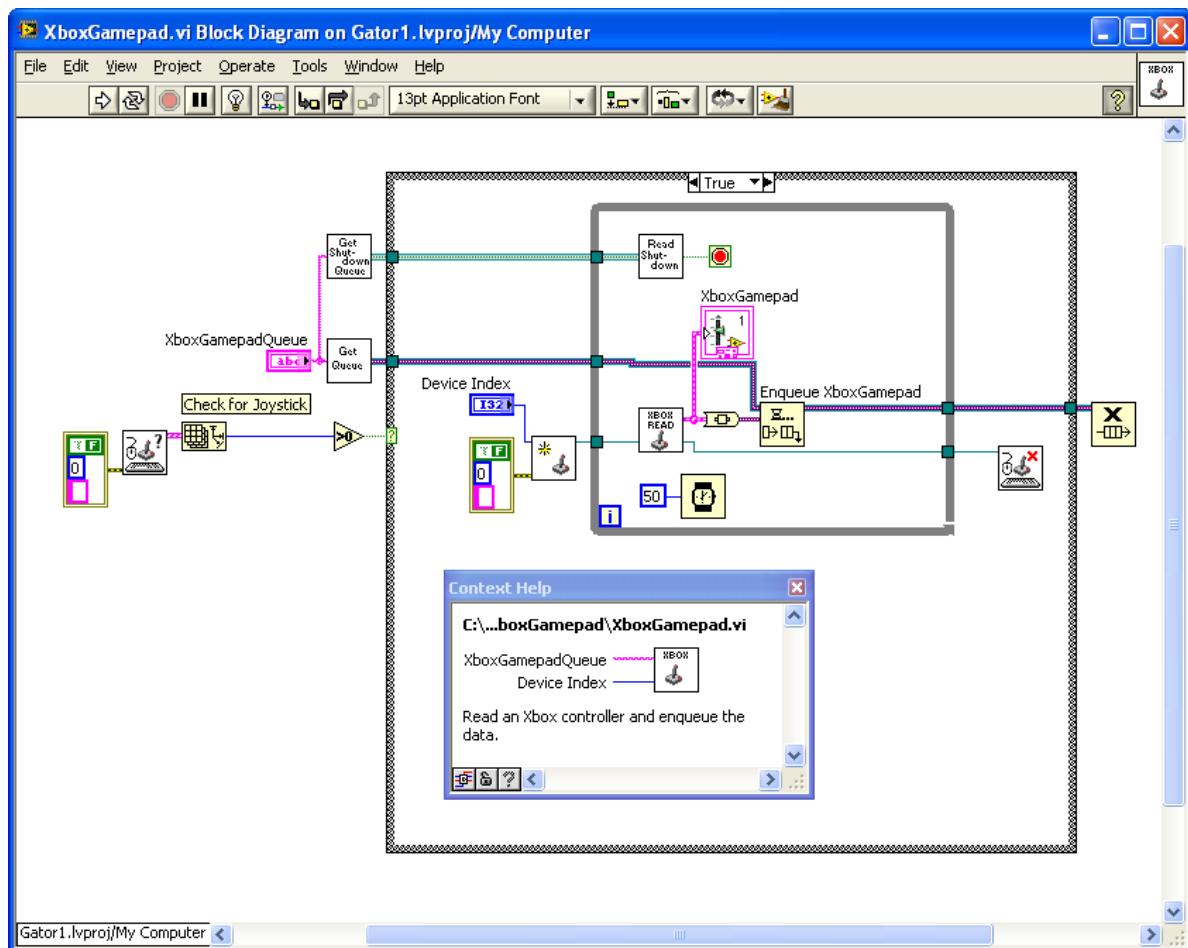


Fig. 35. Our full Xbox Sense block. Note that the reading functions are in the Xbox Read block, which outputs our typedef with data filled in and ready to be enqueued. Also note that we use the shutdown mechanism as described in Appendix E.

This is its own separate thread, so we'll create a new while loop that reads the data. To do this, we use *GetQueue.vi*, giving it our queue's name (XboxGamepadQueue). We then *Preview* the queue, so we read the data without removing it from the queue. With the data, we must decide what to do. The particular logic is not important here, but you can see we are converting the left joystick's position to a turning command, the trigger buttons to speed controls (left trigger = set speed slower, right trigger = set speed faster) and the B button to brake (set speed to zero). To send this data to the robot, we use another queue which we call the "XboxControlQueue". Note that we need to initialize this queue just as we did for the XboxGamepadQueue.

The typedef for the XboxControlQueue is already defined for us, because it is a standard driving command queue. Thus, we use the *Vehicle Control Typedef*, convert to a variant, and lossy enqueue the data. Figure 36 shows our think VI.

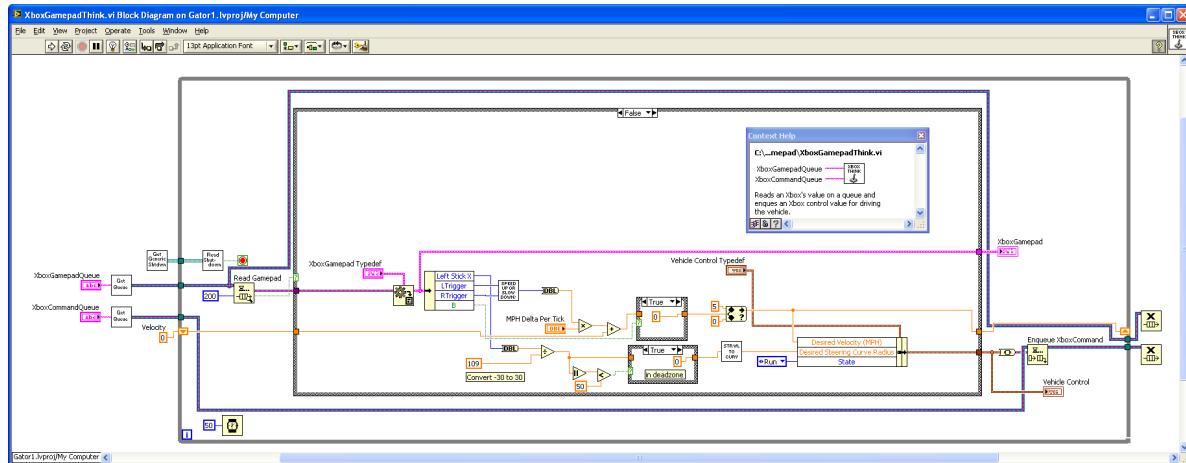


Fig. 36. Our full Xbox Think block.

Integration Finally, we must integrate the XboxControlQueue into the rest of the system. To do this, we first add a new entry to our program select box (Figure 45). We open the typedef for this and add the item. Now, inside *ActMain.vi*, we find *SelectDriveCommand.vi*. In here (Figure 37), we add our new queue in the pattern of the existing ones:

- Add the preview queue block.
- Add a case statement (right click → Add Case for Every Value).
- Add Pass if Ok block.
- Route data wires.

Testing We have completed the integration of the new Xbox controller! Now turn the car on (Appendix A) and test it!

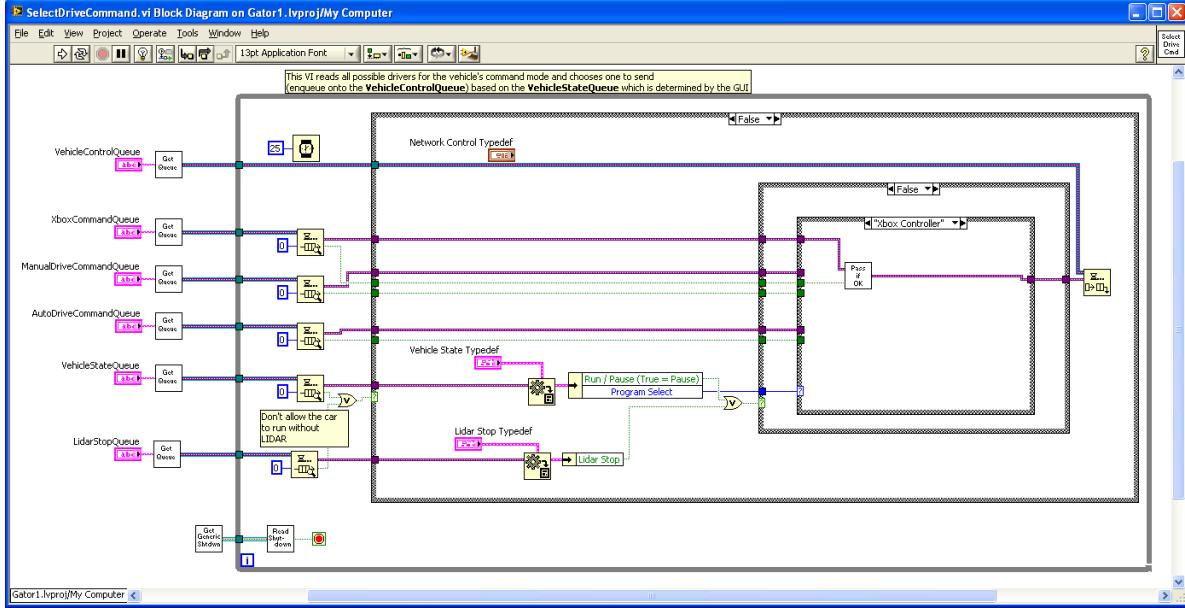


Fig. 37. Select drive command block with the new Xbox Controller added.

7.5 Hybrid FPGA / Scan Mode Interface

LabVIEW supports two modes for the cRIO: *scan interface* and *FPGA interface* modes. Scan interface allows the user to operate supported cards without having to compile FPGA code. FPGA mode allows for faster, more in-depth access at the cost of increased code time.

We chose to use a hybrid solution, using the NI 9401 digital I/O card in scan mode and the NI 9403 and NI 9263 cards in FPGA mode. A hybrid mode requires slightly longer for an FPGA compile, but allows us to use the built-in quadrature encoder code on the NI 9401. We need high-speed control of the 9403 to use it with the Victor speed controller, so that requires FPGA access.

7.6 cRIO Cards

NI 9041 Digital I/O We use the 9401 exclusively to read the steering and speed encoders. Its built-in scan mode quadrature encoder mode allowed us to work around a difficult bug where we would lose encoder counts in FPGA mode. This also simplifies the code we have to write. Table 3 shows all the connections currently in use on the 9401. To continue to use scan mode, no more encoders can be added.

NI 9403 Digital I/O The NI 9403 supports substantially more digital I/O at a slower rate than the 9401. We use the 9403 to run the Victor speed controller's hobby servo PWM input as well as perform all of the other digital I/O for the system. Table 4 shows all of the connections currently in use on the 9403. May more digital IOs are possible for this card.

Name	Wire Color	Connector Color	Pin #	DIO#
Speed Encoder A	Yellow	Green	14	0
Speed Encoder B	Orange	Blue	16	1
Speed Index	-	-	-	2
Steering Encoder A	Brown	White	19	3
Steering Encoder B	Purple	Green	20	4
Steering Index	-	-	-	5
Ground	Black	Brown	1	-

Table 3. NI 9401 Connections

Name	Wire Color	Connector Color	Pin #	DIO#
E-stop	Green	Orange	1	0
V48 Status	Blue	Purple	2	1
V24 Status	Purple	Green	3	2
V12 Status	Orange	Blue	4	3
Victor PWM	White	White	5	4
Ground	Black	Black	10	-
Limit Switch Right	Gray	Blue	6	5
Limit Switch Left	Yellow	Yellow	7	6

Table 4. NI 9403 Connections

NI 9263 Analog I/O The NI 9263 Analog output card controls our gas and brake actuators. Those actuators must always come online at the same voltage (we chose 1V) for consistent results. The card has four outputs, of which we are using two.

7.7 LIDAR Safety System

Obstacles in the path of the Gator are detected by a SICK LIDAR rangefinder mounted on the rooftop sensor rack. The LIDAR is 6.7 feet from the ground, and inclined -12.7 degrees, giving it a maximum effective range of approximately 30 feet. It is configured to sweep 180 degrees with one scan per degree, returning 180 distances per scan. These data are truncated to remove 20 points from each edge, as these points are almost directly out to the side of the vehicle, do not touch ground within the maximum range of the LIDAR, and are therefore usually mostly noise. From here, the distance/angle pairs are converted into a series of 3-dimensional vectors relative to the vehicle.

The point detection vectors are then used to populate an adjustable-resolution occupancy grid describing the areas around the vehicle. Each point is tested to see if it above the obstacle height threshold (1.2 feet), and if so, the location of this obstacle is converted to the nearest position on the grid, and that point's certainty value is incremented. If the obstacle is not detected on the next pass, that point is decremented by a fadeout value (currently 0.2) until either an obstacle is detected or the certainty value reaches 0. If the obstacle is outside the maximum grid size (set to be larger than the effective LIDAR range), it is ignored.

From here, the obstacle detection algorithm takes effect. The algorithm divides the map into four zones:

1. The “safe” zone – the Gator can safely travel, even if obstacles are present here.

2. The “in-path” zone – represents the projected path of the Gator, plus some padding; the Gator can travel here, unless an obstacle exists in 3 absolute halt distances (currently 24 feet).
3. The “driving stop” zone – a conical section in front of the Gator; the Gator can travel here, unless an obstacle exists within 2 stopping distances (from current speed).
4. The “absolute halt” zone – a conical section in front and over the quarterpanels of the gator; the Gator must be at a halt if any obstacles exist within this area (currently 8 feet away).

Stopping distance from current speed is calculated from current vehicle speed and a deceleration constant, taken from empirical data. Absolute halt distance is user controlled, and set to a distance deemed safe by the team Safety and Ethics Officer.

8 Logging/Replay System

The logging/replay system offers us substantial capability for debugging sensors and algorithms, supporting offline coding, and providing a limited simulation-like environment. For these reasons, time has been invested to make the system useful and robust. The communication and threading system provides a natural layer for connecting to logging and replay, allowing the system to support almost any sensor or data type immediately. The logging mechanism and formats are detailed here, and the replay implementation is explained.

8.1 Logging

Generic sensor and data logging presents a significant challenge. How can the system, without any prior knowledge of sensor or data type, log in a format that can replay automatically? To do this, all sent data is required to go through LabVIEW’s queue structure using a *variant* data type. The variant type simultaneously encodes the data type and data, allowing the system to write all necessary information to a file and recreate those data types during replay. Moreover, using the queue structure allows us to read the data flowing between system components non-destructively and without delaying data transmission, allowing for seamless operation with and without logging.

Logging Threads All data-transmission queues are logged by default to allow for easy development, but support user-exclusion of data deemed unimportant. For each queue that is set to log, a new thread is spawned, either *LogQueue.vi* or *LogImageQueue.vi*, that logs only that queue to disk. Thus, if five queues (often associated with five threads) are being logged, there are an additional five threads running for logging. If one of these logging threads fails for whatever reason, logging capabilities are lost only on that thread and do not impact the other logs or robot operation. Thus, in the ideal case, no single thread can corrupt the entire system’s log and no logging thread can cause a total system failure. However, since all threads are currently running on the same hardware and under the same LabVIEW instance, a catastrophic failure in one could cause failures in the others.

XML Log Format The ideal logging format is compact, human readable, and easy to parse. Given these constraints, the team decided that on a development vehicle, the emphasis should be on a system that is human readable and easy to parse over a system that is compact, leading to an XML-based LabVIEW generated format. A known bug in the LabVIEW XML variant converter¹ causes variants to unflatten as empty strings, so the system must first flatten data to a string, then convert to XML. Unfortunately, this workaround produces a difficult to read but easy to parse file format.

To the best of our knowledge, this variant-based system works for every LabVIEW data type except images. To log images, images are written to disk as PNG files and also write a companion XML variant file. This produces a large number of individual image files that are human-viewable and can also be replayed with only slightly modified replay code. The combination of variant and image logging allows seamless and automatic logging of every type of data.

8.2 Replay

Overview To replay data, the existing system must be fooled into thinking whatever component that is being replaying is actually online and producing data. This action requires 1) disabling the actual component being replayed, 2) reading data from the appropriate file, and 3) putting those data onto the appropriate queue. When the system starts, the baseline assumption is that no components will be replayed and all available sensor and data-processing threads are brought online. When the user selects component(s) to replay, shutdown commands are sent to just those threads, taking them offline. For each file to be replayed, a new thread is brought online (*ReplayQueue.vi* or *ReplayImageQueue.vi*) to read the log file, enqueue data to the requested time, and report its current position on another queue.

One difficulty with replay is that not all queues have the same time-resolution, start, and/or end dates. For example, a LIDAR will have a much faster update rate than a camera, and the camera may take a long time to come online. Thus, when replaying data, the LIDAR will start while the camera is still offline. To provide a simple interface, the user can request one single time which all replay threads attempt to move to. If a queue cannot move to that time because, for example, the time is before data starts, it will move to the closest available time. Each queue displays its current timestamp so the user is aware of any discrepancies that might exist. Figure 38 shows this interface.

Random Access The replay system currently does not provide random access to data elements. To reach an element, the user must “Play” all previous data points in the log file. This is a concern when attempting to review and debug using logs from long operations. We are considering creating a post-processing mechanism or adding further real-time processing to support random access, but have not yet committed the resources to implement this functionality. To support data rewind, a file location caching mechanism that is similar to

¹ http://zone.ni.com/reference/en-XX/help/371361B-01/lvconcepts/converting_data_to_and_from_xml

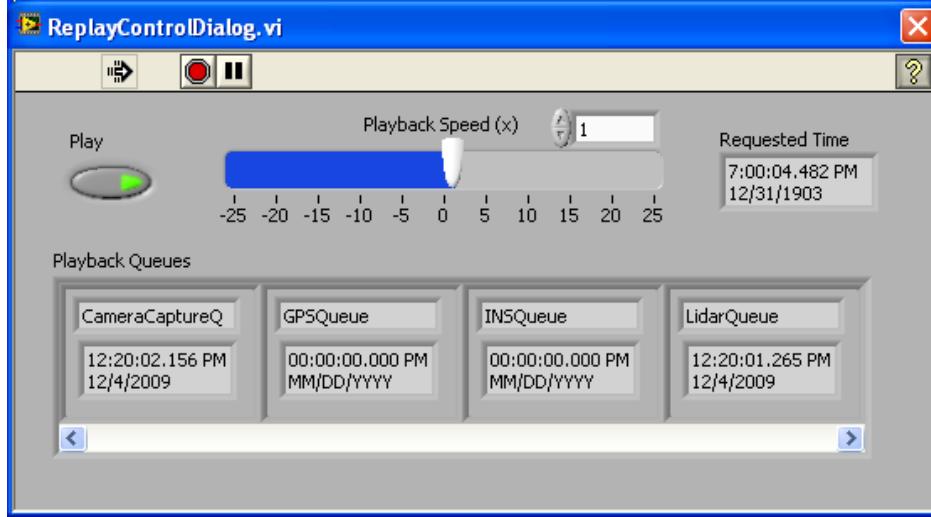


Fig. 38. Replay user interface.

the one proposed for random access is used. It is important to note that in both random-access and rewind situations, state-holding elements are affected, a consideration that the replaying user must take into account.

9 Controls

In order to achieve full drive-by-wire functionality, three main components of vehicle functionality must be in place: acceleration, brakes, and steering. These components are not all dealt with in the same manner, and therefore they have slightly different requirements in terms of control.

9.1 Acceleration

The accelerator pedal will be used to handle almost all of the vehicle's velocity control. This pedal will be actuated by a linear actuator, which is described in greater detail in the mechanical portion of this report. To provide the necessary analog control signal to the actuator, we are using an analog output cRIO module, the NI 9263. It is important to note that tolerable velocity error is relatively high (within a few mph), as the mission focus is waypoint following at relatively low speeds. As such, proportionally large deviations in velocity are not necessarily problematic. The velocity control system has two levels. A coarse-level controller gives an initial input to the system based on the steady-state characteristics of the vehicle. For instance, if we want to drive at 3 mph, it inputs a voltage to the vehicle that has experimentally been shown to cause it to drive at 3 mph at steady state on flat pavement. On top of this controller is a fine-tuning PID loop, which corrects for perturbations such as hills or bumps. This PID loop has saturated output, which prevents it from affecting

dynamics too much or too quickly, while still permitting velocity adjustments. Testing has indicated that this two-tiered system may lead to overshoot in the time response, because the PID loop and the coarse controller inputs are additive. For this reason, the coarse controller has been scaled down to provide only 85 percent of the requisite voltage. This still gives a reasonable time response without as dramatic overshoot effects, approximating a critically damped system. This is preferable to a quicker oscillatory response because velocity control is not as critical to our mission as position control.

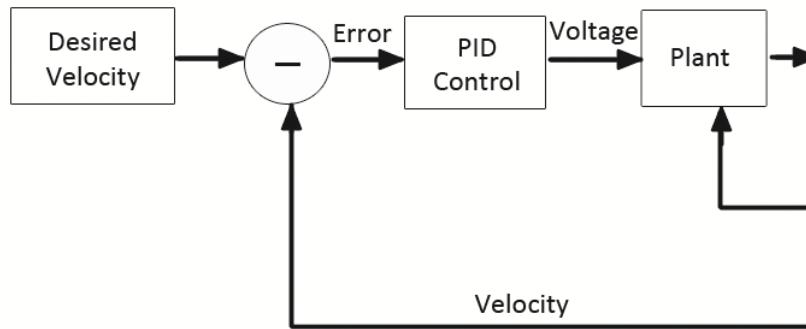


Fig. 39. Velocity control block diagram.

In order to determine the coarse controller characteristics, we measured the steady-state behavior of the vehicle. This was done by inputting various voltages to the actuator and measuring the corresponding steady-state velocity of the vehicle. The results of this are shown in Figure 40. Note that at pedal depressions below 4.5cm, the vehicle did not move at all, which motivated our use of a piecewise linear fit. This data is relatively linear, which allows us to avoid using lookup tables and instead simply extrapolate the describing function

$$V_{ss} = \begin{cases} 1.10d - 4.79 & : d > 4.5 \\ 0 & : d \leq 4.5 \end{cases} \quad (1)$$

This relation is described in Velocity Control.vi. Note that it will need to be modified in the event that the linear actuator is somehow reset, possibly via mechanical failure.

9.2 Steering

At a low level, the steering control is conceptually a little bit different from the velocity control. Because we are controlling a DC motor, the controlling PWM signal encodes motor

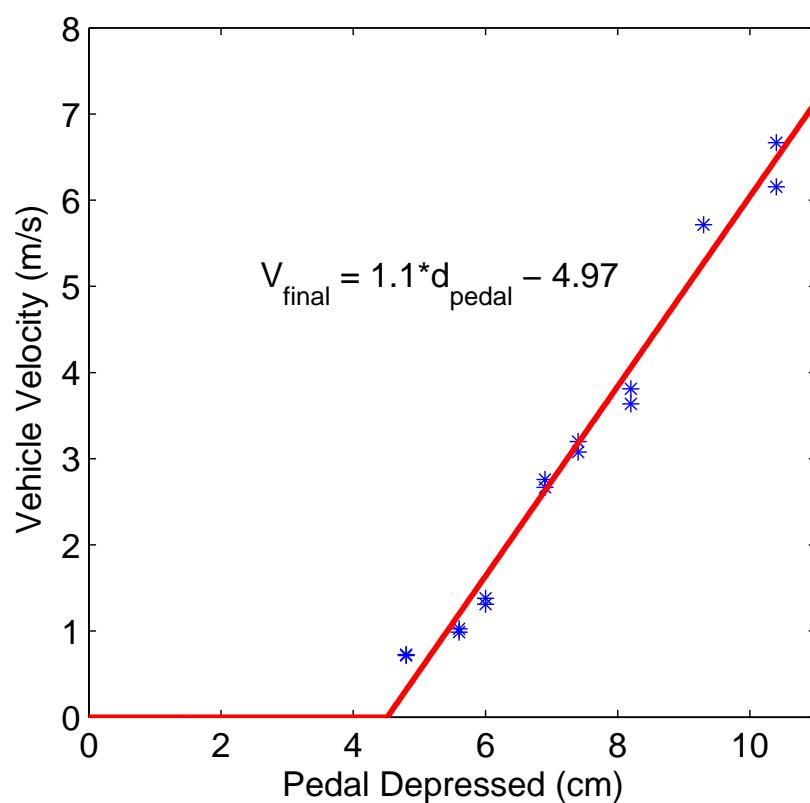


Fig. 40. Steady state vehicle behavior.

velocity rather than position. As such, in order to achieve position control on the steering wheel, we must reach a corresponding number of encoder ticks. This means that the output of our PI block yields motor velocity, which is integrated to get steering wheel position. This control signal will be produced by an NI 9505 cRIO motor controller module; with the appropriate drivers, we will be able to input a percentage and a heading, and the card will produce as output the appropriate PWM signal. Aside from this, the control is similar to the velocity case, though only PID control was used; we discarded the coarse controller. At an interface level, steering control is done in terms of radii of curvature (RoC). That is, the user inputs a desired radius of curvature, the VI converts this to a steering wheel angle/number of encoder ticks, and then the PID loop takes care of the physical control. A positive RoC indicates a turn to the right. In general, the control loop tends to achieve the desired steering wheel angle within two degrees, which is well within tolerance. A more likely source of error in the steering is our conversion from radius of curvature to desired steering wheel angle. This transformation was obtained in the following manner:

1. Turning the steering wheel to a specific angle, measured by the steering motor encoder reading.
2. Record start time.
3. Drive for long enough to trace out a reasonably circular arc.
4. Record end time.
5. Repeat 1-4 for as many angles as desired.
6. Use the GPS logs to plot each arc in meters. Remember to convert latitudes/longitudes into meters.
7. Use the resulting arc to determine radius of curvature. This is done with basic Euclidian geometry.

An example of this technique is illustrated in figure 41, and the relevant code is included in the appendix. The resulting relation was found to be

$$R = e^{(- 0.9171r + 7.02)} \quad (2)$$

where R is the radius of curvature and r is the natural log of the steering wheel angle. This relation can be found or altered as needed in Steering Motor.vi. We currently do not distinguish between left and right steering angles. This is something that would ideally be included in the analysis, as the loads on the vehicle are not necessarily symmetric.

9.3 Braking

Currently, the brakes will play a comparatively small part in the vehicle's velocity control algorithm. Qualitatively, the vehicle decelerates very quickly in the absence of accelerator pedal pressure at reasonably low speeds. For this reason, we plan primarily to use the brakes in cases where we wish to come to a full stop, such as when the software stop is activated. This is a reasonable choice because during drive-by-wire operation (i.e. not during startup or after the e-stop has been pressed), we can achieve a full range of steady-state velocities, as indicated by our preliminary vehicle characterization (Figure 40). The brake pedal will be

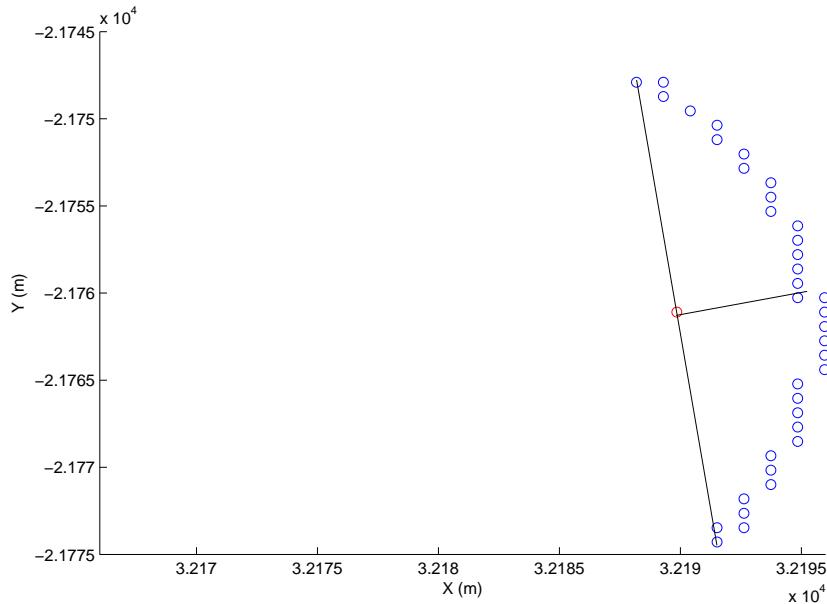


Fig. 41. Sample steering calibration curve at 30 degrees steering wheel angle.

controlled by the NI 9263 cRIO analog output module, with which we can depress the brake pedal in the same way as the accelerator pedal (at least from a software point of view).

10 Waypoint Following and Path Planning

In order to achieve waypoint following, we decided to use an occupancy grid scheme over a directed node framework. This was done for a few reasons, the most significant being that occupancy grids are easier and more intuitive for future implementation of probabilistic algorithms like Kalman filters. The path planning loop was initially derived from LabView's Dynamic A* algorithm in the robotics module. The block that orchestrates this aspect of the project is Path Planning Loop.vi. There are several important points to make about this block.

1. Simulation Mode: At one point, there was a simulation mode in this VI that allowed you to test the A* algorithm as well as other aspects of the path planning scheme. This simulation mode has not been used for a while, as we have moved on almost exclusively to field testing. It is operationally different from the "real" mode because it uses GPS for localization rather than deterministic grid point movement, and it also does not simulate the vehicle in any way. Rather, it simply assumes that we have perfect control over the vehicle, and therefore follows the path perfectly. Getting this simulation working again, if needed, should be fairly simple—just remember that all the queues need to be artificially populated.

2. The optimal path is constantly planned based on obstacles on the occupancy grid. The curve planning is intended to follow this optimal path, but it currently does not do so exactly, largely because the A* path planning does not take into account heading at all, so it assumes that the vehicle can move in any direction. This should be modified both so that it is realistic in terms of turning radius and so that the curve planning algorithm can actually follow it more precisely. The curve planning module does attempt to follow the planned path by using mini waypoints as targets along the projected path. This is discussed in more detail in the following section.
3. The curve planning block is what physically dictates the radius of curvature that should be written to the vehicle command queue. This uses past and future mini-waypoints to determine the desired heading, which prevents us from making drastic maneuvers as the waypoints get more complicated.

10.1 Curve Planning

While the A* algorithm calculates a path around obstacles and through major waypoints and breaks this path down into a series of shorter waypoints, an extra layer of logic is still needed to turn this path into a real-time curvature command. We refer to this algorithm as "Curve Planning."

The Curve-planning algorithm works by looking ahead a certain distance along the path calculated by the A* algorithm and calculating a reasonable curved path given the vehicle's steering dynamics. So that the vehicle can always retain a reasonable heading, it is important that the algorithm be able to simultaneously achieve both a desired destination and a desired final bearing. In order to accomplish this, the software uses a system based heavily on a construct known as Dubins curves.

Dubins curves are a method of path planning designed to optimize paths using distance as a criterion. The theory behind Dubins curves is that given any series of two locations and bearings, one a starting point and the other an intended final destination, a shortest-distance optimal path can be found by considering six structured paths. These paths are either a series of two curves, interspersed by a straight section, or three curved sections. Each curved section is defined as having the minimum turn radius of the vehicle. These six paths are usually abbreviated as LSL, LSR, RSL, RSR, RLR, LRL (R means right, L means left, S means straight). In practice, one of these paths will be shorter than the other five, and therefore it will be more desirable. For illustration, Some sample Dubins Curves are shown in Figure 42.

A VI titled "PlanCurveToNextWaypoint" uses the vehicle's current and desired location and bearings to calculate four of the six Dubins curves (RLR and LRL remain unimplemented) and choose the path with the shortest length. Within this VI, a subVI also calculates a path to the next waypoint consisting of a single curve of constant radius. While a path of this type will always be able to reach the next waypoint, it does not guarantee that the final heading will be correct. However, if the calculated final heading is within an acceptable margin of error, this single-curve path is used as an override to the calculated Dubins curve path. In practice this override usually comes into play as the vehicle nears a

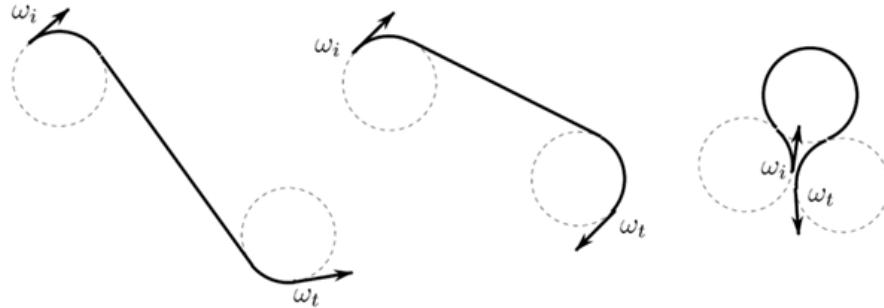


Fig. 42. Examples of Dubins Curves.

waypoint. This is a desirable feature because hitting a waypoint position exactly is more important than achieving an exact heading. It also tends to result in smoother paths when more complicated maneuvering is unnecessary.

Once a smooth curve has been calculated to the current waypoint, it is necessary to calculate a curvature command in real-time. This is accomplished by a VI titled "CurvatureOutputCalculator," which takes in a curve plan from "PlanCurveToNextWaypoint" and examines the first arc segment. If the segment is sufficiently long, the curvature output is simply equal to the curvature of that segment. However, if the arc segment is very short, the curvature output is proportionally reduced. This has the effect of smoothing out the transition to the straight segment, and it avoids a rail-to-rail control scenario.

Within the main path planning loop, there is a subVI titled "GetCurvatureOutput" which runs both of the VIs described above and returns a single number for a command curvature, which is then passed as a command to the vehicle's steering system.

It is important to note that the vehicle never follows the planned curve exactly because of variations in terrain and limits on the steering yaw rate. However, as the vehicle constantly calculates the ideal path, minor variations are compensated for in real-time.

11 Conclusion

11.1 First Semester Accomplishments

The focus of the first semester was chiefly on building infrastructure. As such, the major accomplishments of the semester were the design of the major physical systems and creation of a robust software architecture, as outlined below:

Actuation The physical actuation systems for steering, acceleration, and brake were designed and installed. All systems are transparent when power is removed (by activating any of the emergency stops), as well as being fully computer controllable.

Power Distribution A full power distribution system has been designed, with all power coming from a single gas generator mounted in the bed of the Gator. While there are multiple DC power buses in the system, all buses share a common ground near the AC source to avoid ground loops.

Emergency Stops A full wired emergency stop system has been designed. This system is accessible from inside the main cab. Triggering the hardware emergency stop disconnects power to the actuation system, enabling full manual override.

Computation All computational components have been purchased, installed, and wired to the DC power system.

Sensors A small number of essential sensors have been chosen for first semester autonomy. For each of these sensors, drivers have been developed, mount points have been chosen, and mounting systems have been designed.

Software Architecture A fully asynchronous software architecture, developed over the previous summer, has been improved and documented for future use. This system allows for parallelized operation of many sensors and supports networking of multiple computers for compartmentalization and diagnosis of various systems.

Controls Low resolution models of vehicle characteristics have been built based on experimental field data. These models allow for development of first pass controls systems, as well as allowing for incremental improvements to the controls simulations as more accurate vehicle dynamics data becomes available.

11.2 Second Semester Accomplishments

While significant progress was made during the first semester, there were still many aspects of the project that required improvement or completion to achieve the year-end goal of high accuracy waypoint following. The focus of the second semester was on finalizing the vehicle's hardware systems and developing behavioral software towards the ultimate goal of autonomy:

Electrical System A number of small improvements to the electrical system were implemented, centered primarily around power electronics. The system was redesigned to include a 48V bus, and new power supplies were migrated into the electrical system to rebalance the DC power buses. Additionally, a 24V DC UPS was included to make the computational systems more robust against power losses. A wireless router was also added to the system to allow external control of the vehicle from a groundstation. The addition of these new components necessitated a redesign of the internal electronics enclosure layout.

Emergency Stops The wired emergency stop system was supplemented with an additional emergency stop mounted on the rear of the vehicle, to allow for low speed unmanned operation. A wireless emergency stop was chosen and purchased via Draper Labs, but will not be installed by the end of this semester.

Sensors A pegboard was installed on the roof of the vehicle, and a LIDAR and GPS antenna were mounted on the pegboard. Additionally, an INS was installed inside of the main electronics enclosure and a stereoscopic camera was mounted inside of the main cab, facing forward. Finally, an encoder was installed on the rear drive shaft of the vehicle to allow basic velocity calculation and ultimately dead-reckoning of position.

Controls Velocity and steering controls have been developed, allowing integrated drive-by-wire control of the vehicle.

Autonomy With a stable hardware platform achieved, the project is now very code-oriented. Second semester work has focused specifically on achieving stable high-accuracy waypoint following. This has been implemented at both a macro and micro level; high level path planning is done on a discrete occupancy grid using an A* search algorithm and low level point-to-point path planning is done using Dubins curves. In field testing, the vehical has regularly been able to hit a series of waypoints within half a meter of accuracy. Additionally, the vehical incorporates static obstacle detection using the LIDAR sensor and will enter software pause mode if it determines that an obstacle lies in its path.

11.3 Future Plans

There are a variety of options for expansion of this project. Substantial progress towards a completely autonomous vehicle has been made during the 2009-2010 academic year, allowing for a number of potential future goals.

GPS Denied Navigation The ability to navigate without GPS is crucial in complex outdoor environments, given that accurate GPS data cannot be guaranteed in all areas. This goal focuses on developing a waypoint navigation system that incorporates, but does not rely on, GPS. This waypoint following system would be tested initially in a parking lot environment with GPS denied zones at medium speeds (4-6mph), emphasizing robust recovery from unpredictable loss of GPS data, and would then be transitioned to woodland operations.

Rough Terrain Path Planning An essential part of operating in complex outdoor environments is the ability to safely navigate rough terrain. This objective emphasizes path planning in a three-dimensional environment, where elevation and terrain dynamics cannot be estimated as being ideal. This system would most likely be developed and tested in the GPS accessible areas of Olin's outdoor robotics test track, with the vehicle operating at low speeds (2-3mph).

Dynamic Obstacle Avoidance Obstacle avoidance is crucial to safe autonomous operation. This goal emphasizes robust waypoint following that incorporates obstacle avoidance, for both static and dynamic obstacles. This system would be developed and tested under ideal conditions in a parking lot environment with both static and dynamic obstacles, with the vehicle operating at medium speeds (4-6mph).

References

1. K.J. strm and T. Hggelund, *PID Controllers: Theory, Design, and Tuning*, International Society for Measurement and Controls, 1995.
2. S. Kozk and M.Huba, *Control Systems Design*, Proceedings from 2nd IFAC Conference, Bratislava, Slovak Republic, Elsevier, 2003.

A Turning On the Robot

A.1 Turning On

1. Turn on the Generator (Figure 43):
 - (a) Fill with 87 Octane gas
 - (b) Turn gas cap switch to “On.”
 - (c) Turn main power switch to “On.”
 - (d) Move choke full right.
 - (e) Pull cord until the generator starts. This can take a while in very cold weather.
 - (f) Slowly move choke to the left until it clicks into full left position. In warm weather, you can move it pretty fast and be okay.
2. Turn on the main box power strip.
3. Turn on the computer (Figure 44)

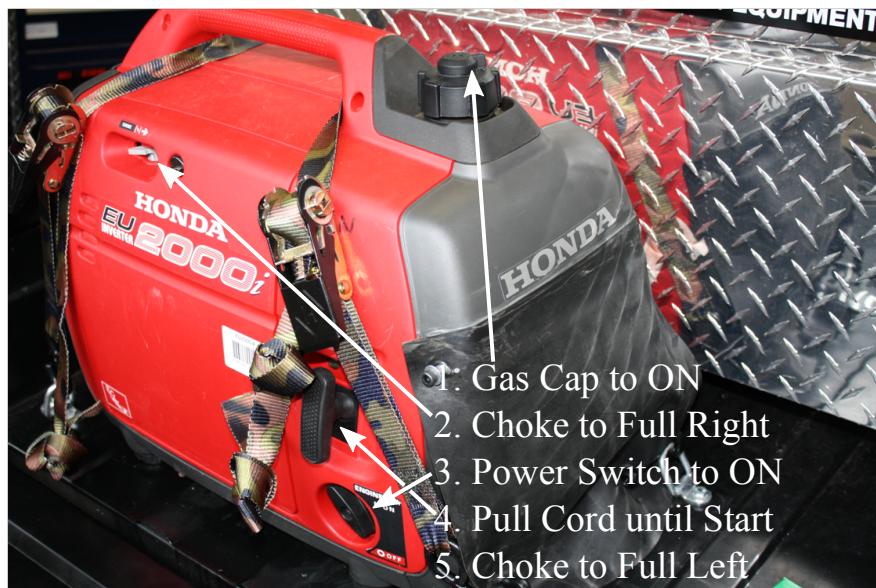


Fig. 43. Steps for turning on the generator.

A.2 Initializing Software

1. Windows should automatically login, start LabVIEW, and load the project. If it does not, you can start it with these data:
 - (a) Username: User
 - (b) Password: olin
 - (c) Project File: C:\Documents and Settings\User\Desktop\Medea\Gator1.lvproj

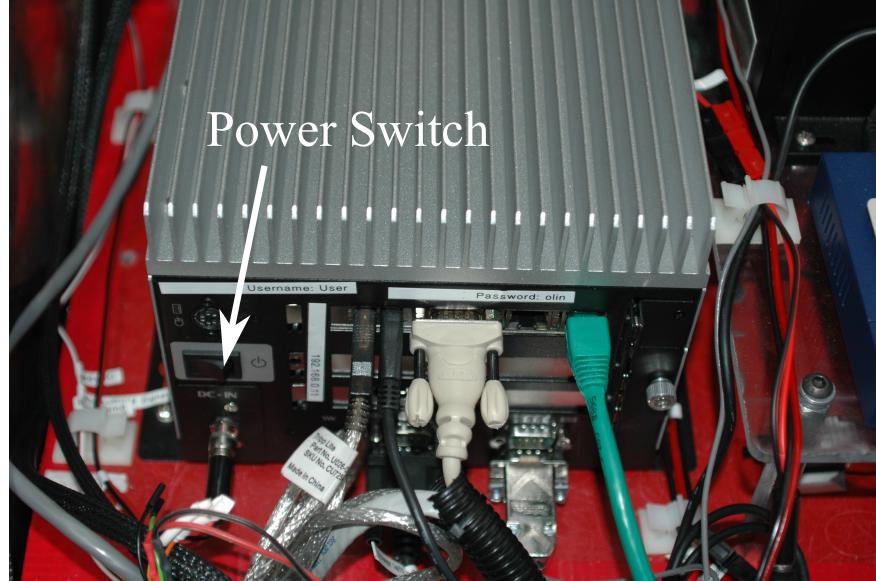


Fig. 44. The power switch for the Sea Level PC.

2. Load the PC's main code by opening Main.vi.
3. Load the cRIO's main code by opening NI-cRIO9074-013E431F (192.168.0.12) → cRIO → cRIO Main.vi.
4. There is a bug in the project file that requires you to fix the scan mode card:
 - (a) In the Project Explorer, open NI-cRIO9074-013E431F (192.168.0.12) → Chassis (cRIO-9074)
 - (b) Drag Mod2 (Slot 2, NI 9074) into FPGA Target (RIO0, cRIO-9074)
 - (c) Open the FPGA target and drag Mod2 back into the Chassis.

A.3 Software Settings

1. In Main.vi, you will see a number of settings on the **Setup** tab. The defaults will work if you haven't changed any hardware settings.
2. Set **GPS Map** to show the map you would like to see during your run. See *MissionMap.vi* and *Create Map Description.vi* for help creating a new map.
3. Run the PC's main code.
4. Run the cRIO's main code.

A.4 Run Self-Test

1. With both main VIs running, you should see the network comm lights go green, in addition to a number of other status lights such as 48V power, 24V power, etc. To center the steering wheel (and set the steering encoder correctly) select **Self-Test** in the **Program Select** box on Main (Figure 45). When there are no E-Stops set, the self test will begin. If it completes successfully, the "Steering" status light will turn green.

2. The Speed Encoder status will go green as soon as a speed larger than 0.1 MPH is recorded.
3. Once tested, you do not need to complete another self-test until you reset the cRIO.

A.5 Following Waypoints

1. Set Waypoints
 - (a) Go to the **GPS Map** tab (Figure 46)
 - (b) If this is the wrong map, shutdown Main, select the correct map on the Setup tab, and restart Main.
 - (c) Set the velocity you want to go to your waypoints with.
 - (d) Click on where you want your waypoints to be located.
 - (e) Select **Auto** in the Program Select box to go to those waypoints.



Fig. 45. The program selection widget on Main.vi.



Fig. 46. The GPS Map tab with four waypoints.

B Electrical System Diagrams and CAD

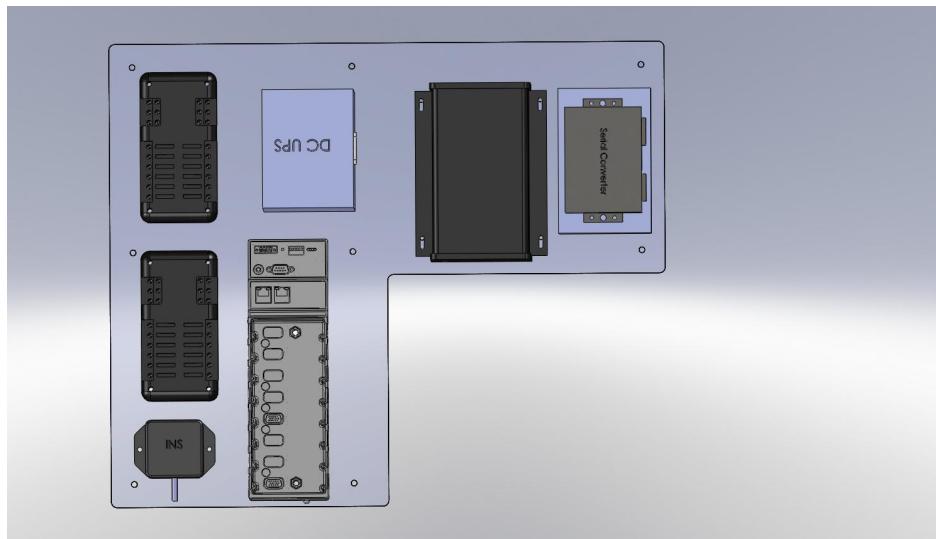


Fig. 47. CAD Drawing of the top level electronics board.

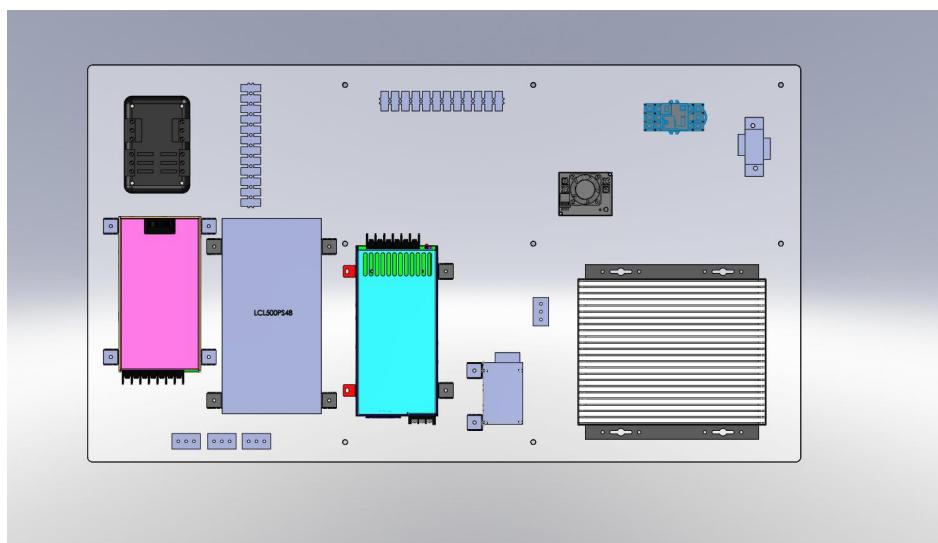
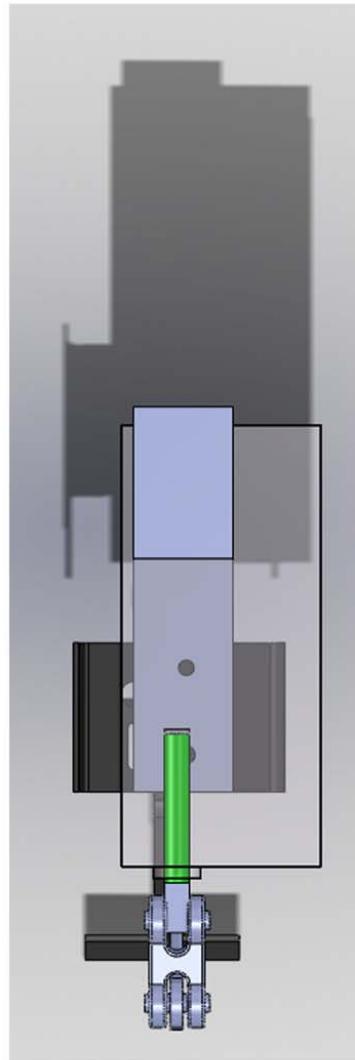
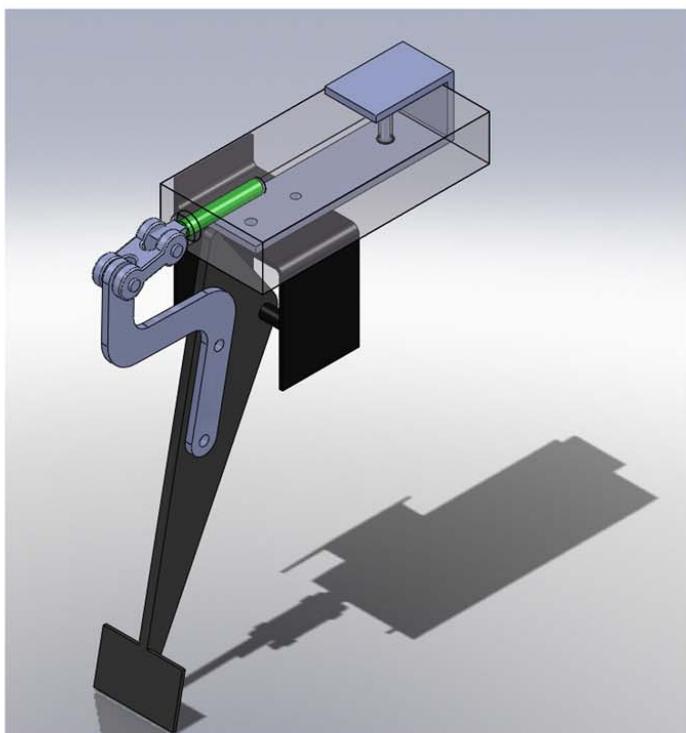
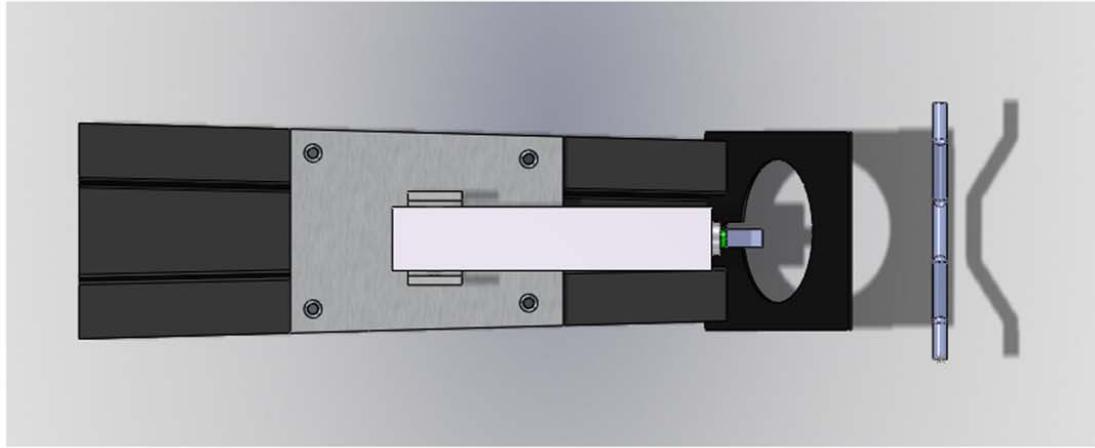
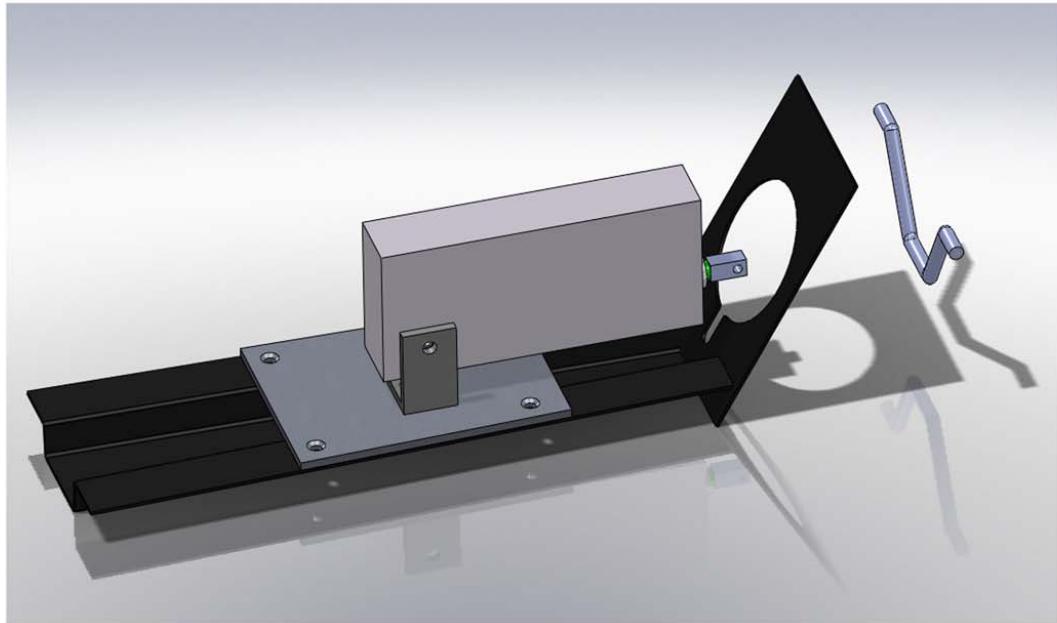


Fig. 48. CAD Drawing of the bottom level electronics board.

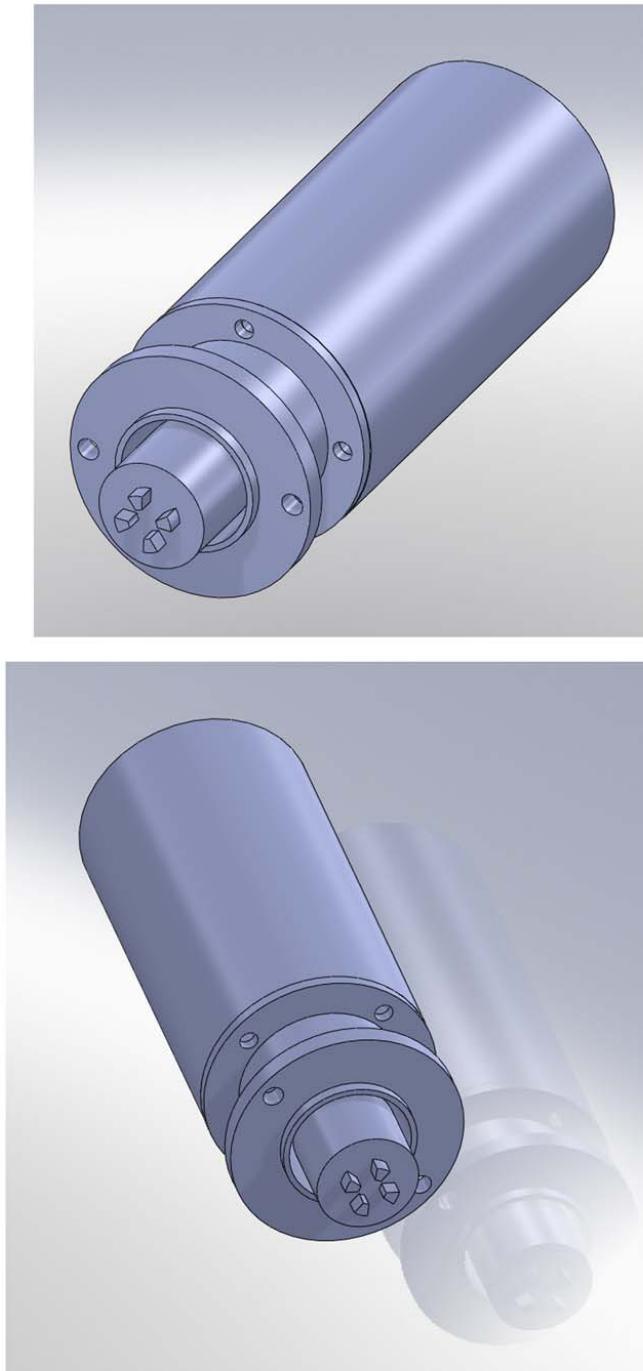
C Mechanical System Diagrams and CAD



Brake actuation assembly, with linear actuator mounted or brake pedal



Gas actuation system, with linear actuator mounted to cup holder tray, showing rocker bar for accelerator pedal



Steering motor with adaptor plate and spider coupling

D Datasheets

The Limited Edition Gator™ XUV Camo 4x4

Get one before they disappear.

- REALTREE HARDWOODS HD® Camo pattern is professionally applied through advanced hydrographics and 3M® film for a long-lasting and realistic appearance
- Independent rear suspension for a superior ride on the roughest terrain
- 30 mph (48 km/h) top speed, fast acceleration and quick-starting EFI engines on gas models
- Exclusive electronic governor and throttle for more power under load
- On-demand, true 4WD for superior terrain capability
- Over 100 available attachments and accessories for enhanced versatility

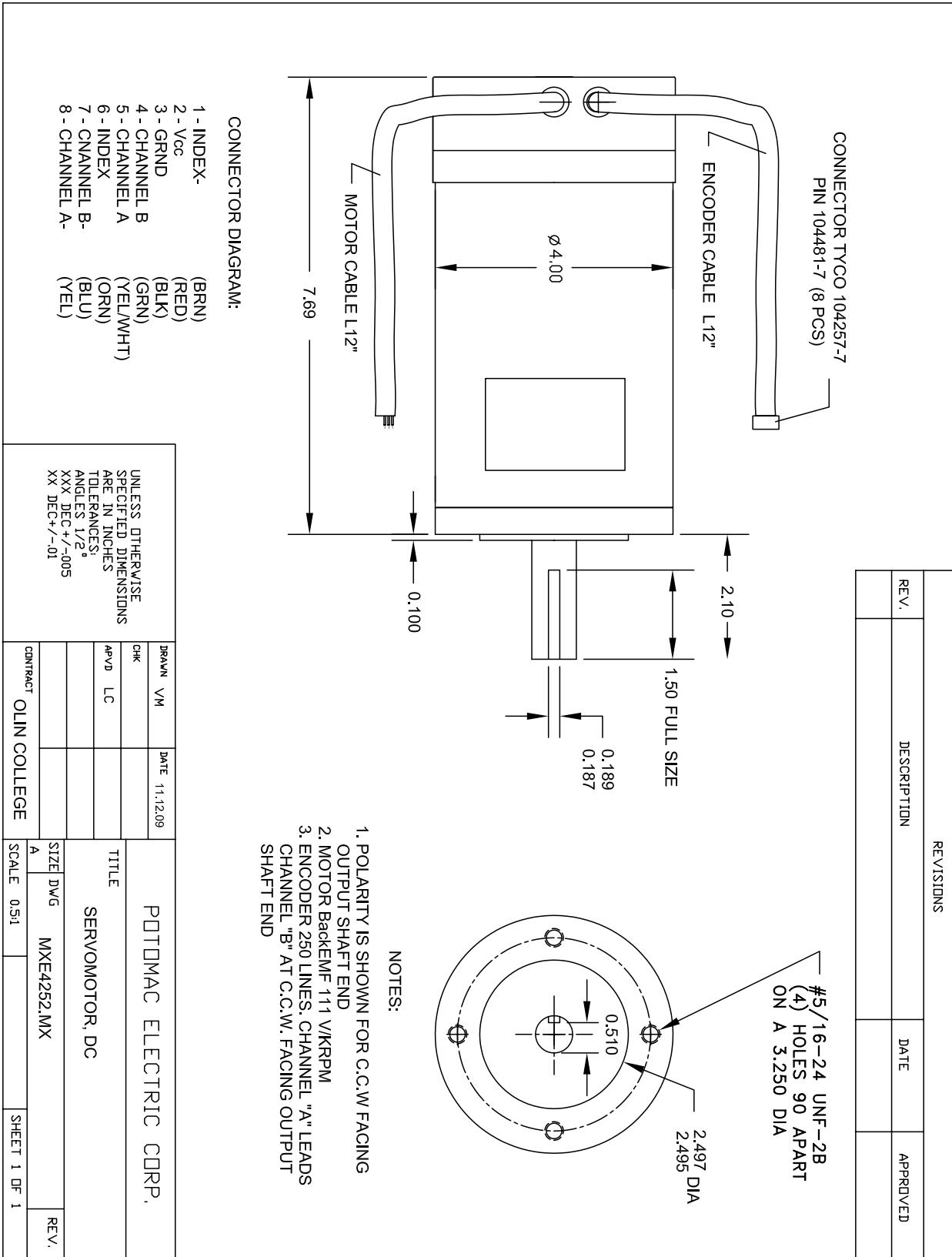
See local dealer to learn more.

Gator 620i XUV Camo 4x4		Gator 850D XUV Camo 4x4
Engine and Electrical		
Engine	23 hp** (617 cc), 2-cylinder, 4-cycle OHV gas, EFI	24.6 hp** (854 cc), 3-cylinder, 4-cycle OHV diesel
Maximum Torque	34.6 ft.-lb. torque (46.9 Nm) @2,100 rpm	36.9 ft.-lb. torque (50 Nm) @2,400 rpm
Ignition	Solid state 12V	Compression ignition
Lubrication	Full pressure	Full pressure
Cooling System	Liquid	Liquid
Air Cleaner	Intake	Intake
Battery	340 CCA	480 CCA
Alternator	25.5 amp @ 3,200 rpm, regulated, 306 watts	40 amp @ 3,200 rpm regulated
Headlights	Two 37.5-watt halogen	Two 37.5-watt halogen
Fuel Capacity	5.3 U.S. gal. (20.1 L)	5.3 U.S. gal. (20.1 L)
Transmission		
Type	Continuously Variable Transmission (CVT)	Continuously Variable Transmission (CVT)
4WD	On-demand true four-wheel-drive system	On-demand true four-wheel-drive system
Drive Belt	Spun top cog, 31 mm wide	Spun top cog, 31 mm wide
Ground Speed, Forward	0–30 mph (0–48 km/h) hi 0–17 mph (0–27 km/h) low 0–20 mph (0–32 km/h)	0–30 mph (0–48 km/h) hi 0–15 mph (0–24 km/h) low 0–17 mph (0–27 km/h)
Ground Speed, Reverse	Two speed; oil bath	Two speed; oil bath
Transaxle	Forward (hi-lo), neutral, reverse	Forward (hi-lo), neutral, reverse
Gear Selection	Front and rear hydraulic disk	Front and rear hydraulic disk
Brakes	Driveline mechanical disk, hand operated	Driveline mechanical disk, hand operated
Parking Brake	Sealed, double-row ball	Sealed, double-row ball
Bearings	27 mm dia. forged CV-shaft w/double offset joint	27 mm dia. forged CV-shaft w/double offset joint
Axle		
Suspension and Steering		
Suspension, Front	Independent with McPherson Strut	Independent with McPherson Strut
Front Suspension Travel (Total)	5.15 in. (131 mm)	5.2 in. (131 mm)
Suspension, Rear	Independent with coil over shock	Independent with coil over shock
Rear Suspension Travel (Total)	7 in. (170 mm)	7 in. (170 mm)
Steering	Rack and pinion	Rack and pinion
Occupant Protective System (OPS)	SAE J2194 and OSHA ROPS standards steel tubular structure with 3-pt. belts and multiple handholds	SAE J2194 and OSHA ROPS standards steel tubular structure with 3-pt. belts and multiple handholds
Dimensions		
Length/Width/Height (w/Bumper and OPS)	113-in. L x 59.3-in. W x 74-in. H (2870 mm L x 1506 mm W x 1880 mm H)	113-in. L x 59.3-in. W x 74-in. H (2870 mm L x 1506 mm W x 1880 mm H)
Front-Tread Centers	44.8 in. (1137 mm)	44.8 in. (1137 mm)
Rear-Tread Centers	47.9 in. (1216 mm)	47.9 in. (1216 mm)
Wheelbase	79 in. (2007 mm)	79 in. (2007 mm)
Weight (Including Fuel/Fluids)	1,383 lb. (627 kg)	1,532 lb. (695 kg)
Seating Capacity/Type	2/Professional high back, bucket (tilt forward)	2/Professional high back, bucket (tilt forward)
Towing Capacity*	1,300 lb. (590 kg)	1,300 lb. (590 kg)
Payload Capacity*	1,400 lb. (635 kg)	1,400 lb. (635 kg)
Ground Clearance		
Cargo Box	11 in. (267 mm)	11 in. (267 mm)
Volume	Heavy-duty steel	Heavy-duty steel
Weight (On Level Terrain)	11.2 cu. ft. (0.32 m³)	11.2 cu. ft. (0.32 m³)
Length/Width/Depth	1,000 lb. (554 kg)	1,000 lb. (554 kg)
Tailgate	43.9-in. L x 49-in. W x 9-in. D	43.9-in. L x 49-in. W x 9-in. D
Front Tires	1116 mm L x 1244 mm W x 229 mm D	1116 mm L x 1244 mm W x 229 mm D
All Trail II = 25x10-12	Removable; hinged at bottom	Removable; hinged at bottom
All Trail II = 25x9-12	AT489 = 25x10-12	All Trail II = 25x9-12
All Trail II = 25x11-12	AT489 = 25x11-12	AT489 = 25x11-12
All Trail II = 25x11-12	All Trail II = 25x11-12	All Trail II = 25x11-12
Available Tread Types	Standard (All Trail II), Aggressive (AT489)	Standard (All Trail II), Aggressive (AT489)
Available Colors	REALTREE HARDWOODS HD® Camouflage	REALTREE HARDWOODS HD® Camouflage
Rear Receiver Hitch	2-inch, standard	2-inch, standard



*Includes 200-lb. (91 kg) operator and 200-lb. (91 kg) passenger and maximum box capacity. **The engine horsepower information is provided by the engine manufacturer to be used for comparison purposes only.

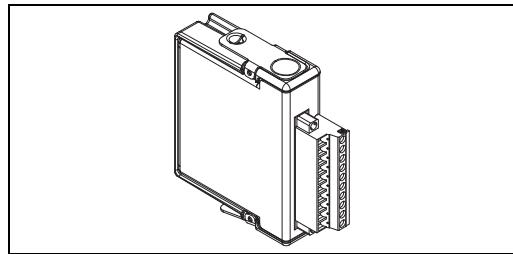
This literature has been compiled for worldwide circulation. While general information, pictures and descriptions are provided, some illustrations and text may include finance, credit, insurance, product options and accessories NOT AVAILABLE in all regions. PLEASE CONTACT YOUR LOCAL DEALER FOR DETAILS. John Deere reserves the right to change specification, design and price of the products described in this literature without notice. John Deere's green and yellow color scheme, the leaping deer symbol, and JOHN DEERE are trademarks of Deere & Company.



**OPERATING INSTRUCTIONS AND SPECIFICATIONS
NI 9263**

4-Channel, ± 10 V, 16-Bit Analog Voltage Output Module

Français	Deutsch	日本語	한국어	简体中文
ni.com/manuals				



This document describes how to use the National Instruments 9263 and includes specifications and terminal assignments for the NI 9263. Visit ni.com/info and enter `rdsoftwareversion` to determine which software you need for the modules you are using. For information about installing, configuring, and programming the system, refer to the system documentation. Visit ni.com/info and enter `cseriesdoc` for information about C Series documentation.



Note The safety guidelines and specifications in this document are specific to the NI 9263. The other components in the system might not meet the same safety ratings and specifications. Refer to the documentation for each component in the system to determine the safety ratings and specifications for the entire system. Visit ni.com/info and enter `cseriesdoc` for information about C Series documentation.

Safety Guidelines

Operate the NI 9263 only as described in these operating instructions.



Hot Surface This icon denotes that the component may be hot. Touching this component may result in bodily injury.

Connecting the NI 9263

The NI 9263 has a 10-terminal, detachable screw-terminal connector that provides connections for 4 analog output channels.

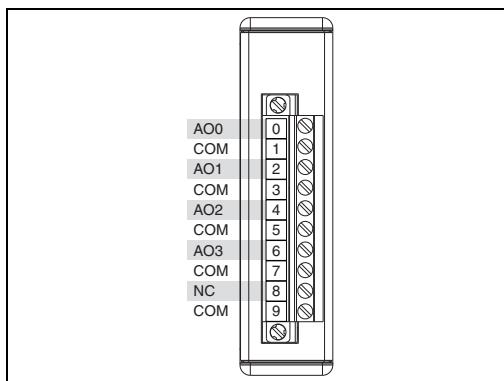


Figure 2. NI 9263 Terminal Assignments

© National Instruments Corp.

7 NI 9263 Operating Instructions and Specifications

Each channel of the NI 9263 has an AO terminal and a common terminal, COM, and there is an additional COM terminal at the bottom of the connector. All of the COM terminals are internally connected to the isolated ground reference of the module. You can connect a load to each channel of the NI 9263. Connect the positive lead of the load to the AO terminal, and the ground of the load to the corresponding COM terminal. Refer to Figure 3 for an illustration of connecting a load to the NI 9263.

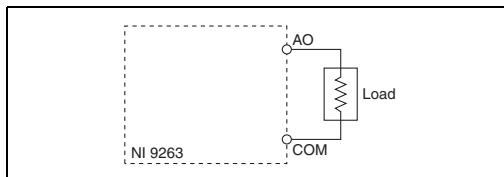


Figure 3. Connecting a Load to the NI 9263



Note You must use 2-wire ferrules to create a secure connection when connecting more than one wire to a single terminal on the NI 9263.

Each channel has a digital-to-analog converter (DAC) that produces a voltage signal. Each channel also has overvoltage and short-circuit protection. Refer to the *Specifications* section for more information about the overvoltage and short-circuit protection. Refer to Figure 4 for an illustration of the output circuitry for one channel of the NI 9263.

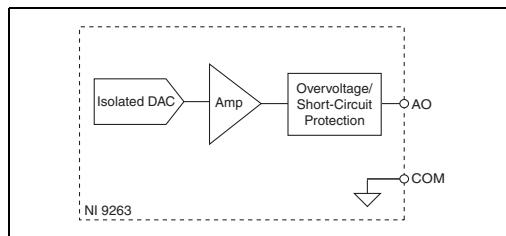


Figure 4. Output Circuitry for One Channel of the NI 9263

When the module powers on, the channels output the startup voltage. Refer to the *Specifications* section for more information about startup voltage. Refer to the software help for information

about configuring startup output states in software. Visit ni.com/info and enter `cseriesdoc` for information about C Series documentation.

Wiring for High-Vibration Applications

If an application is subject to high vibration, National Instruments recommends that you either use ferrules to terminate wires to the detachable screw-terminal connector or use the NI 9932 backshell kit to protect the connections. Refer to Figure 5 for an illustration of using ferrules. Refer to Figure 1 for an illustration of the NI 9932 connector backshell.

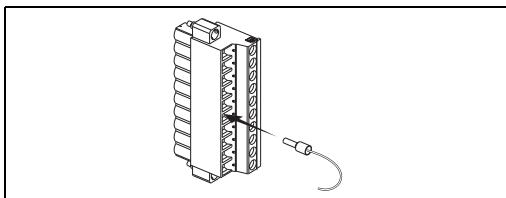


Figure 5. 10-Terminal Detachable Screw-Terminal Connector with Ferrule

Sleep Mode

This module supports a low-power sleep mode. Support for sleep mode at the system level depends on the chassis that the module is plugged into. Refer to the chassis manual for information about support for sleep mode. If the chassis supports sleep mode, refer to the software help for information about enabling sleep mode. Visit ni.com/info and enter `cseriesdoc` for information about C Series documentation.

Typically, when a system is in sleep mode, you cannot communicate with the modules. In sleep mode, the system consumes minimal power and may dissipate less heat than it does in normal mode. Refer to the *Specifications* section for more information about power consumption and thermal dissipation.

Specifications

The following specifications are typical for the range -40 to 70 °C unless otherwise noted. All voltages are relative to COM unless otherwise noted.

Output Characteristics

Number of channels	4 analog output channels
DAC resolution	16 bits
Type of DAC.....	String
Power-on output state	Channels off
Startup voltage	0 V ¹
Power-down voltage	0 V ²

¹ When the module powers on, a glitch occurs for 20 µs peaking at -1.5 V.

² The power-down voltage peaks at 1.8 V before exponentially discharging to 0 V in 100 µs. You can add a 10 kΩ load to reduce the peak voltage.

Output voltage range
Nominal ± 10 V
Minimum ± 10.4 V
Typical ± 10.7 V
Maximum ± 11 V
Current drive ± 1 mA per channel max
Output impedance 2Ω
Accuracy

Measurement Conditions	Percent of Reading (Gain Error)	Percent of Range* (Offset Error)
Calibrated, max (-40 to 70 °C)	0.35%	0.75%
Calibrated, typ (25 °C, ± 5 °C)	0.03%	0.1%
Uncalibrated, max (-40 to 70 °C)	2.2%	1.7%
Uncalibrated, typ (25 °C, ± 5 °C)	0.3%	0.25%

* Range equals ± 10.7 V

Stability

- Gain drift 14 ppm/°C
Offset drift 110 µV/°C

Protection

- Overvoltage ±30 V
Short-circuit Indefinitely

Update time

Number of Channels	Update Time for NI cRIO-9151 R Series Expansion Chassis	Update Time for All Other Chassis
1	3.5 µs min	3 µs min
2	6.5 µs min	5 µs min
3	9 µs min	7.5 µs min
4	12 µs min	9.5 µs min

Noise

- Updating at 100 kS/s 600 µV_{rms}
Not updating 260 µV_{rms}

Slew rate	4 V/ μ s
Crosstalk	76 dB
Settling time (100 pF load, to 1 LSB)	
Full-scale step.....	20 μ s
1 V step.....	13 μ s
0.1 V step.....	10 μ s
Capacitive drive	1,500 pF min
Monotonicity.....	16 bits
DNL	\pm 1 LSB max
INL (endpoint).....	\pm 12 LSB max
MTBF	1,732,619 hours at 25 °C; Bellcore Issue 2, Method 1, Case 3, Limited Part Stress Method



Note Contact NI for Bellcore MTBF specifications
at other temperatures or for MIL-HDBK-217F
specifications.

Power Requirements

Power consumption from chassis	
Active mode (at -40 °C).....	500 mW max
Sleep mode	25 µW max
Thermal dissipation (at 70 °C)	
Active mode	750 mW max
Sleep mode	25 µW max

Physical Characteristics

If you need to clean the module, wipe it with a dry towel.



Note For two-dimensional drawings and three-dimensional models of the C Series module and connectors, visit ni.com/dimensions and search by module number.

Screw-terminal wiring	12 to 24 AWG copper conductor wire with 10 mm (0.39 in.) of insulation stripped from the end
Torque for screw terminals	0.5 to 0.6 N · m (4.4 to 5.3 lb · in.)

Ferrules 0.25 mm² to 2.5 mm²
Weight 150 g (5.3 oz)

Safety

Isolation Voltages

Connect only voltages that are within the following limits.

Channel-to-channel None				
Channel-to-earth ground	<table><tbody><tr><td>Continuous</td><td>..... 250 V_{rms}, Measurement Category II</td></tr><tr><td>Withstand</td><td>..... 2,300 V_{rms}, verified by a 5 s dielectric withstand test</td></tr></tbody></table>	Continuous 250 V _{rms} , Measurement Category II	Withstand 2,300 V _{rms} , verified by a 5 s dielectric withstand test
Continuous 250 V _{rms} , Measurement Category II				
Withstand 2,300 V _{rms} , verified by a 5 s dielectric withstand test				
Division 2 and Zone 2 hazardous locations applications (Channel-to-earth ground) 60 VDC, Measurement Category I				

Measurement Category I is for measurements performed on circuits not directly connected to the electrical distribution system referred to as *MAINS* voltage. *MAINS* is a hazardous live electrical supply system that powers equipment. This category is for measurements of voltages from specially protected secondary

circuits. Such voltage measurements include signal levels, special equipment, limited-energy parts of equipment, circuits powered by regulated low-voltage sources, and electronics.



Caution Do not connect to signals or use for measurements within Measurement Categories II, III, or IV.

Measurement Category II is for measurements performed on circuits directly connected to the electrical distribution system. This category refers to local-level electrical distribution, such as that provided by a standard wall outlet, for example, 115 V for U.S. or 230 V for Europe.



Caution Do not connect to signals or use for measurements within Measurement Categories III or IV.

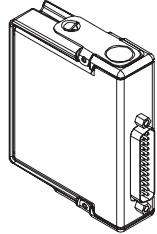
OPERATING INSTRUCTIONS AND SPECIFICATIONS

NI 9401

8-Channel, TTL Digital Input/Output Module

Français Deutsch 日本語 한국어 简体中文

ni.com/manuals



This document describes how to use the National Instruments 9401 and includes specifications and pin assignments for the NI 9401. Visit ni.com/info and enter `rdsoftwareversion` to determine which software you need for the modules you are using. For information about installing, configuring, and programming the system, refer to the system documentation. Visit ni.com/info and enter `cseriesdoc` for information about C Series documentation.



Note The safety guidelines and specifications in this document are specific to the NI 9401. The other components in the system might not meet the same safety ratings and specifications. Refer to the documentation for each component in the system to determine the safety ratings and specifications for the entire system. Visit ni.com/info and enter `cseriesdoc` for information about C Series documentation.

Connecting the NI 9401

The NI 9401 has a 25-pin DSUB connector that provides connections for eight digital input/output channels.

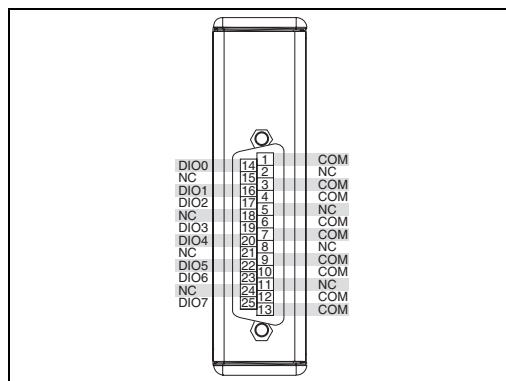


Figure 1. NI 9401 Pin Assignments

Each channel has a DIO pin to which you can connect a digital input or output device. The eight DIO channels are internally referenced to COM, so you can use any of the nine COM lines as a reference for the external signal.

The DIO channels are grouped in two ports, one containing channels 0, 1, 2, and 3, and one containing channels 4, 5, 6, and 7. You can independently configure each digital port in software for input or output. Note that all four channels in the port must share the same line direction. Refer to the software help for information about configuring ports on the NI 9401.

Each channel also has a pull-down resistor and includes overvoltage, overcurrent, and short-circuit protection. Refer to the *Specifications* section for more information about input thresholds and overvoltage protection. Refer to the *Overcurrent Protection* section for more information about overcurrent protection.

Figure 2 illustrates how to connect an SPI device to the NI 9401. In this example, the three channels assigned to output signals are on one port and the channel assigned to an input signal is on the other port.

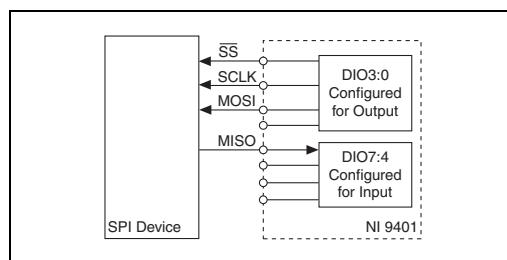


Figure 2. Connecting an SPI Device to the NI 9401

Figure 3 illustrates how to connect several types of digital devices to the NI 9401.

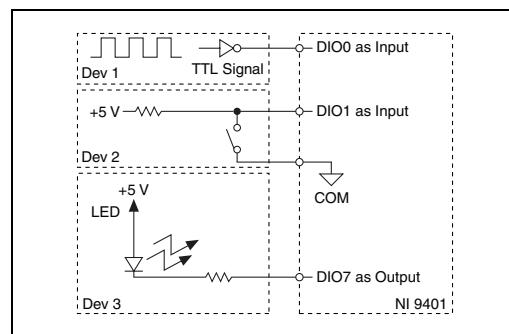


Figure 3. Connecting Digital Devices to the NI 9401

Overcurrent Protection

Overcurrent protection on the NI 9401 allows only a specified amount of current for switching DIO channels or sourcing the output load. If the NI 9401 goes into an overcurrent state, by exceeding the specified maximum switching frequency or the output load, the module power supply begins to drop in voltage until it completely turns off or the overcurrent condition is removed. When the module is in this state, it can accept new line direction configuration and output state data but cannot pass valid input data to the software. Refer to the *Specifications* section for more information about the maximum switching frequency and output load for each channel.

Sleep Mode

This module supports a low-power sleep mode. Support for sleep mode at the system level depends on the chassis that the module is plugged into. Refer to the chassis manual for information about support for sleep mode. If the chassis supports sleep mode, refer to the software help for information about enabling sleep mode. Visit ni.com/info and enter `cseriesdoc` for information about C Series documentation.

Typically, when a system is in sleep mode, you cannot communicate with the modules. In sleep mode, the system consumes minimal power and may dissipate less heat than it does in normal mode. Refer to the *Specifications* section for more information about power consumption and thermal dissipation.

Specifications

The following specifications are typical for the range –40 to 70 °C unless otherwise noted. All voltages are relative to COM unless otherwise noted.

Input/Output Characteristics

Number of channels 8 DIO channels
Default power-on line direction Input
Input/output type TTL, single-ended

Digital logic levels	
Input	
Voltage	5.25 V max
High, V_{IH}	2 V min
Low, V_{IL}	0.8 V max
Output	
High, V_{OH}	5.25 V max
Sourcing 100 μ A	4.7 V min
Sourcing 2 mA.....	4.3 V min
Low, V_{OL}	
Sinking 100 μ A	0.1 V max
Sinking 2 mA.....	0.4 V max

Maximum input signal switching frequency by number of input channels, per channel

8 input channels.....	9 MHz
4 input channels.....	16 MHz
2 input channels.....	30 MHz

Maximum output signal switching frequency by number of output channels with an output load of 1 mA, 50 pF, per channel

8 output channels.....	5 MHz
4 output channels.....	10 MHz
2 output channels.....	20 MHz

I/O propagation delay 100 ns max

I/O pulse width distortion 10 ns typ

Input current ($0 \text{ V} \leq V_{\text{in}} \leq 4.5 \text{ V}$) $\pm 250 \mu\text{A}$ typ

Input capacitance 30 pF typ

Input rise/fall time 500 ns max

Overvoltage protection,
channel-to-COM $\pm 30 \text{ V}$ max on one channel at
a time; however, continued
use at this level will degrade
the life of the module.

MTBF 1,244,763 hours at 25 °C;
Bellcore Issue 2, Method 1,
Case 3, Limited Part Stress
Method



Note Contact NI for Bellcore MTBF specifications
at other temperatures or for MIL-HDBK-217F
specifications.

Power Requirements

Power consumption from chassis

Active mode 580 mW max
Sleep mode 1 mW max

Thermal dissipation (at 70 °C)

Active mode 580 mW max
Sleep mode 1 mW max

Physical Characteristics

If you need to clean the module, wipe it with a dry towel.

Weight..... 145 g (5.1 oz)

Safety

Maximum Voltage¹

Connect only voltages that are within the following limits.

Channel-to-COM ±30 V max on one channel
at a time, Measurement
Category I

Isolation Voltages

Channel-to-channel..... None

Channel-to-earth ground
Continuous 60 VDC, Measurement
Category I
Withstand..... 1,000 V_{rms}, verified by a 5 s
dielectric withstand test

Measurement Category I is for measurements performed on
circuits not directly connected to the electrical distribution system
referred to as *MAINS* voltage. *MAINS* is a hazardous live electrical

¹ The maximum voltage that can be applied or output between any channel and COM without damaging the module or other devices.

supply system that powers equipment. This category is for measurements of voltages from specially protected secondary circuits. Such voltage measurements include signal levels, special equipment, limited-energy parts of equipment, circuits powered by regulated low-voltage sources, and electronics.



Caution Do not connect the NI 9401 to signals or use for measurements within Measurement Categories II, III, or IV.

Safety Standards

This product is designed to meet the requirements of the following standards of safety for electrical equipment for measurement, control, and laboratory use:

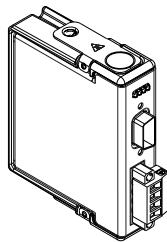
- IEC 61010-1, EN 61010-1
- UL 61010-1, CSA 61010-1



Note For UL and other safety certifications, refer to the product label or visit ni.com/certification, search by module number or product line, and click the appropriate link in the Certification column.

**OPERATING INSTRUCTIONS AND SPECIFICATIONS
NI 9505
DC Brushed Servo Drive**

Français Deutsch 日本語 한국어 简体中文
ni.com/manuals



This document describes how to use the National Instruments 9505 module and includes specifications and pin assignments for the NI 9505. Visit ni.com/info and enter `rdssoftwareversion` to determine which software you need for the modules you are using. For information about installing, configuring, and programming the system, refer to the system documentation. Visit ni.com/info and enter `cseriesdoc` for information about C Series documentation.



Note The safety guidelines and specifications in this document are specific to the NI 9505. The other components in the system may not meet the same safety ratings and specifications. Refer to the documentation for each component in the system to determine the safety ratings and specifications for the entire system.



Caution This product may cause radio interference in a domestic environment, in which case supplementary mitigation measures may be required.

NI 9505 Hardware Overview

The NI 9505 provides unique flexibility and customization. The NI 9505 works together with the LabVIEW FPGA Module to create a highly customizable motor drive or actuator amplifier. Figure 1 illustrates the functionality of the NI 9505 working in conjunction with the LabVIEW FPGA Module in a typical motion control application. Figures 2 and 3 show more detailed versions of the position, velocity, and current loops implemented in the LabVIEW FPGA Module. A typical application contains a position loop, velocity loop, and current loop, implemented in the LabVIEW FPGA Module block diagram. Depending on the application, you may not need to use all three loops. The examples installed in the `labview\examples\CompactRIO\` `Module Specific\NI 9505` directory illustrate methods for implementing each of these loops.

The NI 9505 returns the motor or actuator current data to the LabVIEW FPGA Module for use in a current loop or for monitoring. The NI 9505 also returns status information such as drive fault status, V_{SUP} presence, and emergency stop status to the LabVIEW FPGA Module for use in system monitoring. Refer to the *NI 9505 Reference Help* book in the *LabVIEW Help*.

available by selecting **Help>Search the LabVIEW Help**, for more information about the available status information.

The LabVIEW FPGA Module generates a PWM signal and sends the signal to the NI 9505. The PWM signal is proportional to the desired current or torque you want to provide to the motor or actuator. Increasing the PWM duty cycle results in increased current and thus increased torque.

Quadrature encoder signals pass through the NI 9505 and are processed in the LabVIEW FPGA Module for use in the position and velocity loops. Refer to Figure 4 for a typical NI 9505 connection example, including encoder and E-Stop inputs.

For more advanced motion control applications, NI SoftMotion provides functions for trajectory generation, spline interpolation, position and velocity PID control, and encoder implementation using both the LabVIEW Real-Time Module and the LabVIEW FPGA Module. With NI SoftMotion you can create a custom motion controller without the need to develop the trajectory generator or spline engine yourself. Refer to the `labview\examples\CompactRIO\Module Specific\NI 9505` directory for example VIs using the NI 9505 and NI SoftMotion.

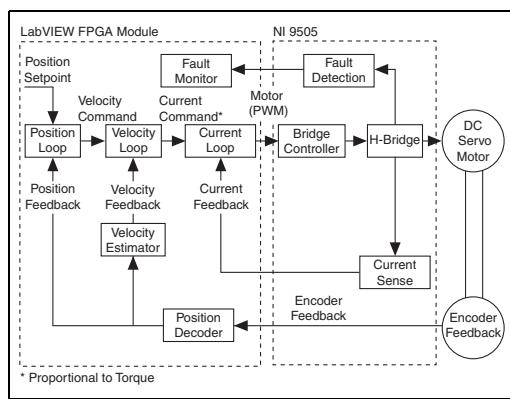


Figure 1. NI 9505 Block Diagram

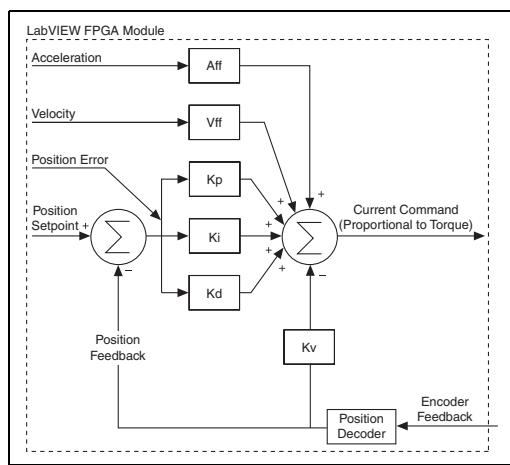


Figure 2. LabVIEW FPGA Module NI 9505 PID Loop

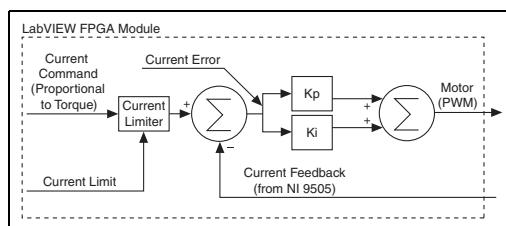


Figure 3. LabVIEW FPGA Module NI 9505 Current Loop

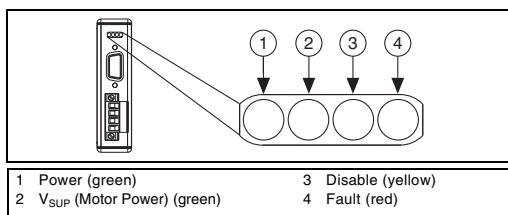
Hot-Swap Behavior

The NI 9505 is always disabled when it is inserted in the chassis, regardless of whether V_{SUP} is present or not. You can enable the drive using the **Enable Drive** method in software. Refer to the *NI 9505 Reference Help* book in the *LabVIEW Help*, available by selecting **Help>Search the LabVIEW Help**, for more information about enabling the drive.

When the NI 9505 is removed from the chassis while it is enabled, the power to the motor is removed and the motor decelerates to a stop based on its own friction.

LED Indicators

The NI 9505 has four LEDs to display status information.



Power

The Power LED (green) illuminates when the NI 9505 is properly inserted into a powered chassis.



Note The Power LED does not illuminate when the chassis is in sleep mode.

V_{SUP}

The V_{SUP} LED (green) illuminates when the motor DC power supply is properly connected and powering the drive.

Disable

The Disable LED (yellow) illuminates when the drive is disabled. The drive is disabled by default at power-on. You can enable the drive using the **Enable Drive** method in software. Refer to the *NI 9505 Reference Help* book in the *LabVIEW Help*, available by selecting **Help>Search the LabVIEW Help**, for more information about this method.

Fault

 **Caution** If the Fault LED is lit, determine the cause of the fault and correct it before enabling the drive.

The Fault LED (red) illuminates when a fault occurs. A fault disables the drive. Causes for fault are the following:

 **Caution** V_{SUP} greater than 40 V will result in damage to the NI 9505.

- Overvoltage
- Undervoltage

- Motor terminal (MOTOR \pm) short to V_{SUP}
- Motor terminal (MOTOR \pm) short to COM
- Module temperature exceeds 115 °C
- Sending commands to the motor before enabling the drive



Note Do not command motor movement until the drive is enabled with the **Enable Drive** method. Attempting to control the motor before it is enabled will result in a fault.

- Violating PWM minimum pulse width requirements. Refer to the *Specifications* section for more information about PWM.

Sleep Mode

This module supports a low-power sleep mode. Support for sleep mode at the system level depends on the chassis that the module is plugged into. Refer to the chassis manual for information about support for sleep mode. If the chassis supports sleep mode, refer to the software help for information about enabling sleep mode. Visit ni.com/info and enter `cseriesdoc` for information about C Series documentation.

Typically, when a system is in sleep mode, you cannot communicate with the modules. In sleep mode, the system consumes minimal power and may dissipate less heat than it does

in normal mode. Refer to the *Specifications* section for more information about power consumption and thermal dissipation.



Note The Power LED does not illuminate when the chassis is in sleep mode.

Wiring the NI 9505

The NI 9505 has a 9-pin female DSUB connector that provides connections for the encoder inputs, a +5 V connection for encoder power, a connection for an emergency stop input, and a connection to COM. Refer to Table 1 for the pin assignments.

The NI 9505 also has a screw terminal connector that provides connections to a motor DC power supply and a DC brushed servo motor. Connect the positive lead of the power supply to terminal 4, V_{SUP} , and the negative lead to terminal 3, COM. Refer to Table 2 for the terminal assignments.



Note You must use 2-wire ferrules to create a secure connection when connecting more than one wire to a single terminal on the NI 9505 screw terminal.



Caution Do *not* turn on or plug in the motor DC power supply until the screw terminal connector is fully inserted.

Table 1. NI 9505 DSUB Pin Assignments

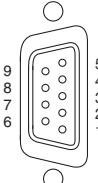
Connector	Pin	Signal
	1	Encoder Phase A+
	2	Encoder Phase B+
	3	Encoder Index+ (Phase Z+)
	4	Emergency Stop (E-Stop)
	5	+5 V (output)
	6	Encoder Phase A-
	7	Encoder Phase B-
	8	Encoder Index- (Phase Z-)
	9	Common (COM)

Table 2. NI 9505 Screw Terminal Terminal Assignments

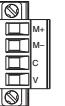
Module	Terminal	Signal
	M+	MOTOR+
	M-	MOTOR-
	C	COM (motor DC power supply reference)
	V	V _{SUP} (motor DC power supply)

Figure 4 shows a typical NI 9505 connection example, including encoder and E-Stop inputs.

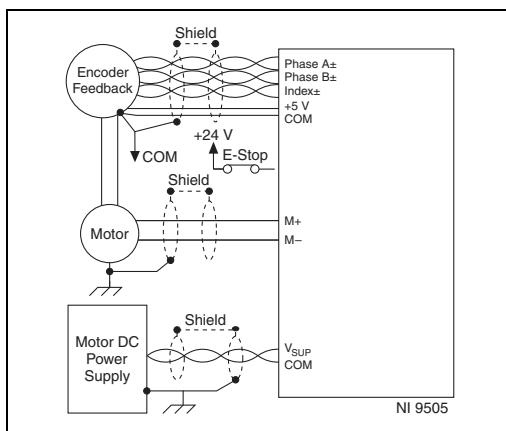


Figure 4. NI 9505 Connections

Optional Screw Terminal Accessory

Use the NI 9931 Screw Terminal Accessory instead of the detachable screw terminal connector to increase the output power of the module at temperatures below 70 °C. The NI 9931 is available from ni.com (NI part number 780571-01) or by calling your National Instruments sales representative. Refer to the *Specifications* section for more information. Refer to Figure 5 for an illustration.

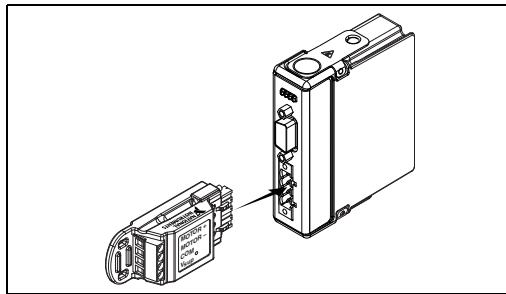


Figure 5. NI 9505 Module with Optional Screw Terminal Accessory

Wiring for High Vibration Applications

National Instruments recommends using ferrules to terminate wires to the detachable screw terminal connector or the NI 9931 Screw Terminal Accessory when you use the NI 9505 in high vibration applications. Refer to Figure 6 for an illustration.

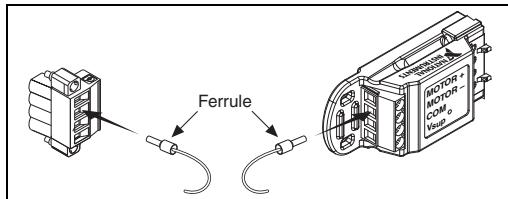


Figure 6. 4-Terminal Screw Terminal Connector or Accessory with a Ferrule
Motor Power Signals

The MOTOR+ and MOTOR- signals power the servo motor.
Motor direction is as follows:

- **Forward**—Clockwise (CW) facing motor shaft
- **Reverse**—Counterclockwise (CCW) facing motor shaft

Figure 7 shows clockwise and counterclockwise motor rotation.

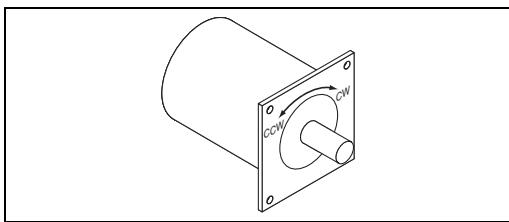


Figure 7. Clockwise and Counterclockwise Motor Rotation



Tip If the motor does not turn in the desired direction, reverse the MOTOR+ and MOTOR- signals.

Encoder Signals

The encoder signals consist of a Phase A, Phase B, and Index (Phase Z) input. The NI 9505 supports differential and single-ended inputs for Phase A, Phase B, and Index (Phase Z) signals. Figures 8 and 9 show simplified schematic diagrams of the encoder input circuit connected to differential and single-ended inputs. You can also accommodate open-collector output encoders by using a 1 k Ω pull-up resistor on each line to +5 VDC. Refer to the [Specifications](#) section for more information about the encoder inputs.

The encoder signals are raw digital input signals. These signals are used in the LabVIEW FPGA Module for position and/or velocity feedback. Figures 1 and 2 illustrate the use of the encoder signals in a position and velocity loop in the LabVIEW FPGA Module. Refer to the examples installed at `labview\examples\CompactRIO\Module Specific\NI 9505` for examples of using the encoder signals. Refer to the *NI 9505 Reference Help* book in the *LabVIEW Help*, available by selecting **Help>Search the LabVIEW Help**, for more information.

If the encoder cable length is greater than 3.05 m (10 ft), use encoders with differential line driver outputs for your applications. Power for a +5 V encoder—generated by a power supply inside the NI 9505—is available on pin 5 of the DSUB connector.

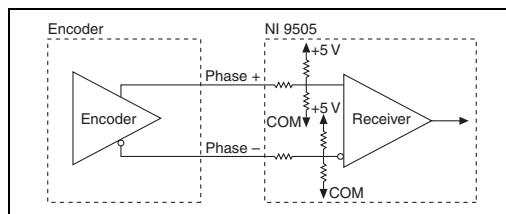


Figure 8. Differential Encoder Input Circuit

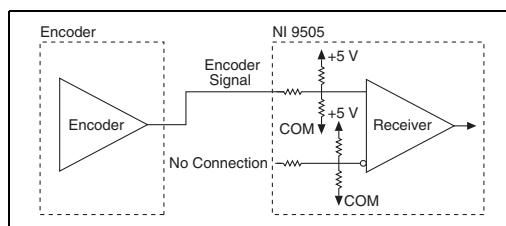


Figure 9. Single-Ended Encoder Input Circuit

Closed-loop servo applications require consistent directional polarity between the motor and encoder for correct operation. One industry-standard directional polarity is as follows:

- Positive = forward = clockwise (CW) facing motor shaft
- Negative = reverse = counterclockwise (CCW) facing motor shaft

Refer to Figure 7 for a depiction of clockwise and counterclockwise rotation. If encoder counting does not behave as expected, change the encoder polarity in the FPGA or swap the Phase A and Phase B connections.

When connecting the encoder wiring to the NI 9505, use shielded wire of at least 24 AWG. You must use cables with twisted pairs and an overall shield for improved noise immunity. Refer to Figure 4 for a connection example.



Note Using an unshielded cable may produce noise, which can corrupt the encoder signals and cause lost counts, reduced accuracy, or other erroneous encoder and drive operation.

Emergency Stop Signal

The E-Stop signal is an input to the drive from an emergency stop switch. Figure 10 shows a simplified schematic of the emergency stop input circuit. When the emergency stop switch is closed, current flows through the circuit, and the drive is enabled. When an external event activates the emergency stop switch, the switch opens and current stops flowing, disabling the drive. The E-Stop functionality is disabled by default. Refer to the *NI 9505 Reference Help* book in the *LabVIEW Help*, available by selecting **Help» Search the LabVIEW Help**, for information about how to enable this signal using the **Enable E-Stop** Property.

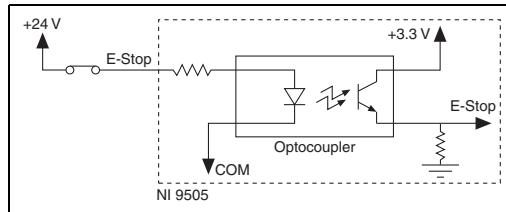


Figure 10. Emergency Stop Input Circuit

Cable Requirements for EMC Compliance

Use the following guidelines when selecting cables for the NI 9505:

- Use shielded cables with a low impedance connection to chassis ground to minimize noise and signal crosstalk.
- Tie the V_{SUP} cable shield to chassis ground at the module side only.
- Tie the motor cable shield to chassis ground at the motor side only.

- Tie the encoder cable shield to COM at the encoder side only.
- Wire encoder signals and their ground connections separately from all other connections to prevent lost encoder counts.
- Route wires along the machine frame to reduce high frequency noise.
- Add clamp-on ferrites to cables to further reduce emissions.
- Add a balun to the power cable to attenuate conducted and radiated emissions.

Using the NI 9505 with Other C Series Modules

Due to additional ambient heating of the NI 9505 when supplying more than 1 A to the load, the room temperature (25°C , $\pm 5^{\circ}\text{C}$) specifications of adjacent modules are not valid. The full operating temperature (-40°C to 70°C) specifications for these modules are still valid.

Specifications

The following specifications are typical for the temperature range –40 to 70 °C and a PWM rate of 20 kHz unless otherwise noted. All voltages are relative to COM unless otherwise noted.

Operating Conditions

Motor DC power supply (V_{SUP})	+8 to +30 VDC, 12 A max
Motor continuous current ¹	
(Motor±)	1 A @ 70 °C 5 A @ 40 °C
With NI 9931	
screw terminal accessory	1 A @ 70 °C 7.3 A @ 40 °C
Peak current ²	12 A < 2 s max
PWM	
Rate.....	20 kHz recommended, 40 kHz max

¹ For more information about maximum continuous current at temperatures less than 70 °C, visit ni.com/info and enter rdmot2.

² Allow at least 3.4 s between peak current intervals.



Caution Violating minimum pulse width will result in unpredictable performance.

Minimum pulse width (high or low).....	2 µs
Drive direction update rate	Nominally 20 µs
Current loop	
ADC resolution	12 bits
Current range.....	±12.7 A
Maximum update rate.....	20 µs
Minimum inductance.....	500 µH
MTBF	821,178 hours at 25 °C; Bellcore Issue 2, Method 1, Case 3, Limited Part Stress Method



Note Contact NI for Bellcore MTBF specifications at other temperatures or for MIL-HDBK-217F specifications.

Drive Protection

Undervoltage.....<6 V

 **Caution** V_{SUP} greater than 40 V will result in damage to the module.

Overvoltage.....>32 V

Reverse polarity -30 V

Motor terminal (MOTOR±)

short to ground..... Yes

Motor terminal (MOTOR±)

short to V_{SUP} Yes

Temperature fault trip point 115 °C (internal module temperature)

Encoder Input Characteristics

Number of inputs	3
Input type	Differential or single-ended
Voltage range	0 to 5.5 VDC
Digital logic levels	
Single-ended.....	TTL compatible
Input high threshold.....	2.4 V
Input low threshold	0.8 V
Differential	
Input threshold	±700 mV, line driver compatible
Common-mode voltage.....	-7 to 12 V
Input current	±1 mA
Maximum quadrature frequency.....	5 MHz

E-Stop Input

Input voltage range	0 to 30 V
Input ON voltage	3.5 to 30 V
Input OFF voltage	0 to 2 V
Turn-on current	500 µA, typical 1 mA, maximum

Power Requirements

Power consumption from chassis	
Active mode	100 mW max
Sleep mode	0.4 mW max
Thermal dissipation (at 70 °C)	
Active mode	1.5 W max
Sleep mode	0.4 mW max

Encoder Power Supply

5 V regulated output	
Voltage tolerance	5 V ±5%, $V_{SUP} \geq 8$ V
Current.....	125 mA

Physical Characteristics

If you need to clean the module, wipe it with a dry towel.



Note For two-dimensional drawings and three-dimensional models of the C Series module and connectors, visit ni.com/dimensions and search by module number.

Screw-terminal wiring	12 to 24 AWG copper conductor wire with 10 mm (0.39 in.) of insulation stripped from the end
Torque for screw terminals	0.5 to 0.6 N · m (4.4 to 5.3 lb · in.)
Ferrules	0.25 mm ² to 2.5 mm ²
Weight.....	155 g (5.5 oz)
NI 9931 Screw Terminal Accessory	
Screw terminal wiring	14 to 26 AWG copper conductor wire with 7 mm (0.28 in.) of insulation stripped from the end
Torque for screw terminals.....	0.5 to 0.6 N · m (4.4 to 5.3 lb · in.)
Ferrules.....	0.25 mm ² to 1.5 mm ²
Weight	40 g (1.4 oz)

Safety**Safety Voltages**

Connect only voltages that are within the following limits.

Channel-to-COM 0 to +30 VDC max,
Measurement Category I

Isolation

Channel-to-channel None

Channel-to-earth ground

Continuous 60 VDC,
Measurement Category I

Withstand 750 V_{rms}, verified by a 5 s
dielectric withstand test

Measurement Category I is for measurements performed on circuits not directly connected to the electrical distribution system referred to as *MAINS* voltage. *MAINS* is a hazardous live electrical supply system that powers equipment. This category is for measurements of voltages from specially protected secondary circuits. Such voltage measurements include signal levels, special equipment, limited-energy parts of equipment, circuits powered by regulated low-voltage sources, and electronics.

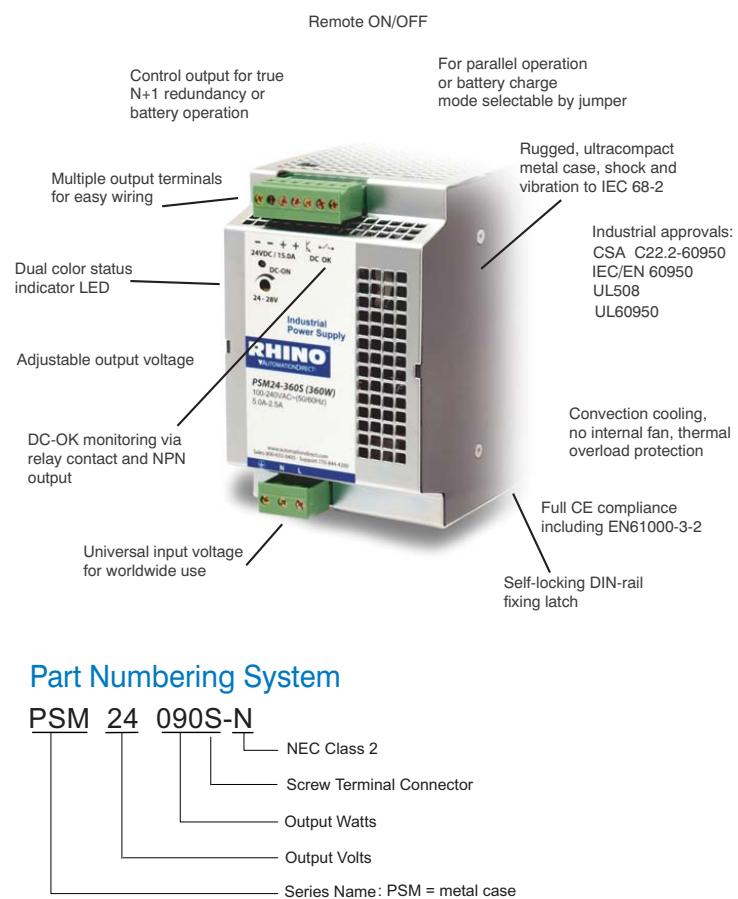
RHINO PSM Series Power Supplies

Versatile switching power supplies are DIN-rail mountable

AUTOMATIONDIRECT offers the most practical industrial control power supplies available. The RHINO PSM series power supplies are industrial grade switching DC output supplies with a sturdy steel case to withstand harsh environments. Autoselect inputs for 115 VAC or 230 VAC and international agency approvals make the RHINO PSM series suitable for worldwide use. RHINO PSM power supplies are available in 12 or 24 VDC output, with adjustable output voltages, and feature low output ripple along with overload and overtemperature protection. The seven models offer power ratings from 78W to 600W, and up to 25A output current.

Features

- Industrial grade design
 - Sturdy metal case to withstand harsh industrial environments
 - Model PSM24-0905-N meets NEC Class 2
 - Universal 100/230 VAC input voltage
 - Adjustable output voltage
 - Low output ripple
 - Short-circuit, overvoltage and overtemperature protection
 - Power Good signal
 - Remote ON/OFF
 - Optional wall mounting
 - Specialty modules for redundancy, power backup and UPS
 - Terminal connectors included
 - 3-year product warranty



Part Numbering System

PSM 24 090S-N

NEC Class 2
Screw Terminal Connector
Output Watts
Output Volts

Series Name: PSM = metal cas

RHINO PSM Industrial Power Supplies			
Part Number	*Output Voltage (V_{nom})	**Output Current (I_{max})	***Output Power (P_{max})
PSM12-078S	12 VDC	6.5 A	78 W
PSM24-090S	24 VDC	3.75 A	90 W
PSM24-090S-N	24 VDC	3.75 A	90 W
PSM12-156S	12 VDC	13.0 A	156 W
PSM24-180S	24 VDC	7.5 A	180 W
PSM24-360S	24 VDC	15.0 A	360 W
PSM24-600S	24 VDC	25.0 A	600 W

**12V models adjustable from 12 to 14 VDC. 24V models adjustable from 24 - 28 VDC*

****Maximum current at nominal output voltage**

***Up to an operating temperature of +40°C

RHINO PSM Series Power Supplies Specifications

Input Specifications									
Part Number	Input Voltage Range	Input Frequency Range	Input Current (Typical) at full load		Inrush Current max (<2ms) @ +25°C		Holdup Time	Efficiency (Typical) @ 115VAC	Circuit Breaker or Fuse (slo-blo)
			115 VAC	230 VAC	115 VAC	230 VAC			
PSM12-078S	100 - 240 VAC 85 - 264 VAC (47 - 63 Hz)	47-63 Hz	2.0 A	1.0 A	<12 A	<20 A	20 ms min. (full load 115/230 VAC)	82%	6.0 A to 16.0 A
PSM24-090S			2.1 A	1.0 A				85%	
PSM24-090S-N			2.1 A	1.0 A				85%	
PSM12-156S	100 - 120 VAC/ 220 - 230 VAC 85 - 132 VAC/ 187 - 264 VAC (47 - 63 Hz) Autoselect	47-63 Hz	2.5 A	1.4 A	<13 A	<25 A		85%	10.0 A to 16.0 A
PSM24-180S			2.8 A	1.5 A			88%		
PSM24-360S			5.0 A	2.5 A	<16 A	<25 A		87%	
PSM24-600S			10.0 A	5.0 A	<25 A	<30 A		89%	16.0 A to 25.0 A

Output Specifications										
Part Number	Price	Output Voltage	Output Voltage Adj. Range	Output Current (Max.)	Output Power (Max.)	Output Overvoltage Protection	Power - Good Signal		MTBF (IEC 61709 @ 25°C)	
							Trigger Threshold	Active Output Signal		
PSM12-078S	<-->	12 VDC	12 - 14 VDC	6.5 A	78 watts	20 V	9 - 11 V	11 V ± 1 V/20 mA max.	350,000 hours	
PSM24-090S	<-->	24 VDC	24 - 28 VDC	3.75 A	90 watts	35 V	18 - 22 V	22 V ± 2 V/10 mA max		
PSM24-090S-N	<-->			3.75 A	90 watts	35 V				
PSM12-156S	<-->	12 VDC	12 - 14 VDC	13.0 A	156 watts	20 V	9 - 11 V	11 V ± 1 V/40 mA max.		
PSM24-180S	<-->	24 VDC	24 - 28 VDC	7.5 A	180 watts	35 V	18 - 22 V	22 V ± 2 V/20 mA max		
PSM24-360S	<-->			15.0 A	360 watts	35 V				
PSM24-600S	<-->			25.0 A	600 watts	35 V				

General Specifications	
Specification	Description
Temperature	Operating (ambient): -25°C to + 70°C max (-13°F to 158°F). Above +40°C(104°F) load derating Storage (non-operating): -25°C to + 85°C max (-13°F to 185°F). Temperature drift: 0.02%/C. Cooling: convection, no internal fan
Humidity	95% (non-condensing) relative humidity maximum
Isolation	According to IEC/EN 60950, EN50178, EN61558-2-8, EN60204, CSA
Output Regulation	Input variation: 0.5% maximum. Load variation (10 to 100%): 0.5% maximum
Output Voltage Ripple	100 mV peak-to-peak typical (20 MHz bandwidth), (200 mV peak - peak maximum at I _{max})
Output Protection	Current limit: 110% constant current, automatic recovery, thermal protection, output rating, Voltage limit: 140% V _{out} nom
Over-temperature Protection	Switch off at over-temperature, automatic restart
Status Indicator	Dual color LED (green: DC Ok; Red: DC Off)
Remote ON/OFF	By external contact. DC On: -S contact open. DC Off: -S connected via 1 kΩ to -V _{out} , [3VDC max across V _{out} (+) and V _{out} (-)]
Maximum Capacitive Load	Unlimited
Vibration	IEC 60068-2-6: 3 axis, sine sweep, 10-55 Hz, 1g, 1 oct/min
Shock	IEC 60068-2-27: 3 axis, 15g half sine, 11ms
Enclosure Rating	IP20 (IEC 529)
Enclosure Material	Aluminum (chassis) / zinc plated steel (cover)
Mounting	Snap-on with self-locking spring for 35mm DIN rails per EN 50022-35x15/75, or wall mount with bracket
Connection	Pluggable screw terminals (plugs included) 2 terminals per output (not available in 600 watt unit.)
Agency Approvals	UL 508 Listed File E157382, UL 60950 Recognized File E198298; CSA C22.2-60950 File 229285; CE
<i>Note: Unless otherwise stated all specifications are valid at nominal input voltage, full load and +25°C after warmup time.</i>	

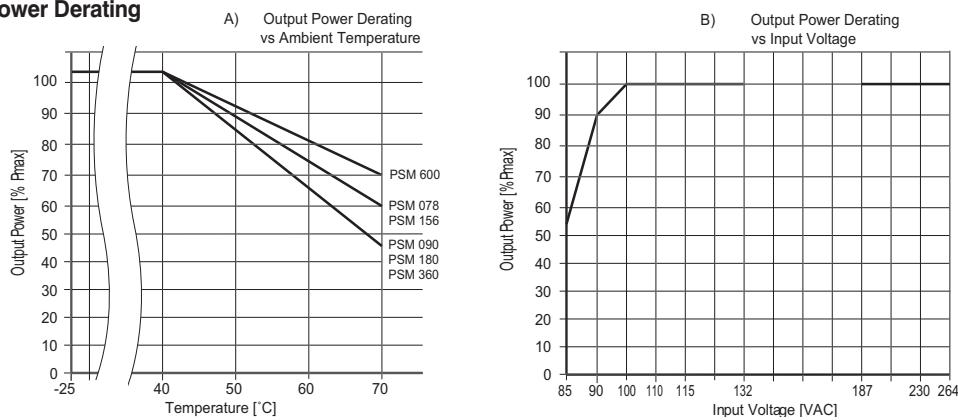


RHINO PSM Series Power Supplies Specifications

General Specifications (continued)		
Specification	Standard	Document Number
Harmonic Limits	Harmonic Current Limits	EN 61000-3-2, Class A for limited output power
Safety Standards	Information technology equipment	IEC/EN60950; CSA 60950-1-03/UL 60950-1
	Industrial control equipment	UL 508
	Electrical equipment of machines	EN 60204
	Electronic equipment for power installation	EN 50178
	Safety, transformers	EN 61558-2-8
	Limited power source (model PSM24-090S-N)	EN 60950 sect. 2.5 and NEC Class 2
Safety Approvals	CB-Report per IEC 60950	EN 50178, EN 60079-15 EN 61558-2-8, CSA
Safety Class	Degree of electrical protection Class1	IEC 536
Electromagnetic Compatibility (EMC), Emissions	EMC, Emissions	EN 61204-3, EN61000-6-3
	Conducted RI suppression on input	EN 55011 class B, EN 55022 class B
	Radiated RI suppression	EN 55011 class B, EN 55022 class B
	EMC, Immunity	EN 61000-6-2, EN 61204-3
	Electrostatic Discharge (ESD)	IEC / EN 61000-4-2 4 kV (contact discharge) / 8 kV (air discharge)
	Radiated RF field immunity (80-1000 MHz)	IEC / EN 61000-4-3 10 V / m
Electromagnetic Compatibility (EMC), Immunity	Electrical fast transient / burst immunity	IEC / EN 61000-4-4 2 kV
	Surge immunity	IEC / EN 61000-4-5 1 kV / 2 kV
	Immunity to conducted RF disturbances (0.15 to 80 MHz)	IEC / EN 61000-4-6 10 V
	Power frequency field immunity	IEC / EN 61000-4-8 30 A / m
	Voltage dips	IEC / EN 61000-4-11(70% UN Crit. B/40%/100% UN Crit. C)
Pollution Degree	2*	

*Note: Normally, only non-conductive pollution occurs. Temporary conductivity caused by condensation is to be expected.

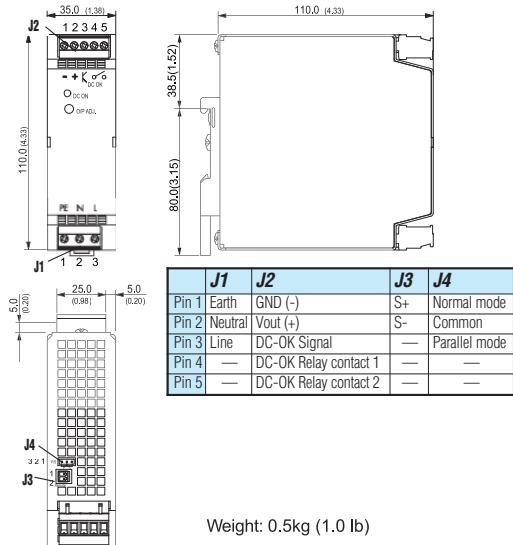
Output Power Derating



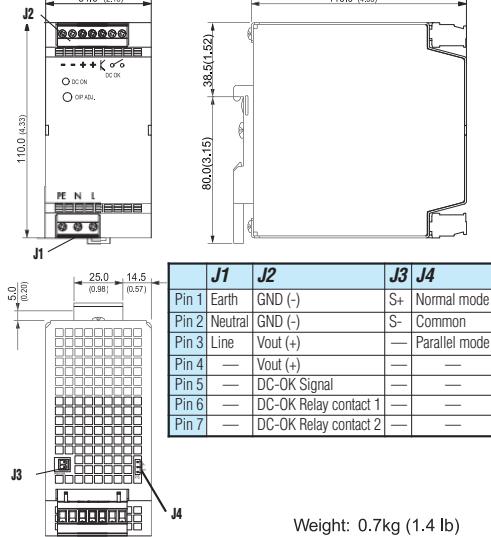
Note: Unless otherwise stated, all specifications are valid at nominal input voltage, full load and +25°C after warmup time.

RHINO PSM Series Dimensions/Connections

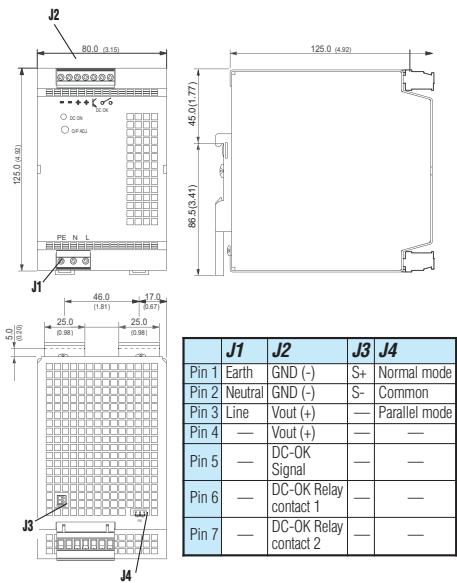
**PSM12-078S, PSM24-090S,
PSM24-REM360S, PSM24-BCM360S**



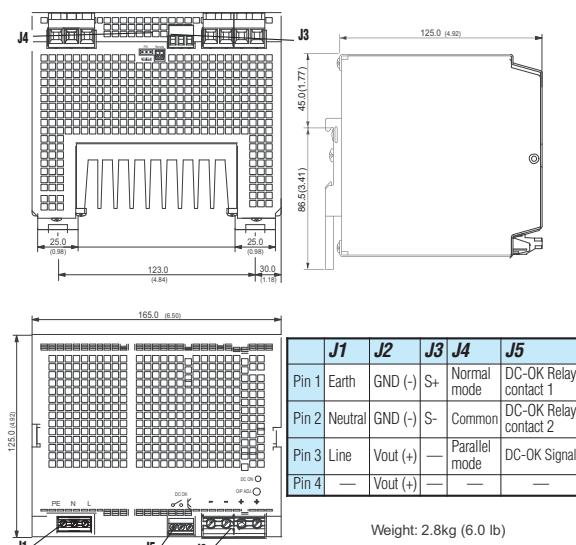
**PSM12-156S, PSM24-180S,
PSM24-BFM600S**



PSM24-360S



PSM24-600S



All dimensions in millimeters (inches)
Tolerances: ±0.5mm (±0.02")

Automation Direct	PLC Overview
	DL05/06 PLC
	DL105 PLC
	DL205 PLC
	DL305 PLC
	DL405 PLC
	Field I/O
	Software
	C-more HMIs
	Other HMI
	AC Drives
	Motors
	Steppers/ Servos
	Motor Controls
	Proximity Sensors
	Photo Sensors
	Limit Switches
	Encoders
	Current Sensors
	Pushbuttons/ Lights
	Process
	Relays/ Timers
	Comm.
	TB's & Wiring
	Power
	Circuit Protection
	Enclosures
	Appendix
	Part Index

Encoder Technology's HD2.0 Heavy Duty

Industrial Encoder is a compact version of the rugged HD2.5. This robust unit incorporates all the features of the HD2.5 including cast aluminum housing, stainless steel shaft, 100# load rated industrial ball bearings, mil spec sealed connector, all metal code disk, single monolithic phased array opto IC with integrated commutation signals, reverse voltage, ESD and EMI protection, single IR light source, and no internal electrical bias or mechanical adjustments. The HD2.0 is ideally suited for machine tool, process control, robotics, textile, pulp and paper factory automation, elevator control, and many other industrial applications. Covered by a three year warranty (one year for bearings) the HD2.0 provides quality, reliability and value available only through Encoder Technology's proprietary design and manufacturing capabilities.

Technical Specifications**Mechanical**

Shaft diameter	As specified
Flat on shaft	0.60 long x 0.018 deep
Shaft loading	Up to 100 lbs axial and radial
Shaft runout	0.0005 TIR at midpoint
Shaft	303 stainless steel (passivated)
Starting torque at 25°C	<i>Without shaft seal:</i> 1.0 in-oz. maximum <i>With optional shaft seal:</i> 2.5 in-oz. maximum
Bearings	5200 ZZ double row
Bearing life	5 x 10 ⁸ revs at rated shaft Loading, 5 x 10 ¹¹ revs at 10% of rated shaft loading. (manufacturers' specs)
Housing and cover	Die Cast Aluminum
Disc material	Metal or mylar
Moment of inertia	1.5 x 10 ⁻⁴ oz-in-sec ²
Weight	11 ounces, typical

Electrical

Code	Incremental
Cycles per Revolution	See "Current Resolutions" list
Supply voltage	See ordering information
Current requirements	50M (no load condition)
Output format	Channels A and B
Output format options	In quadrature ± 15° electrical Index, complementary outputs, and commutation signals on ET7272, ET7273 only
Output IC's	2N2222, ET7272, ET7273
Illumination	LED
Frequency response	125 kHz (data and index)
Output termination	See Table 1
Circuit Protection	Reverse over voltage and output short circuit

Environmental

Operating temp	-40 to 100°C
Operating temp ATEX	-40 to 80°C
Storage temperature	-40 to 100°C
Shock	50G's for 11msec duration
Vibration	5 to 2000Hz @ 20 G's
Humidity	98%RH without condensation
NEMA 4 and 13	When ordered with shaft seal

The above specifications are subject to change without notice.
Dimensions shown in inches.

HD2.0 Heavy Duty Optical Encoder

for Harsh Industrial Environments with Space Limitations

**Ordering Information - This model series is available in an intrinsically safe version Certified to ATEX Ex ia IIB T4**

Example part number:

HD2.0 [] D [] B [] 37F [] SS [] 2000 [] ABZ [] C [] 72 [] S18 [] 28

1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11.

1. Housing Configuration

- Square Flange (standard) D
- Servo Mount (ø2.00) E

7. Output Channels

- Single A
- Dual quadrature AB
- Dual with index (standard) ABZ
- Communication signals available. Contact the factory.

2. Pilot Configuration

- 1.181 in. female(shift seal not available)... A
- 1.25 in. (standard)... B

8. Complements

- With Complements..... C (available on Outputs 72 and 73 only)
- Without complements (blank)

3. Shaft Diameter

- 0.2497/0.2495 25
- 0.3747/0.3745(standard) 37F
- 0.3935/0.3942 (10mm) 39
- "F" = flat on shaft..... i.e. 37F

9. Output ICs

- 2N2222 open collector (5 to 28V) 220C
- 2N2222 with pull-ups 221K or 222K

10. Output Termination

- Differential line drivers
- ET7272 (5 to 28V) 72
- ET7273 open collector (5 to 28V) 73
- MS3102R14S-6P (6 pin) S14
- MS3102R16S-1P (7pin) S16
- MS3102E18-1P (10 pin) S18
- Side Cable with seal (18° std) SCS18

4. Face Mount

- If not required (blank)
- Or specify..... F5, F12, or F28

11. Voltage - standard

- 5 to 28Vdc 28
- 5 to 28Vdc,in, 5Vdc out (ET7272 only).... 28/5

12. Voltage - ATEX

- 5Vdc 5
- 7 to 28Vdc 28
- 5 to 28Vdc,in, 5Vdc out (ET7272 only).... 28/5

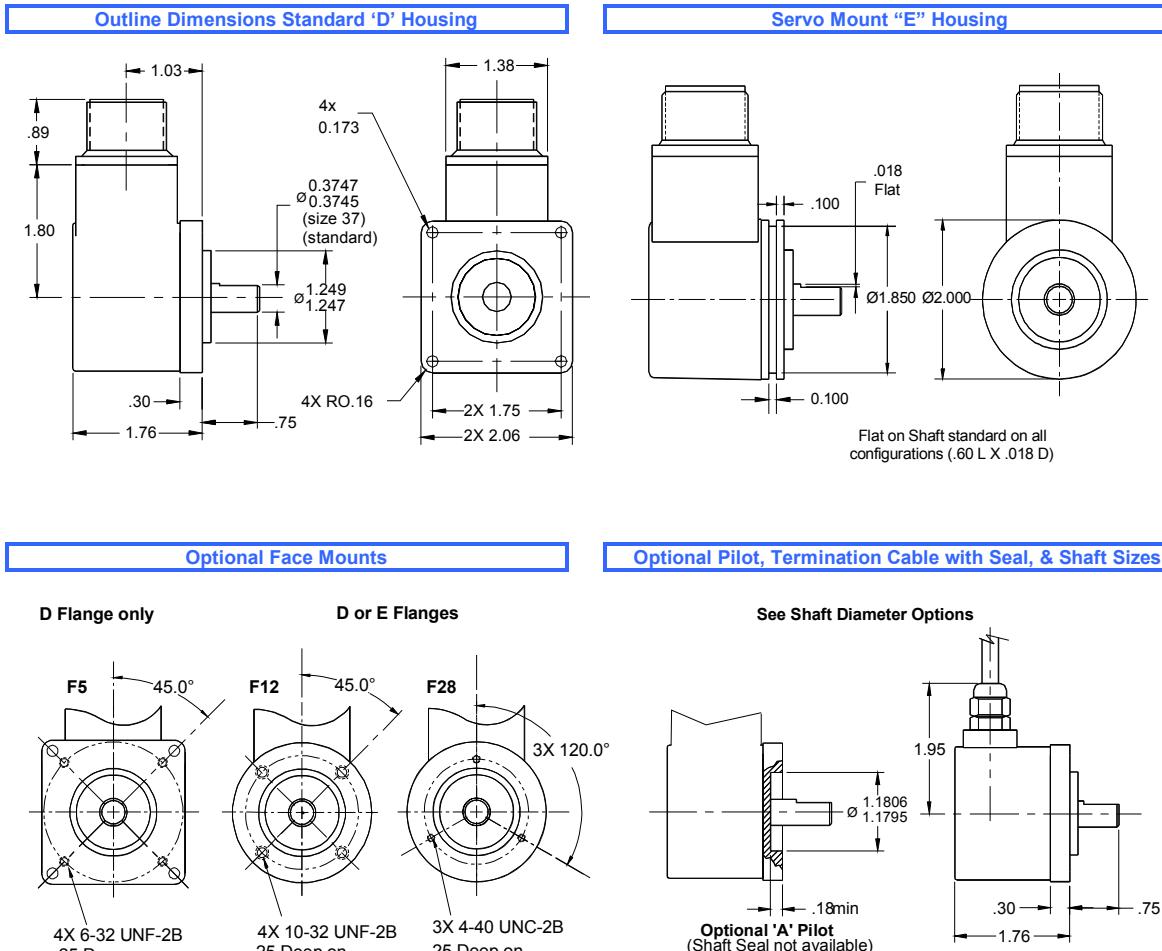


Table 1: Output Terminations (Pinout)							Output Format		
Connector Pins			Output Channels		Cable Termination				
M14	M16	M18	ABZ	ABC	ABZC	Wire Color	Function		
E	A	A	A	A	A	Green	A		
D	B	B	B	B	B	Blue	B		
C	C	Z	Z	A ^{Bar}	Z	Orange	Z		
B	D	+V _z	+V _x	+V _y	+V _x	Red	+Vdc		
F	E	N/C	N/C	B ^{Bar}	N/C	Black	Ground		
A	F	F	Circuit	Ground		Violet	A ^{Bar}		
G	G	Case Ground				B ^{Bar}			
H		N/C	N/C	A ^{Bar}		Yellow	Z ^{Bar}		
I		N/C	N/C	B ^{Bar}		White	Case Gnd		
J		N/C	N/C	Z ^{Bar}					

Case Ground not available on ATEX Certified Units

DATA AND INDEX
Not all complements shown.
A shown for reference.

(180° ELEC) → (90° ELEC)

Data A

Data Ā

Data B

Index

(Optional) COMMUTATION TRACKS
Not all complements shown.
C1 shown for reference.

1/3 Cycle

C1

C2

C3

2/3 Cycle

1 Cycle = 360° / N Mech

1/2 Cycle

A leads B, CCW

E LvROS Overview

OLIN/LABVIEW ROBOTICS ARCHITECTURE

ANDY BARRY | OLIN COLLEGE INTELLIGENT VEHICLES LAB | SUMMER 2009

ARCHITECTURE CONCEPTS

GOALS

- Parallel processing
- Hierarchical code structure interface
- Code reuse
- Not necessary to modify existing, tested, working code

CONCEPTS

We use a number of concepts we use that intermediate LabVIEW users are often not familiar with. Here we examine:

- Parallel processing with embedded **while** loops
- Queues
- Notifiers
- Thread spawning
- Variants
- Reentrant VIs

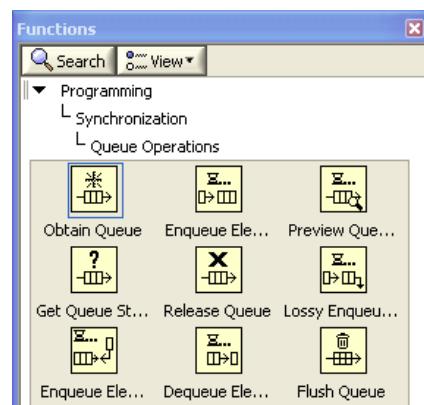
PARALLEL PROCESSING

To run a large number of sensors at different speeds efficiently, we use many parallel threads. Creating new threads in LabVIEW is very simple – just use two (not nested) while loops!

Given the number of sensors we require, a system with many loops all on the main back-panel would quickly become unreadable. Thus, we embed our while loops in Sub-VIs. It is important to understand that this makes our code look different than other programs you might be used to. Now, you might ask, how do we communicate between these independent threads?

QUEUES

The most difficult part of parallel processing is the communication between two processes. With both processes running in parallel we risk clobbering data, data loss, and deadlocks. LabVIEW solves this issue in an elegant way. LabVIEW's queues implement thread-



safe data passing, allowing us to provide the most recent data to processes regardless of thread speed.

QUEUE INITIALIZATION

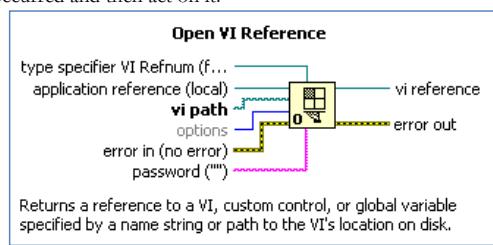
Unlike many LabVIEW functions, we do not need wires to obtain a reference to queues (we can use wires, we just do not have to). The **Obtain Queue** block allows us to specify the name of an existing queue and obtain the reference to that queue.

NOTIFIERS

Sometimes we require notification of new data, often new data on a queue. LabVIEW implements Notifiers which allow us to safely wait until a notification has occurred and then act on it.

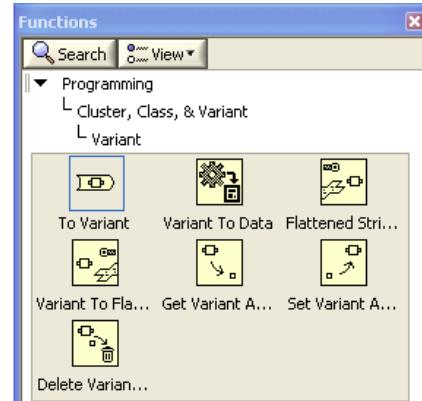
THREAD SPAWNING

Occasionally, we require the ability to launch VIs in a programmatic manner. For example, we do not know how many logging-replay threads we will require until we have started and the program is running and the user has specified the sensors to replay. Luckily, LabVIEW allows us to programmatically spawn threads using **Open VI Reference**. You should be careful here because we must reference VIs by file name and path.



VARIANTS

A variant in LabVIEW is a datatype that is “no type,” very similar to a void pointer in C. By using variants, we can utilize data in generic ways, such as logging and replaying your sensor, even though we do not know what datatype your sensor is. There are two blocks you need to know when using variants: **To Variant** and **Variant to Data**.



REENTRANT VIs

Some VIs, such as XMLLog.vi, need to be called more than once with different arguments. These VIs are set to be **reentrant**, so that more than one instance of them can be running at the same time. To set a VI to be reentrant, right click on its connector, choose *VI Properties*, then select *Execution* and check *Reentrant execution*.

USING THE SYSTEM

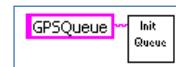
ADDING YOUR OWN CODE

Here we will step through adding new code (a sensor driver in this example) to the system.

INITIALIZATION

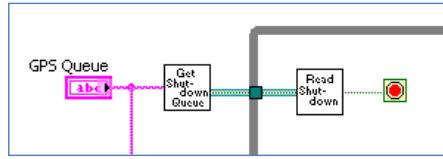
The first step in adding new code is to initialize the communication channels required.

For that we use the **Init Queue** block. It is important to use our Init Queue block because it adds logging support to your sensor. If you need a notifier you can specify that here.



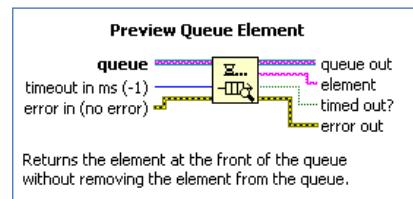
INTEGRATING SHUTDOWN

With your own **while** loop running inside your SubVI, you'll need some way to stop the VI when execution is completed. We have a **Shutdown Queue** block that you should always use to learn when to shutdown. **It is important to listen to the Shutdown Queue at least every 200-500ms.** If you do not, the system will not shutdown cleanly. This means that you **must** have timeouts on every item that could block (previewing queues, reading from ports, etc.)

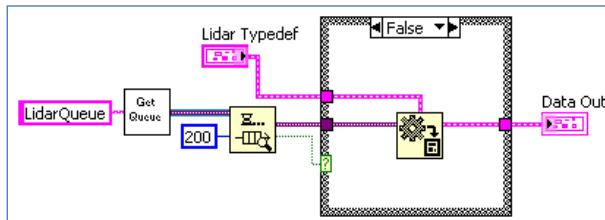


READING DATA

Our sensor might need to read data from another queue. All of our queues hold only **one** piece of data at a time, so we must be careful to only overwrite when we mean to. In this way, only data writing systems dequeue data and all reading systems preview the queue. Thus, ten processes can read a queue and not damage each other. For you, this means you should always use **Preview Queue** to read data from a queue. As stated above for shutdown reasons, you should always use a timeout with a case statement catching the timeout.



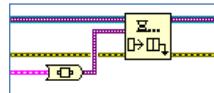
Here's a quick example how to read data:



2. Preview Queue (with timeout)
3. Check for timed out
 - a. If true, do nothing
 - b. If false, convert variant to data
4. Use data

WRITING DATA

Once we have new data to write to a queue, we will want to write it to our queue. This process is very easy – we simply convert to a variant and **Lossy Enqueue**. We use a **lossy** enqueue to overwrite existing (now old) data that is on the queue.



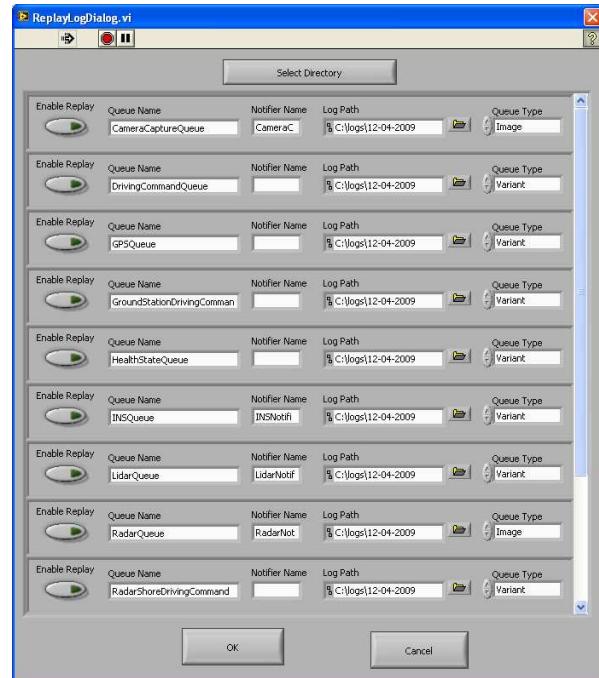
LOGGING AND REPLAY

One of the system's goals is to make sure that you don't have to write code that isn't specific to your application. Thus, we take care of all of the logging and replay for the system. If you use our **Init Queue** block, we automatically log your data. Note that if you require images you should use the **Init Image Queue** block and use the **Image Queue** typedef instead of a variant.

If you do not want your queue to be logged, you need to add it to the Log Exclusion List (currently a constant in the main code, soon to be added to a configuration file).

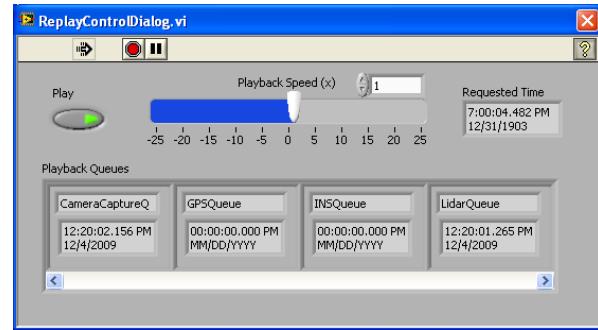
REPLAY AND WHY YOU CARE

Logging replay is designed to allow you to run into a problem in the field and debug it in the lab. Say, for example, you have an obstacle avoidance algorithm that fails in the field and runs into the obstacle. The problem appears to be too complicated on the spot so you scrub the test. Back in the lab, using replay you can **step through each input** and determine what went wrong. In addition, you



can also **implement new code** and **test the new code** on the old, real-life data. Thus, you can make sure your new algorithm works in the case that your old one did not without leaving the lab. In a sense, we have provided a bit of a **simulation environment** for free.

To use replay, simply uncheck **Enable Logging** and start Main. Then click **Configure Replay Settings** and select the logs to replay and check **Enable Replay** for the queues you want to replay. These queues will be sent shutdown commands and then they will be filled from data that comes out of the log file.



Once you click **OK** a dialog will appear that allows you to control playback speed for all sensors or each individual sensor. The rest of the system **does not know** that you are replaying data and thinks that the information on the queues is coming directly out of the sensors.

F Table of Queues

PC Queues: Queue Name	Notifier Name	Description	Typedef	Queue Type	Notes
AutoDriveCommandQueue	AutoDriveCommandNotifier	Holds the autonomous driving command (used when the user selects "Auto" mode)	VehicleControl.ct: Cluster containing desired velocity in MPH, Desired steering curve radius, and state (Run, Pause, Self-Test)	Variant	
CameraCaptureQueue	CameraCaptureNotifier	Holds the current camera image.	Camera image.	Image	
CleanRadarQueue		Cleaned RADAR image data.	Image queue.	Image	
CrioNetwork		cRIO network status	Boolean	Variant	
DeadReckonQueue		Attempts to provide location data using the INS. Not very far along.	Dead Reckoning.ct: Cluster of absolute position in X, Y, Z and velocities in X, Y, Z.	Variant	
GPSQueue		Holds current GPS data.		Variant	
GroundStationDrivingCommandQueue		Groundstation driving command	Not implemented yet	Variant	Not implemented yet.
GroundStationNetwork		Status of the ground station network connection	Boolean	Variant	
HealthStateQueue		Groundstation communication queue	Determined by groundstation (To be implemented)	Variant	Not used at the current time.
INSQueue	INSNotifier	Holds untransformed INS data	INSData.ct: Cluster of roll, pitch, yaw, x, y, z accelerations, and x, y, z, angular rates.	Variant	Currently written to but not read from.
INSTransQueue		Transformed INS data.	Where Am I.ct	Variant	

LidarQueue	LidarNotifier	Holds untransformed LIDAR data	LidarPolarControl.ct: Array of polar coordinates, scanning angle constant, and scan resolution constant (constants must be matched to the LiDAR's configuration)	Variant
LidarStopQueue		Alerts to a LiDAR obstacle and commands a stop	LidarStopControl.ct: Cluster containing a boolean Lidar Stop value.	Variant
LidarXYZQueue	LidarXYZNotifier	Contains LiDAR hits that have been transformed to be in XYZ coordinate space using all data available (including INS).	LidarXYZ.ct: Cluster of three arrays (X, Y, Z)	Variant
LocalCoordsQueue		Information about local coordinates. Not fully implemented	Cluster containing doubles for X and Y.	Variant
LogQueue		Queue for logging that records what other queues and notifiers are in the system. InitQueue.vi writes to this queue.	Not a variant queue! Array containing clusters of queue names, notifier names, and queue type.	Other
LogReplyQueue		Information about what queues the user wants to replay	Not a variant queue! LogReplyControl.ct: an array of Enable Replay, Queue Name, Notifier Name, Log Path, and Queue Type items.	Other
ManualDriveCommandQueue	ManualDriveCommandNotifier	Holds the current manual driving commands.	VehicleControl.ct: Cluster containing desired velocity in MPH, Desired steering curve radius, and state (Run, Pause, Self-Test)	Variant
MissionMapQueue		Image queue that holds the mission map image for display on the GUI.	Image Queue with Image and variant containing: MissionMapDataControl.ct: Image Descriptor.	Image
MissionTimeQueue		Elapsed mission time	Double	Variant
NetworkStatusQueue		Status of the cRIO network	Boolean	Variant
NumberOfLogsQueue		Counts the number of logs running.	NumberOfLogs.ct: Cluster containing one number.	Variant
OccupancyGridQueue		Occupancy grid info (used in Path Planning).	OccupancyGrid.ct: Cluster containing 2D array for grid, grid resolution (m/cell), grid size (m), edge tolerance (%), grid size (number of cells), GPS origin (latitude and longitude).	Variant

ReplayUserQueue	Hold the user's current desired replay time.	ReplayCommandControl.cti: Cluster containing time desired.	Variant
ShutdownQueue	Generic shutdown queue. If it is true, the system is shutting down.	Boolean queue (not a variant)	Other
SystemStatusGrdQueue	Groundstation system status. Holds all of the data coming from the cRIO for the current vehicle state. This includes steering, gas, and brake feedback, speed encoder, and system status / self test commands. This queue is managed exclusively by SelectDriveCommand.vi.	SystemStatus.cti: timestamp, system status, emergency stop, notifications (string), error VehicleControlFeedback.cti: Speed in MPH, Gas Voltage, Brake Voltage, Current radius of curvature, Self test cluster (48, 24 12V power, Steering centered, Speed encoder online, Hardware e-stop, system e-stop).	Variant
VehicleControlFeedbackQueue	Queue that sends the cRIO driving commands.	VehicleControl.cti: Cluster containing desired velocity in MPH, Desired steering curve radius, and state (Run, Pause, Self-Test)	Variant
VehicleControlQueue	Contains the user's selections for the vehicle state (run/pause, program select (auto, xbox, manual mode, etc)).	Vehicle State.cti: Cluster of run/pause boolean, program select (ProgramSelectionControl.cti). Waypoint Command.cti: Cluster of Waypoint Array, containing WaypointControlLowestLevel.cti which has a current waypoint index.	Variant
WaypointCommandQueue	Holds the current waypoint array and current waypoint target (as an index in the waypoint array).	Where Am I.cti: Cluster containing Course over Ground, latitude, longitude, and velocity. Cluster also contains current waypoint index.	Variant
WhereAmIQueue	Holds all Where Am I data, including latitude, longitude, velocity, acceleration, course over ground, speed over ground.	Speed over Ground, Cluster of Position (lat, long, etc), Cluster of Velocity, Cluster of Acceleration, and clusters for position, velocity, and acceleration confidence.	Variant
XboxCommandQueue	Xbox game controller commands.	VehicleControl.cti: Cluster containing desired velocity in MPH, Desired steering curve radius, and state (Run, Pause, Self-Test)	Variant
XboxGamepadQueue	Xbox game controller state. Each sensor has its own shutdown queue (created by InitShutdownQueue.vi). This allows us to turn off the sensor driver when replying.	XboxGamepad.cti: Cluster containing button states and axis states.	Variant
<QueueName>ShutdownQueue	Boolean queue (not a variant)		Variant

cRIO Queues: Queue Name	Notifier Name	Description	Typedef	Queue Type Notes
EstopQueue		Contains information about the e-stop status, including what is and isn't e-stopped.	cRIOEstopQueueControl.ct: Cluster of e-stop status, e-stop wait counter, PC e-stop, network comm e-stop, and hardware e-stop.	Variant
ManualControlQueue		Contains cRIO's manual control information.	cRIOManualControl.ct: Cluster of steering position, gas voltage, and brake voltage.	Variant
ManualNetworkQueue		Contains if the cRIO is in manual or network control mode (true = in network control)	Boolean	Variant
NetworkControlStatusQueue		Displays if the cRIO's network connection to the PC is online.	Boolean	Variant
SelfTestQueue		Contains current self-test status.	cRIOSelfTestControl.ct: Cluster containing 48, 24, 12V power, Steering centered, Speed encoder online, Hardware e-stop, system e-stop.	Variant
SpeedOutputVoltagesQueue		Contains actual gas/brake voltages.	cRIOSpeedOutputVoltages.ct: Cluster with gas and brake voltage numbers.	Variant
SpeedRequestVoltagesQueue		Contains the desired gas/brake voltages from the speed controller.	cRIOSpeedOutputVoltages.ct: Cluster with gas and brake voltage numbers.	Variant
StatusQueue		Contains 48/24/12V status and e-stop status.	cRIOStatus.ct: Cluster of Hardware e-stop, 48V, 24V, 12V status.	Variant
SteeringControl		Steering command from the Auto block that manages steering self-test, run, and pause states.	Position Control Command.ct: Cluster containing radius of curvature.	Variant
SteeringFeedback		Contains the current steering wheel encoder position and left and right limit switch status.	Position Control Feedback.ct: Cluster containing radius of curvature, steering speed (deg/sec), left limit switch status, and right limit switch status.	Variant
SteeringResetQueue		Contains steering zero position and the self-test state. When the system auto-centers, it writes to this queue.	cRIOSteeringResetControl.ct: Cluster of zero position and self-test state status (cRIOSteeringSelfTestStates.ct)	Variant

SwitchedSteeringControl	Steering command that goes to actual steering system. Is connected to either SteeringControl or ManualControl queues.	Position Control Command ct: Cluster containing radius of curvature.	Variant
VehicleControlQueue	Contains requested velocity (MPH), desired Vehicle Control.ct: Cluster of desired velocity (MPH), steering curvature, and current state.	dRIOVehicleSpeed.ct: Cluster of wheel encoder position, wheel ticks/ms, and speed in MPH.	Variant
VehicleSpeedQueue	Contains information from the speed encoder.		

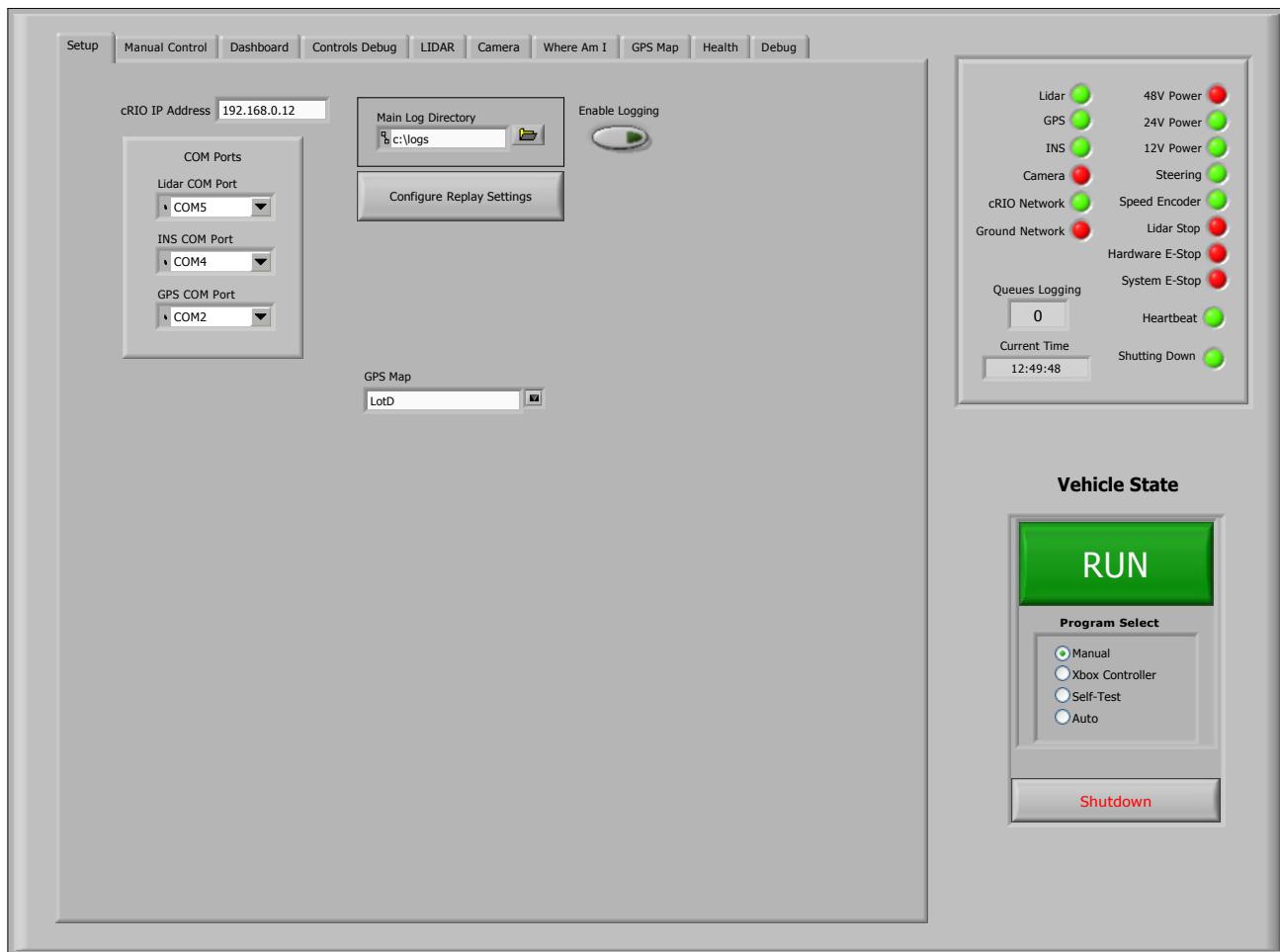
G LvROS Code

Connector Pane

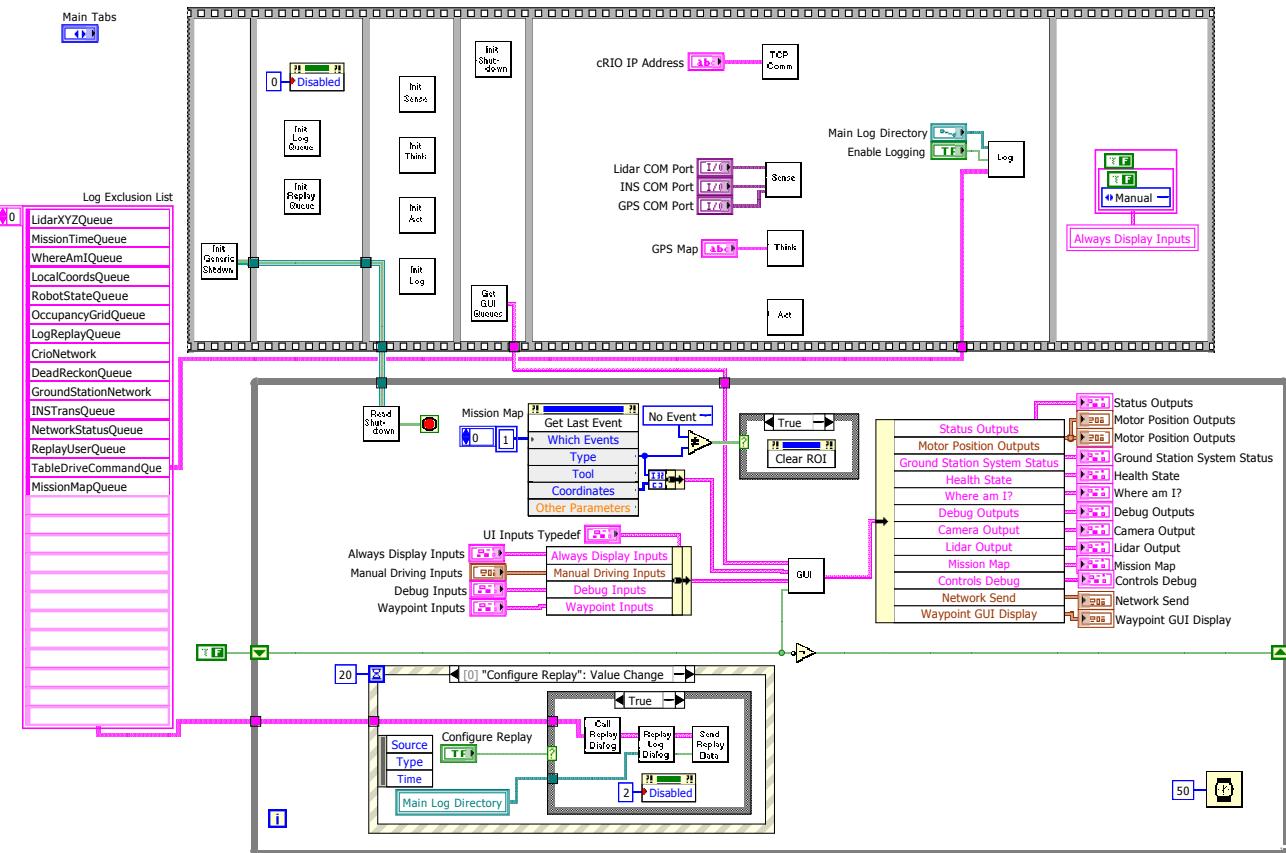
Main.vi

LabVIEW Robot Operating System.

Front Panel



Block Diagram

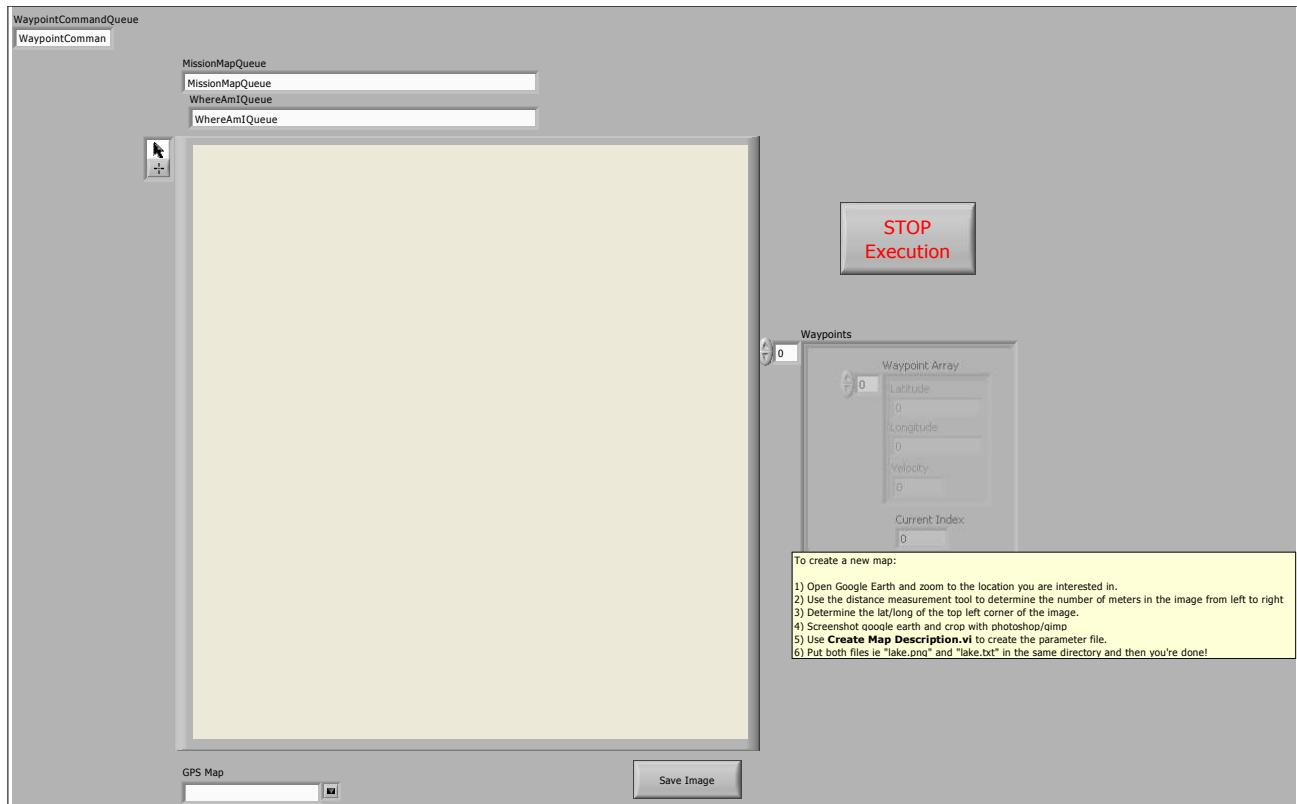


Connector Pane

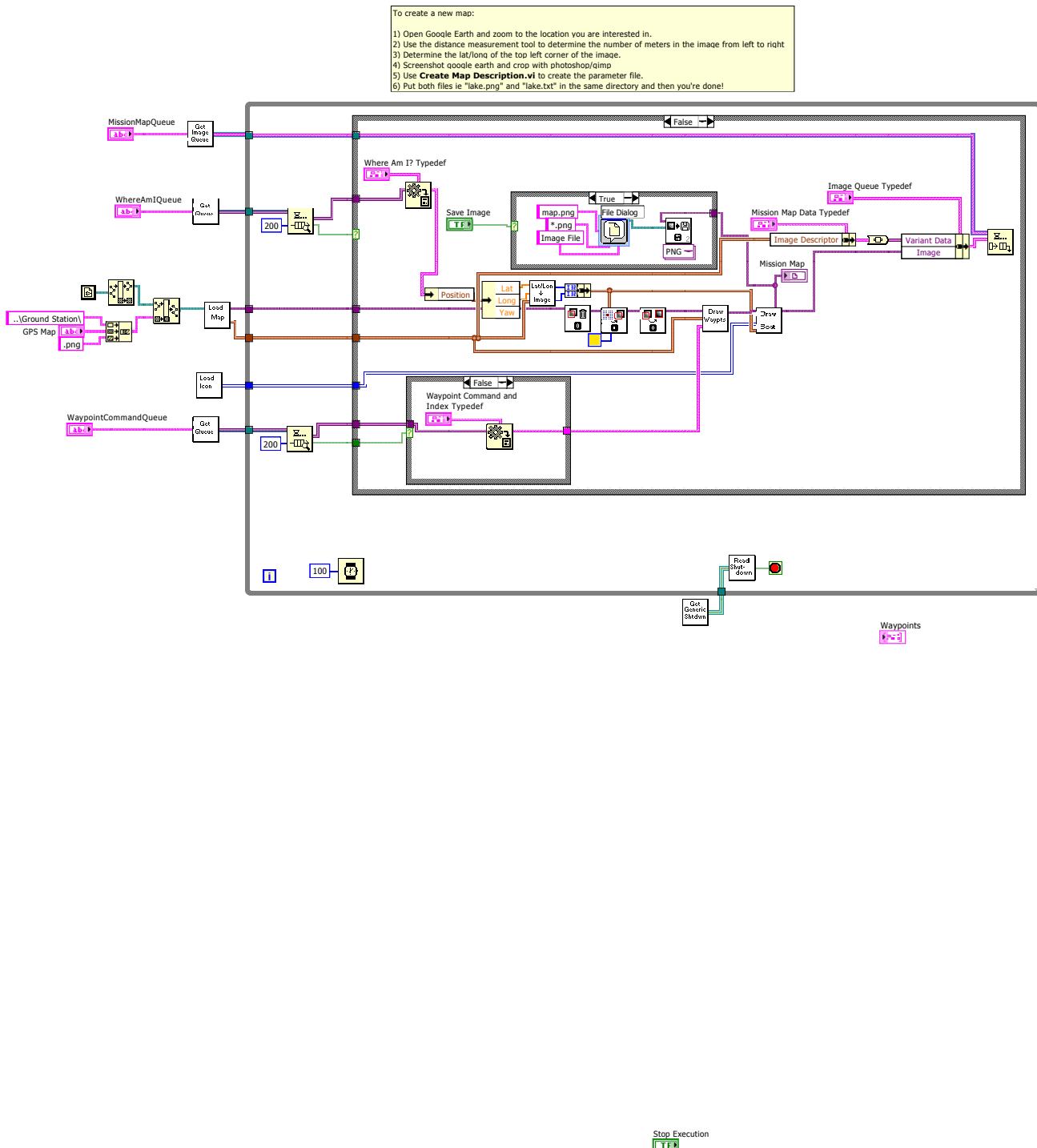
MissionMap.vi

Function that creates the GPS Mission Map.

Front Panel



Block Diagram



Connector Pane

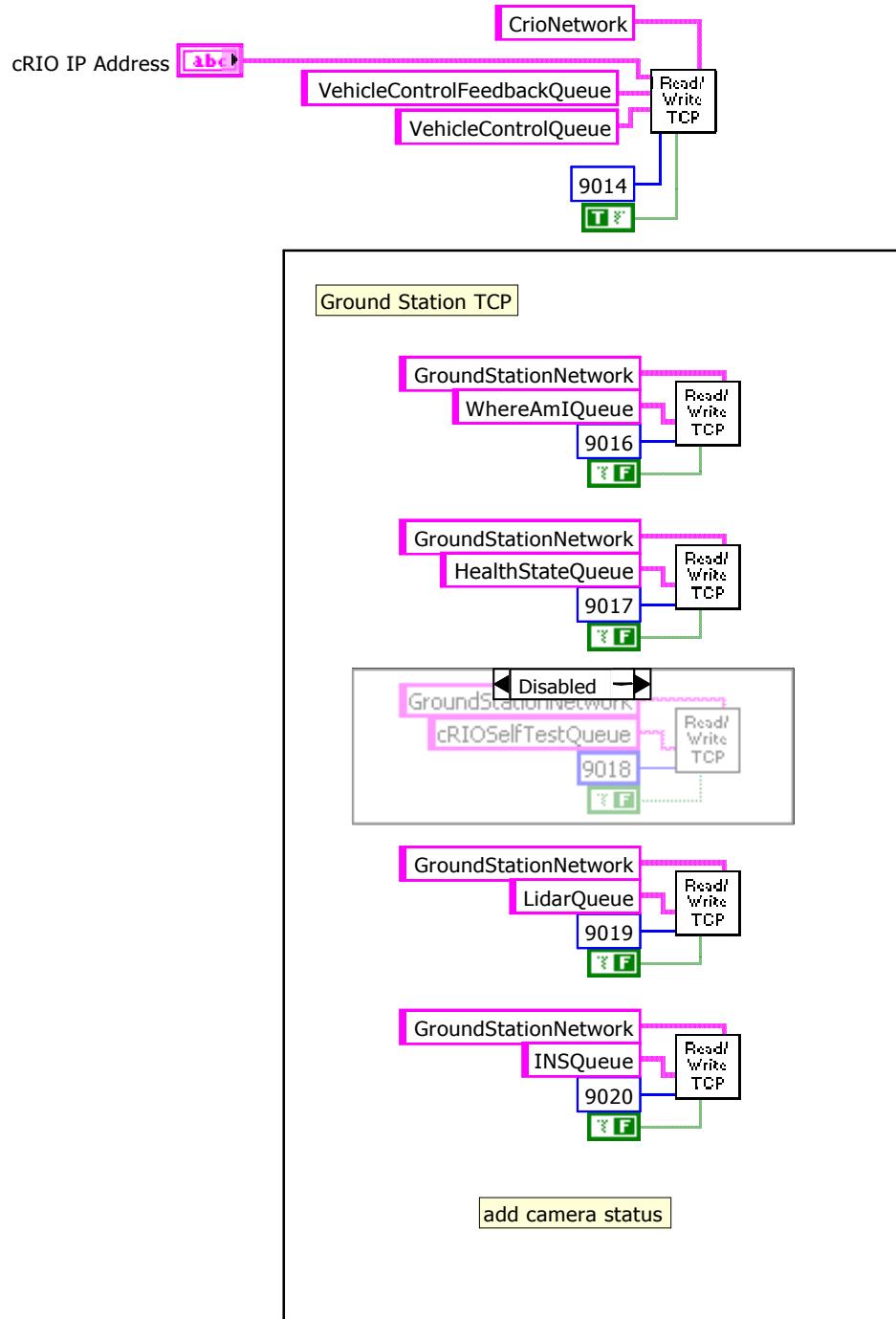
TCPMain.vi

Manages all TCP network communications.

Front Panel



Block Diagram

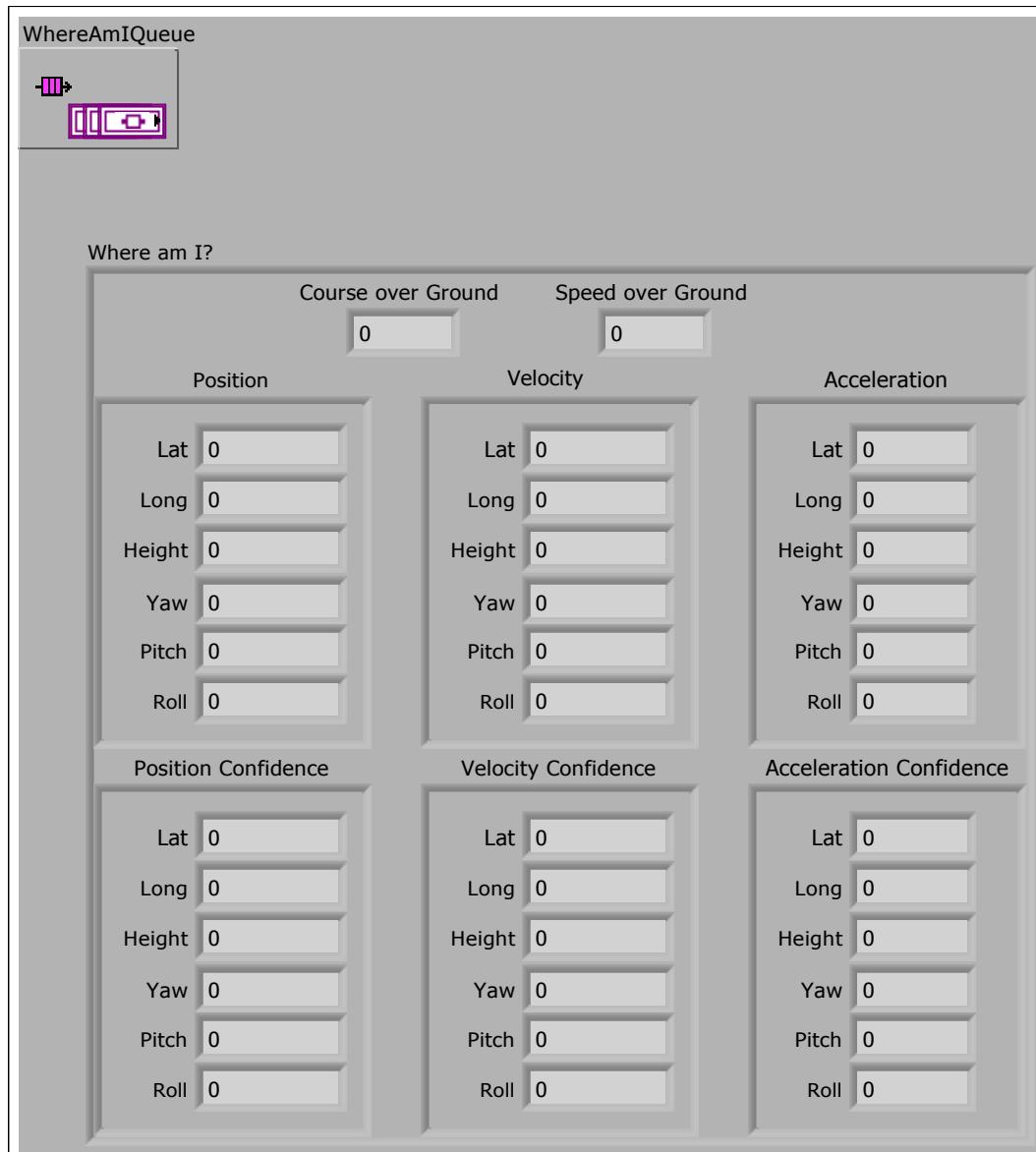


Connector Pane

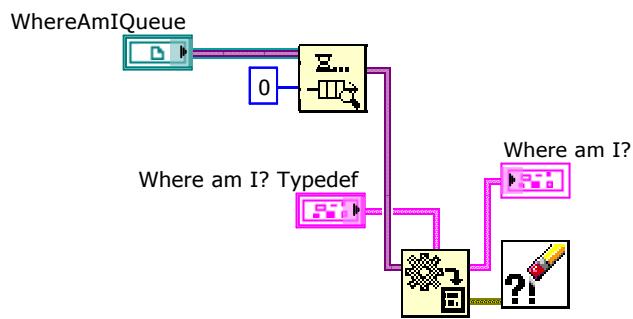
GetWhereAmIState.vi

Reads the state of the WhereAmIQueue, the HealthStateQueue, and the SystemStatusGrdQueue.

Front Panel



Block Diagram



Connector Pane

Speedometer.ctl

Front Panel



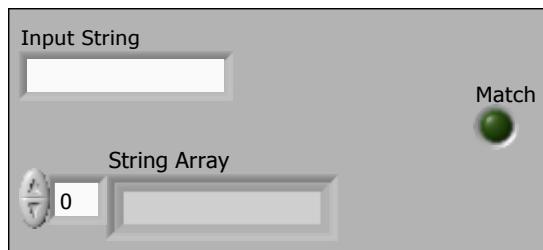
Block Diagram

Connector Pane

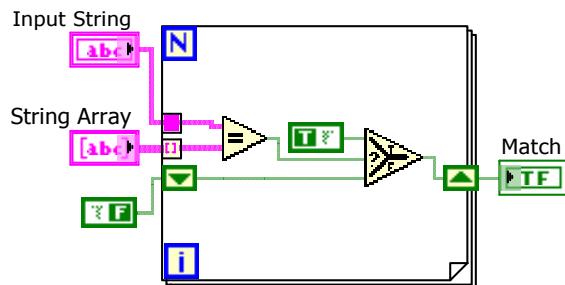
StringInArray.vi

Determines if a string is in an array of strings.

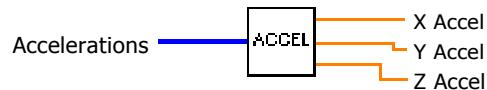
Front Panel



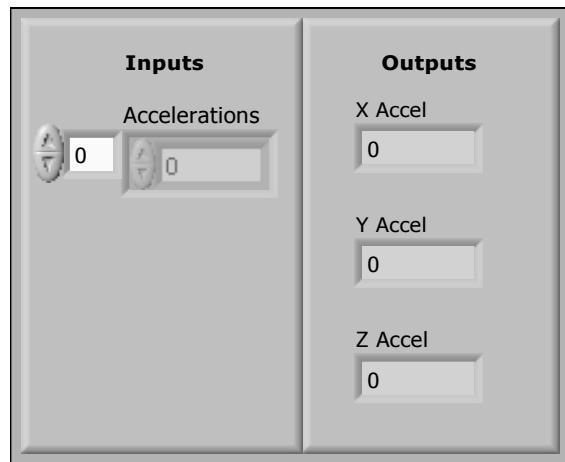
Block Diagram



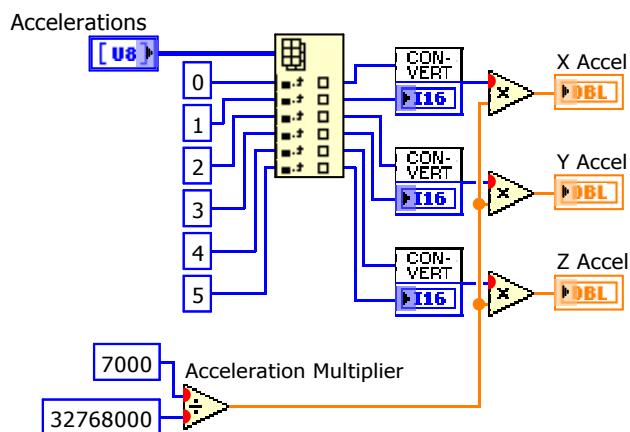
Connector Pane

Interpret Accelerations.vi

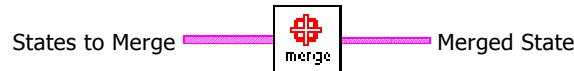
Front Panel



Block Diagram



Connector Pane

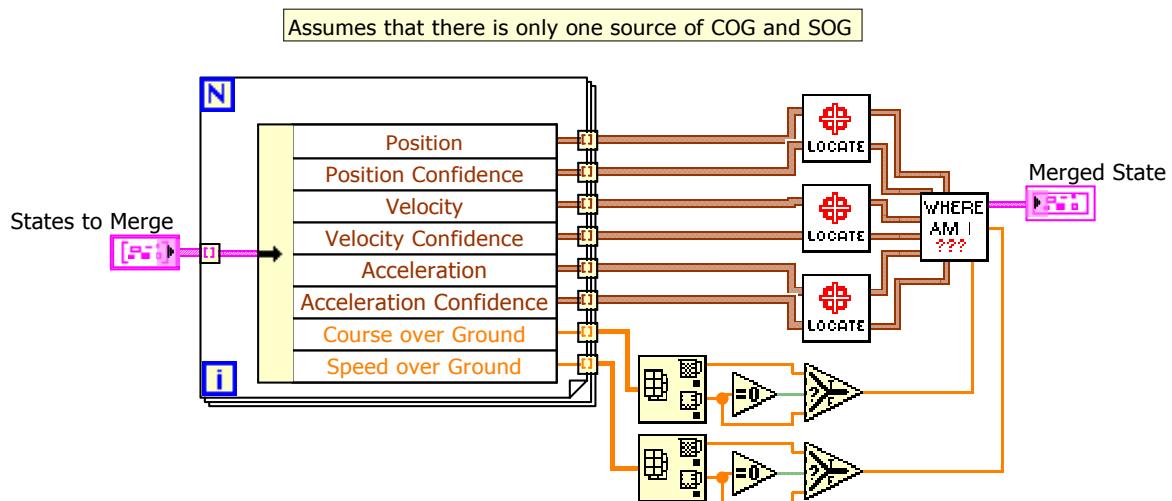
Merge States.vi

Takes in sensor data from multiple sources and finds the weighed average according to corresponding confidences.

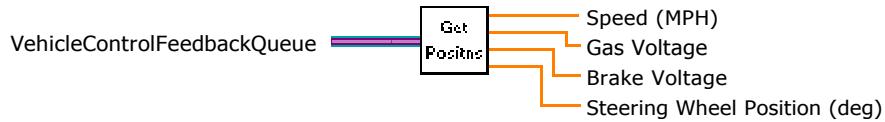
Front Panel



Block Diagram

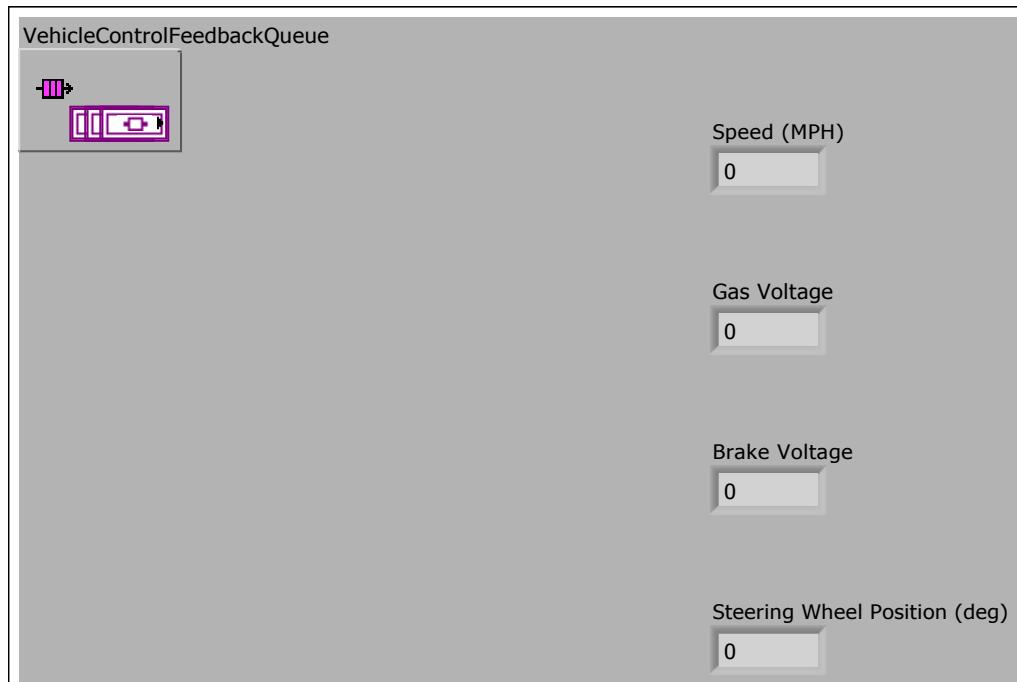


Connector Pane

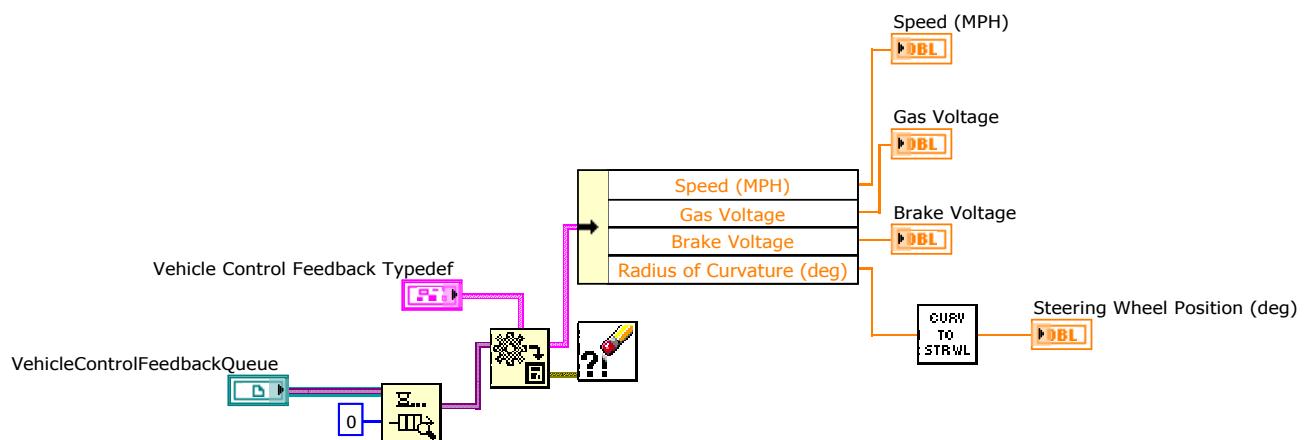
GetPositions.vi

Reads queues to determine motor positions.

Front Panel



Block Diagram

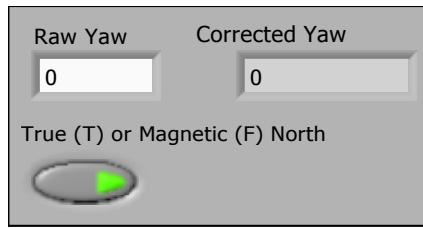


Connector Pane

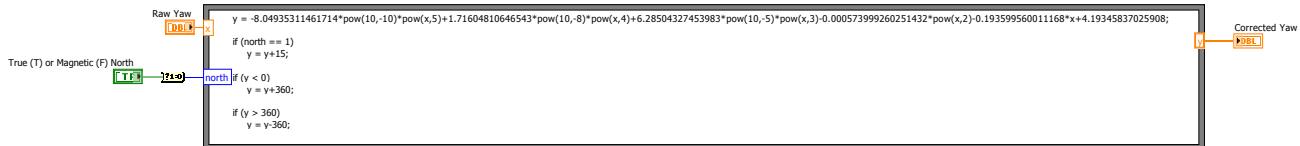
INS Yaw Correction Algorithm.vi

We turned the INS in a circle and it didn't give us data like it had turned in a circle. So we took the data, plotted them in Excel, and fit a curve to them. This is that curve. Supposedly it corrects the INS data, but it doesn't really. Don't use it.

Front Panel



Block Diagram



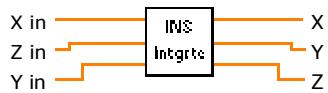
Connector Pane

Health State.ctl

Front Panel

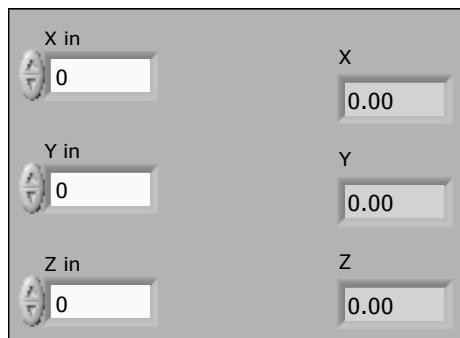
Block Diagram

Connector Pane

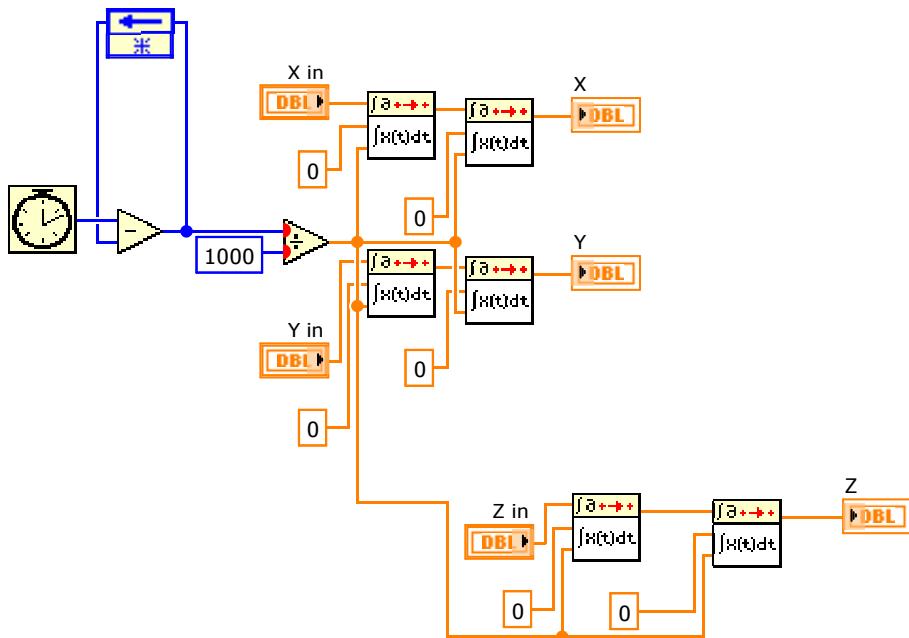
INS Integrate.vi

Integrates INS data over time.

Front Panel



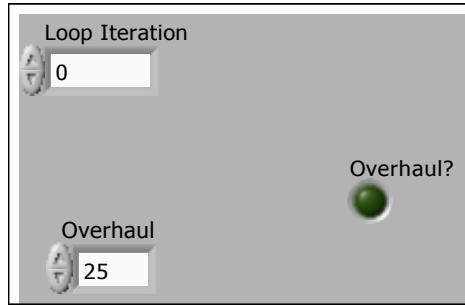
Block Diagram



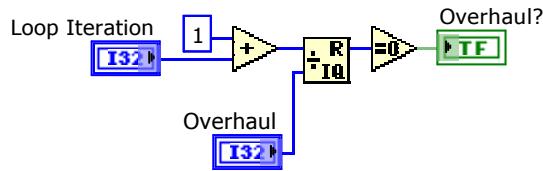
Connector Pane

Overhaul.vi

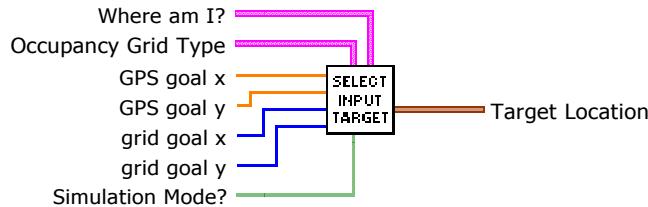
Front Panel



Block Diagram

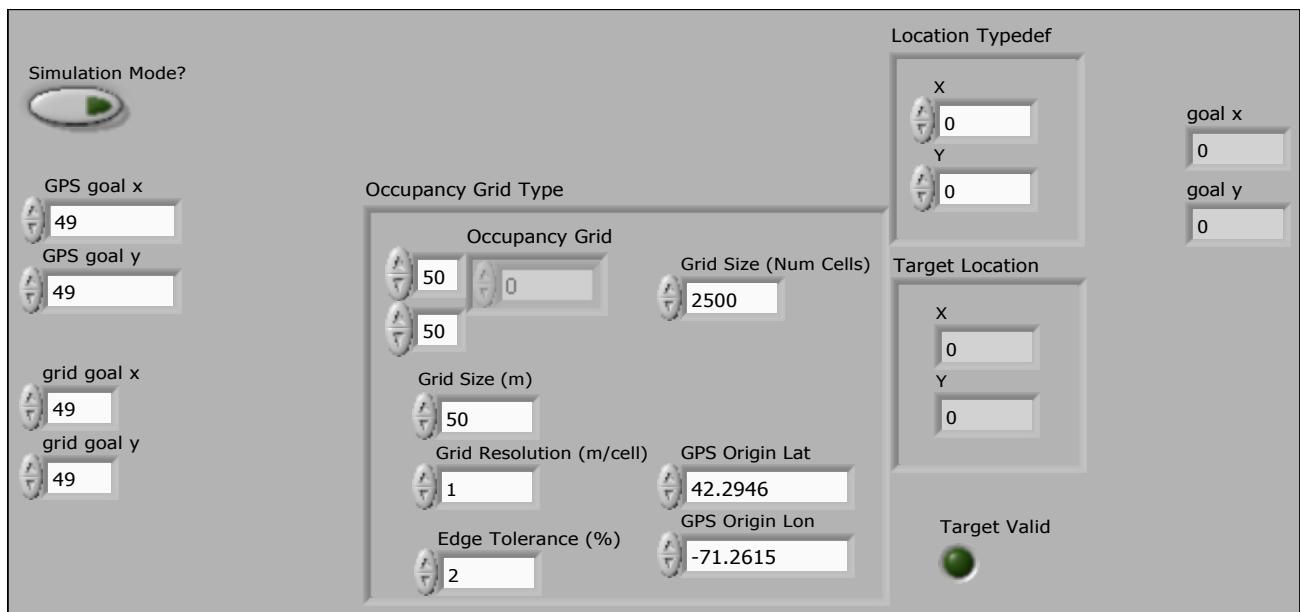


Connector Pane

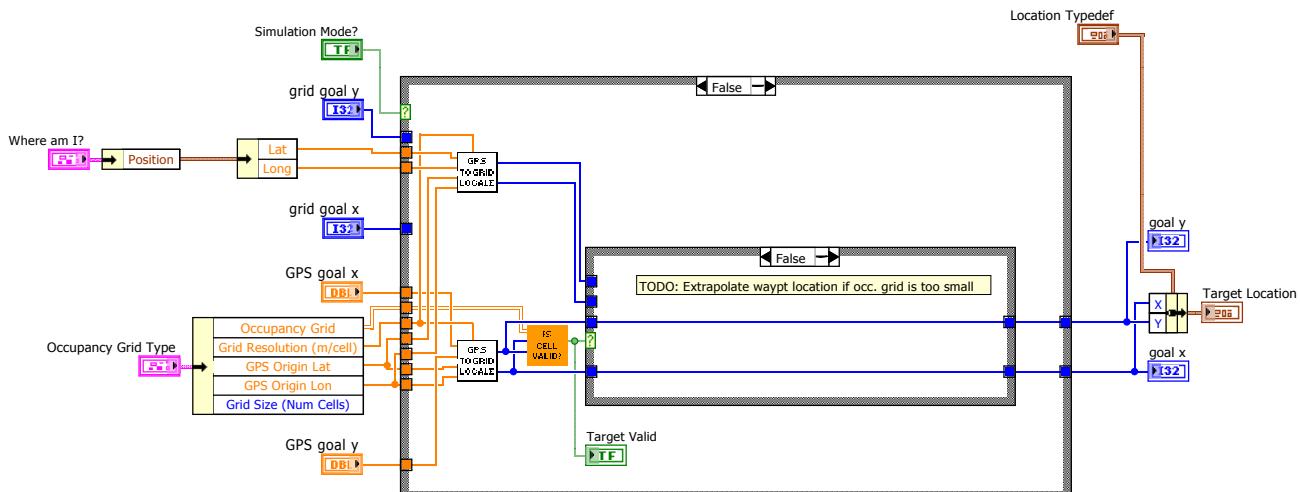
Select Target.vi

Selects either GPS or grid target location based on whether we are in simulation or waypoint-following mode.

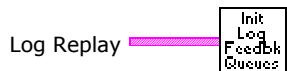
Front Panel



Block Diagram



Connector Pane

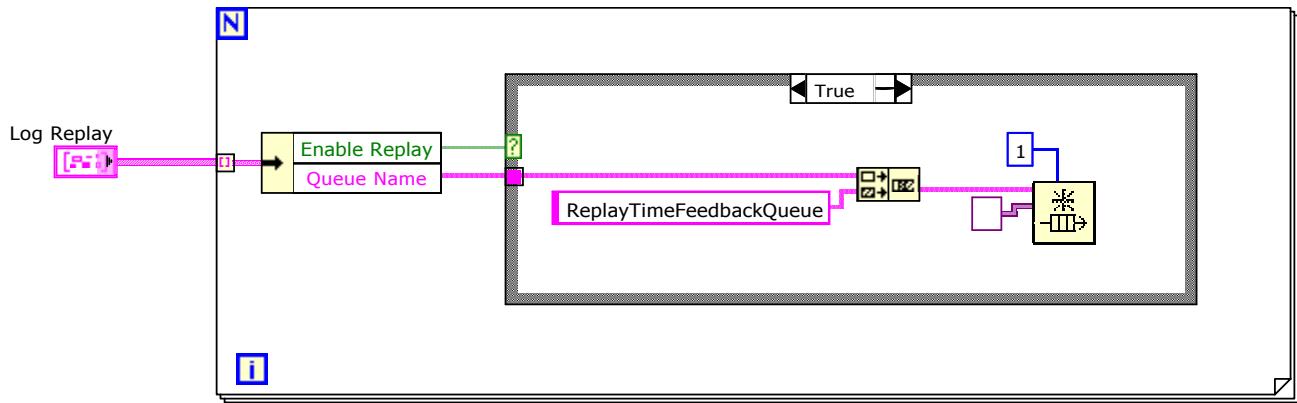
InitLogReplayTimeFeedbackQueues.vi

Initialize the replay feedback queues.

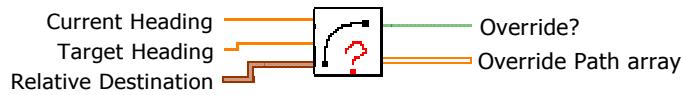
Front Panel



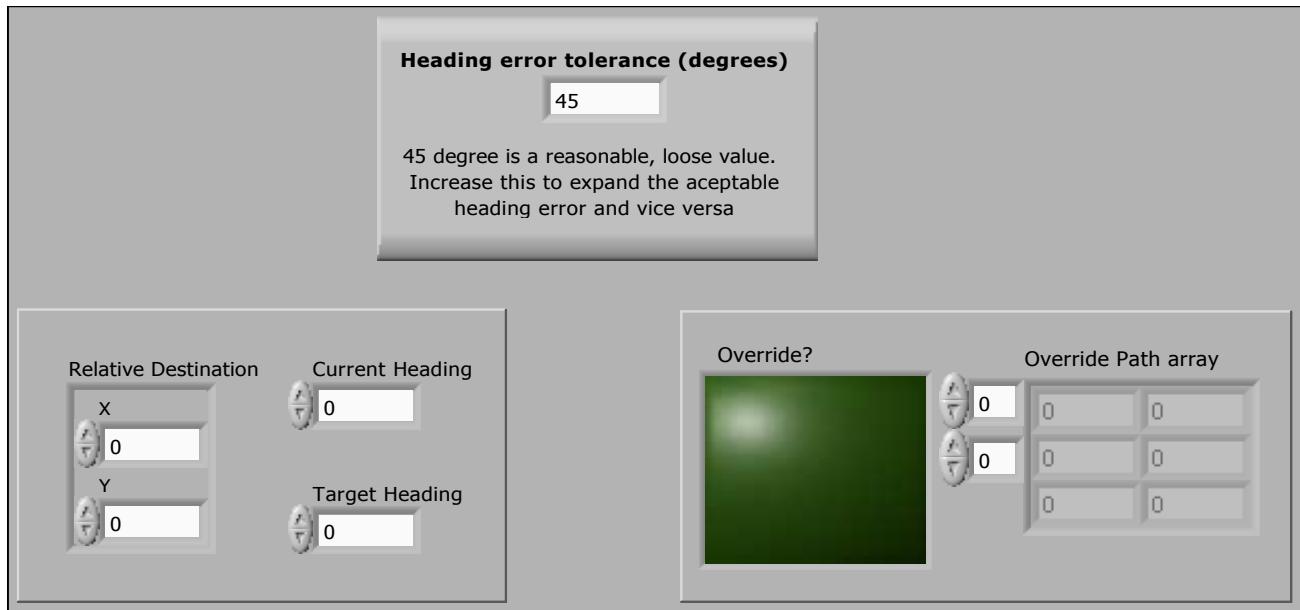
Block Diagram



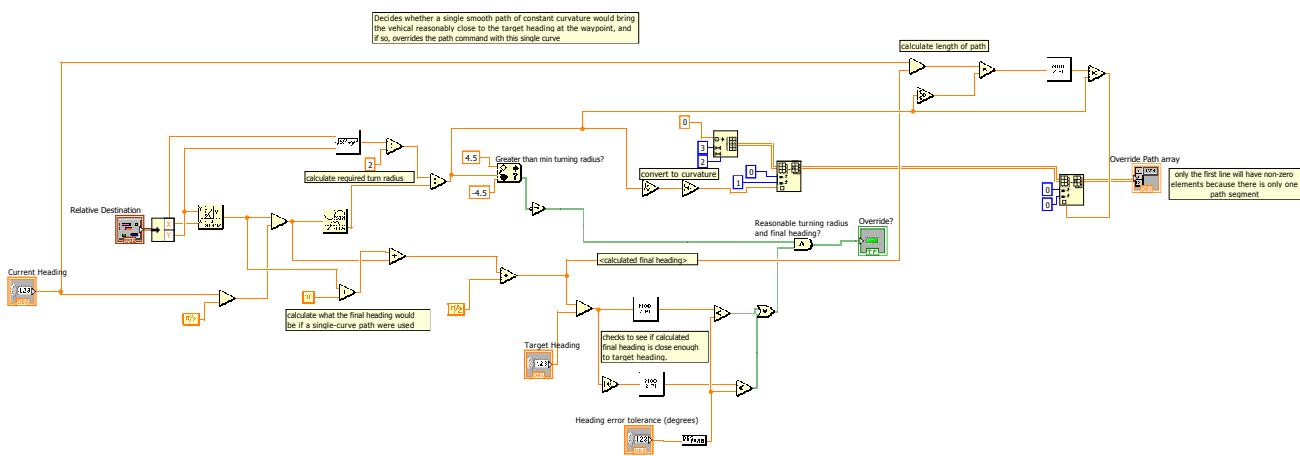
Connector Pane

SingleCurveOverride.vi

Front Panel



Block Diagram



Connector Pane

LogQueues.vi

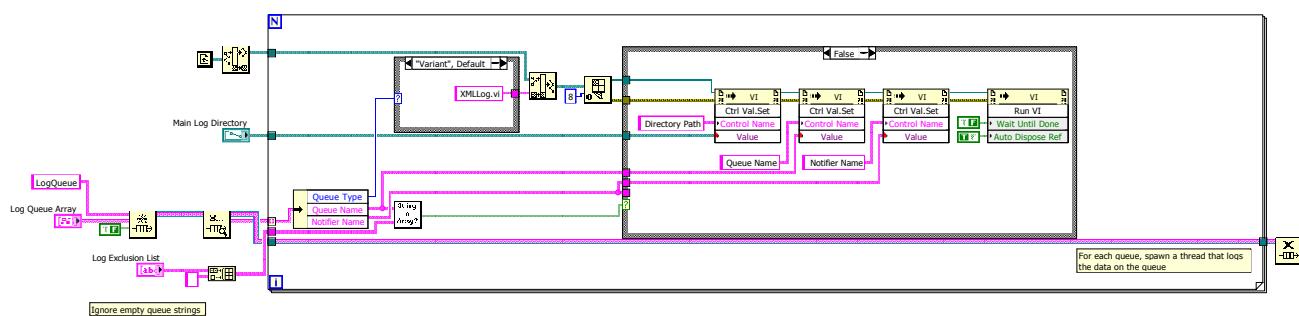
Spawns threads for logging. In other words, for each system that must be logged, runs either **XMLLog.vi** or **XMLImageLog.vi** with arguments including path, queue name, and notifier name.

Note that this calls **XMLLog.vi** and **XMLImageLog.vi** by name so be careful renaming or moving those files.

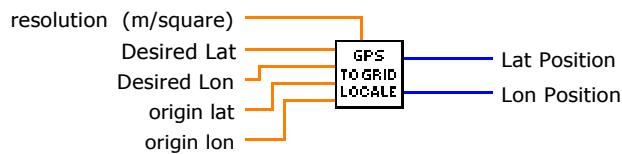
Front Panel



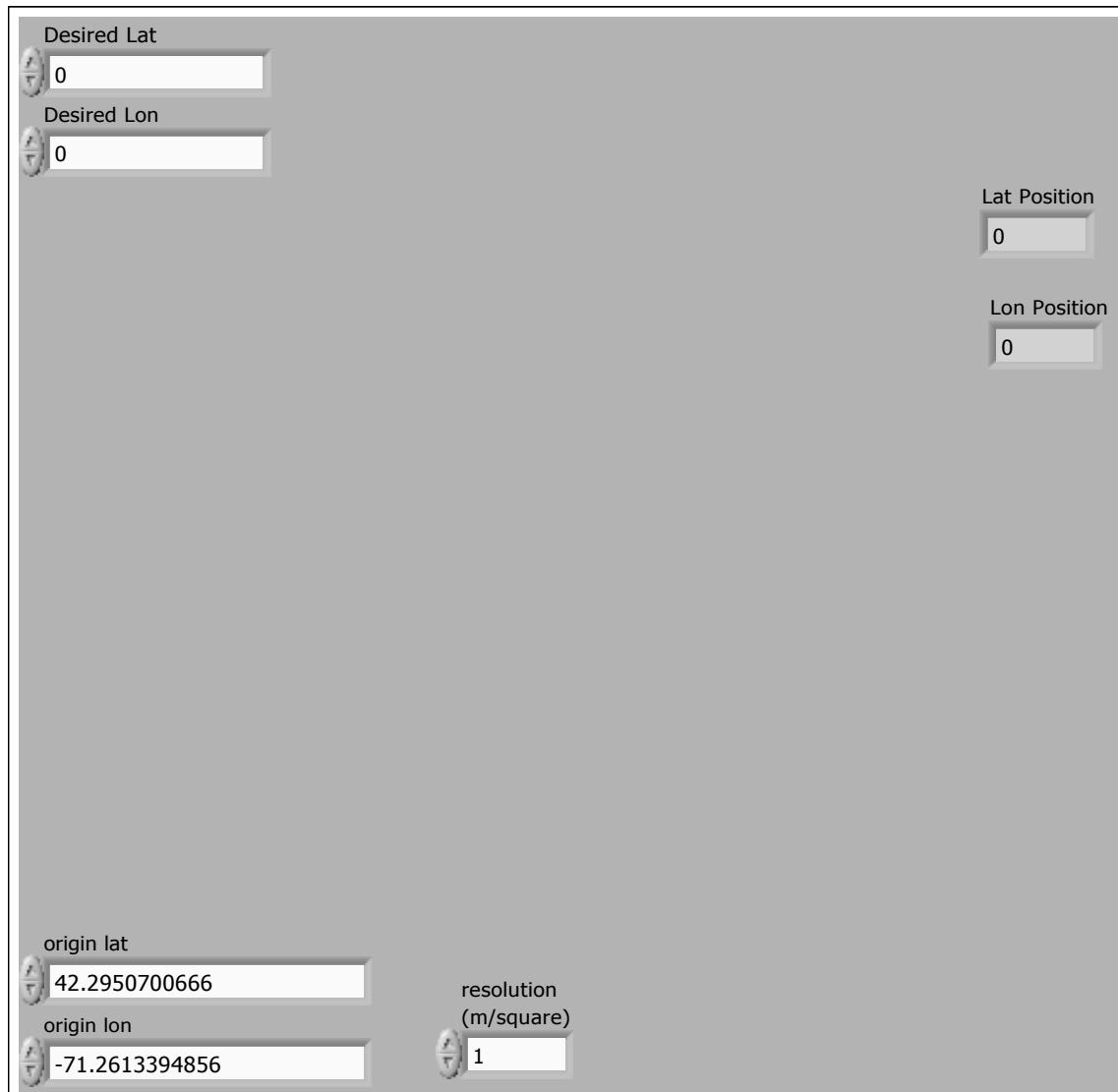
Block Diagram



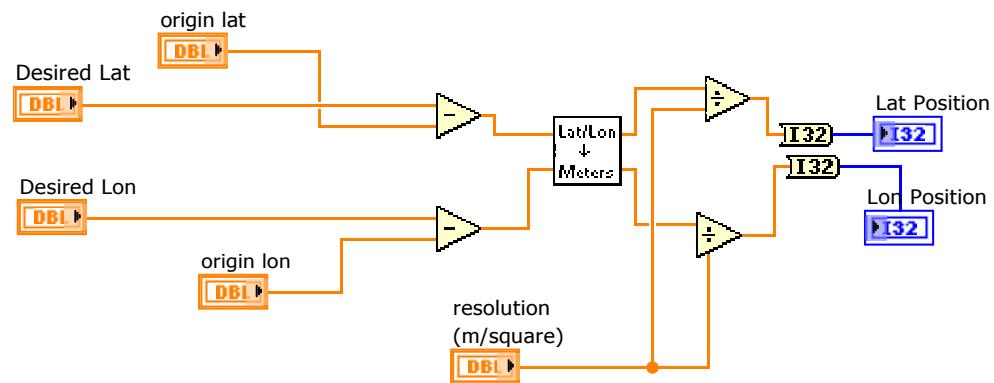
Connector Pane

GPS coords to Grid Position.vi

Front Panel



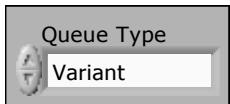
Block Diagram



Connector Pane

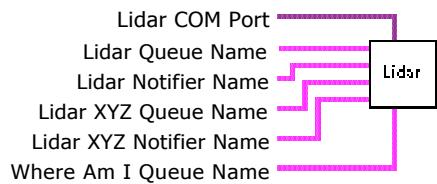
LogTypes.ctl

Front Panel



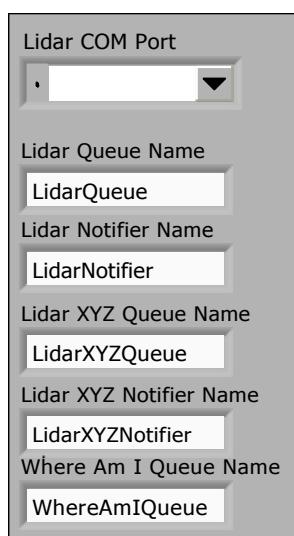
Block Diagram

Connector Pane

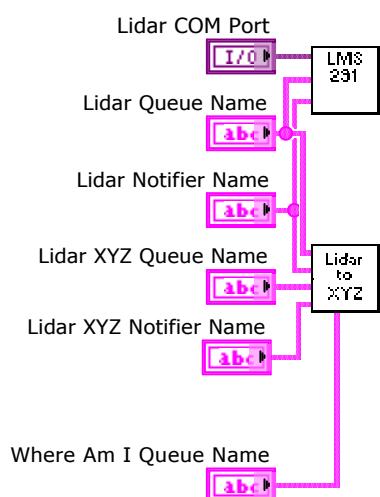
Lidar.vi

Runs a LIDAR driver.

Front Panel



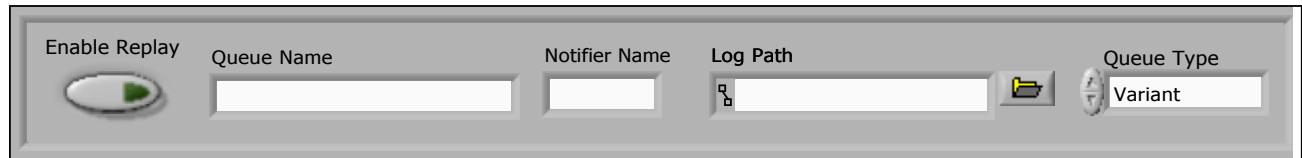
Block Diagram



Connector Pane

LogReplayInitControlIndexed.ctl

Front Panel



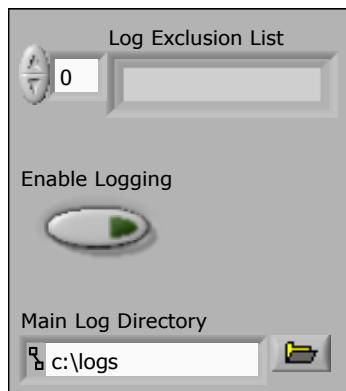
Block Diagram

Connector Pane

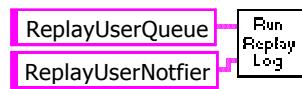
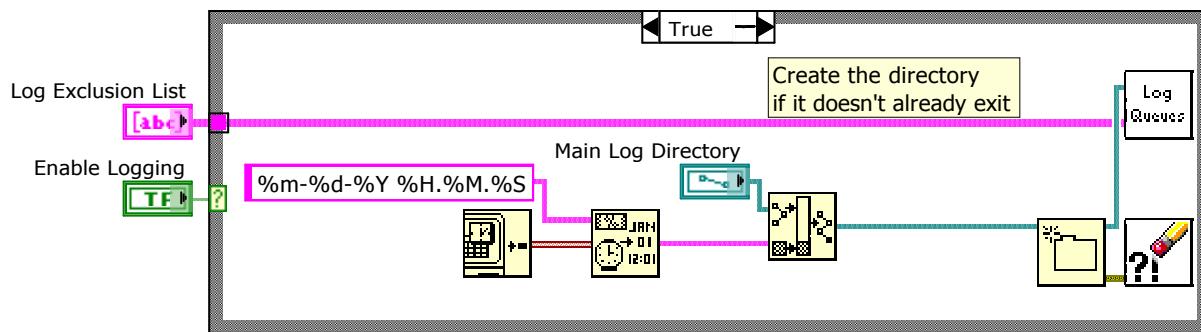
LogMain.vi

Manages all logging and replay backend systems (not the GUI).

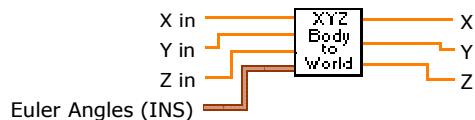
Front Panel



Block Diagram

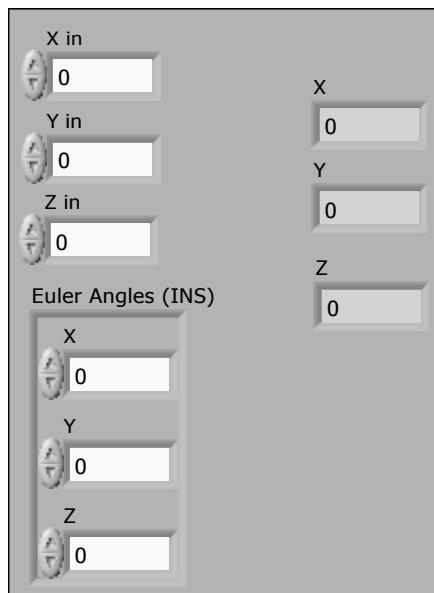


Connector Pane

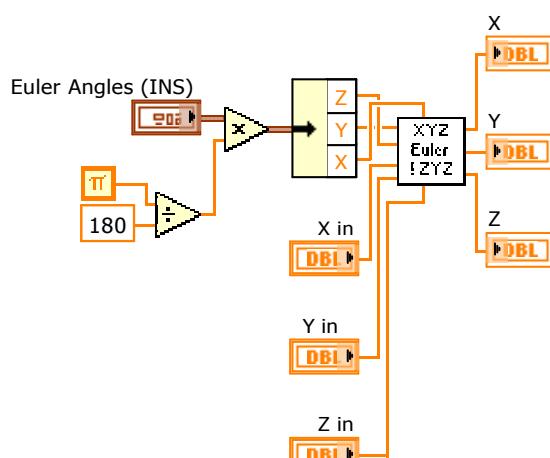
BodyToWorldTransform.vi

Body to World transform based on INS data.

Front Panel



Block Diagram



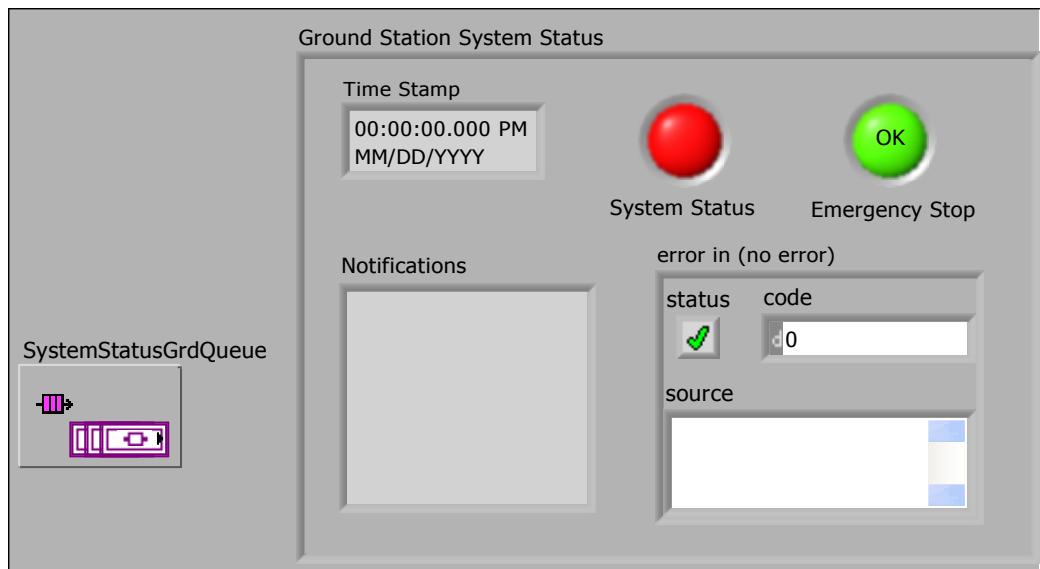
Body to world transform
(provided by INS)

Connector Pane

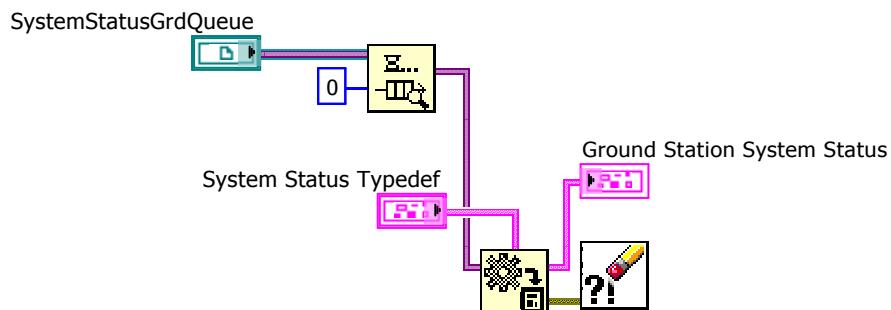
GetGroundStationSystemStatus.vi

GUI function to display groundstation status.

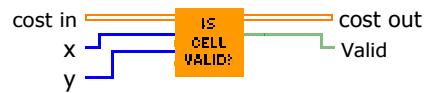
Front Panel



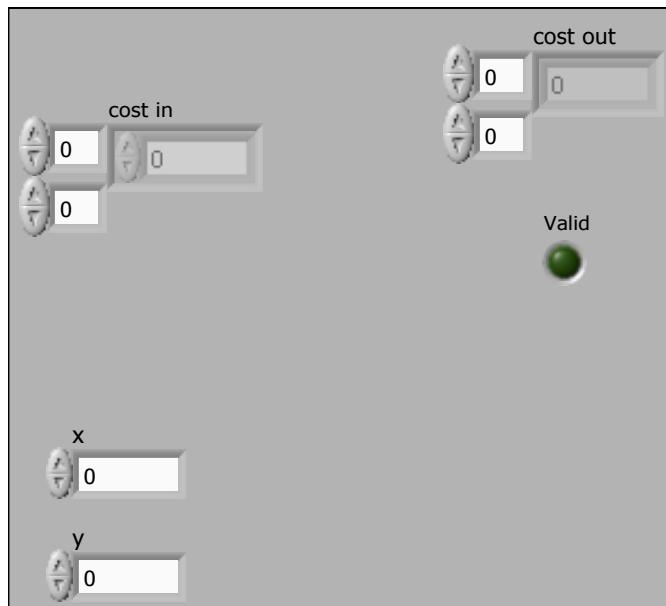
Block Diagram



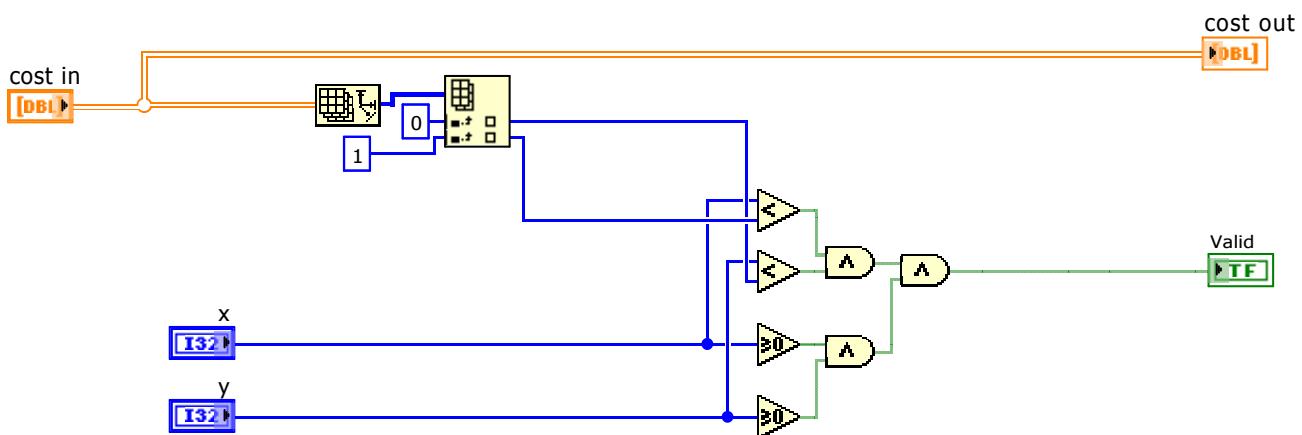
Connector Pane

Is Cell Valid.vi

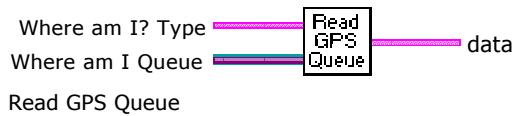
Front Panel



Block Diagram

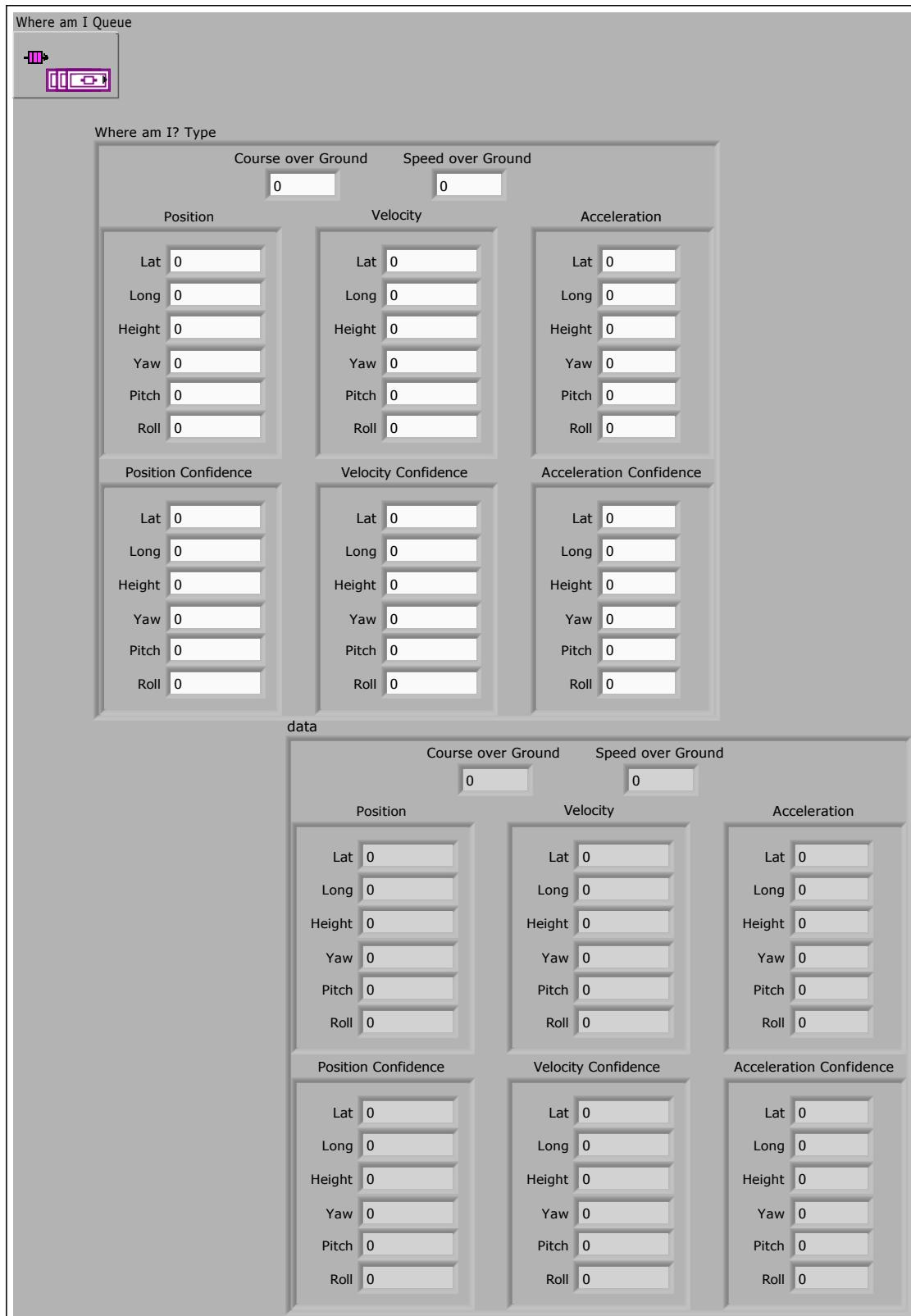


Connector Pane

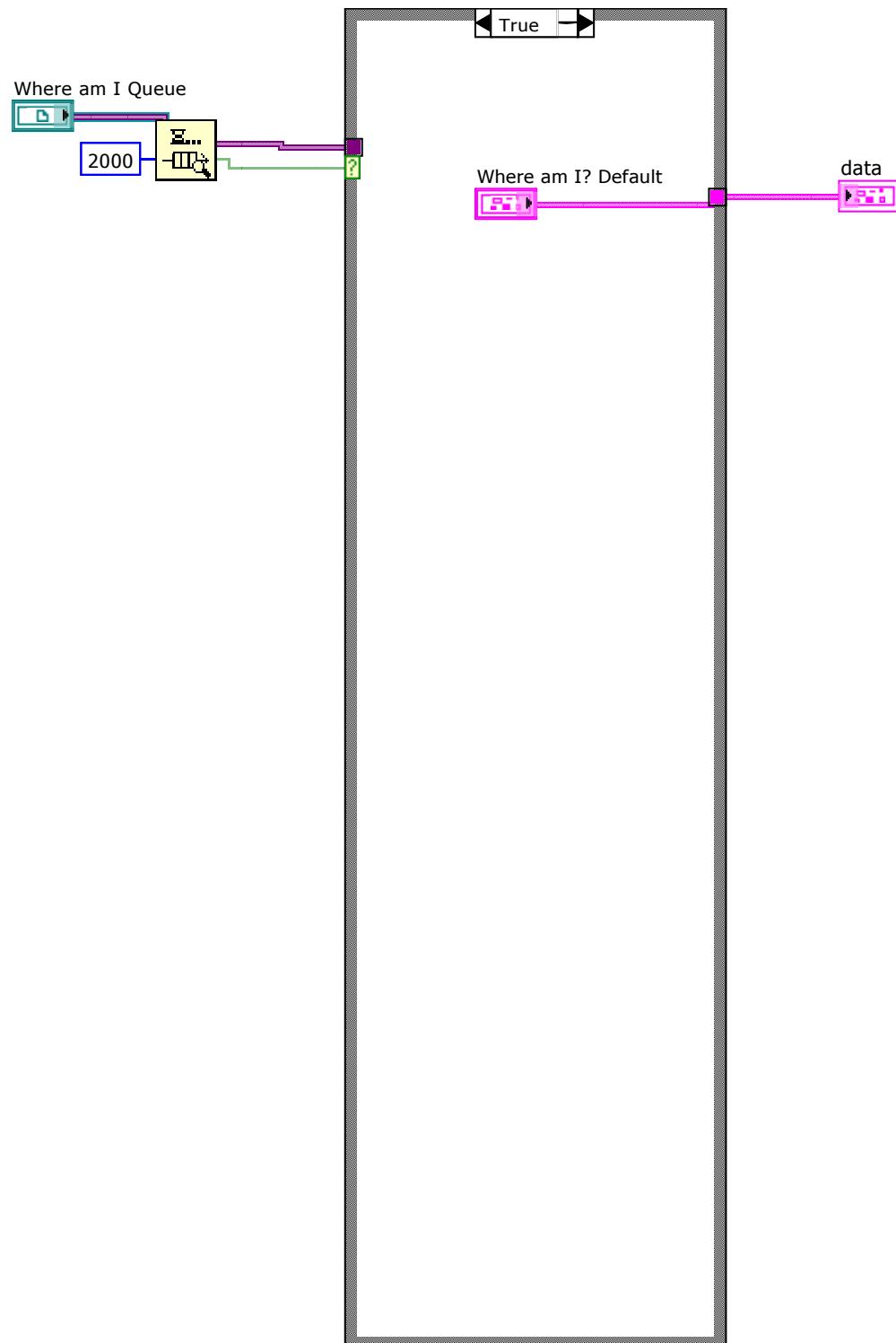
Read GPS Queue.vi

Read GPS Queue

Front Panel



Block Diagram

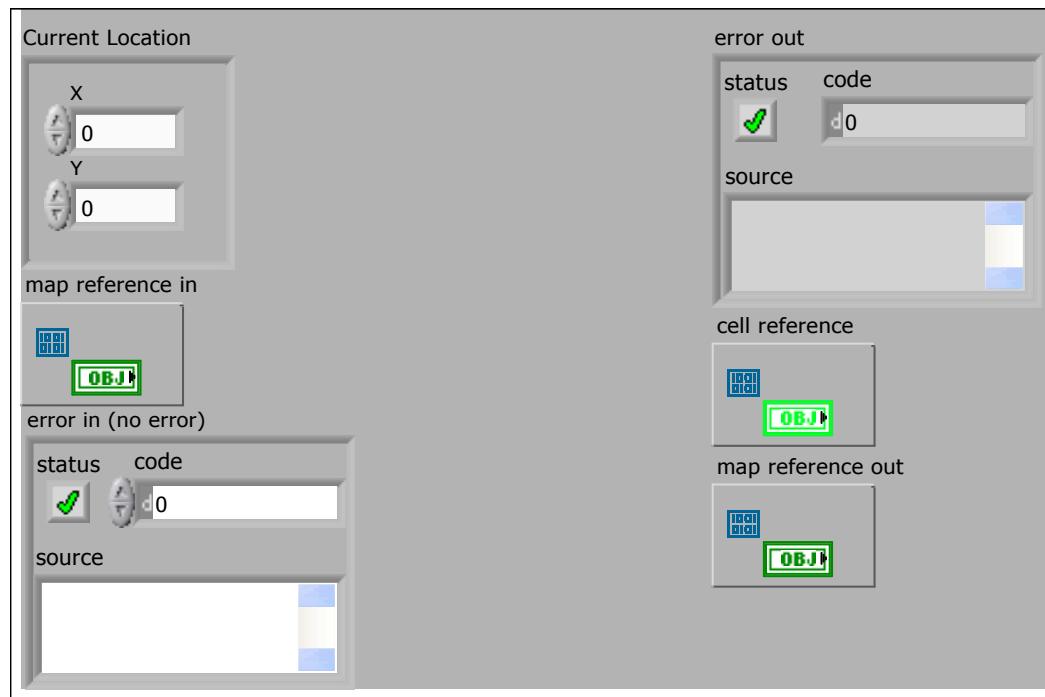


Connector Pane

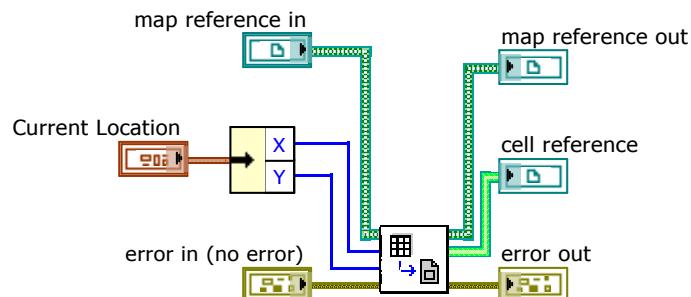
Get Cell Reference.vi

Gets cell reference from a position object

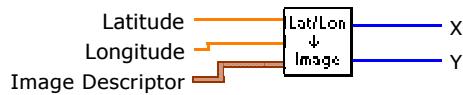
Front Panel



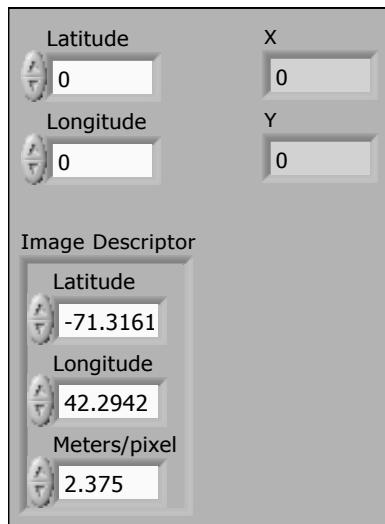
Block Diagram



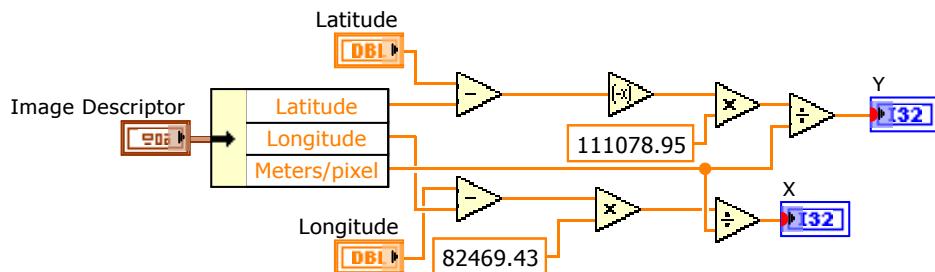
Connector Pane

Lat Lon to Image.vi

Front Panel



Block Diagram



Connector Pane

cRIOSelfTestControl.ctl

Front Panel

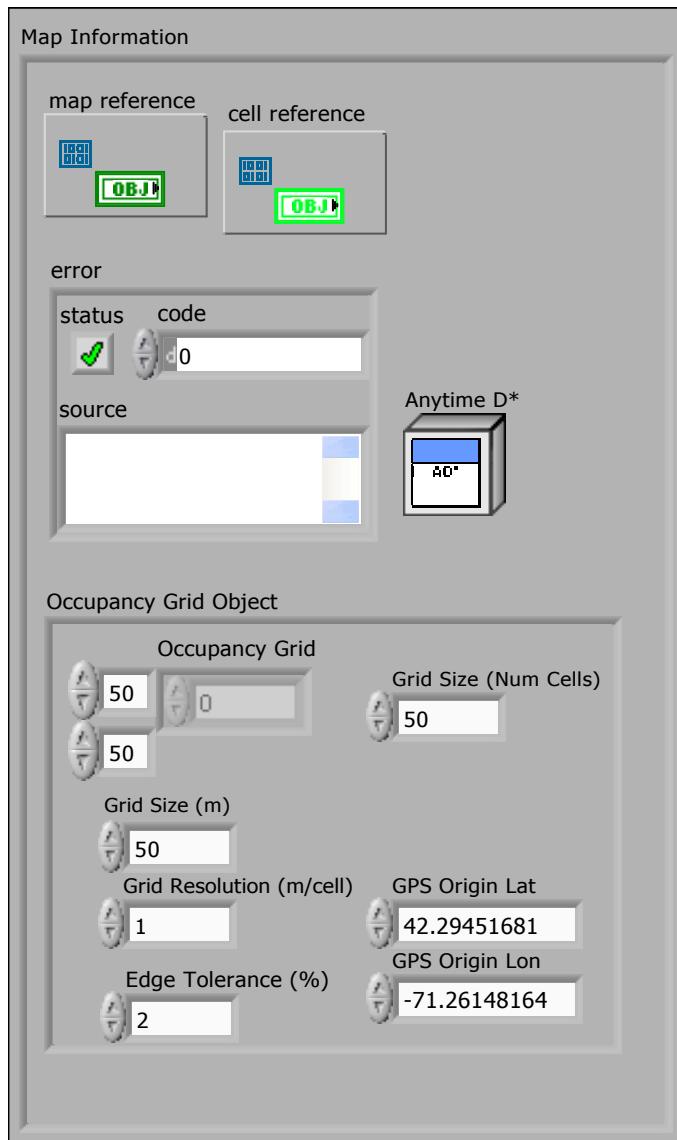


Block Diagram

Connector Pane

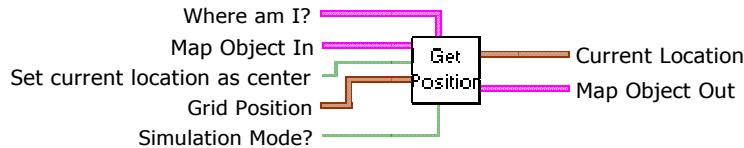
Map Information.ctl

Front Panel



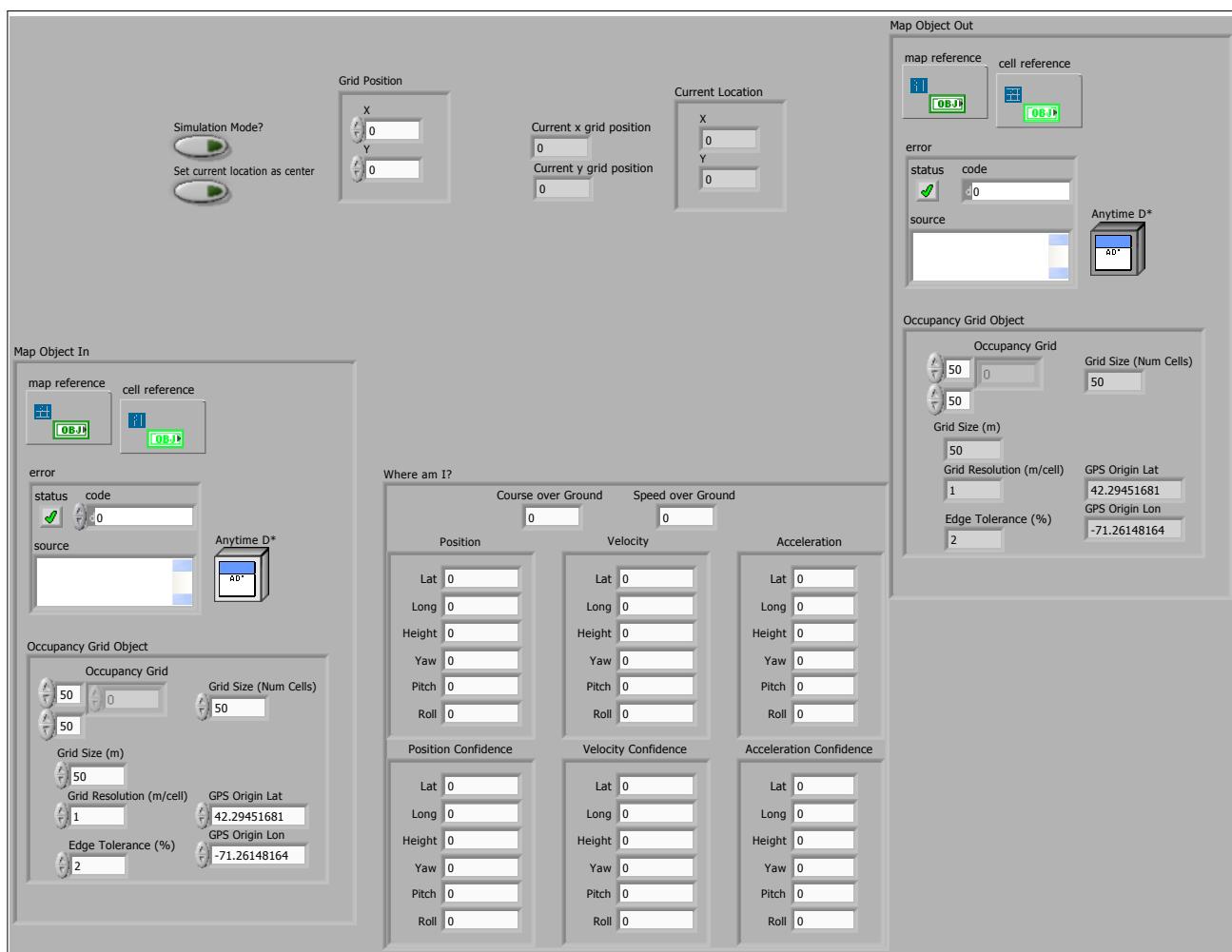
Block Diagram

Connector Pane

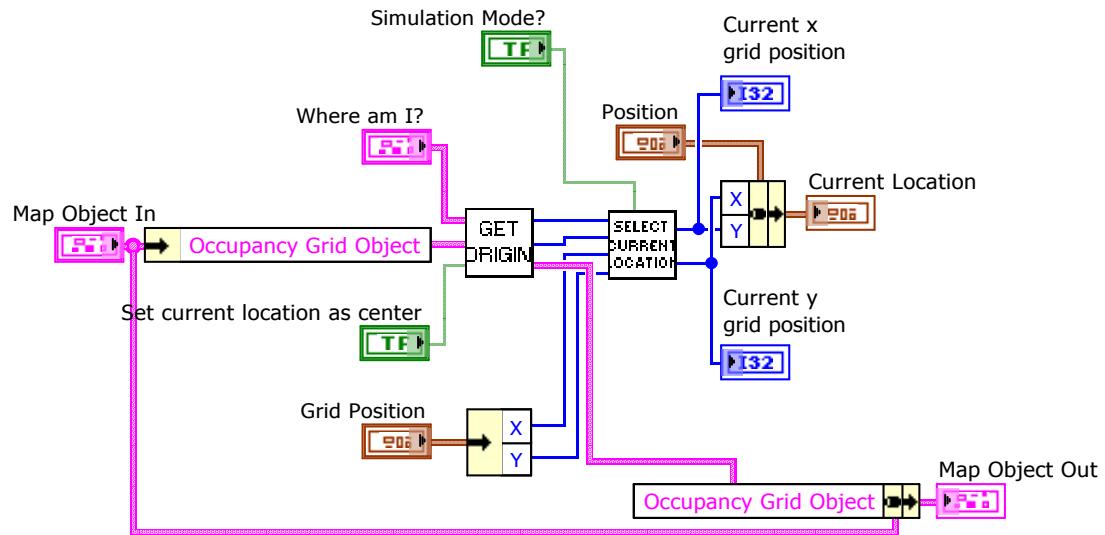
Get Position.vi

Returns the current position on the occupancy grid, in terms of grid indices. Also allows you to switch your origin to the current location.

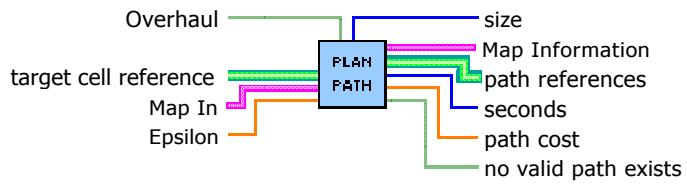
Front Panel



Block Diagram

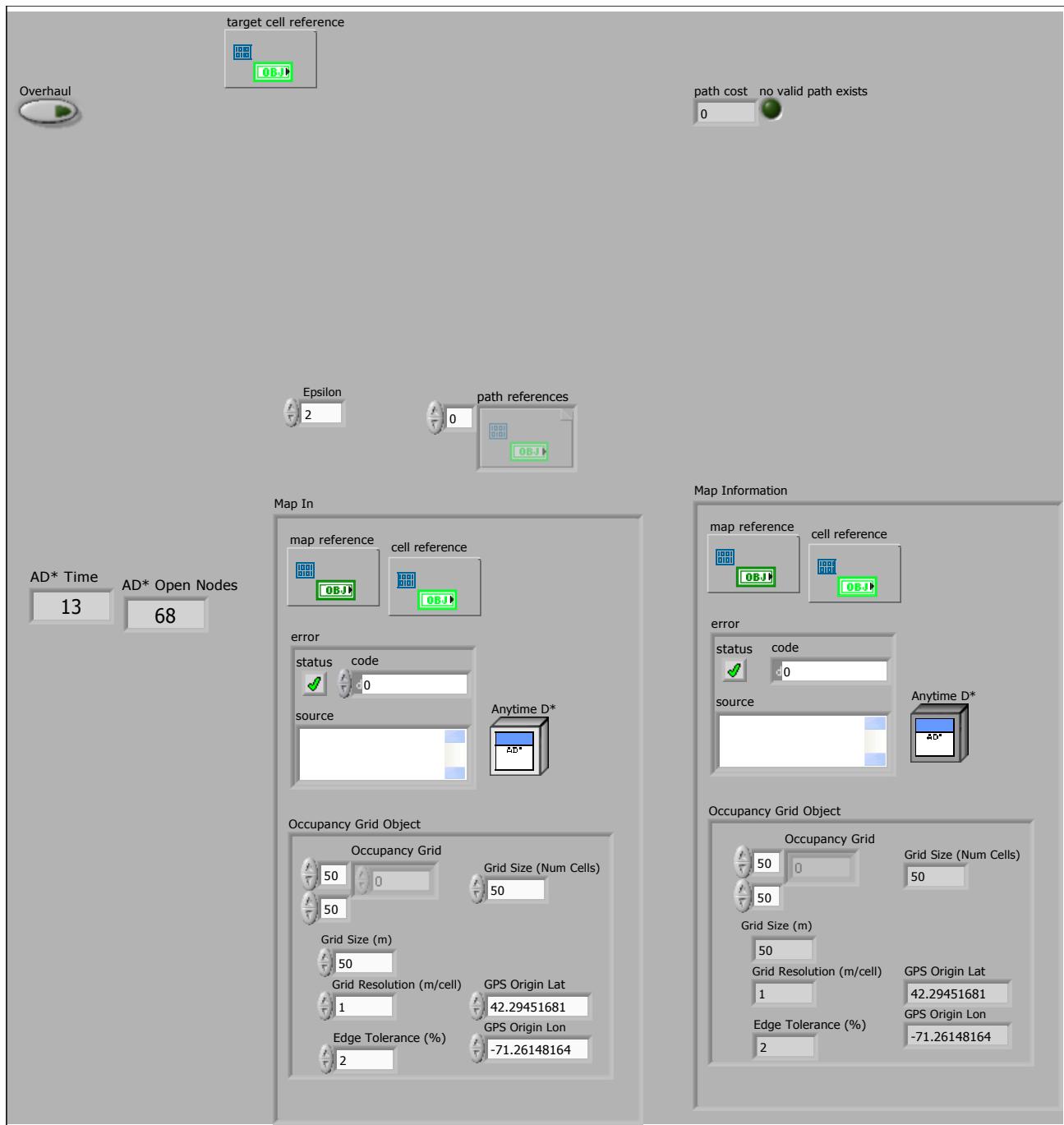


Connector Pane

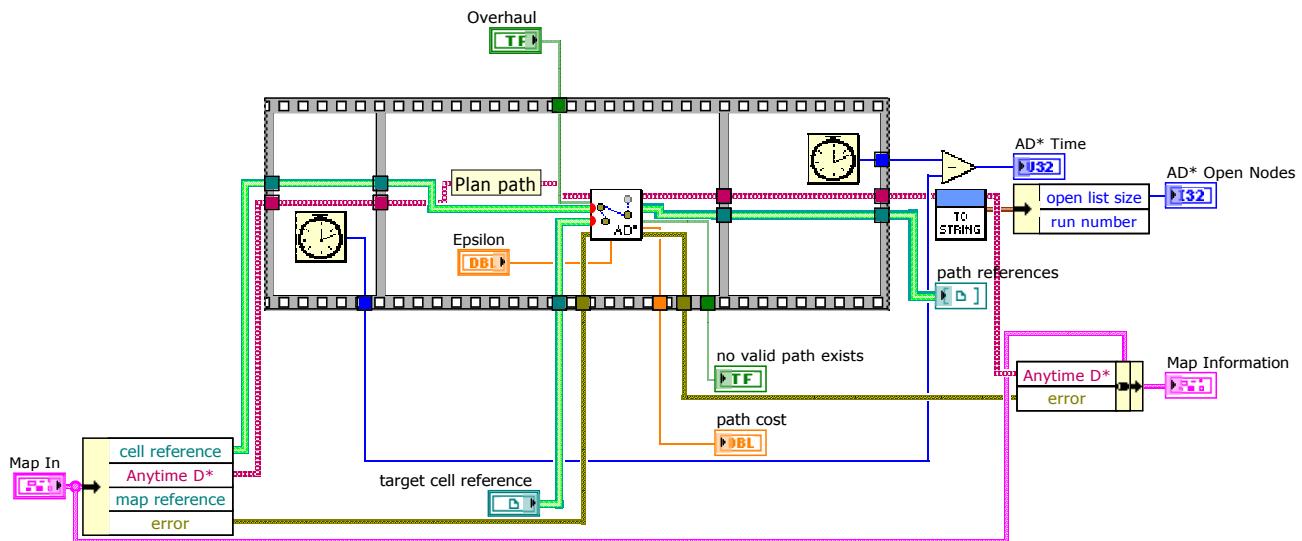
Plan Path.vi

Plans the path with A* algorithm

Front Panel



Block Diagram

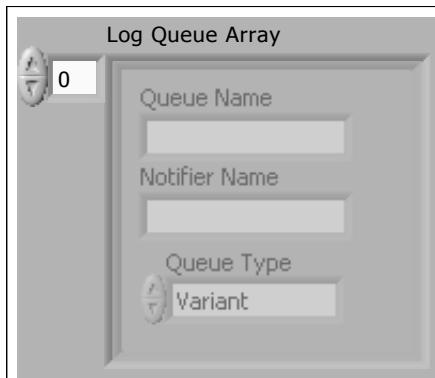


Connector Pane

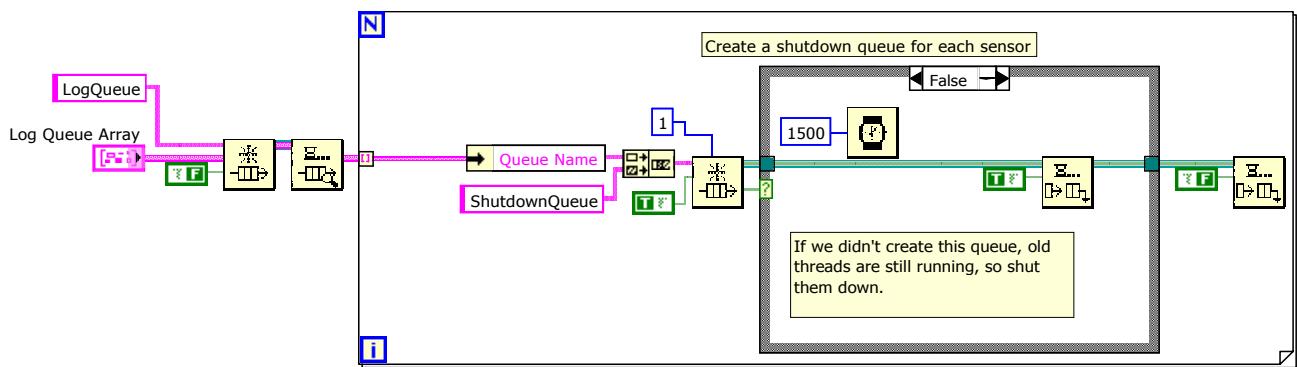
InitShutdownQueue.vi

Initializes non-generic shutdown queues. These queues include anything that has been initialized already with "ShutdownQueue" appended. We initialize these so we can shutdown individual sensors when replaying.

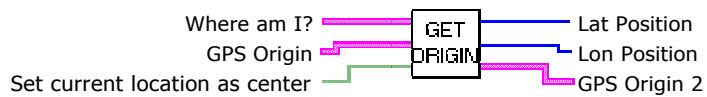
Front Panel



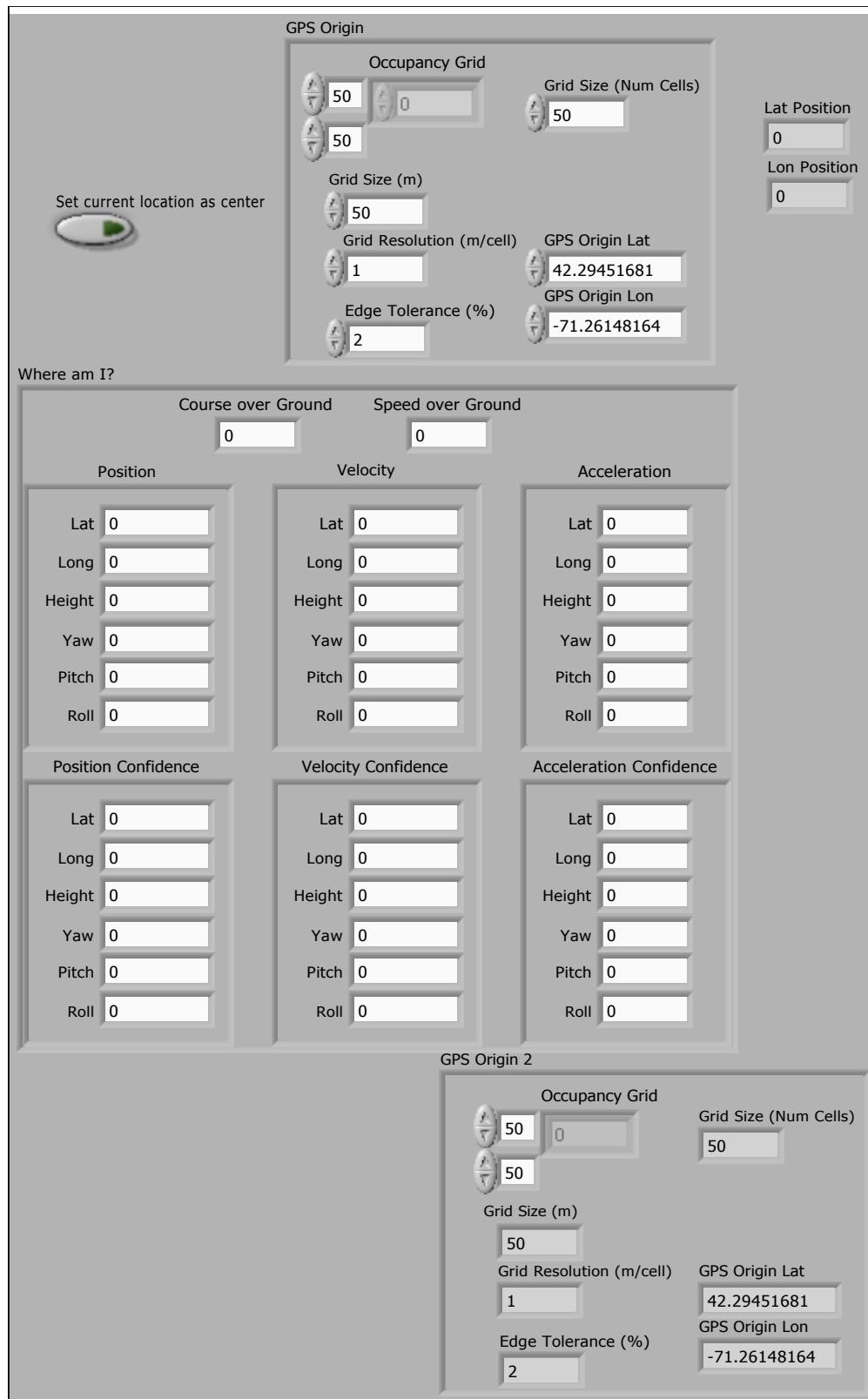
Block Diagram



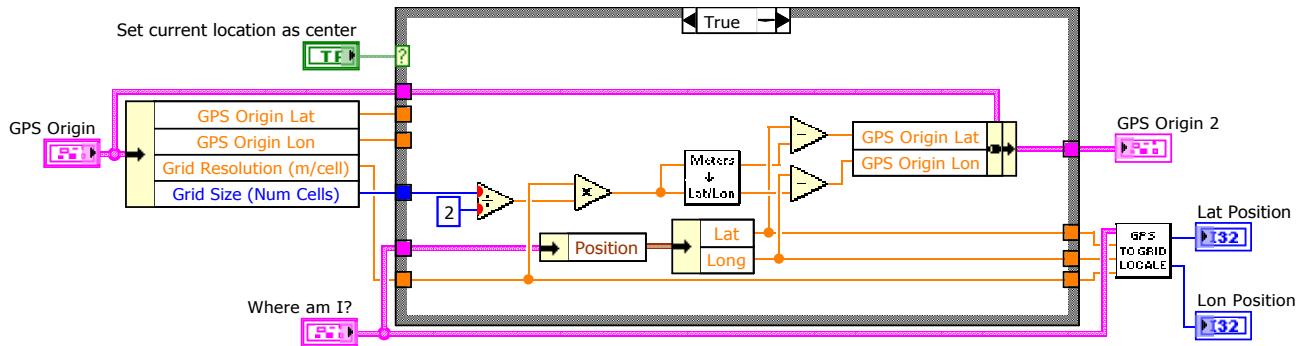
Connector Pane

Select Grid Origin.vi

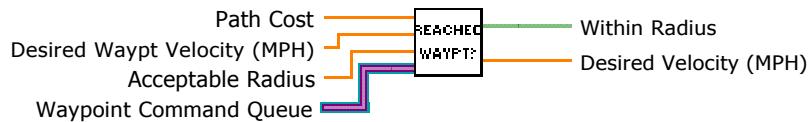
Front Panel



Block Diagram



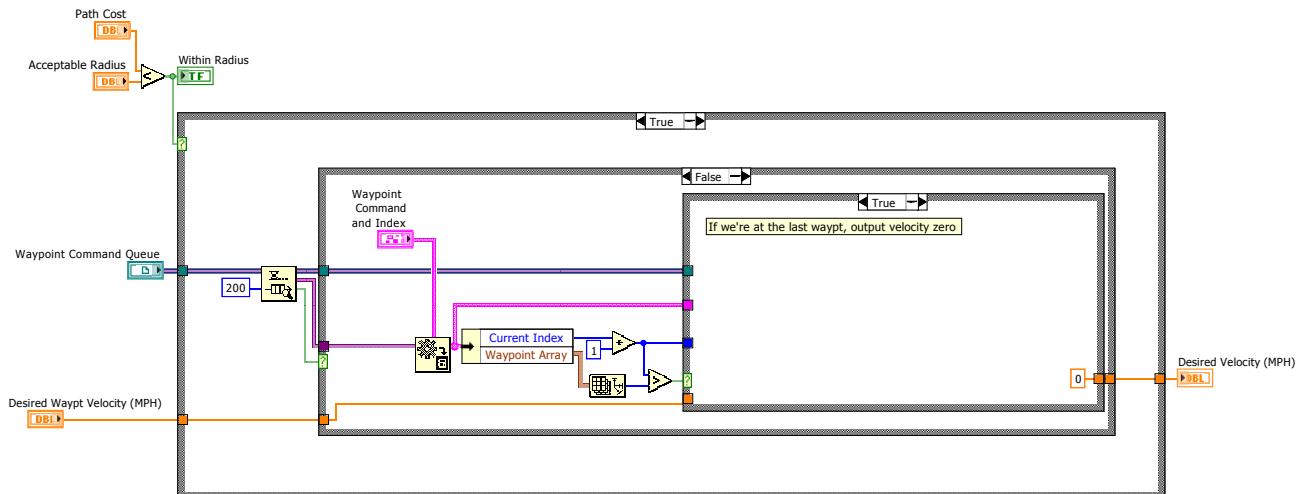
Connector Pane

WaypointReached.vi

Front Panel



Block Diagram



Connector Pane

InitLog.vi

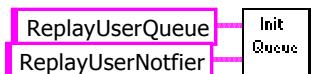
Initializes Log queues.

Front Panel



Block Diagram

This queue commands when to move to the next replay item during replay.

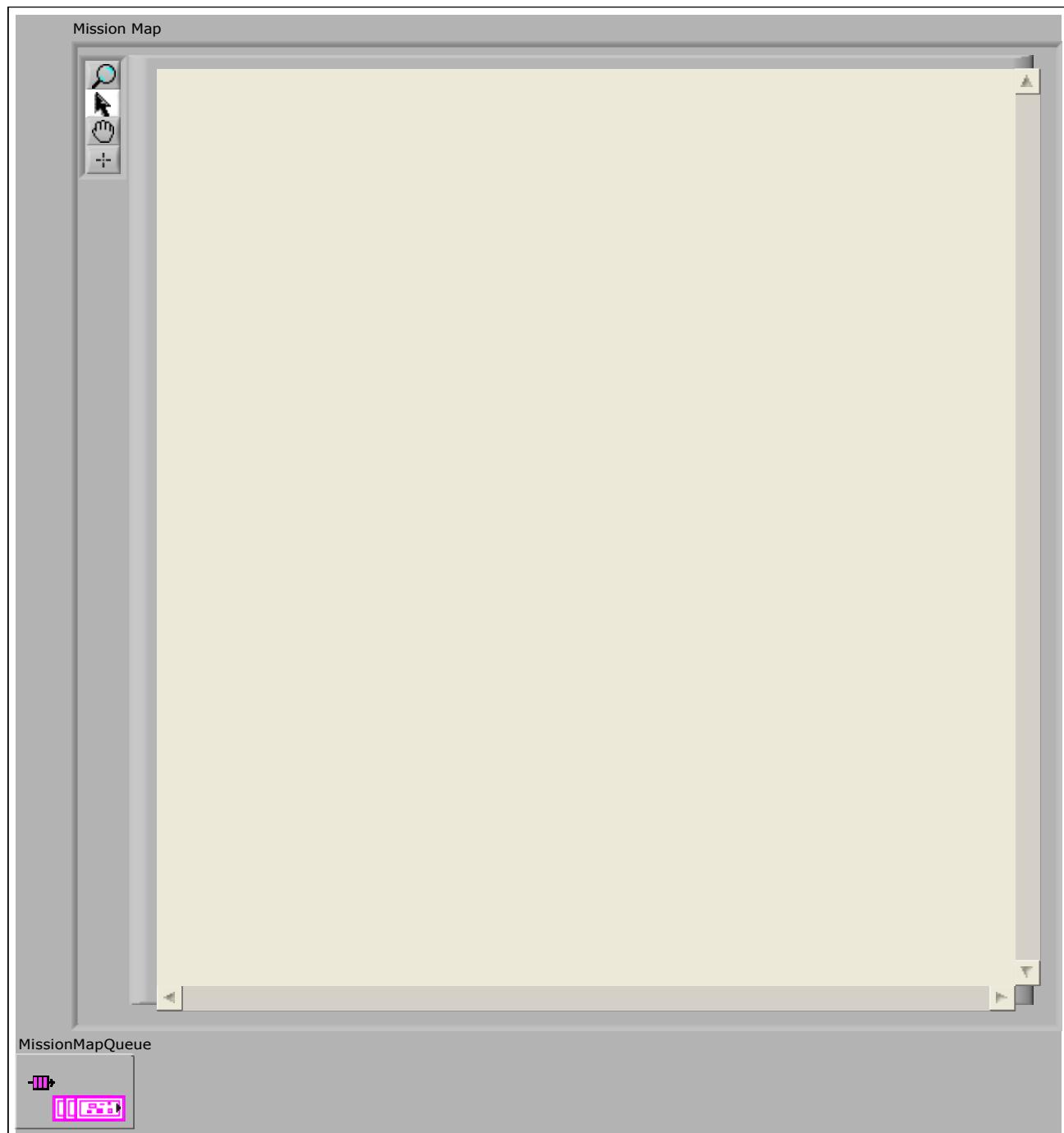


Connector Pane

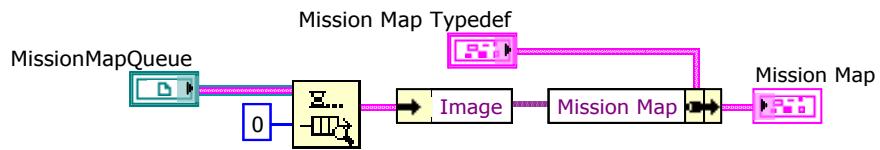
GetMissionMap.vi

GUI function to display the GPS mission map.

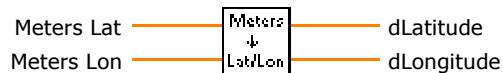
Front Panel



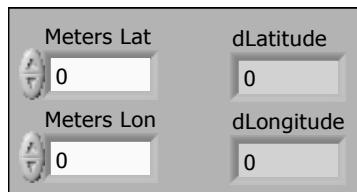
Block Diagram



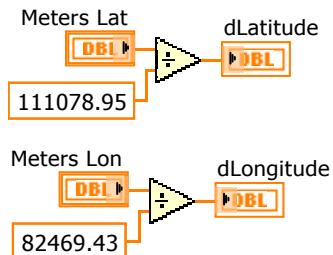
Connector Pane

dMeters to dLat Lon.vi

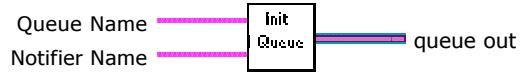
Front Panel



Block Diagram

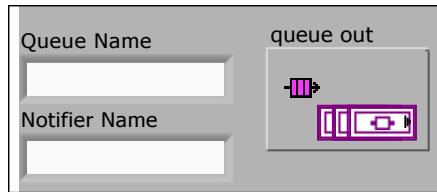


Connector Pane

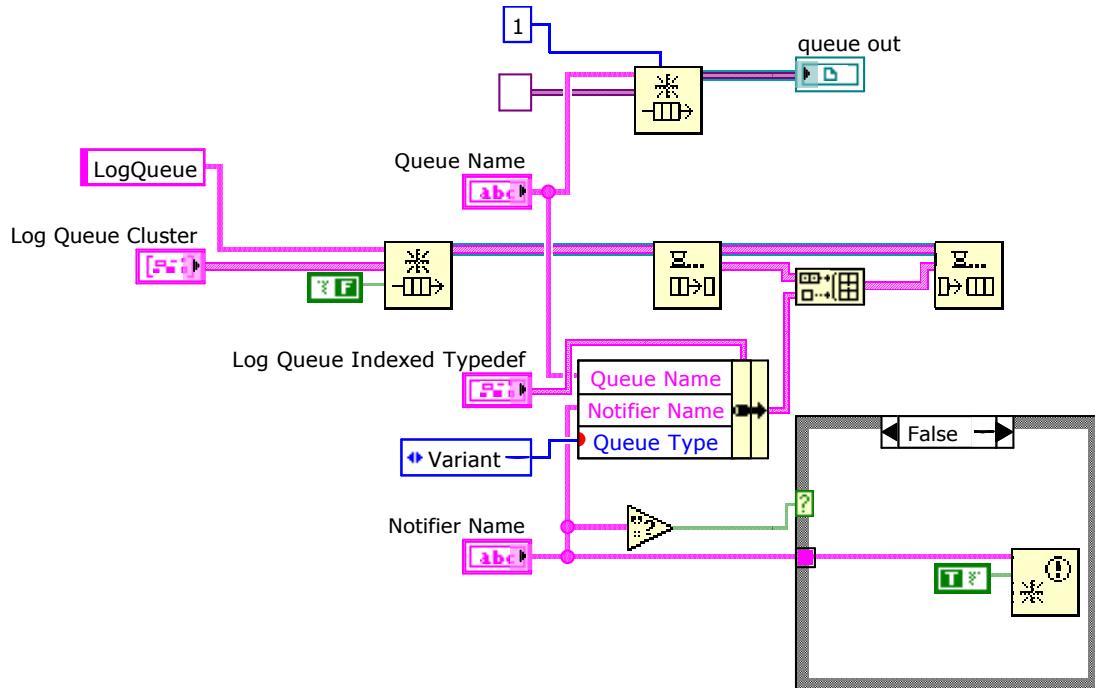
InitQueue.vi

Initializes a new queue (variant type) and enables logging support. Note that this VI is reentrant.

Front Panel

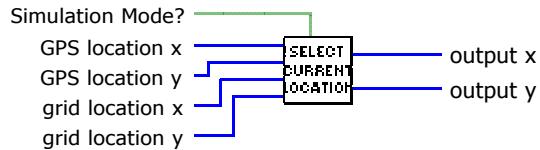


Block Diagram



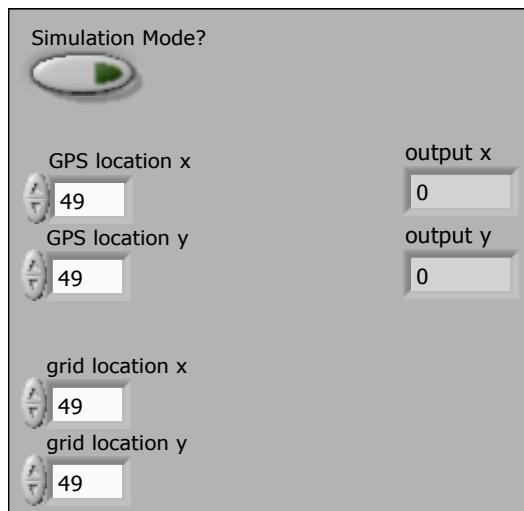
Inits a queue and optionally a notifier. Adds the queue and notifier to the log list.

Connector Pane

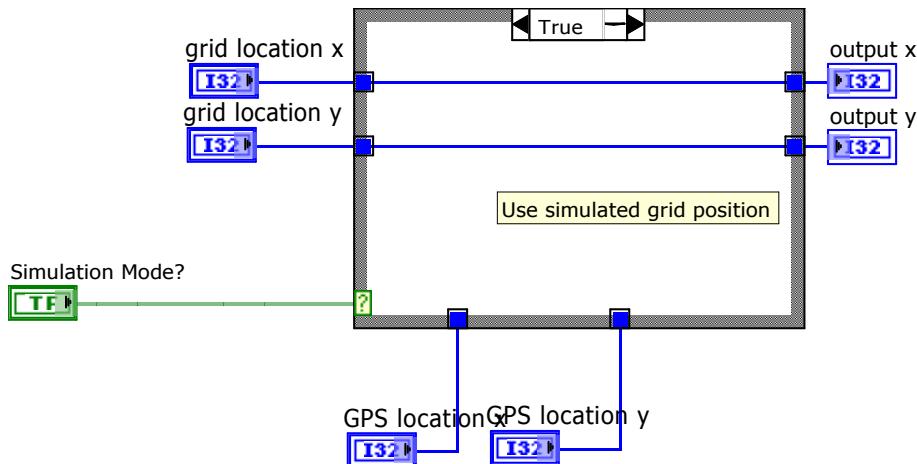
Select Current Location.vi

Selects either GPS or grid location based on whether we are in simulation or waypoint-following mode.

Front Panel



Block Diagram



Connector Pane

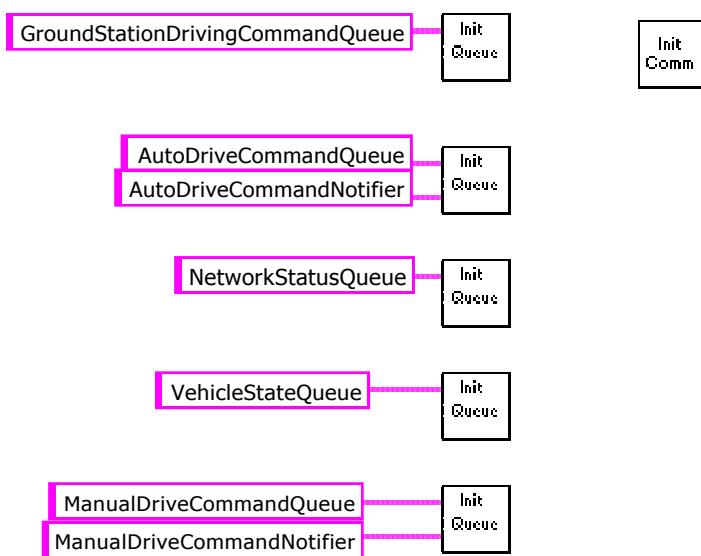
InitAct.vi

Initializes Act queues.

Front Panel



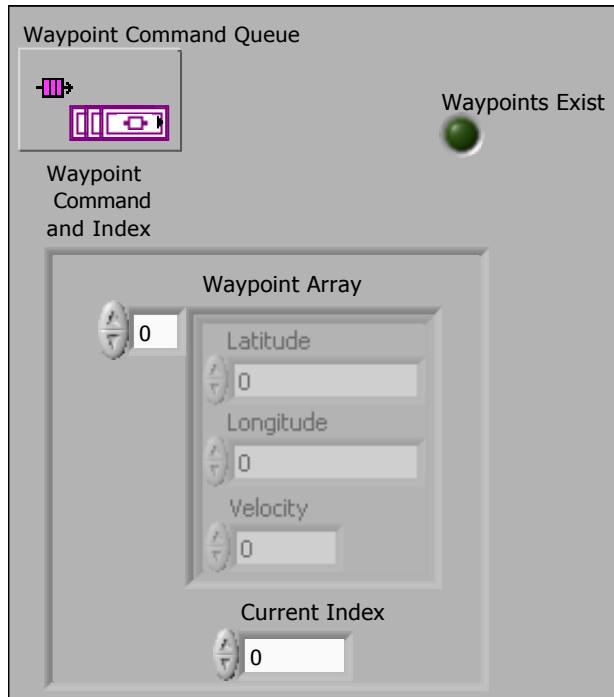
Block Diagram



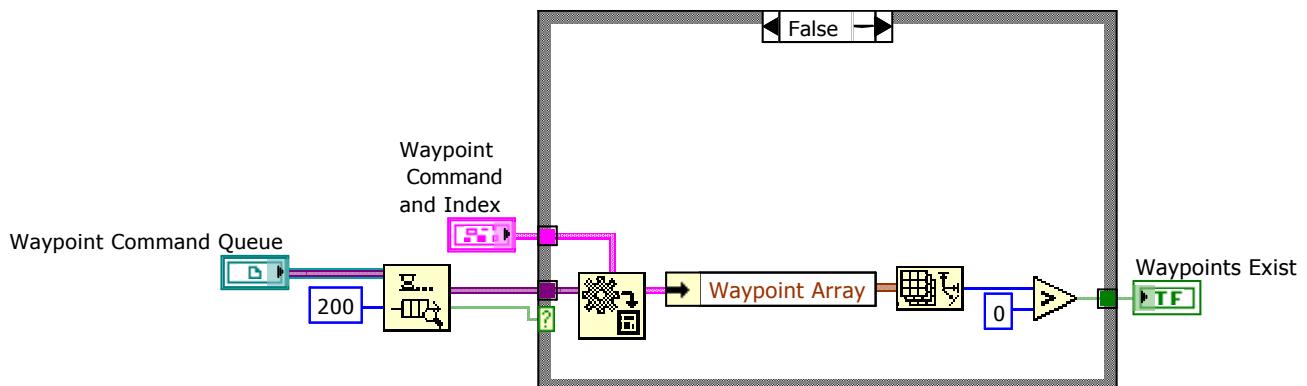
Connector Pane

Do Waypoints Exist.vi

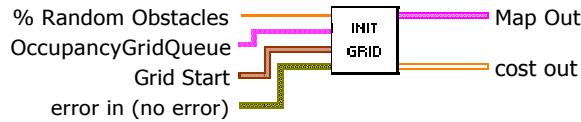
Front Panel



Block Diagram



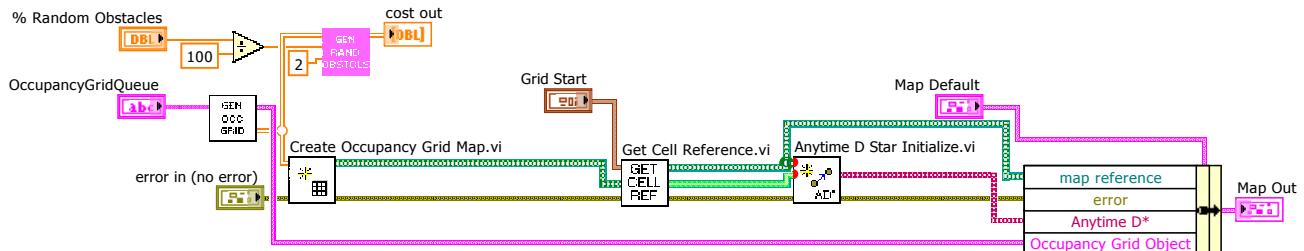
Connector Pane

Init Grid.vi

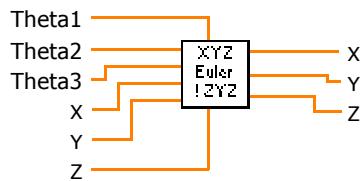
Front Panel



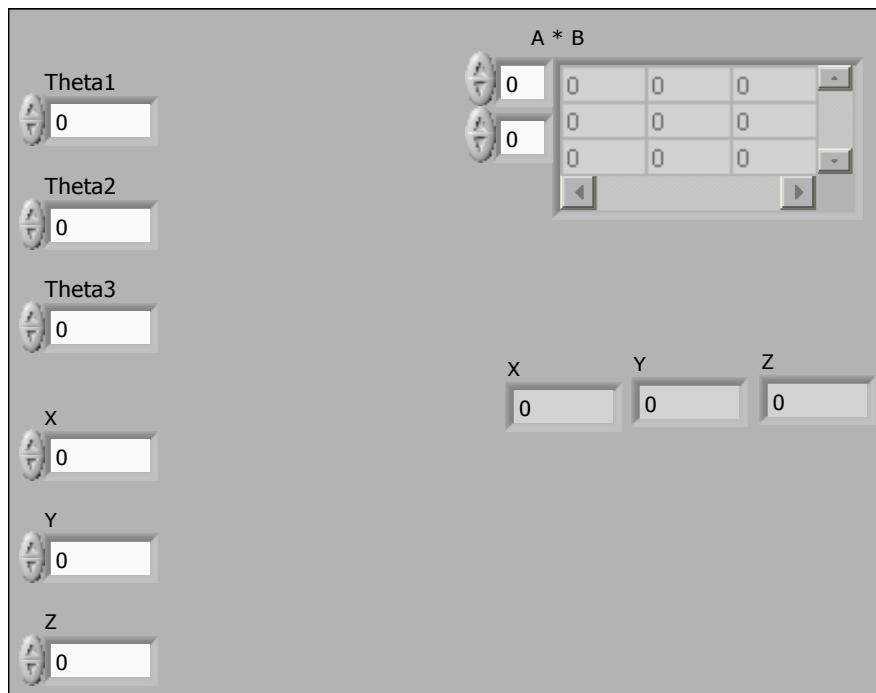
Block Diagram



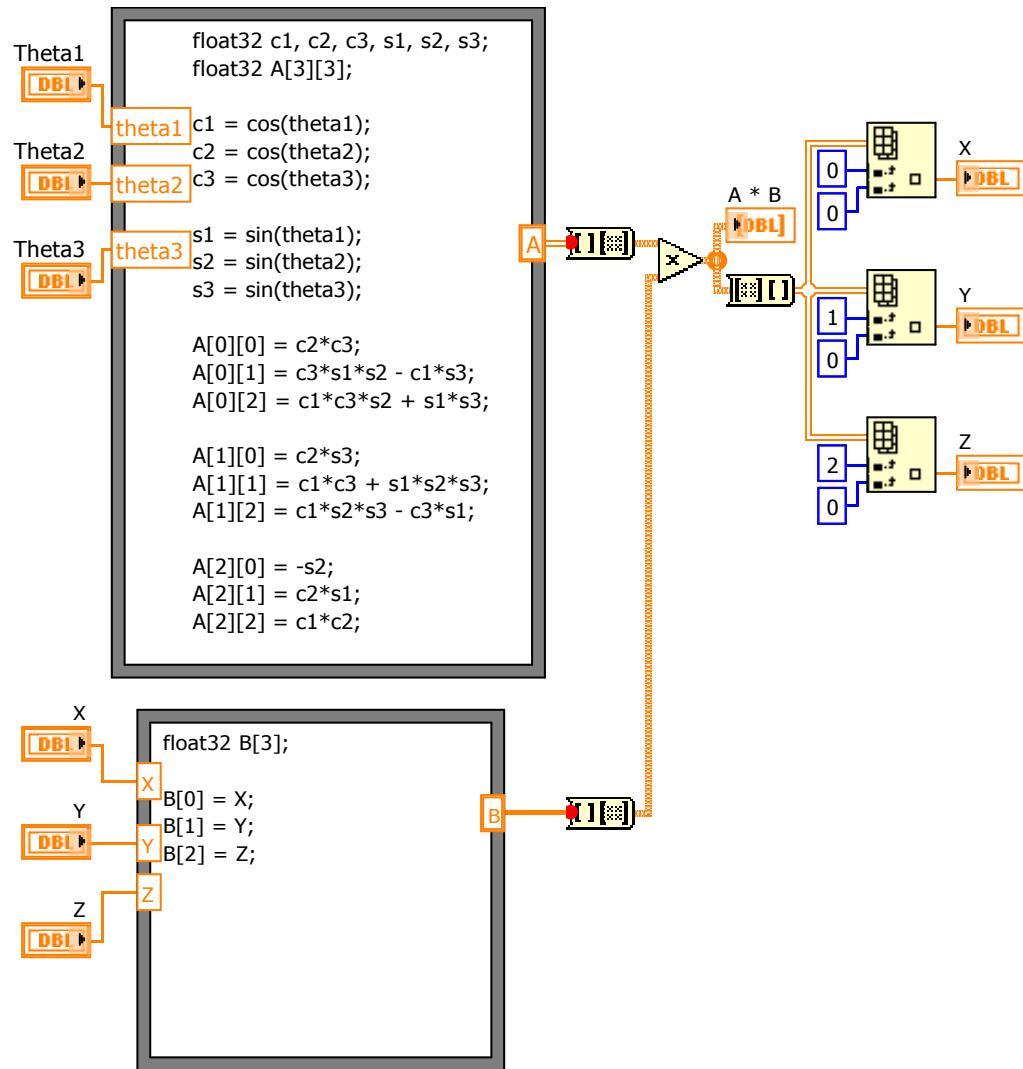
Connector Pane

XYZEuler.vi

Front Panel



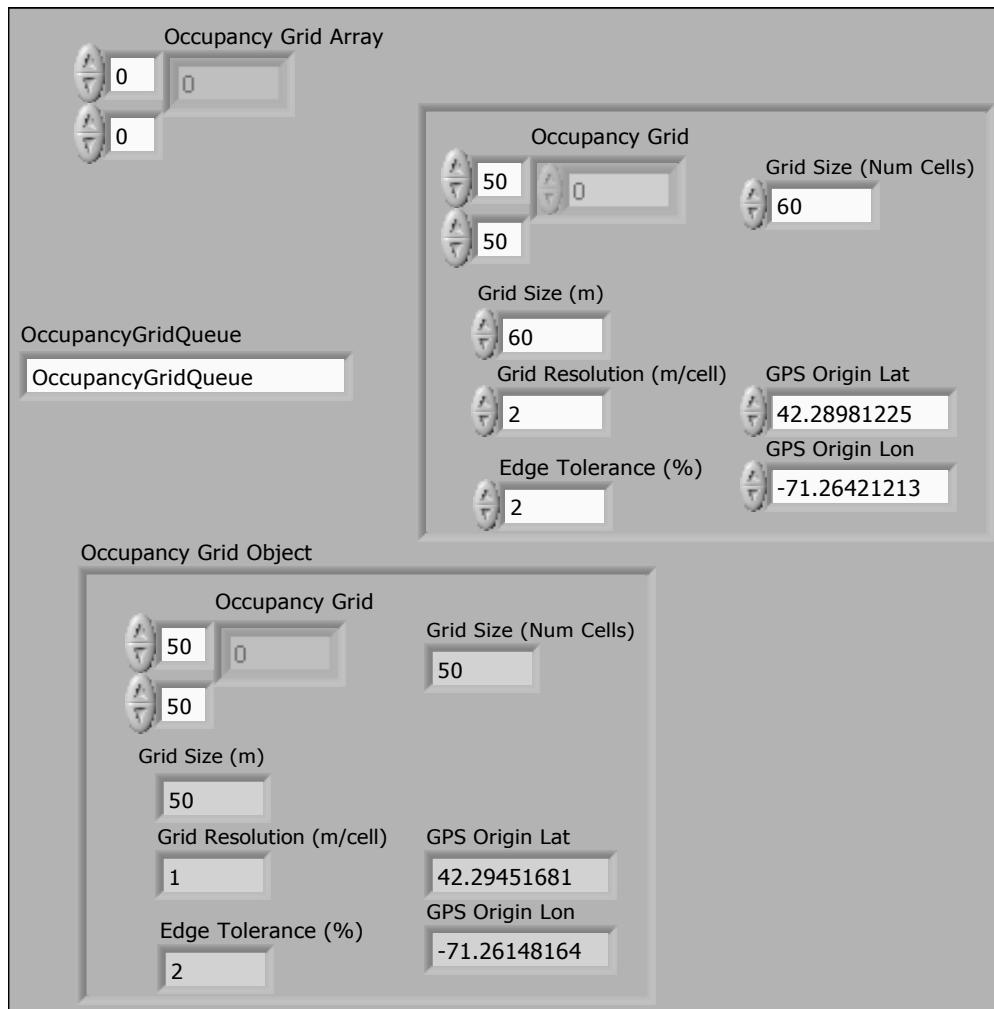
Block Diagram



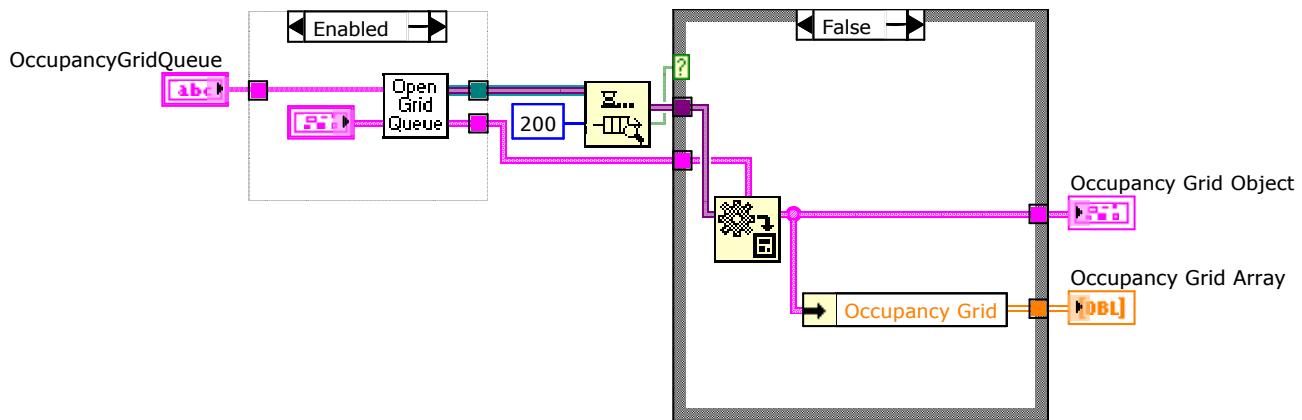
Connector Pane

Generate Occupancy Grid.vi

Front Panel



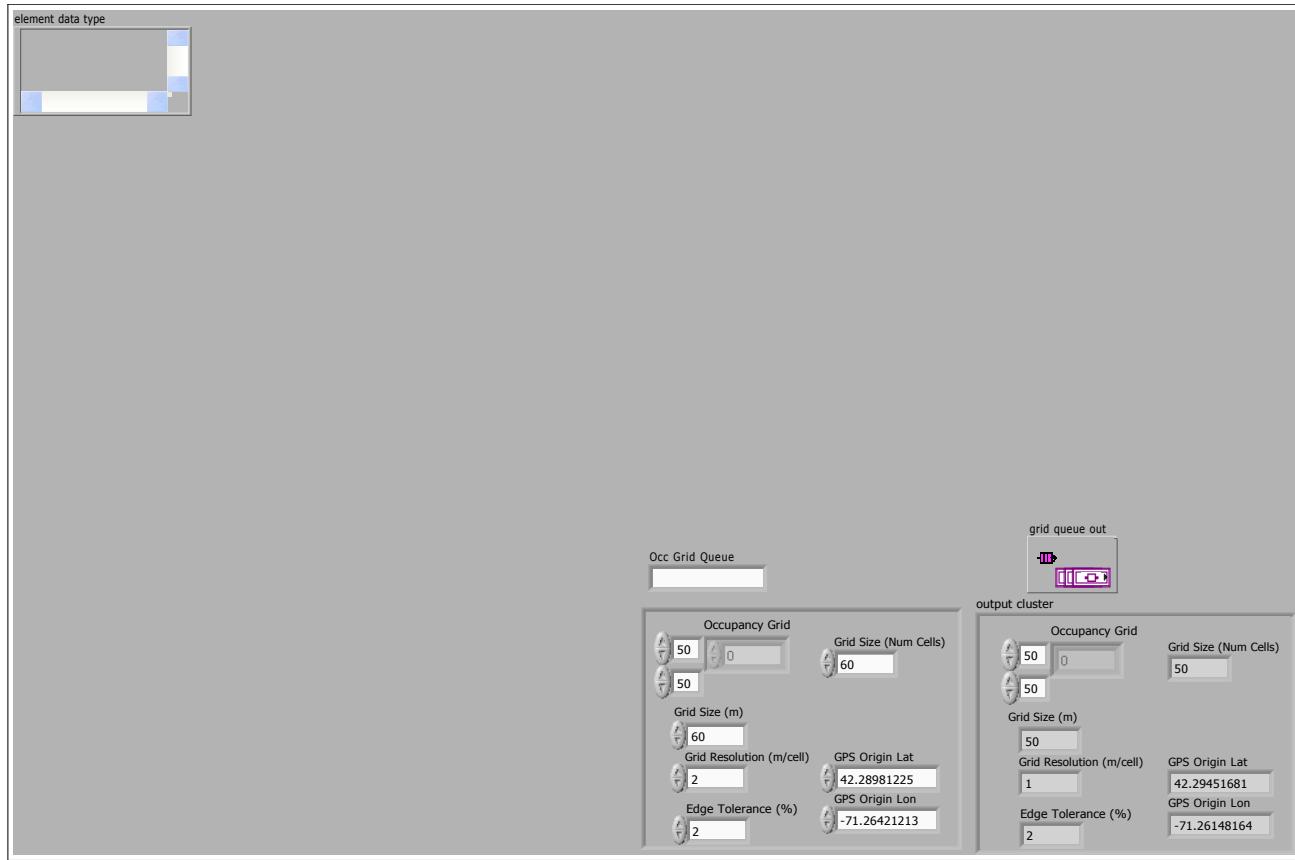
Block Diagram



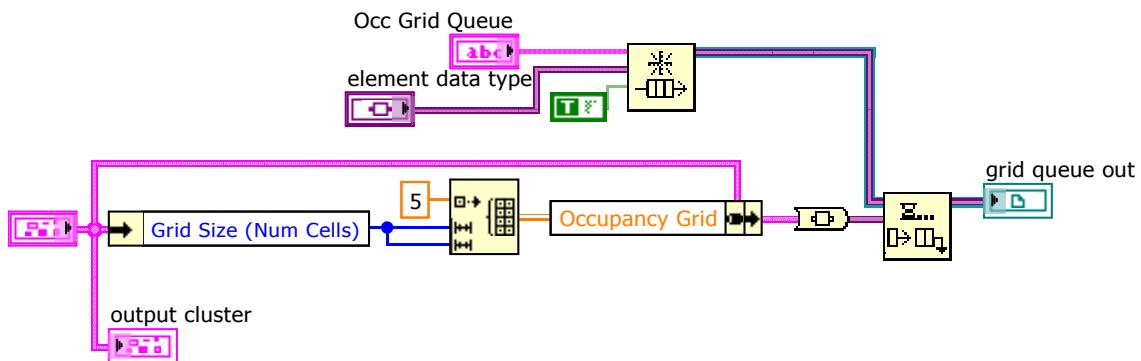
Connector Pane

Read Occupancy Grid Queue.vi

Front Panel



Block Diagram



Connector Pane

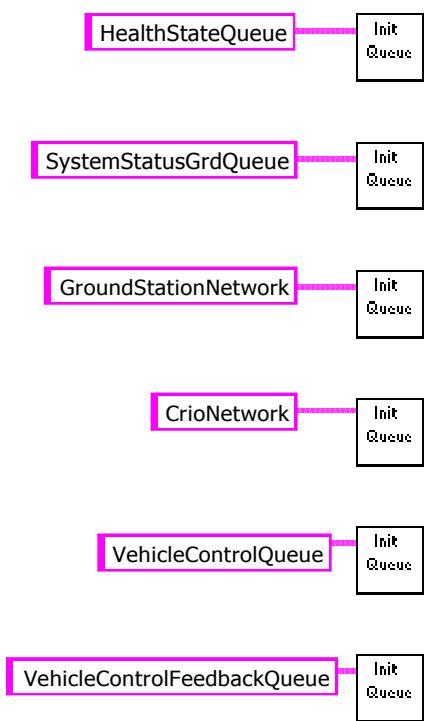
InitComm.vi

Initialize network communication queues.

Front Panel



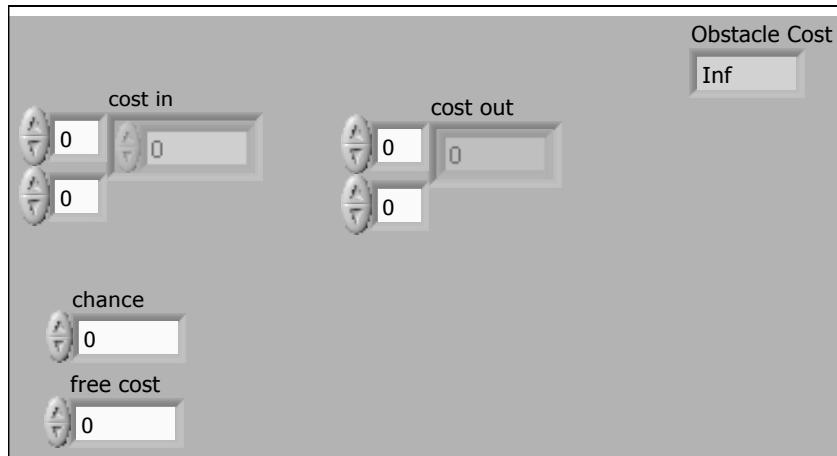
Block Diagram



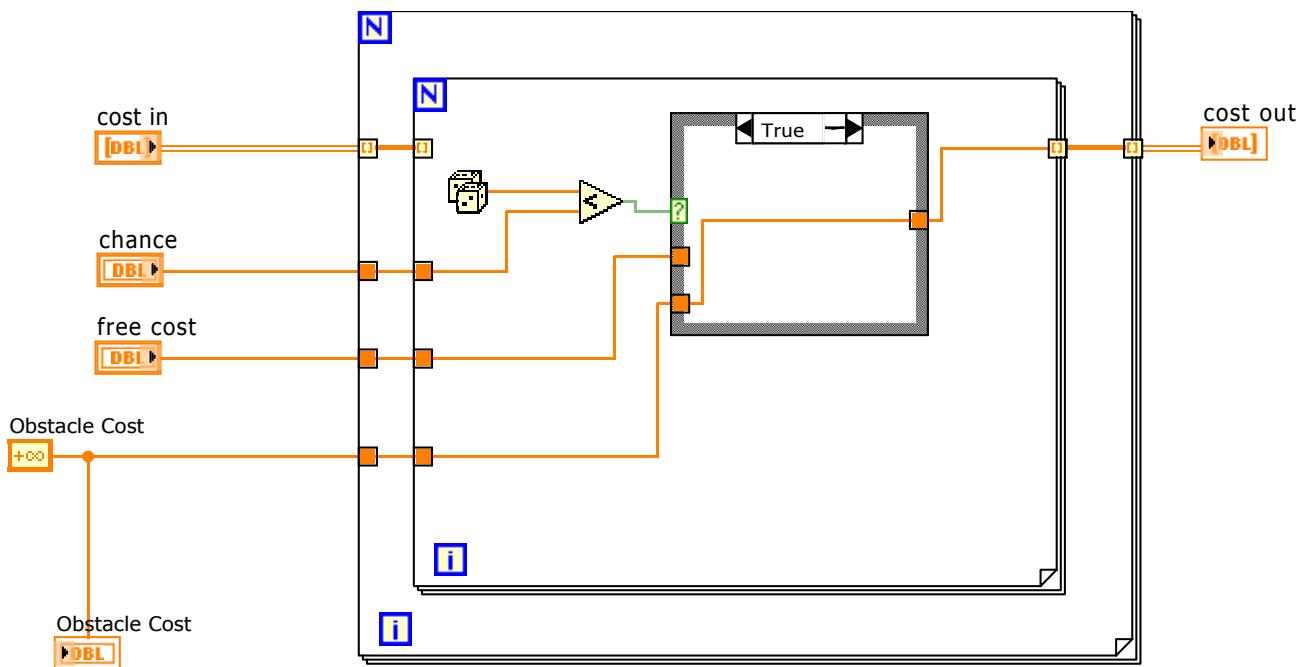
Connector Pane

Generate Random Obstacles.vi

Front Panel



Block Diagram



Connector Pane

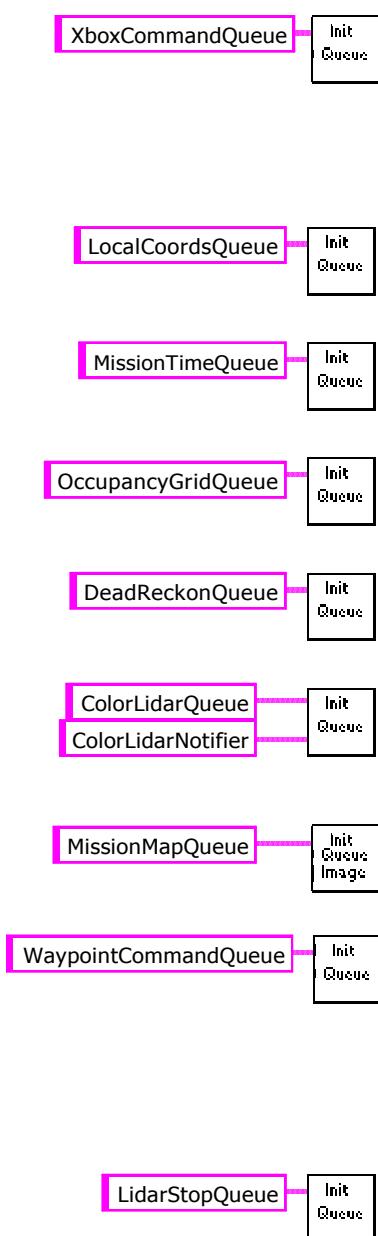
InitThink.vi

Initializes Think queues.

Front Panel



Block Diagram



Connector Pane

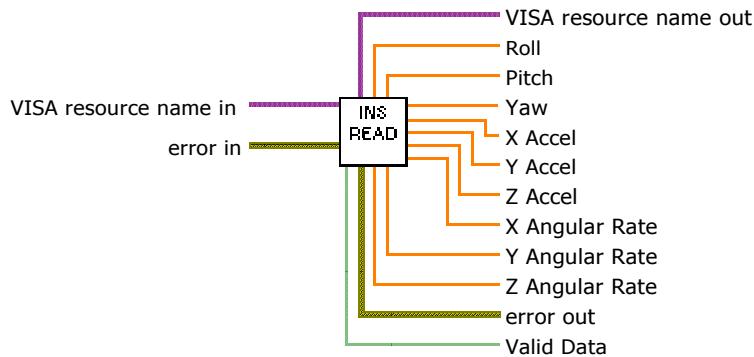
MotorPositionOutputs.ctl

Front Panel



Block Diagram

Connector Pane

INS Read.vi

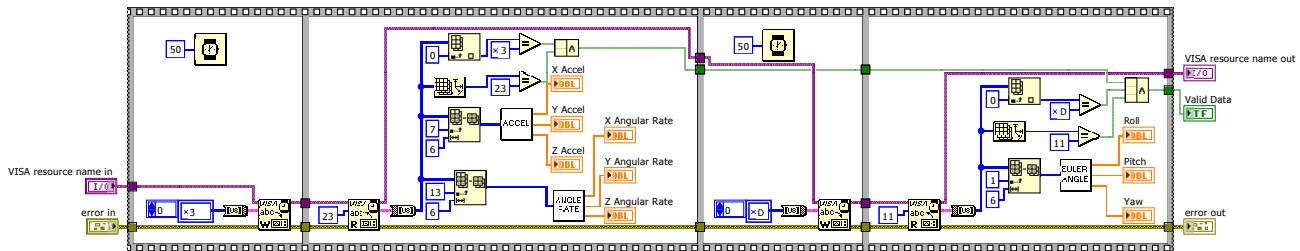
Reads the instantaneous Euler Angles and Vectors from the INS COM port.

This uses polling, so there is a certain delay between the request being sent, and the INS sending back a message. Therefore this VI takes a non-trivial amount of time to execute.

Front Panel



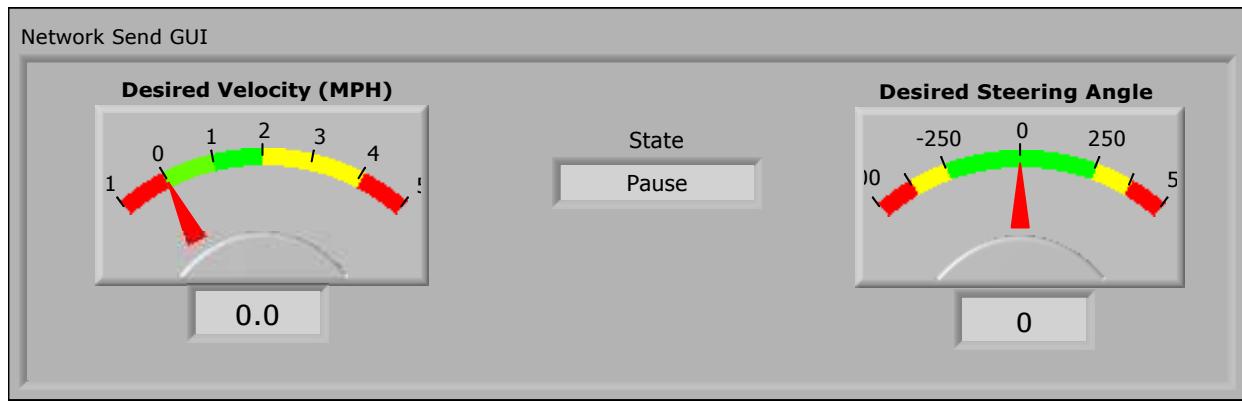
Block Diagram



Connector Pane

NetworkSendGUIControl.ctl

Front Panel

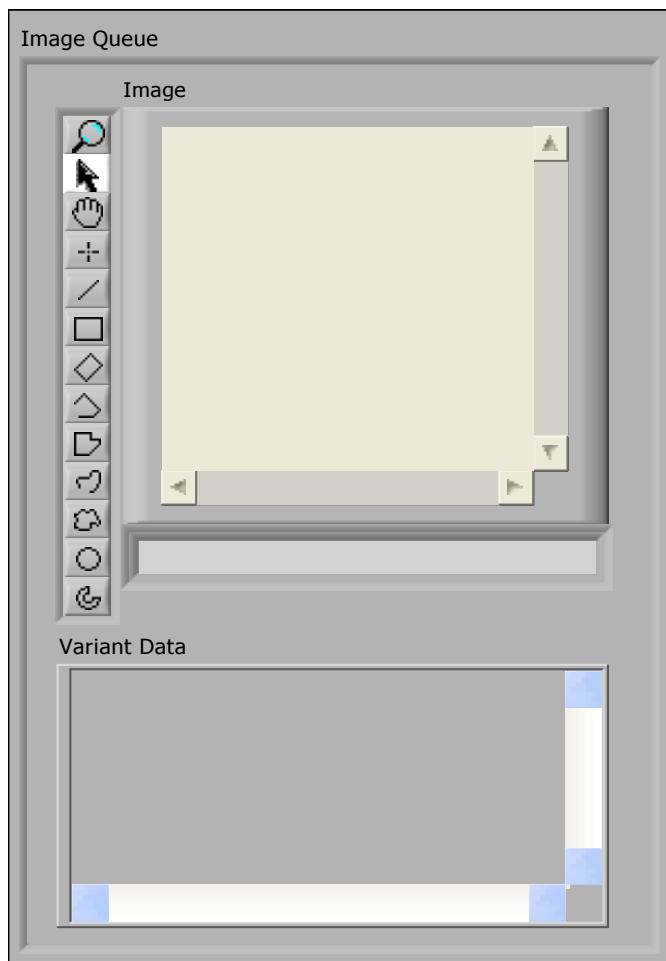


Block Diagram

Connector Pane

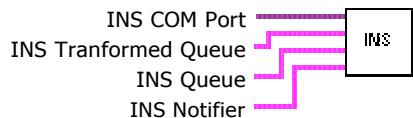
ImageQueue.ctl

Front Panel



Block Diagram

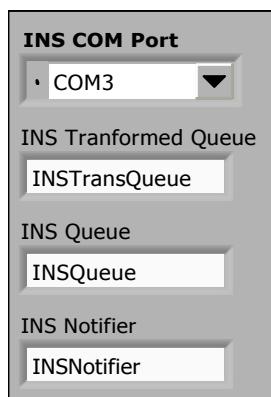
Connector Pane

INSMain.vi

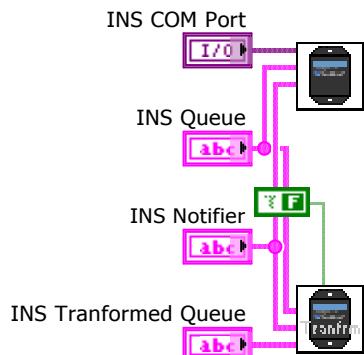
Runs the INS drivers to process data from an INS at the specified COM Port.

Use INS Transformed Queue for use with the Where Am I block, and the INS Queue for use with applications that require raw INS data (such as the Dead Reckoning block).

Front Panel



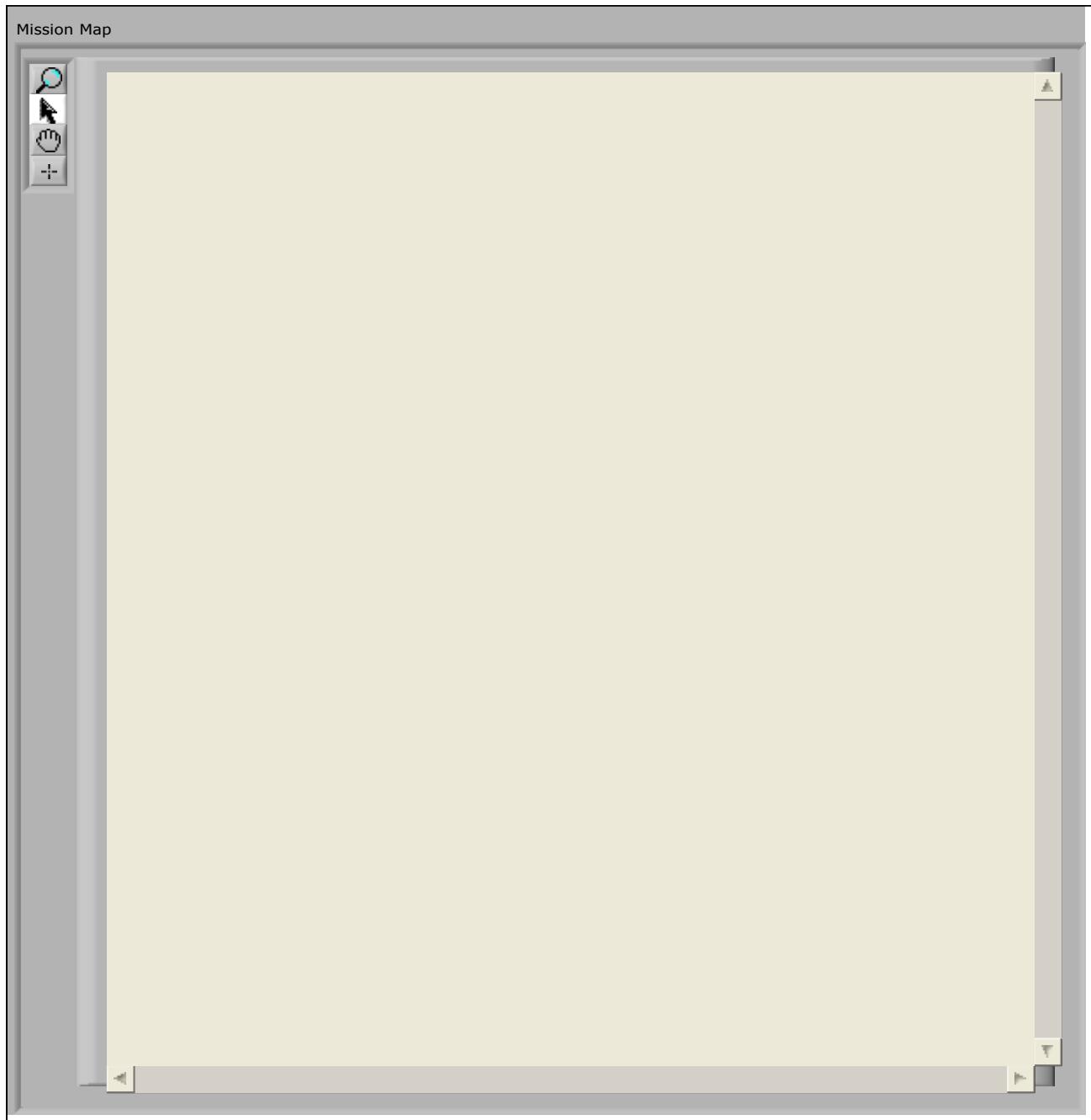
Block Diagram



Connector Pane

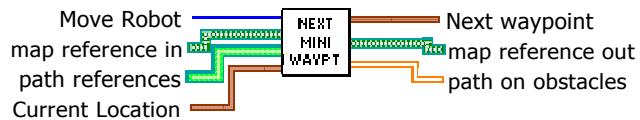
MissionMapControl.ctl

Front Panel

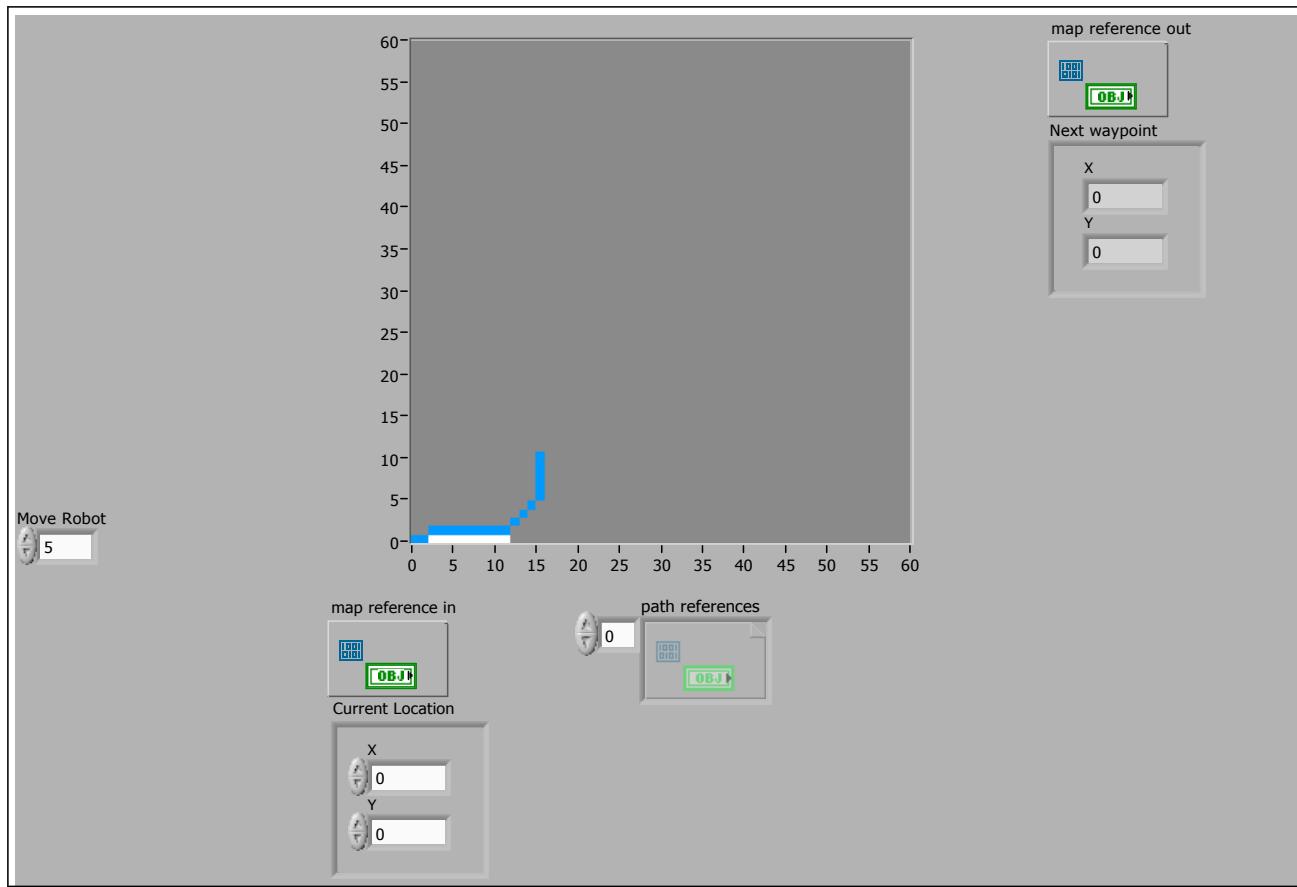


Block Diagram

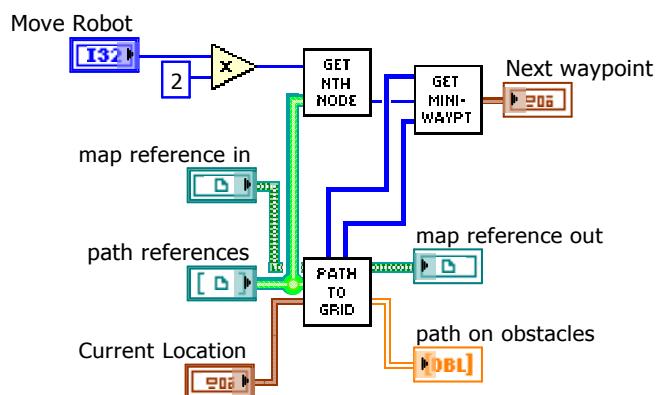
Connector Pane

Get Next Mini Waypoint.vi

Front Panel



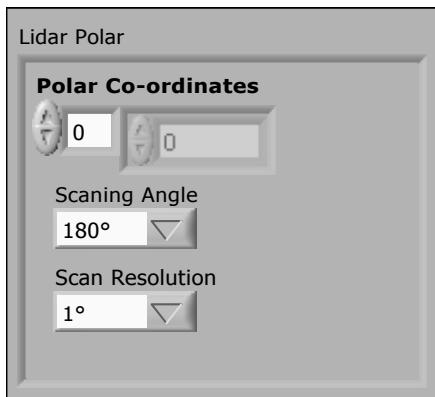
Block Diagram



Connector Pane

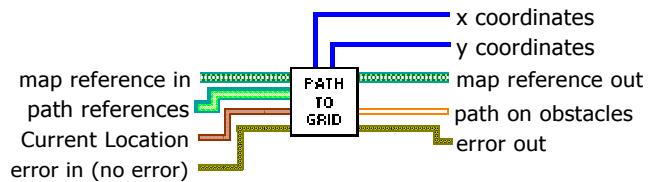
LidarPolarControl.ctl

Front Panel

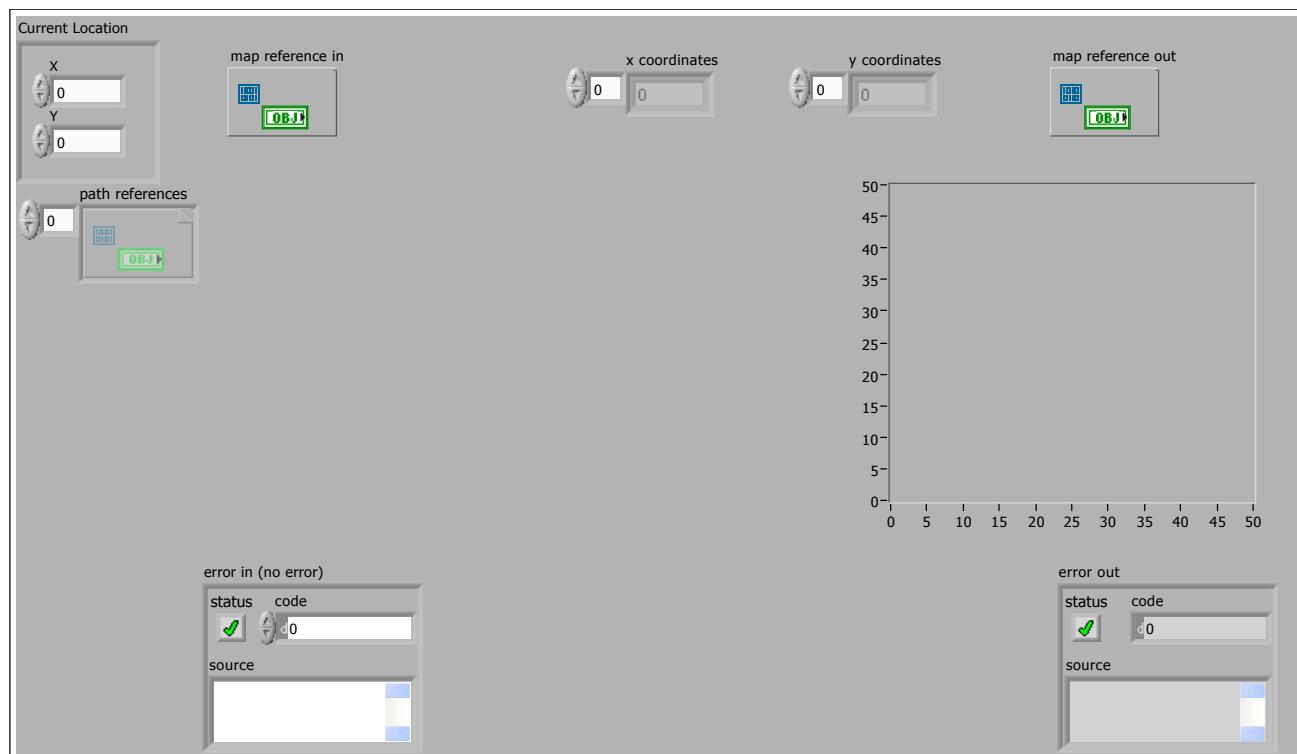


Block Diagram

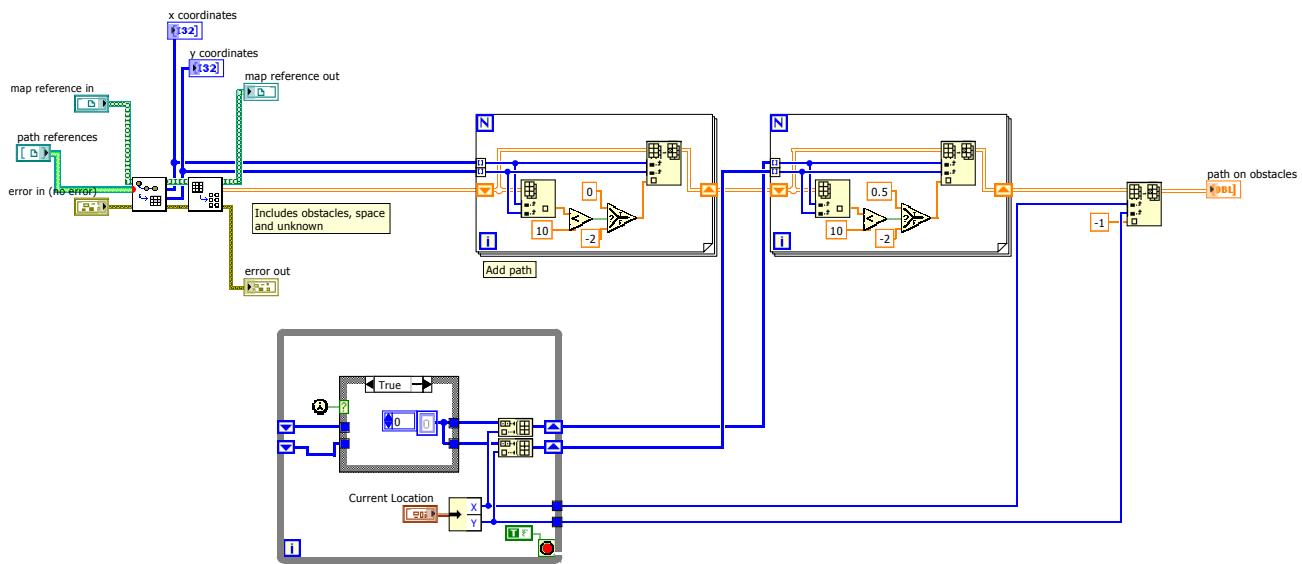
Connector Pane

Path To Grid.vi

Front Panel



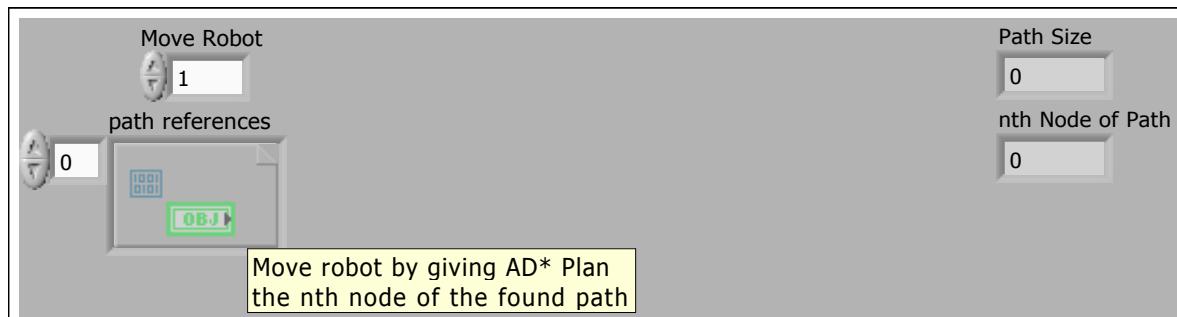
Block Diagram



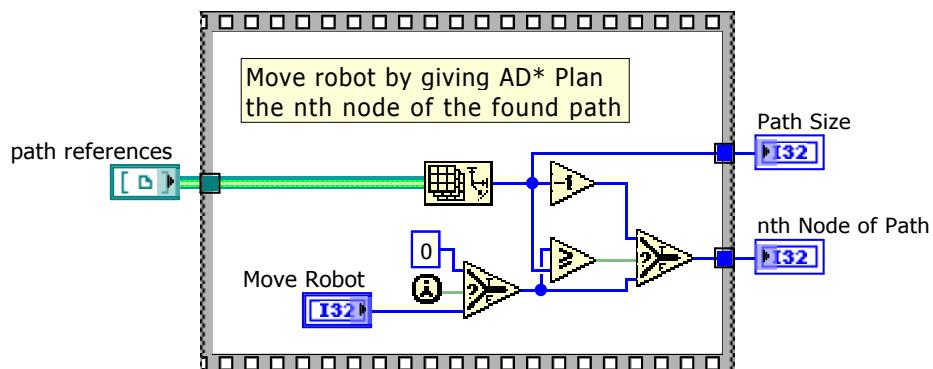
Connector Pane

Retreive Next Path Node.vi

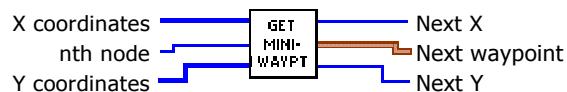
Front Panel



Block Diagram

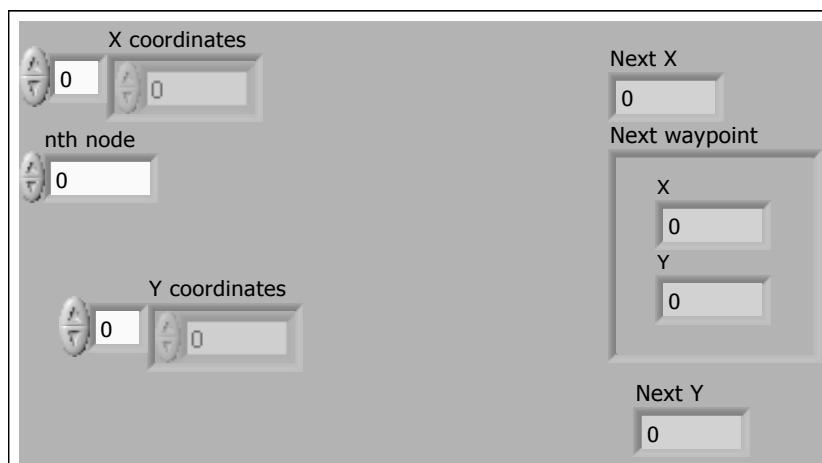


Connector Pane

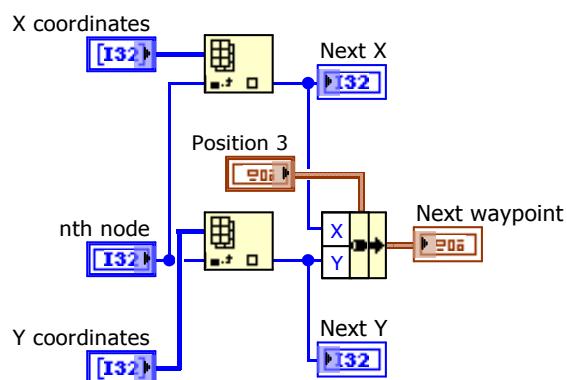
Get Mini Waypt.vi

Takes path coordinates and transforms them to a target mini-waypoint

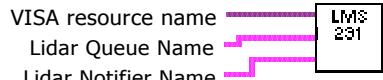
Front Panel



Block Diagram

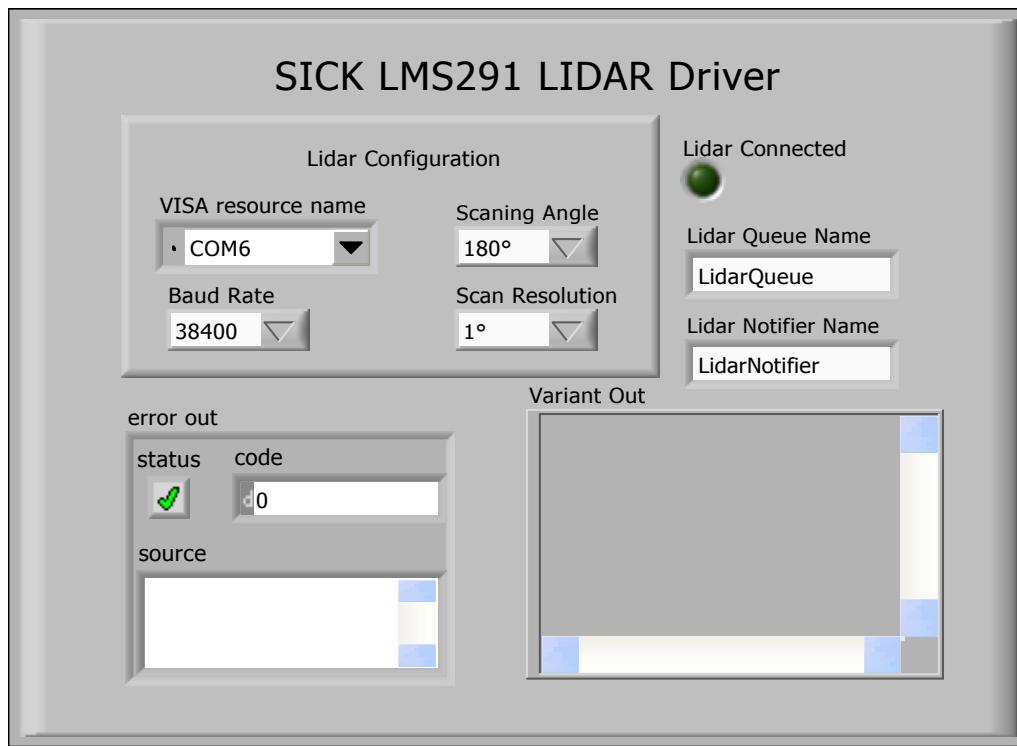


Connector Pane

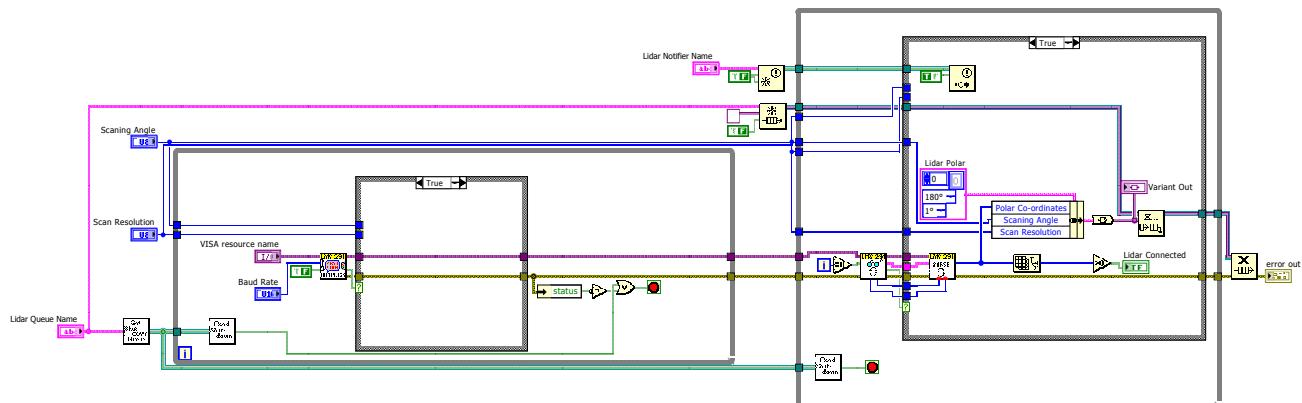
SickLms291Lidar.vi

Low-level LIDAR driver.

Front Panel



Block Diagram

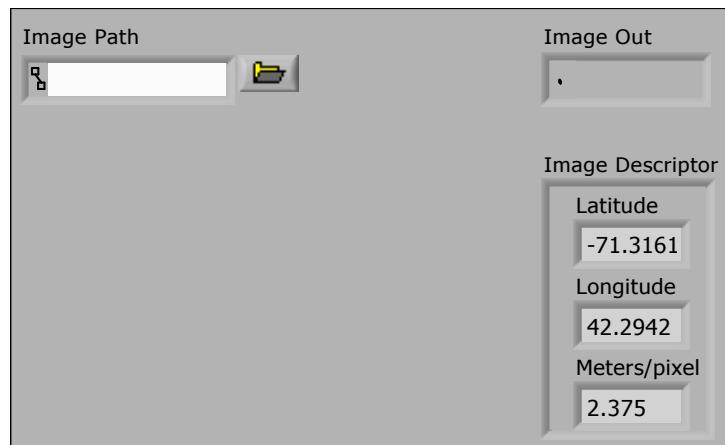


Connector Pane

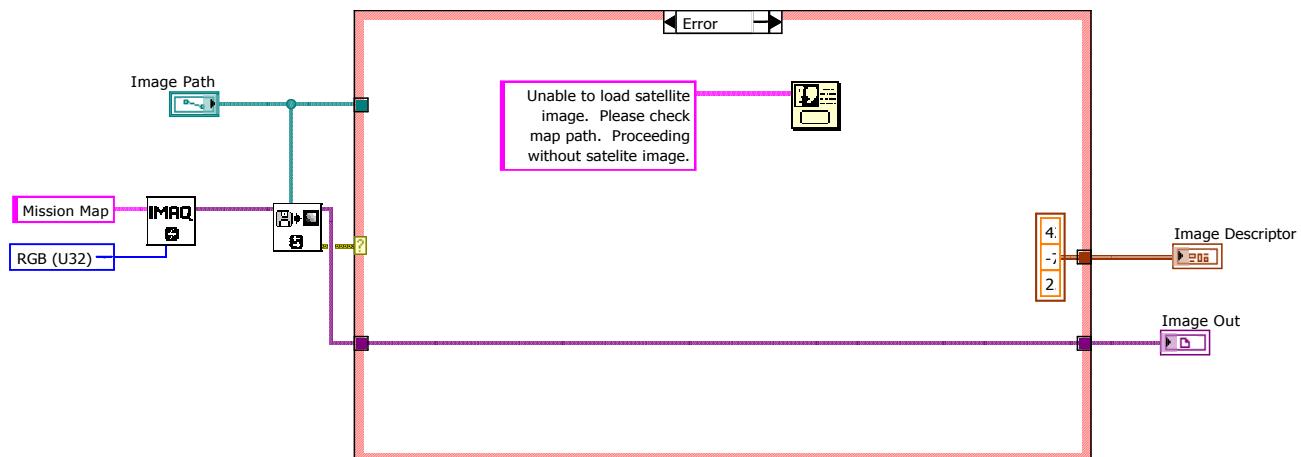
Load Map.vi

Loads a map and its corresponding descriptor text file from the given path. The image and the descriptor file must have the same filename.

Front Panel



Block Diagram



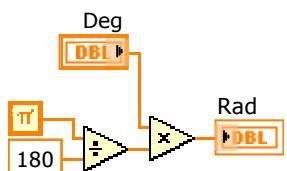
Connector Pane

DegreeToRadian.vi

Front Panel



Block Diagram

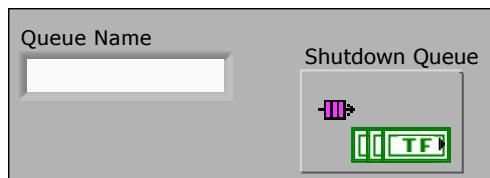


Connector Pane

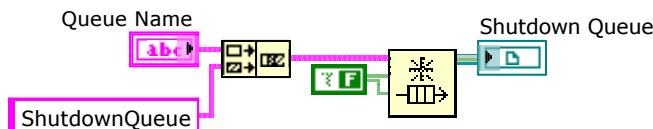
GetShutdownQueue.vi

Returns a reference to the queue-specific shutdown queue (ie LidarQueueShutdownQueue).

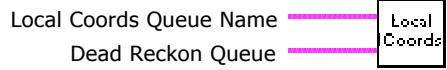
Front Panel



Block Diagram

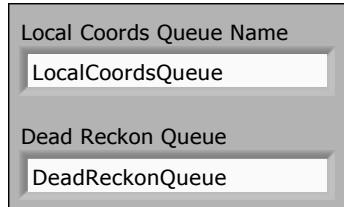


Connector Pane

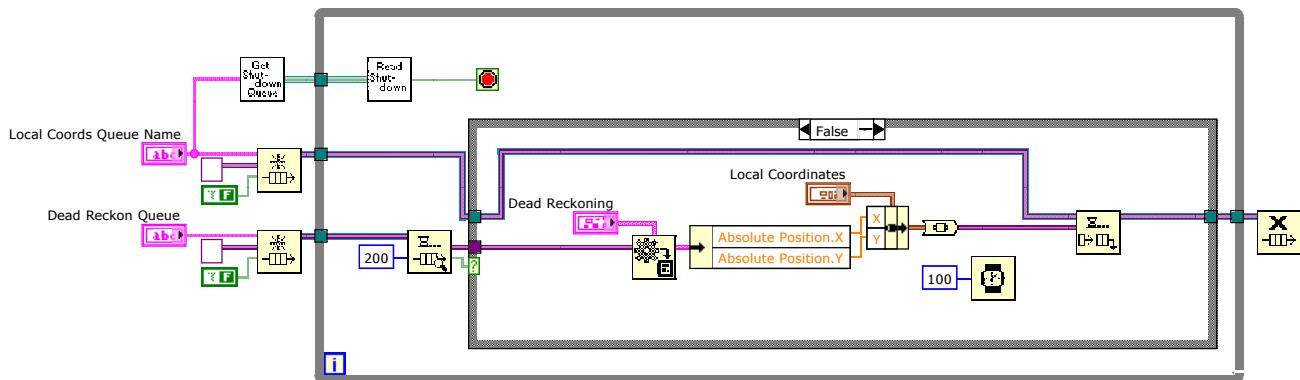
LocalCoords.vi

Use the Dead Reckon Queue to compute Local Coordinates. UNTESTED

Front Panel



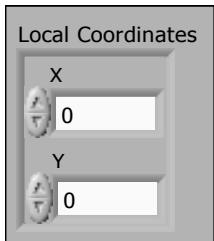
Block Diagram



Connector Pane

LocalCoordinatesControl.ctl

Front Panel

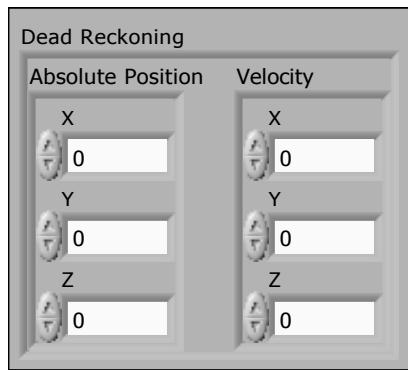


Block Diagram

Connector Pane

Dead Reckoning.ctl

Front Panel

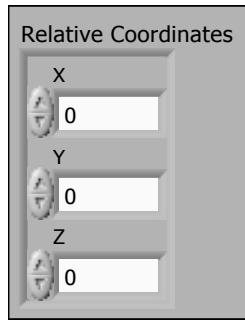


Block Diagram

Connector Pane

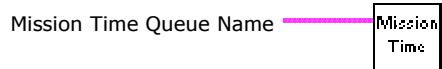
Relative Coordinate Cluster.ctl

Front Panel



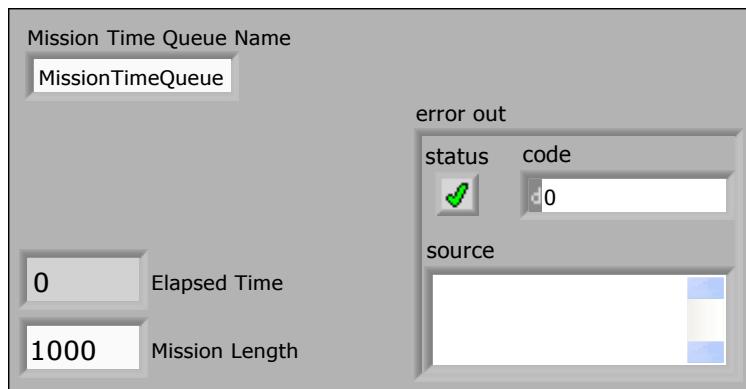
Block Diagram

Connector Pane

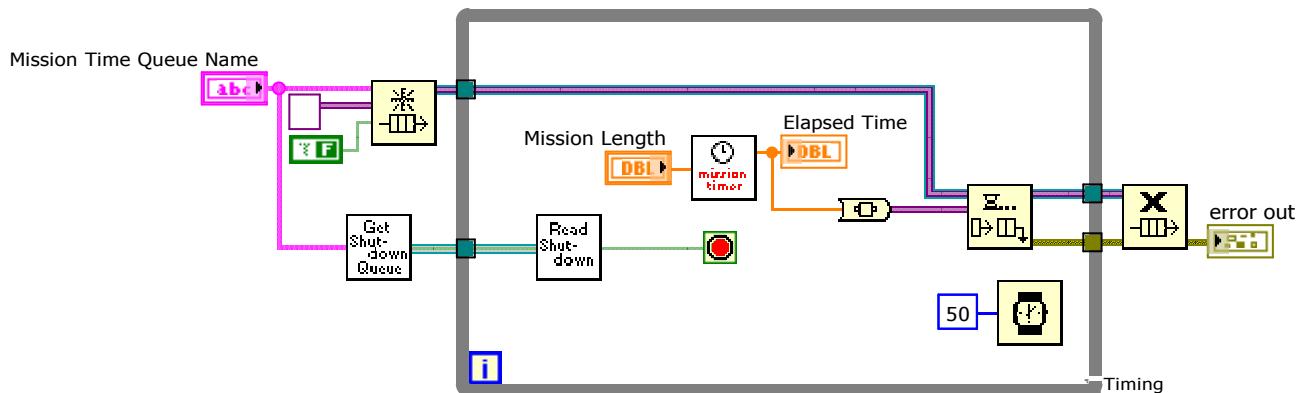
Mission Time with Queue.vi

Enqueue Mission Time.

Front Panel



Block Diagram

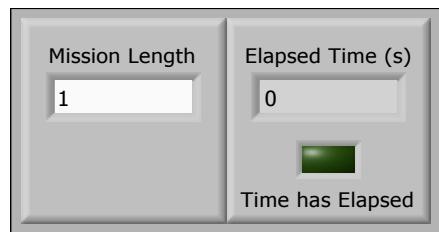


Connector Pane

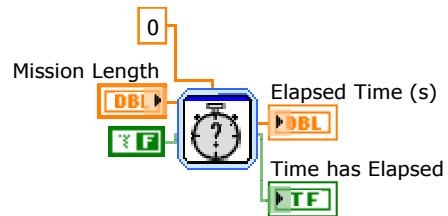
Mission Time.vi

Returns the mission time in seconds, as well as a flag to determine whether the mission time is greater than the mission length

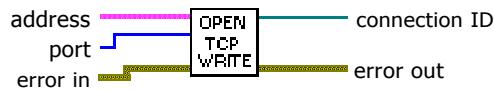
Front Panel



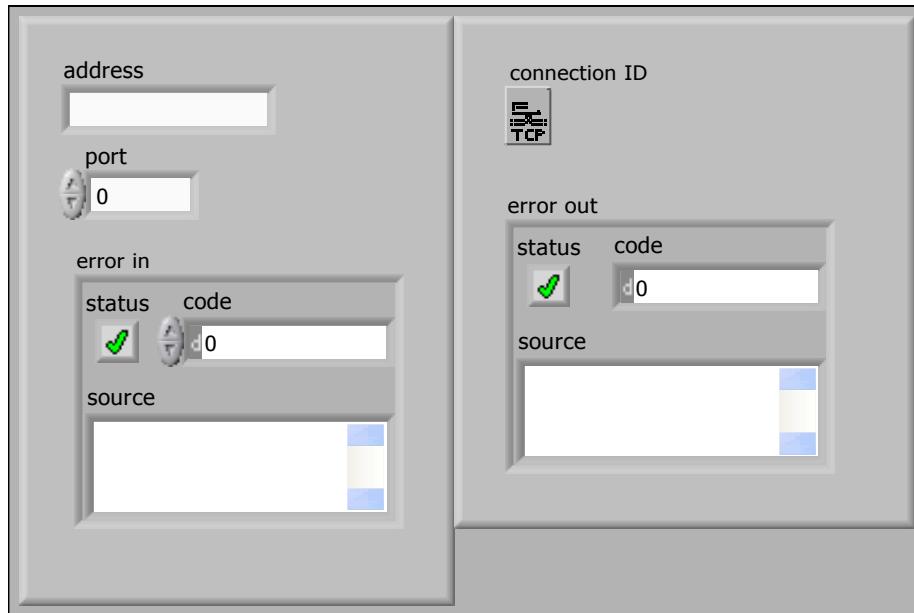
Block Diagram



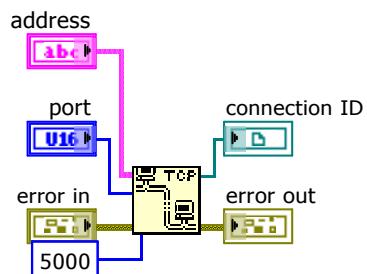
Connector Pane

TCP Open Writer.vi

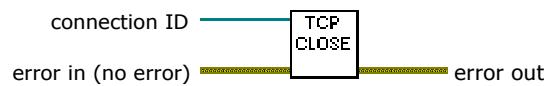
Front Panel



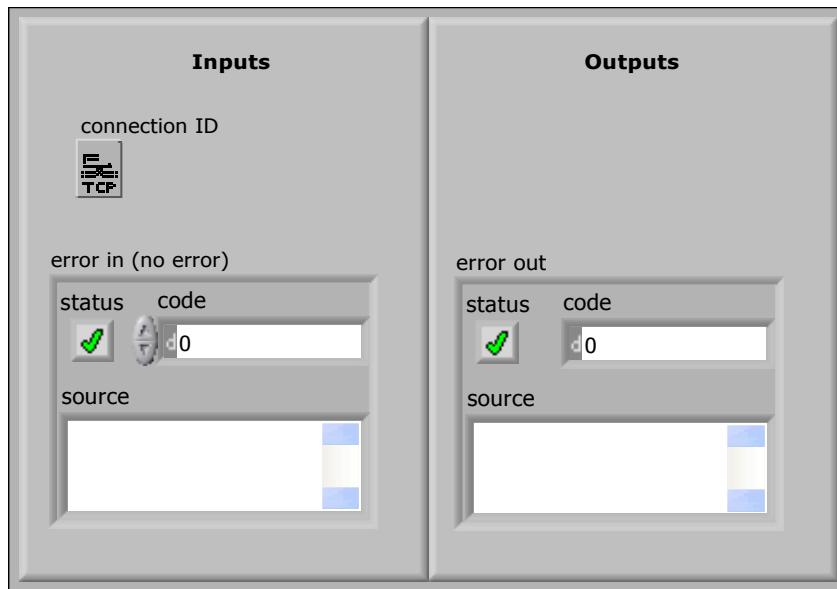
Block Diagram



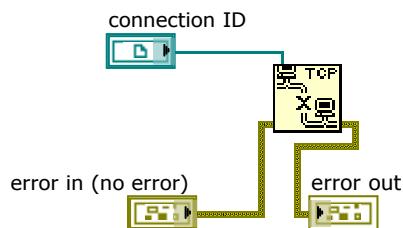
Connector Pane

TCP Close.vi

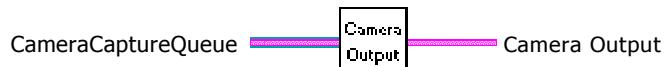
Front Panel



Block Diagram

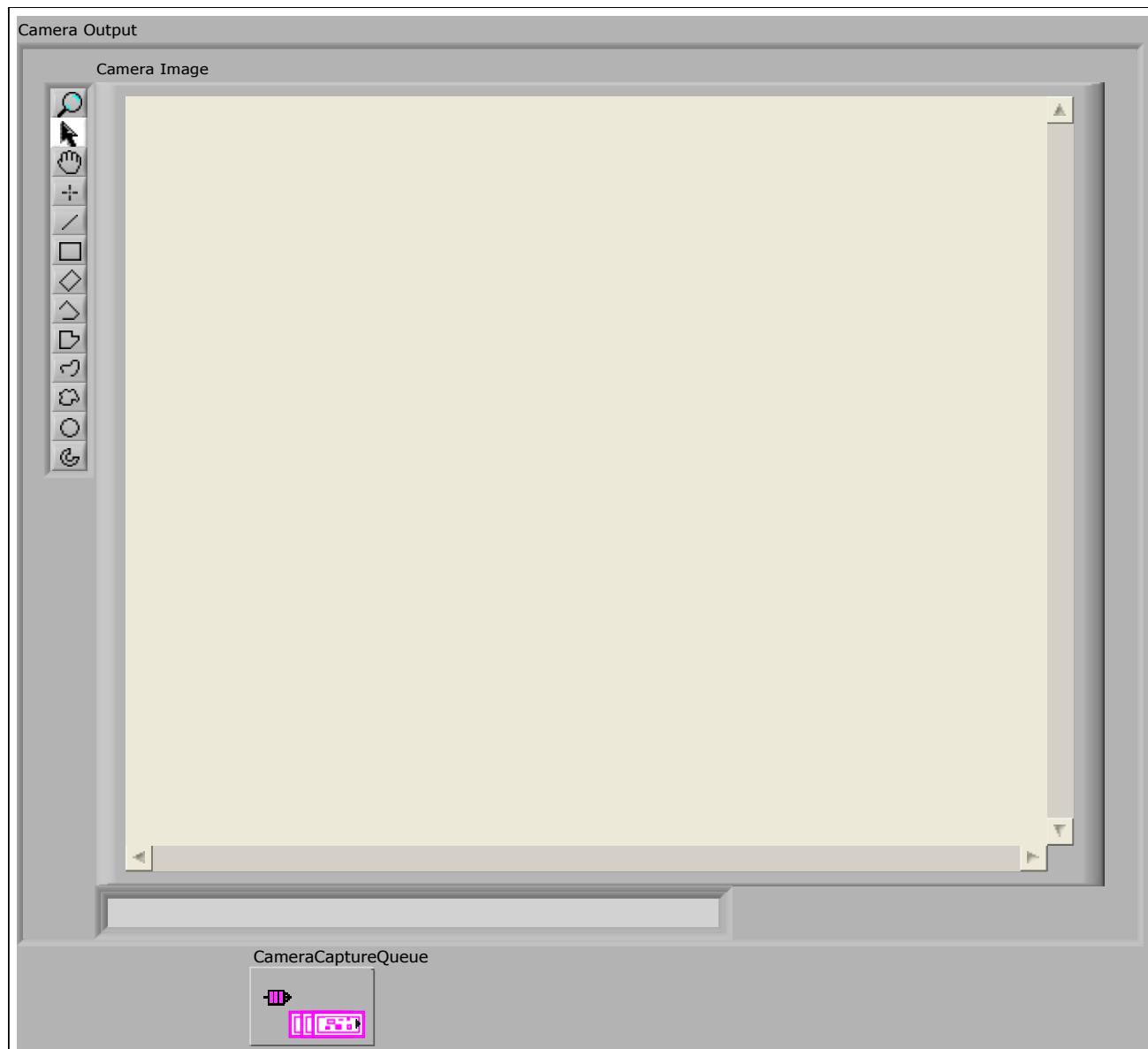


Connector Pane

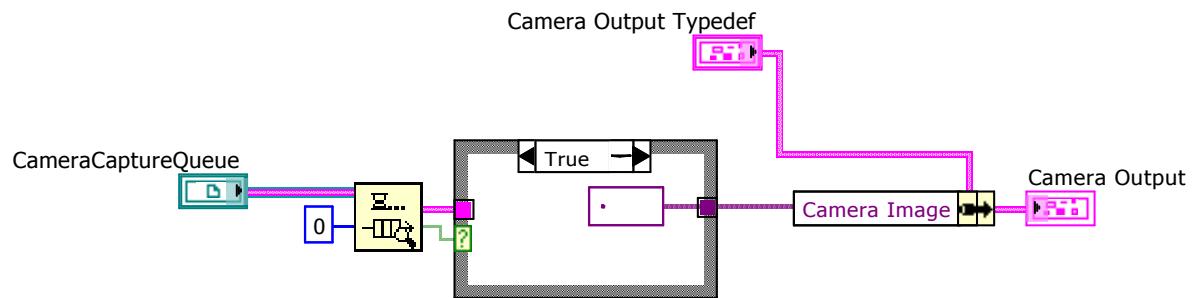
CameraOutput.vi

Display camera output.

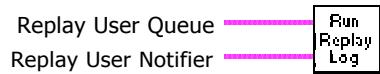
Front Panel



Block Diagram



Connector Pane

RunReplayLog.vi

Spawns a replay thread (ReplayLog.vi or ReplayImageLog.vi) for each queue that is being replayed. Also calls the Replay Control Dialog.

Note that this VI references ReplayLog.vi and ReplayImageLog.vi by name, so be careful renamming or moving those files.

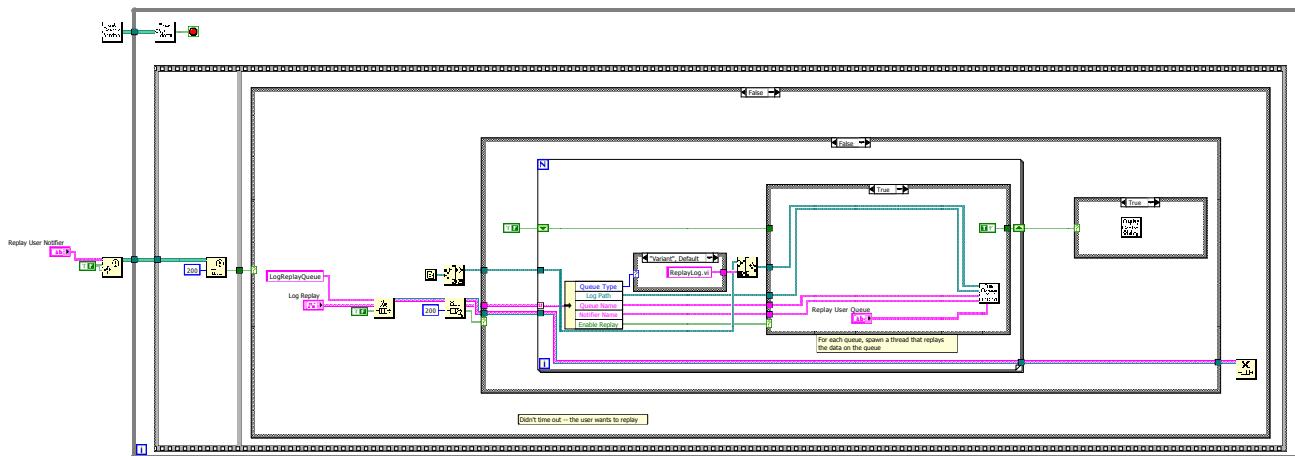
Front Panel

This VI spawns threads that replay systems. Do NOT move this around or **ReplayLog.vi** without chcecking the paths referenced in the block diagram first.

Replay User Queue
ReplayUserQueue

Replay User Notifier
ReplayUserNotfier

Block Diagram

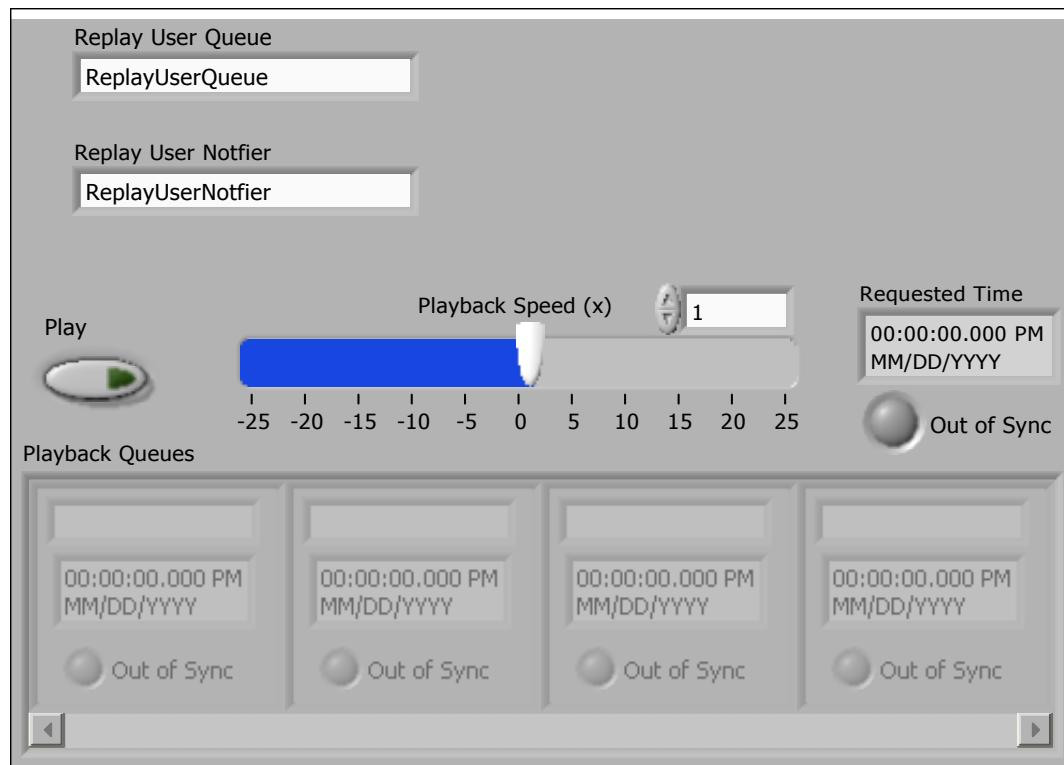


Connector Pane

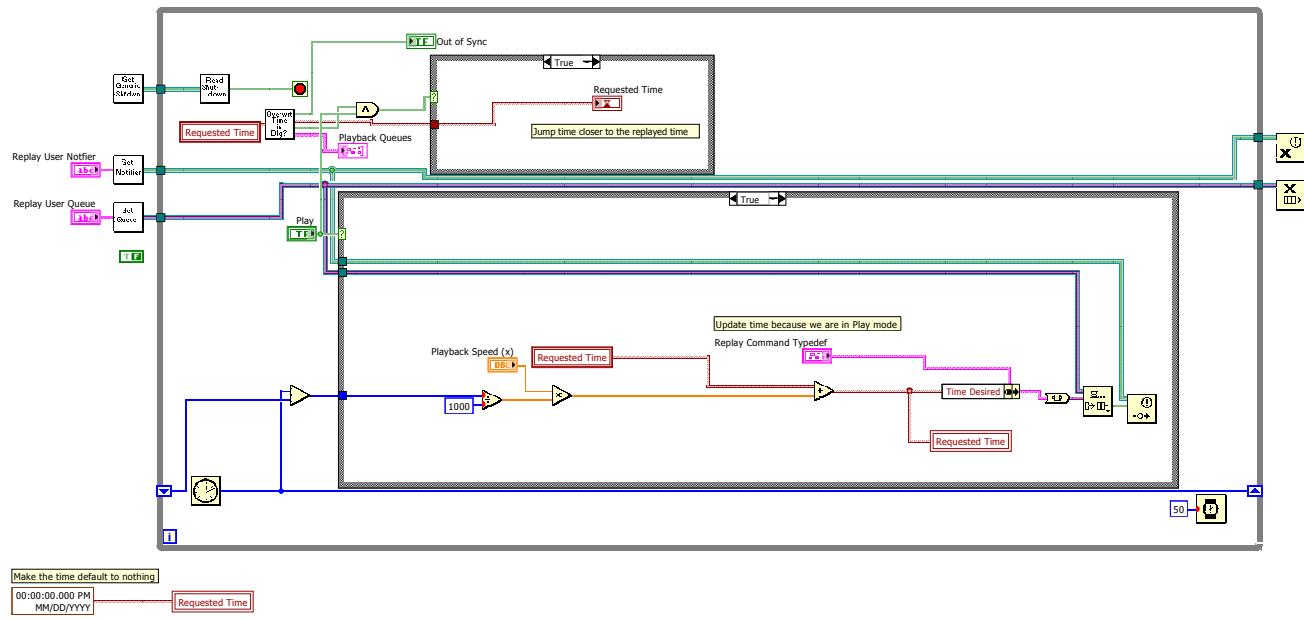
ReplayControlDialog.vi

Replay control dialog. Sends commands to all threads replaying data on the **ReplayUserQueue**.

Front Panel



Block Diagram

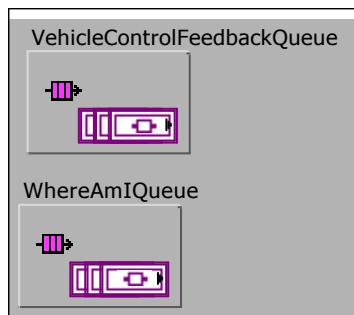


Connector Pane

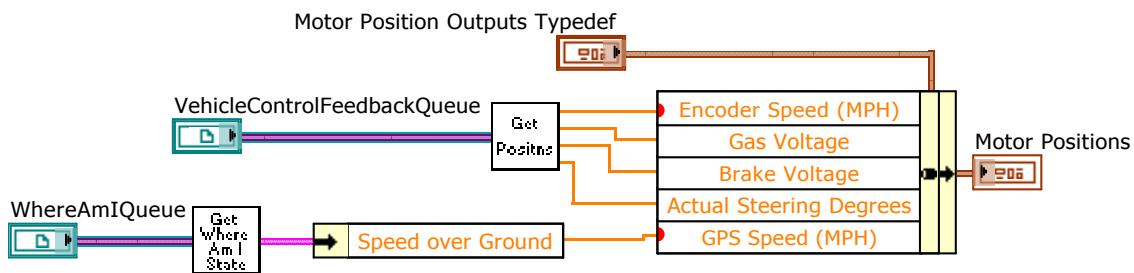
GetMotorPositions.vi

GUI function that reads the gas, brake, steering degrees, and encoder speed in MPH.

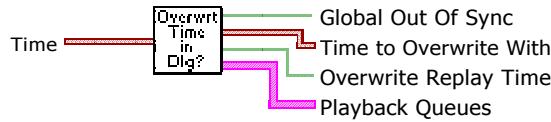
Front Panel



Block Diagram



Connector Pane

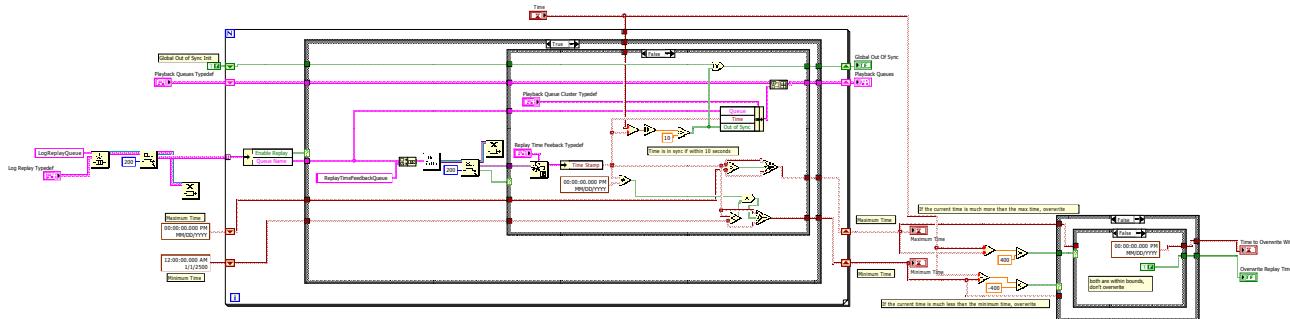
ReplayOverwriteControlTime.vi

Determine if the global replay time (in the replay GUI) should be overwritten because the replaying threads have a very different time.

Front Panel



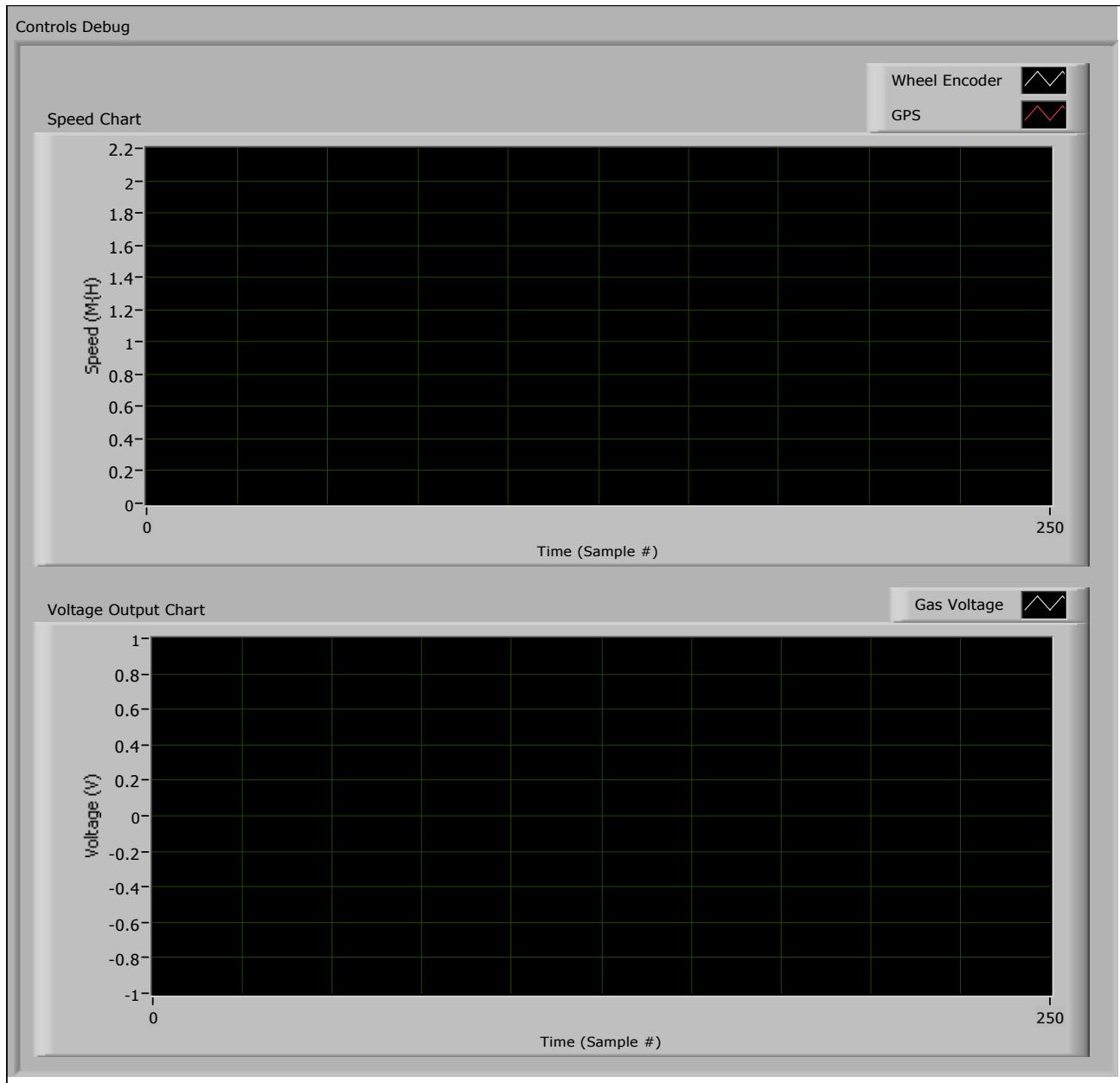
Block Diagram



Connector Pane

ControlsDebugControl.ctl

Front Panel

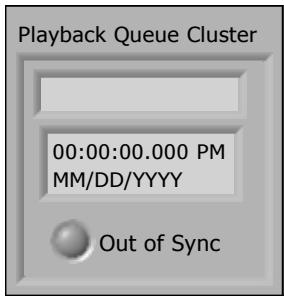


Block Diagram

Connector Pane

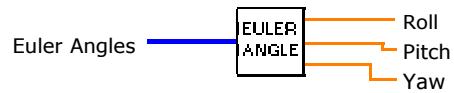
PlaybackQueueTypeDef.ctl

Front Panel

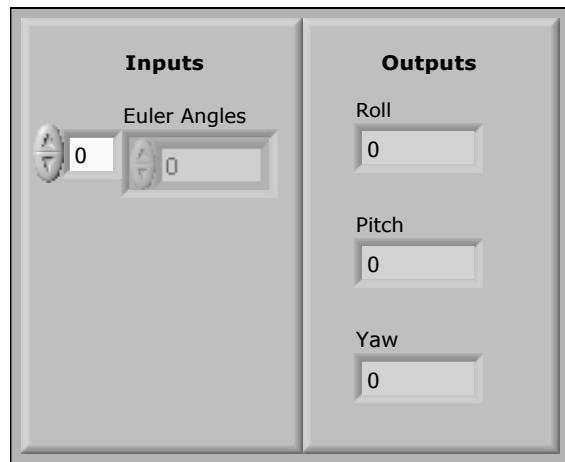


Block Diagram

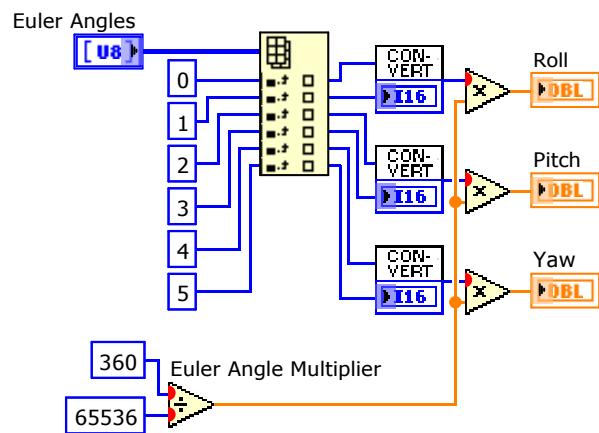
Connector Pane

Interpret Euler Angles.vi

Front Panel



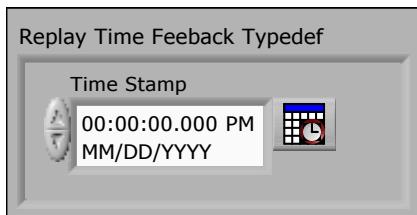
Block Diagram



Connector Pane

ReplayTimeFeebackTypedef.ctl

Front Panel

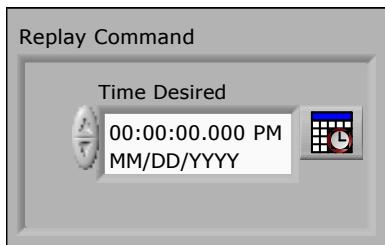


Block Diagram

Connector Pane

ReplayCommandControl.ctl

Front Panel

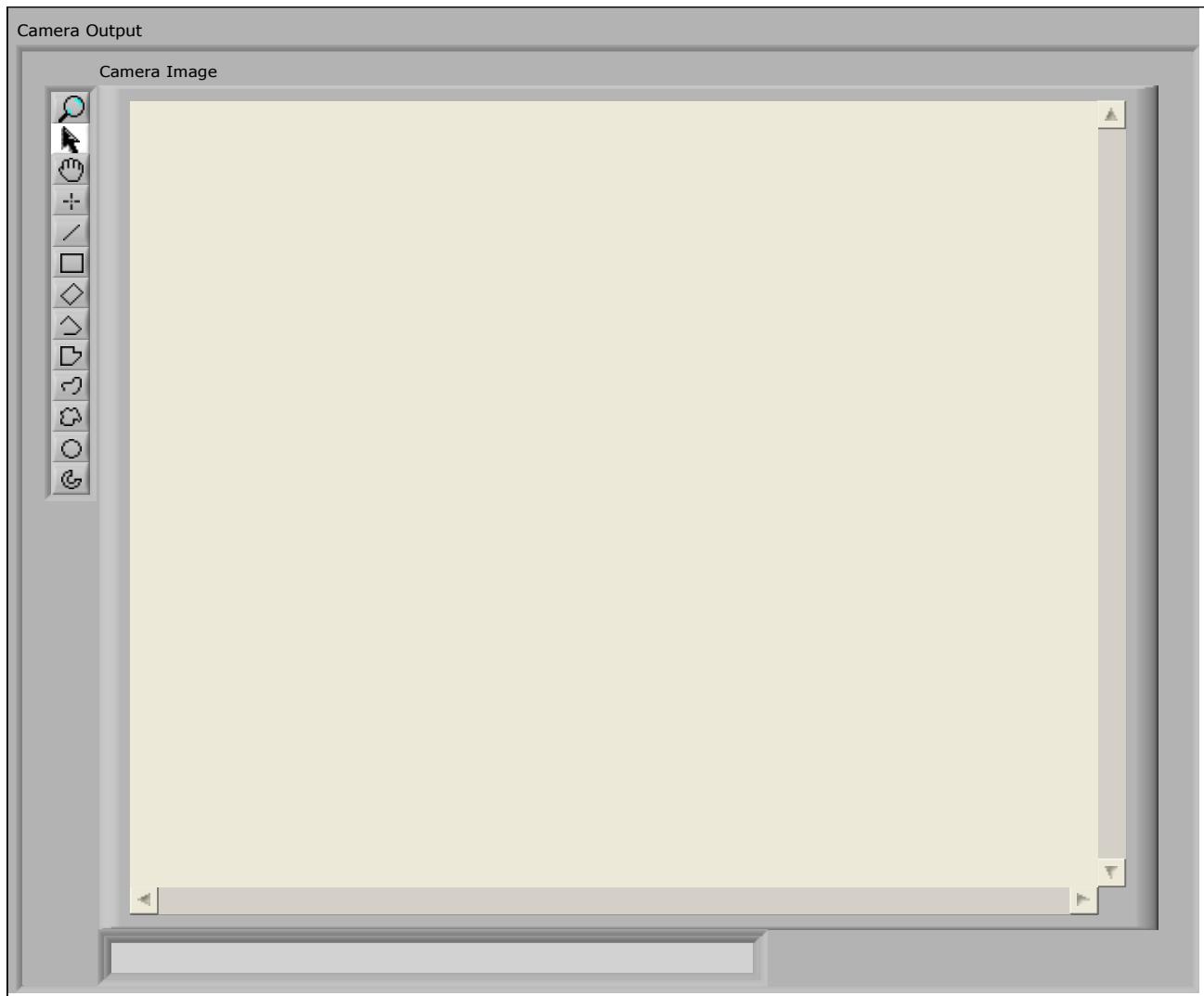


Block Diagram

Connector Pane

CameraOutputs.ctl

Front Panel



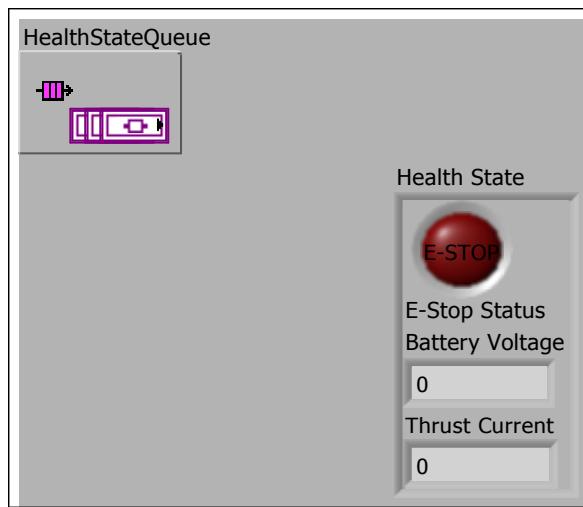
Block Diagram

Connector Pane

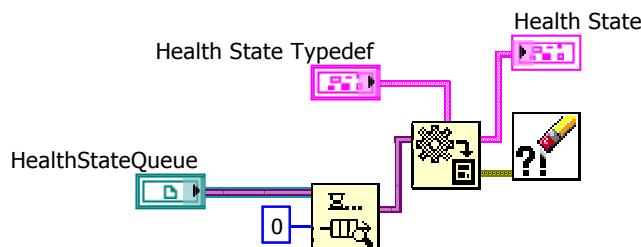
GetHealthState.vi

GUI function to display the health state.

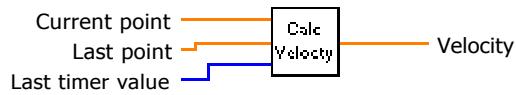
Front Panel



Block Diagram



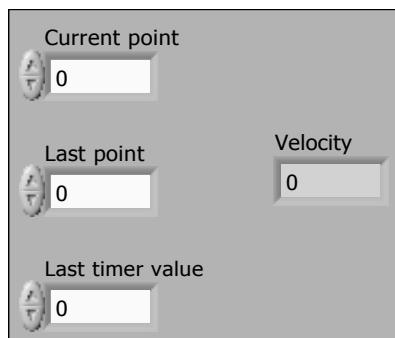
Connector Pane

CalculateVelocity.vi

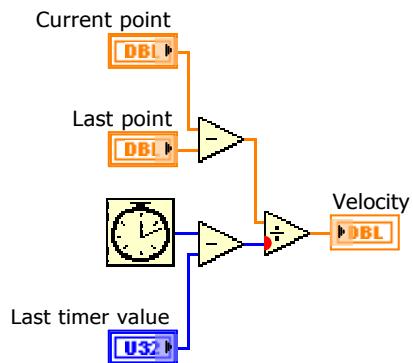
Simple time-based velocity calculator.

$$\text{Velocity} = (\text{Current Point} - \text{Last Point}) / (\text{Current Time} - \text{Last Time})$$

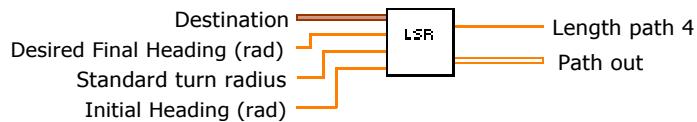
Front Panel



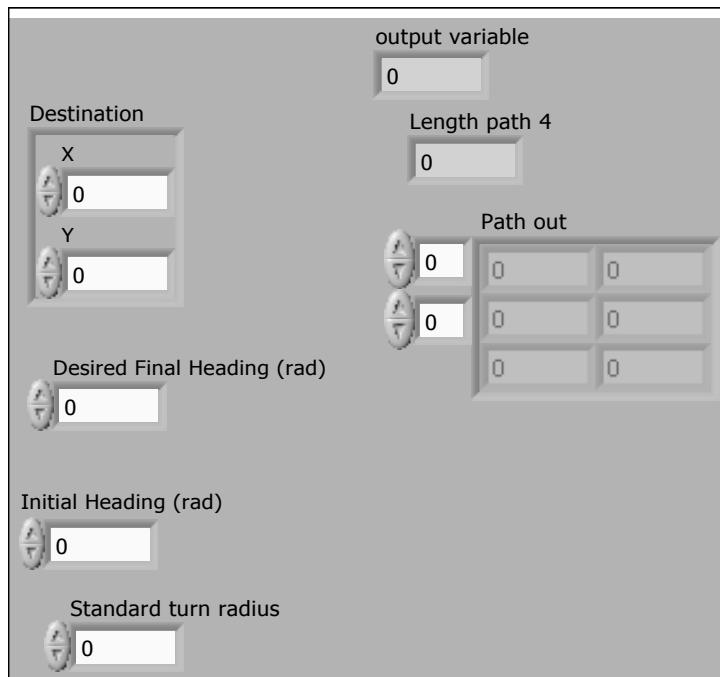
Block Diagram



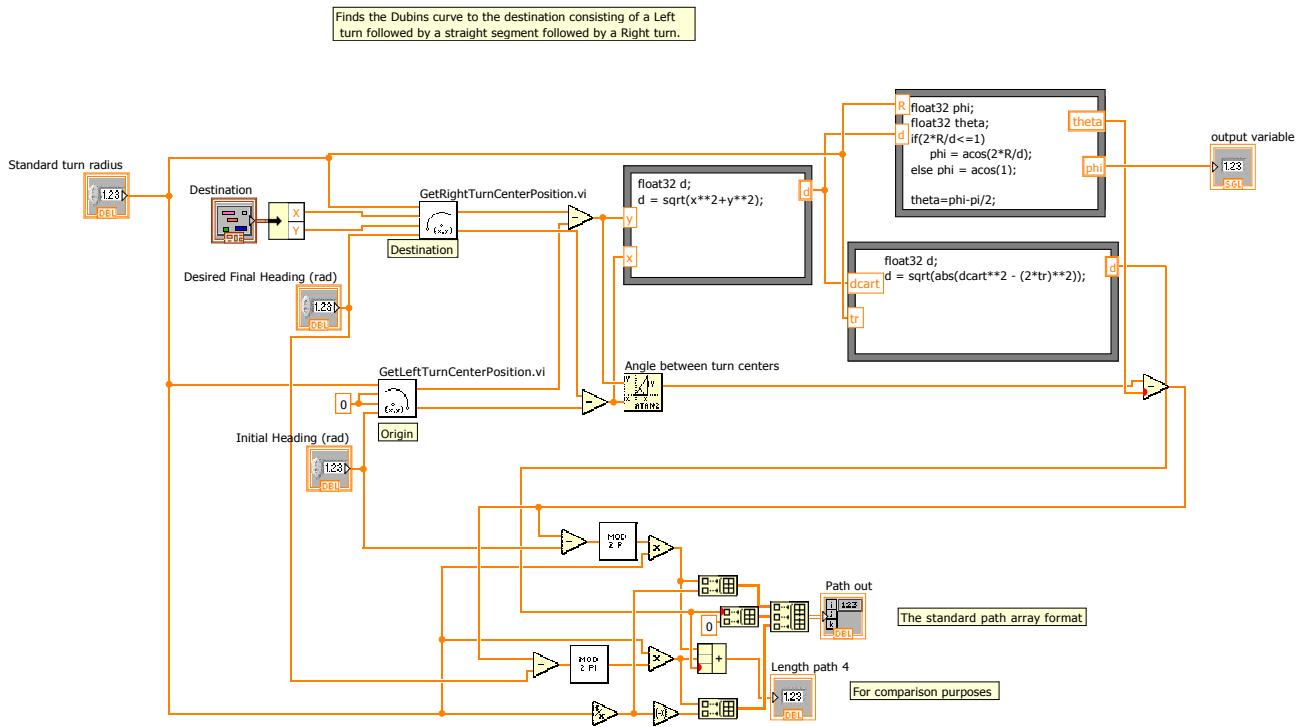
Connector Pane

FindLSRPath.vi

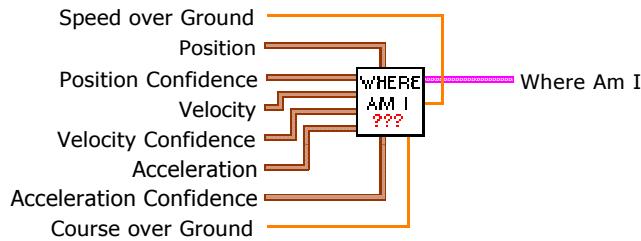
Front Panel



Block Diagram

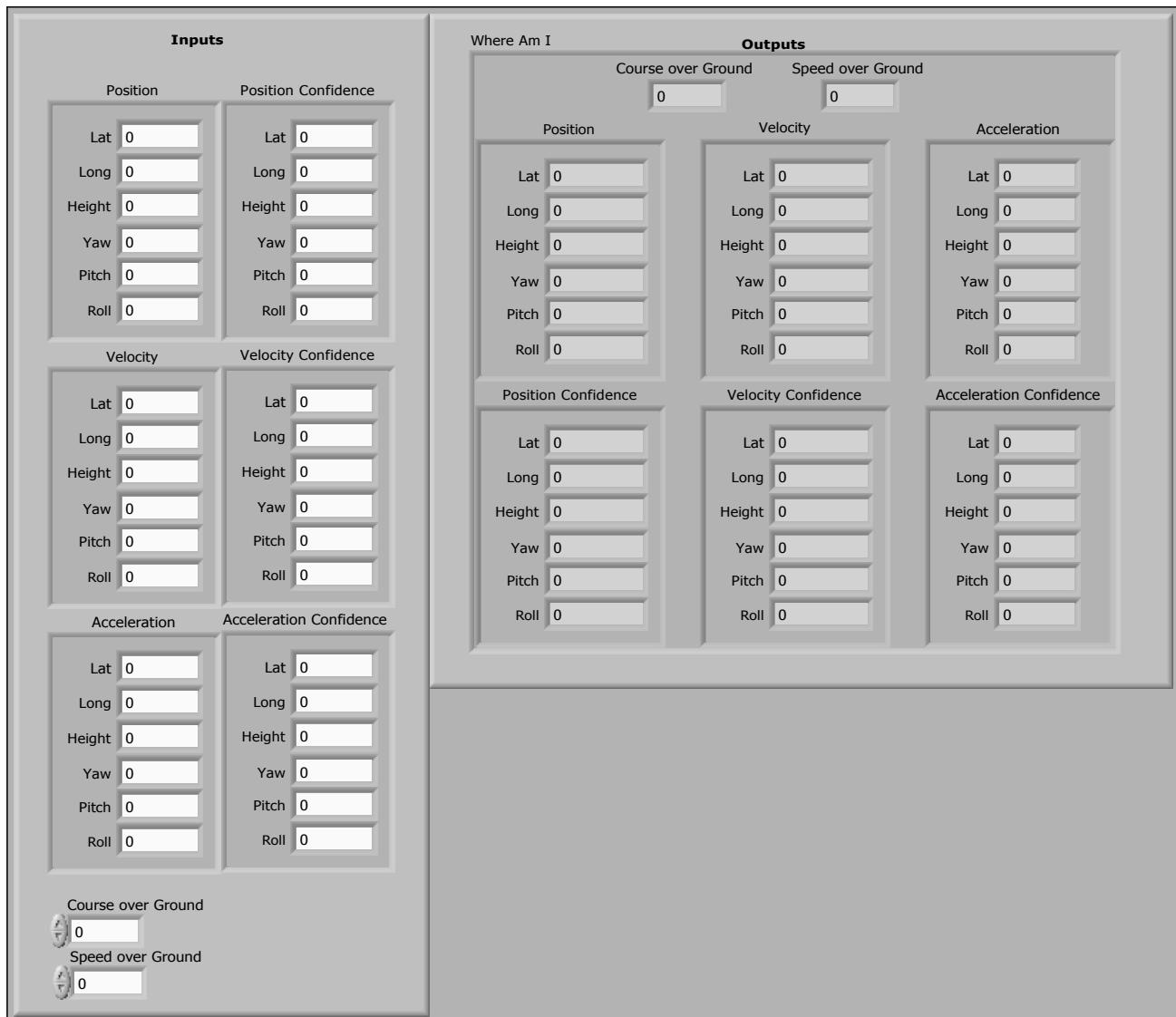


Connector Pane

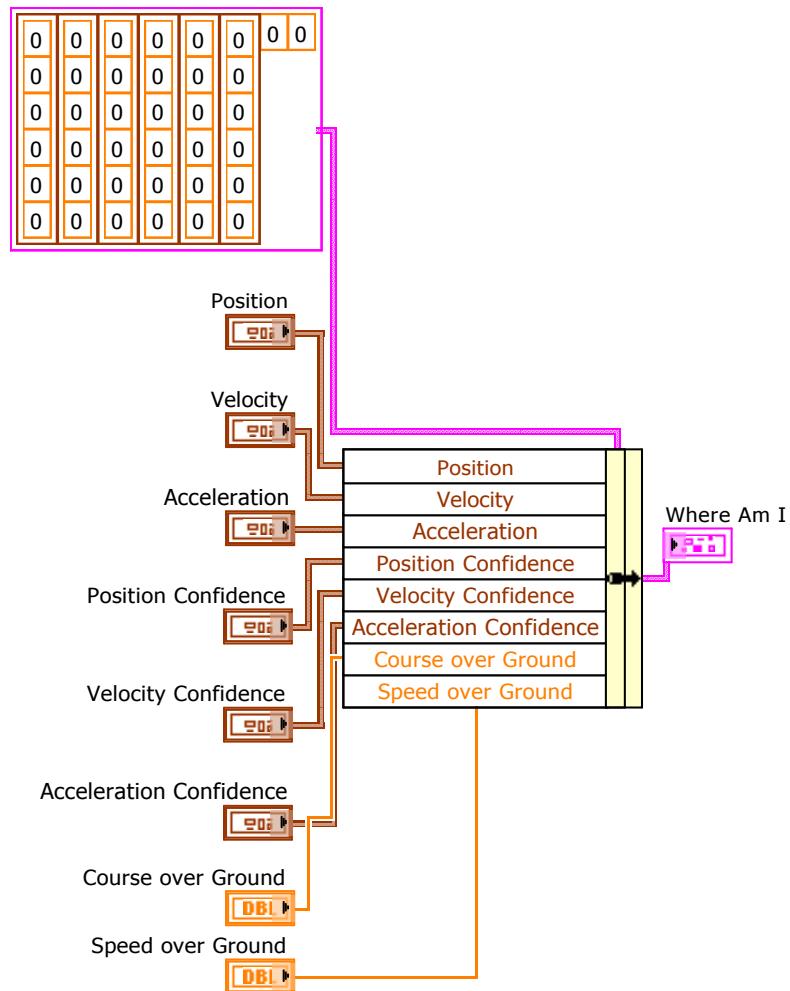
Make Where Am I.vi

Makes a Where Am I block out of its constituent parts.

Front Panel



Block Diagram



Connector Pane

ActMain.vi

Manages all Act systems.

Front Panel



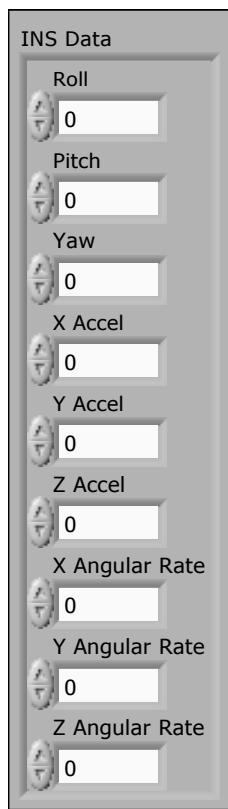
Block Diagram



Connector Pane

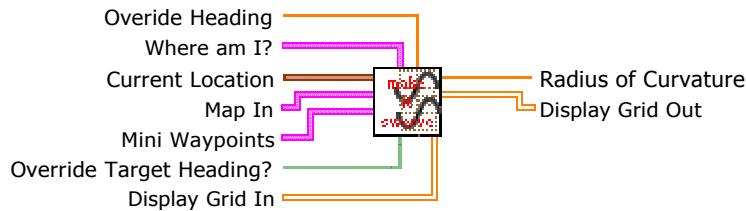
INSData.ctl

Front Panel

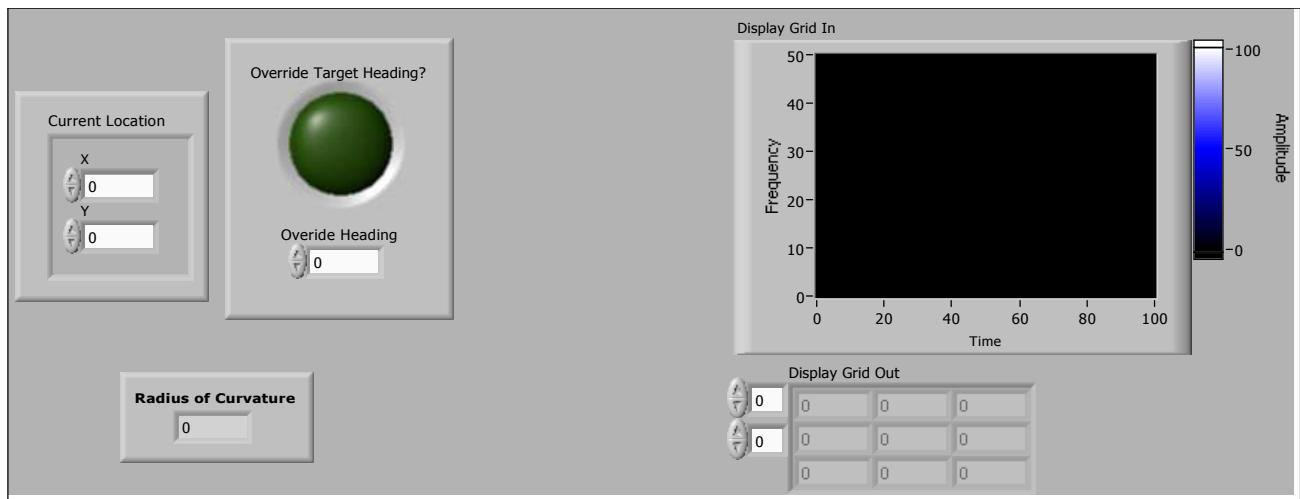


Block Diagram

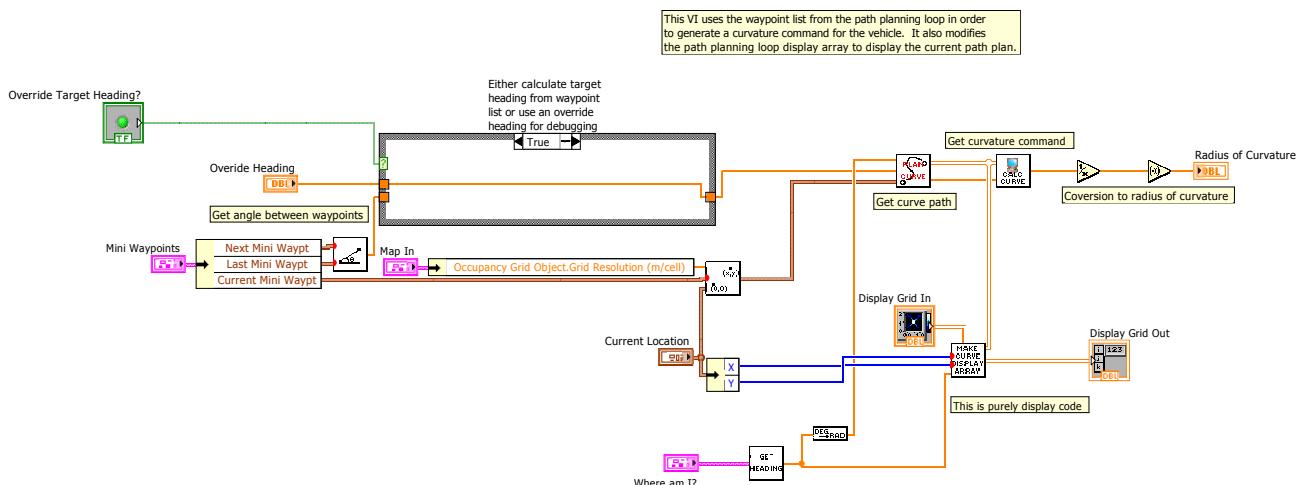
Connector Pane

GetCurvatureCommand.vi

Front Panel



Block Diagram

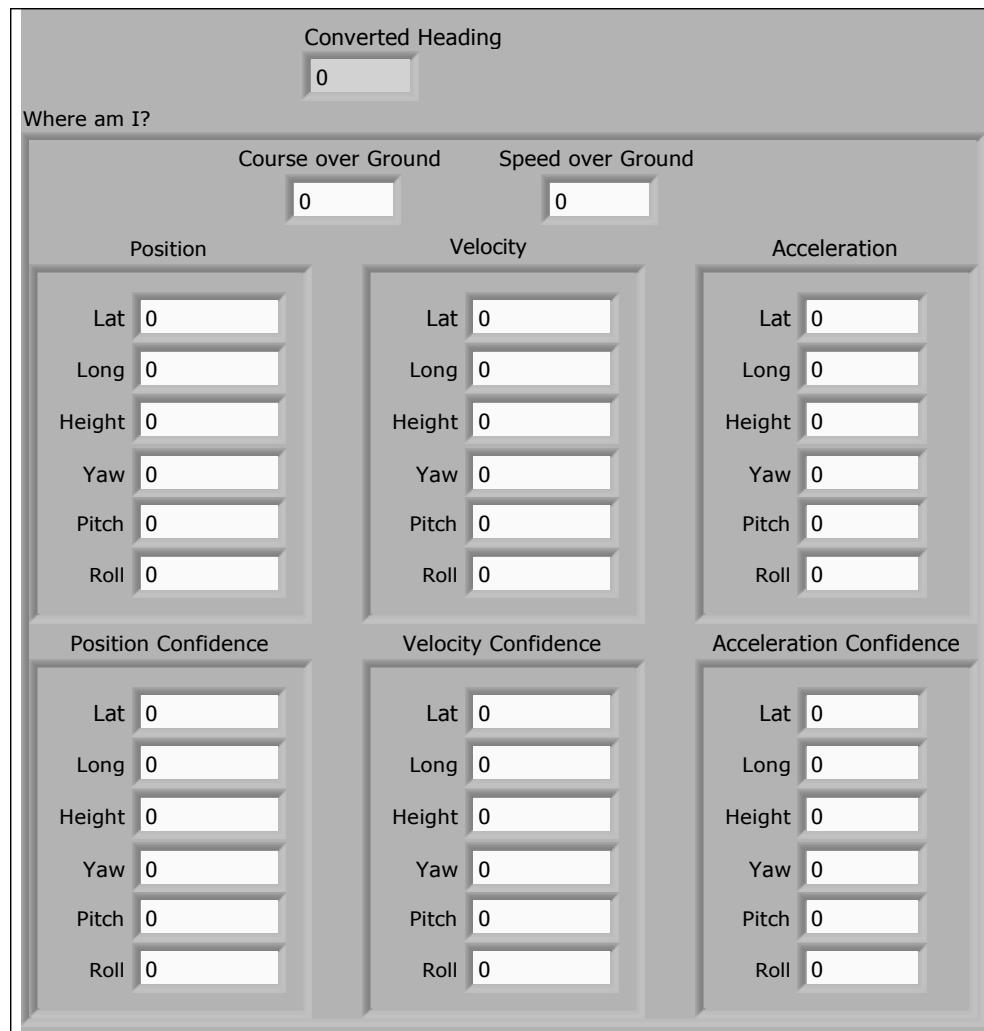


Connector Pane

Get Heading.vi

Converts GPS heading to degrees with 0 degrees along positive x-axis, increasing counter-clockwise

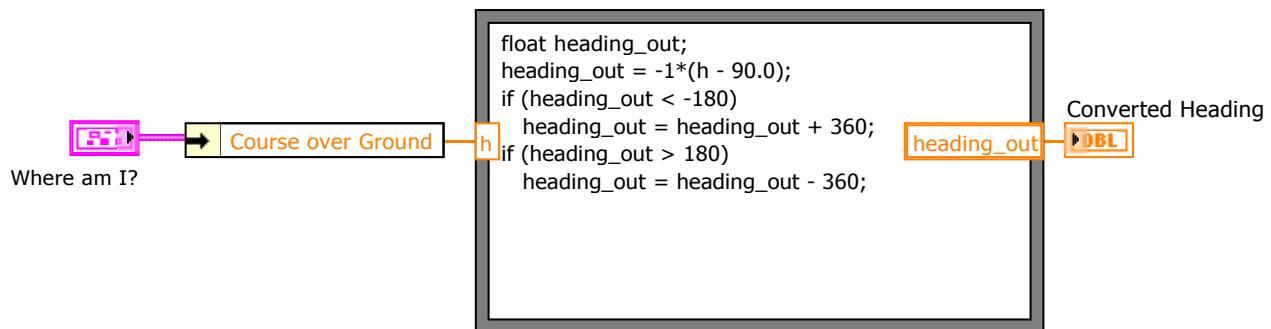
Front Panel



Block Diagram

Change GPS heading to mathematical coordinate system (i.e. East = 0 degrees, angles increase counterclockwise)

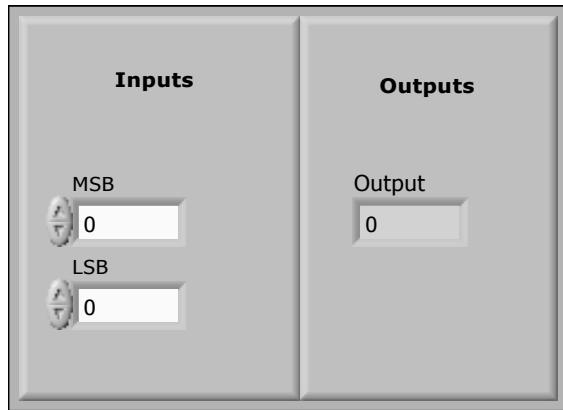
Angles should be greater than -180 and less than +180



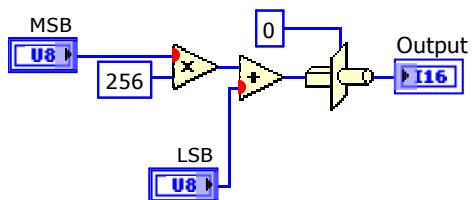
Connector Pane

Convert Bytes to I16.vi

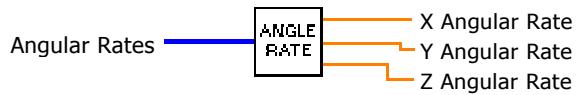
Front Panel



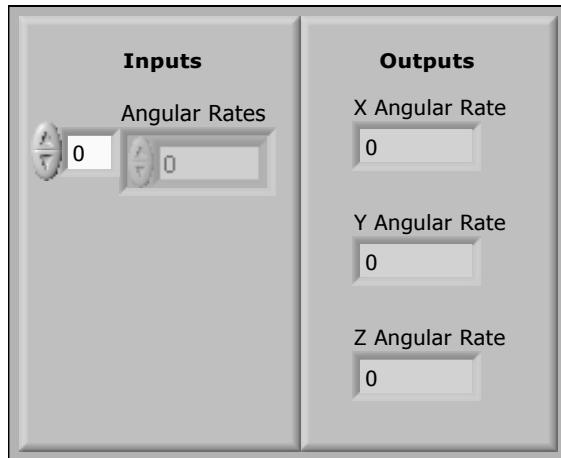
Block Diagram



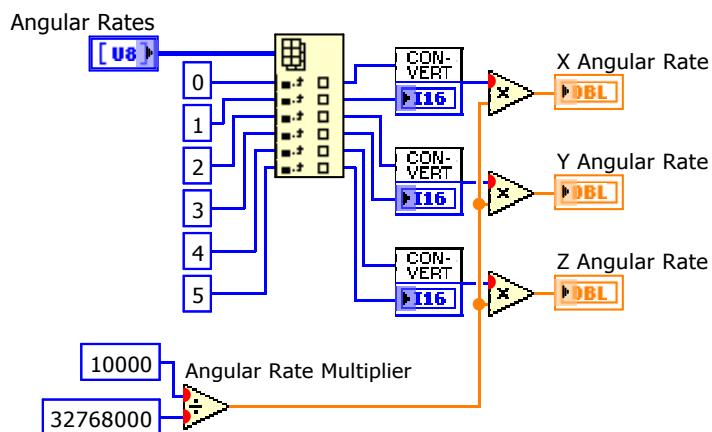
Connector Pane

Interpret Angular Rates.vi

Front Panel



Block Diagram



Connector Pane

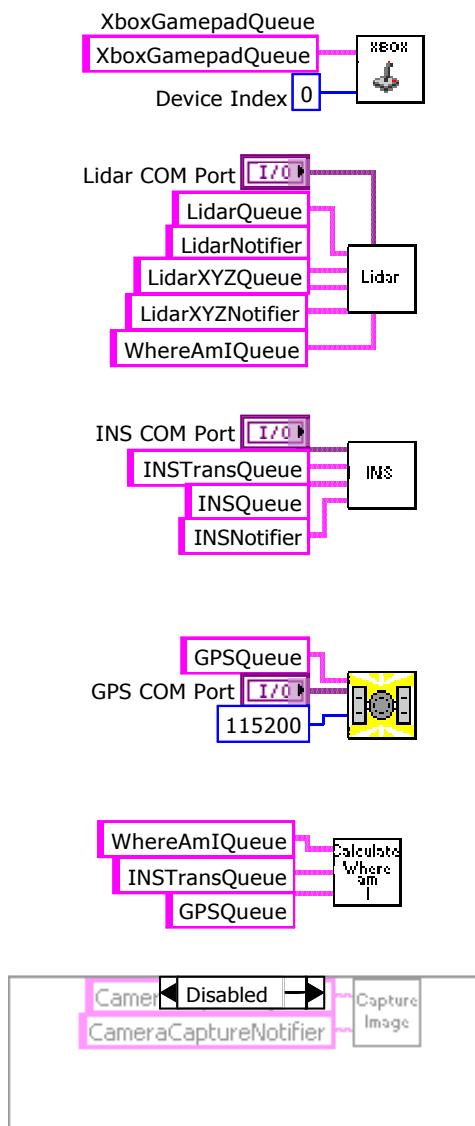
SenseMain.vi

Runs all sensors.

Front Panel



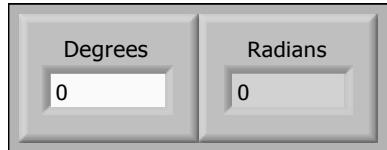
Block Diagram



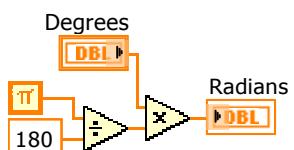
Connector Pane

Degrees to Radians.vi

Front Panel



Block Diagram

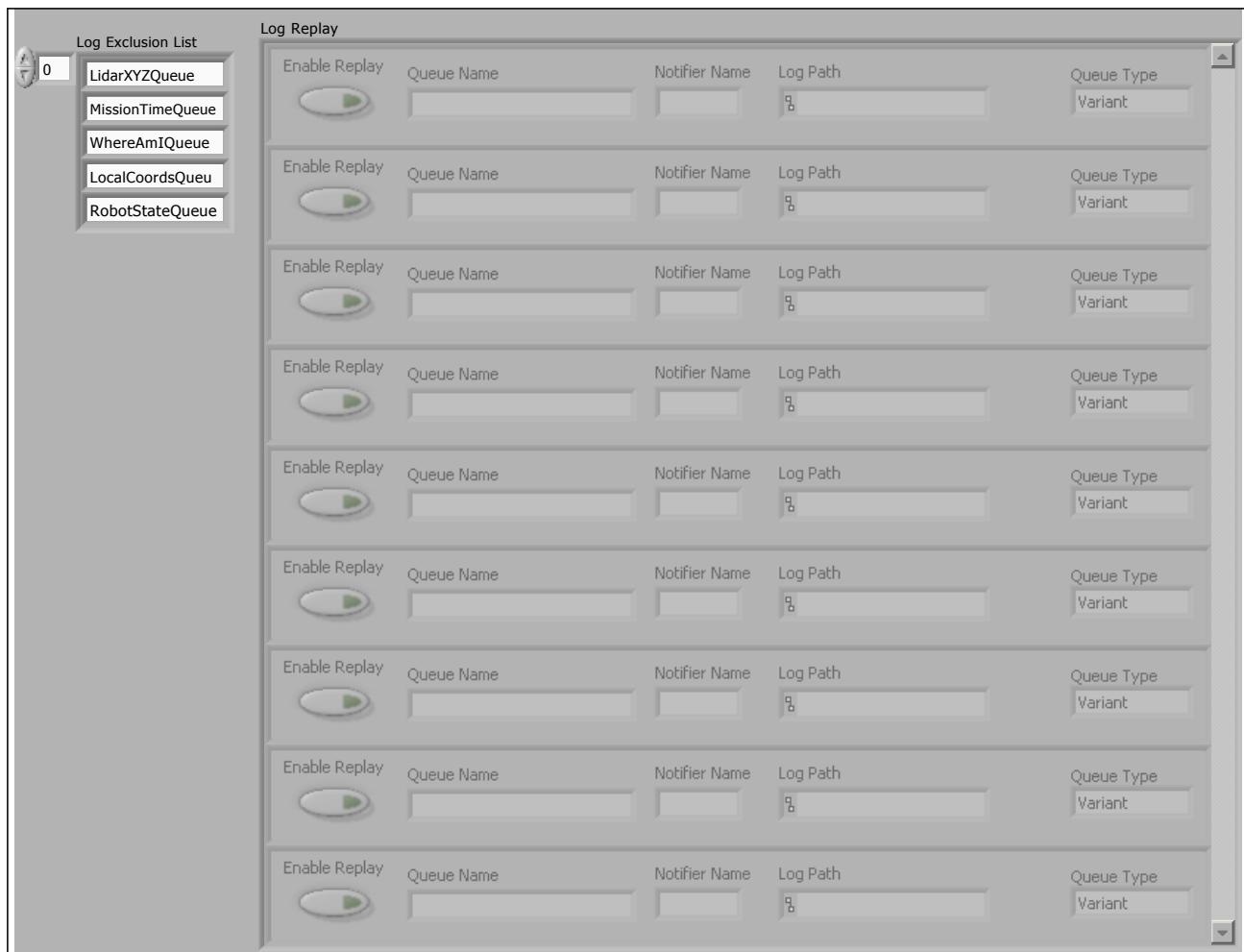


Connector Pane

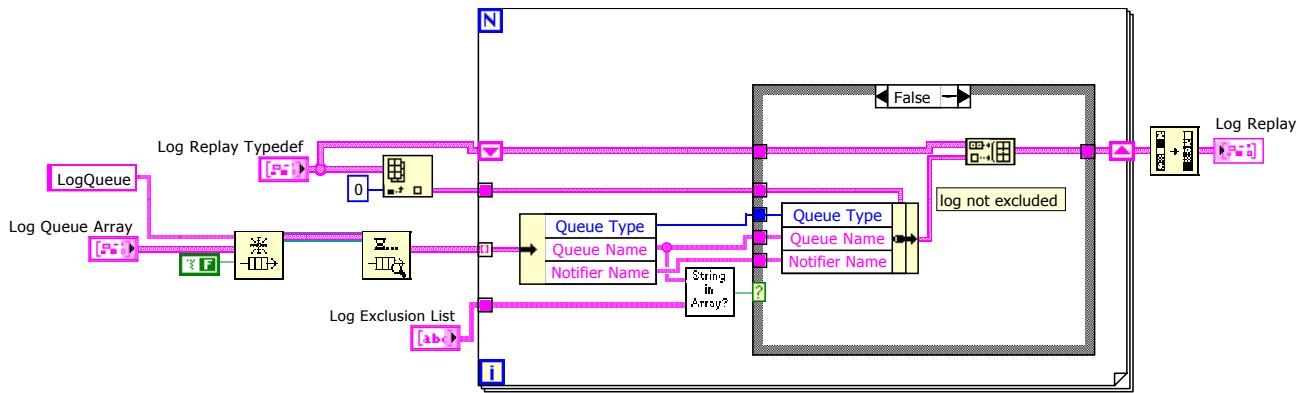
CallLogReplayDialog.vi

Assembles the Replay Dialog array and alphabatizes it.

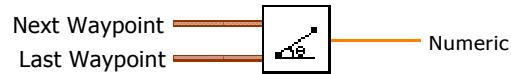
Front Panel



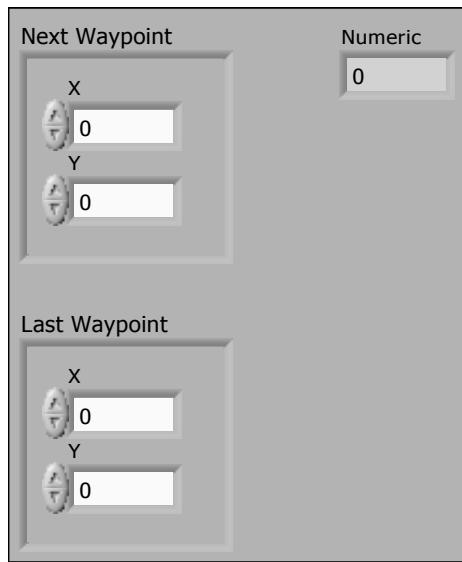
Block Diagram



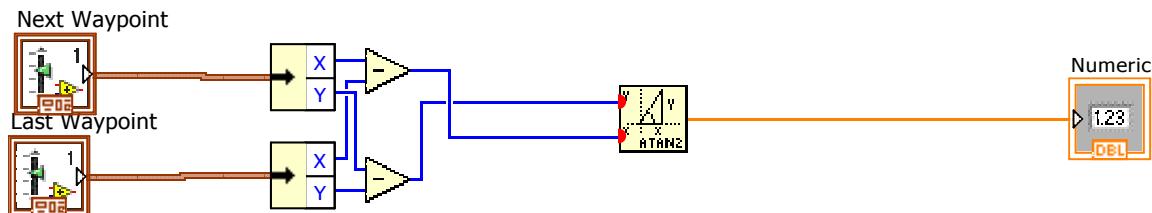
Connector Pane

Angle Between two points.vi

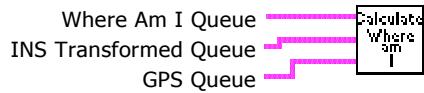
Front Panel



Block Diagram

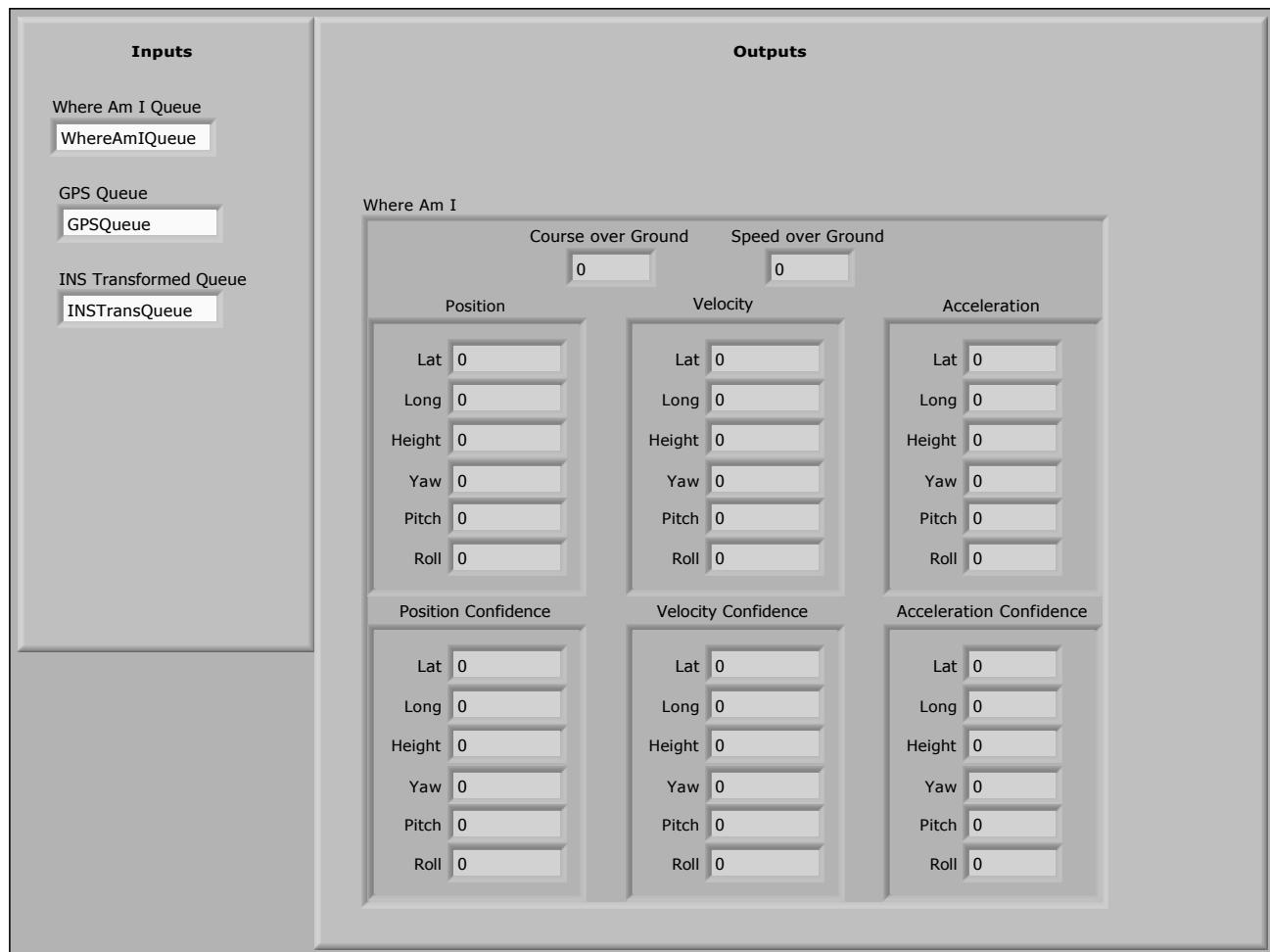


Connector Pane

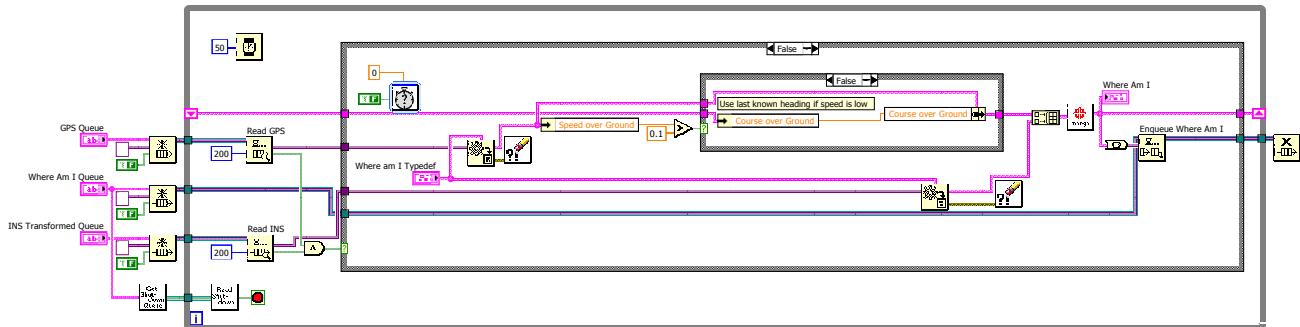
Calculation.vi

Merges the INS and GPS inputs together. For future work, will merge any arbitrary number of Where Am I blocks together.

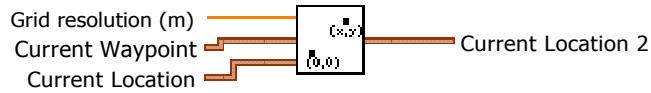
Front Panel



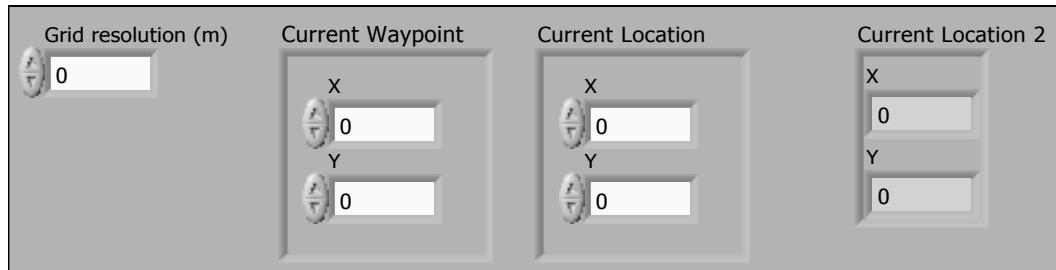
Block Diagram



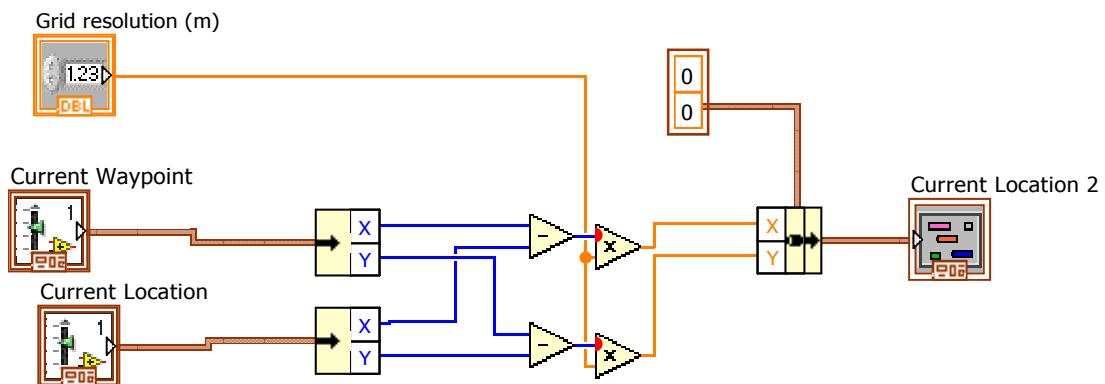
Connector Pane

CalcRelativeWaypointLocation.vi

Front Panel



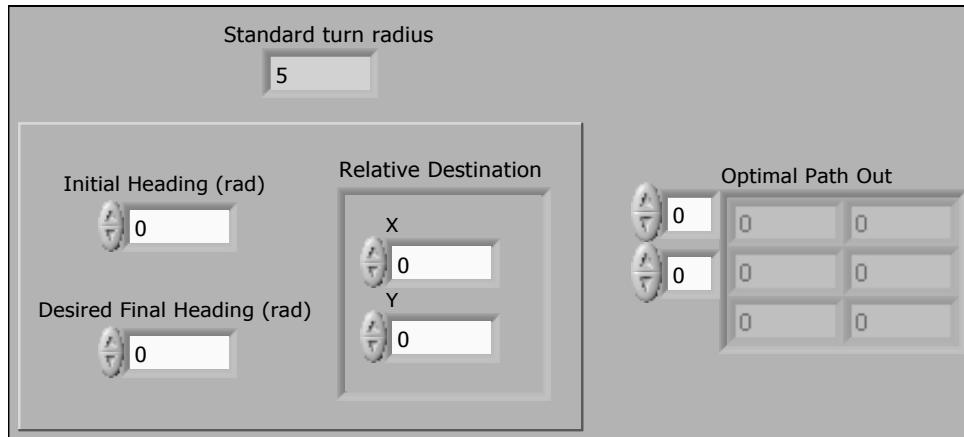
Block Diagram



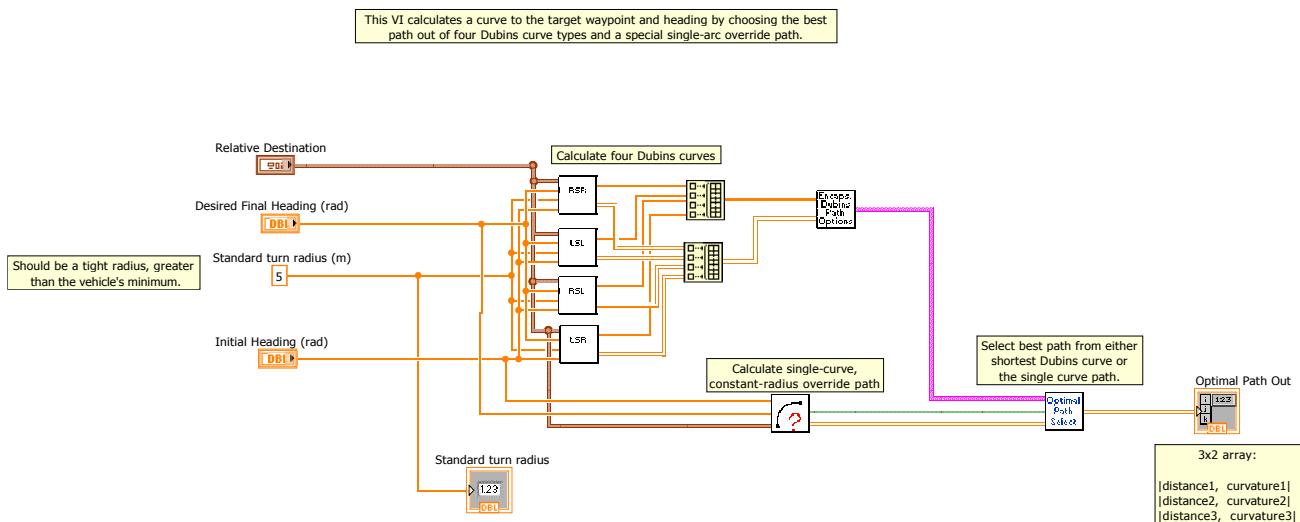
Connector Pane

PlanCurveToNextWaypoint.vi

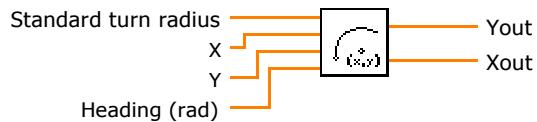
Front Panel



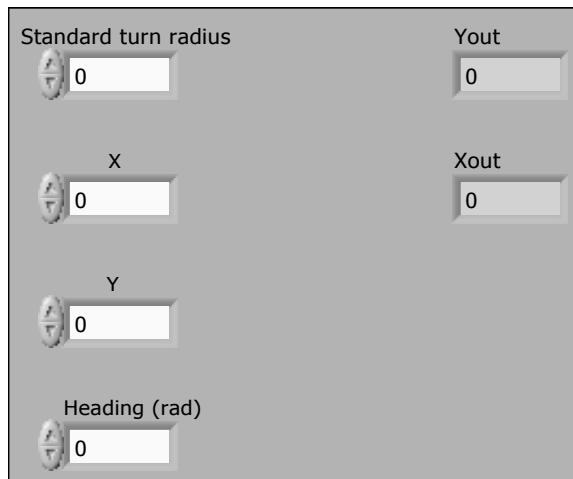
Block Diagram



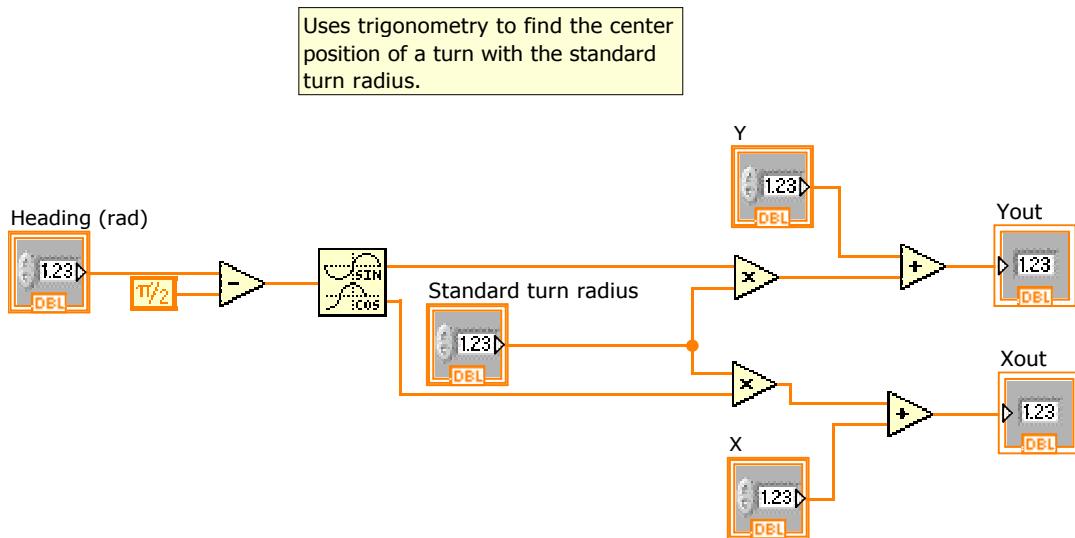
Connector Pane

GetRightTurnCenterPosition.vi

Front Panel



Block Diagram

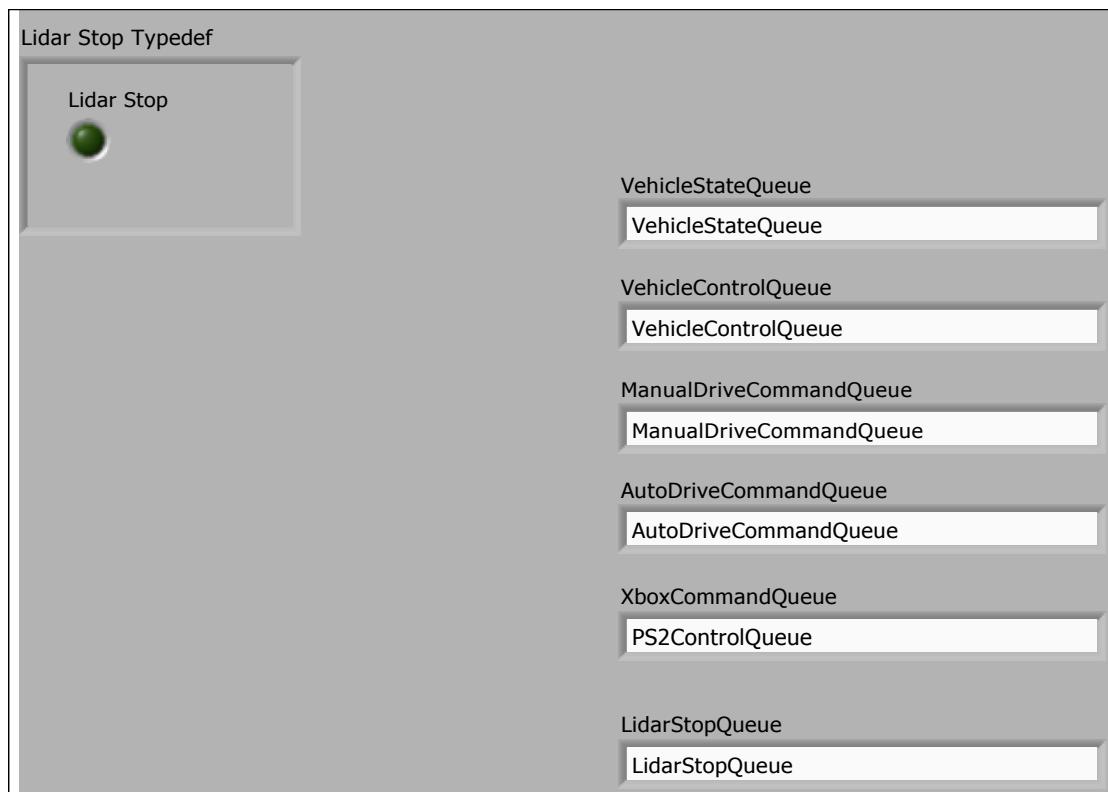


Connector Pane

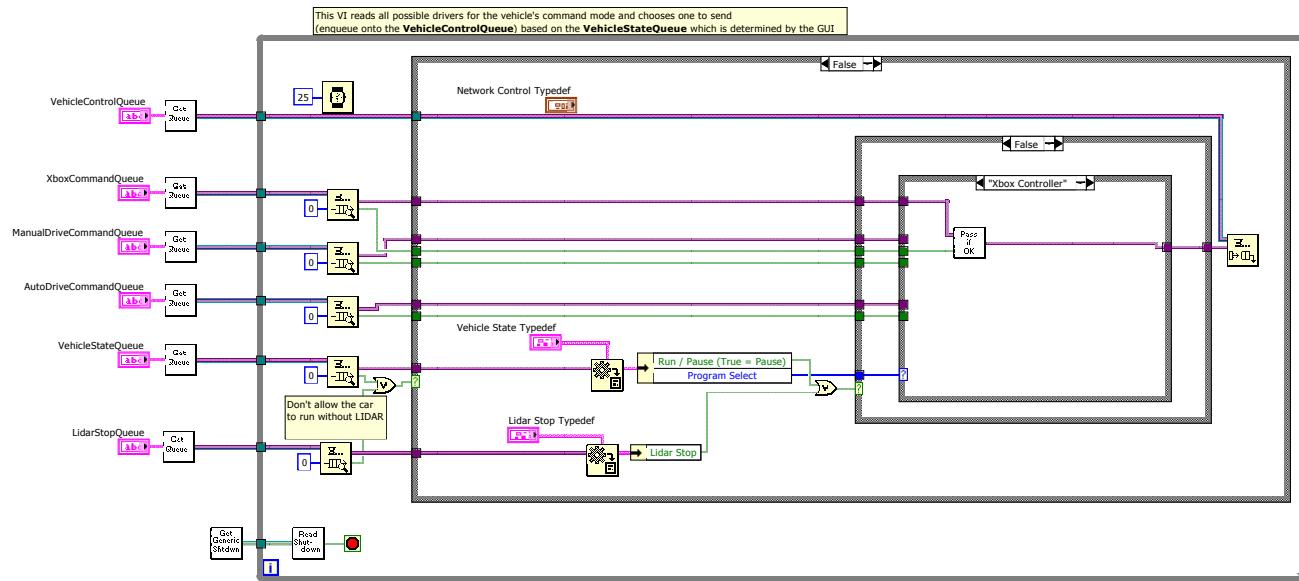
SelectDriveCommand.vi

This VI reads all possible drivers for the vehicle's command mode and chooses one to send (enqueue onto the VehicleControlQueue) based on the VehicleStateQueue which is determined by the GUI

Front Panel



Block Diagram



Connector Pane

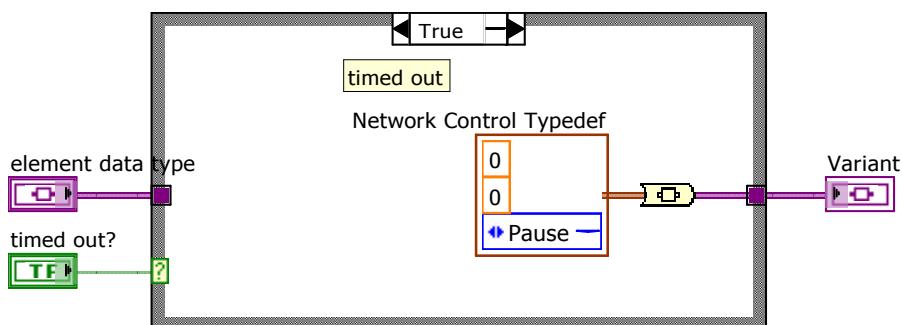
CommandQueuePassThroughIfNotTimedOut.vi

Pass through a command argument if the queue is not timed out.

Front Panel



Block Diagram



Connector Pane

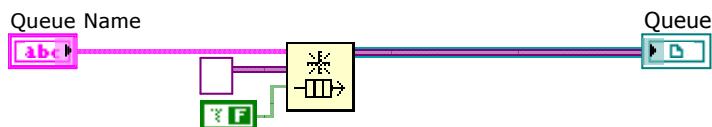
GetQueue.vi

Returns the reference to a queue of type variant with the given name.

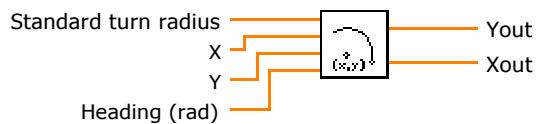
Front Panel



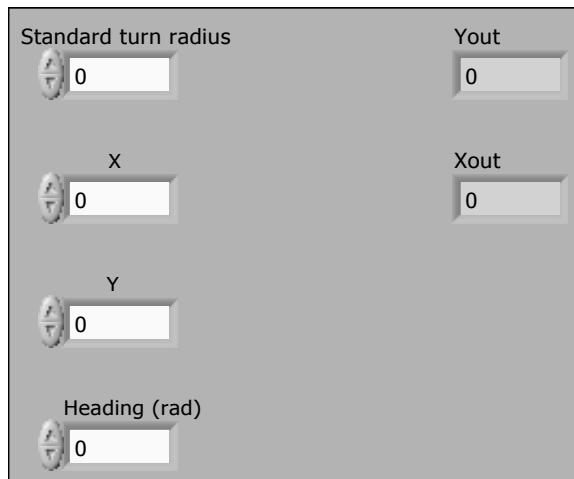
Block Diagram



Connector Pane

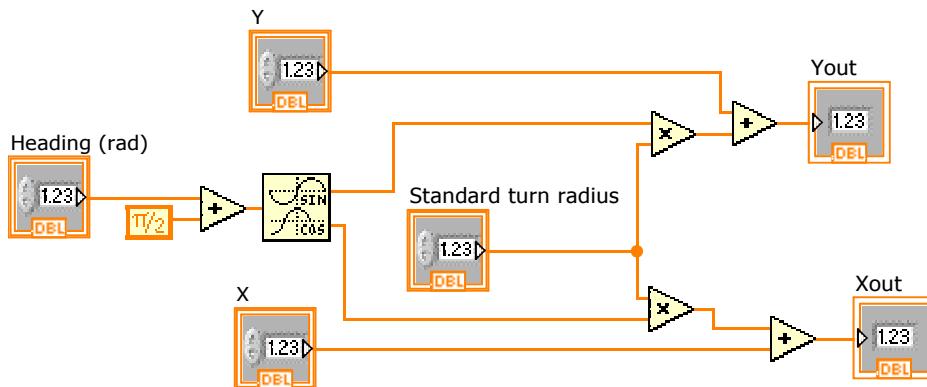
GetLeftTurnCenterPosition.vi

Front Panel



Block Diagram

Uses trigonometry to find the center position of a Left turn with the standard turn radius.

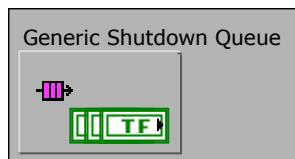


Connector Pane

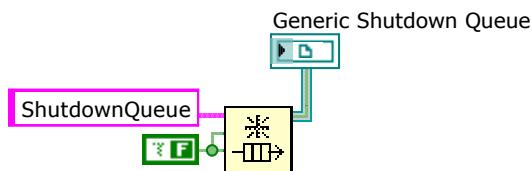
GetGenericShutdownQueue.vi

Returns a reference to the Generic Shutdown Queue (ShutdownQueue).

Front Panel



Block Diagram



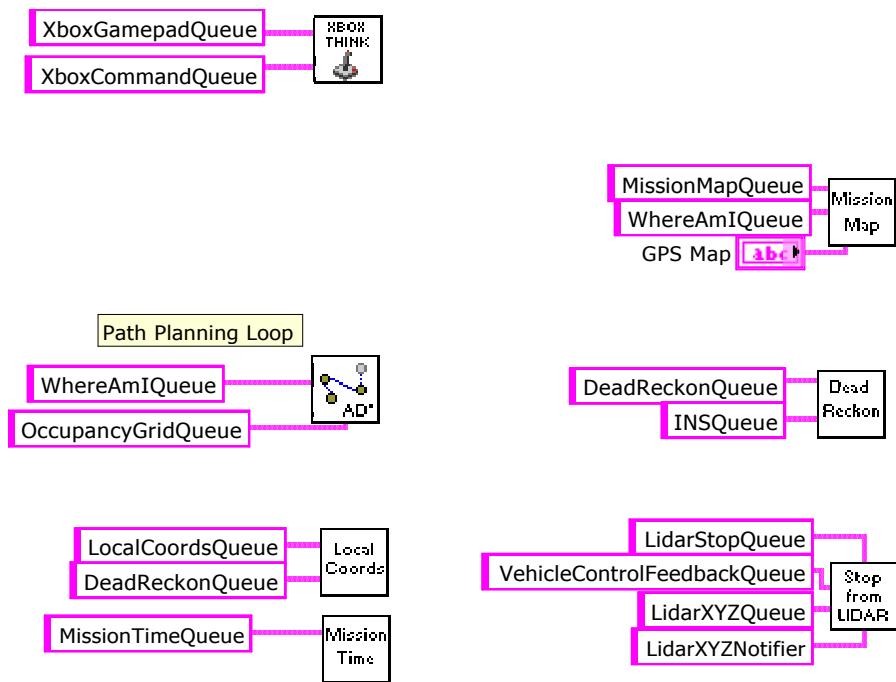
Connector Pane

ThinkMain.vi

Front Panel



Block Diagram

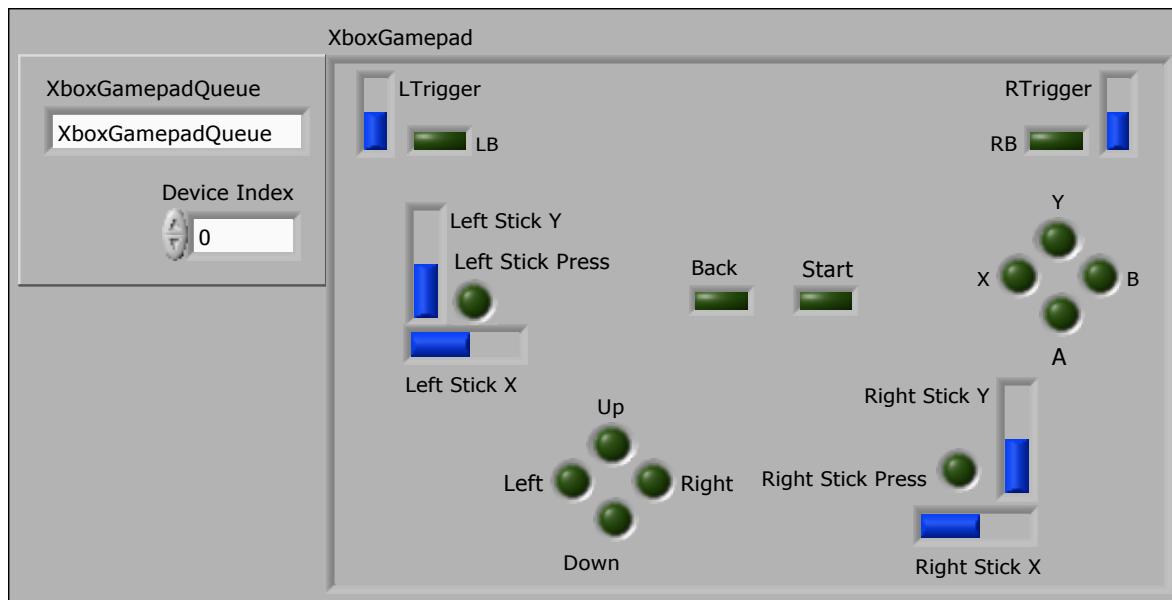


Connector Pane

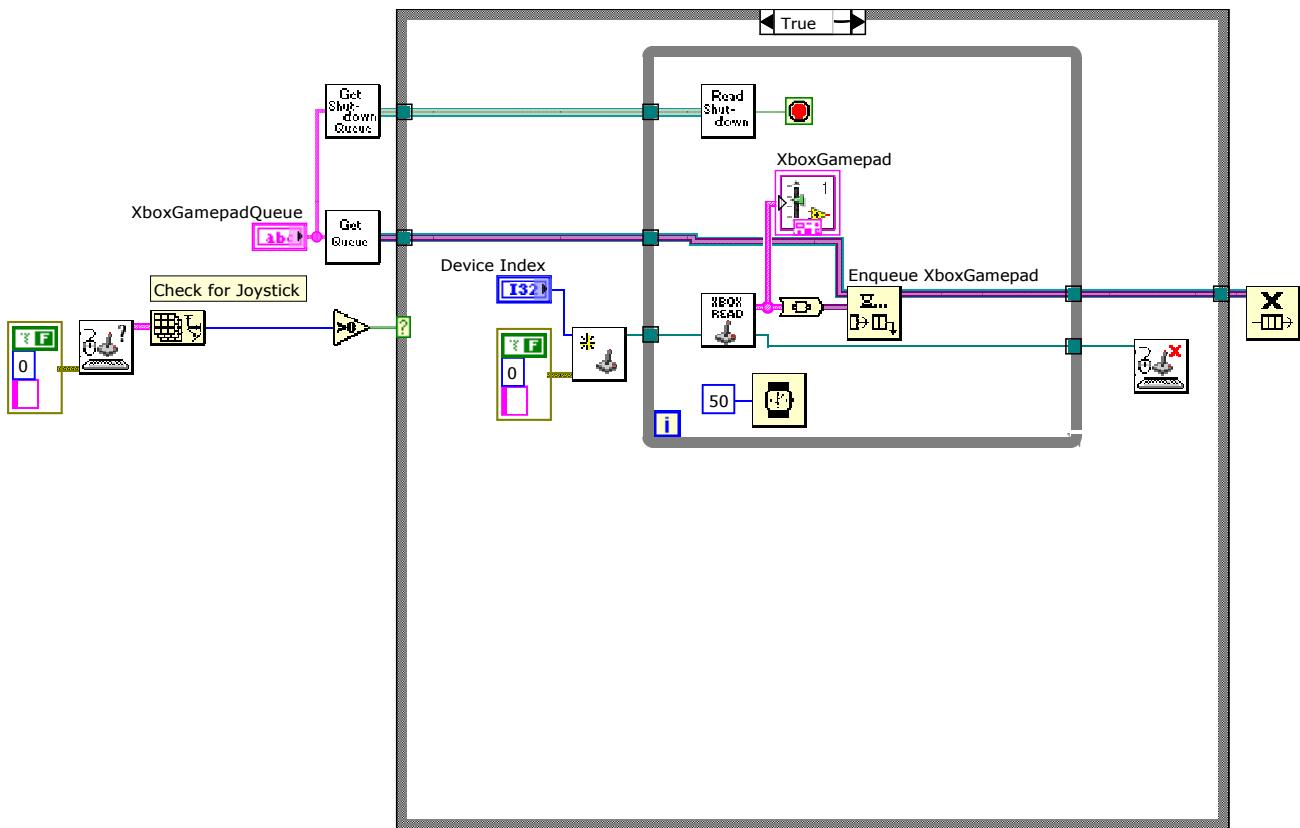
XboxGamepad.vi

Read an Xbox controller and enqueue the data.

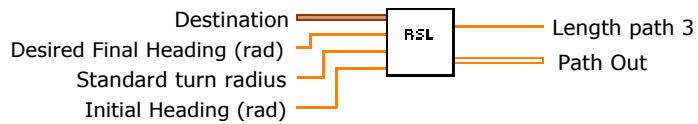
Front Panel



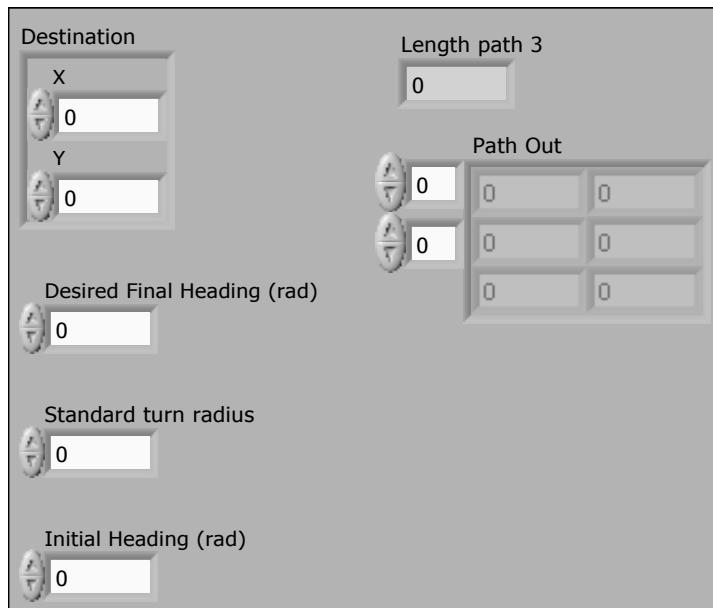
Block Diagram



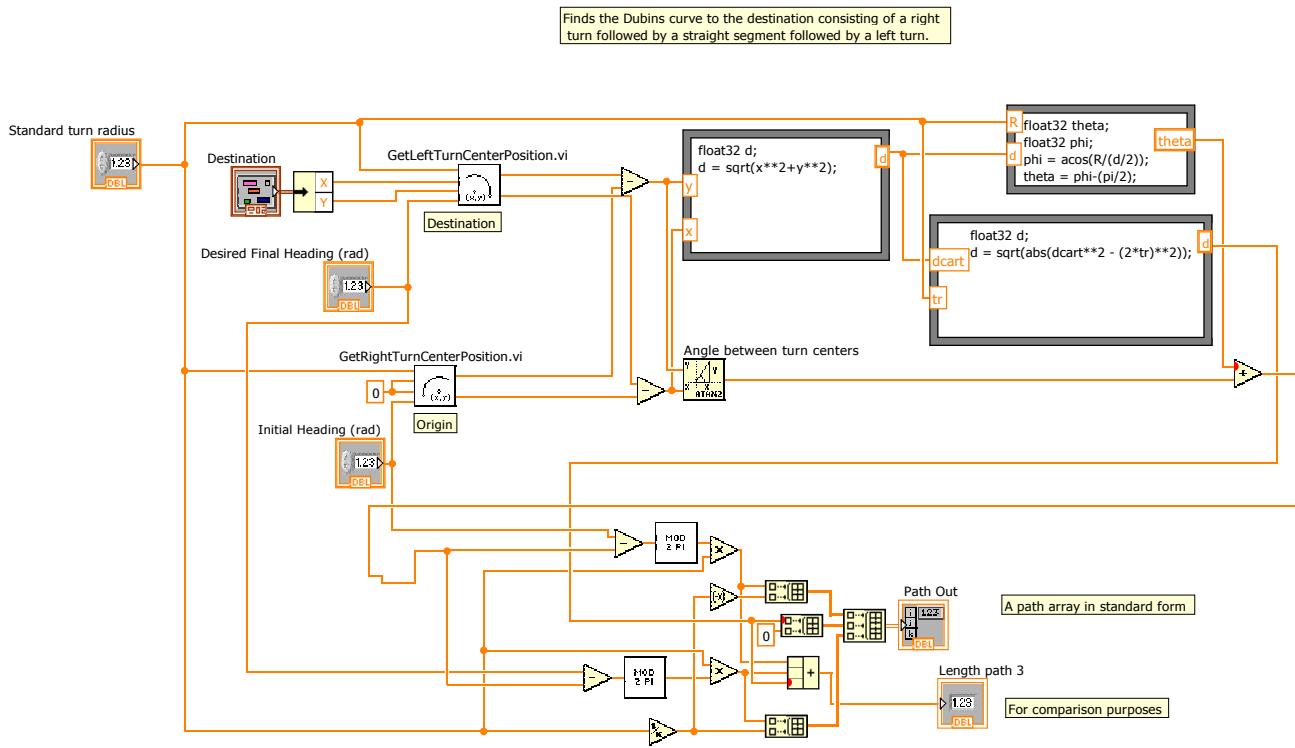
Connector Pane

FindRSLPath.vi

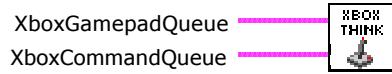
Front Panel



Block Diagram

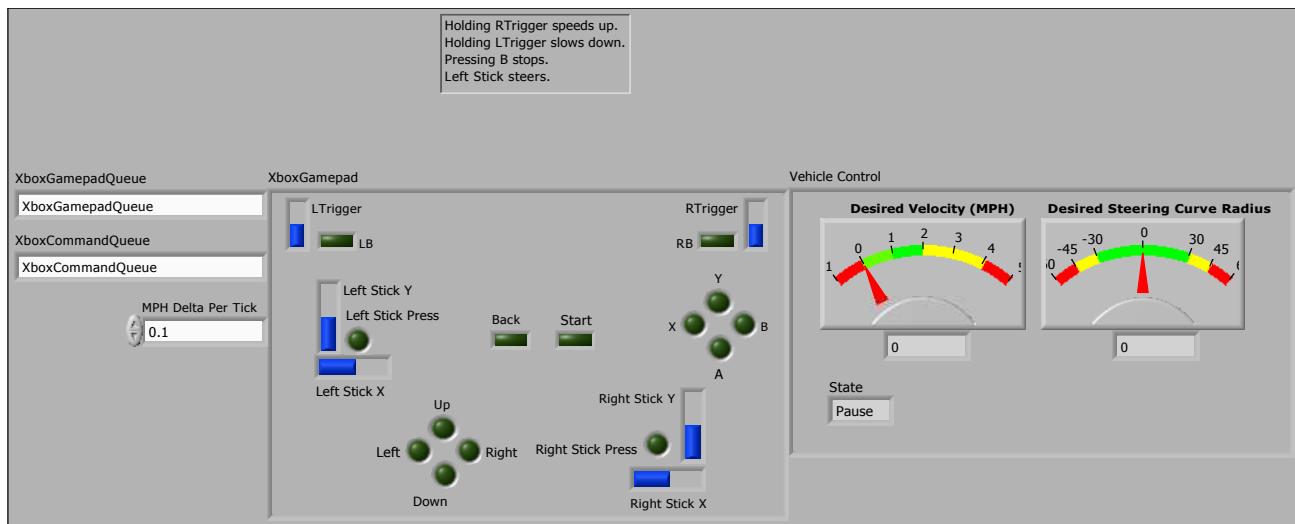


Connector Pane

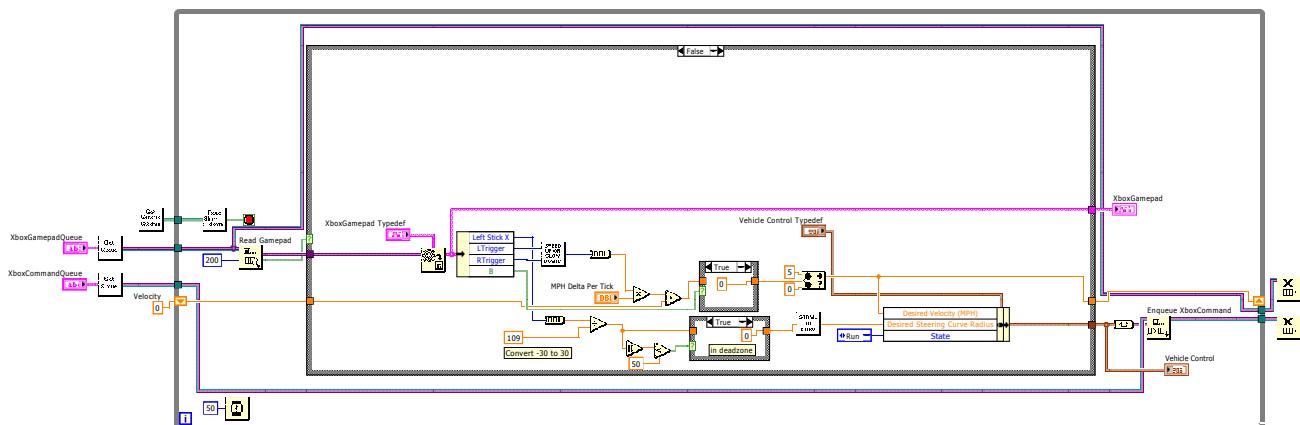
XboxGamepadThink.vi

Reads an Xbox's value on a queue and enqueues an Xbox control value for driving the vehicle.

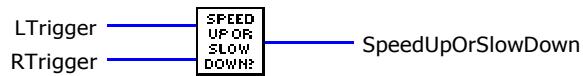
Front Panel



Block Diagram

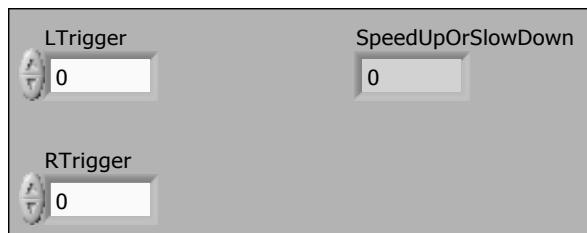


Connector Pane

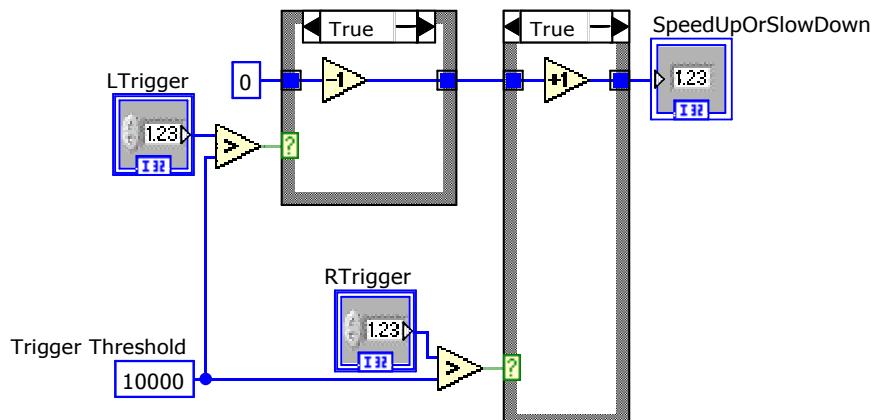
SpeedUpOrSlowDown.vi

Determines if the vehicle should speed up or slow down based on the Xbox controller's left and right trigger values.

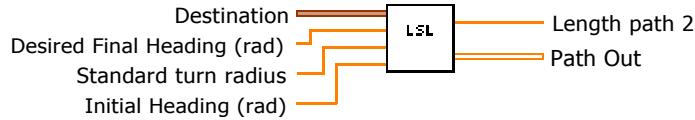
Front Panel



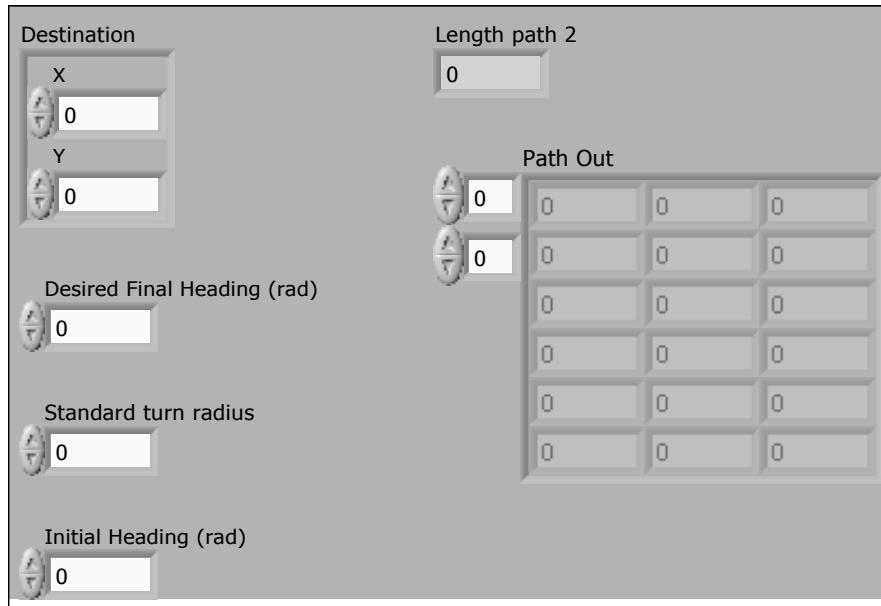
Block Diagram



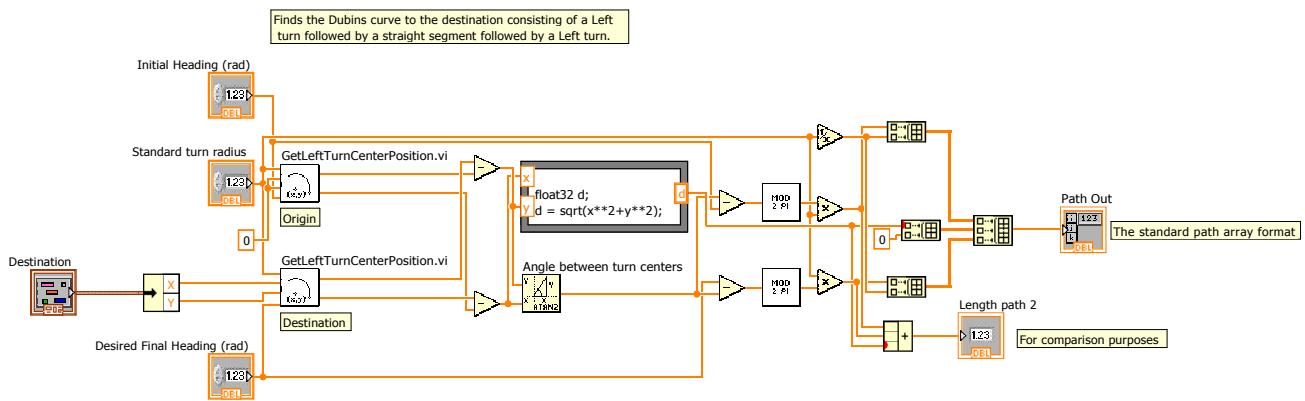
Connector Pane

FindLSLPath.vi

Front Panel



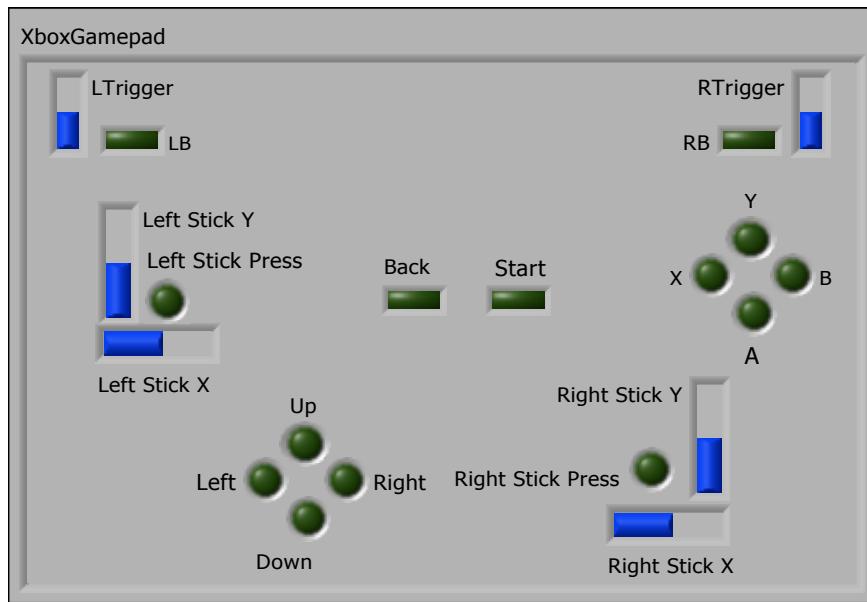
Block Diagram



Connector Pane

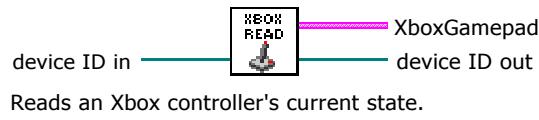
XboxGamepad.ctl

Front Panel

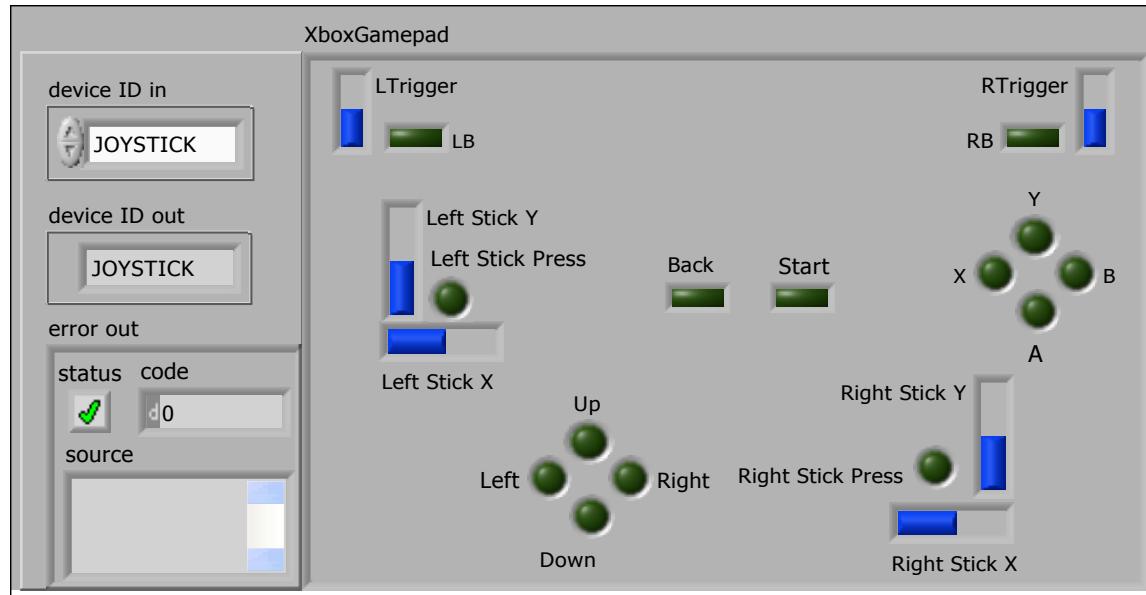


Block Diagram

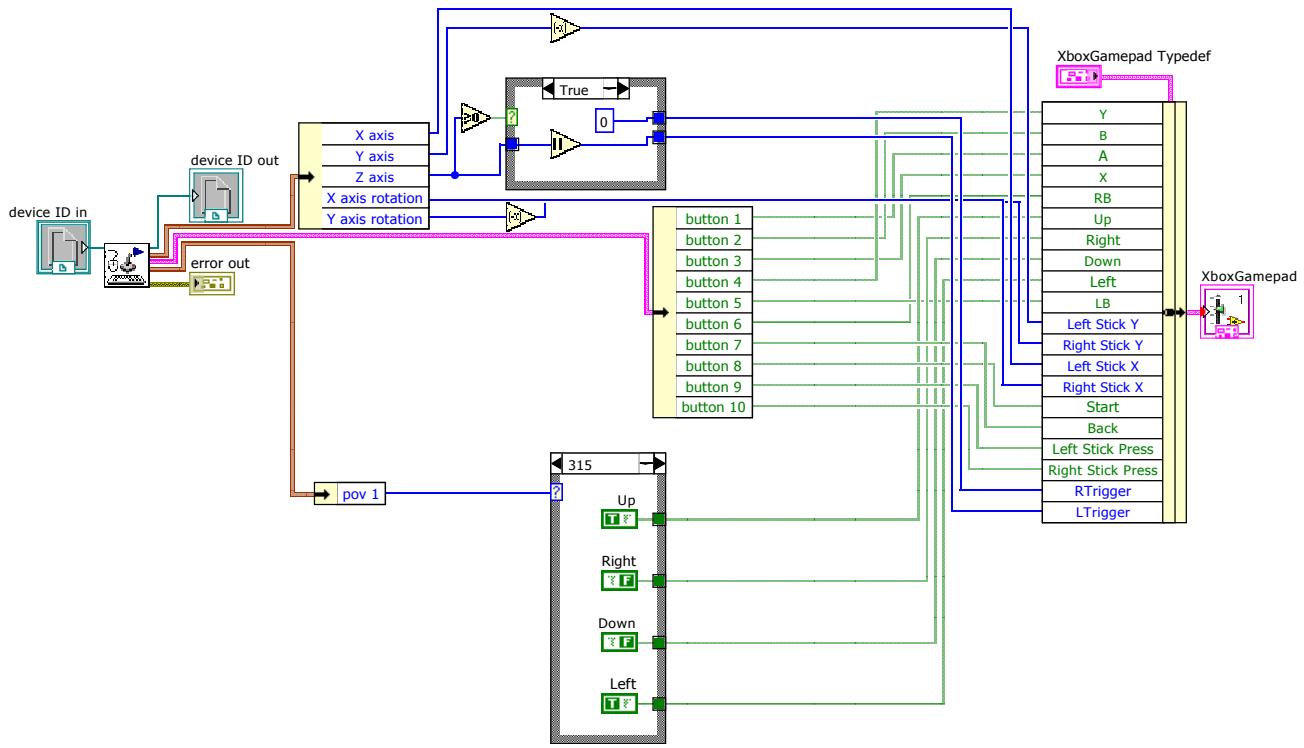
Connector Pane

XboxGamepadRead.vi

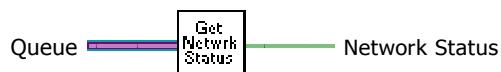
Front Panel



Block Diagram

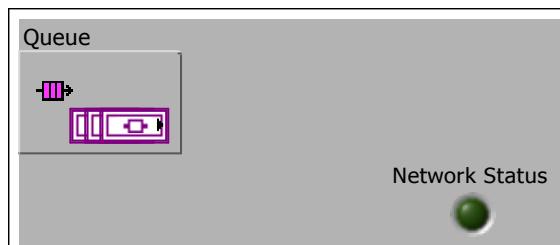


Connector Pane

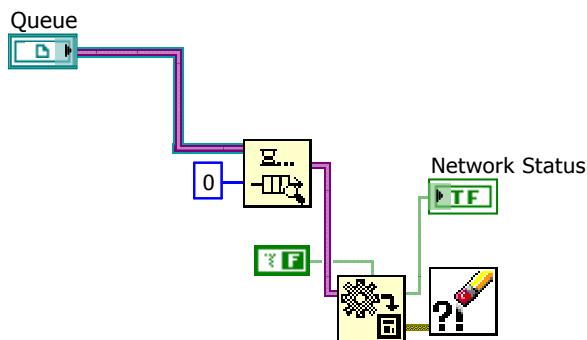
GetNetworkStatus.vi

Read a network queue and determine if it is online.

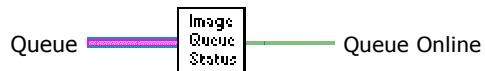
Front Panel



Block Diagram

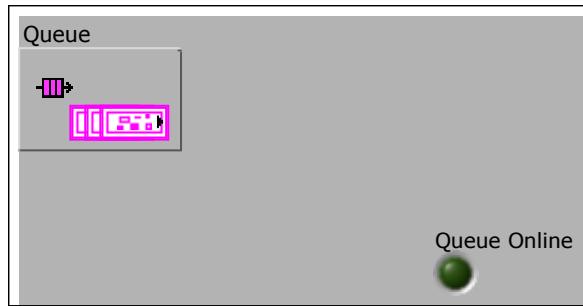


Connector Pane

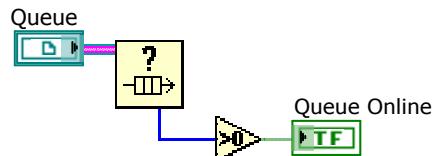
ImageQueueStatus.vi

Determines if an Image Queue is online (has data in it).

Front Panel



Block Diagram



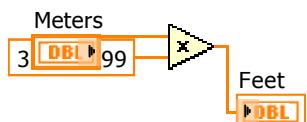
Connector Pane

meters to feet.vi

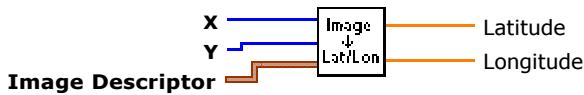
Front Panel



Block Diagram

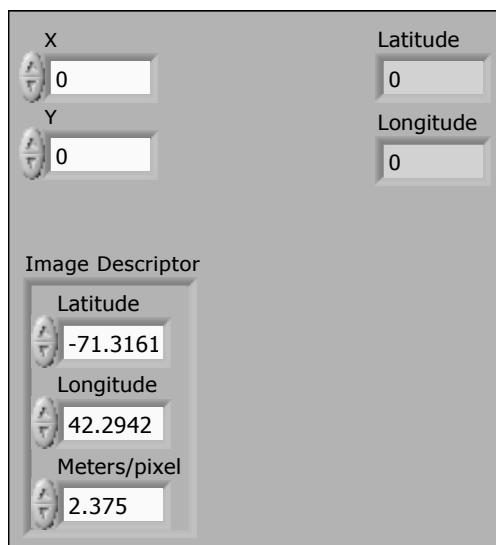


Connector Pane

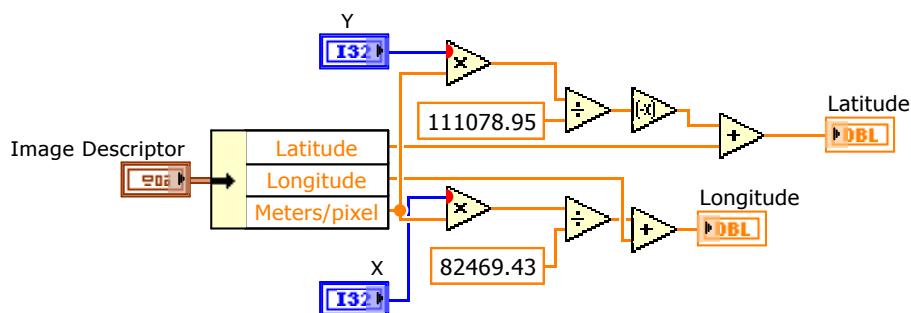
Image to Lat Lon.vi**Image Descriptor**

Converts image pixels to Lat/Lon coordinates using the Image Descriptor.

Front Panel



Block Diagram



Connector Pane

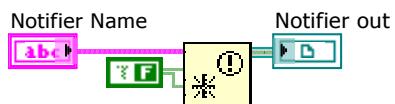
GetNotifier.vi

Returns a notifier with the given name and a boolean as the type.

Front Panel



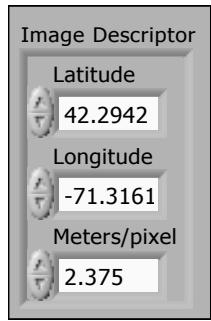
Block Diagram



Connector Pane

Image Descriptor.ctl

Front Panel

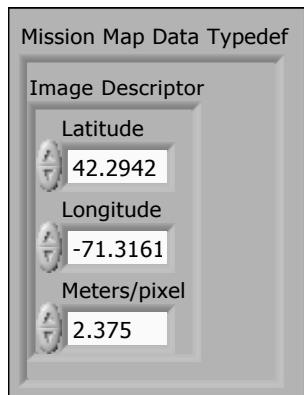


Block Diagram

Connector Pane

MissionMapDataControl.ctl

Front Panel

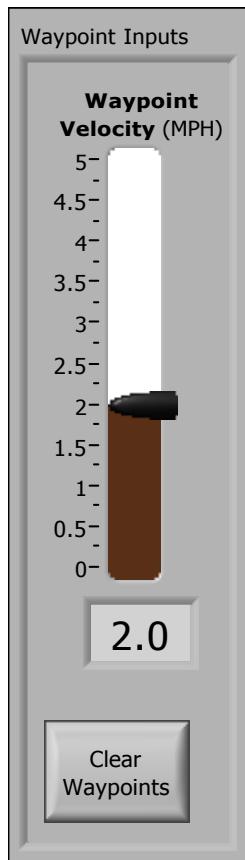


Block Diagram

Connector Pane

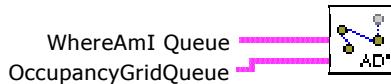
WaypointInputControl.ctl

Front Panel

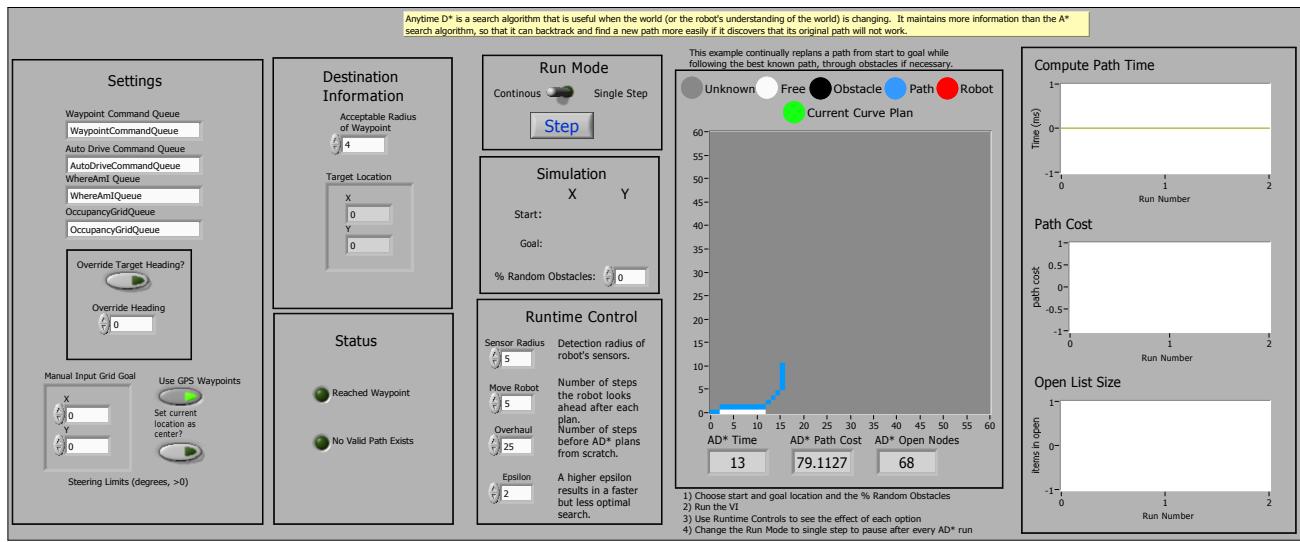


Block Diagram

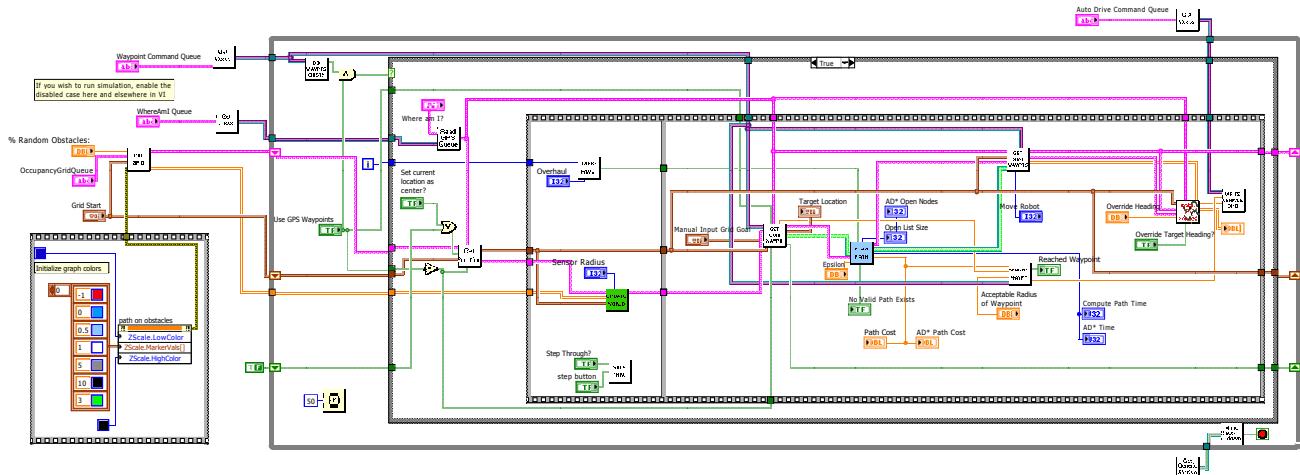
Connector Pane

Path Planning Loop.vi

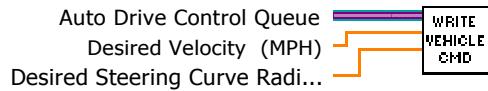
Front Panel



Block Diagram

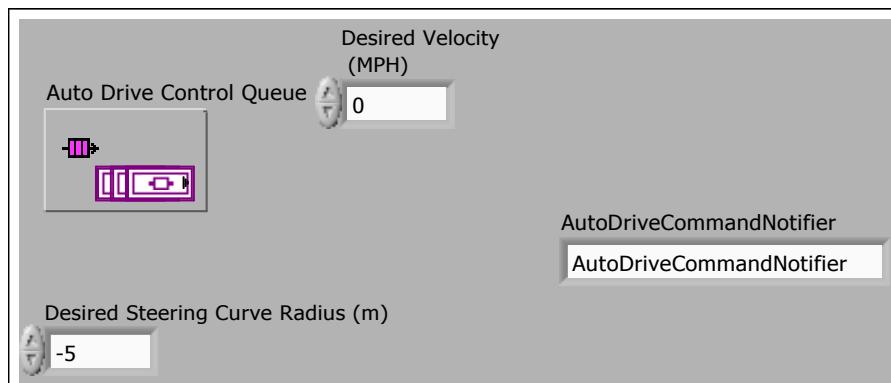


Connector Pane

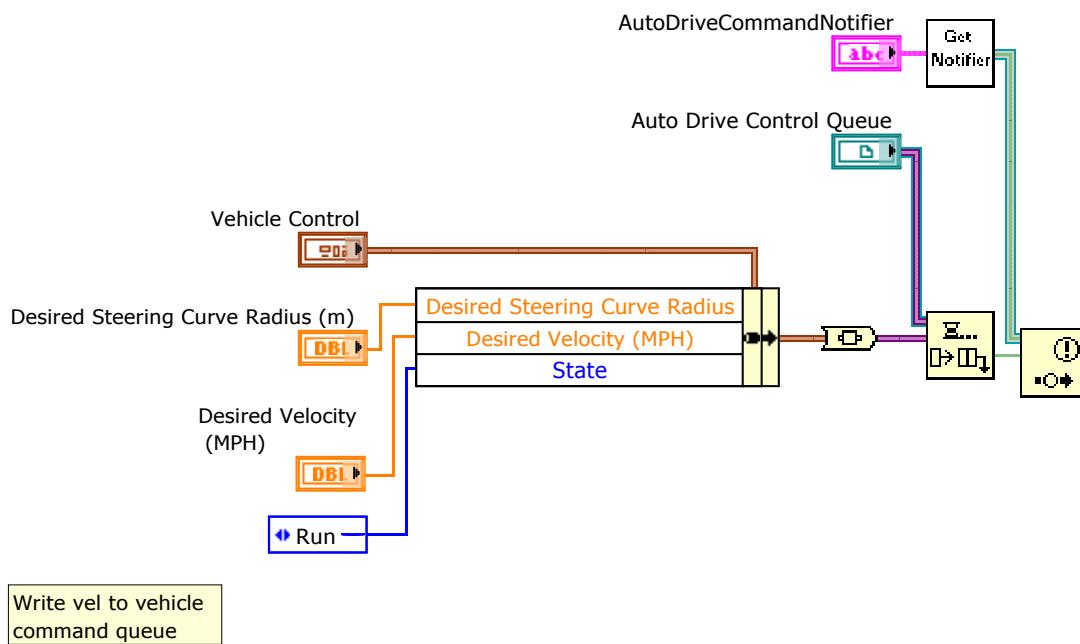
Write Vehicle Command.vi

Writes a command to the Vehicle Control Queue

Front Panel



Block Diagram

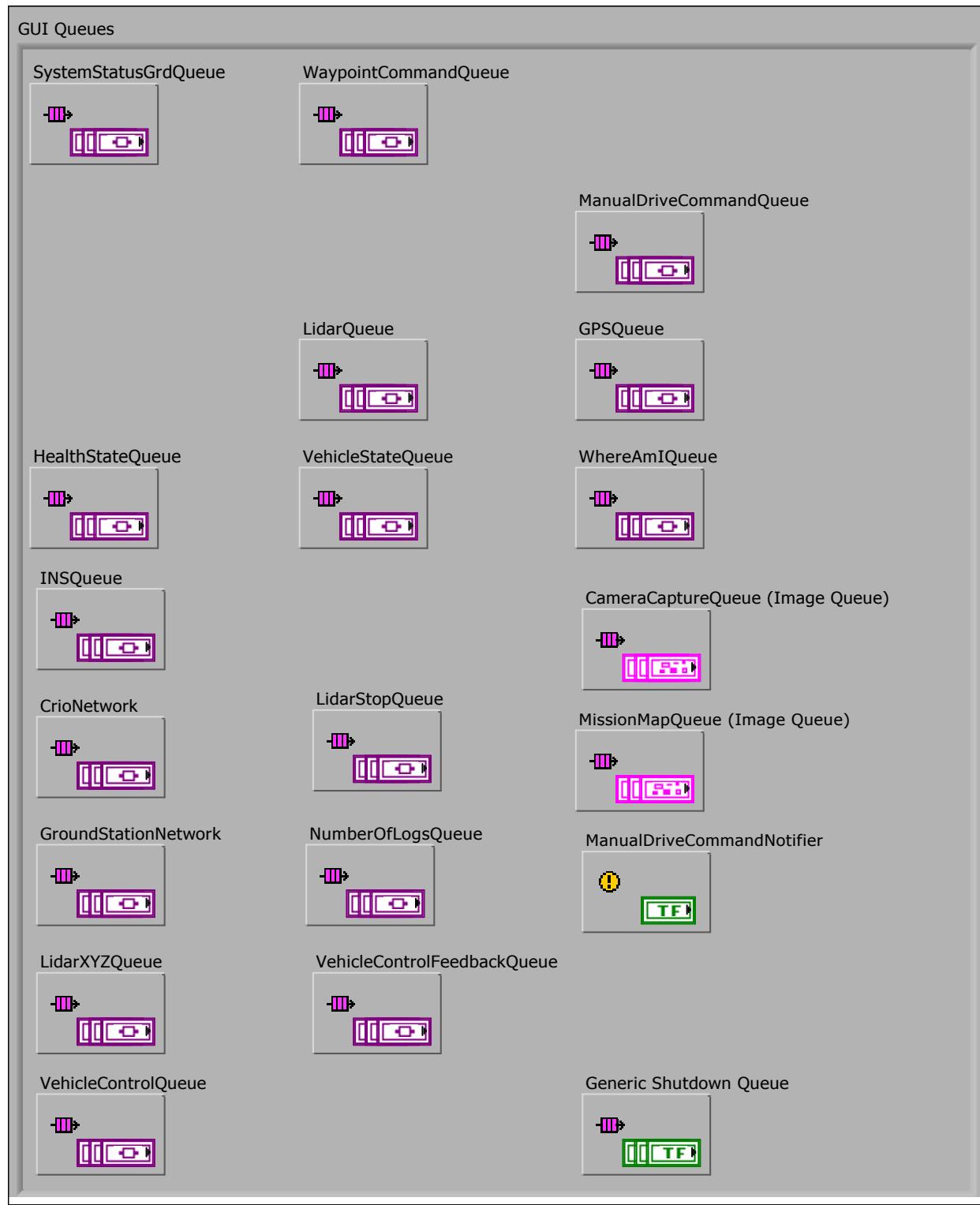


Connector Pane

GUIQueuesControl.ctl

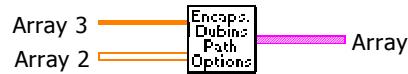


Front Panel

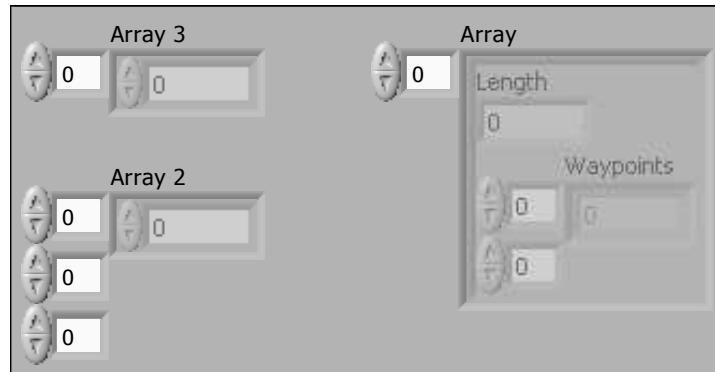


Block Diagram

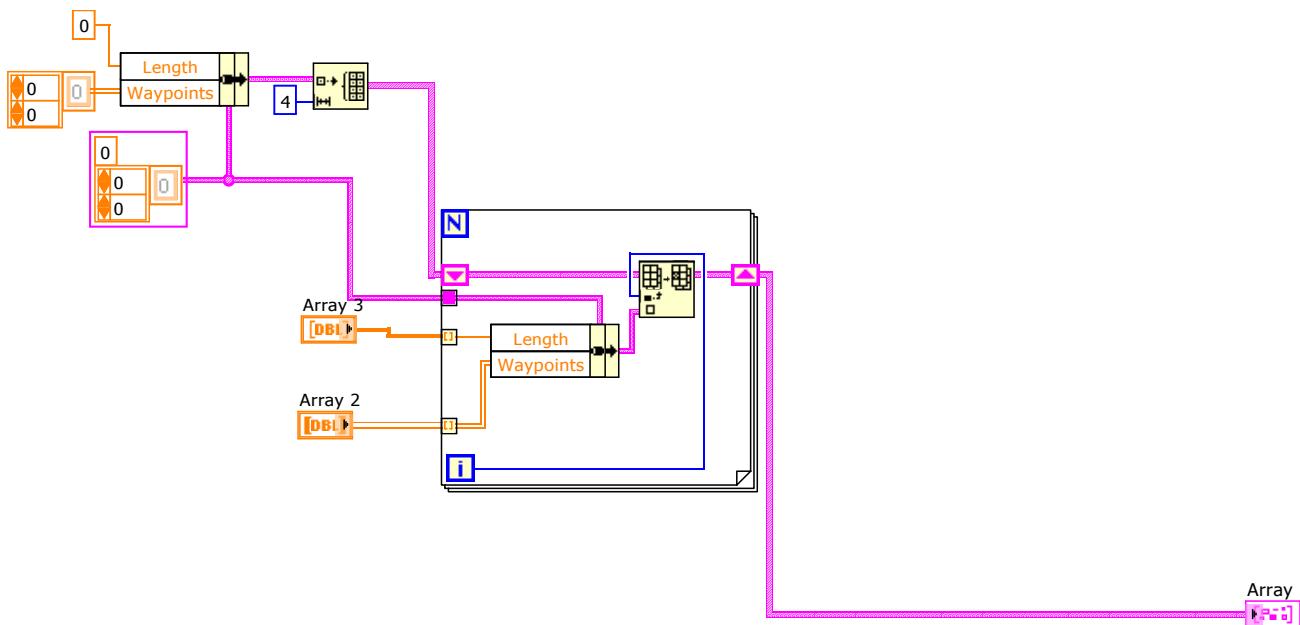
Connector Pane

Encapsulate Path Options.vi

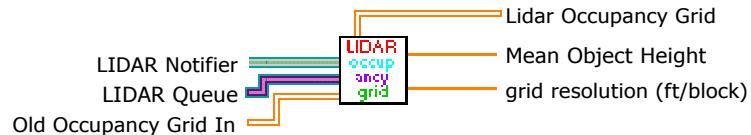
Front Panel



Block Diagram

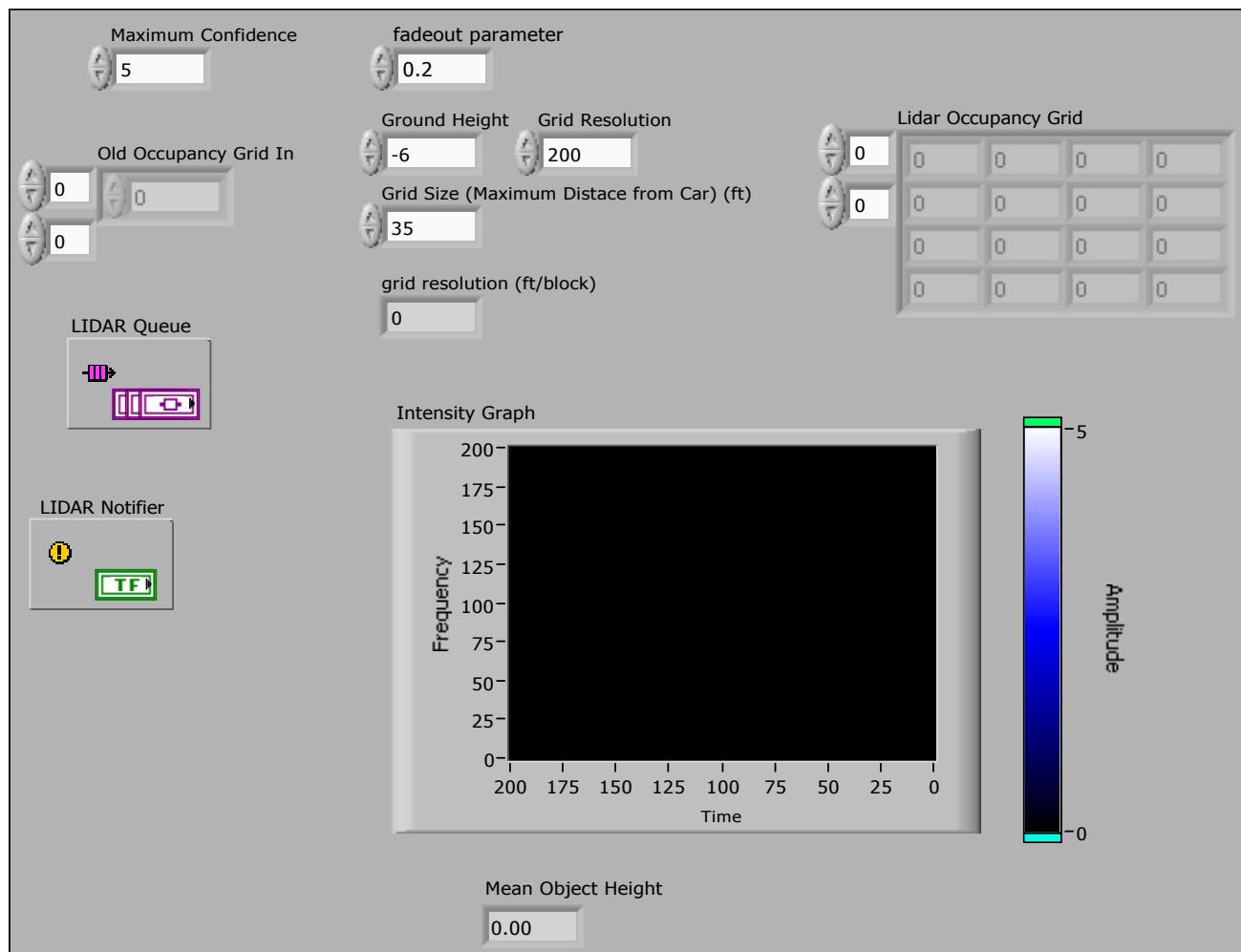


Connector Pane

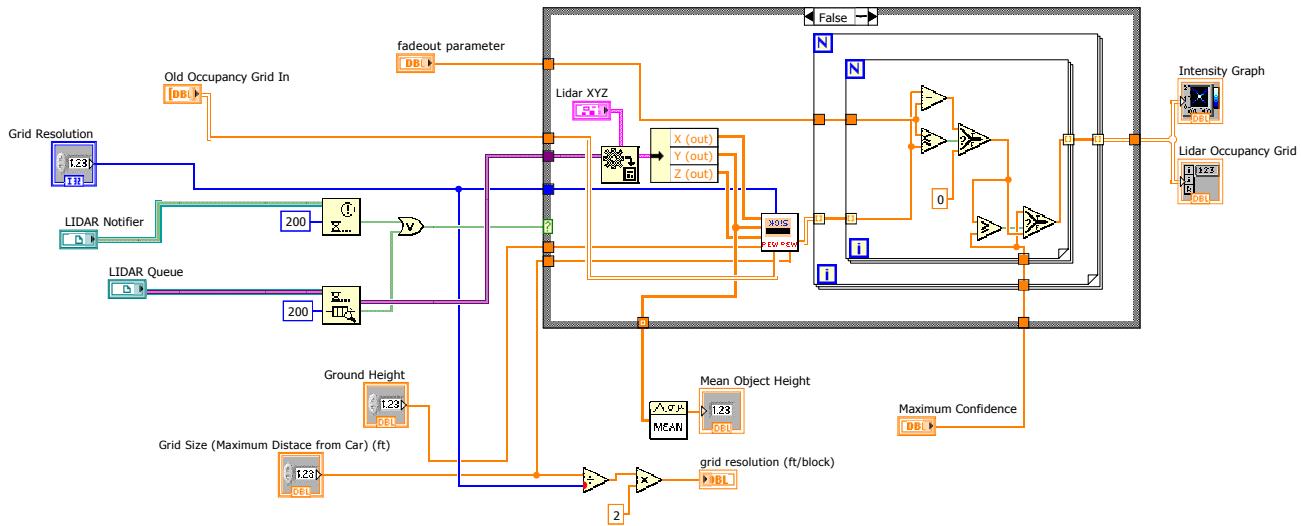
LidarOccupancyWithMemory.vi

Read LIDAR data and update the occupancy grid.

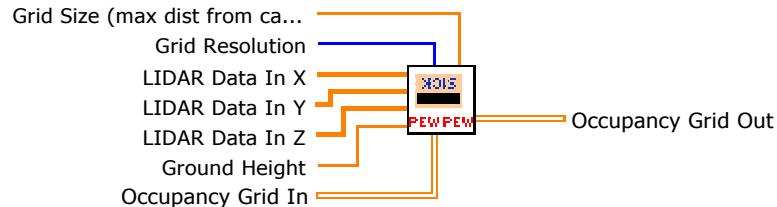
Front Panel



Block Diagram

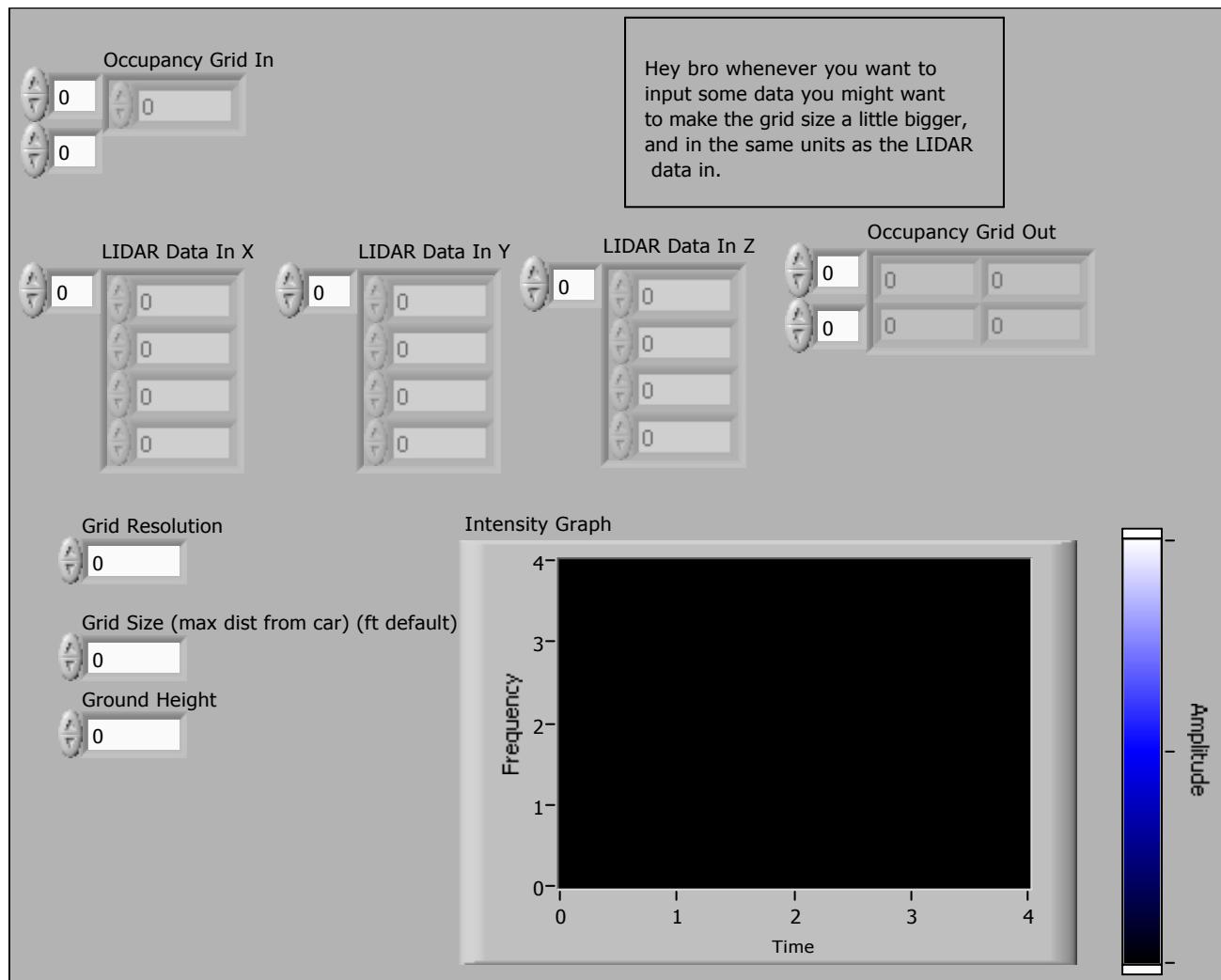


Connector Pane

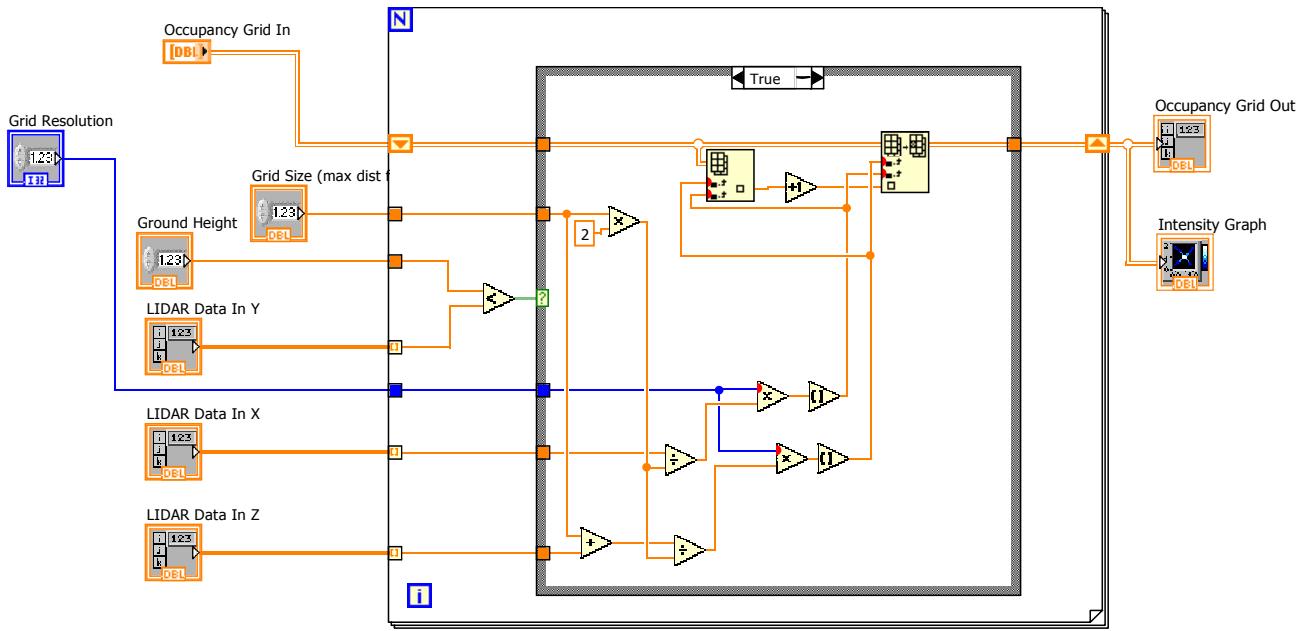
LIDARtoOccupancyGrid.vi

Update an occupancy grid with new LIDAR data.

Front Panel



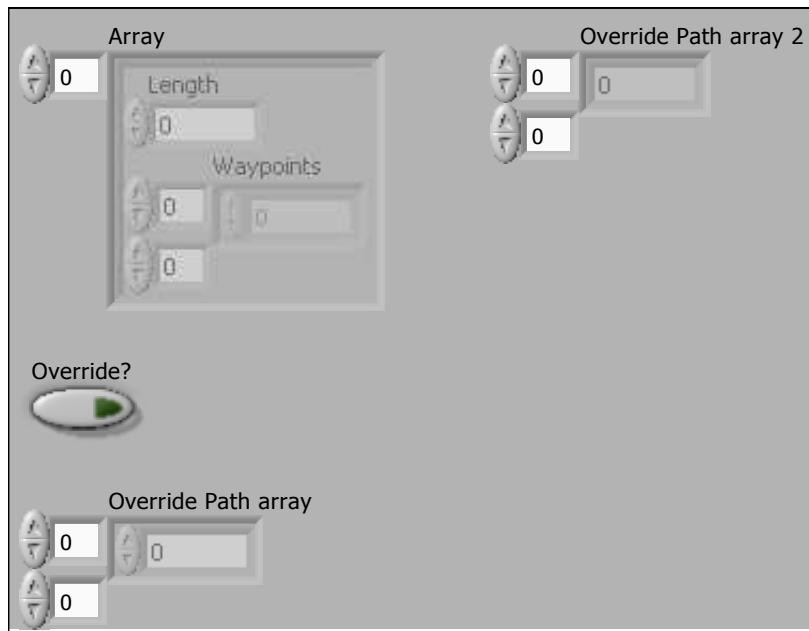
Block Diagram



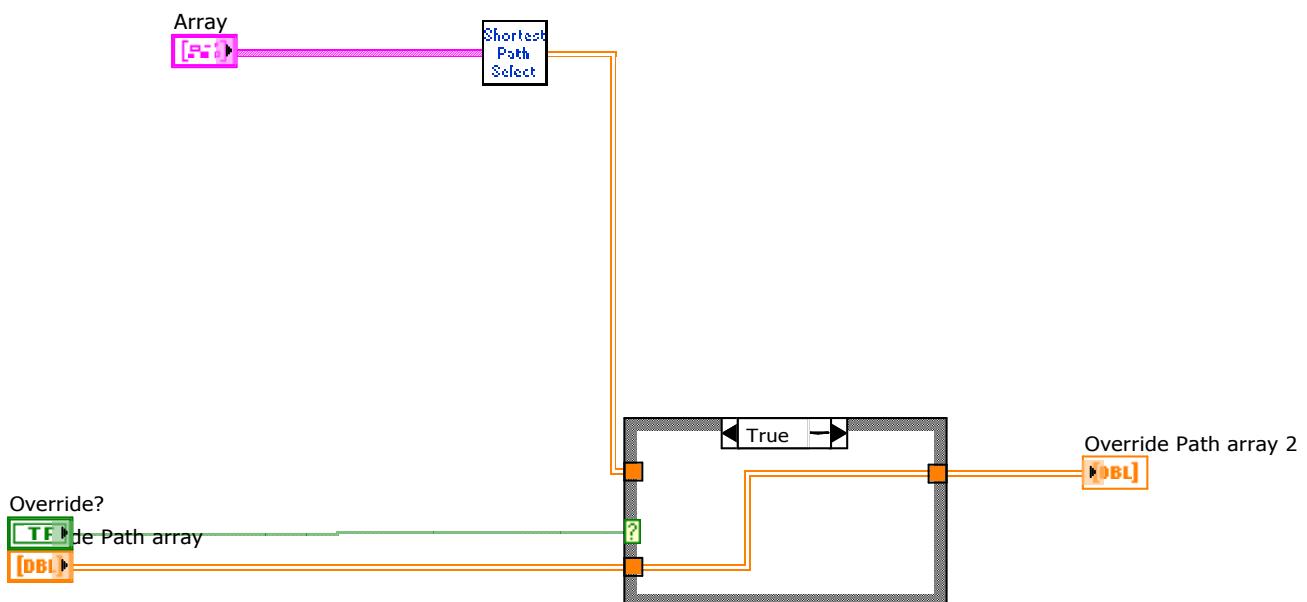
Connector Pane

OptimalPathSelect.vi

Front Panel



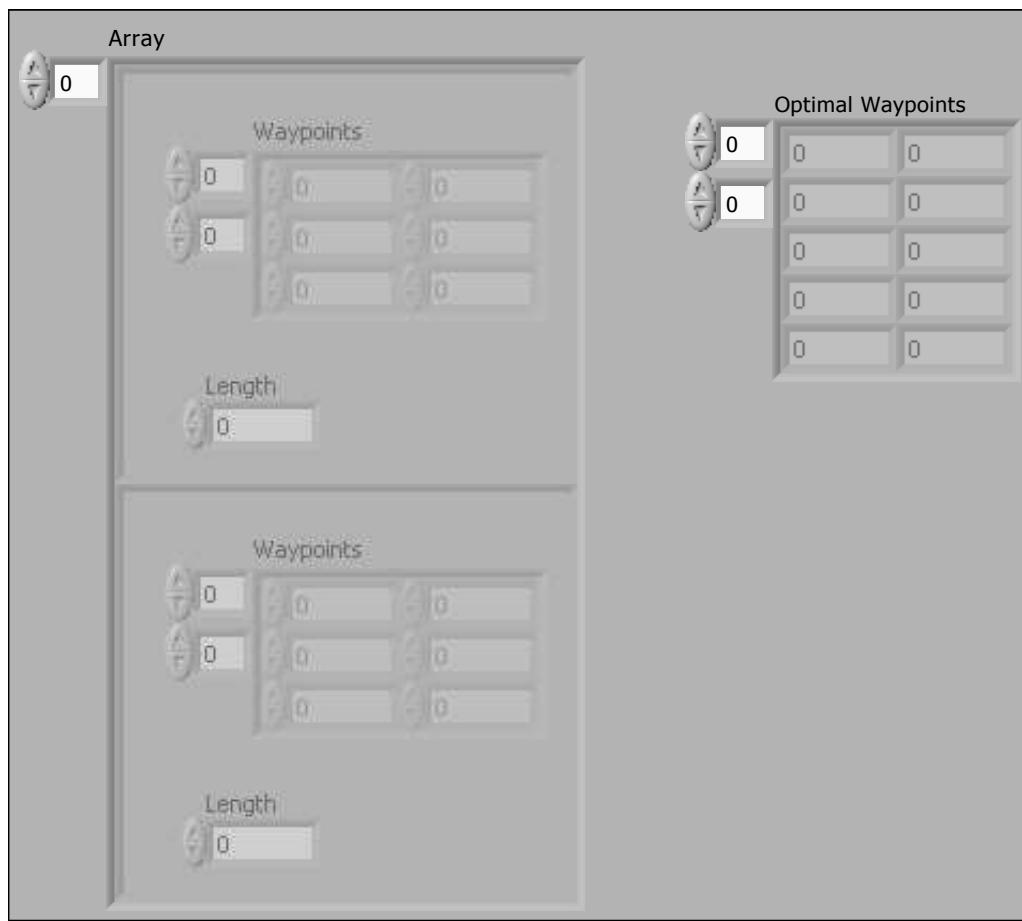
Block Diagram



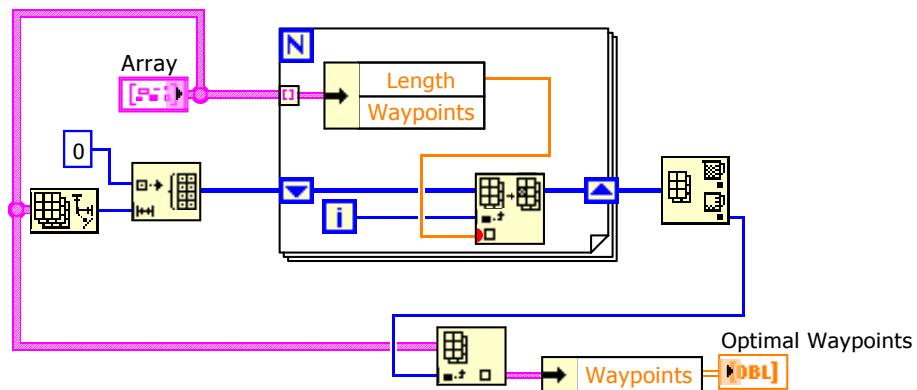
Connector Pane

PathSelector.vi

Front Panel



Block Diagram

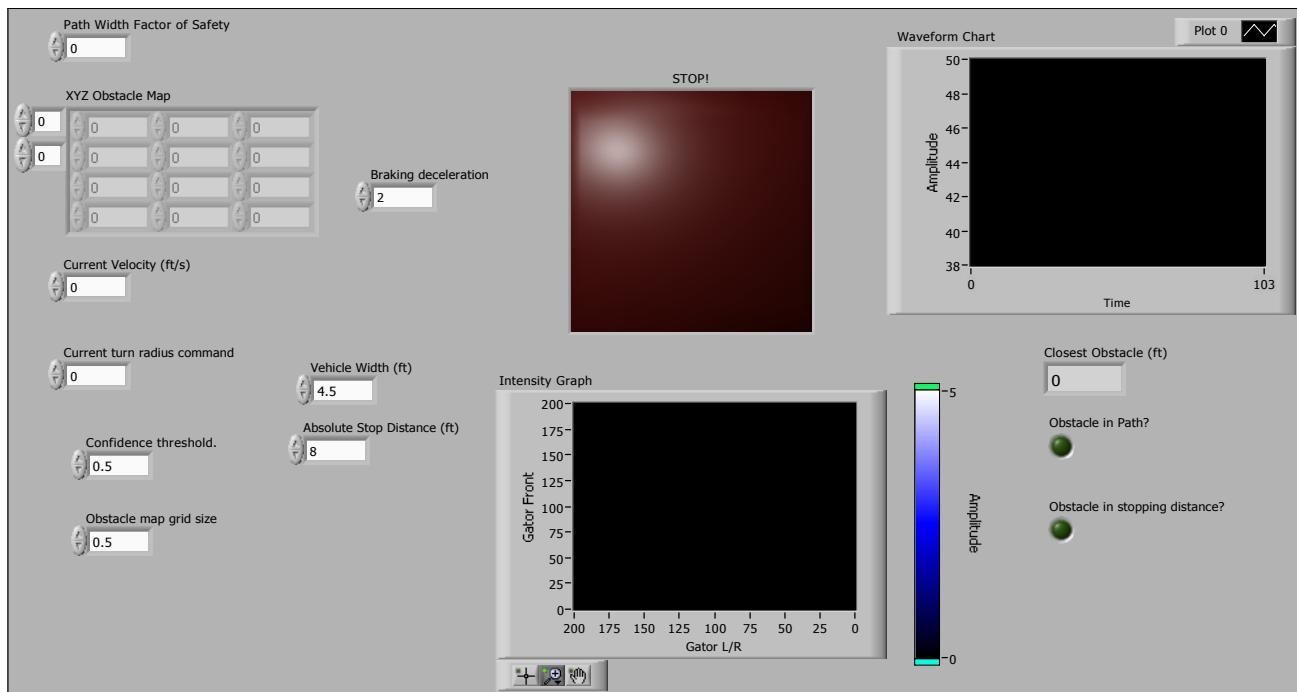


Connector Pane

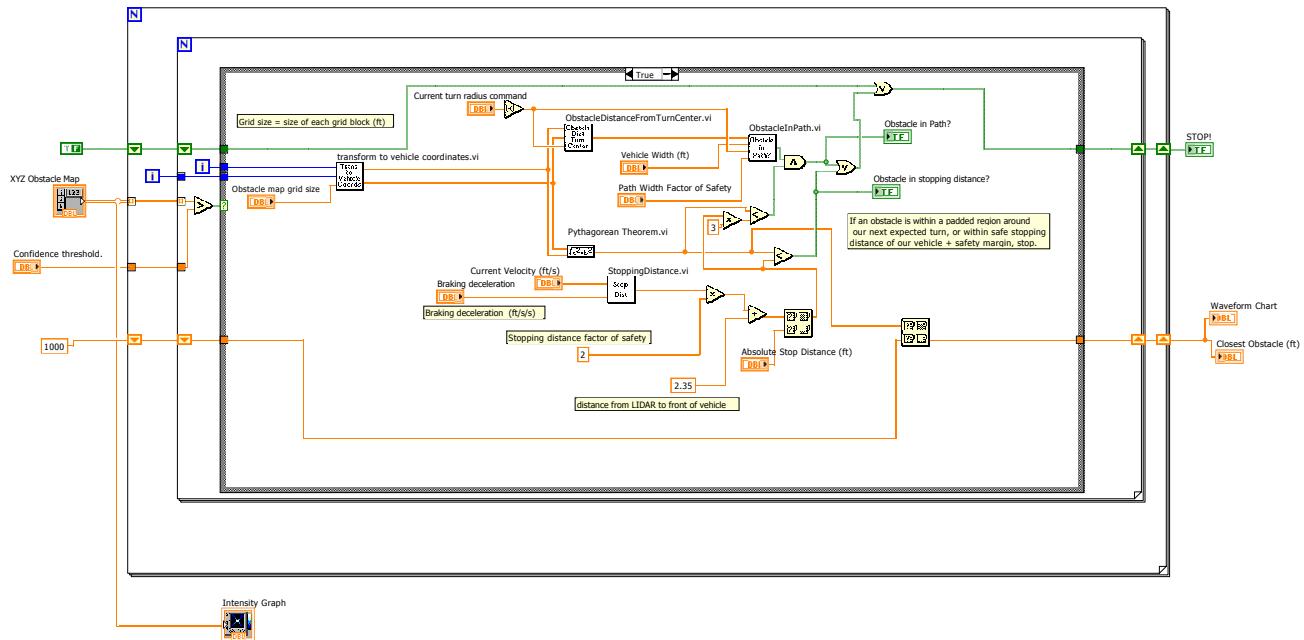
XYZtoStopCommand.vi

Use XYZ LIDAR data to determine if it is safe to move.

Front Panel



Block Diagram

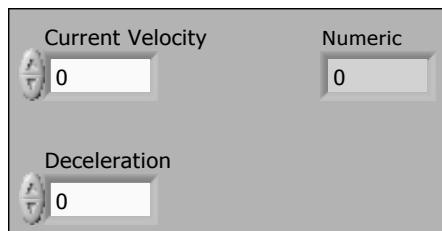


Connector Pane

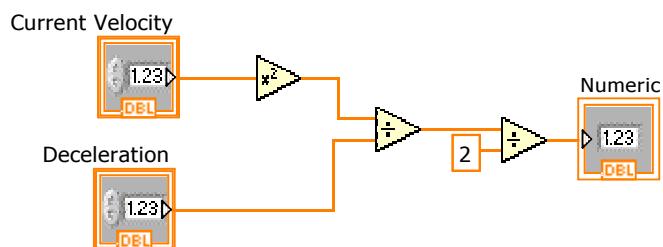
StoppingDistance.vi

Compute the stopping distance for the vehicle given the current velocity.

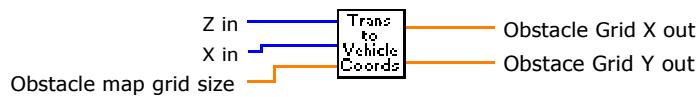
Front Panel



Block Diagram

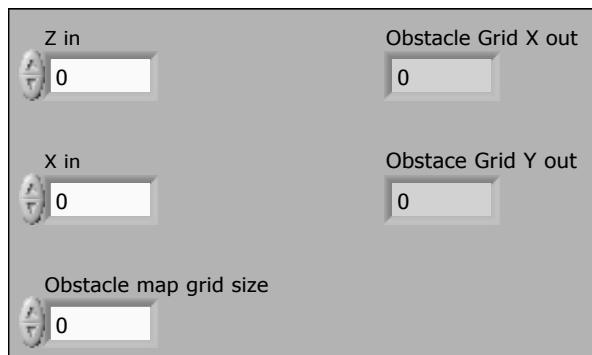


Connector Pane

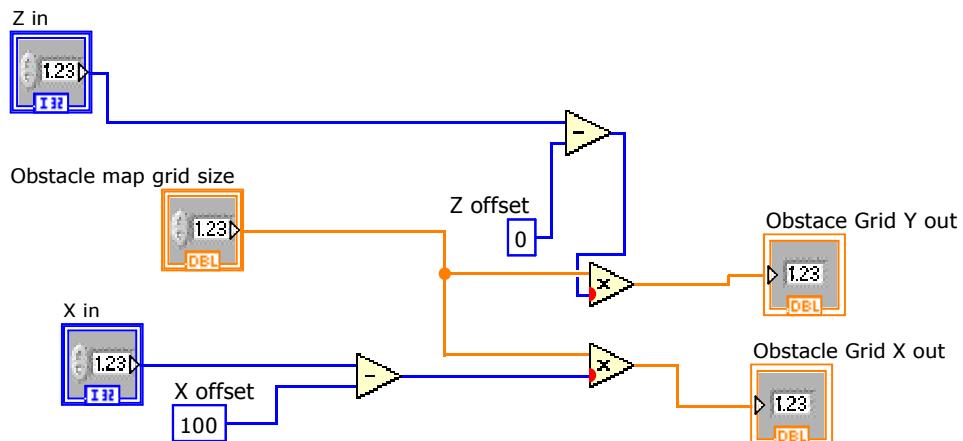
transform to vehicle coordinates.vi

Transform obstacles on the grid to vehicle coordinates.

Front Panel



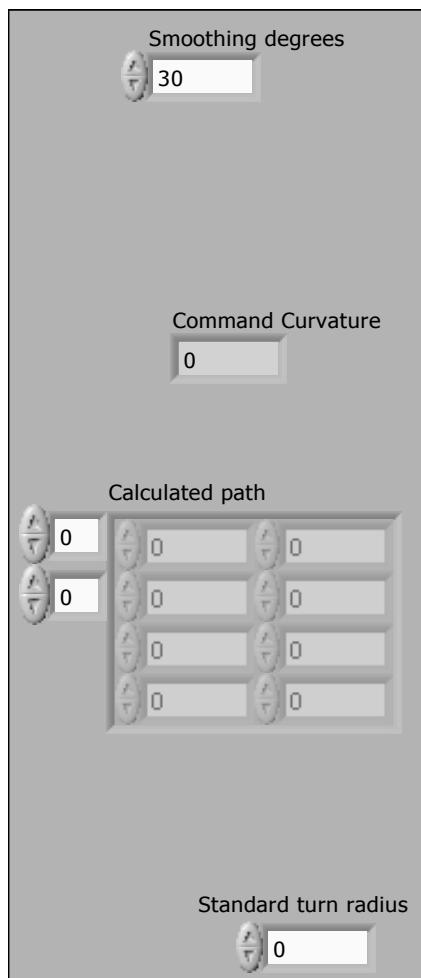
Block Diagram



Connector Pane

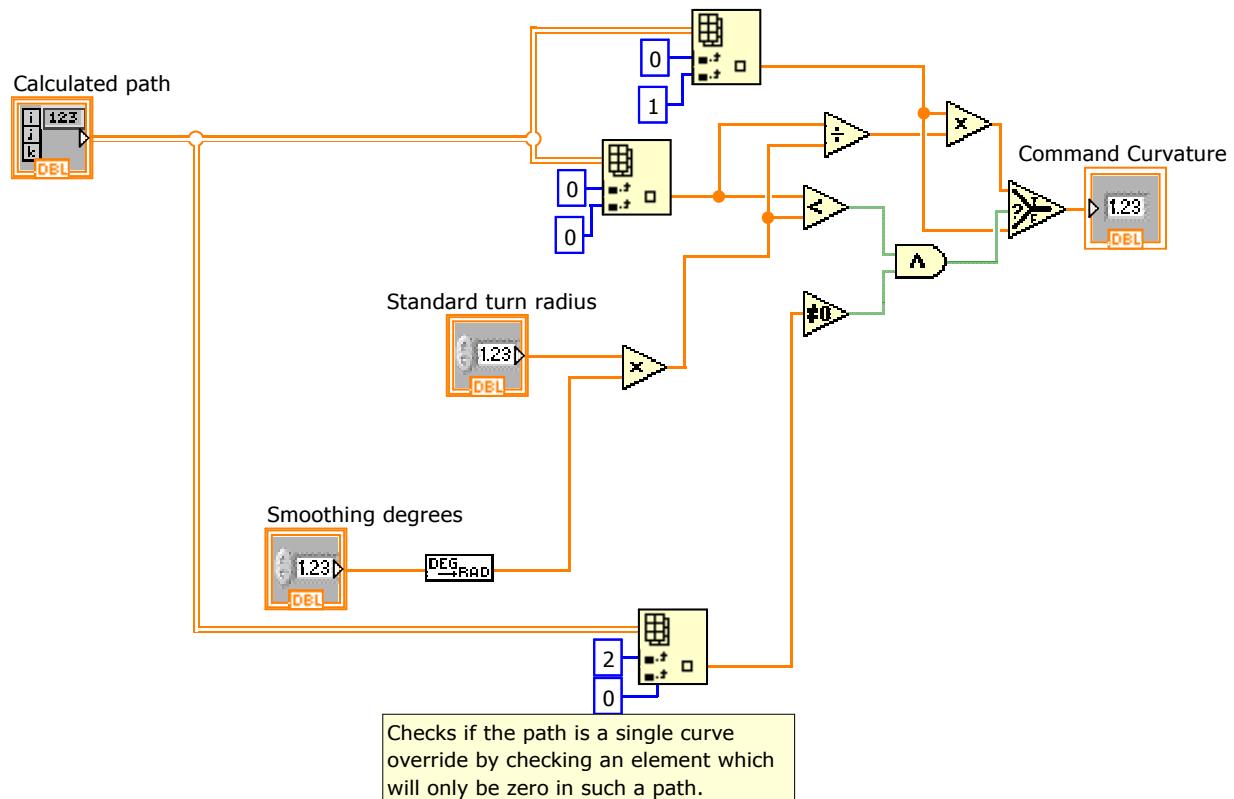
CurvatureOutputCalculator.vi

Front Panel



Block Diagram

This VI calculates a curvature command from the path array. Basically takes the curvature of the first path segment, but also attempts to smooth the curve when the first path segment is short (unless it is a single-segment path).



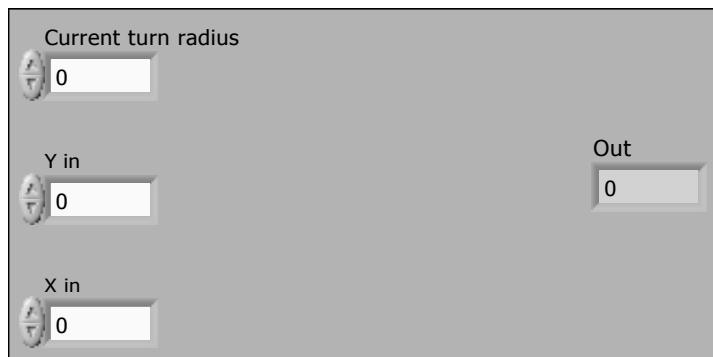
Checks if the path is a single curve
override by checking an element which
will only be zero in such a path.

Connector Pane

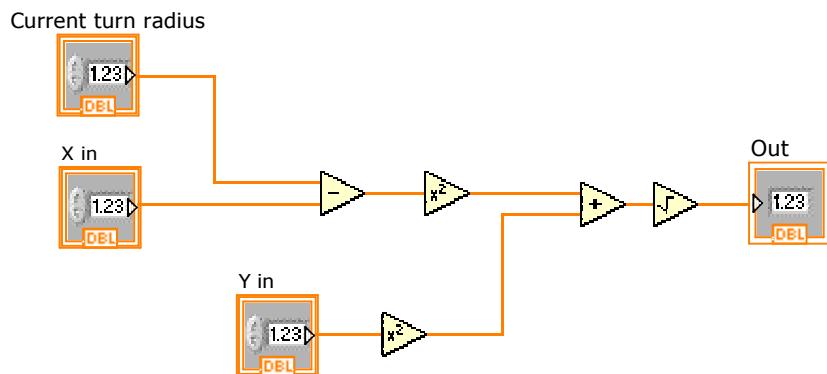
ObstacleDistanceFromTurnCenter.vi

Compute the distance from turn center.

Front Panel



Block Diagram

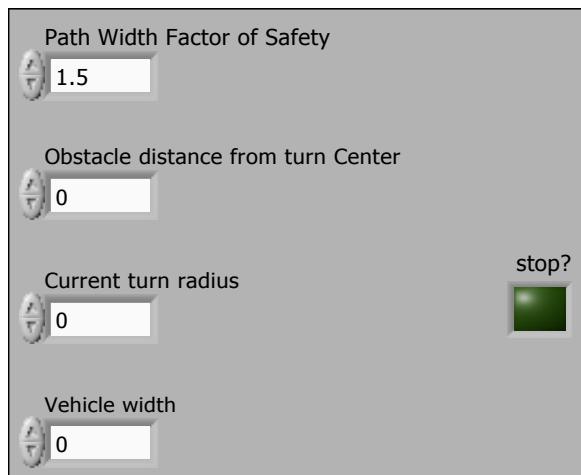


Connector Pane

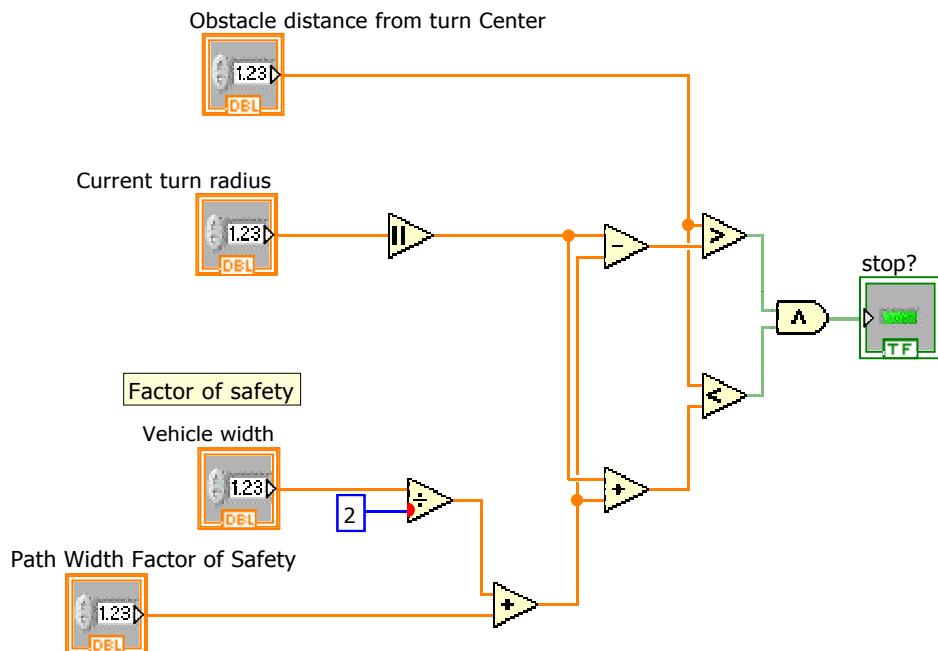
ObstacleInPath.vi

Determine if there is an obstacle in the vehicle's current path.

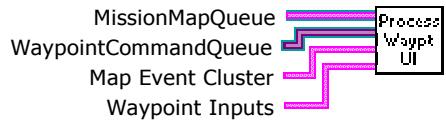
Front Panel



Block Diagram

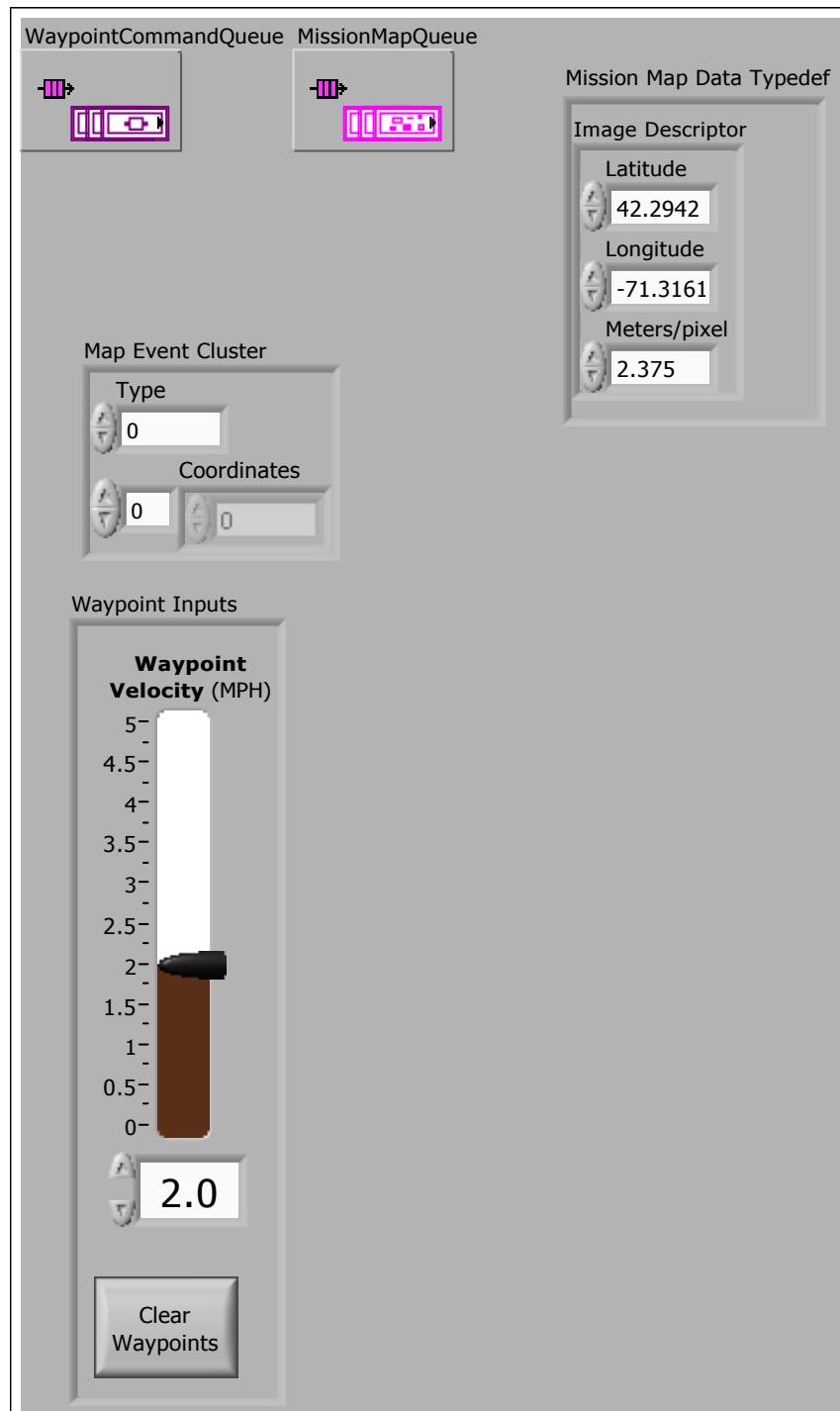


Connector Pane

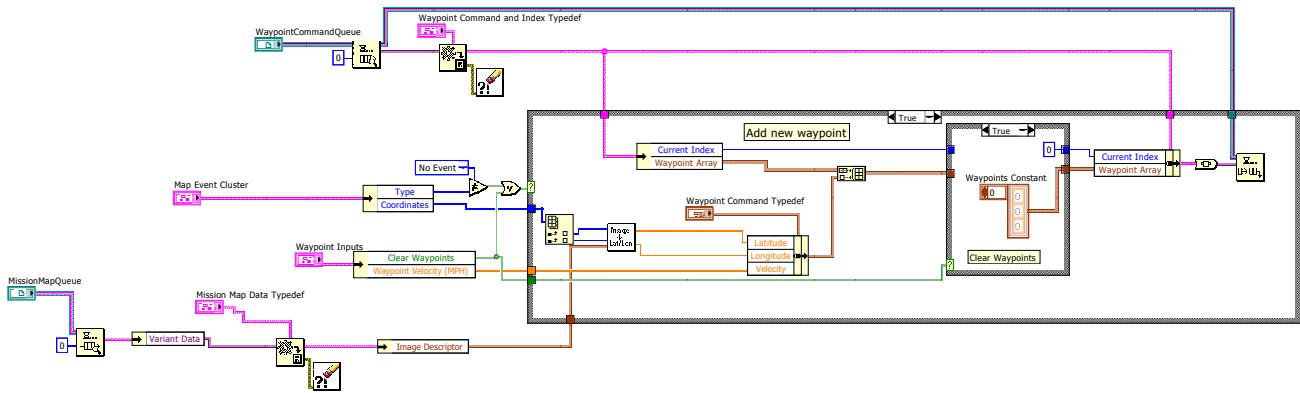
ProcessWaypointUI.vi

Processes clicks on the GPS map to create waypoints.

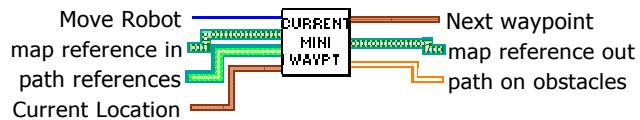
Front Panel



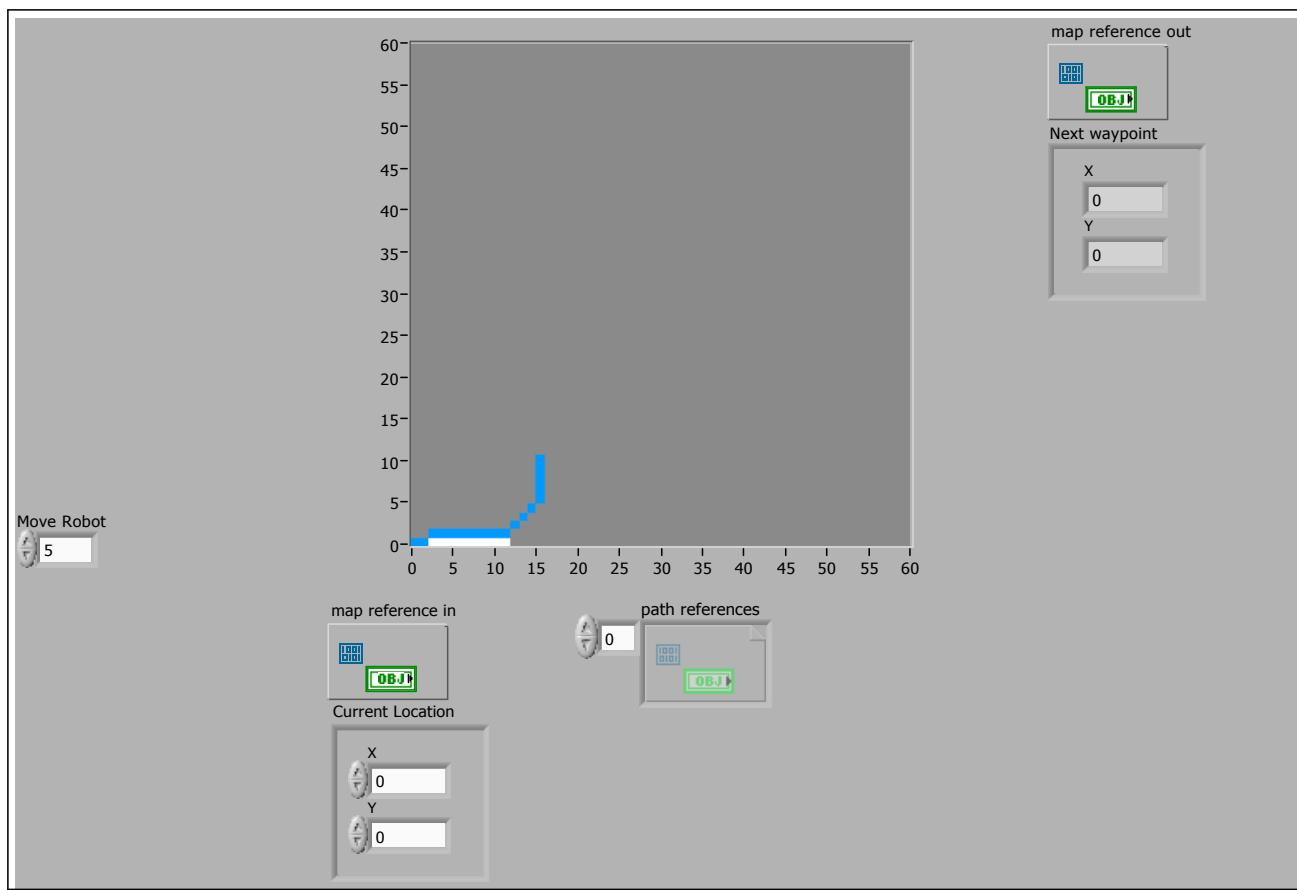
Block Diagram



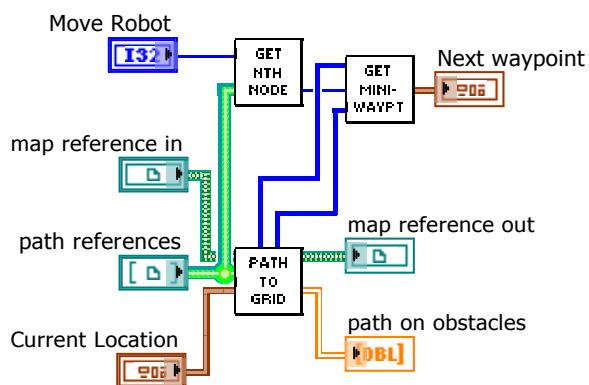
Connector Pane

Get Current Mini Waypoint.vi

Front Panel



Block Diagram

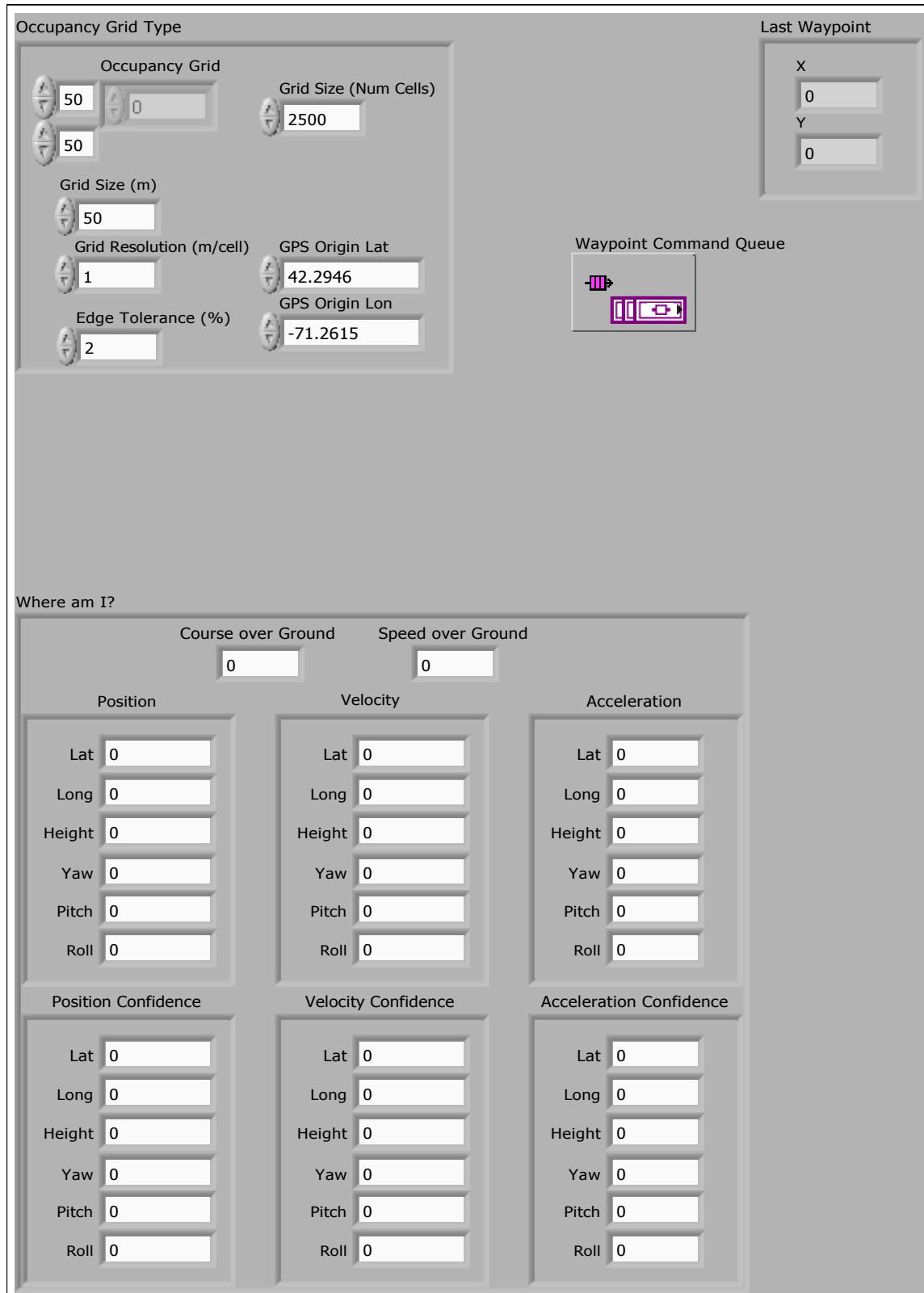


Connector Pane

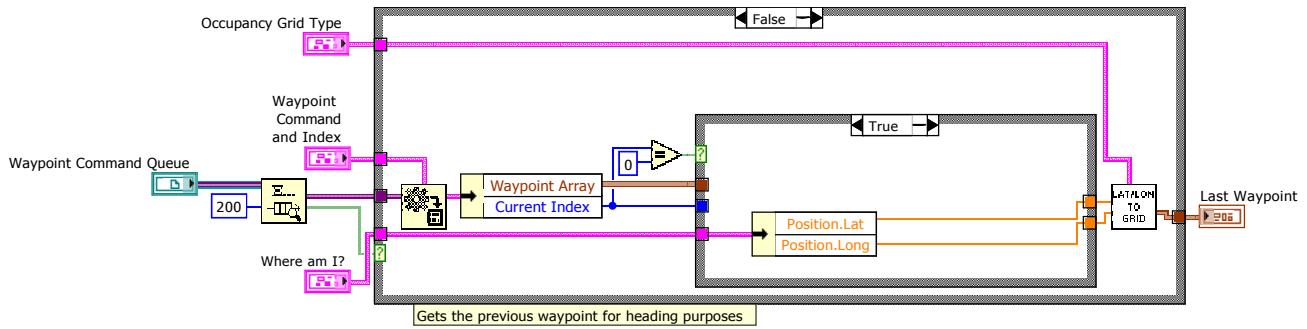
Get Last Waypoint.vi

Gets the previous waypoint from the waypoint queue

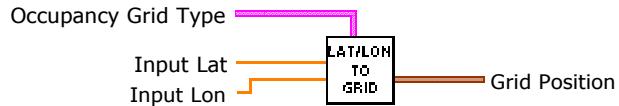
Front Panel



Block Diagram

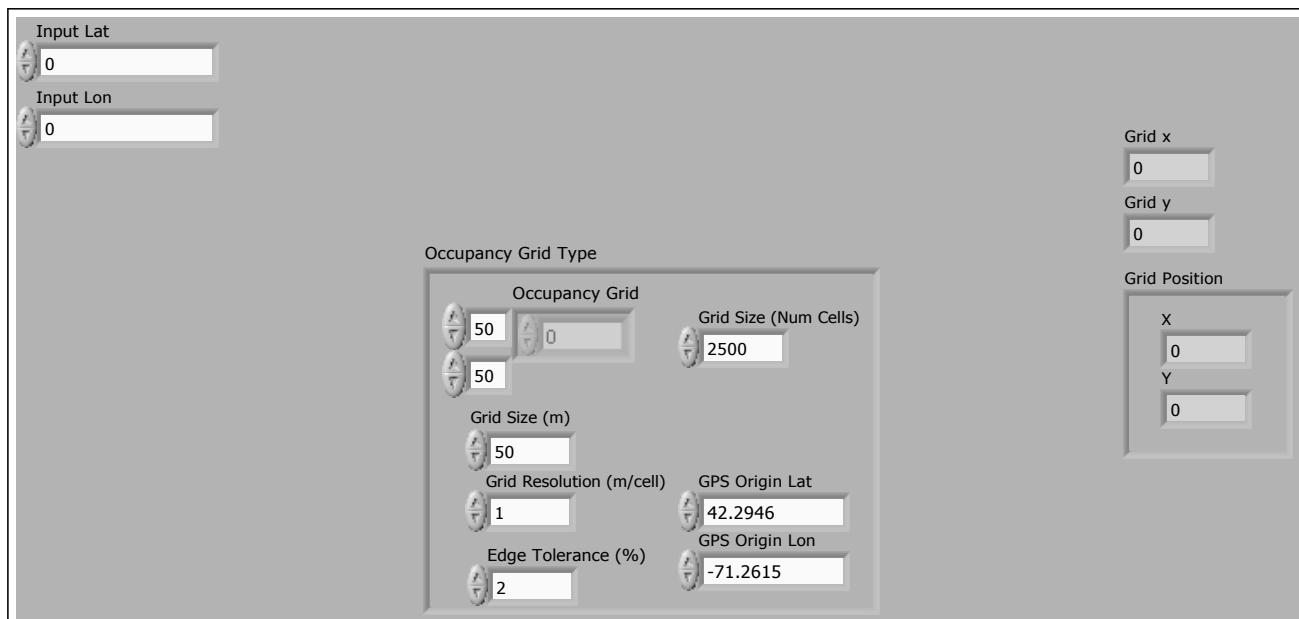


Connector Pane

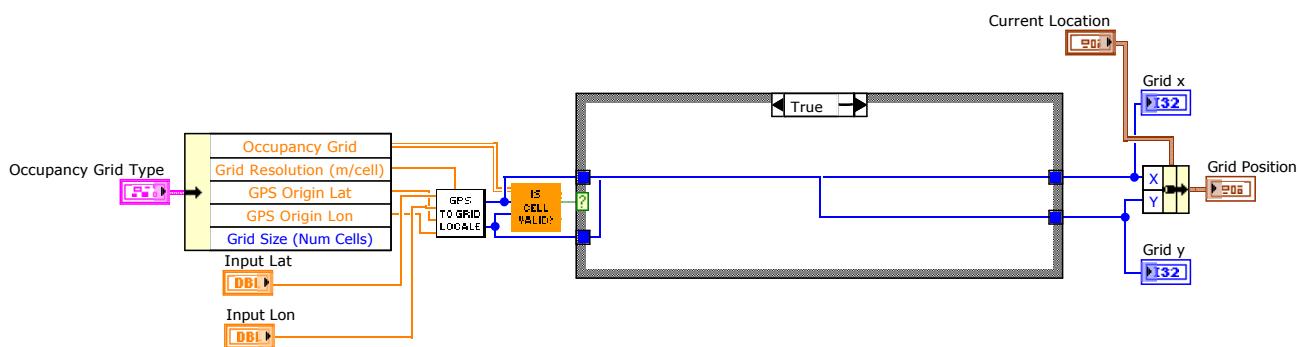
Lat Lon Position to Grid Position.vi

Selects either GPS or grid target location based on whether we are in simulation or waypoint-following mode.

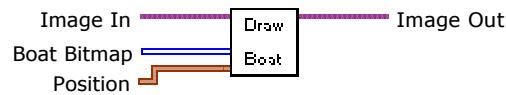
Front Panel



Block Diagram

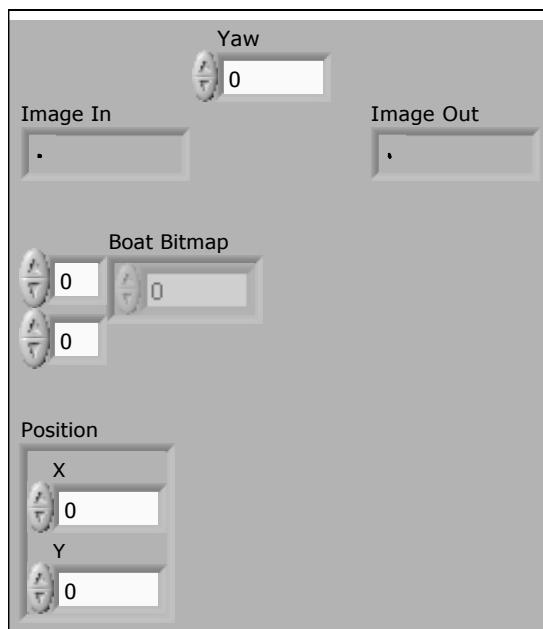


Connector Pane

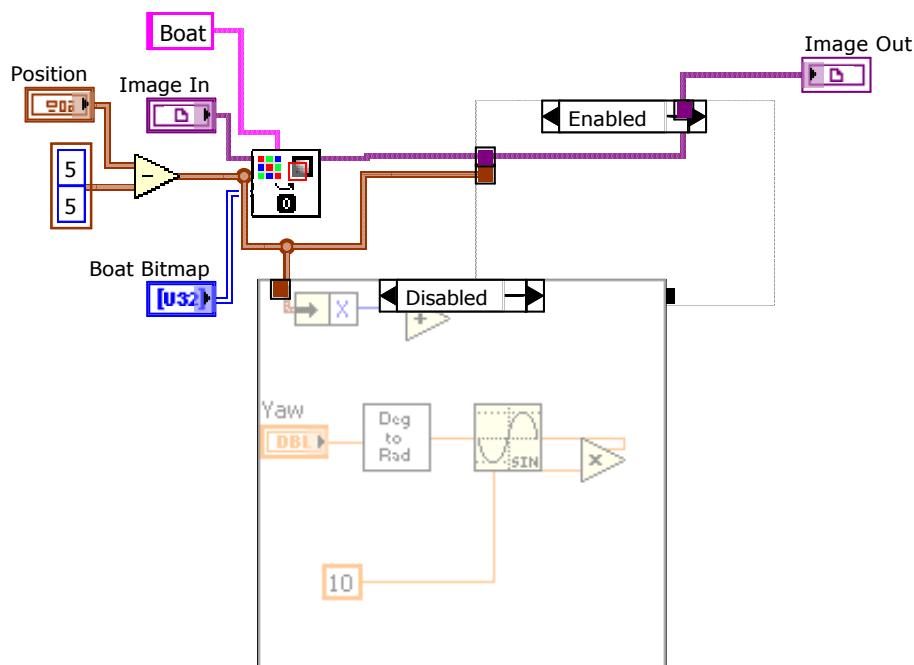
Draw Boat Icon.vi

Draws the boat icon onto the map at the current location of the boat.

Front Panel



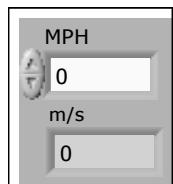
Block Diagram



Connector Pane

MphToMperS.vi

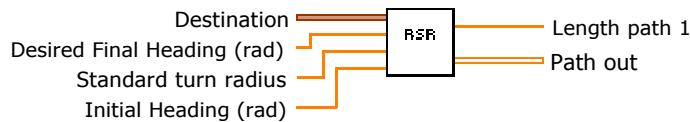
Front Panel



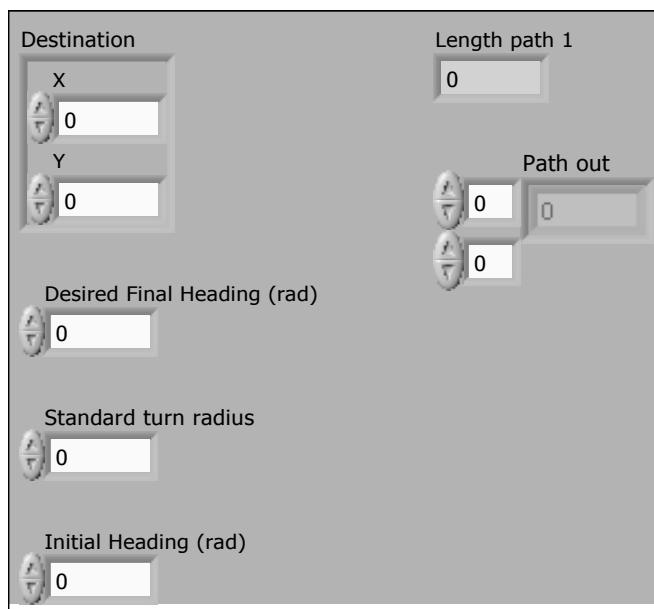
Block Diagram



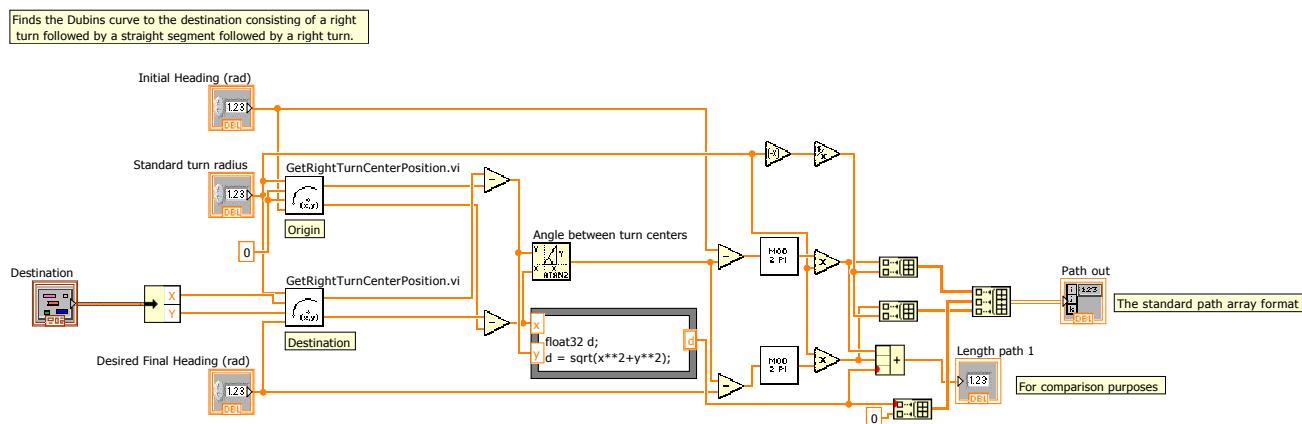
Connector Pane

FindRSRPath.vi

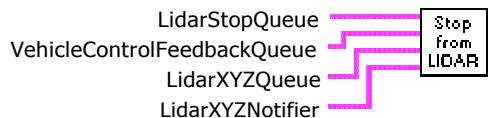
Front Panel



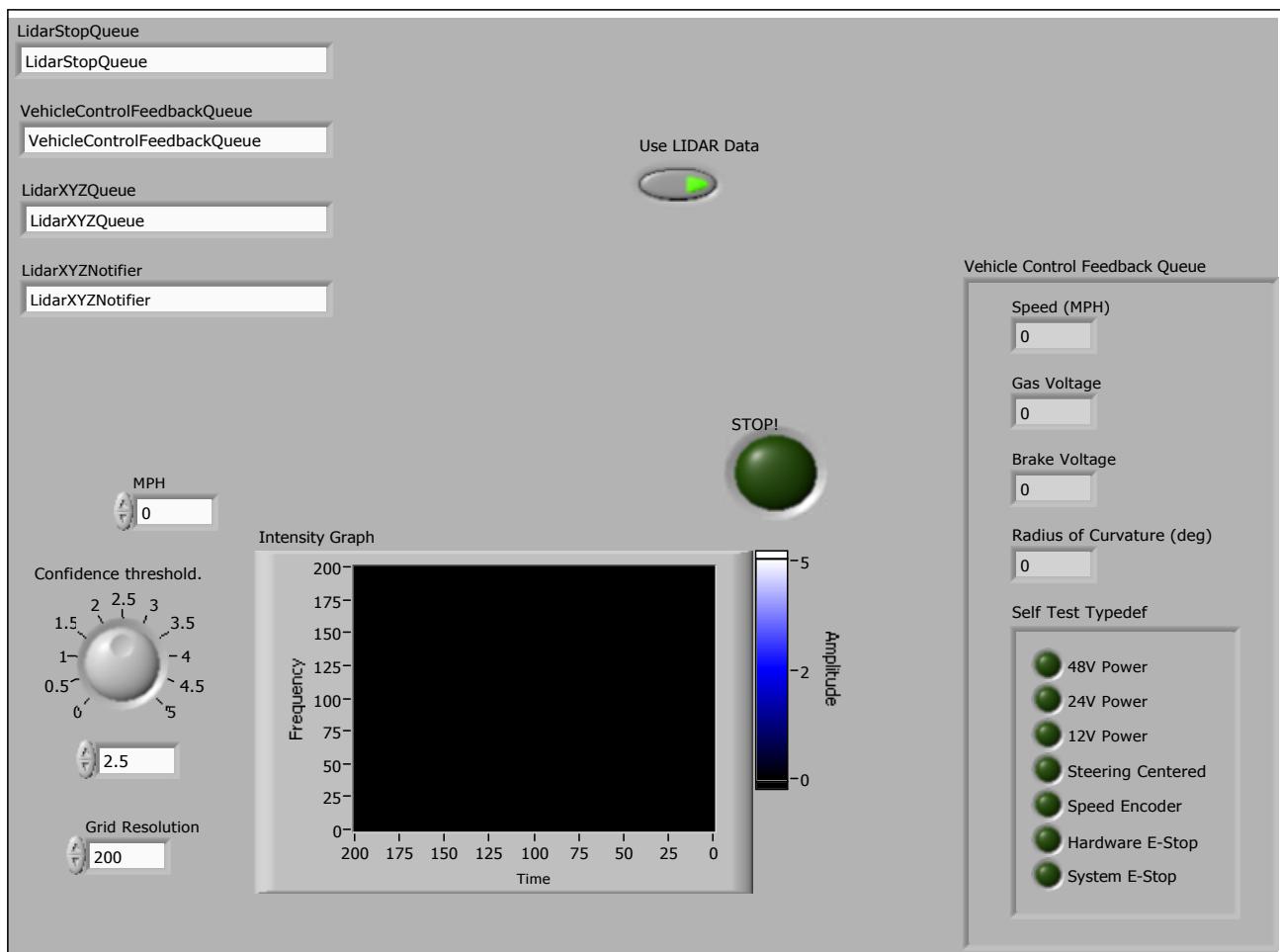
Block Diagram



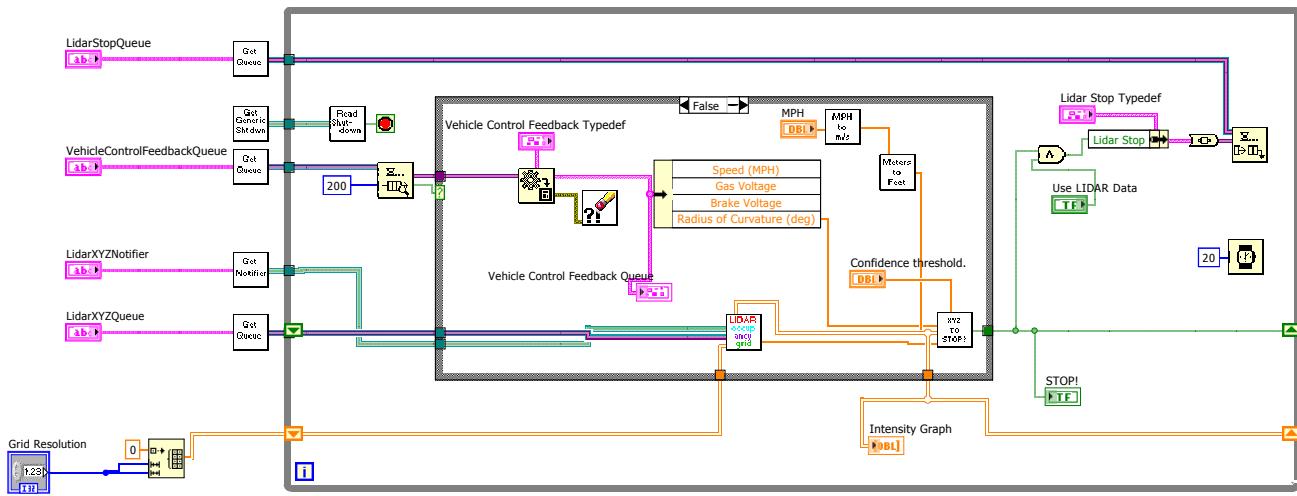
Connector Pane

StopFromLidarData.vi

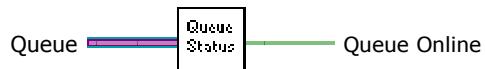
Front Panel



Block Diagram

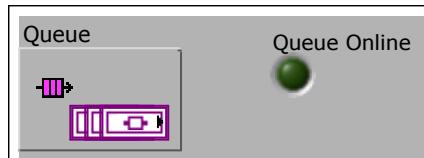


Connector Pane

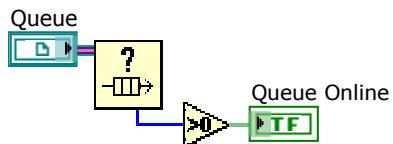
QueueStatus.vi

Determines if a queue is online (has data in it).

Front Panel



Block Diagram

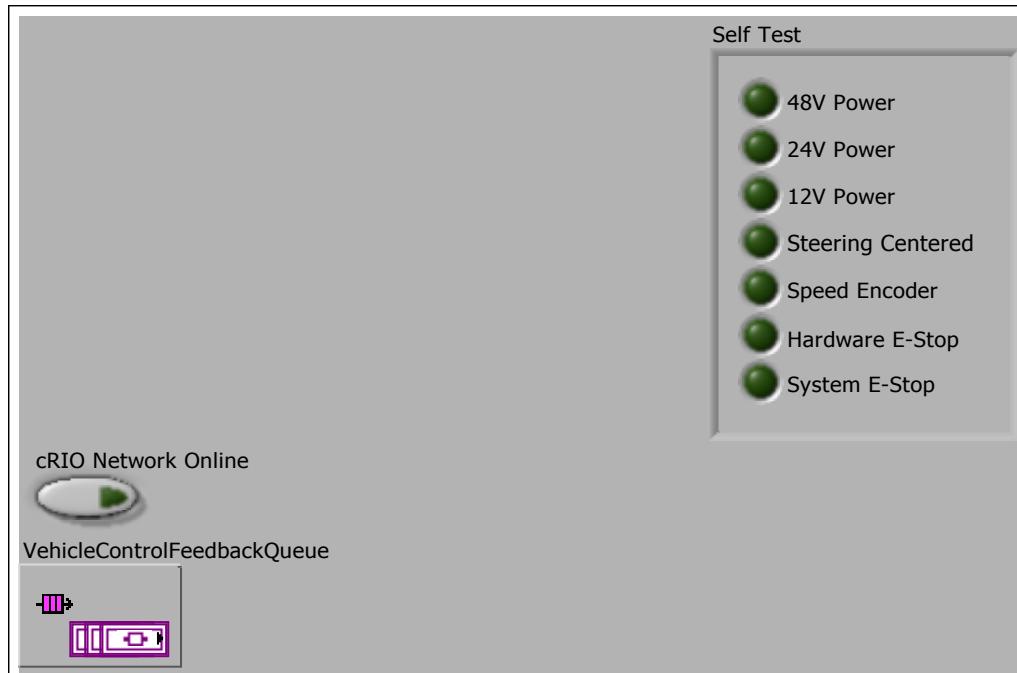


Connector Pane

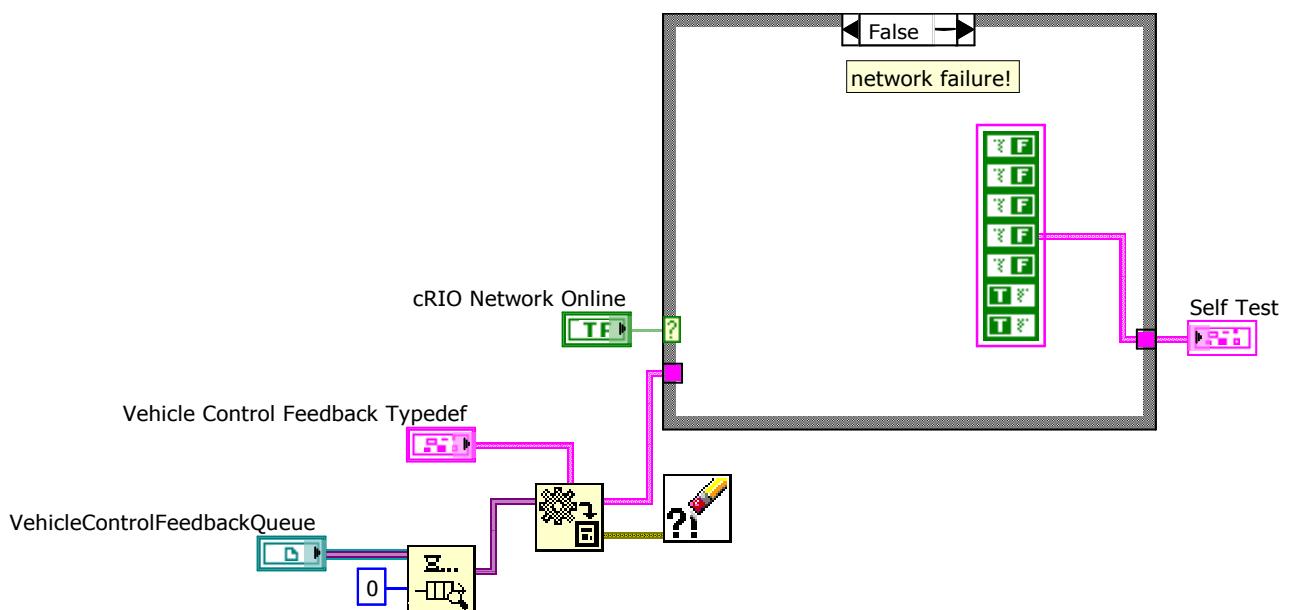
ReadcRIOStatusGUI.vi

Reads the cRIO Status queue for display on the GUI.

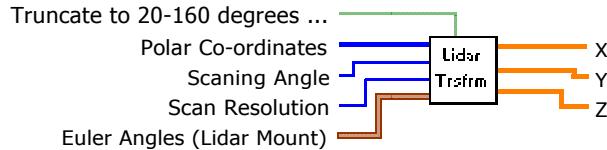
Front Panel



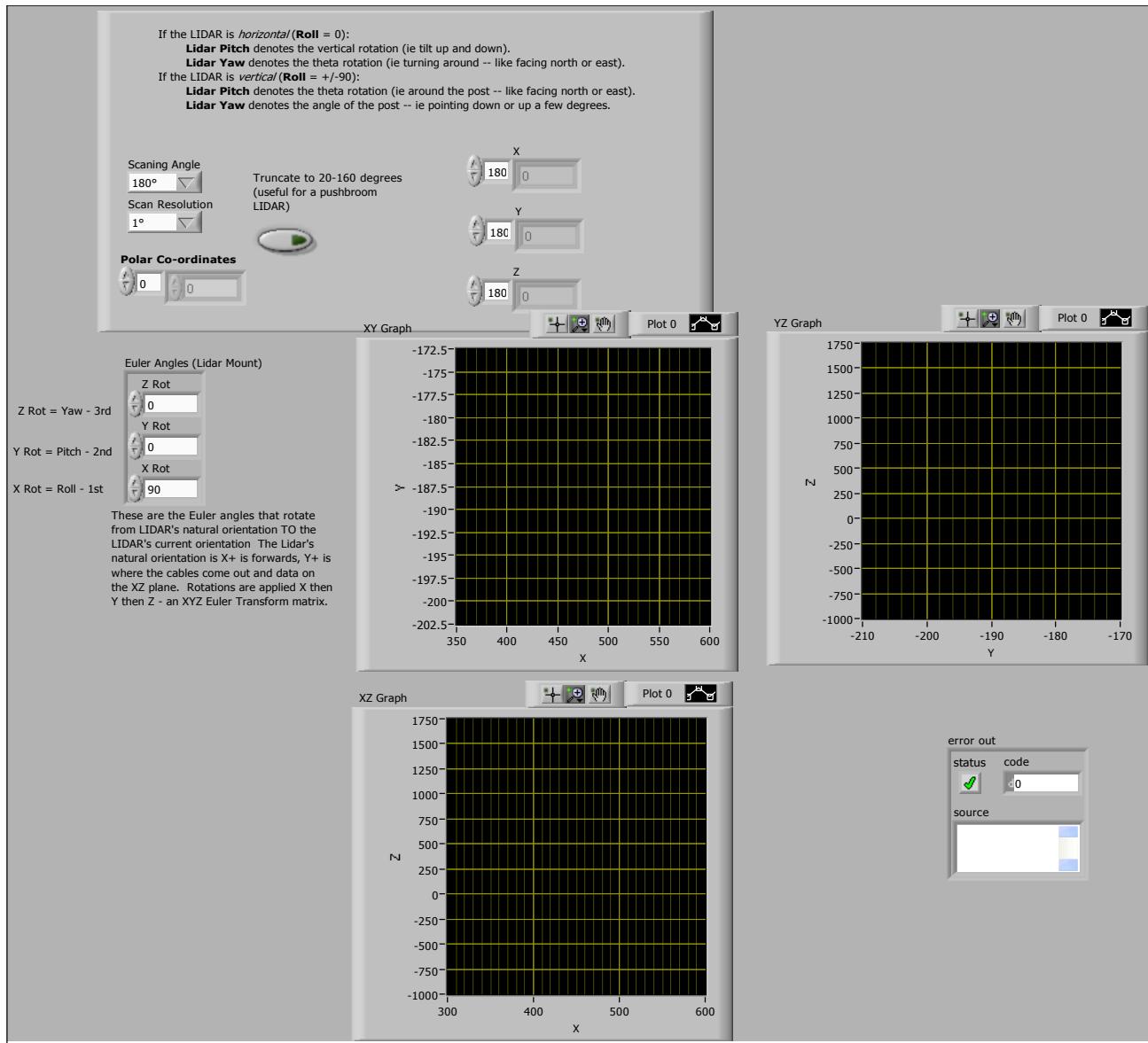
Block Diagram



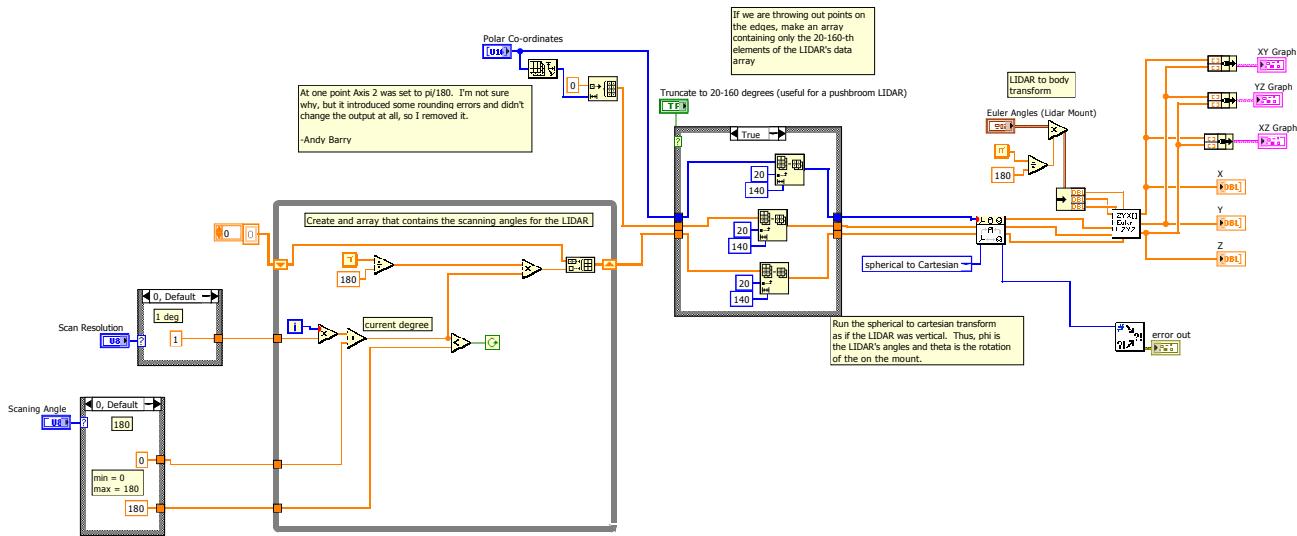
Connector Pane

LidarPolarToCart.vi

Front Panel



Block Diagram

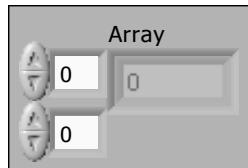


Connector Pane

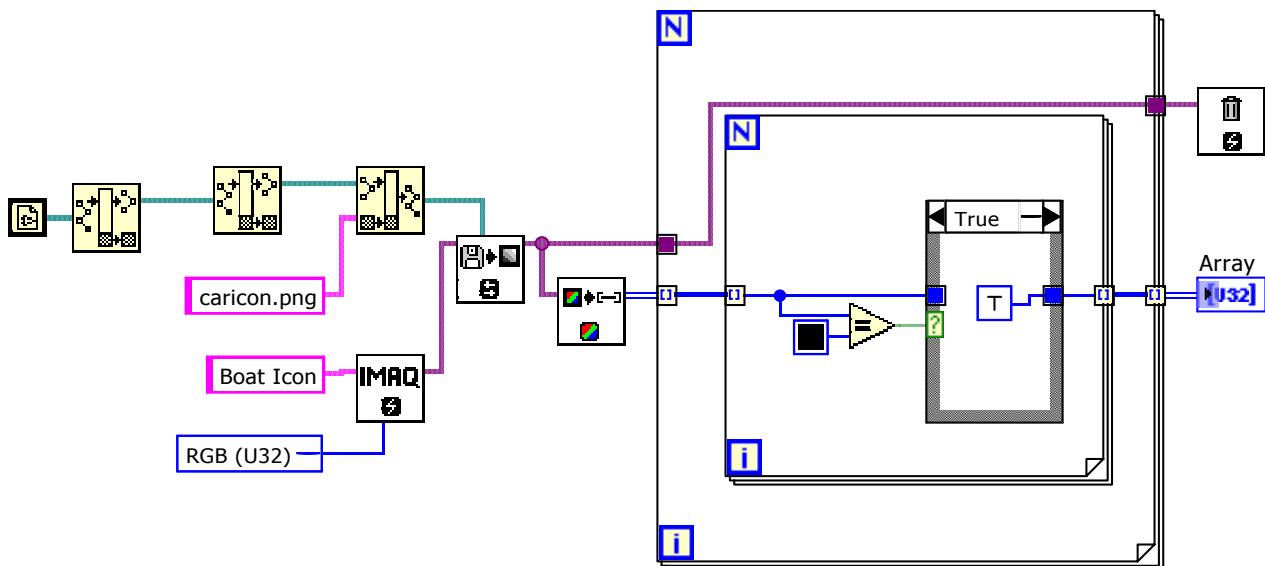
Load Boat Icon.vi

Loads the boat icon for use on the map. Also, replaces all the black pixels with transparent pixels.

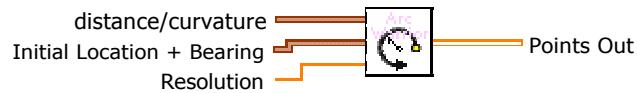
Front Panel



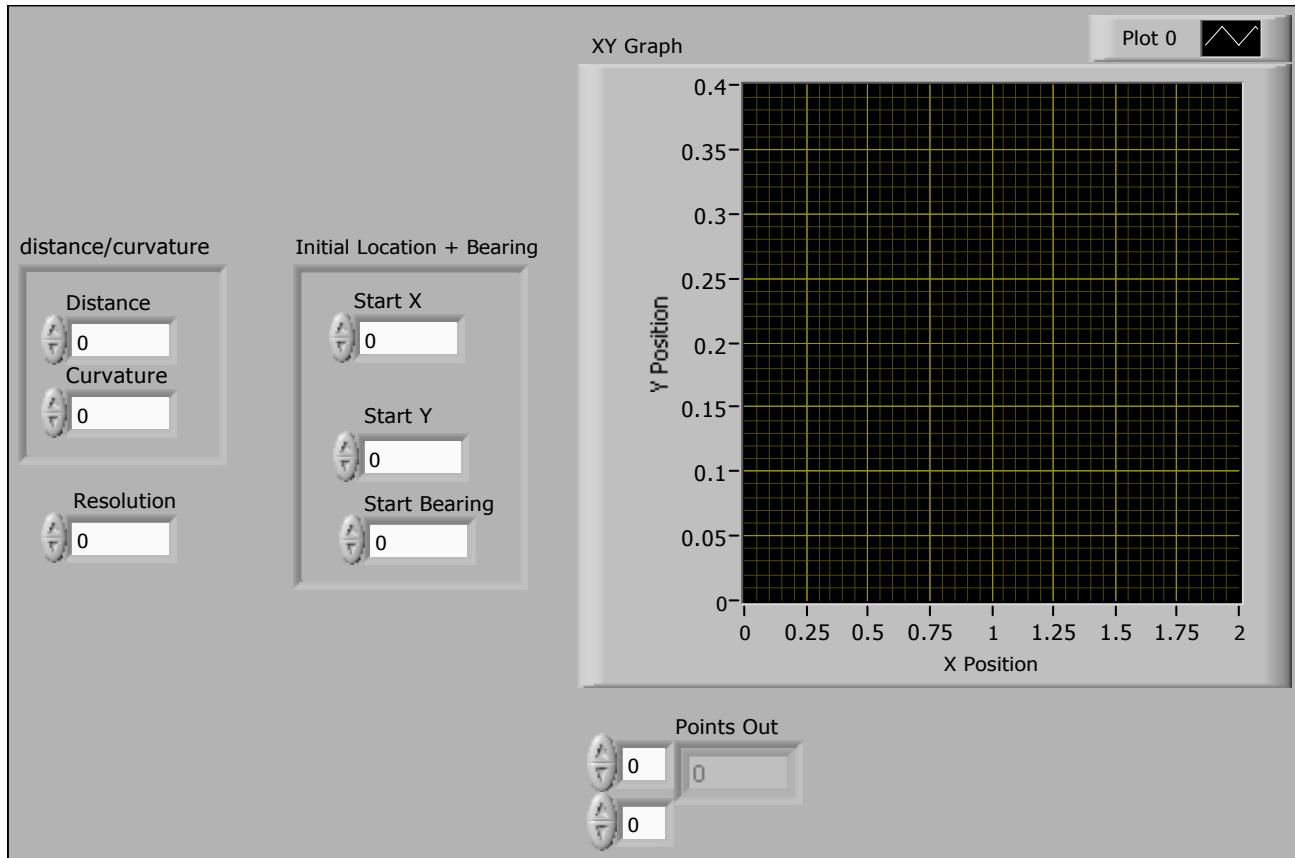
Block Diagram



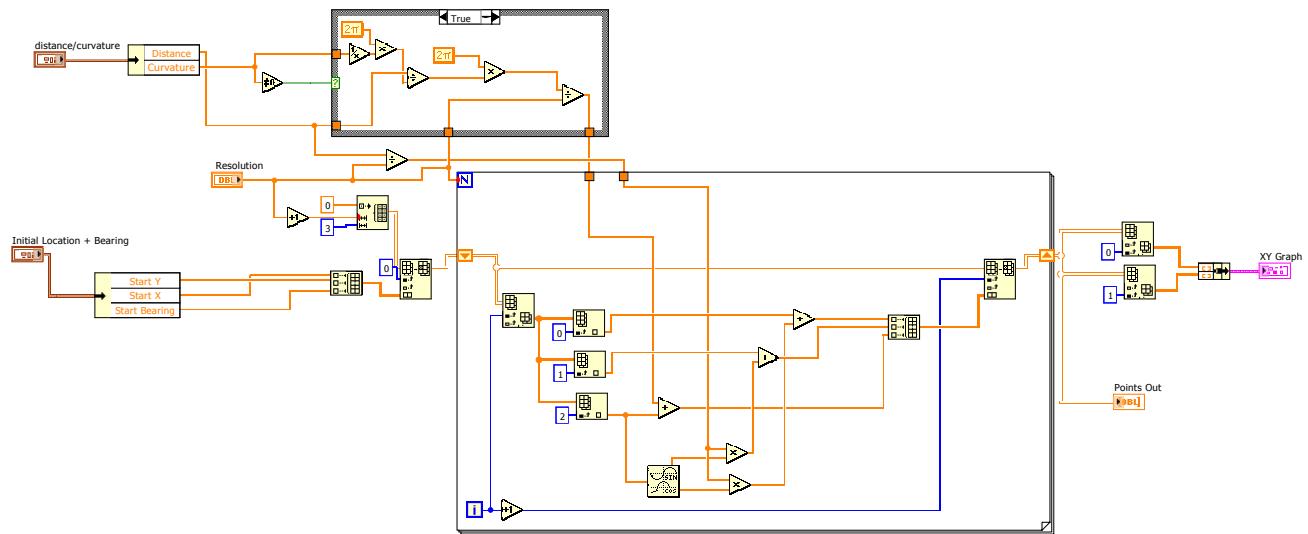
Connector Pane

arcbuilder.vi

Front Panel



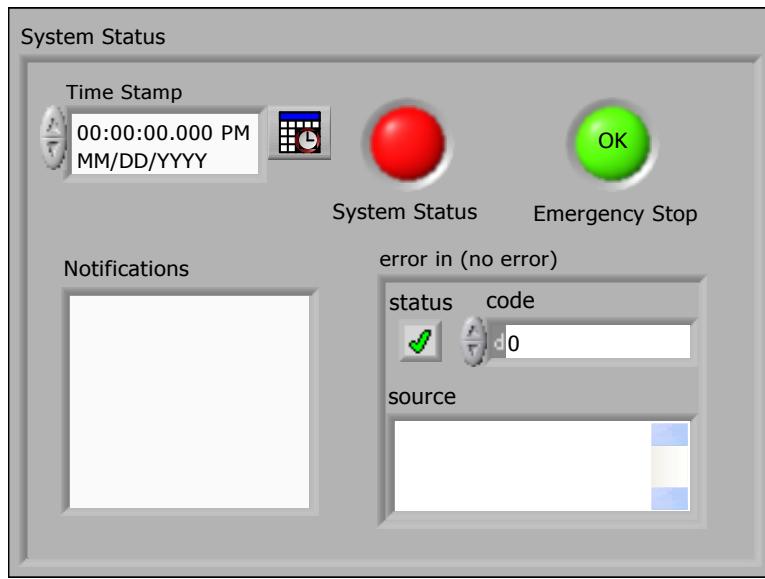
Block Diagram



Connector Pane

SystemStatus.ctl

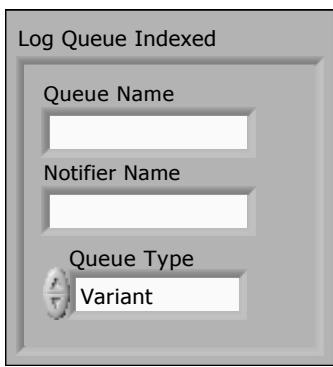
Front Panel



Block Diagram

Connector Pane

LogQueueIndexControl.ctl

Front Panel

Block Diagram

Connector Pane

LogReplayControl.ctl

Front Panel

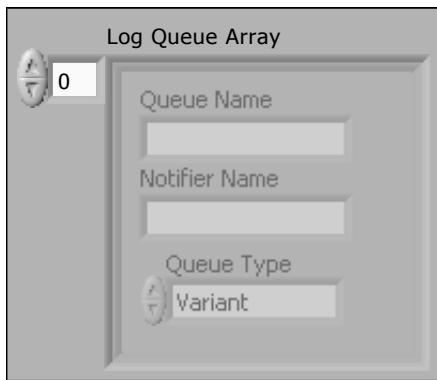


Block Diagram

Connector Pane

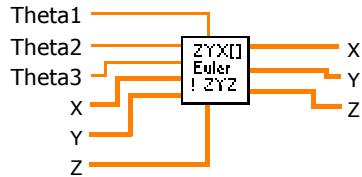
LogQueueControl.ctl

Front Panel



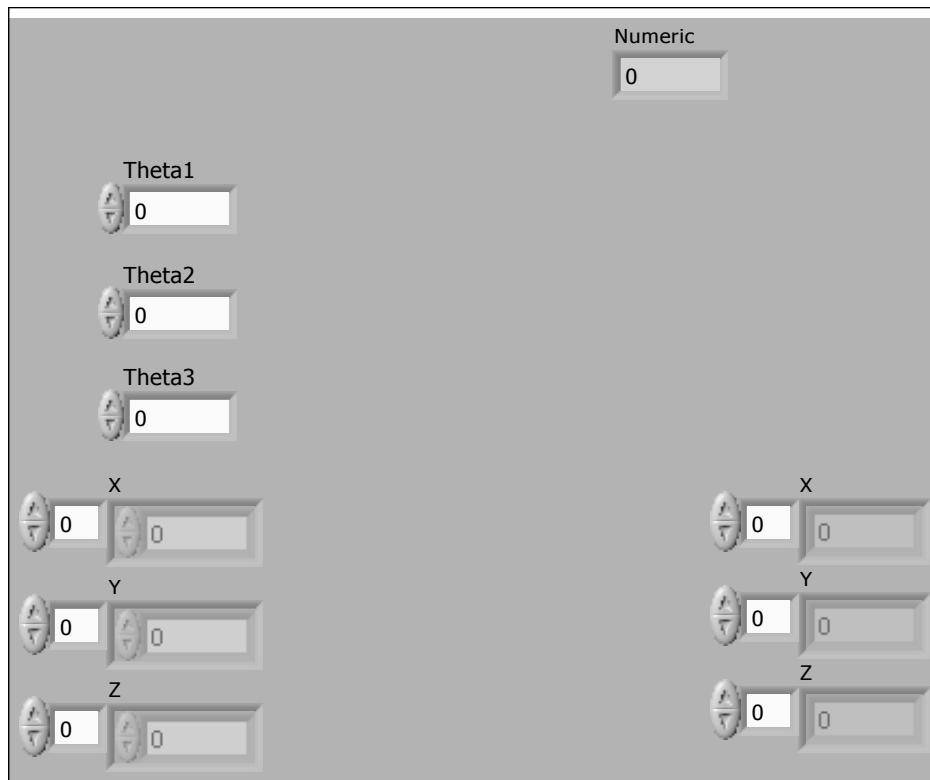
Block Diagram

Connector Pane

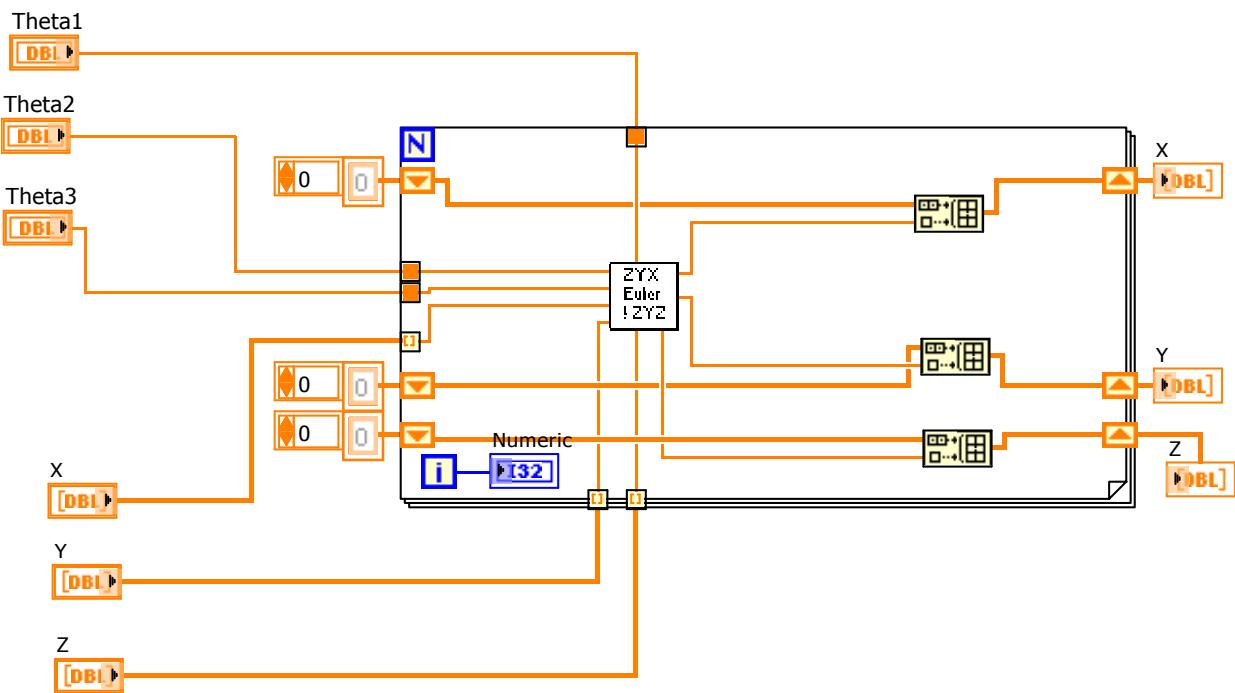
ZYXEulerArray.vi

Performs a ZYX transform on an array of XYZ values with the angles Theta1, Theta2, and Theta3.

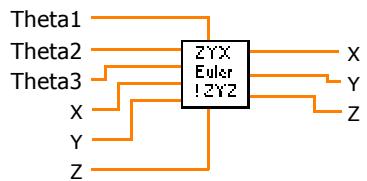
Front Panel



Block Diagram

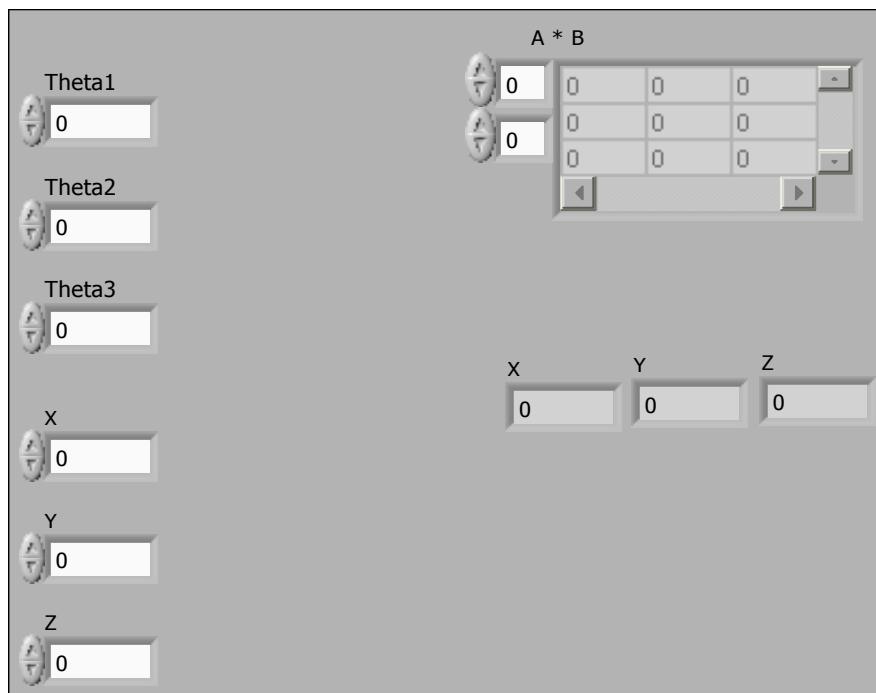


Connector Pane

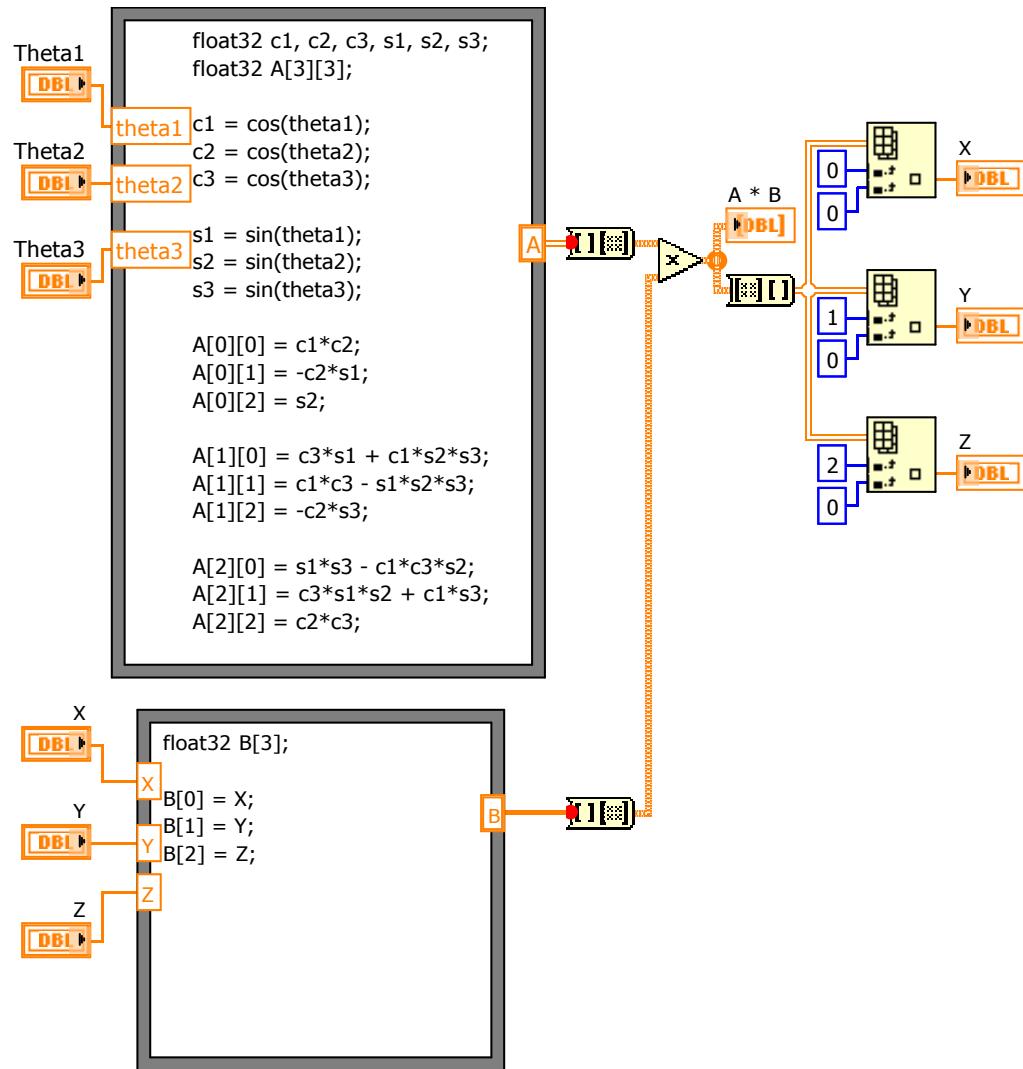
ZYXEuler.vi

Performs an ZYX Euler angle transform.

Front Panel



Block Diagram

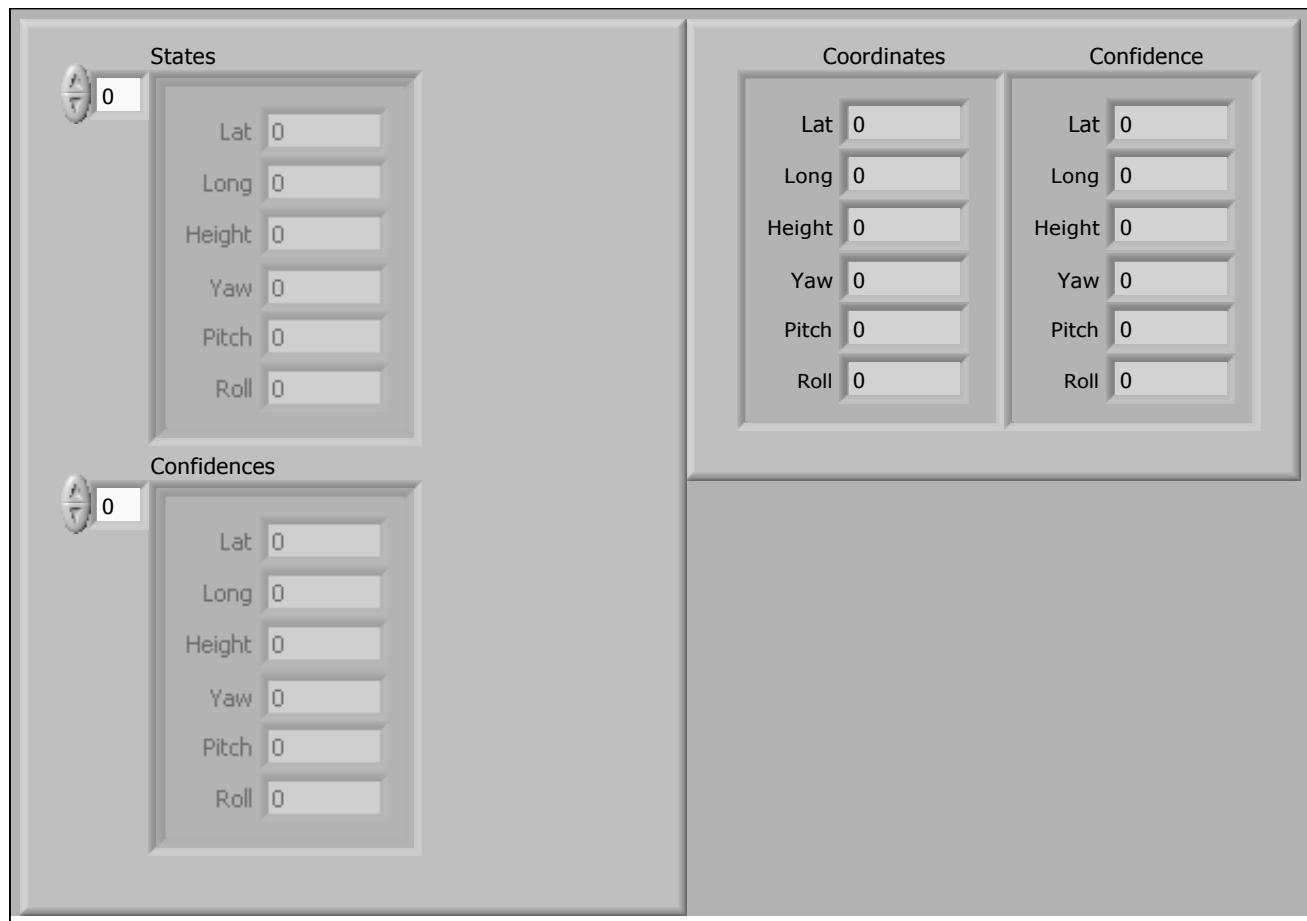


Connector Pane

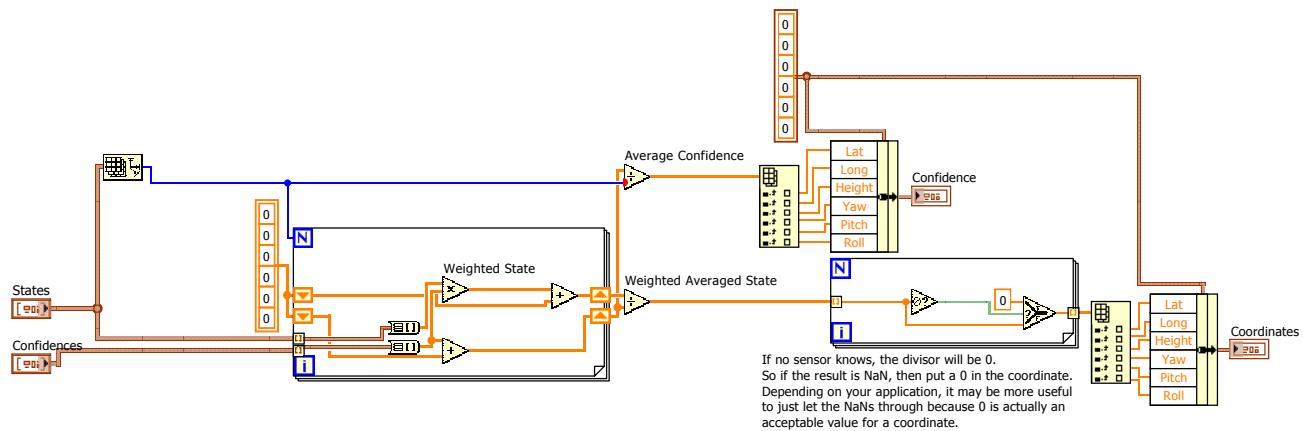
State Weighted Average.vi

Takes in sensor data from multiple sources and finds the weighed average according to corresponding confidences.

Front Panel



Block Diagram



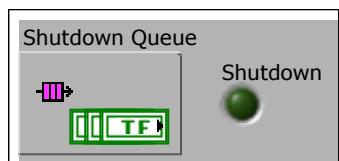
Connector Pane

ReadShutdownQueue.vi

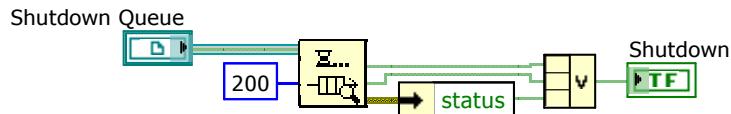
Reads a shutdown queue and returns true for shutdown on any of three conditions:

1. Shutdown queue holds a True.
2. Shutdown queue timed out.
3. Shutdown queue read returned an error.

Front Panel



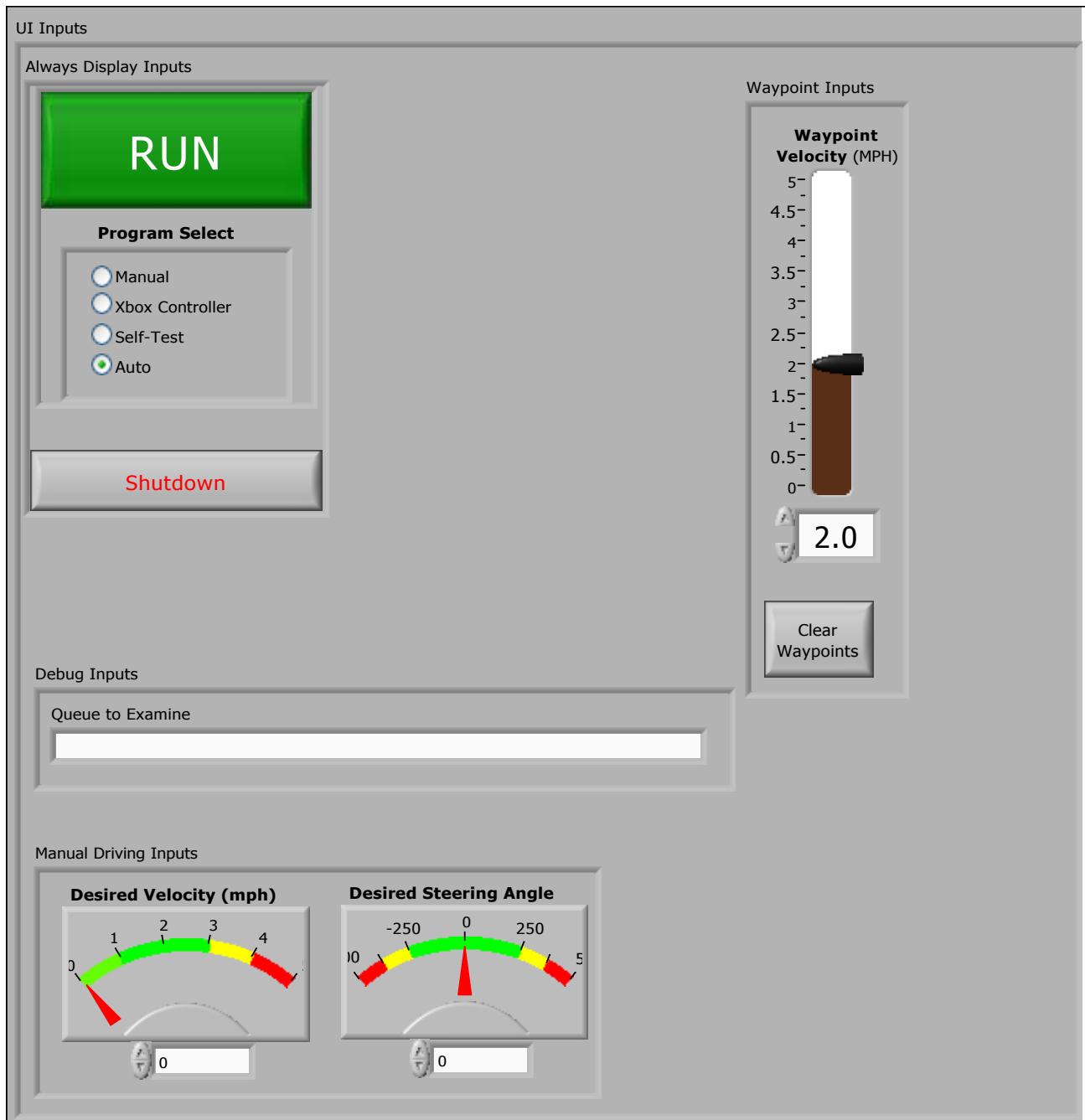
Block Diagram



Connector Pane

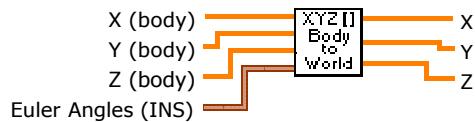
InputsUI.ctl

Front Panel



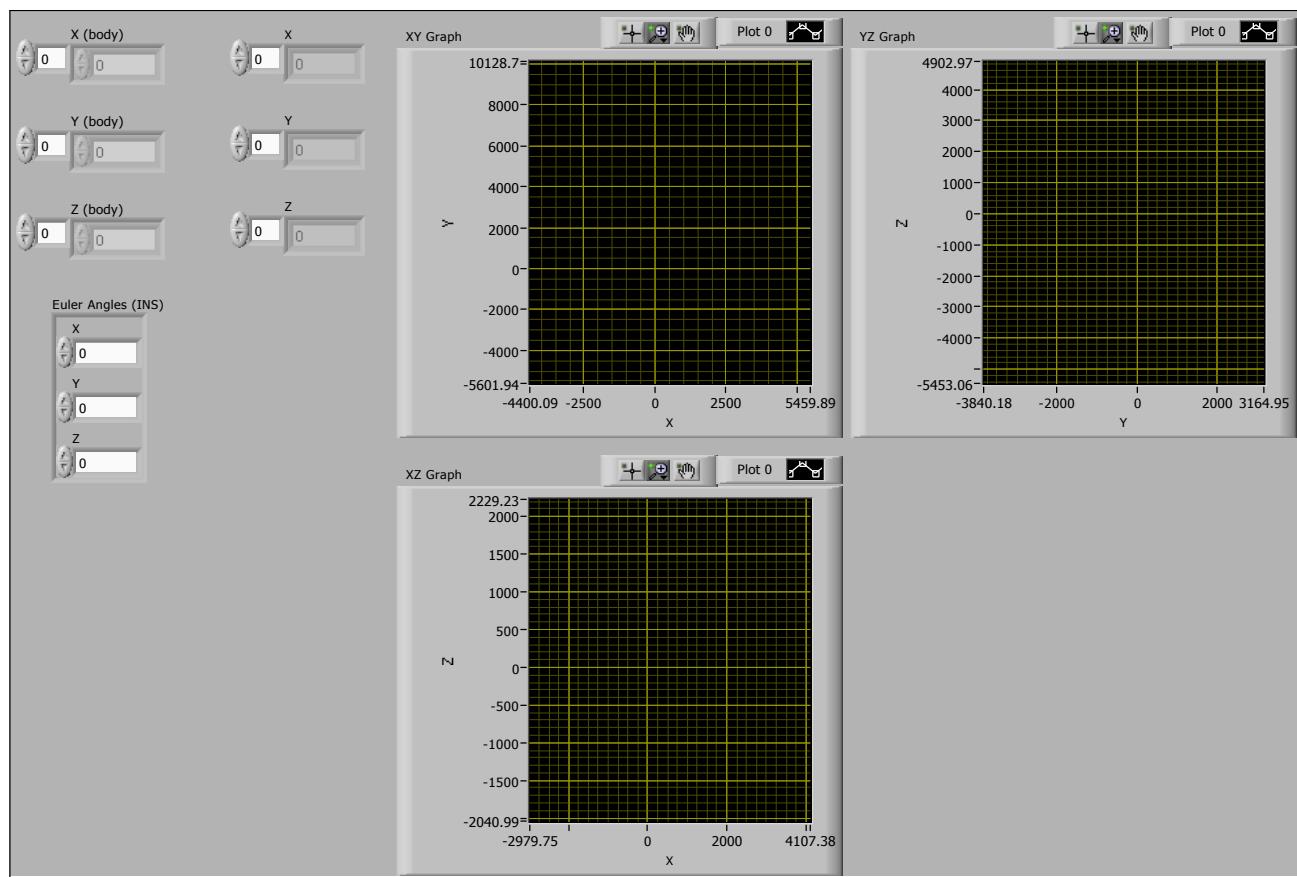
Block Diagram

Connector Pane

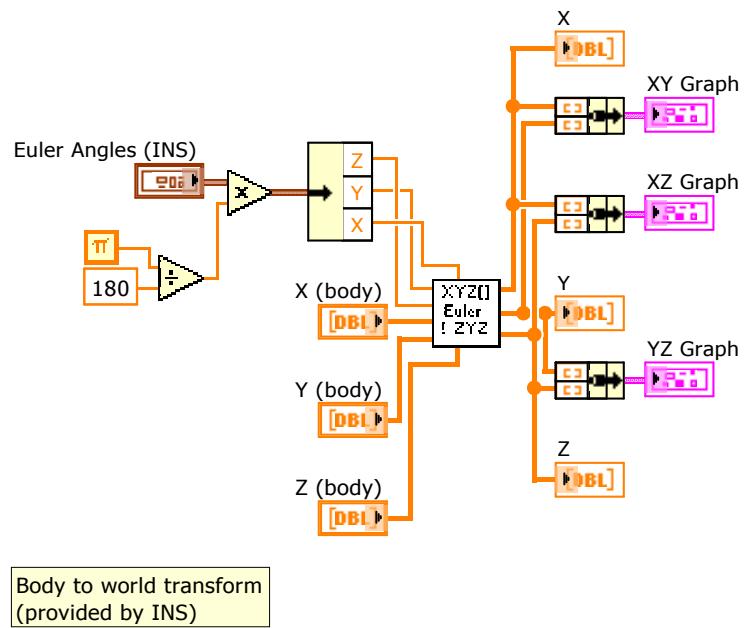
BodyToWorldTransformArray.vi

Performs the Body to World transform given INS angles on an array of data. See **BodyToWorldTransform.vi** for details.

Front Panel



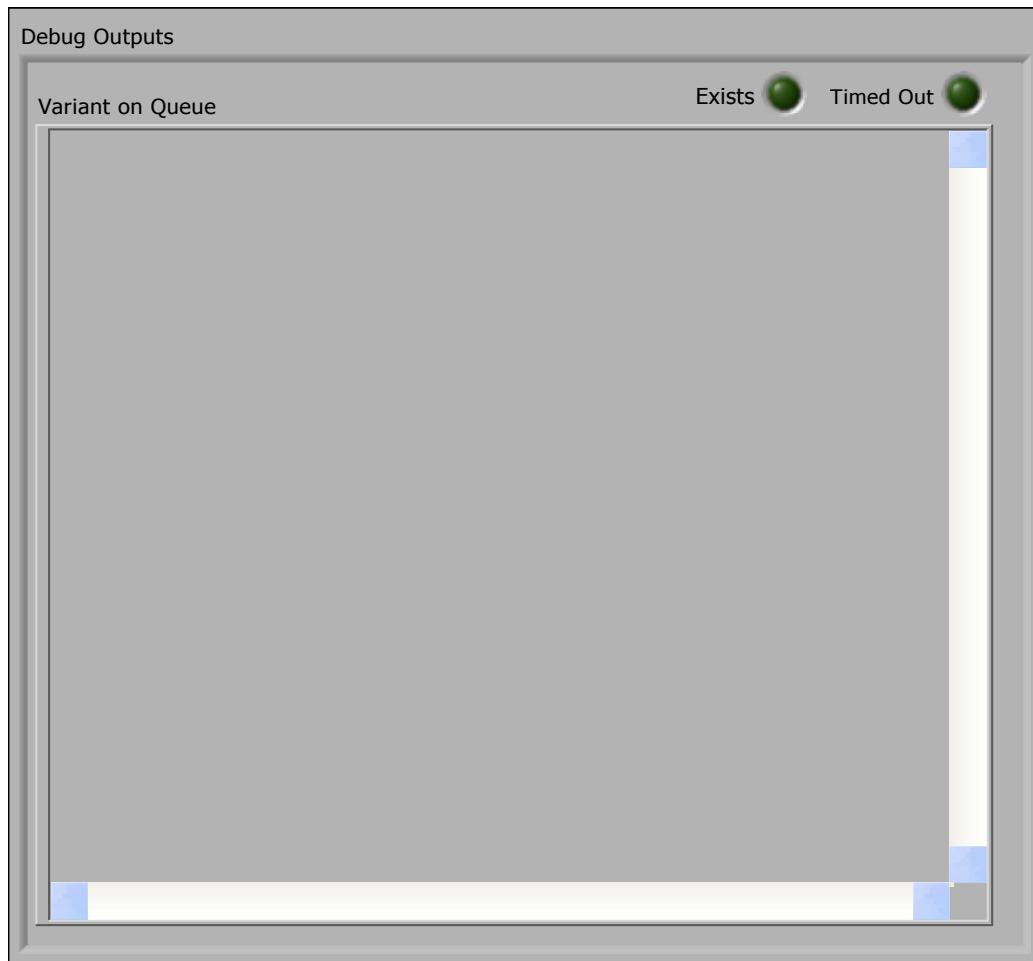
Block Diagram



Connector Pane

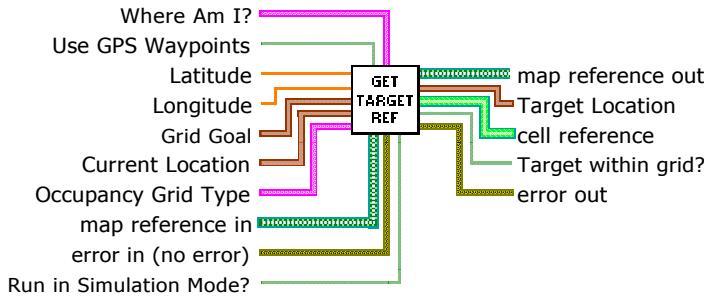
DebugOutputs.ctl

Front Panel



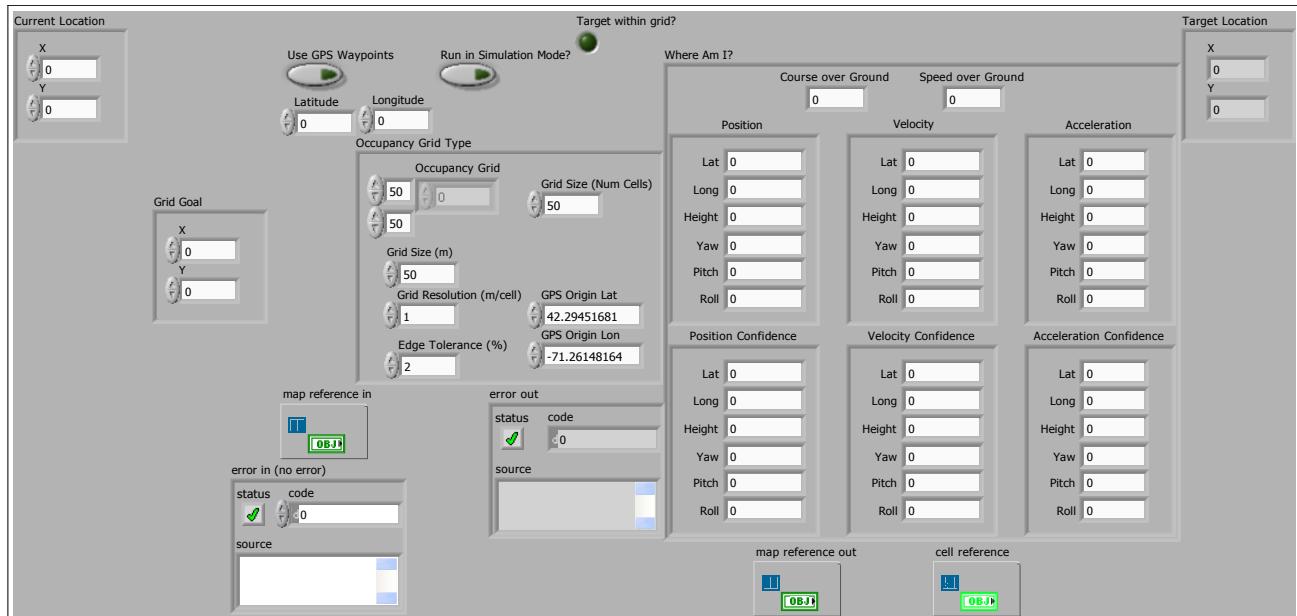
Block Diagram

Connector Pane

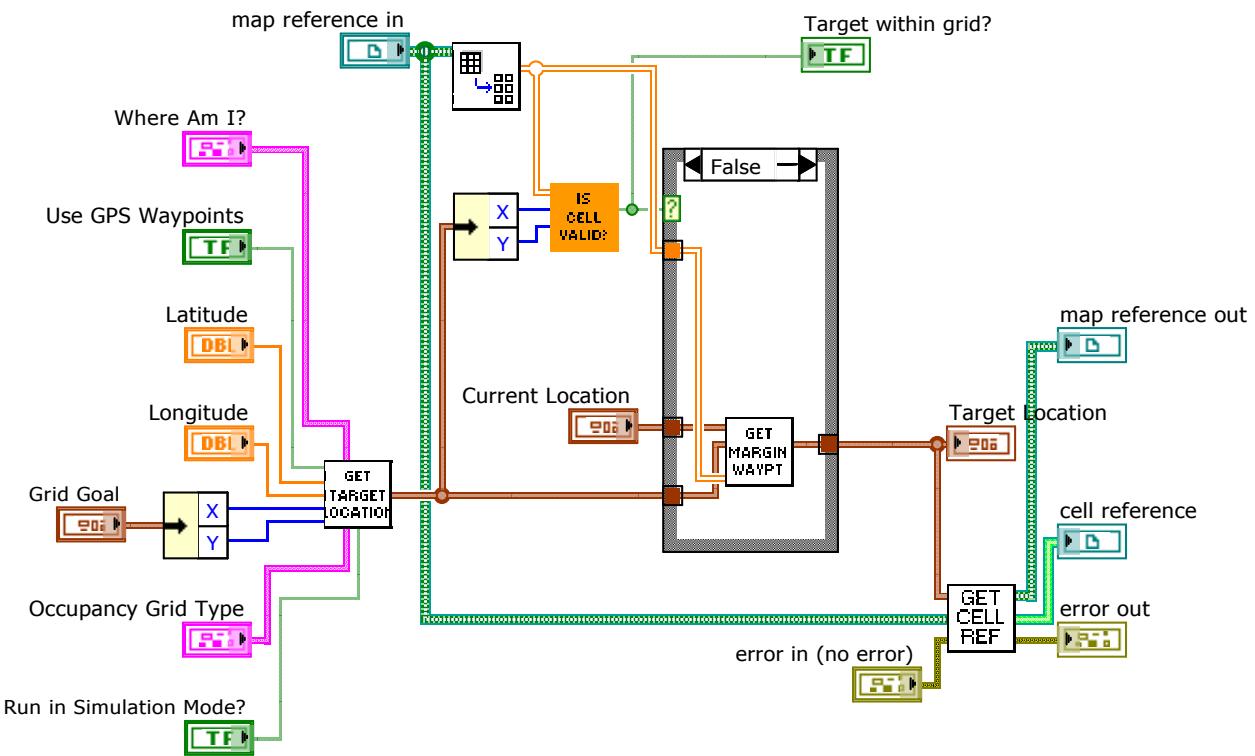
Get Target Reference.vi

Gets a reference to the target cell in the grid

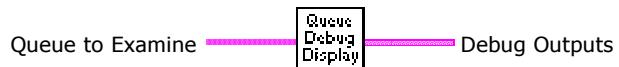
Front Panel



Block Diagram



Connector Pane

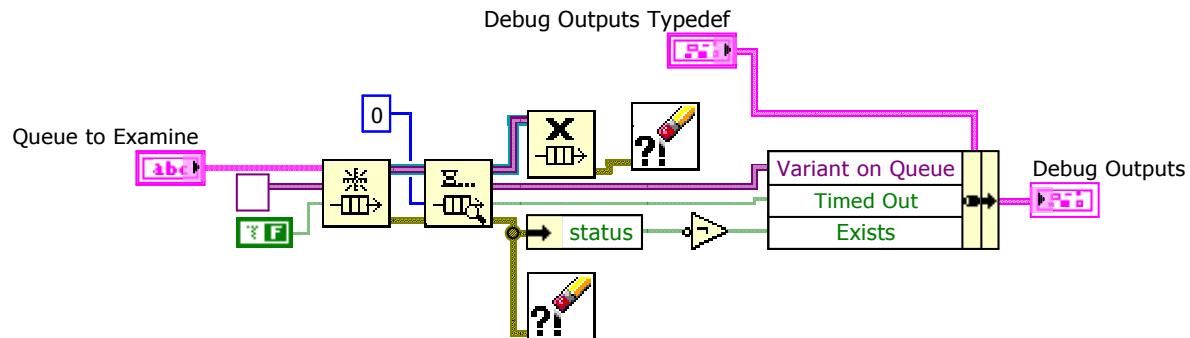
QueueDebugDisplay.vi

Populates the Queue Debug Display.

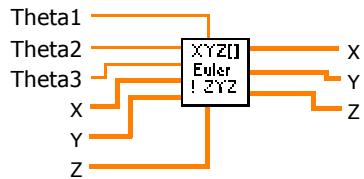
Front Panel



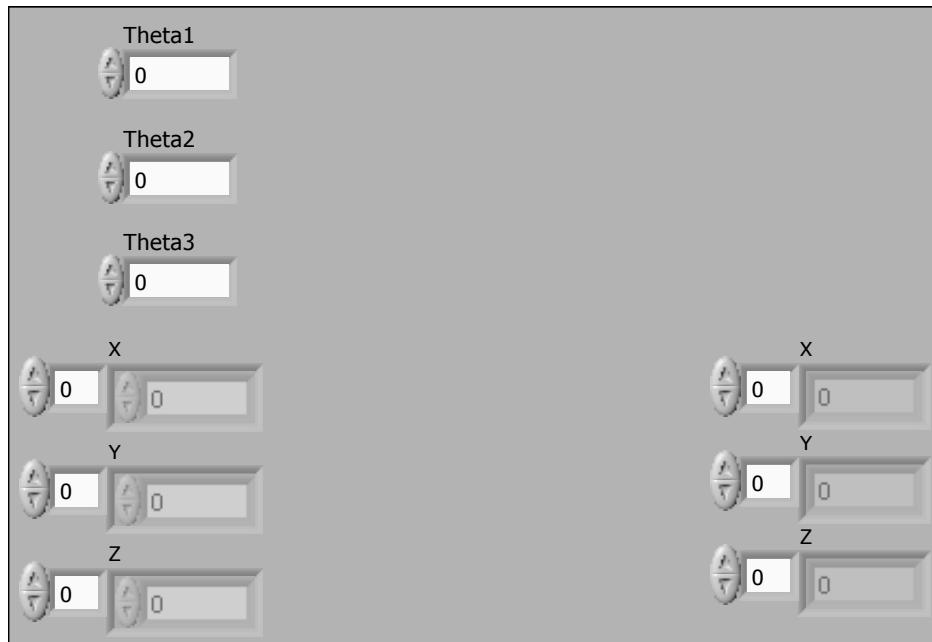
Block Diagram



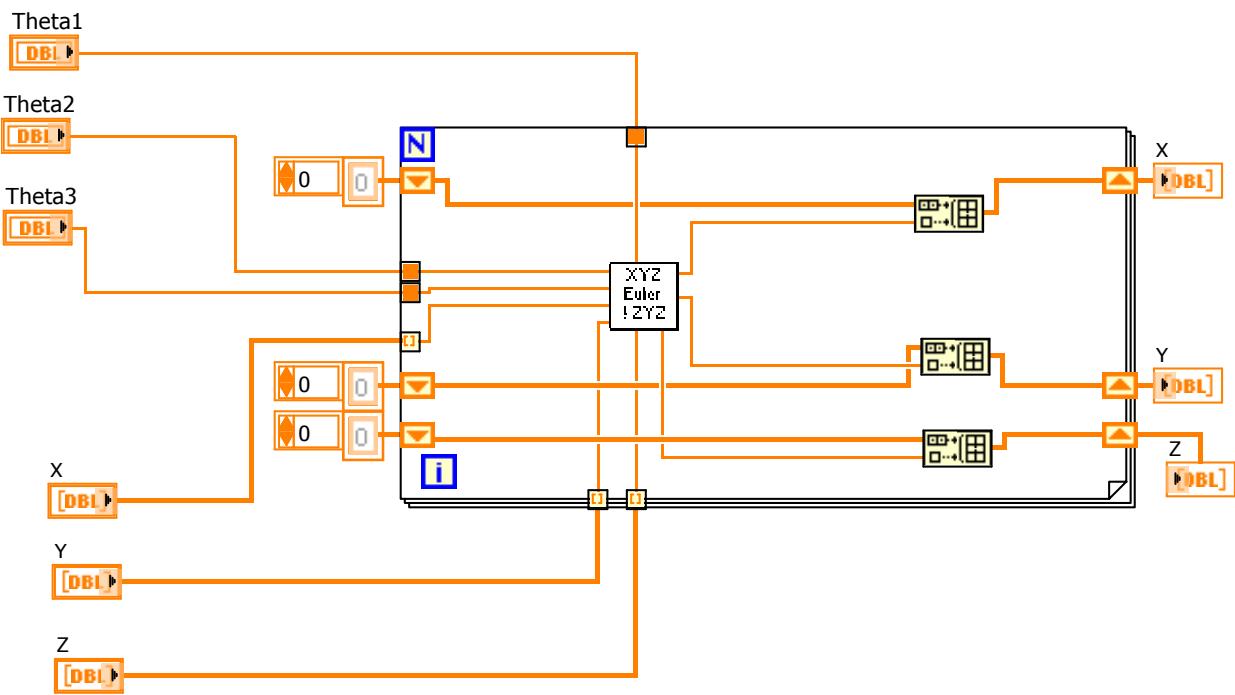
Connector Pane

XYZEulerArray.vi

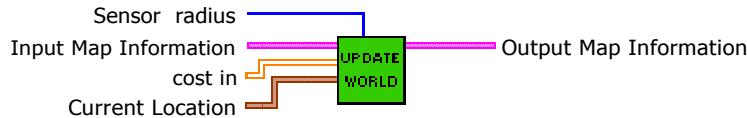
Front Panel



Block Diagram

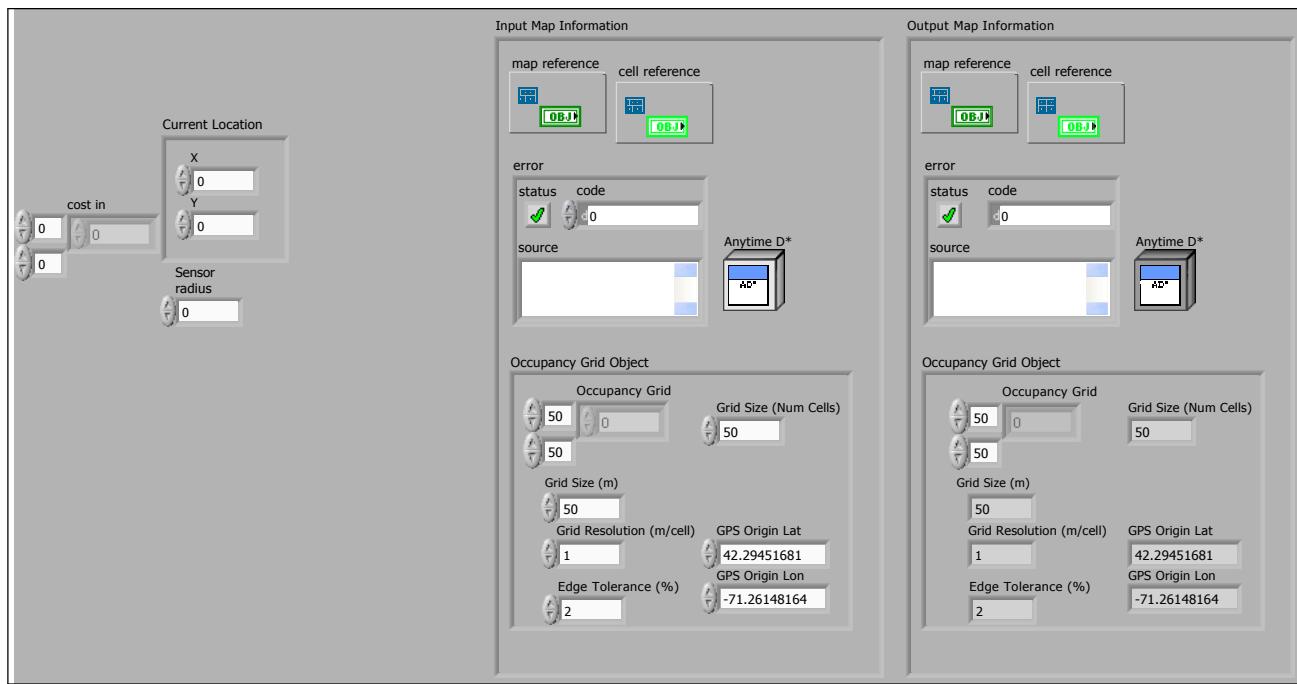


Connector Pane

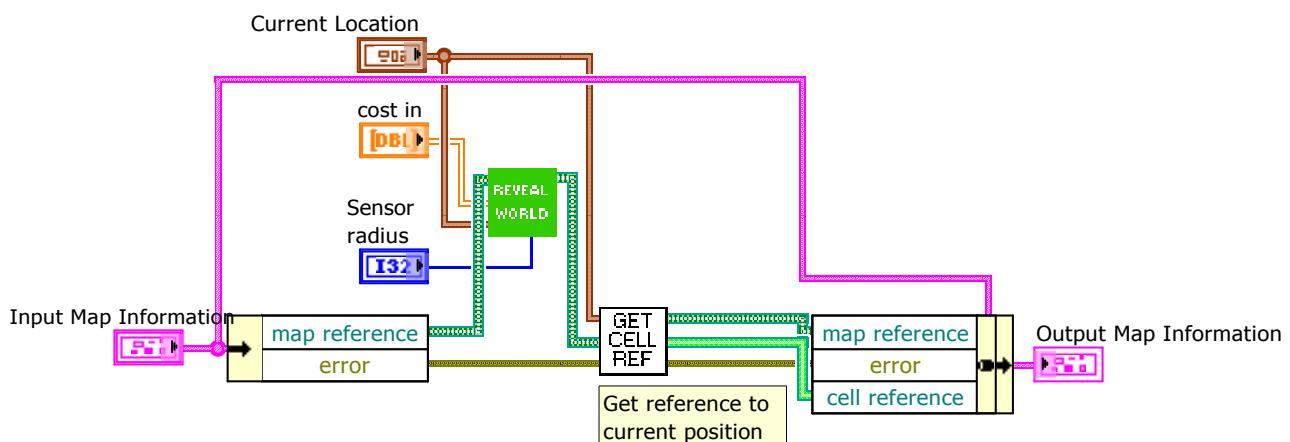
Update World.vi

Updates the grid costs and gets a cell reference to your current position

Front Panel



Block Diagram



Connector Pane

Vehicle State.ctl

Front Panel

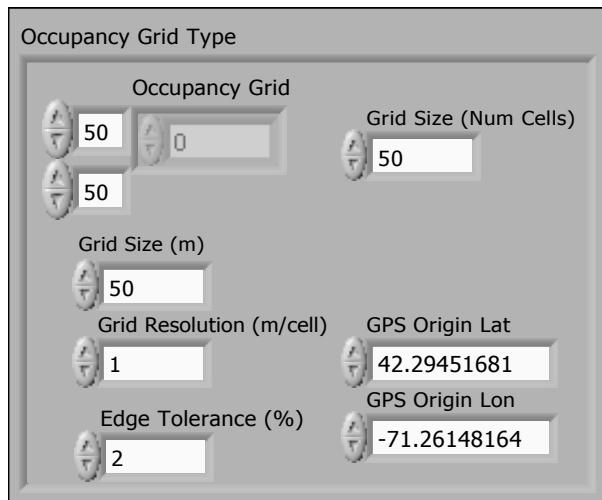


Block Diagram

Connector Pane

OccupancyGrid.ctl

Front Panel

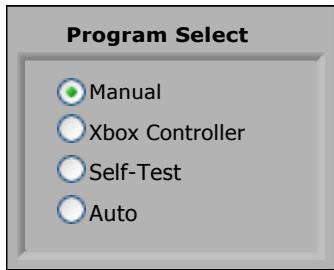


Block Diagram

Connector Pane

ProgramSelectionControl.ctl

Front Panel



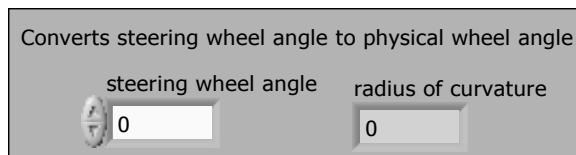
Block Diagram

Connector Pane

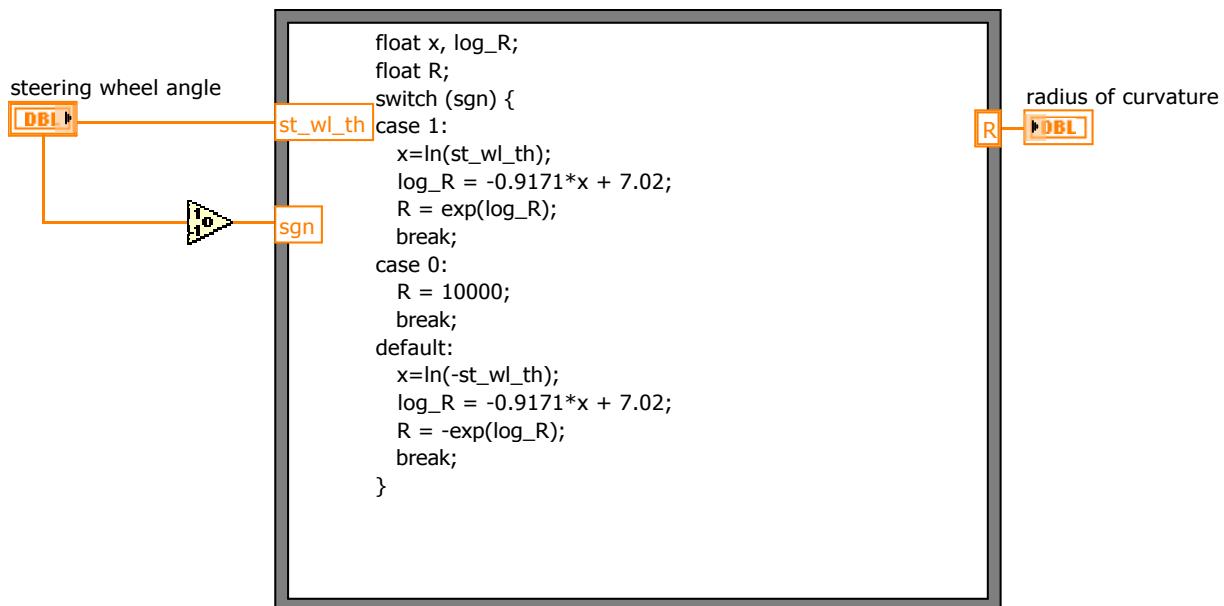
Convert StrWheel to Wheel Angle.vi

Converts steering wheel angle to physical wheel angle

Front Panel



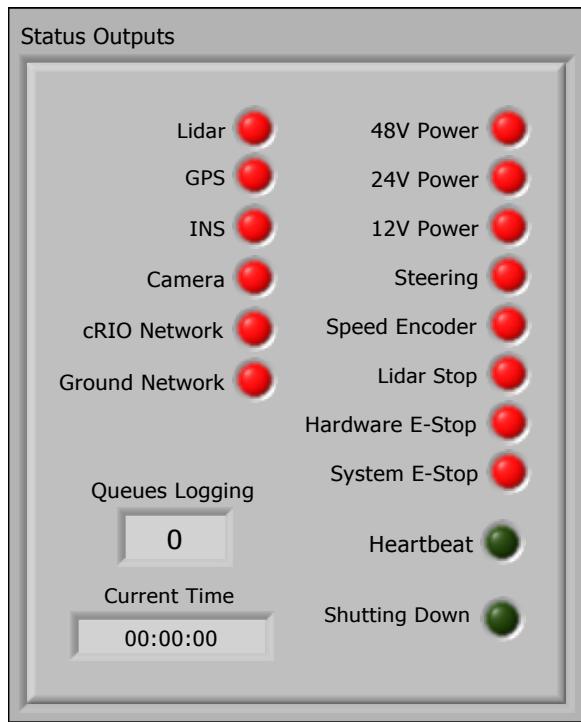
Block Diagram



Connector Pane

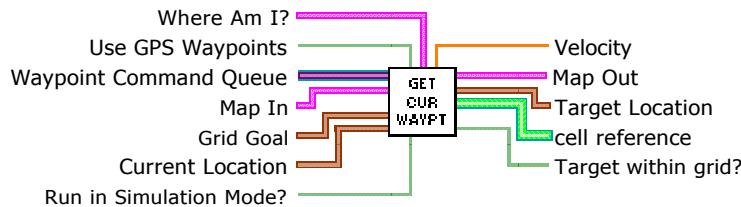
StatusOutputs.ctl

Front Panel

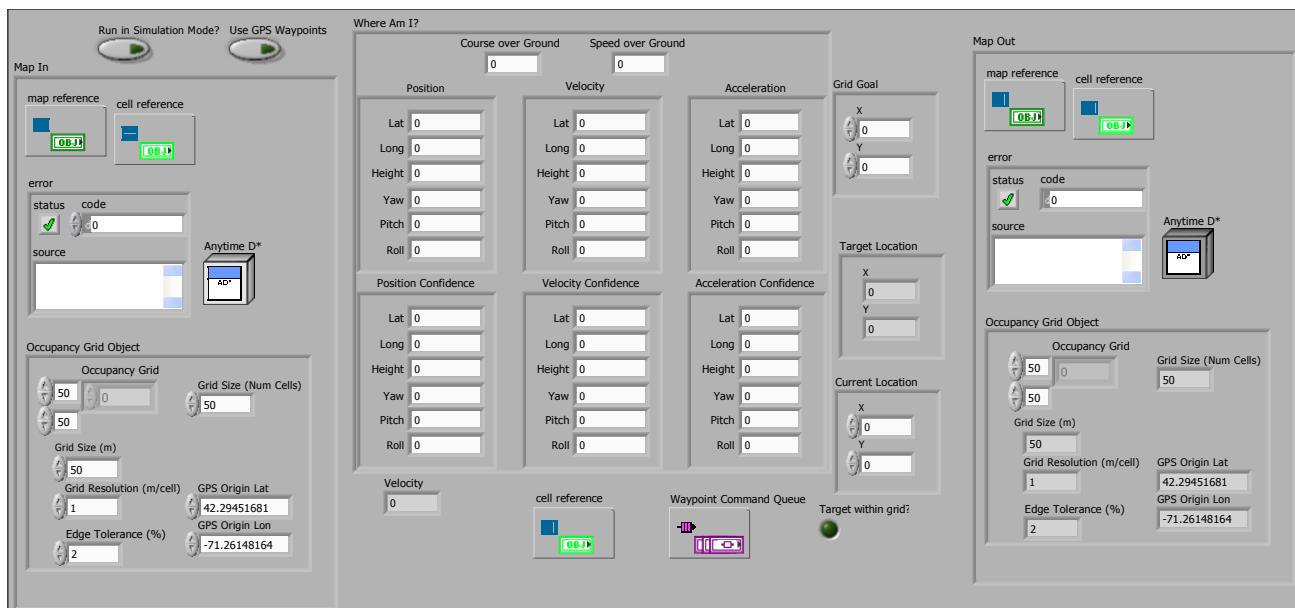


Block Diagram

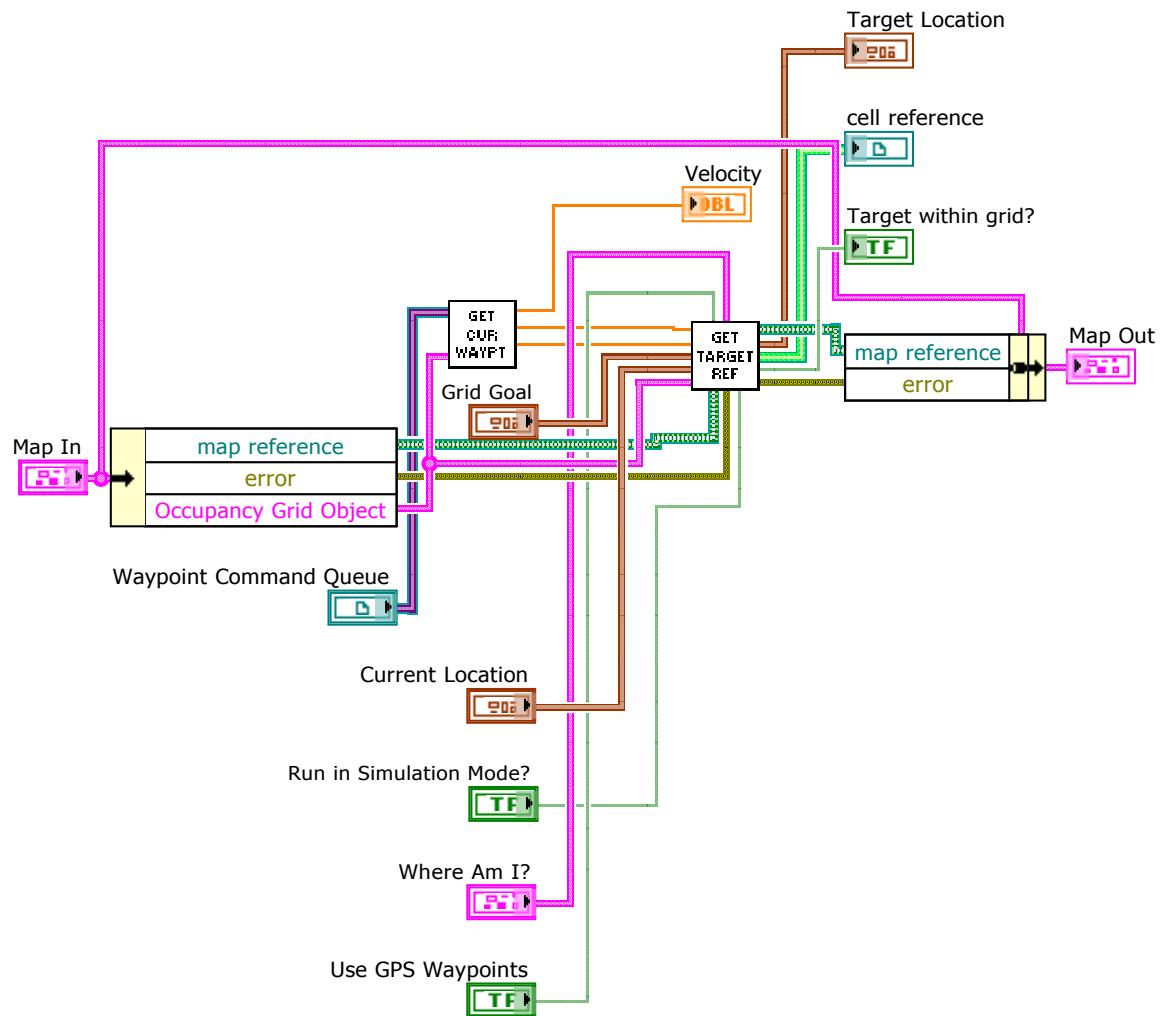
Connector Pane

Get Current Waypoint Reference.vi

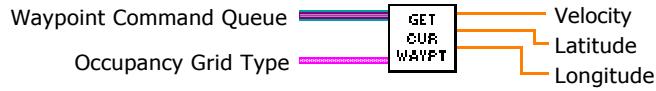
Front Panel



Block Diagram

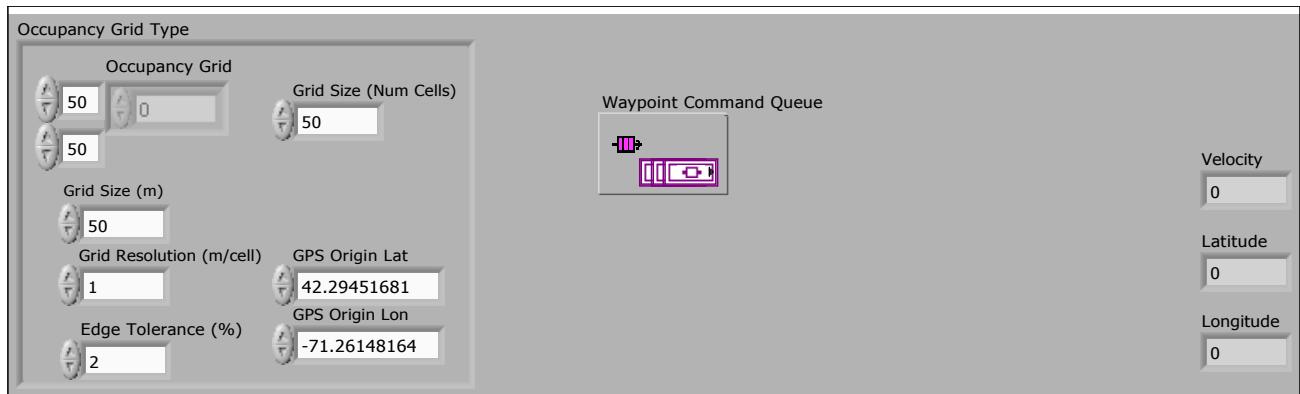


Connector Pane

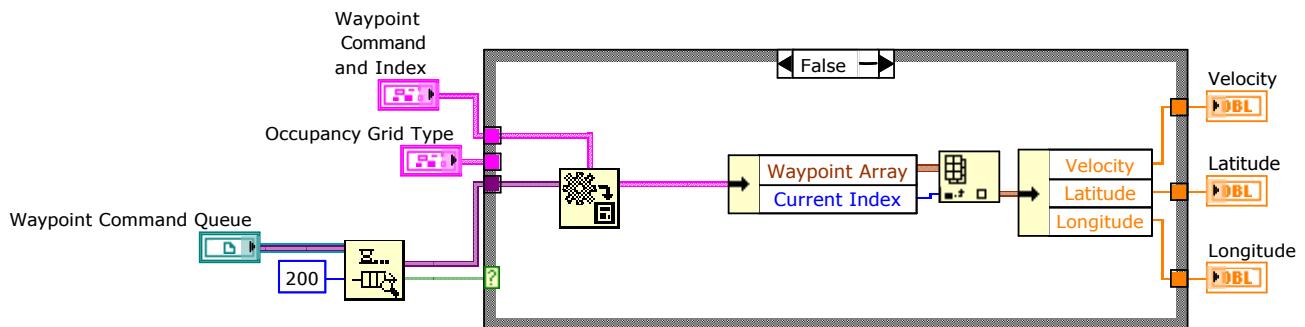
Get Current Waypoint.vi

Gets the current waypoint from the waypoint queue

Front Panel



Block Diagram

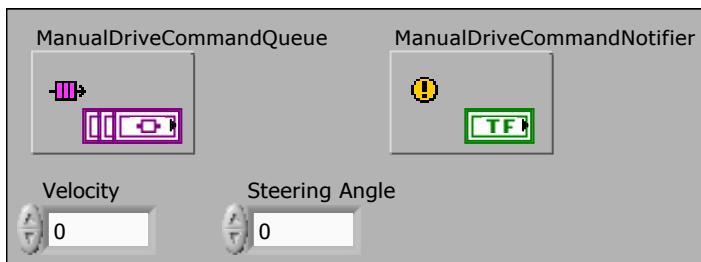


Connector Pane

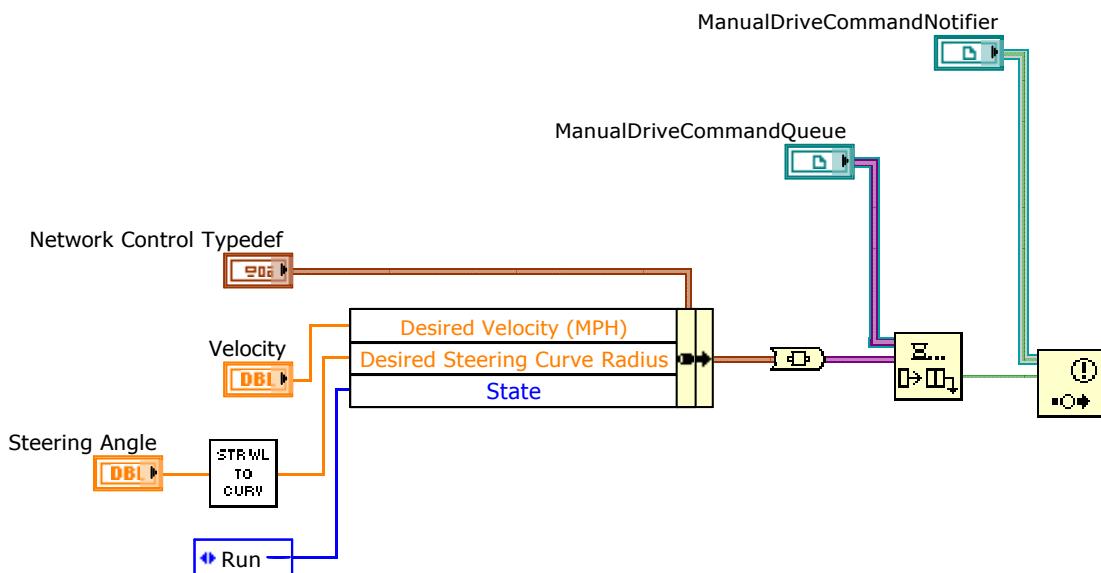
EnqueueDriveCommand.vi

GUI function that enqueues the manual control drive command.

Front Panel



Block Diagram



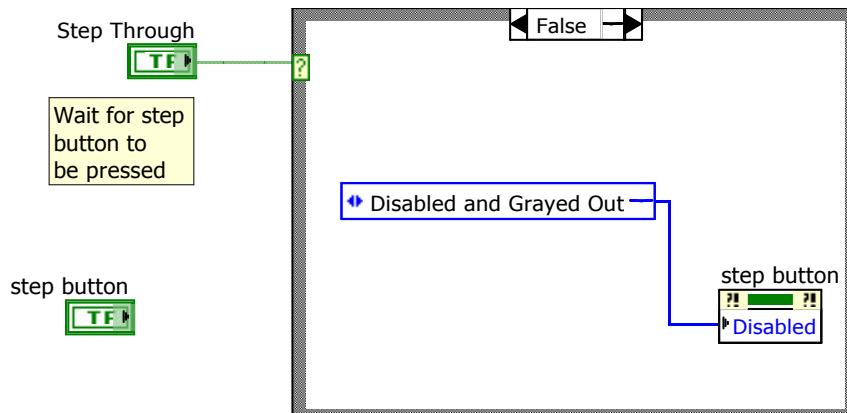
Connector Pane

Step Through Grid.vi

Front Panel



Block Diagram



Connector Pane

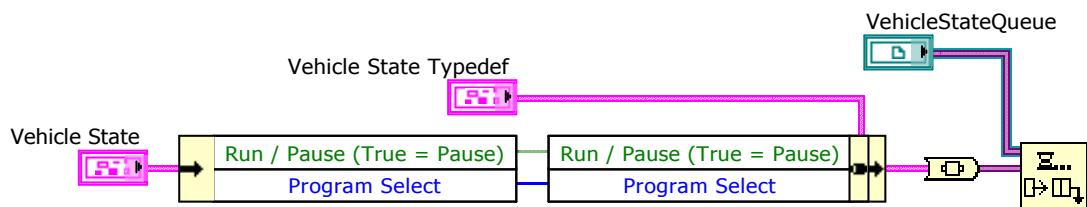
EnqueueVehicleState.vi

GUI function that enqueues the GUI's state (Run/Pause, program to run).

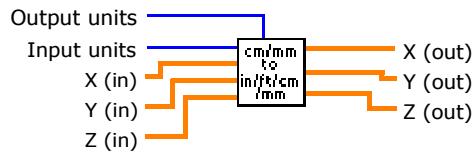
Front Panel



Block Diagram



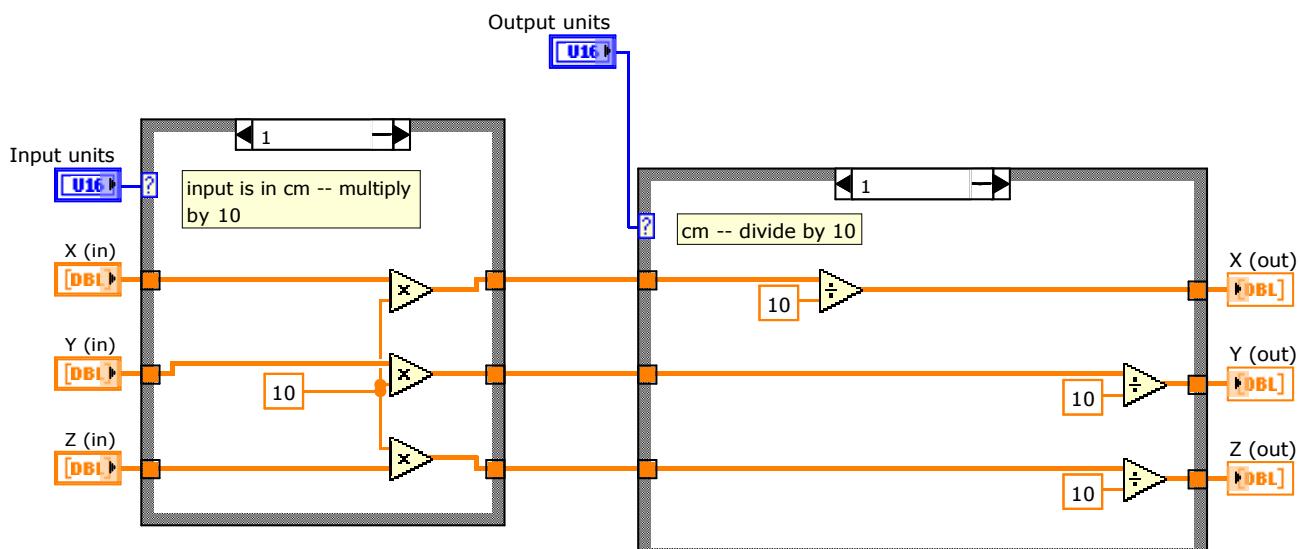
Connector Pane

LidarConvertCoordinates.vi

Front Panel



Block Diagram

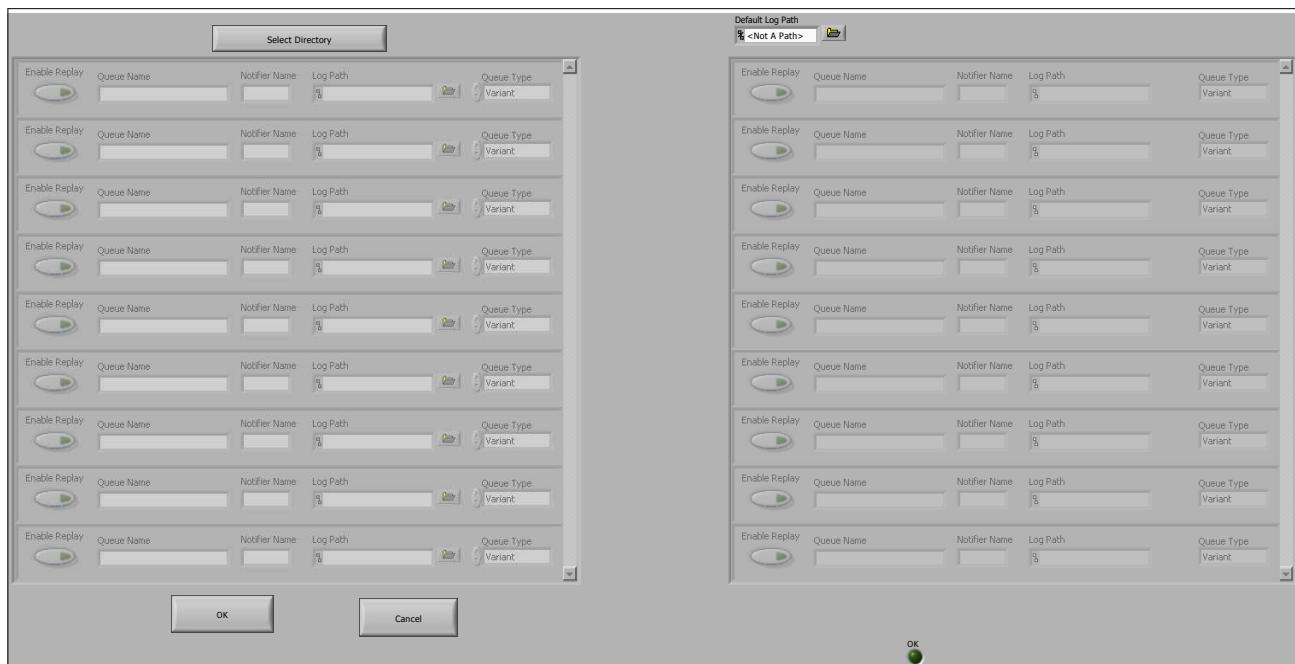


Connector Pane

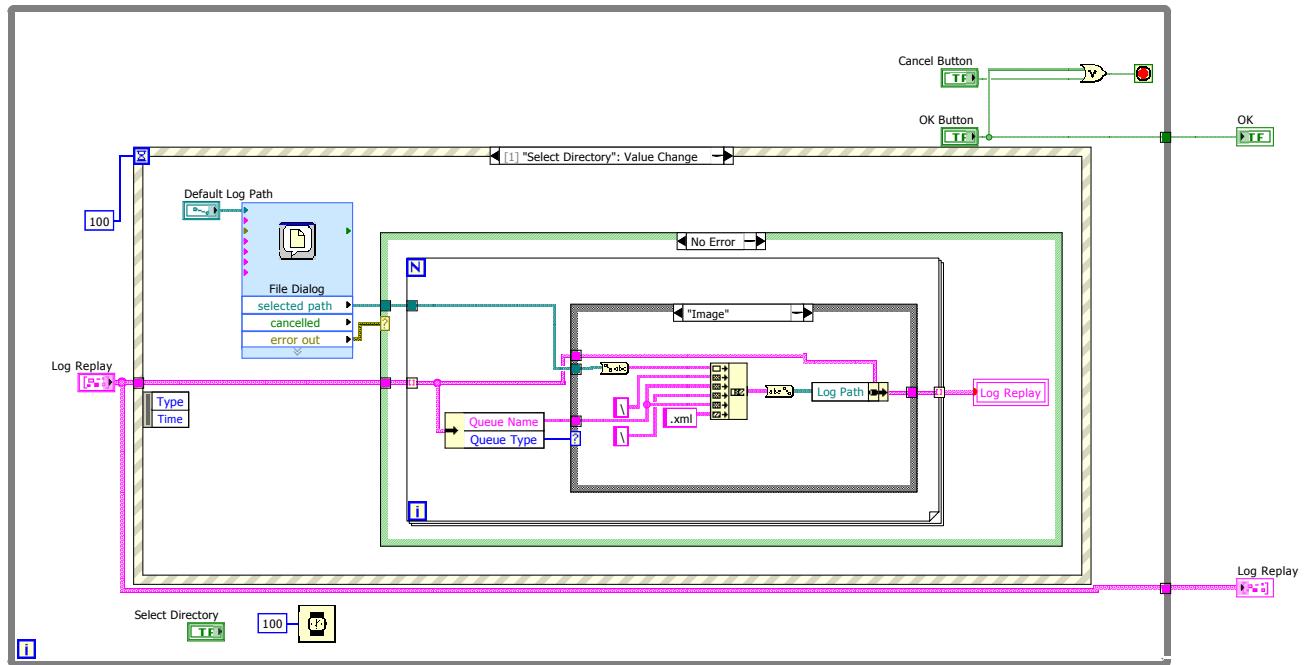
ReplayLogDialog.vi

Replay configuration dialog.

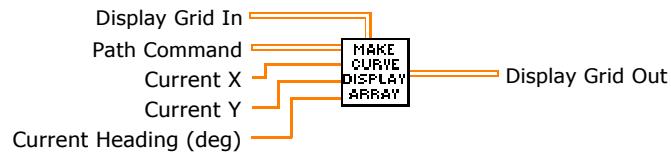
Front Panel



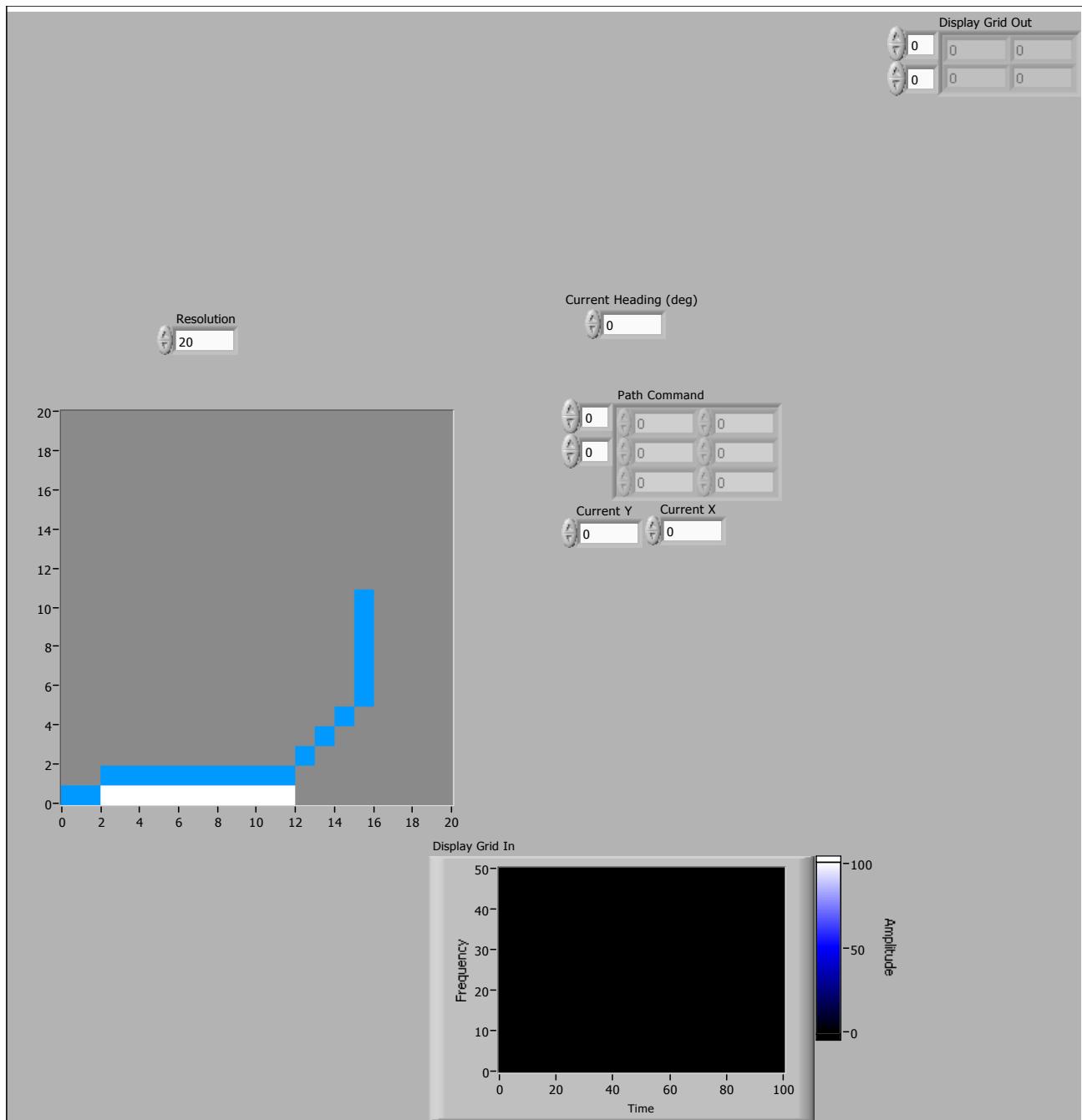
Block Diagram



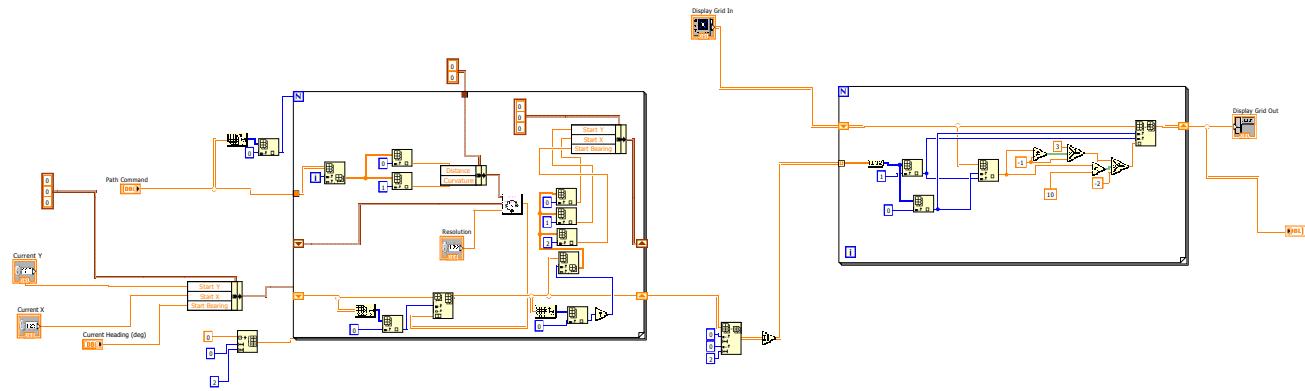
Connector Pane

MakeCurveDisplay.vi

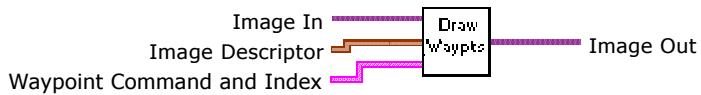
Front Panel



Block Diagram

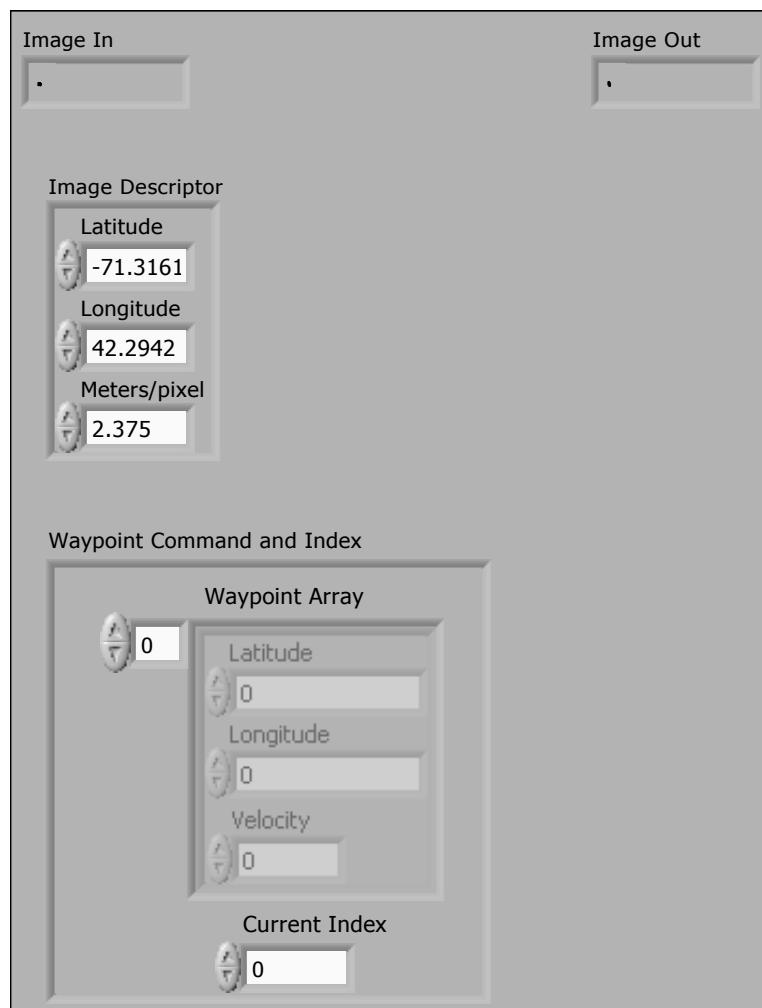


Connector Pane

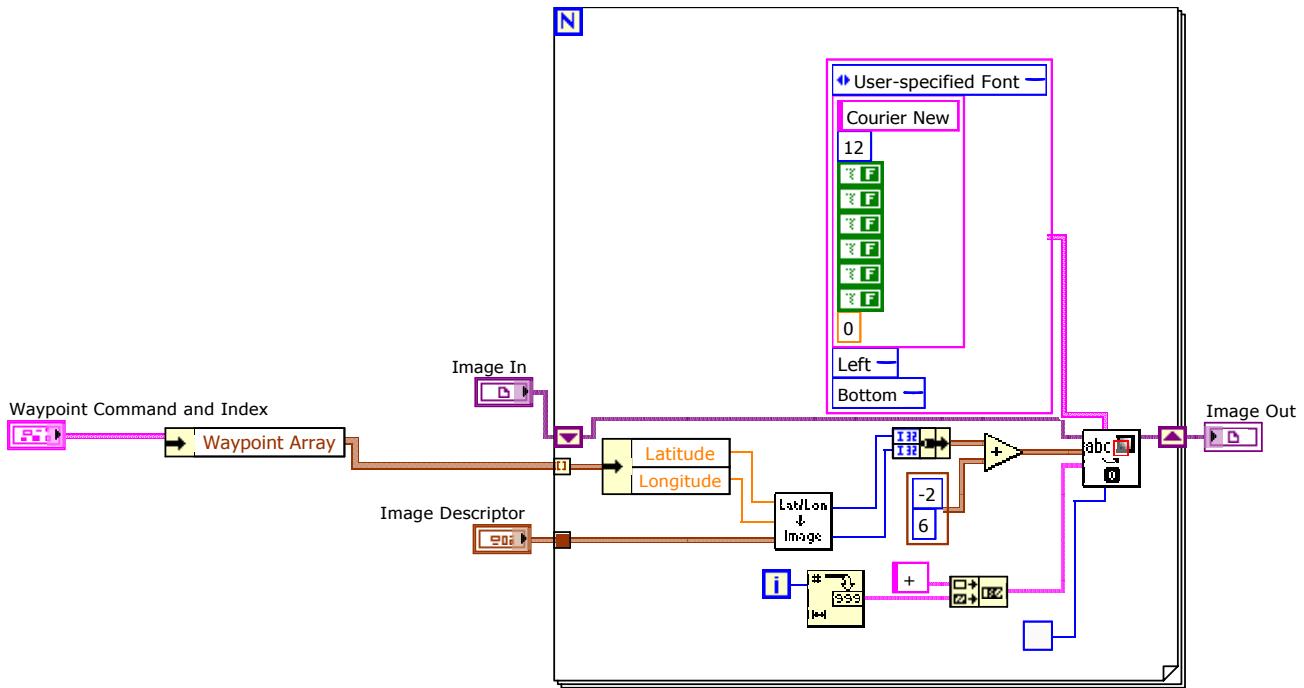
Draw Waypoints.vi

Overlays waypoints onto the map image.

Front Panel



Block Diagram



Connector Pane

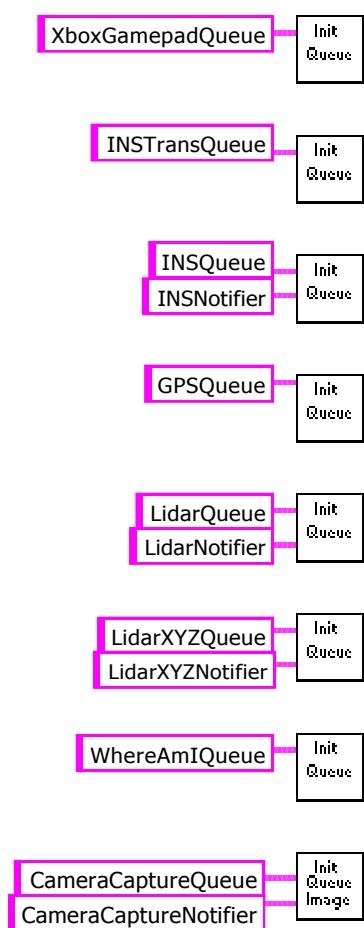
InitSense.vi

Initializes Sense queues.

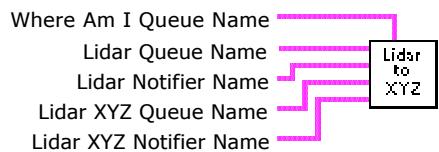
Front Panel



Block Diagram



Connector Pane

LidarToXYZ.vi

Polar to XYZ transform for LIDAR. Also converts to desired units. Use Euler angles to specify the mounting position of the LID.

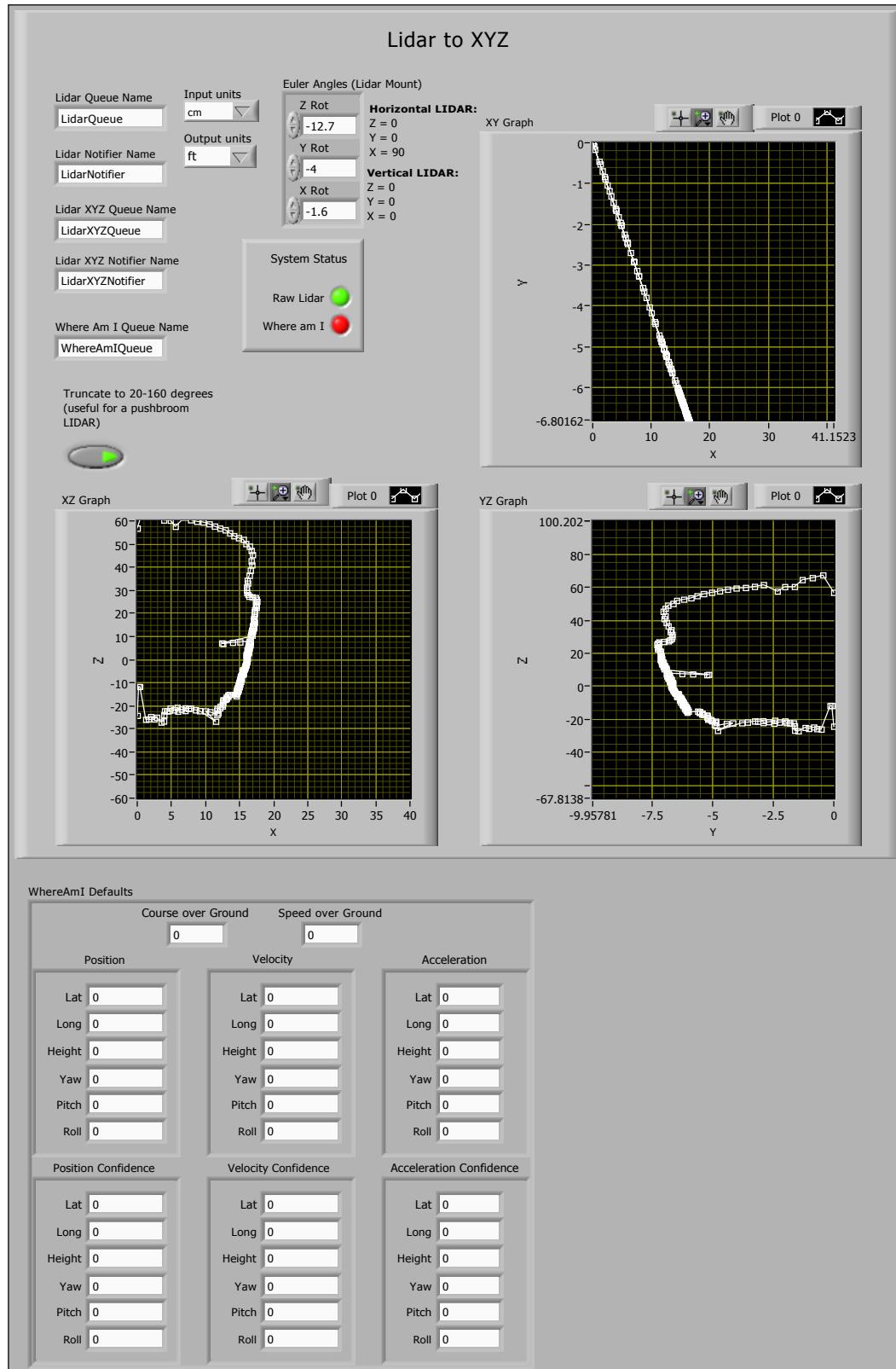
Horizontal LIDAR

Z = 0
Y = 0
X = 90

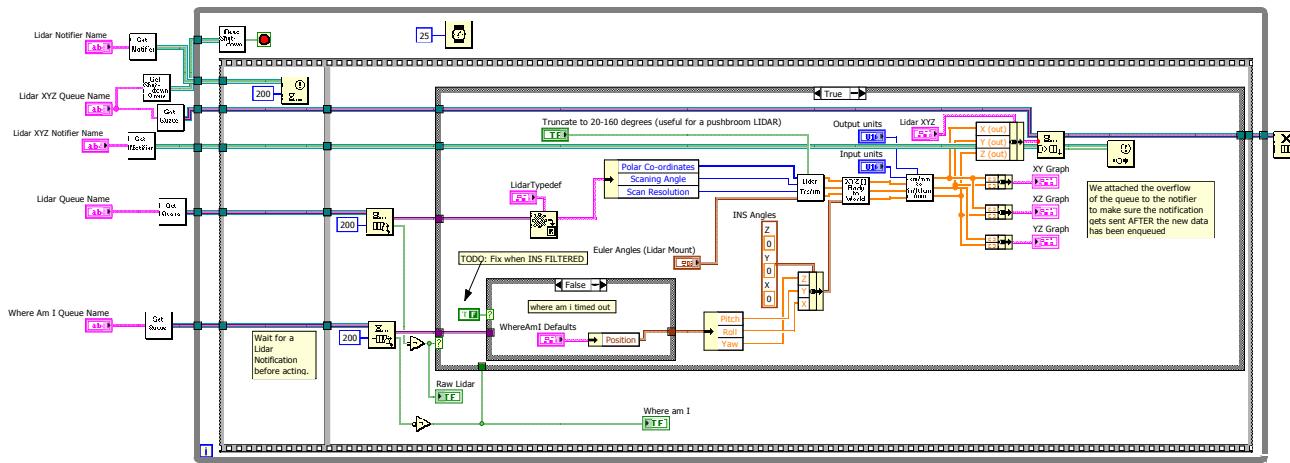
Vertical LIDAR

Z = 0
Y = 0
X = 0

Front Panel



Block Diagram

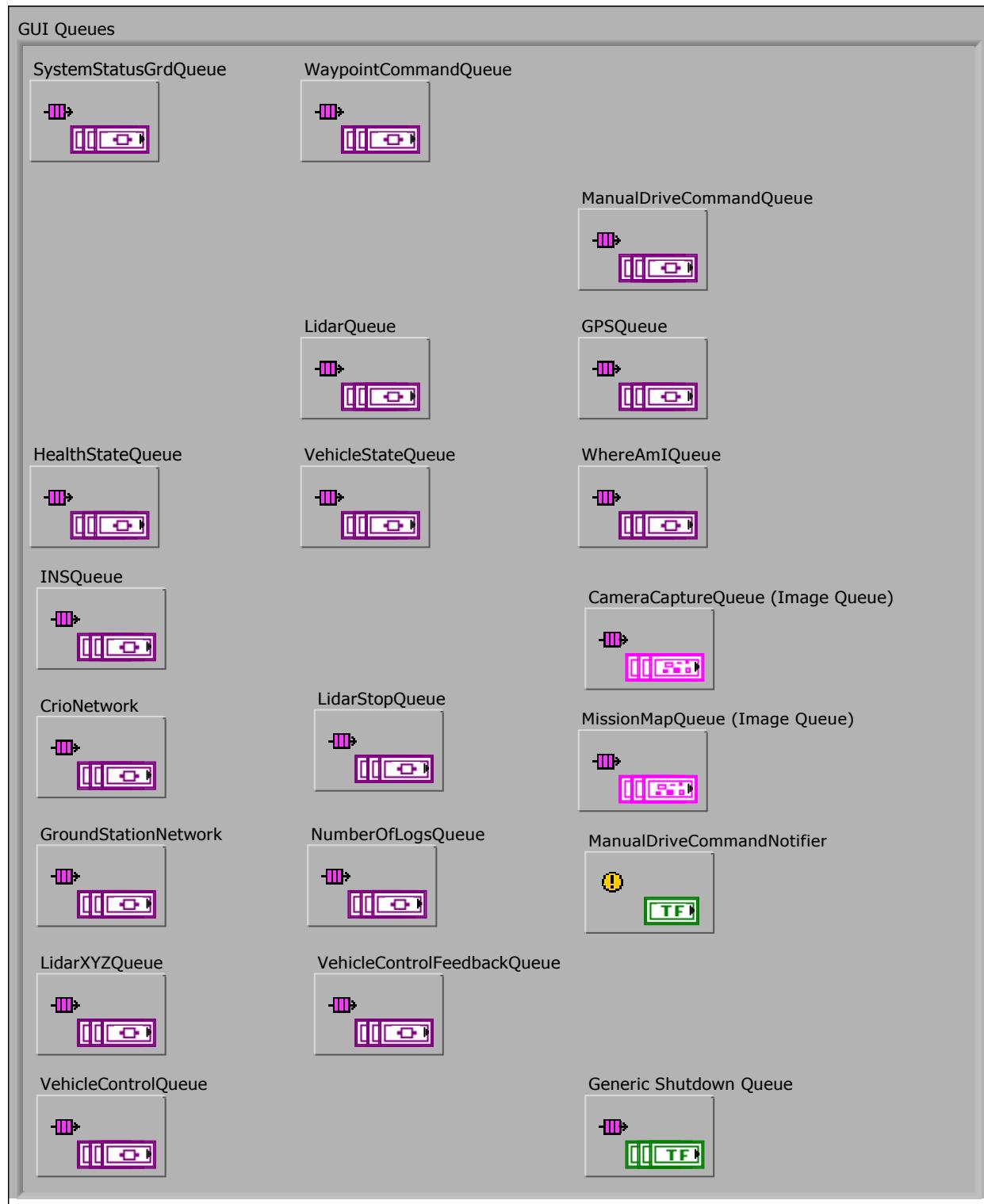


Connector Pane

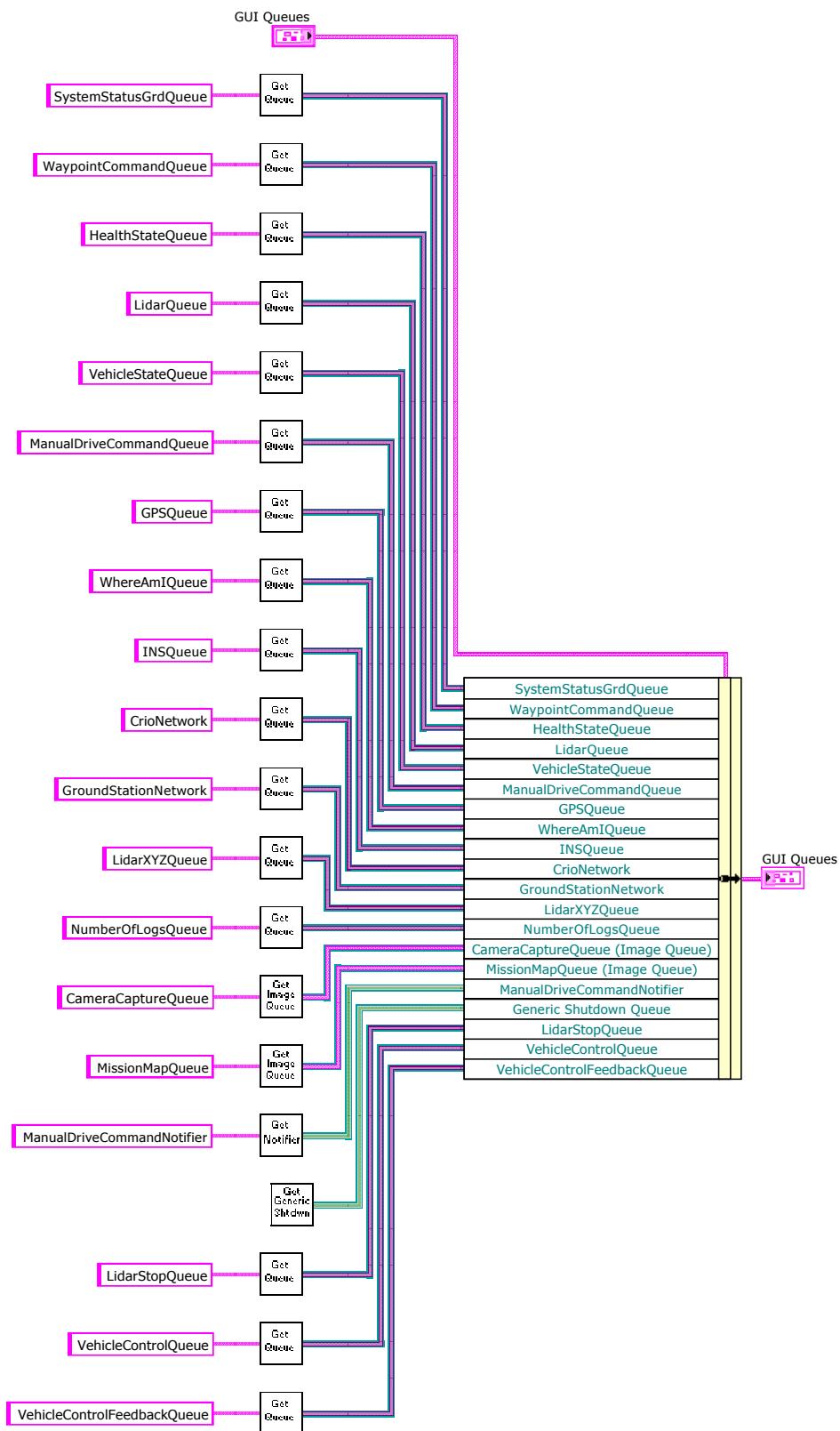
GetGUIQueues.vi

Get the queues that the GUI needs to operate.

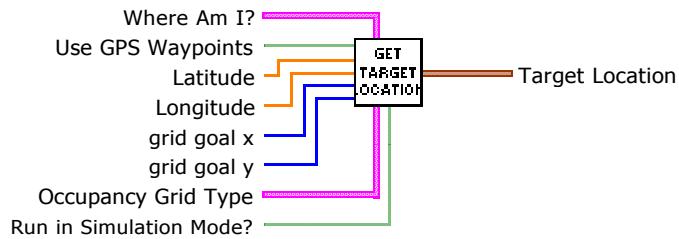
Front Panel



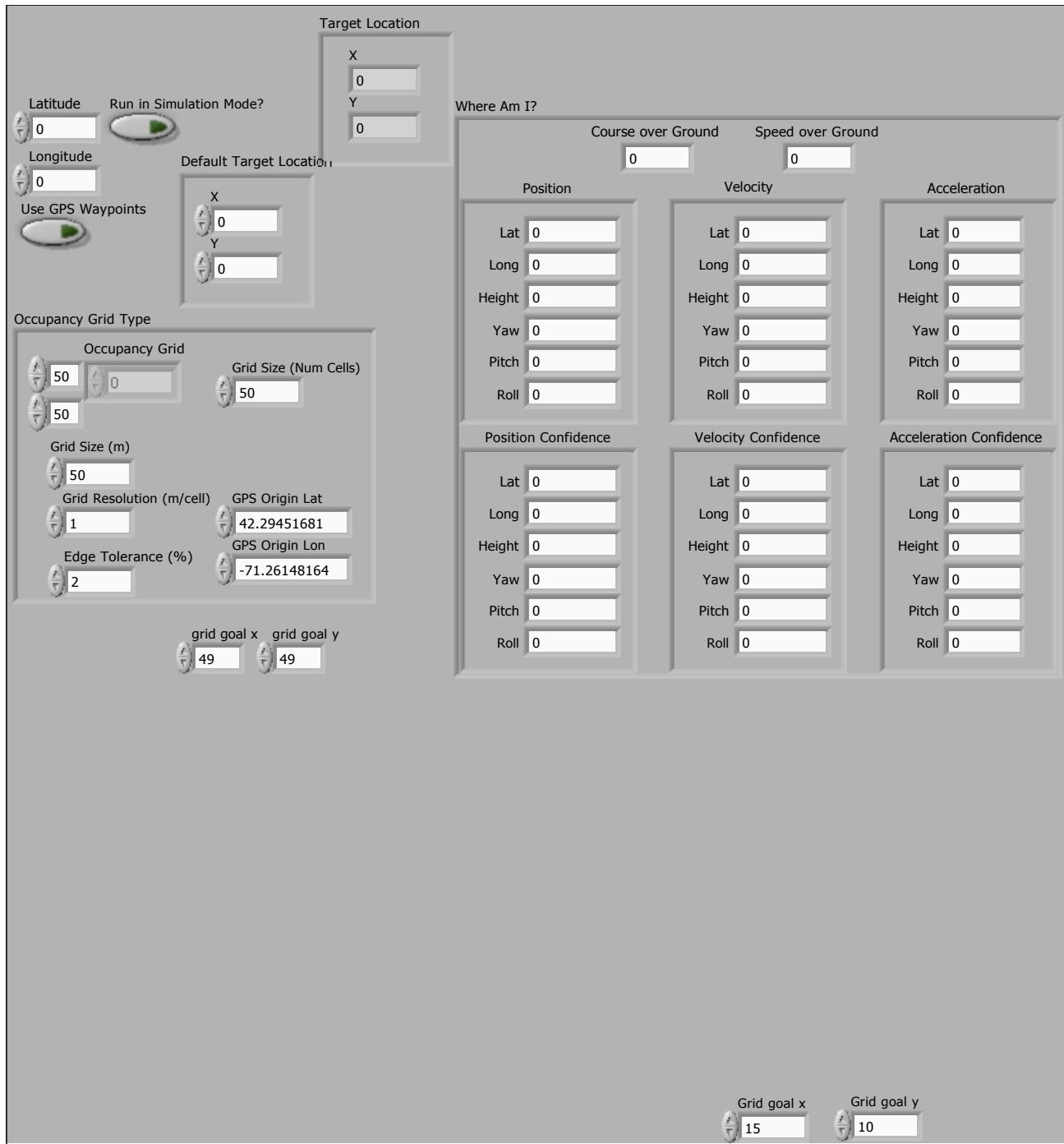
Block Diagram



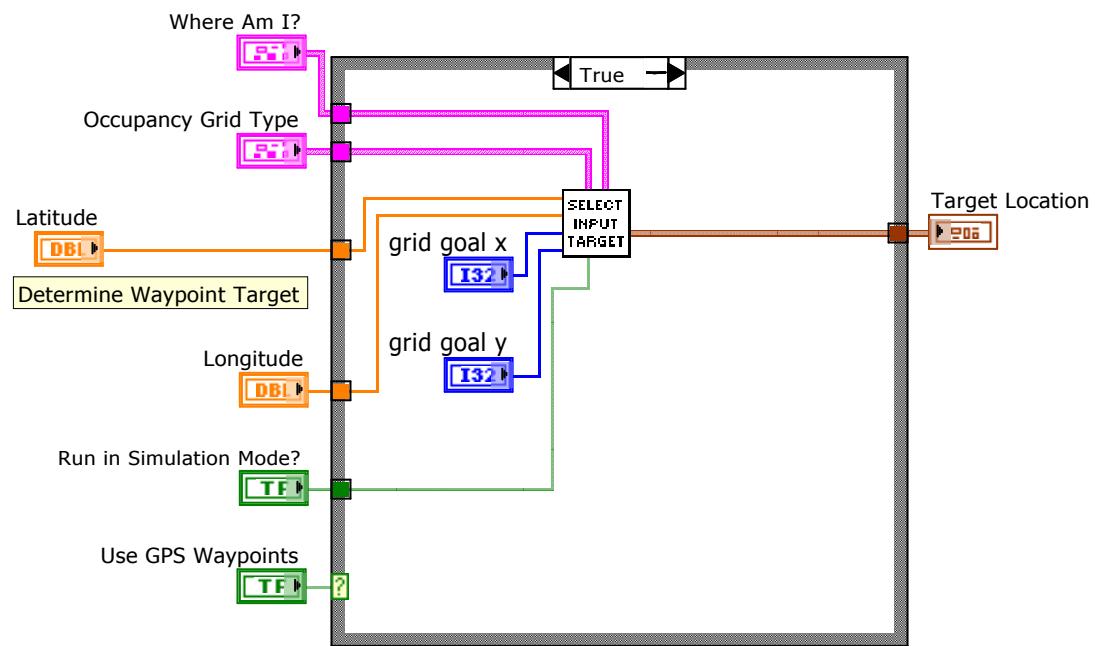
Connector Pane

Get Target Location.vi

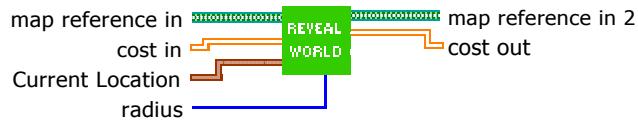
Front Panel



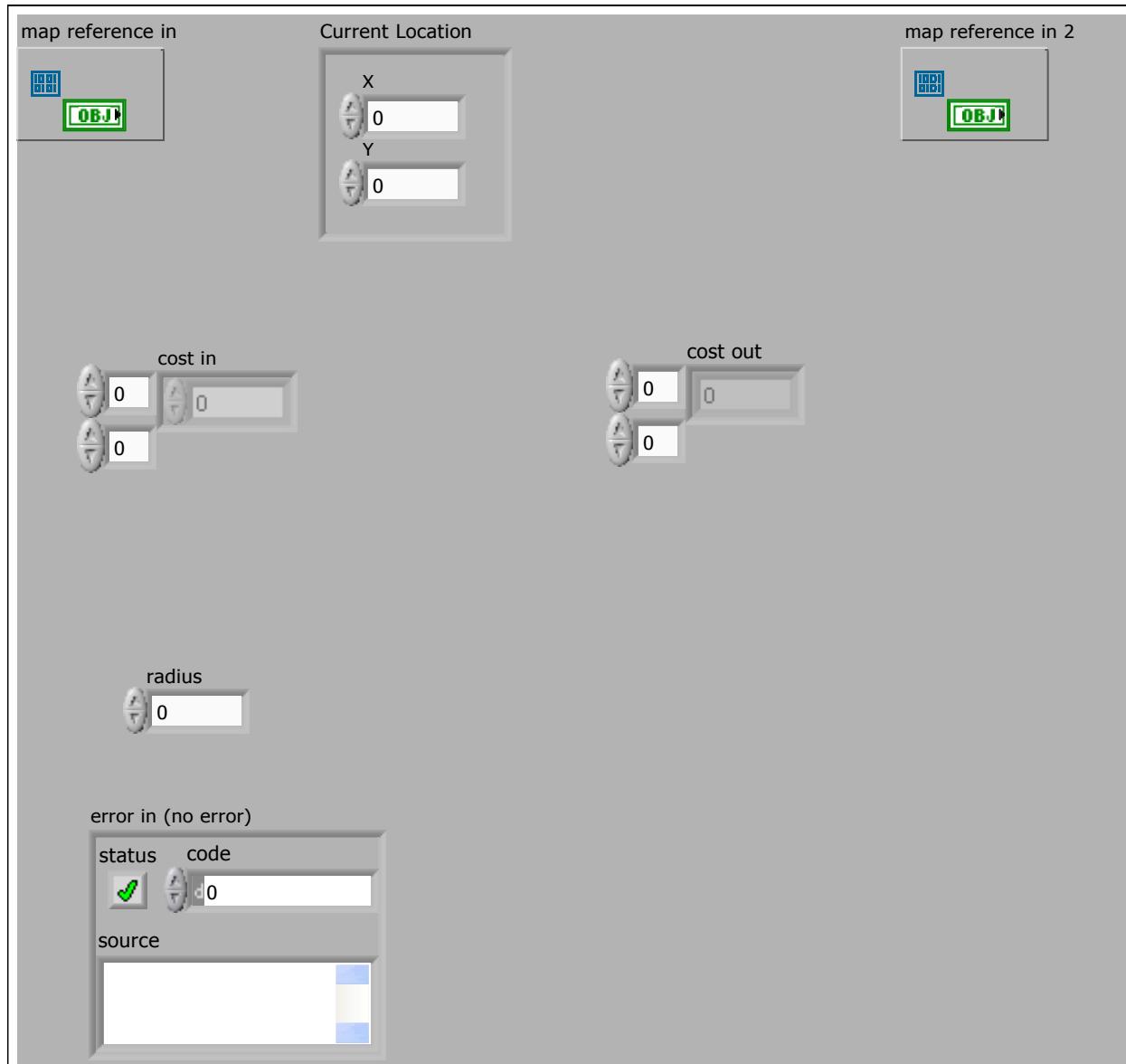
Block Diagram



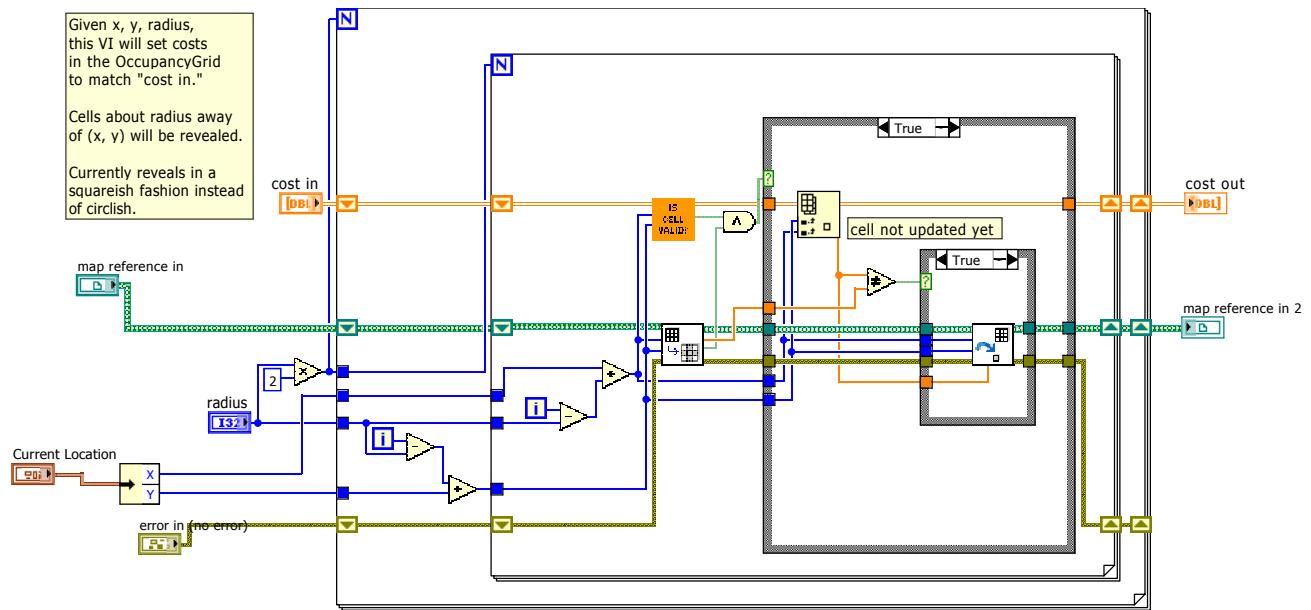
Connector Pane

Reveal World.vi

Front Panel



Block Diagram

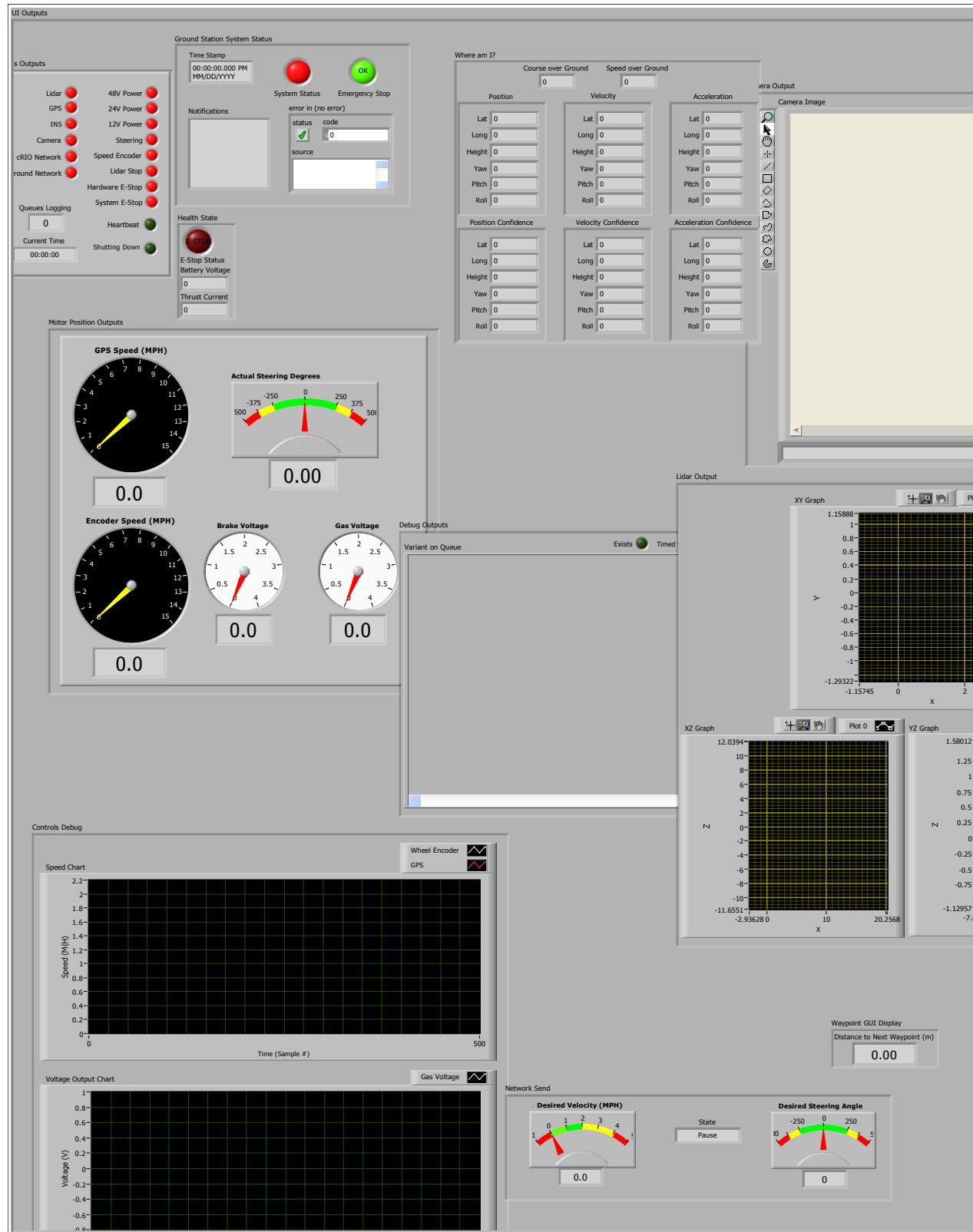


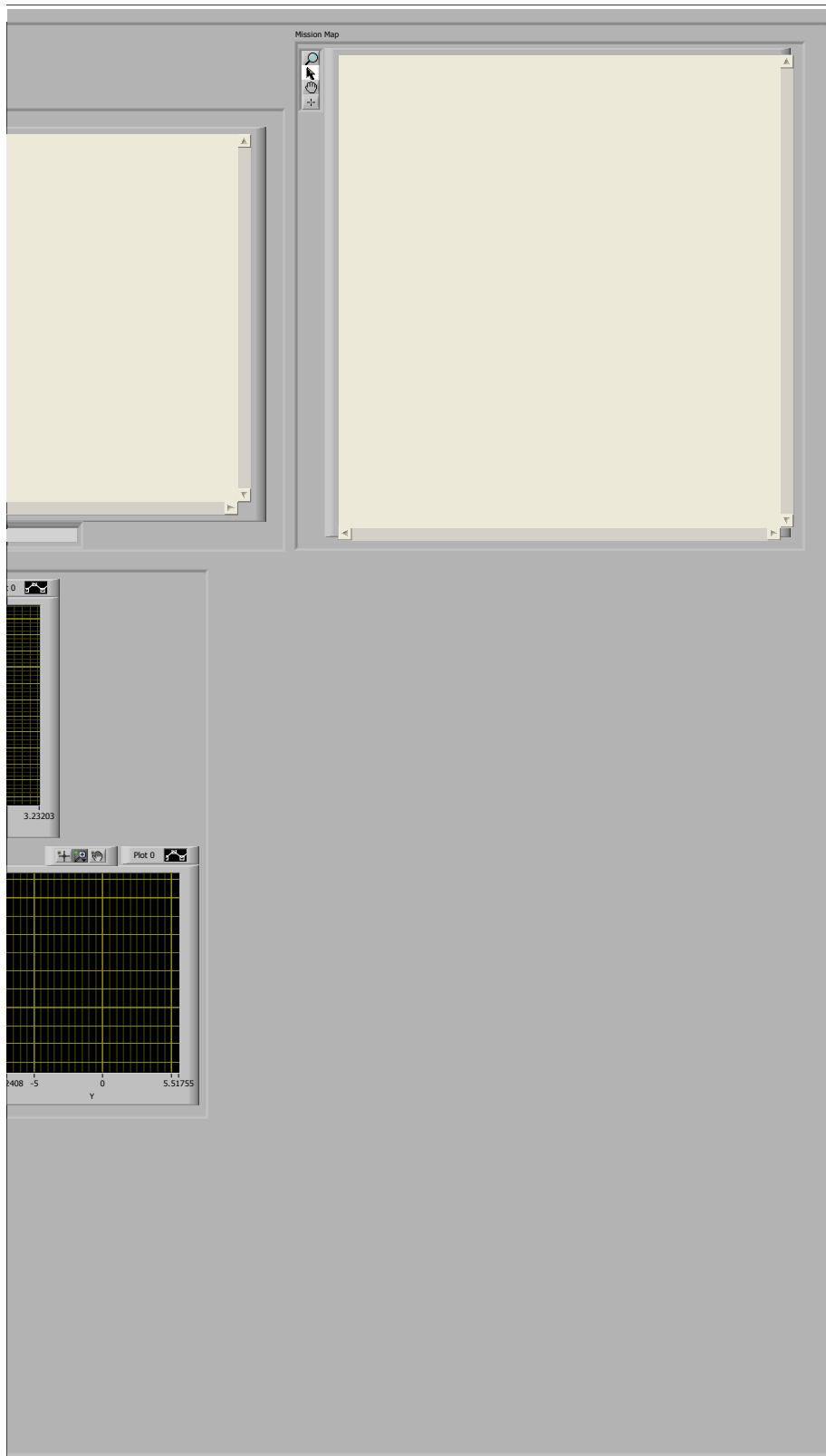
Connector Pane

OutputsUI.ctl



Front Panel





Block Diagram

Connector Pane

InitReplayQueue.vi

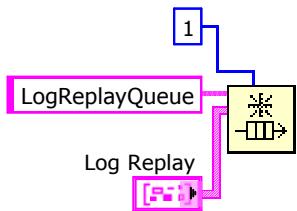
Initializes Logging Replay Queue.

Front Panel



Block Diagram

This queue is the one that is used by the setup dialog to know which queues to replay

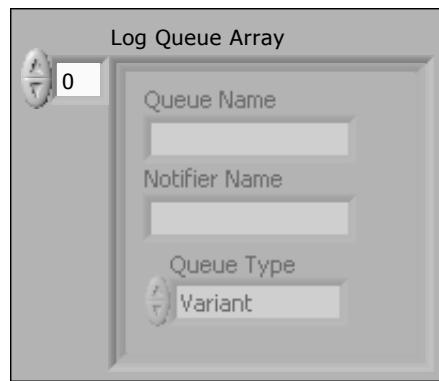


Connector Pane

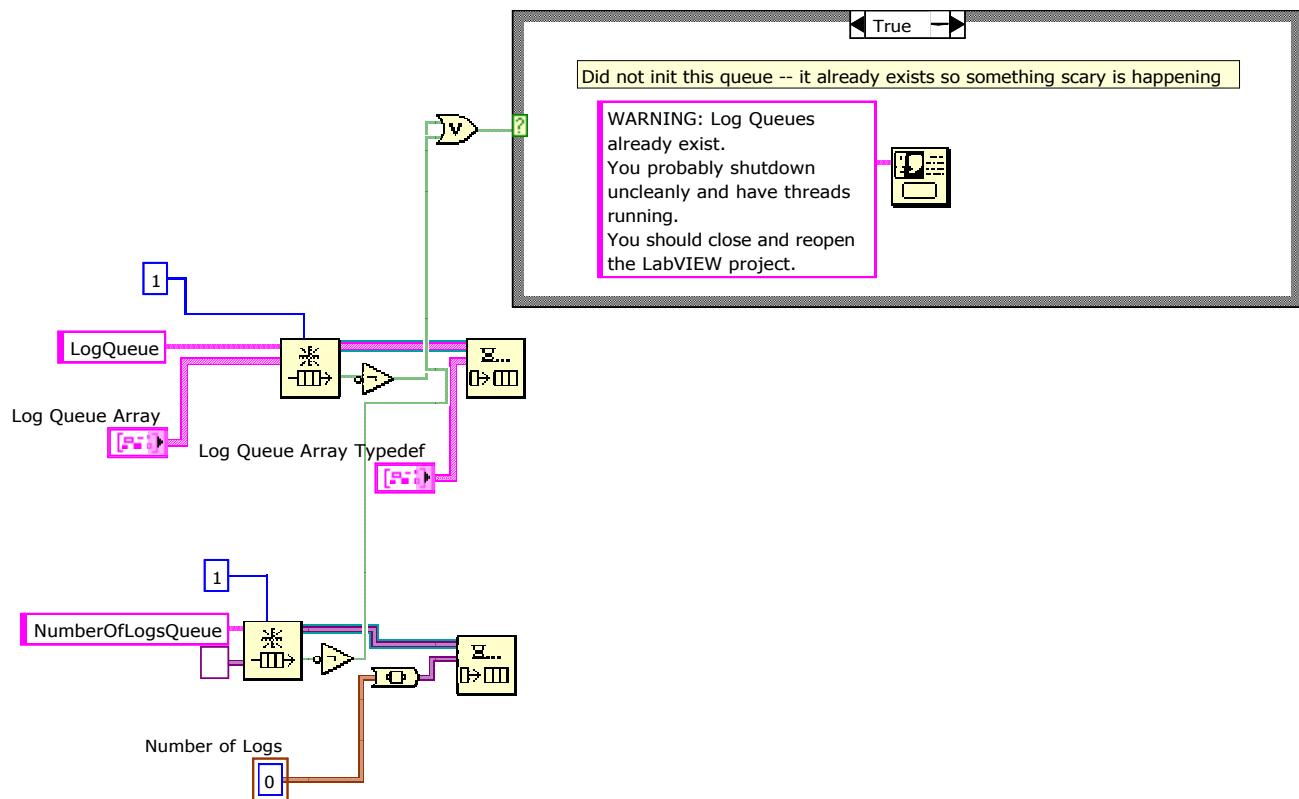
InitLogQueue.vi

Initializes Logging Queue.

Front Panel



Block Diagram

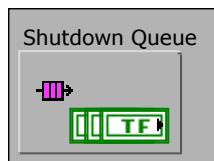


Connector Pane

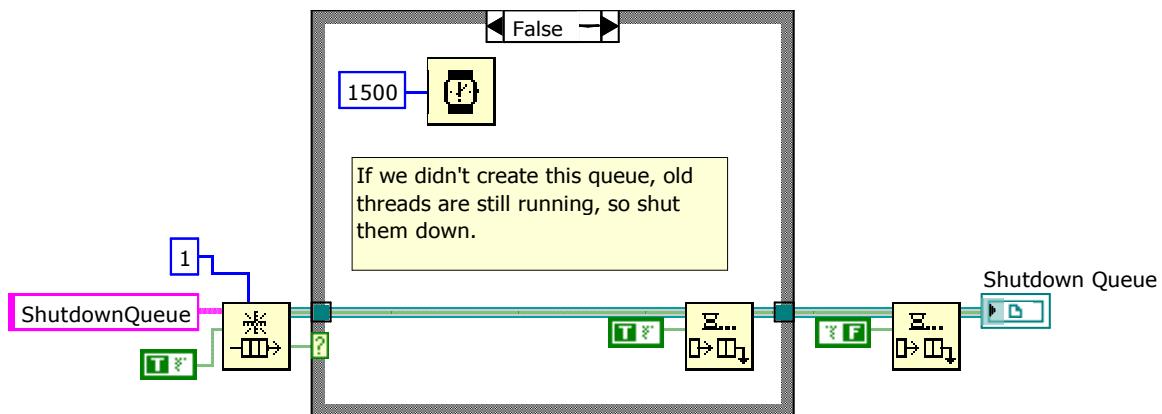
InitGenericShutdown.vi

Initializes the Generic Shutdown Queue.

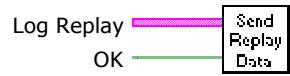
Front Panel



Block Diagram

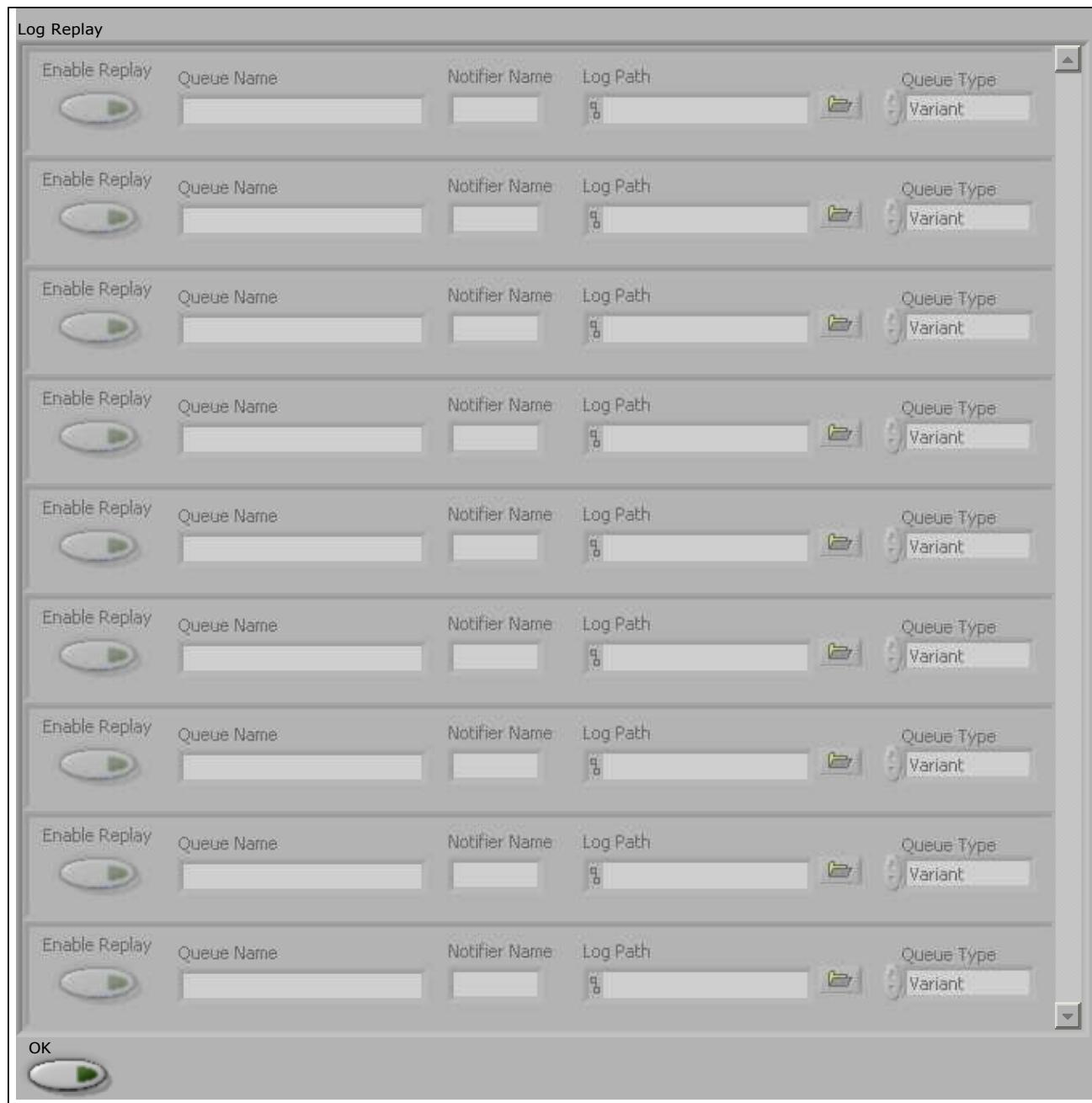


Connector Pane

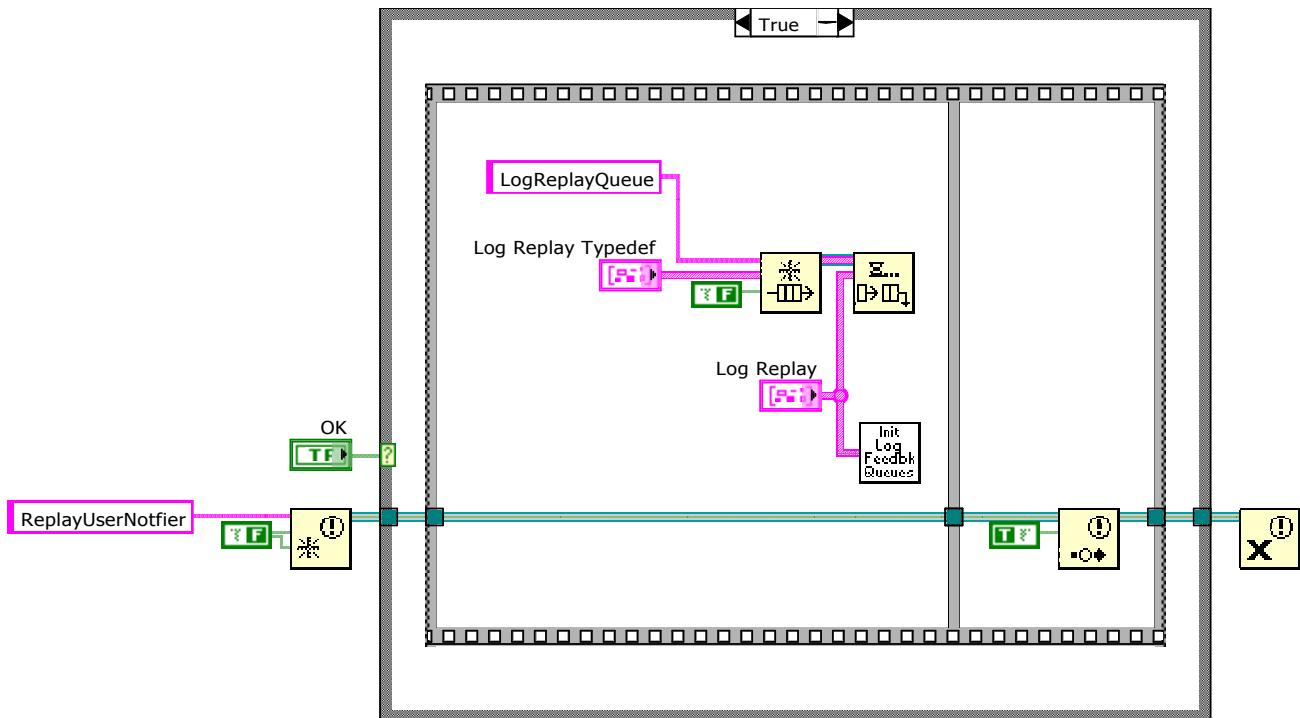
SendReplayData.vi

Sends replay data to the replay thread spawning VI (RunReplayLog.vi).

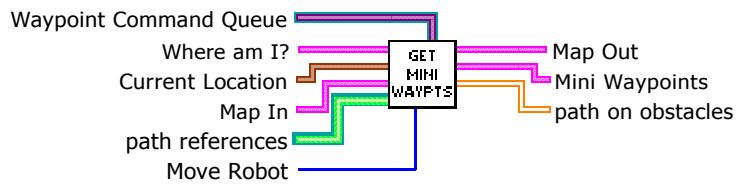
Front Panel



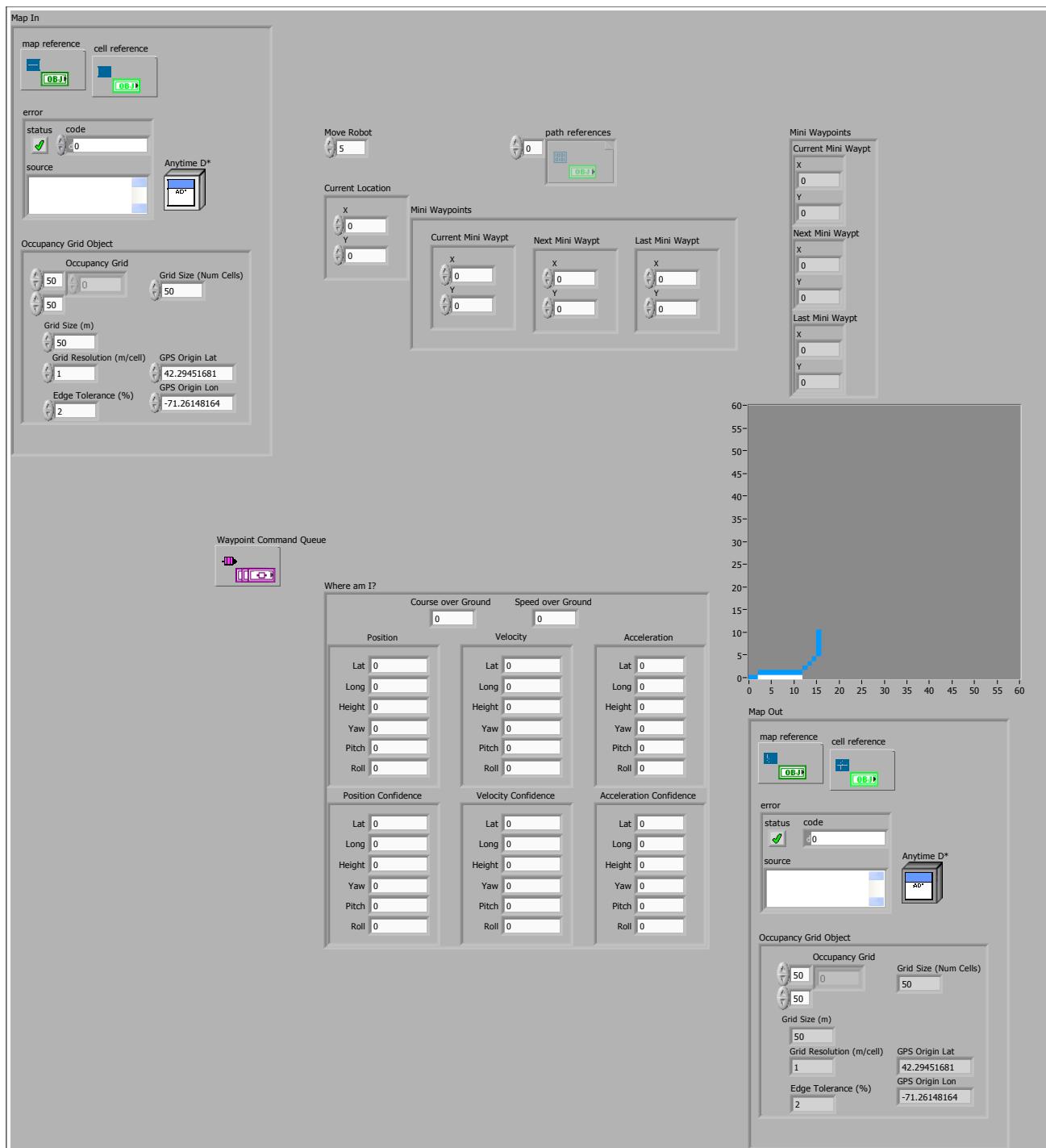
Block Diagram



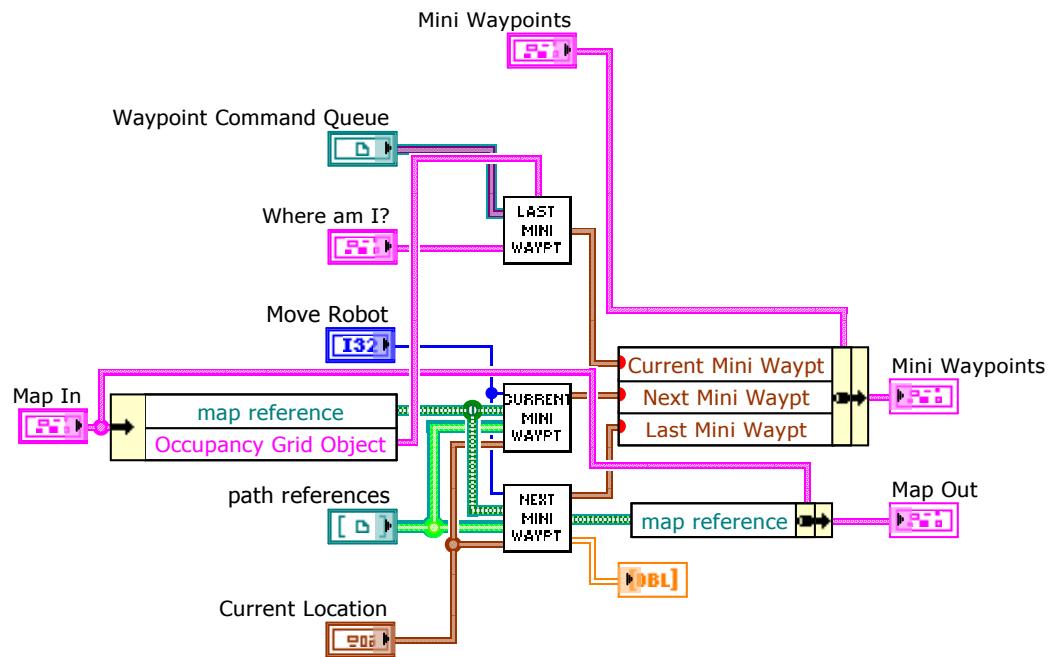
Connector Pane

Get Mini Waypts.vi

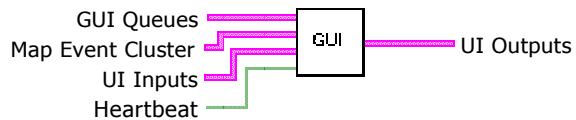
Front Panel



Block Diagram

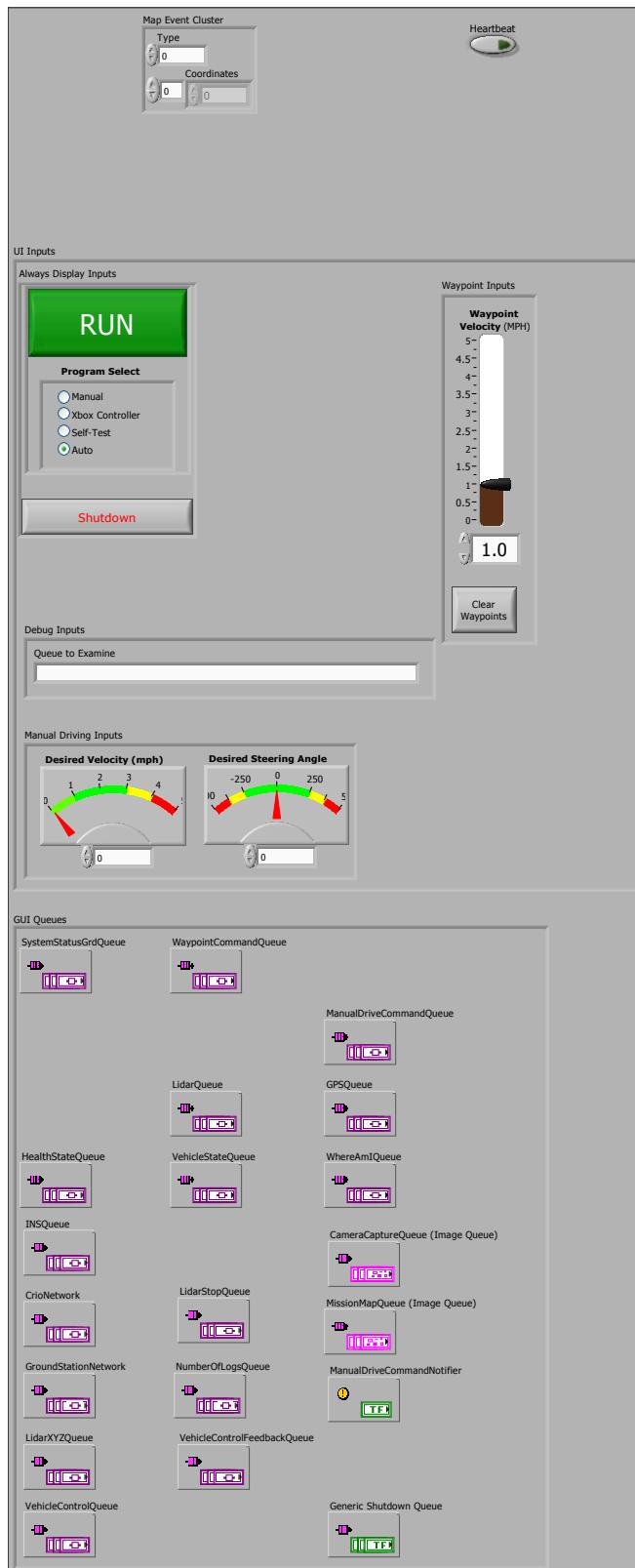


Connector Pane

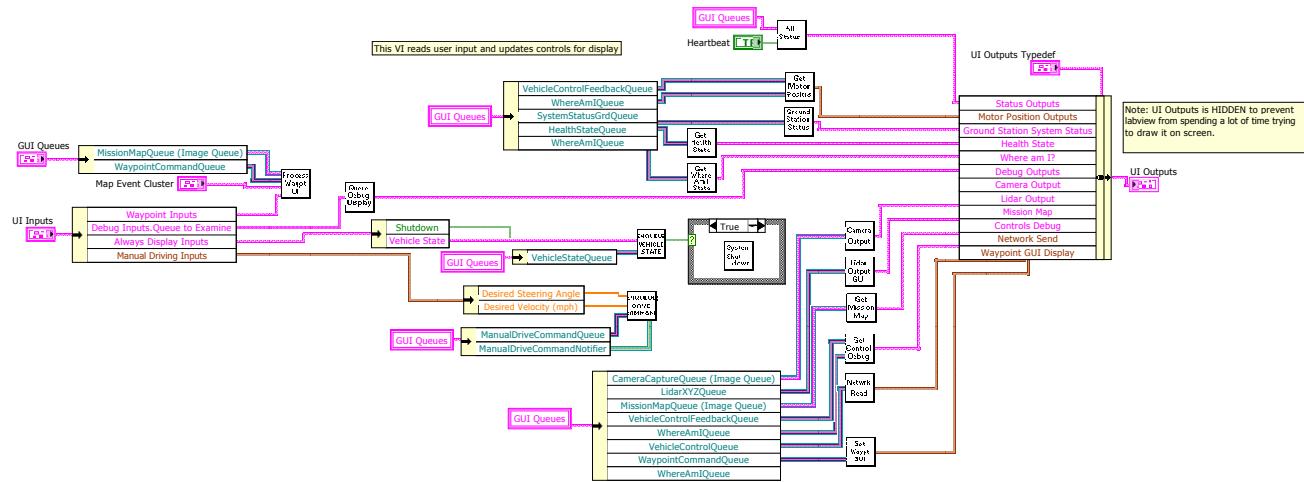
GUI.vi

Manages all GUI inputs and outputs for the system.

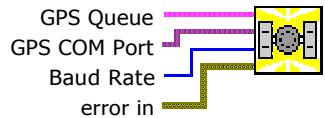
Front Panel



Block Diagram

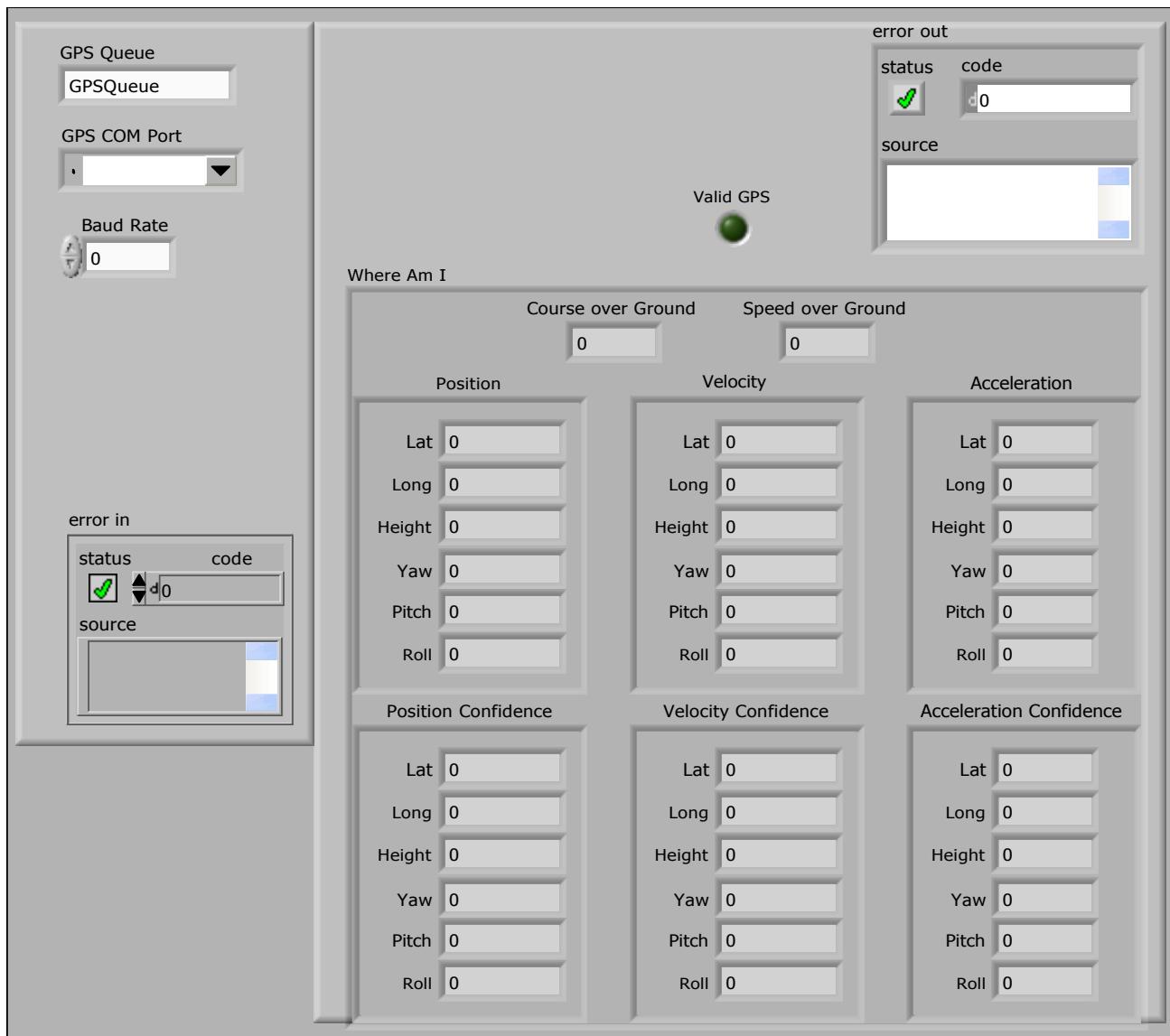


Connector Pane

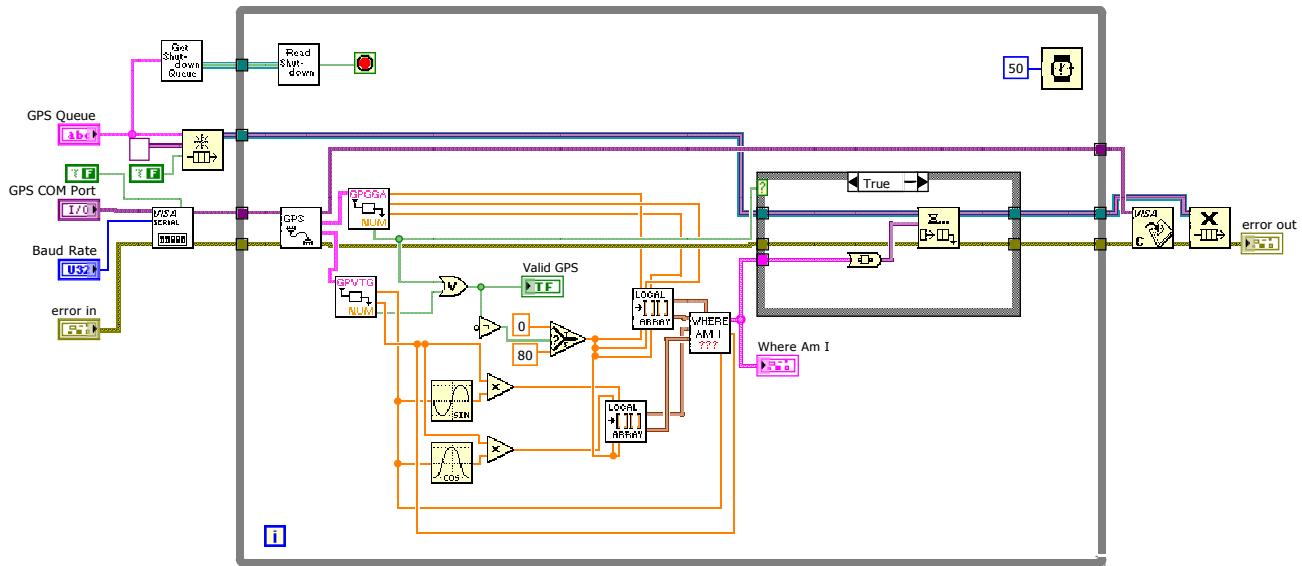
GPS.vi

Set your GPS to output GPGGA NMEA sentences, then plug it into your computer and tell this VI its COM port. This VI will then output a Where Am I block with Lat/Lon/Height and Lat/Lon Velocity, as well as Course over Ground and Speed over Ground.

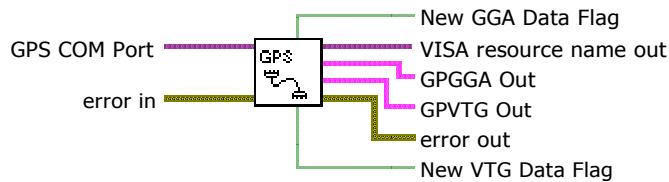
Front Panel



Block Diagram

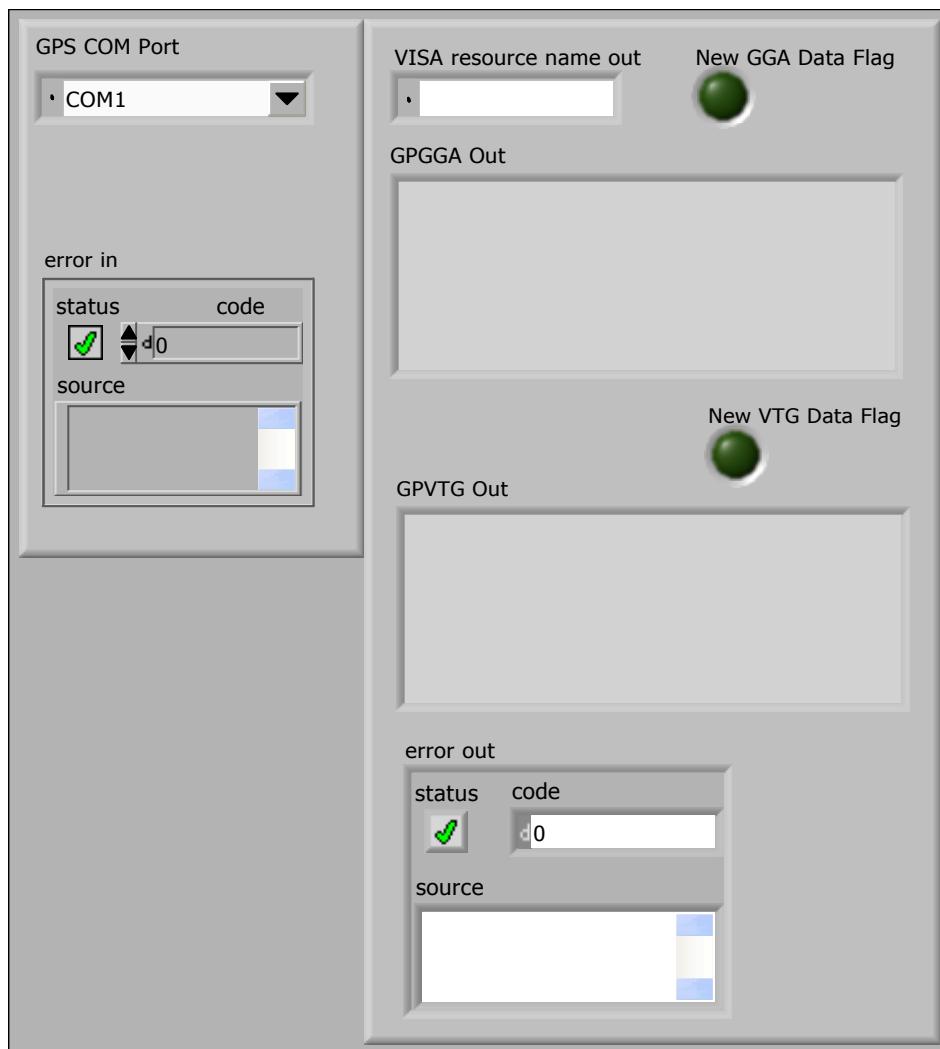


Connector Pane

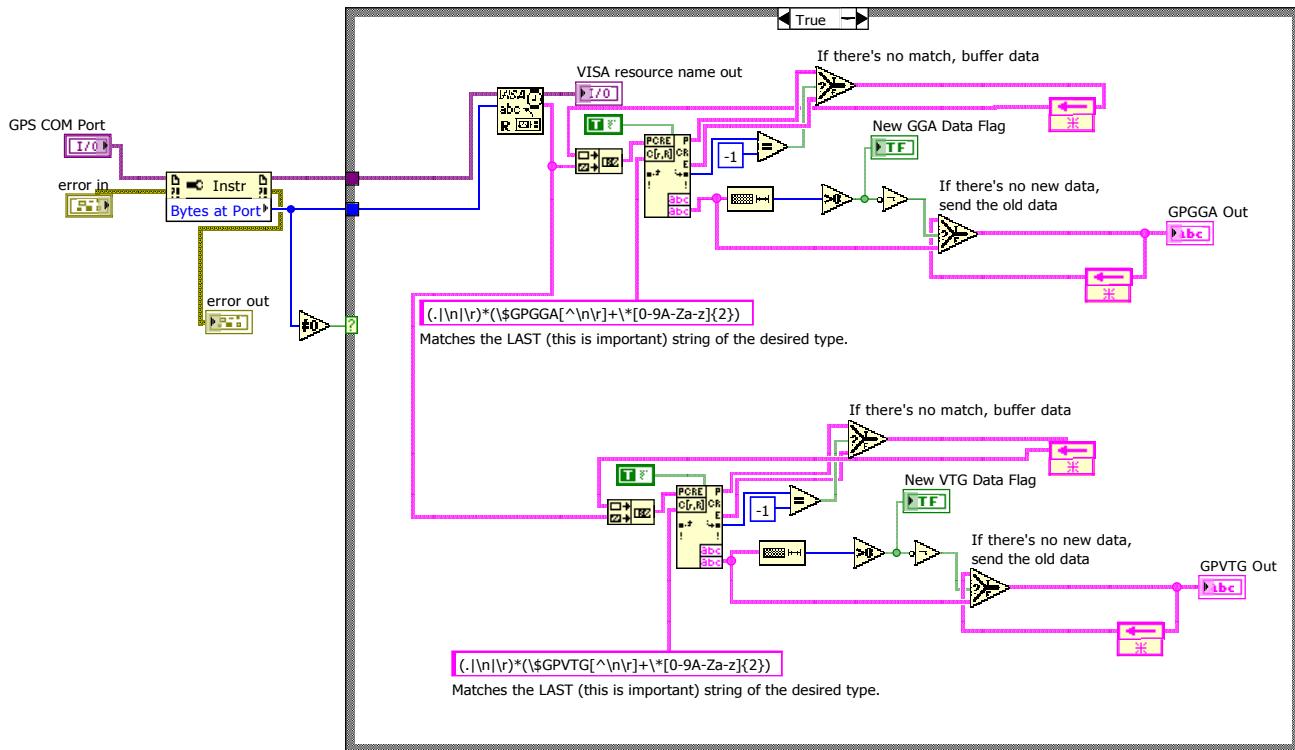
GPS Read.vi

Give this VI a COM port to listen to and it will send out all NMEA sentences it finds, with their leading dollar signs. In between I messages the old message will continue to be output, but the new data flag will go low.

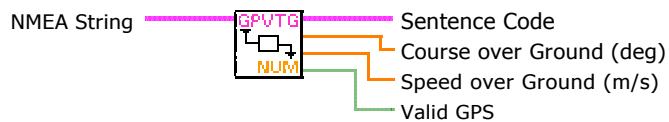
Front Panel



Block Diagram

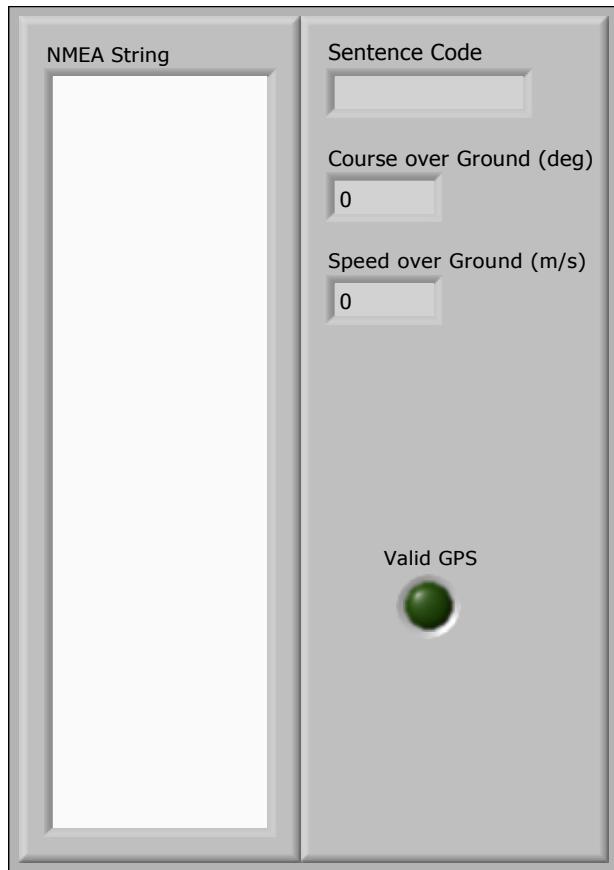


Connector Pane

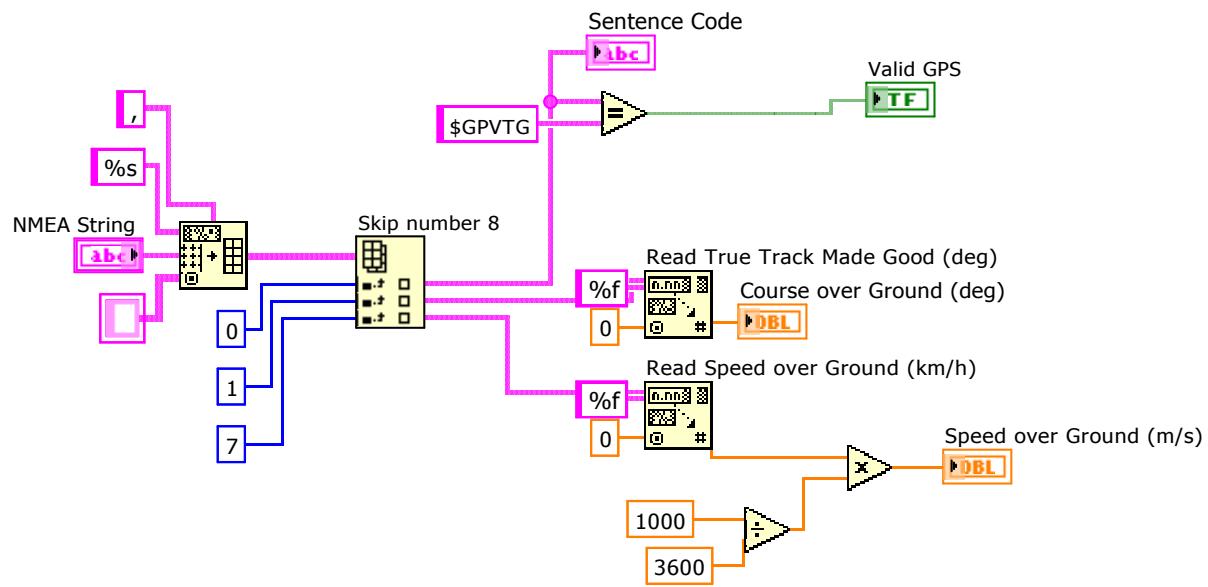
Parse GPVTG.vi

Pass this VI an NMEA sentence and it will parse it as if it's a GPGGA sentence.
The Valid GPS output will be false if there's no fix or if it's not a GPGGA sentence.

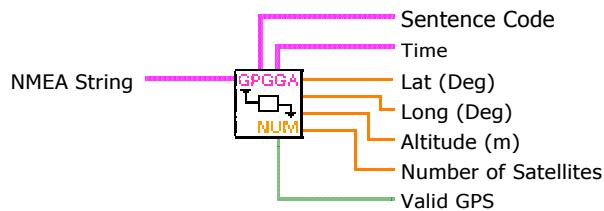
Front Panel



Block Diagram

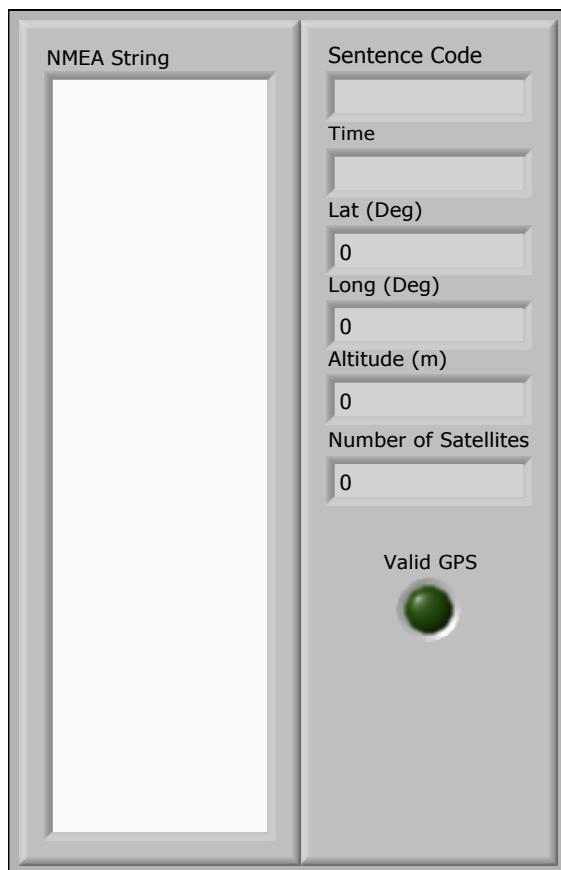


Connector Pane

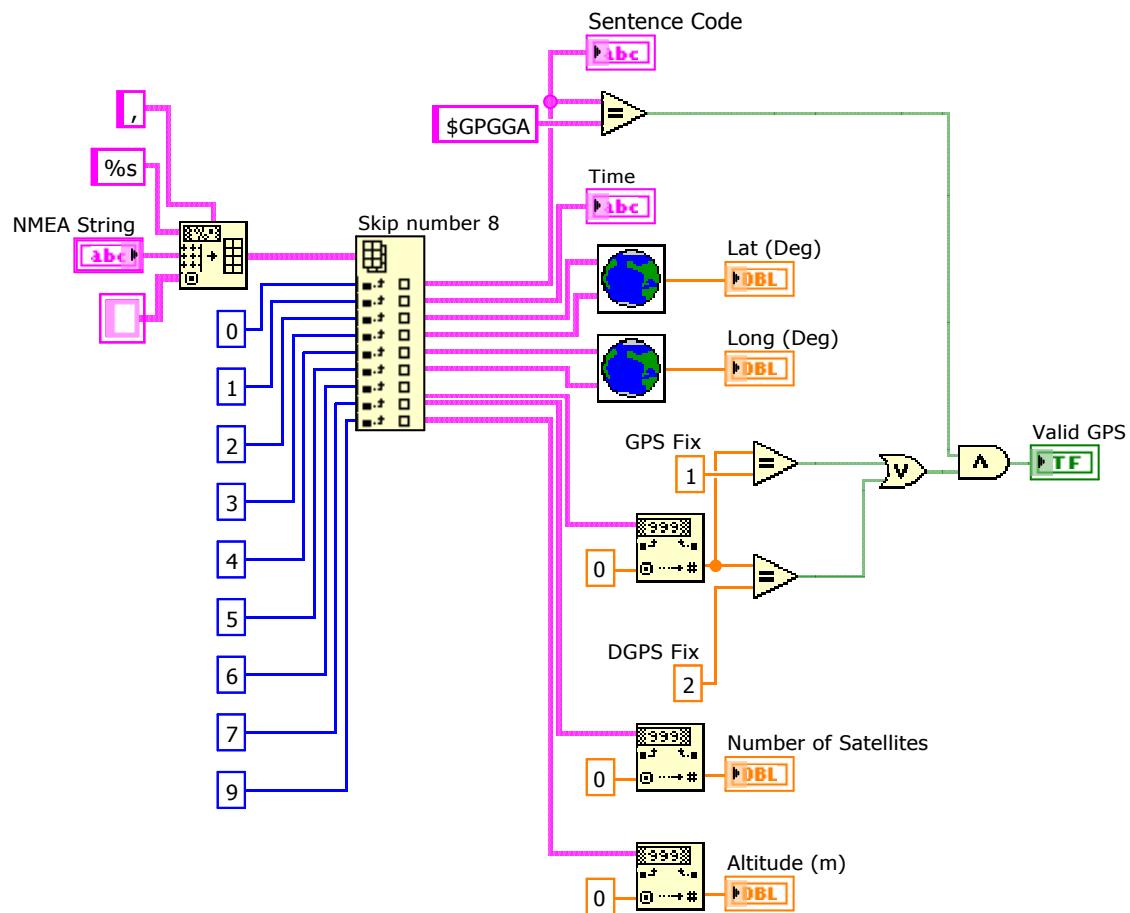
Parse GPGGA.vi

Pass this VI an NMEA sentence and it will parse it as if it's a GPGGA sentence.
The Valid GPS output will be false if there's no fix or if it's not a GPGGA sentence.

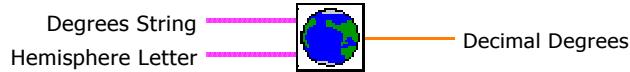
Front Panel



Block Diagram

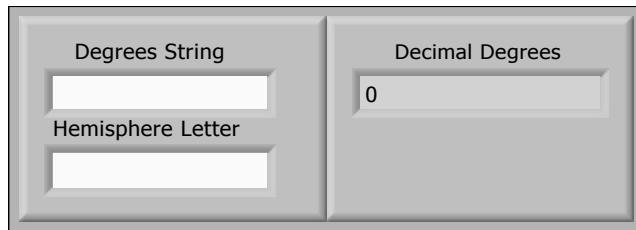


Connector Pane

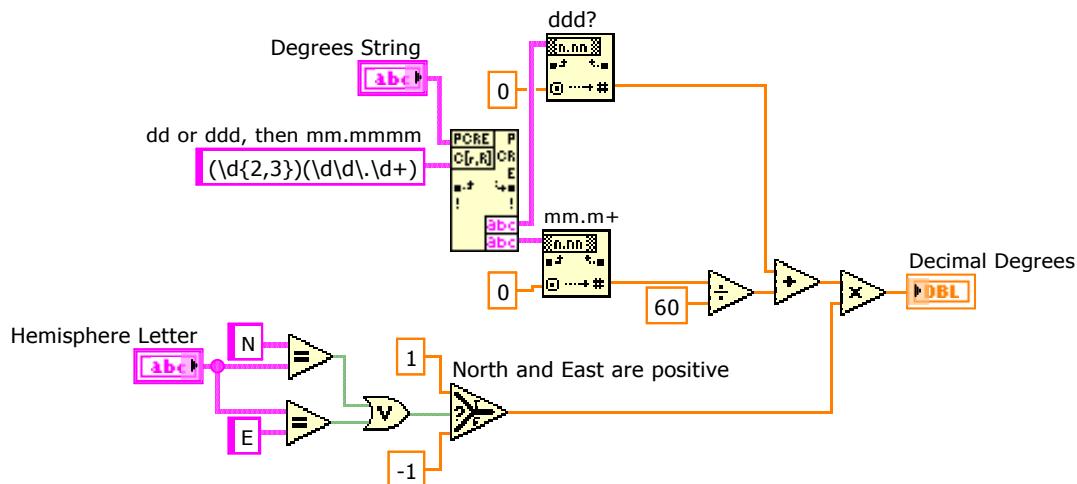
Decimal Degrees from dddmm.mmm.vi

Give this VI decimal degrees in the format [d]ddmm.mmmmmm and a hemisphere letter (N,E,S,W) and it will return decimal degrees with the correct sign.

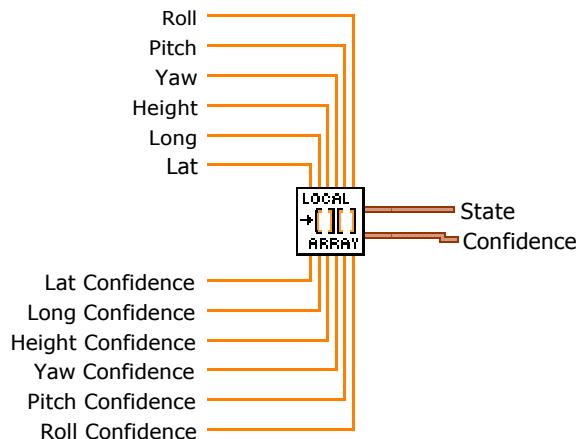
Front Panel



Block Diagram

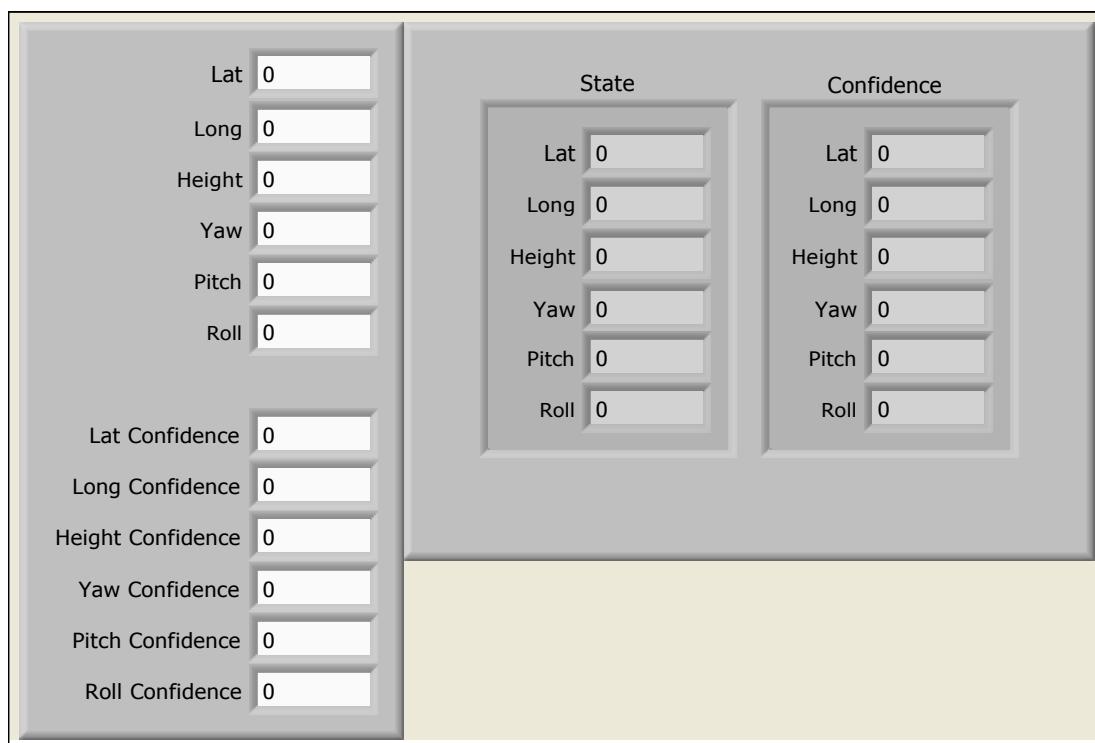


Connector Pane

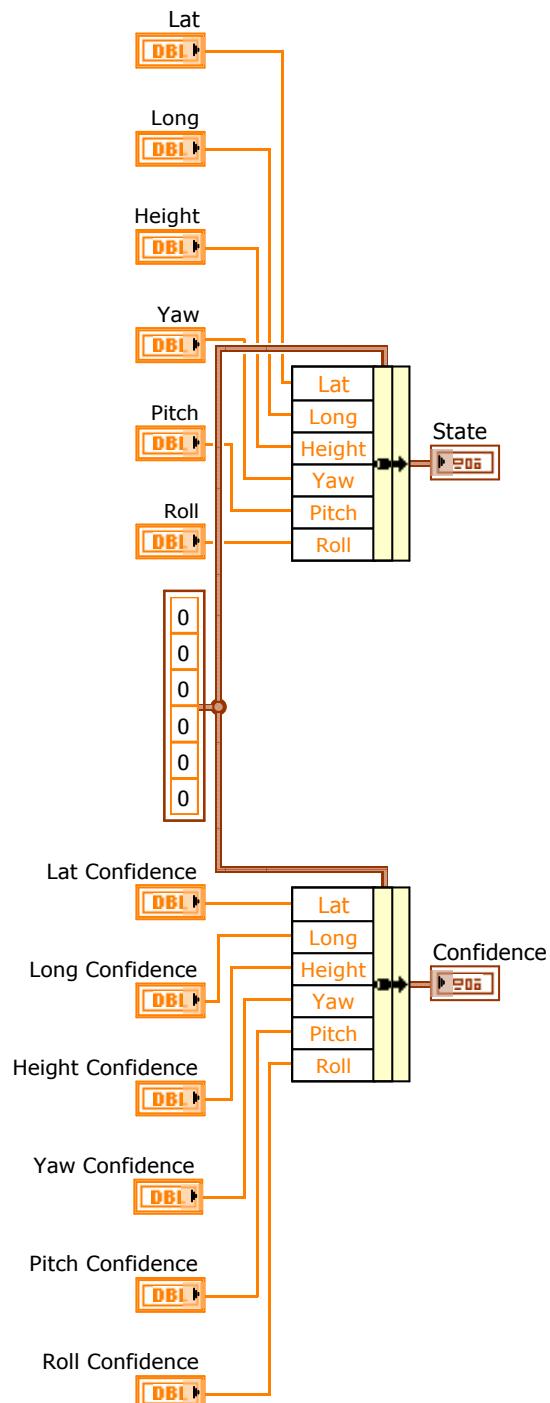
Create Local Array.vi

This VI takes relevant data and builds an array to be sent to the State Weighted Average block.

Front Panel



Block Diagram

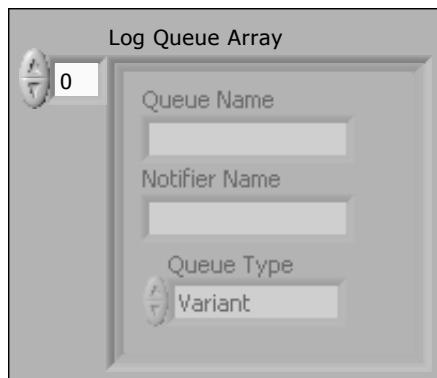


Connector Pane

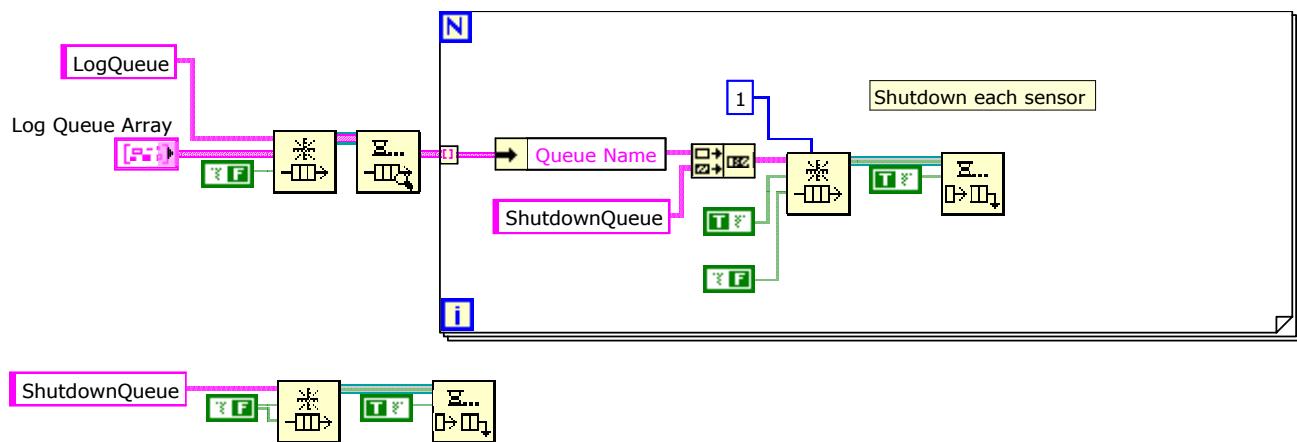
SystemShutdown.vi

Sends a shutdown command to all systems and the generic shutdown queue.

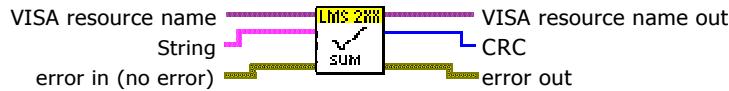
Front Panel



Block Diagram

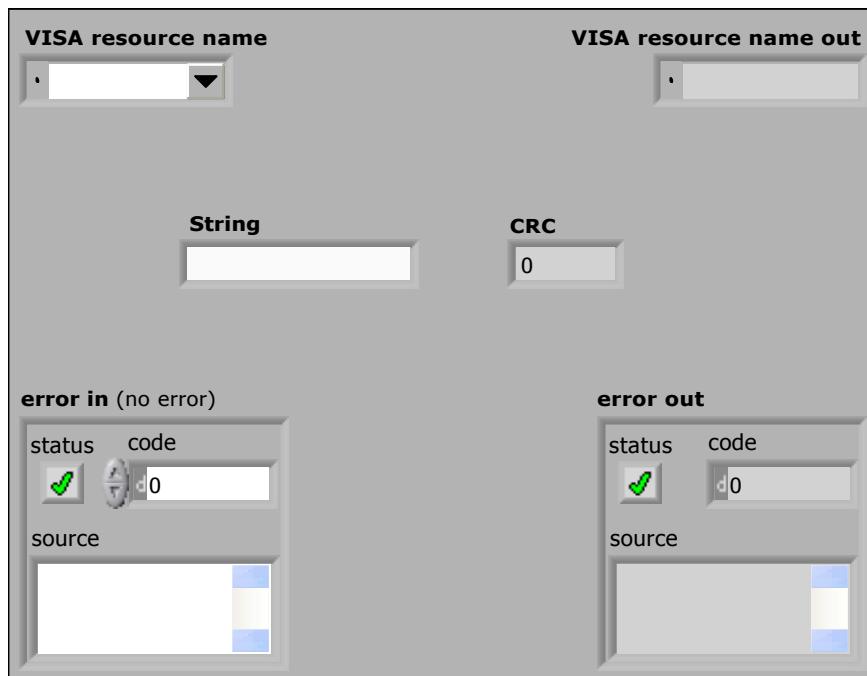


Connector Pane

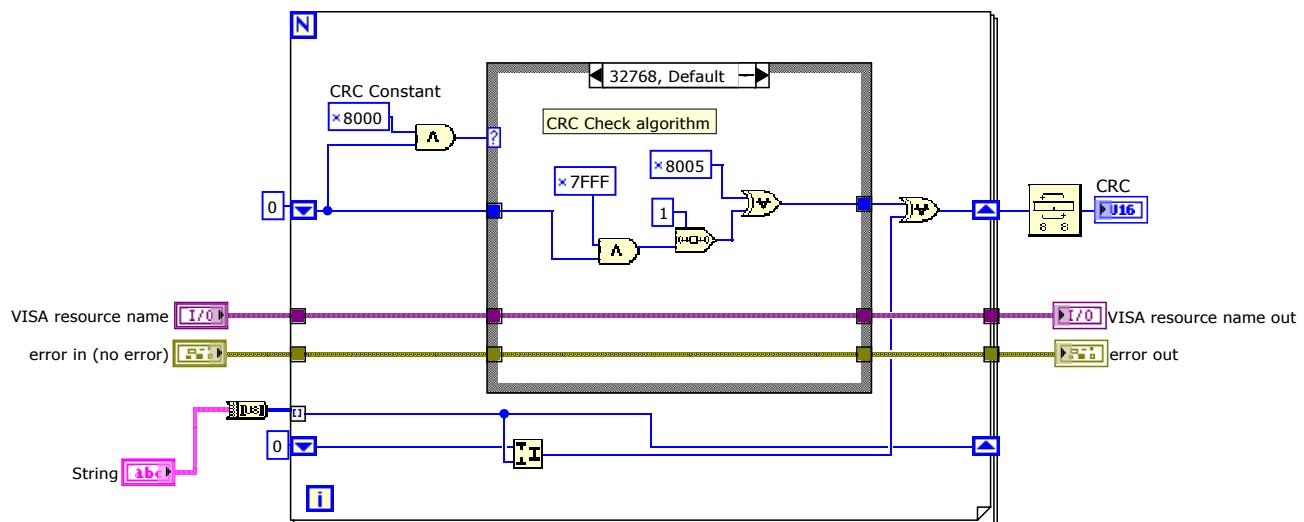
SICK LMS 291.lvlib:Check Sum Status.vi

Calculates what the checksum (CRC) for the header and data should be.

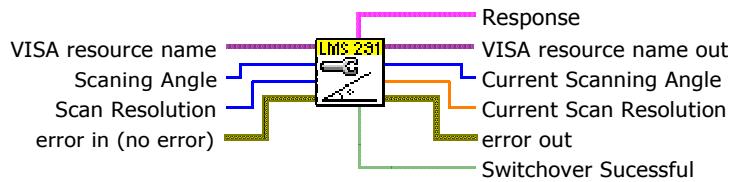
Front Panel



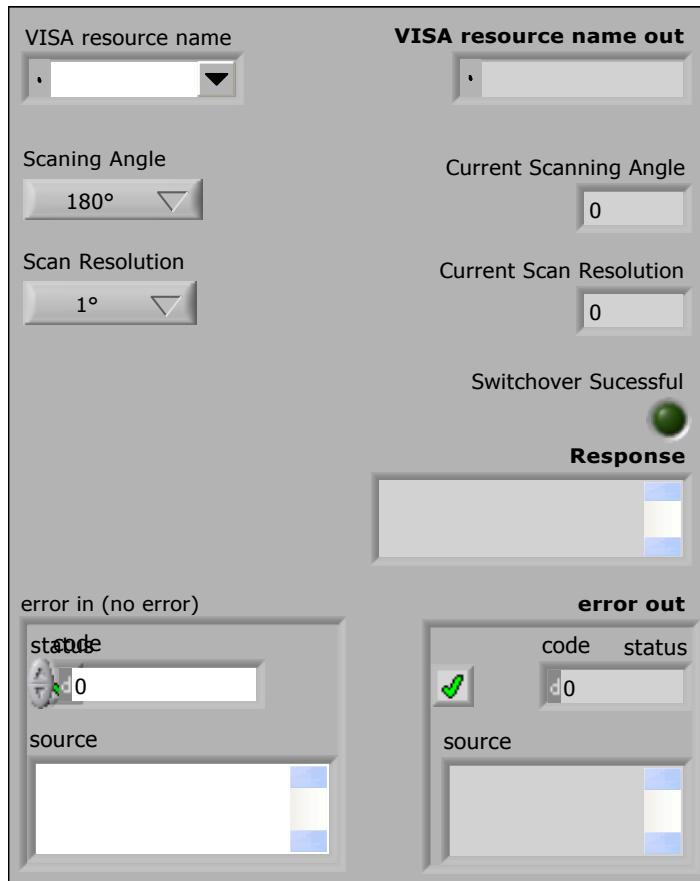
Block Diagram



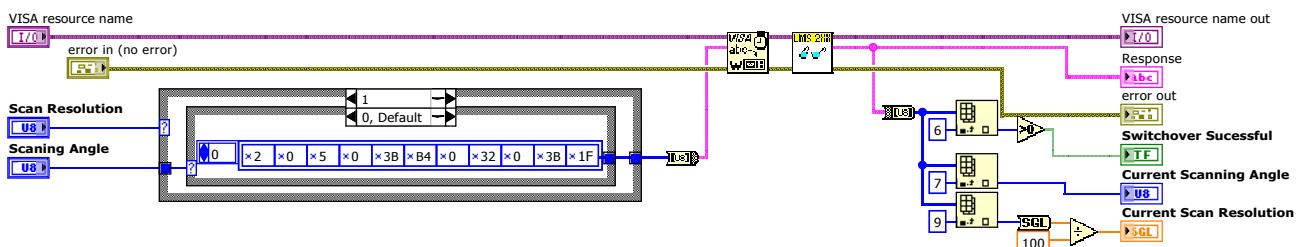
Connector Pane

Configure Angle.vi

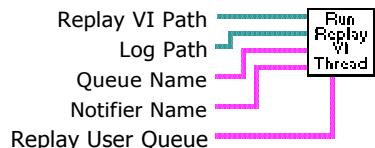
Front Panel



Block Diagram

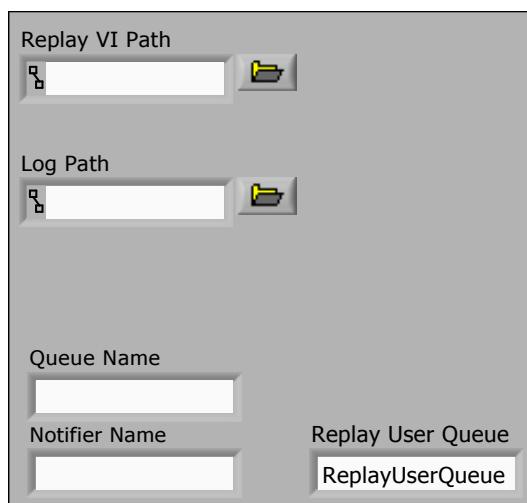


Connector Pane

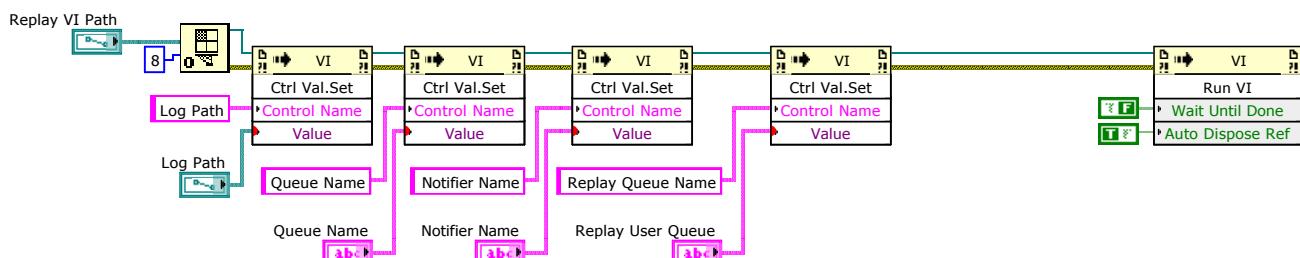
RunReplayVIThread.vi

Start a replaying thread.

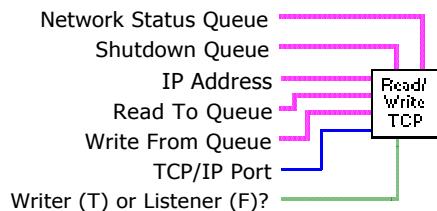
Front Panel



Block Diagram



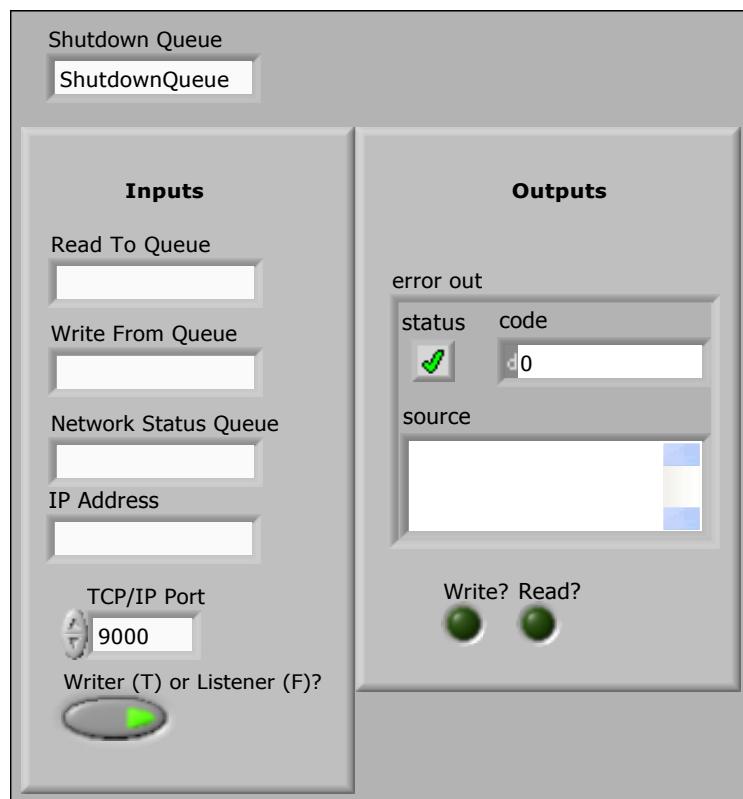
Connector Pane

TCP Read Write.vi

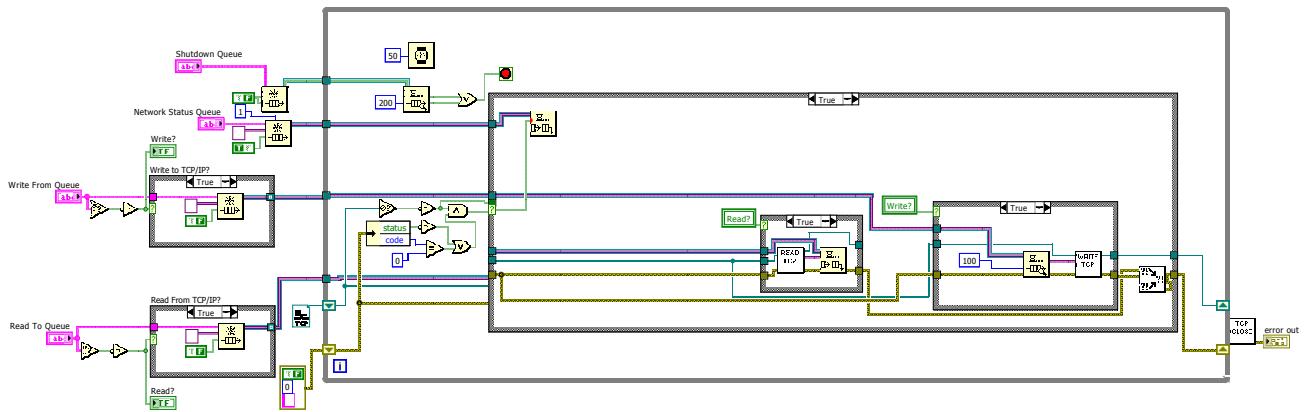
General read/write block for TCP/IP. If you connect Write From Queue, will write data onto the TCP/IP link. If you connect Read To Queue, will write TCP data to the queue.

If you select to be a Writer (a server), you must supply IP Adress.

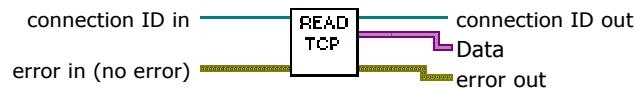
Front Panel



Block Diagram



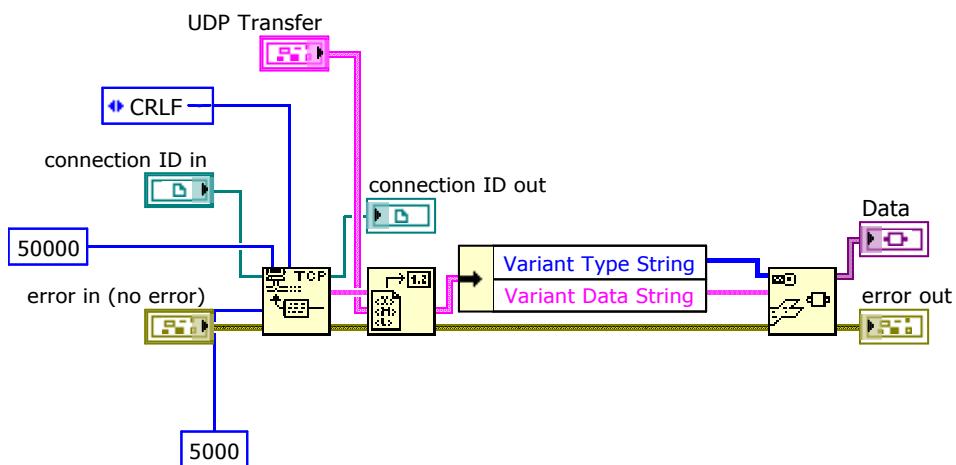
Connector Pane

TCP Read.vi

Front Panel

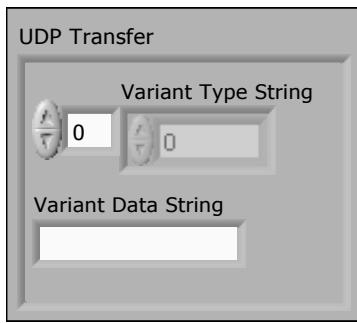


Block Diagram



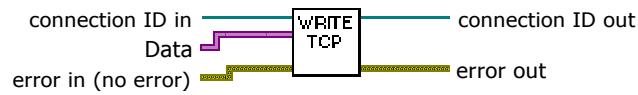
Connector Pane

UDPControl.ctl

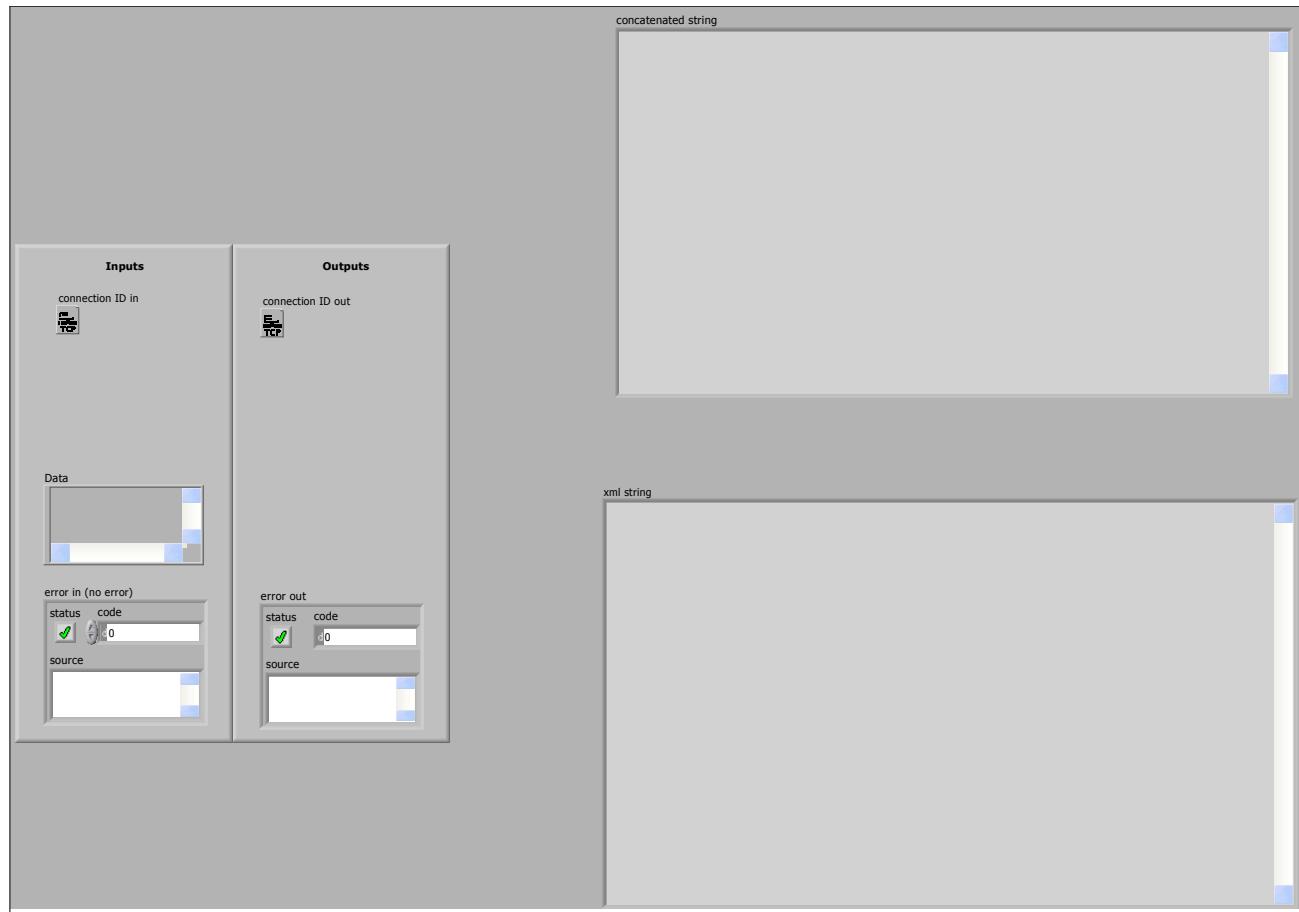
Front Panel

Block Diagram

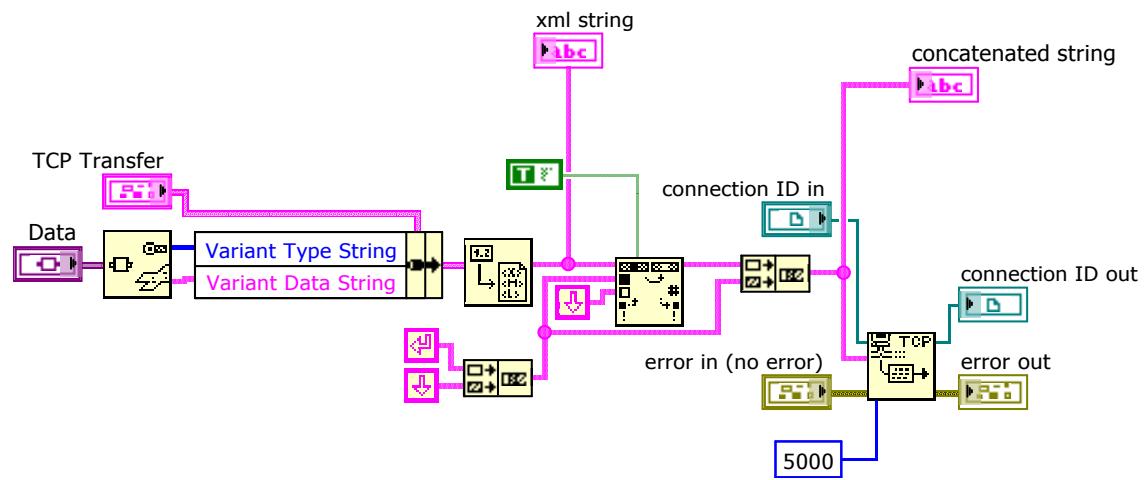
Connector Pane

TCP Write.vi

Front Panel



Block Diagram

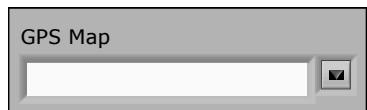


Connector Pane

GPSMapSelector.ctl



Front Panel



Block Diagram

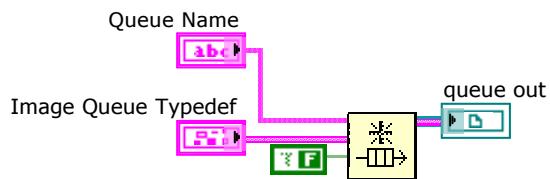
Connector Pane

GetImageQueue.vi

Front Panel



Block Diagram

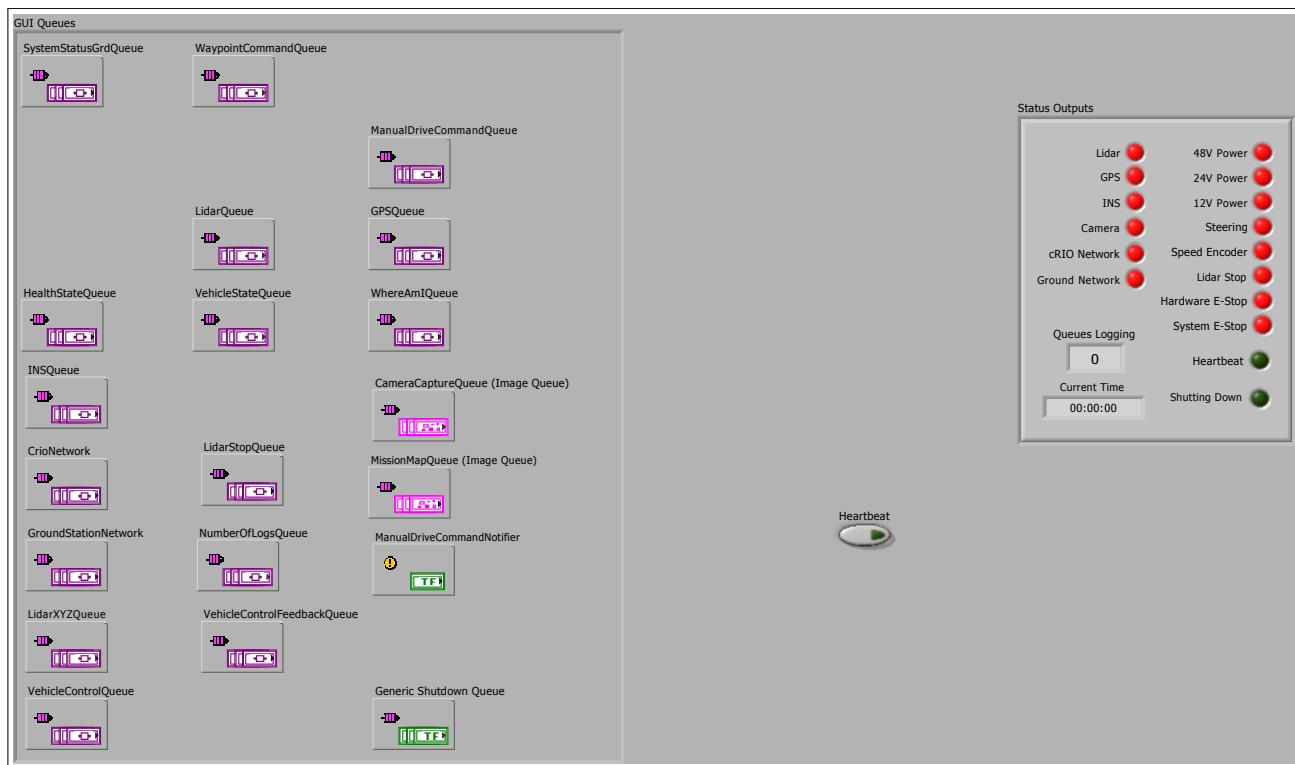


Connector Pane

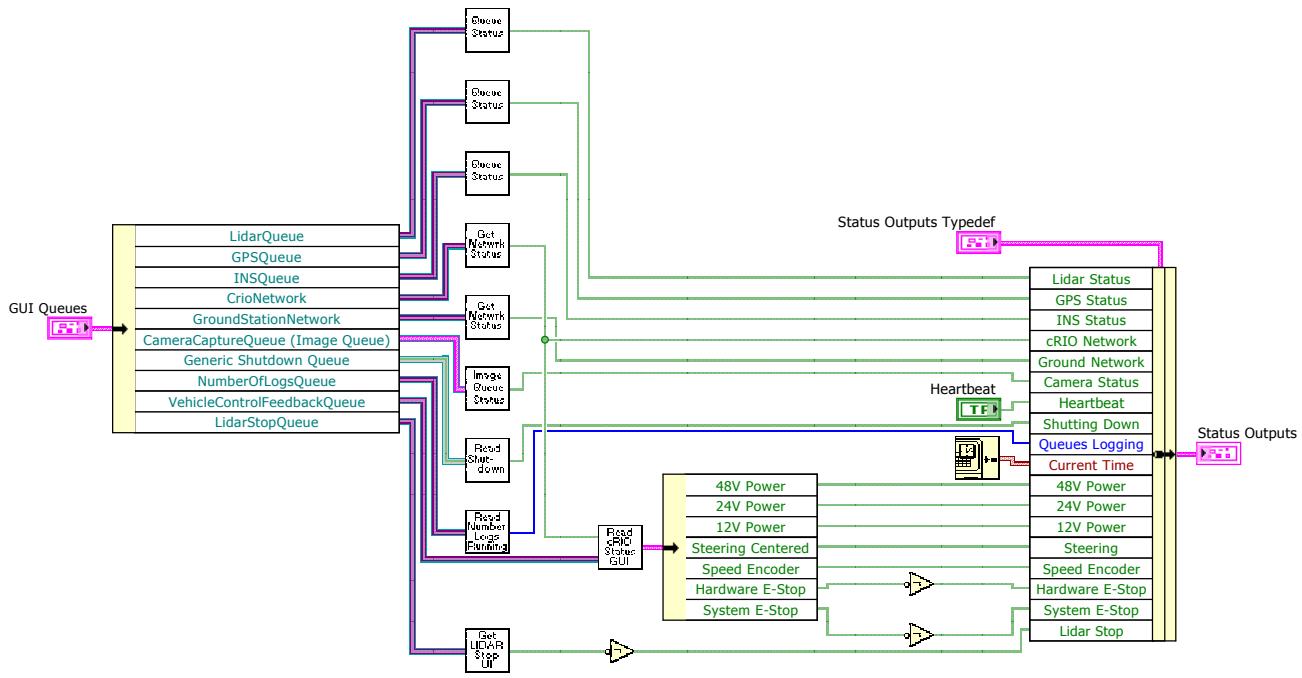
AllQueueStatus.vi

GUI function to get status lights.

Front Panel



Block Diagram



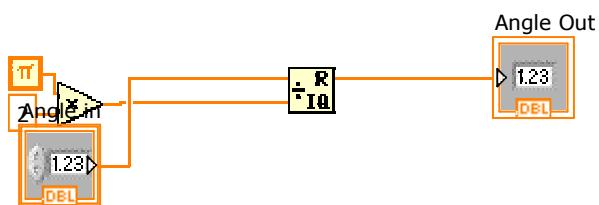
Connector Pane

Mod2Pi.vi

Front Panel



Block Diagram

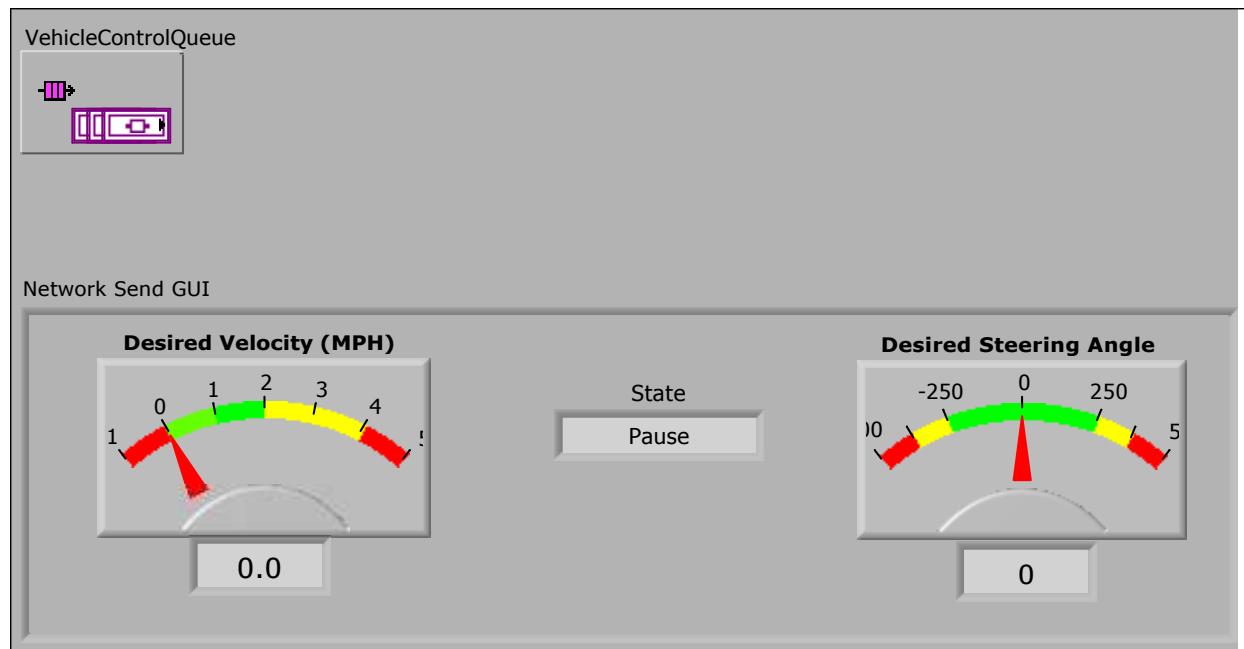


Connector Pane

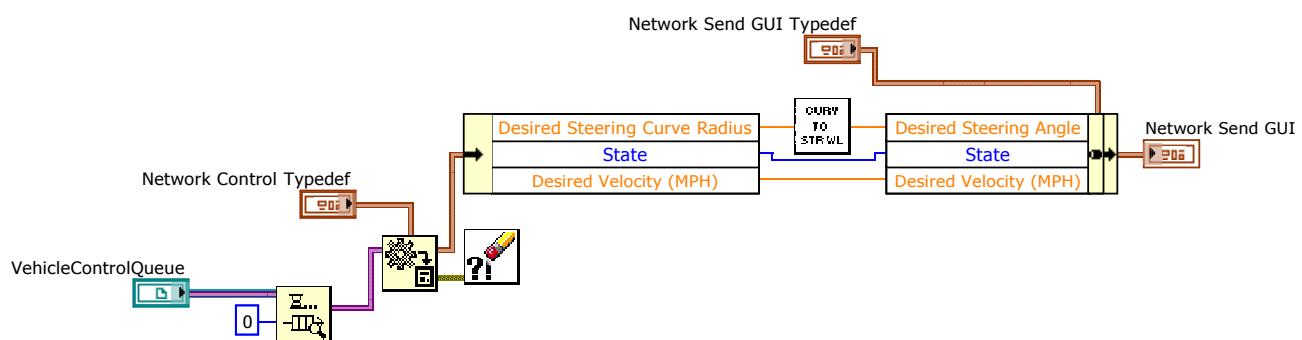
NetworkRead.vi

Reads values being sent over the network to the cRIO for GUI display.

Front Panel



Block Diagram

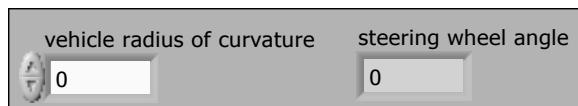


Connector Pane

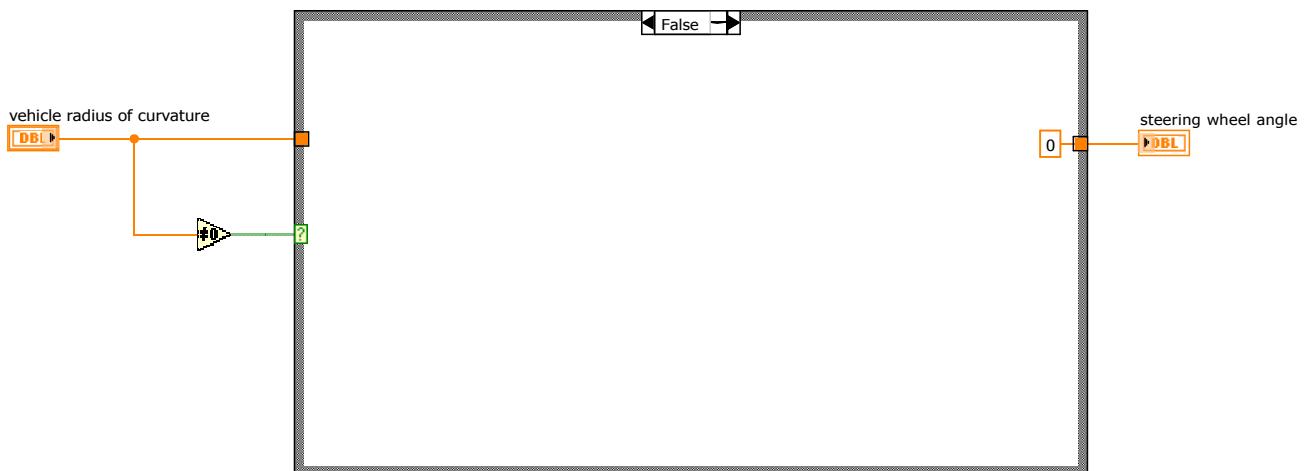
Wheel Angle to StrWheel Angle.vi

Converts physical wheel position to steering wheel position

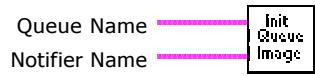
Front Panel



Block Diagram



Connector Pane

InitQueueImage.vi

Initialize an Image Queue and optionally a notifier. Adds the queue and notifier to the logging/replay lists.

Front Panel

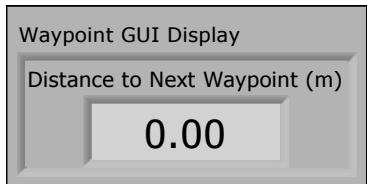


Inits a queue and optionally a notifier. Adds the queue and notifier to the log list.

Connector Pane

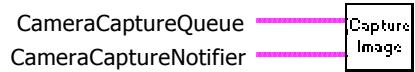
WaypointGUIControl.ctl

Front Panel



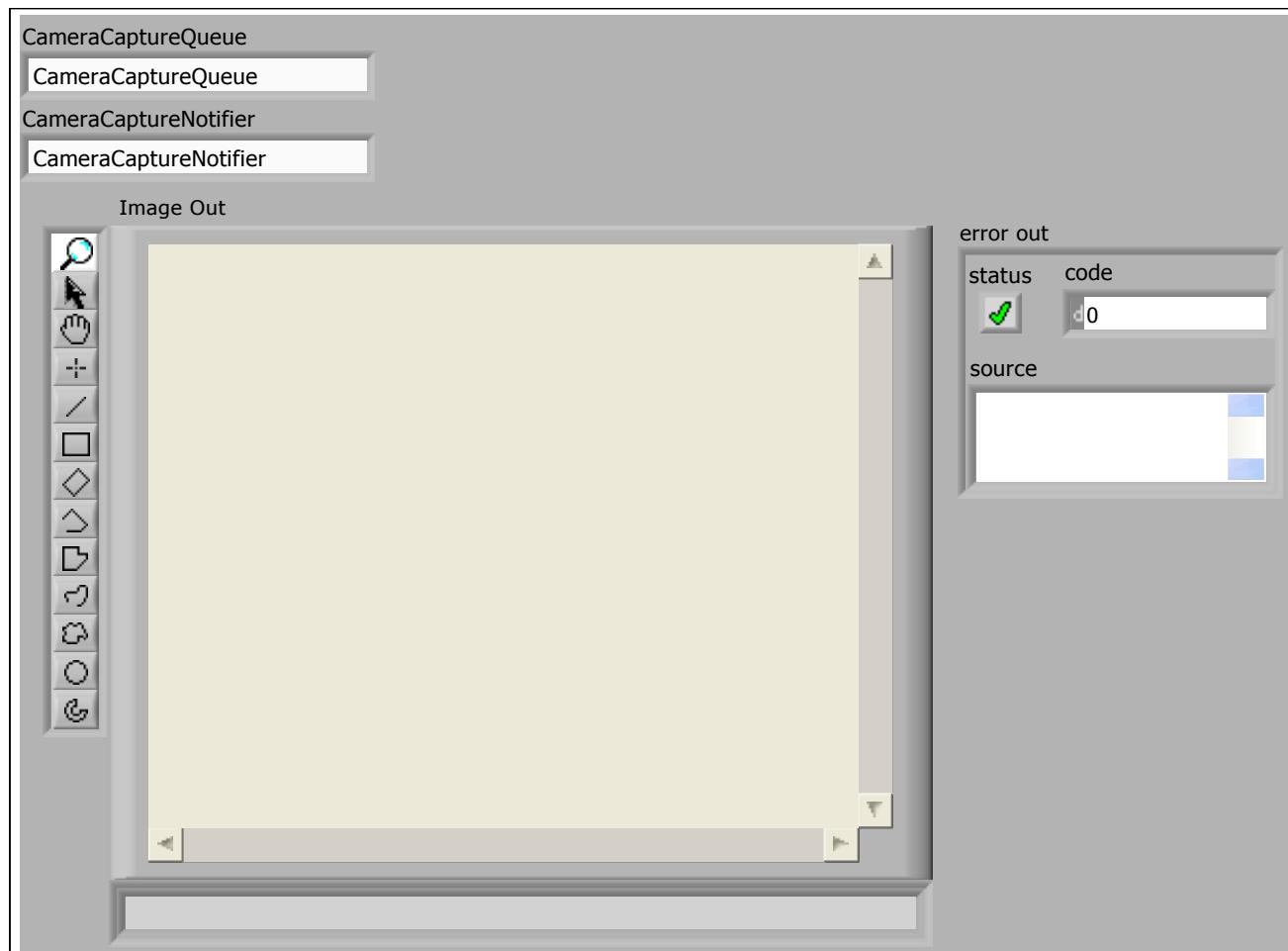
Block Diagram

Connector Pane

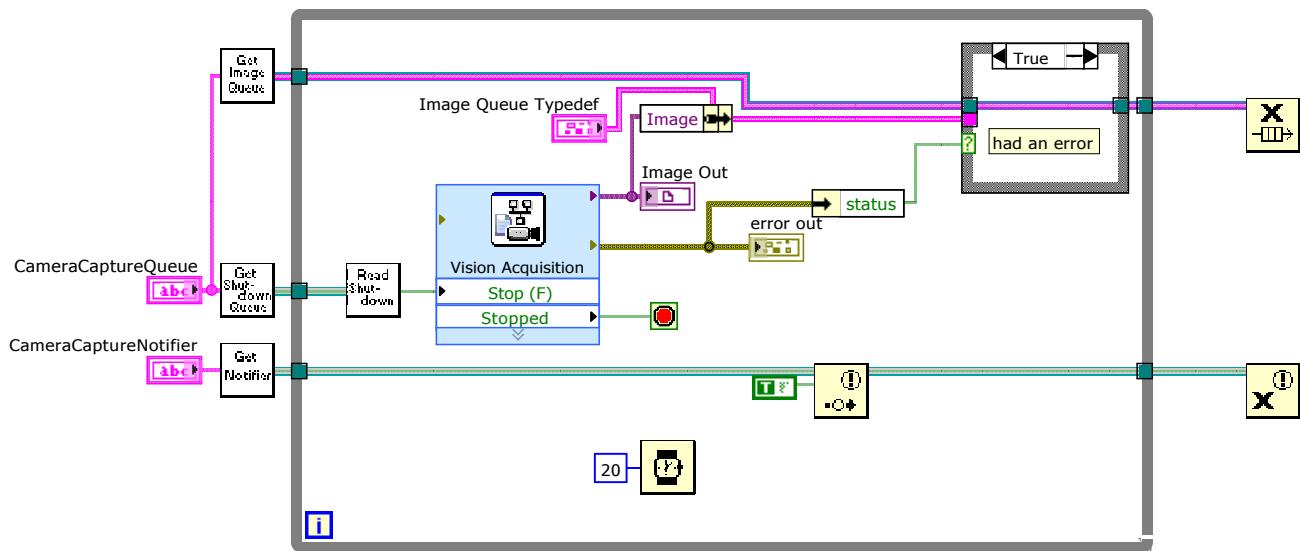
CaptureImage.vi

Capture images from a camera and enqueue them.

Front Panel



Block Diagram



Connector Pane

LidarStopControl.ctl



Front Panel



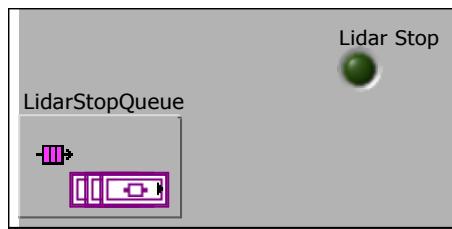
Block Diagram

Connector Pane

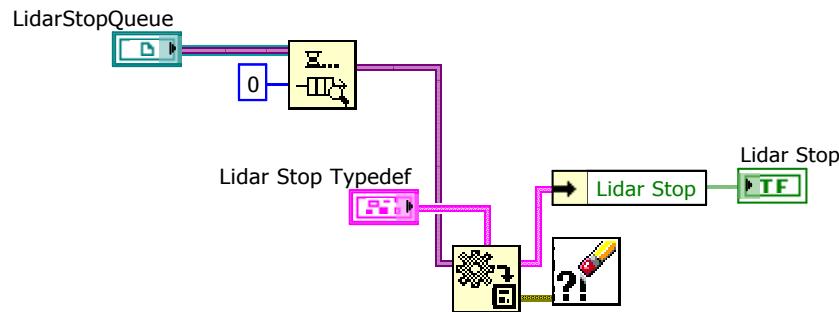
GetLidarStopUI.vi

GUI function to read the LIDAR obstacle detection e-stop state.

Front Panel



Block Diagram



Connector Pane

DebugInputs.ctl

Front Panel

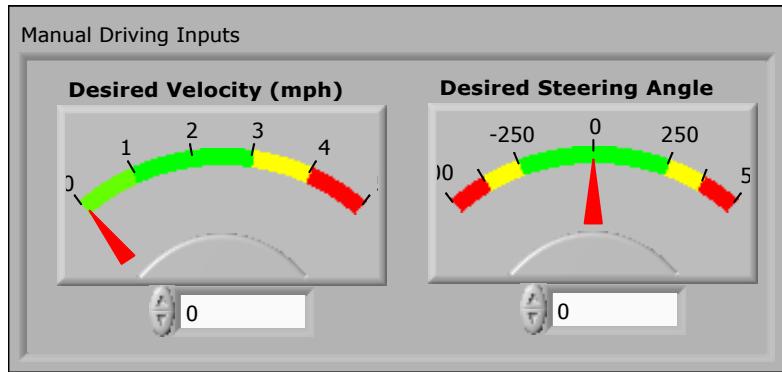


Block Diagram

Connector Pane

ManualDrivingInputs.ctl

Front Panel

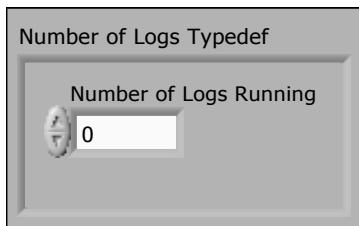


Block Diagram

Connector Pane

NumberOfLogs.ctl

Front Panel



Block Diagram

Connector Pane

AlwaysDisplayInputs.ctl

Front Panel



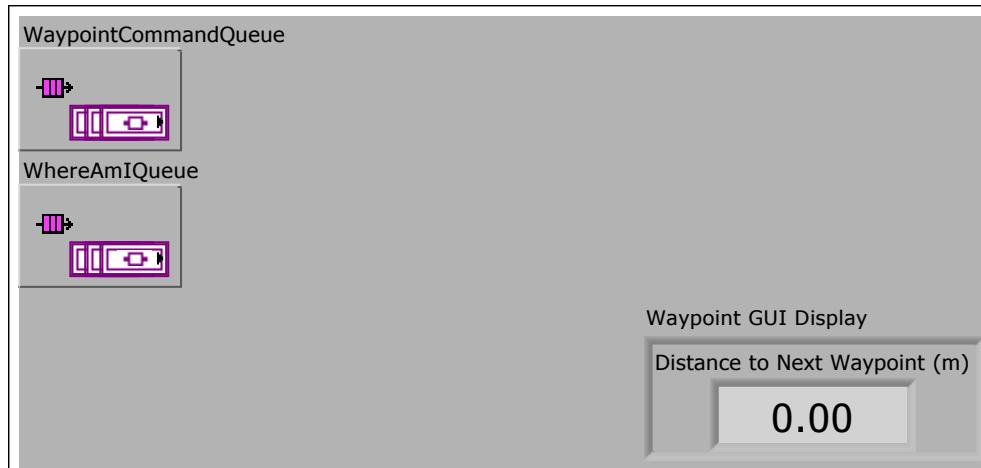
Block Diagram

Connector Pane

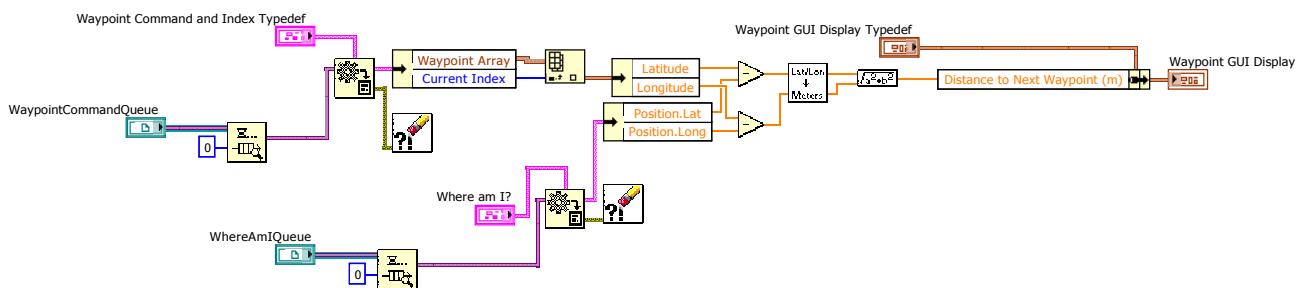
GetWaypointGUI.vi

GUI function to display the distance to next waypoint.

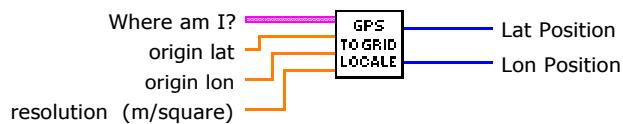
Front Panel



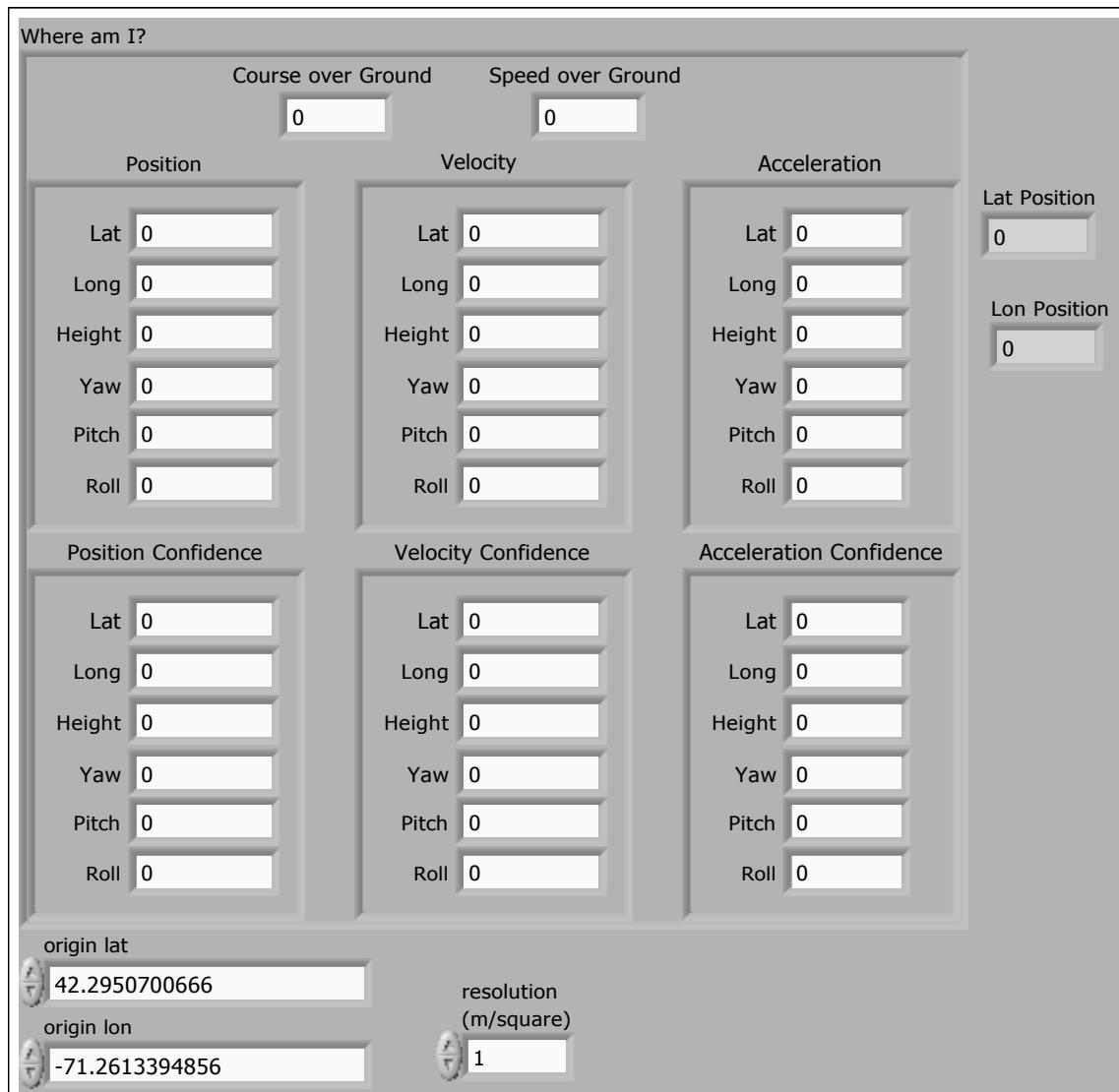
Block Diagram



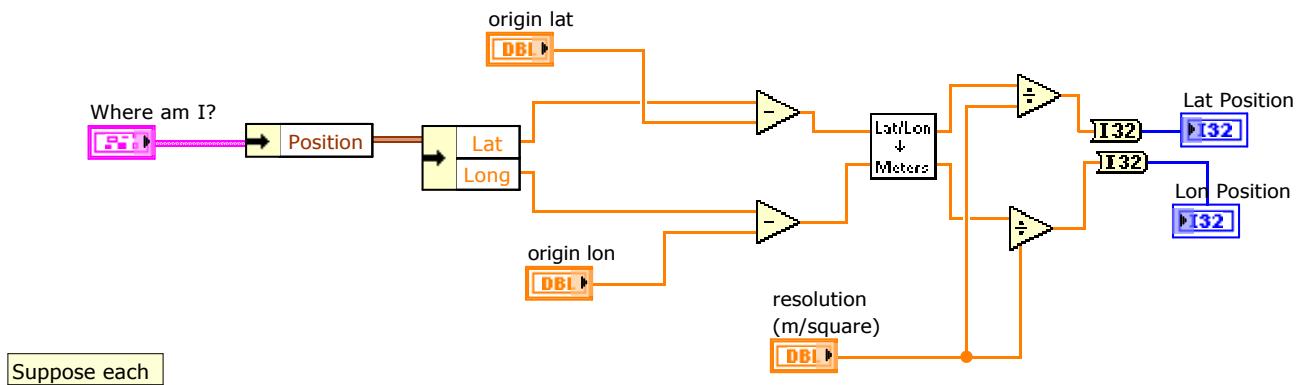
Connector Pane

GPS to Grid Position.vi

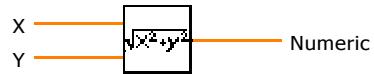
Front Panel



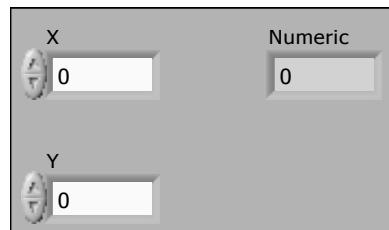
Block Diagram



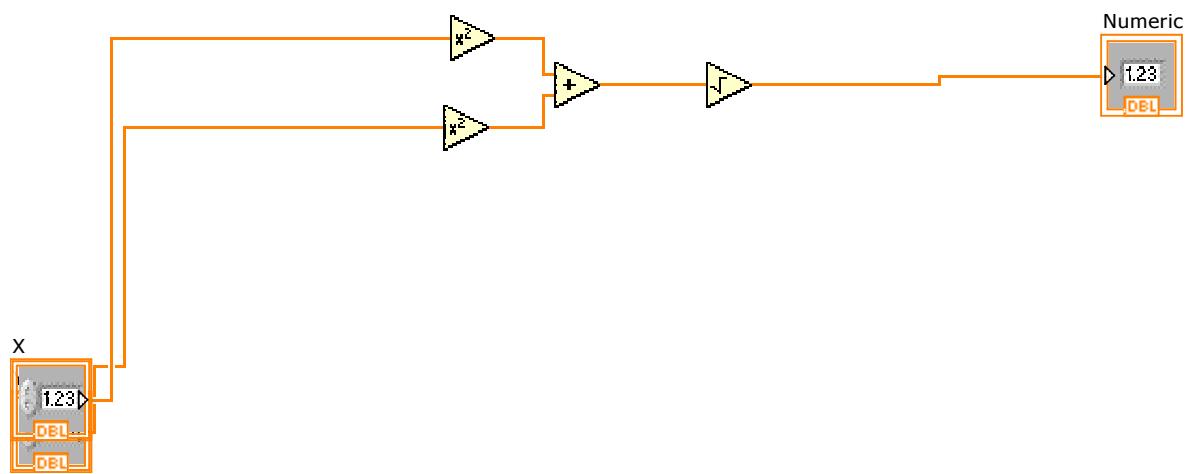
Connector Pane

distance from origin.vi

Front Panel



Block Diagram

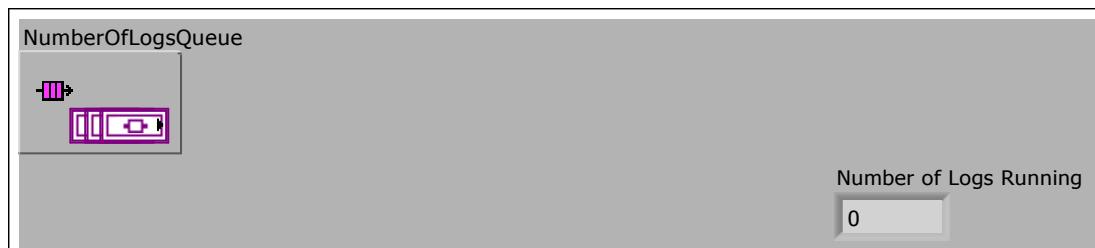


Connector Pane

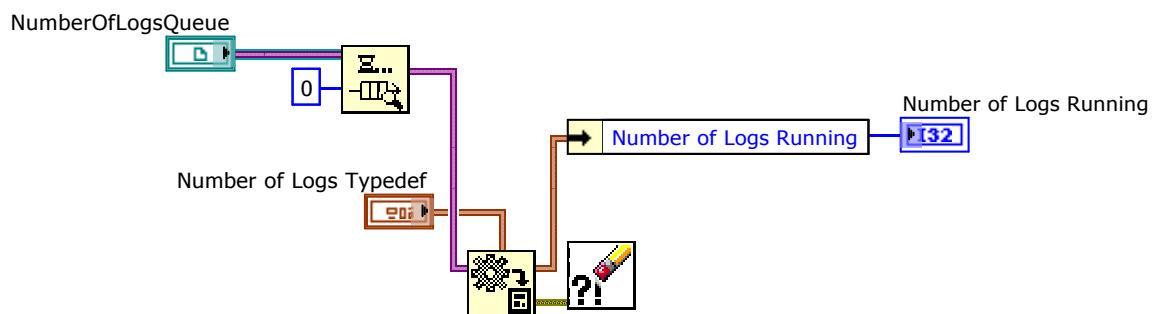
ReadNumberOfLogsRunning.vi

GUI function to display the number of logs currently running.

Front Panel



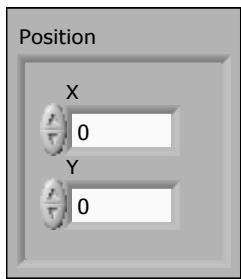
Block Diagram



Connector Pane

Location.ctl

Front Panel



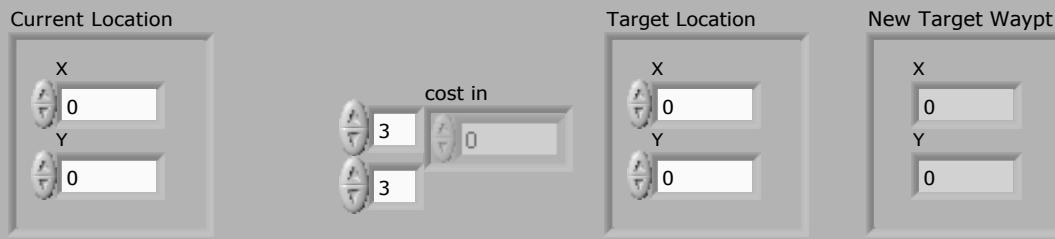
Block Diagram

Connector Pane

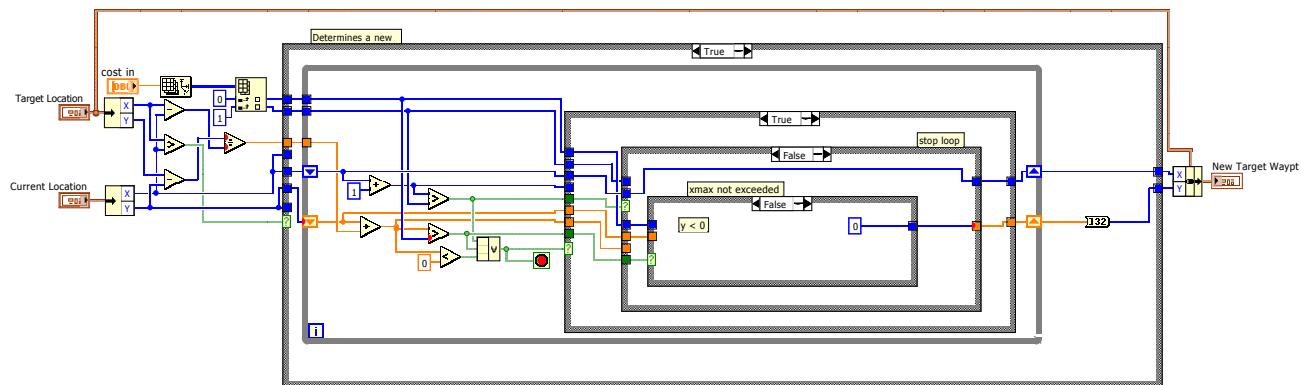
Get Replacement Waypoint.vi

Front Panel

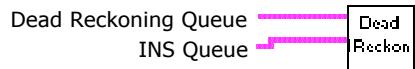
This is used if the desired waypoint is not inside the occupancy grid, so we find the closest waypoint that is in the occupancy grid



Block Diagram

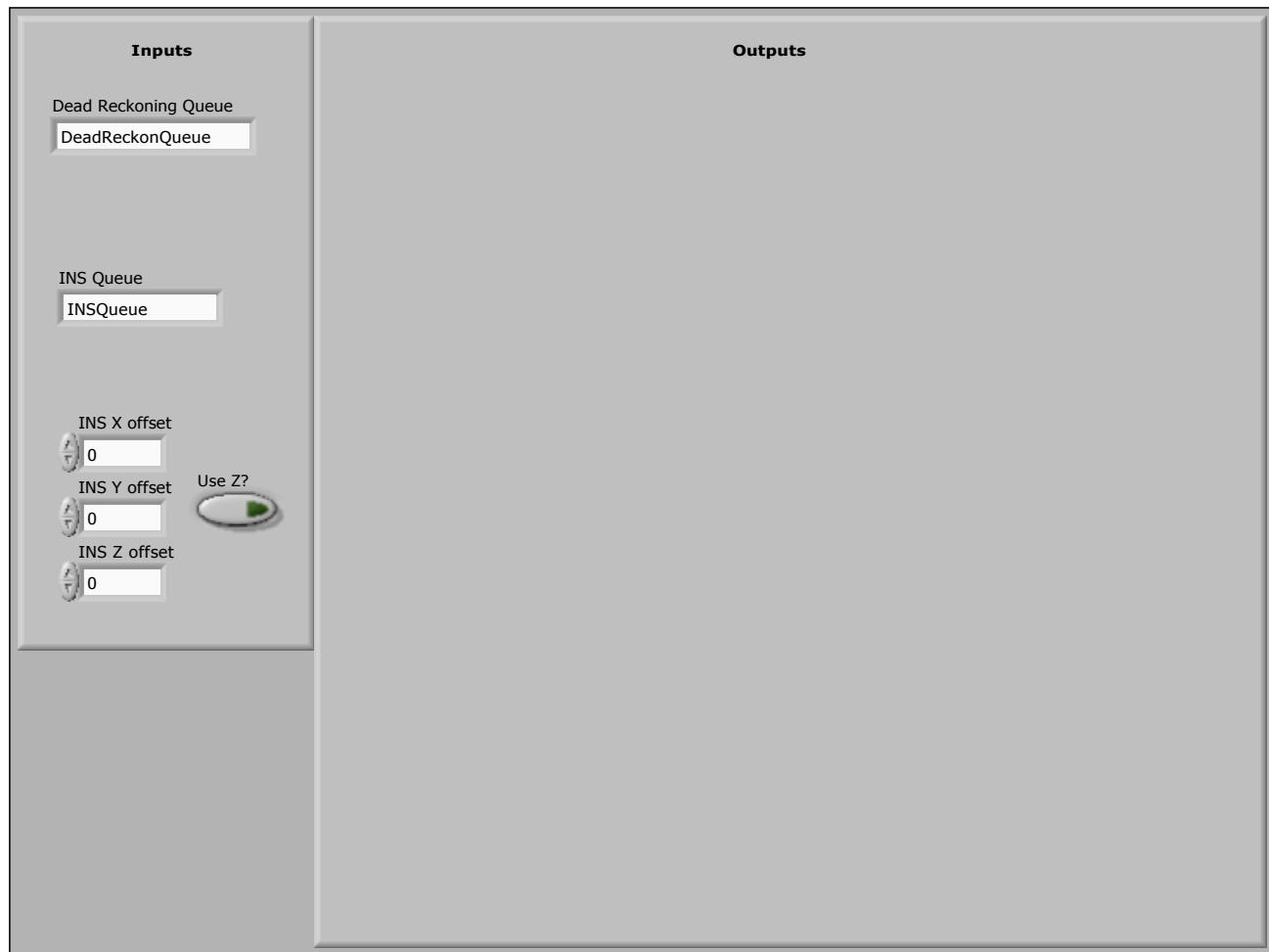


Connector Pane

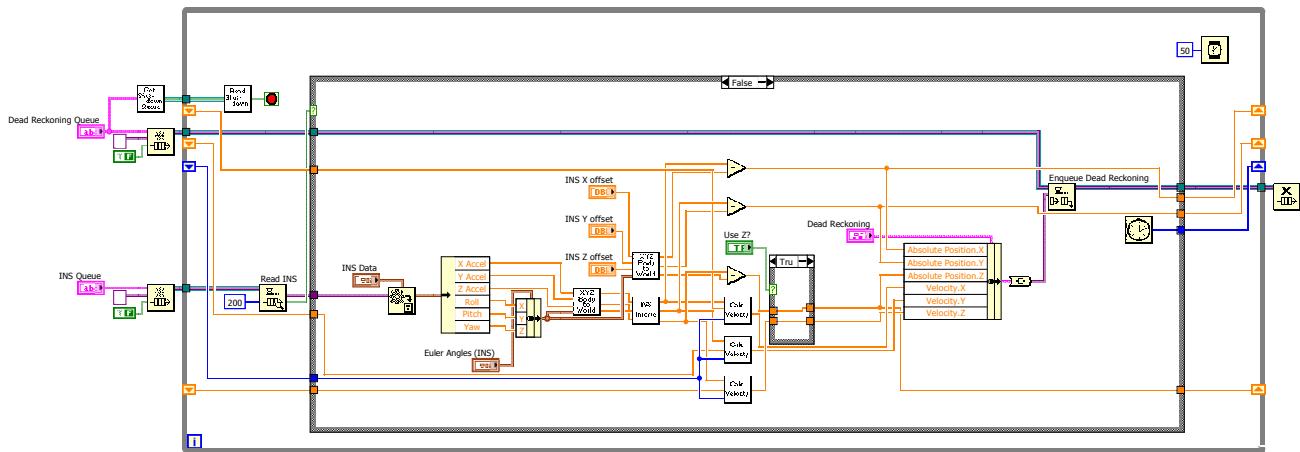
Dead Reckon.vi

Compute Dead Reckoning Data from the INS.

Front Panel



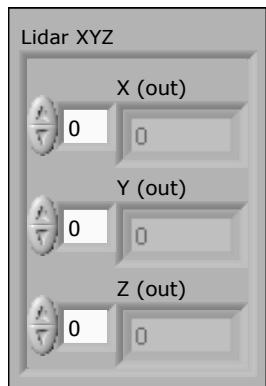
Block Diagram



Connector Pane

LidarXYZ.ctl

Front Panel

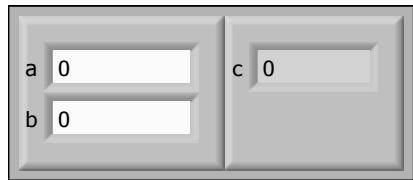


Block Diagram

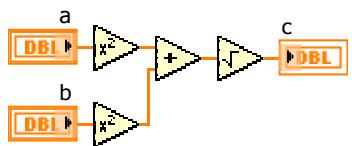
Connector Pane

Pythagorean Theorem.vi

Front Panel



Block Diagram

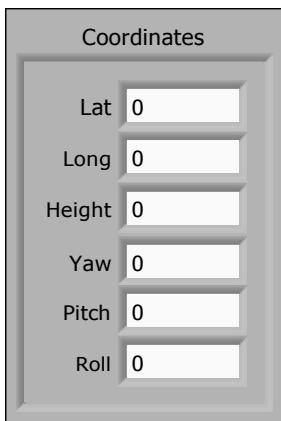


Connector Pane

Coordinate Cluster.ctl

This cluster should be used whenever you're dealing with coordinates or confidences.

Front Panel

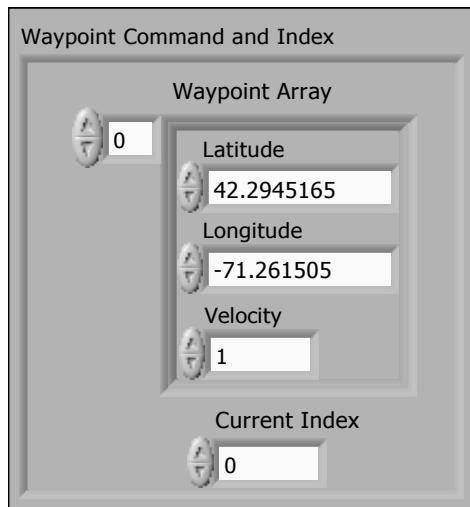


Block Diagram

Connector Pane

Waypoint Command.ctl

Front Panel

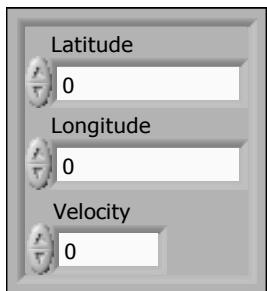


Block Diagram

Connector Pane

WaypointControlLowestLevel.ctl

Front Panel



Block Diagram

Connector Pane

Where Am I.ctl

Front Panel

Where am I?

Course over Ground		Speed over Ground	
0	0	0	0

Position		Velocity		Acceleration	
Lat	0	Lat	0	Lat	0
Long	0	Long	0	Long	0
Height	0	Height	0	Height	0
Yaw	0	Yaw	0	Yaw	0
Pitch	0	Pitch	0	Pitch	0
Roll	0	Roll	0	Roll	0

Position Confidence		Velocity Confidence		Acceleration Confidence	
Lat	0	Lat	0	Lat	0
Long	0	Long	0	Long	0
Height	0	Height	0	Height	0
Yaw	0	Yaw	0	Yaw	0
Pitch	0	Pitch	0	Pitch	0
Roll	0	Roll	0	Roll	0

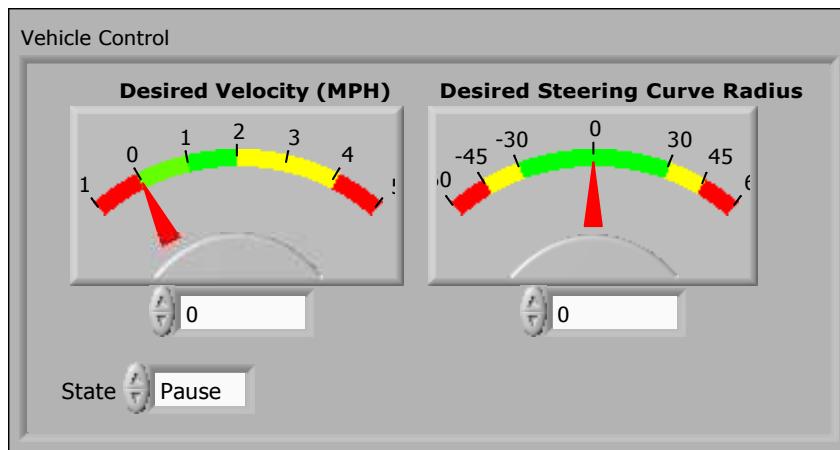
Block Diagram

Connector Pane

VehicleControl.ctl

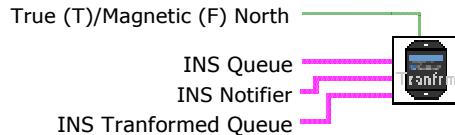
This defines the type sent to the cRIO that contains the main driving commands.

Front Panel



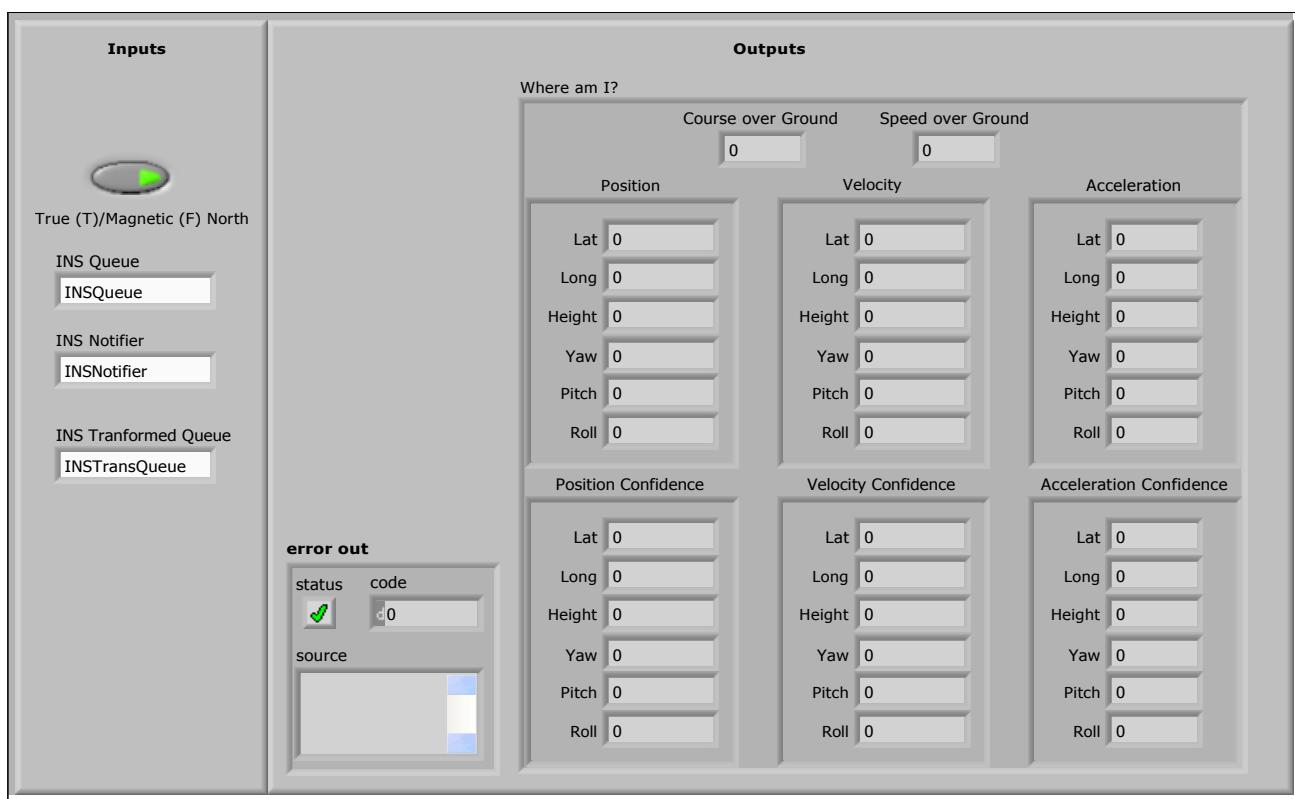
Block Diagram

Connector Pane

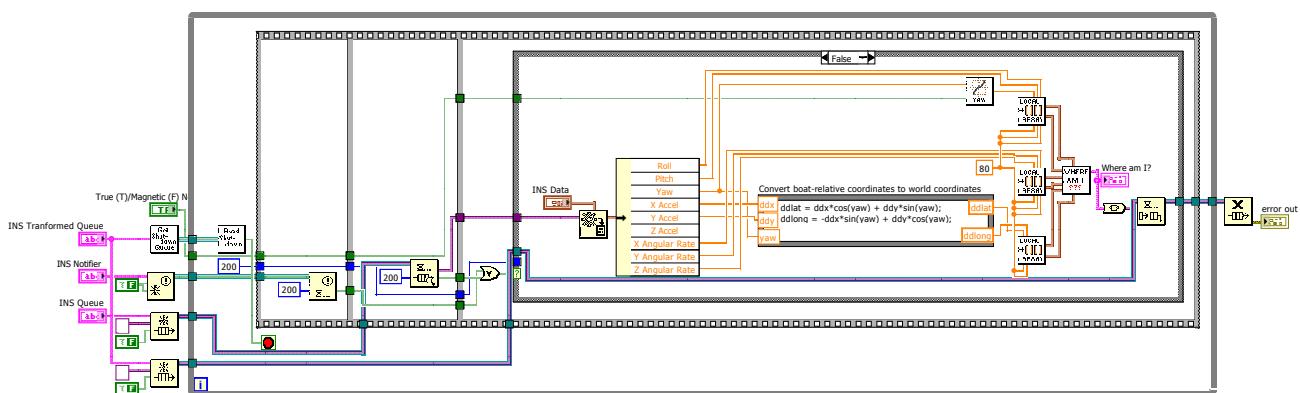
INSTransform.vi

Takes the raw INS data (INS Queue) and transforms it into the format of the Where Am I cluster (Transform INS Queue).

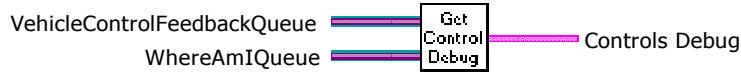
Front Panel



Block Diagram

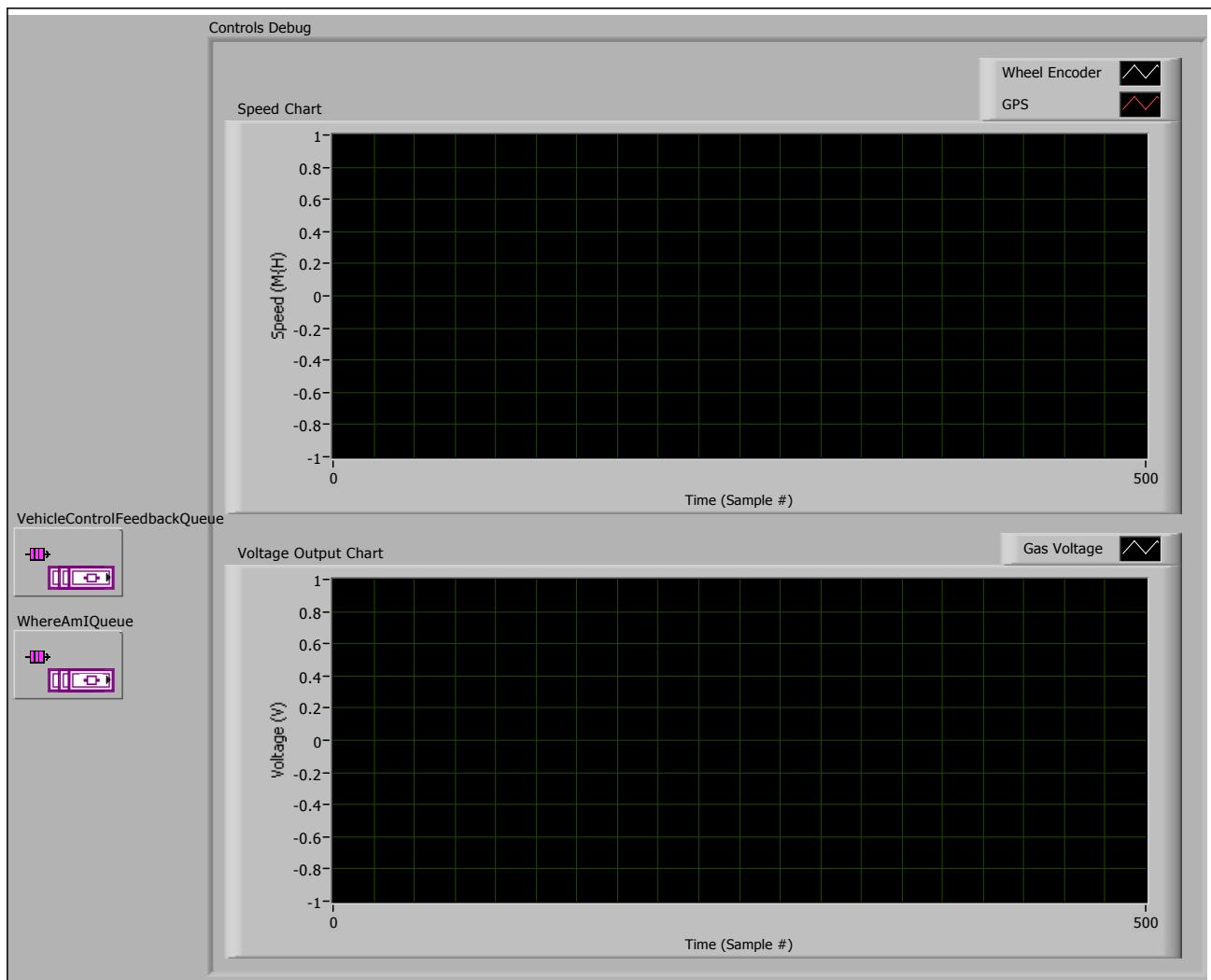


Connector Pane

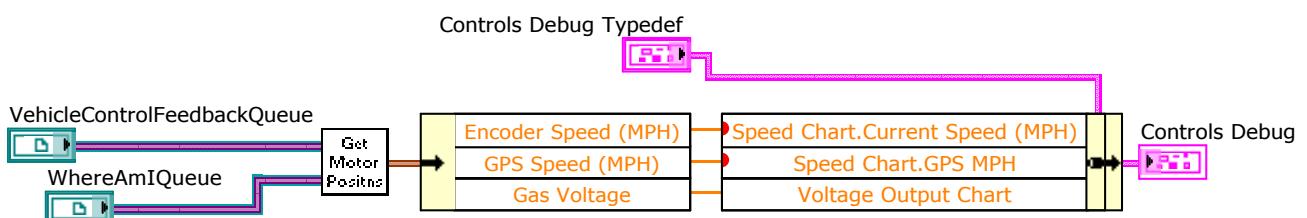
GetControlsDebug.vi

GUI function to display controls debug charts.

Front Panel



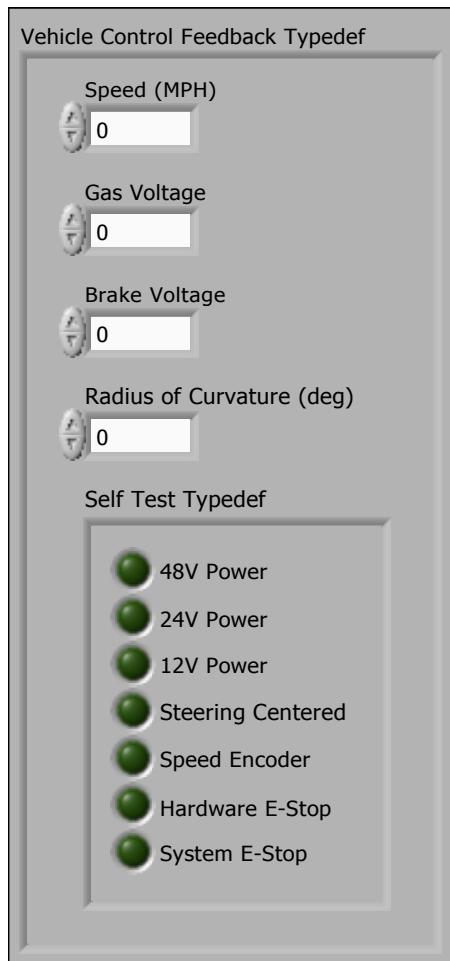
Block Diagram



Connector Pane

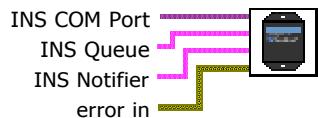
VehicleControlFeedback.ctl

Front Panel



Block Diagram

Connector Pane

INS.vi

This VI listens to the specified COM port and uses Microstrain's Read Euler Angles Accel. and Ang. Rate VI to find out acceleration and angular position.

The corrected yaw option runs the output through a questionably useful yaw correction algorithm, because the INS didn't seem to report the correct values and so we tried to fix it.

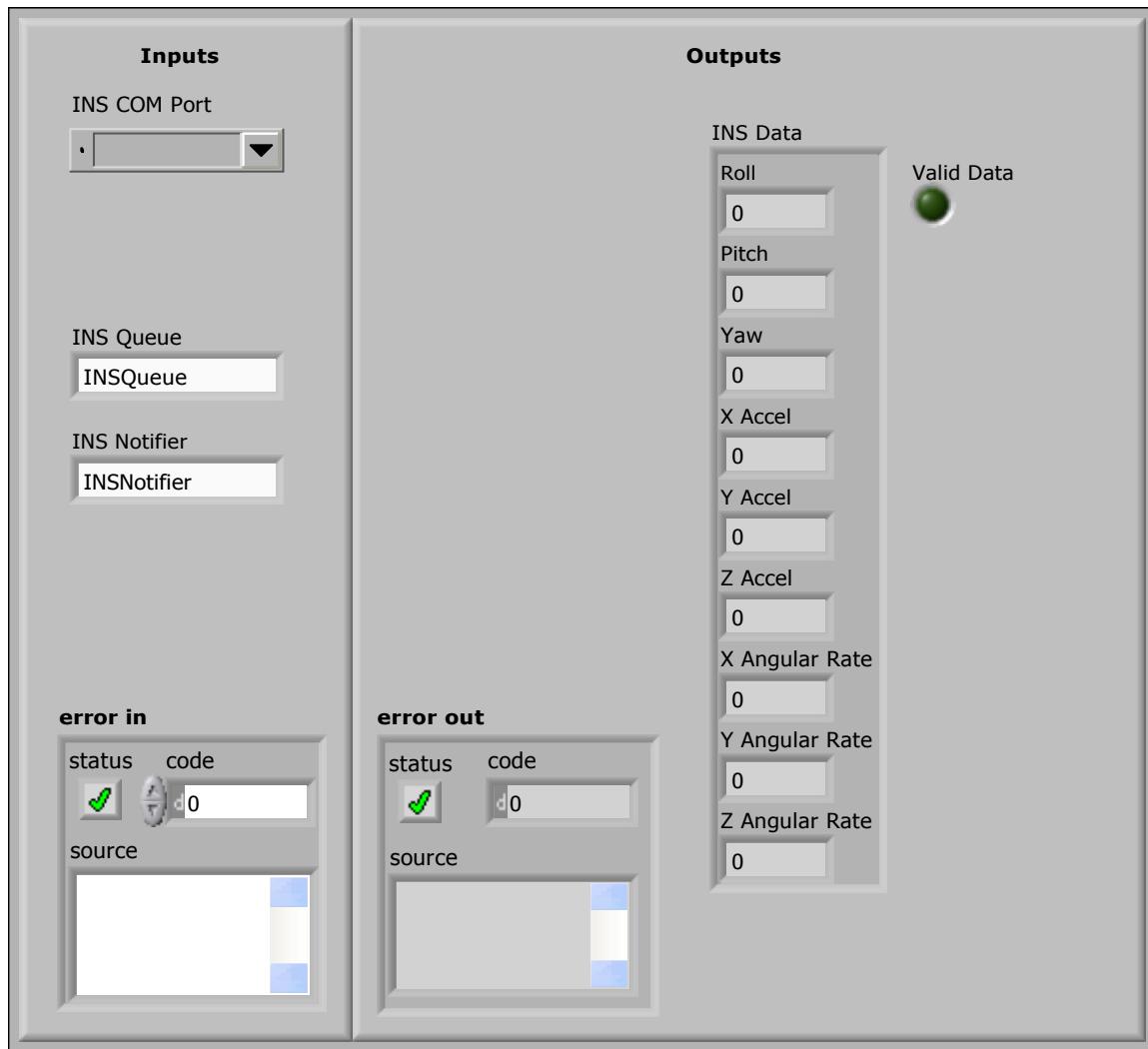
There is also a moving average on the output angles to reduce noise.

Notes from Microstrain's Read Euler Angles VI:

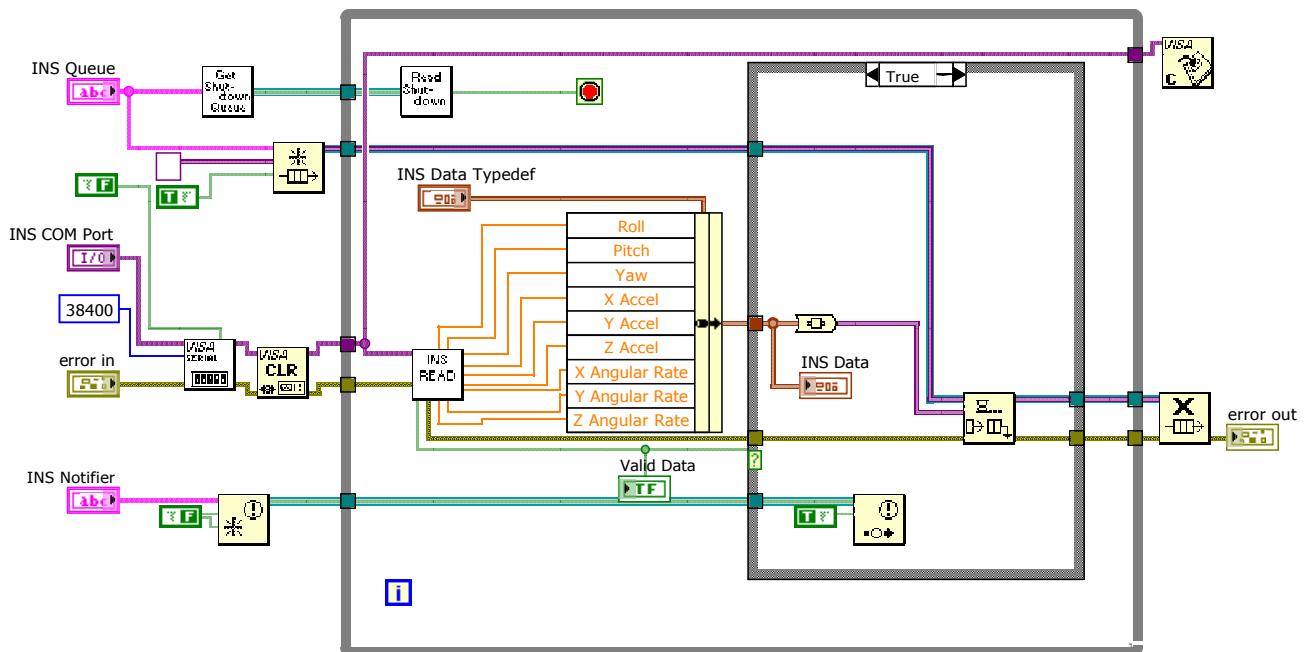
Note: This VI does NOT support the use of the Enable Gyro Stabilization VI. The Euler Angles will always be gyro-stabilized, the components of the Acceleration vector will always be instantaneous and the components of the Angular Rate vector will always be bias corrected.

Note: A mathematical singularity exists at Pitch = +90 or -90 degrees. If pitch exceeds +/- 70 degrees, numerical accuracy is decreased. In applications where Pitch may exceed these values, the orientation matrix should be used.

Front Panel



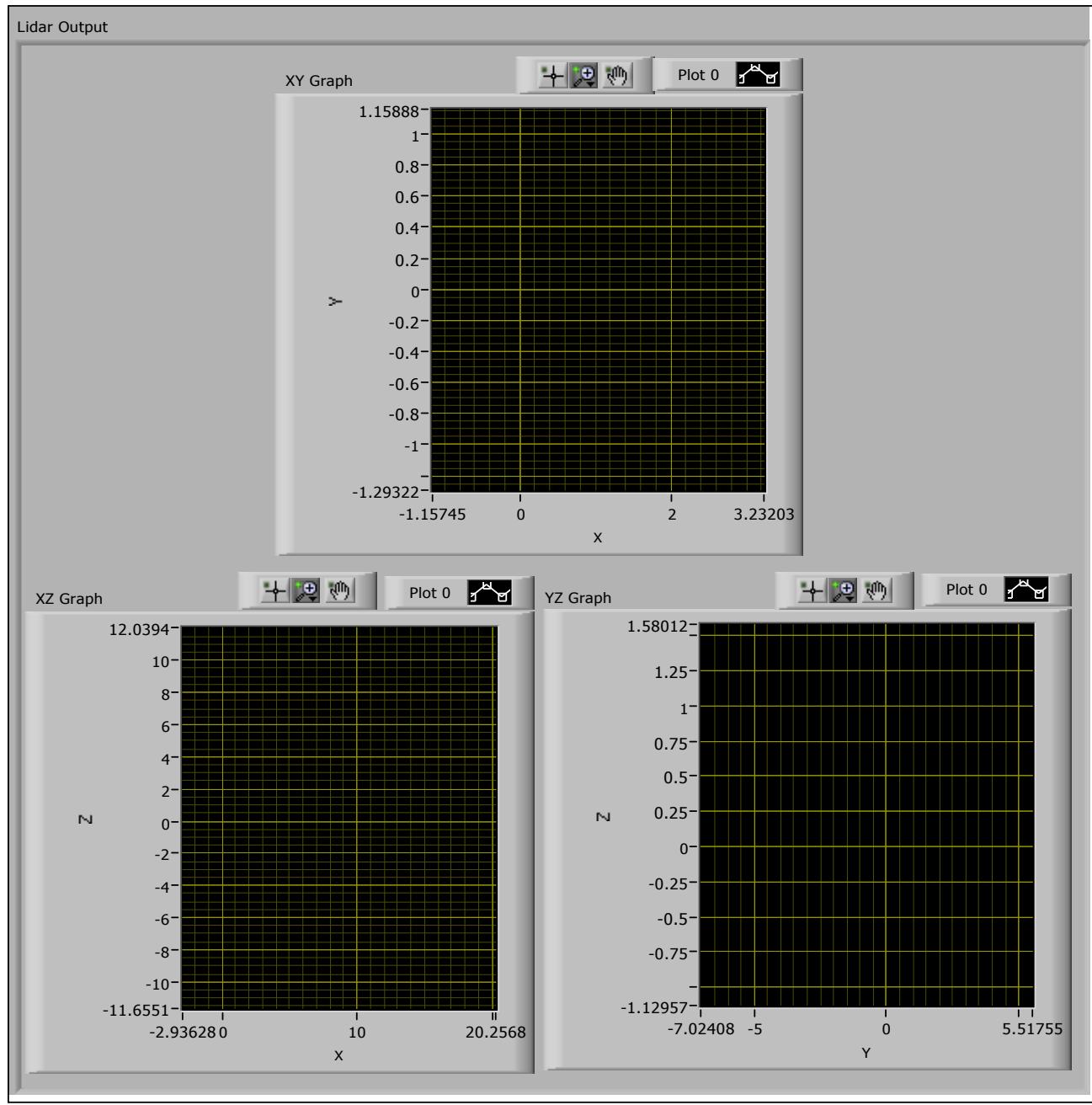
Block Diagram



Connector Pane

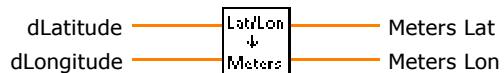
LidarOutputUI.ctl

Front Panel



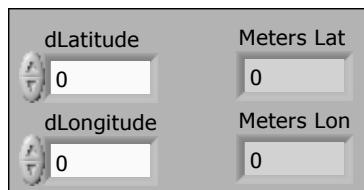
Block Diagram

Connector Pane

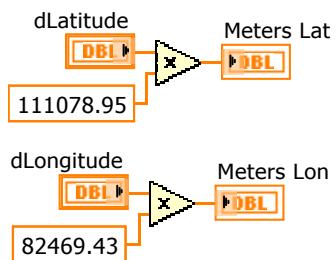
dLat Lon to dMeters.vi

Converts a Lat/Lon difference to distance in meters.

Front Panel



Block Diagram

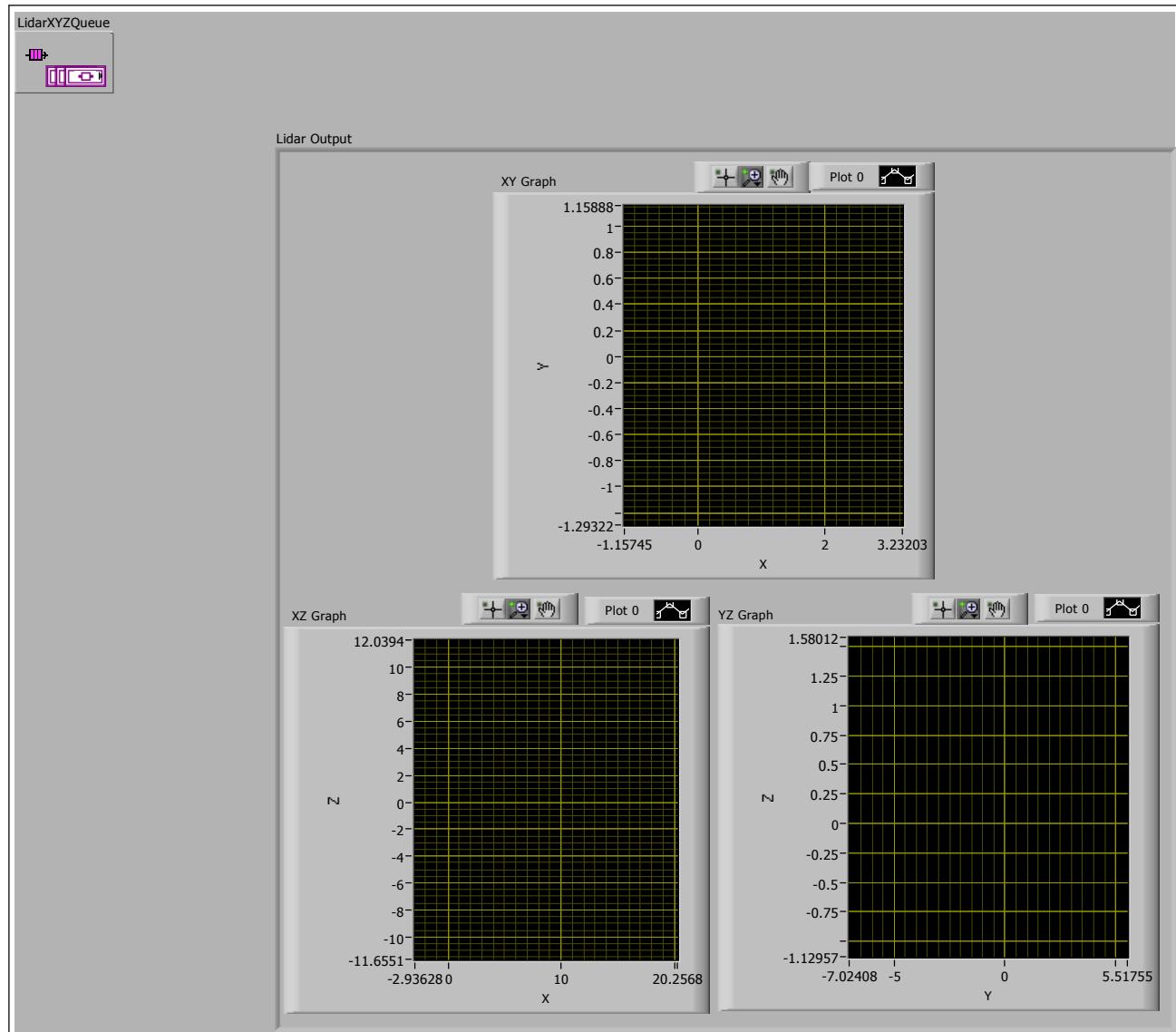


Connector Pane

LidarOutput.vi

GUI function that displays XYZ LIDAR data.

Front Panel



Block Diagram

