RAS at University of Texas at Austin
IGVC 2014 Design Report
Grandpa RASPutin

Robby Nevels, Kevin Gilbert, Gilberto Rodriguez III, Chris Haster,
Joshua Bryant, Jimmy Brisson, Josh James, Xihan Bian
Jonathan Valvano (valvano@mail.utexas.edu)

Faculty Advisor Statement

I certify that the engineering design of Grandpa RASPutin, the robotic vehicle described in this report, has been significant and equivalent to what might be awarded credit in a senior design course.

Signed,

Jonathan Valvano

INTRODUCTION

The IEEE Robotics and Automation Society student branch at the University of Texas at Austin proudly presents Grandpa RASPutin as our 2014 entry in the Intelligent Ground Vehicle Competition. Although Grandpa RASPutin shares the same wheelchair chassis as our 2013 IGVC entry, "Granny" DoloRAS, it has an entirely new frame, components, and software. In this paper, we describe the design and development process we took to build this robot, the object detection innovation we have developed, each component that makes up RASPutin, and the electrical and software systems that provide it with power and intelligence, respectively. We conclude with a performance analysis of Grandpa, assessing how we expect to perform at the competition in June.

DESIGN AND DEVELOPMENT PROCESS

For the past two semesters, our team has met biweekly to work on RASPutin, as well as to plan what to do for the subsequent weeks. Throughout the first semester, we focused on design of the frame using CAD tools and design of the software through data flow diagrams, iteratively redesigning these systems week by week until we were satisfied with them before building and programming. Once we began working on programming, we made small alterations to the design. Fortunately, the frame was built in the same way that it was designed, except for a few mounting differences, as seen in Figure 1. Our most current software design is detailed in the Software System below.

Safety was a major concern when designing the system. We made sure to include a dead man switch on the remote control used for testing. The switch is a button that has to be held down in order to control the robot. If it is released, in the case of the controller being dropped, the robot will stop moving. We also included the mandatory wireless and manual emergency stops, which are detailed in the Electrical System section below, as well as a light that flashes to indicated that the robot is in autonomous mode.

Reliability: code review of programs written for microcontrollers as well as computer, git version control to keep track of code changes, remote-controlled testing during demo days to practice bringing components online; durability: blanket/cloth to protect electronics in case of rain, permanent mounting of electronics and sensors to prevent accidentally unplugging them.

In order to develop reliable software, we used the Git version control system to track the history of the codebase as well as collaborate between people working on different machine. Robby was designated as team leader and in charge of the Extended Kalman Filter, Jimmy and Gilberto were tasked with getting data from the GPS and VN200, Chris was creating visual representation of the data, Kevin was in charge embedded system integration, Xihan was working with the electrical system, Josh James was in charge of vision processing, and Josh Bryant did the mechanical design.

DESIGN INNOVATION

Vision processing as the only obstacle detection

Our design uses only a pair of $40 cameras for obstacle detection through vision processing. Cameras are the most cost-effective sensors available to consumers, and designs based on only cameras have many useful applications due to their low cost and availability.

While limiting ourselves to cameras reduces the total information our robot can obtain from the world, this limitation provides us with several benefits:

1. Wiring is simplified due to the reduction of sensors.

2. Software is simplified due to decreased datapoints.
3. Most of the issues associated with sunlight are removed.
4. Cameras are difficult to damage and easy to replace, and having fewer sensors reduces the number of issues that can occur.
5. The overall organization of our data flow is reduced to a simple pipeline from cameras, through vision processing, and ultimately through to the motor controllers. Modularity of the high-level algorithms is increased due to the assumption of only using cameras.

RASLib - Separation between drivers and application control

To reduce the complexity in our embedded system code, the software was separated into two components which were independently designed. One part encapsulates the driver level code for our various peripherals, and the other contains the application level code for functions necessary for our robot's operation. This allows an increased modularity in our design and a separation of software goals.

A driver library was written titled RASLib, which contains drivers for servos, motors, sonars, rangefinders, and encoders, and also provides a task based scheduler, cyclic linked-list based PWM emulation, and support for serial communication over UART. RASLib has been significantly unit-tested and integration-tested to provide a robust base for higher-level application code.

COMPONENTS

Selection of our computer was based on two main concerns: ability to perform vision processing, and power supply isolation. We selected a Lenovo ThinkPad Edge E431 laptop. This laptop contains an Intel i7 CPU and an Intel 4k integrated GPU suitable for vision processing. The choice of a laptop was to reduce power dependencies and wiring associated with power. All USB based components run directly off the battery in the laptop and prevent power issues that stem from powering both the laptop and motors from the same source.

To provide absolute localization, we use a VectorNavRugged 200 GPS/IMU combined with a Trimble GPS/GLONASS. The VectorNavRugged allows for two meter GPS accuracy when in motion and five meter accuracy at a stand still. The GPS receiver includes 50 channels. This module also has a gyroscope with a measuring range of 2000 degrees per second at a bandwidth of 256Hz, an accelerometer with a range of 8g and a bandwidth of 256Hz, and a magnetometer with a range of 2.5 Gauss and a bandwidth of 200Hz. The Trimble BD982 provides with position with an accuracy of 0.25m + 1ppm Horizonally and 0.50m + 1ppm Vertically, and sends this data over ethernet as it receives it at 50Hz with a maximum latency of 20 ms.

We used the Texas Instruments LM4F120XL Launchpad as the interface between RASputin's control system and the onboard sensors and motor controllers. The launchpad was chosen due to its small form factor, low price, ease of replacement, and ability to both drive and read a high number of Input/Output devices.

We wanted a nice, cheap, and reliable chassis, and based on our experience from last semester, we decided to keep the previous driving base, a Jet 3 Power wheelchair. This wheelchair was built to carry a 300 lbs person at a speed of five mph. The wheelchair operates on a differential drive train, having two motors on either side. This means that rotations are handled by turning the two motors at different speeds, allowing for a zero point turn radius.

MECHANICAL SYSTEM

In this section, we describe the chassis of the robot. The materials and construction method were chosen to create a modular robot that is easy to repair. The modular design will make for easy modification by future teams. First we describe the construction of the chassis, then we explain its design.

The chassis was designed for both strength, and ease of modification and repair. The chassis is made of extruded aluminum square tubing and aluminum sheet metal braces that are riveted together. The aluminum tubing and sheet metal braces were chosen because they are easy to modify with basic tools, a circular saw with a non-ferrous metal cutting blade and a jigsaw with a metal blade, respectively. The rivets are easy to replace and install with a hand drill and a rivet gun. The materials can be sourced from most local hardware stores. The materials being easily sourced and the use of a small number of power tools lends towards an easily repaired and modular chassis.

The chassis was designed to attach to the wheelchair base acquired for a previous competition and to accommodate all of the sensors we plan to use, now and in the future. Like DoloRAS, RASPutin uses a stripped down wheelchair base to provide a robust and stable platform with the chassis bolted to the chair mount. The modular structure makes the chassis easy to modify to accommodate different electrical and sensor systems for future teams. The frame is designed accommodate up to: three 12V motorcycle batteries, one laptop, two DC-AC power inverters, two GPS antenna, a payload per the 2014 IGVC rules, and a large weather-proof lidar (Figure 1).

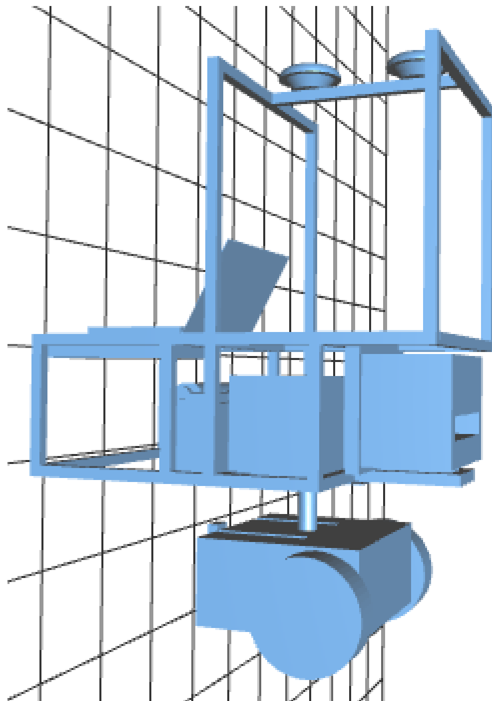


Figure 1: CAD design of robot frame (left) and picture of constructed frame (right)

ELECTRICAL SYSTEM

In this section, we describe the electrical system of the robot. There are 2 12V UB12350 sealed lead universal battery in series, providing 24V DC for the motor, GPS and router. The laptop's battery provides power for the laptop and peripheral attached. There are two emergency stop, one onboard

manual button and one wireless switch, will cut off power supply to the motor when necessary. Figure 2 shows the power distribution of the system.

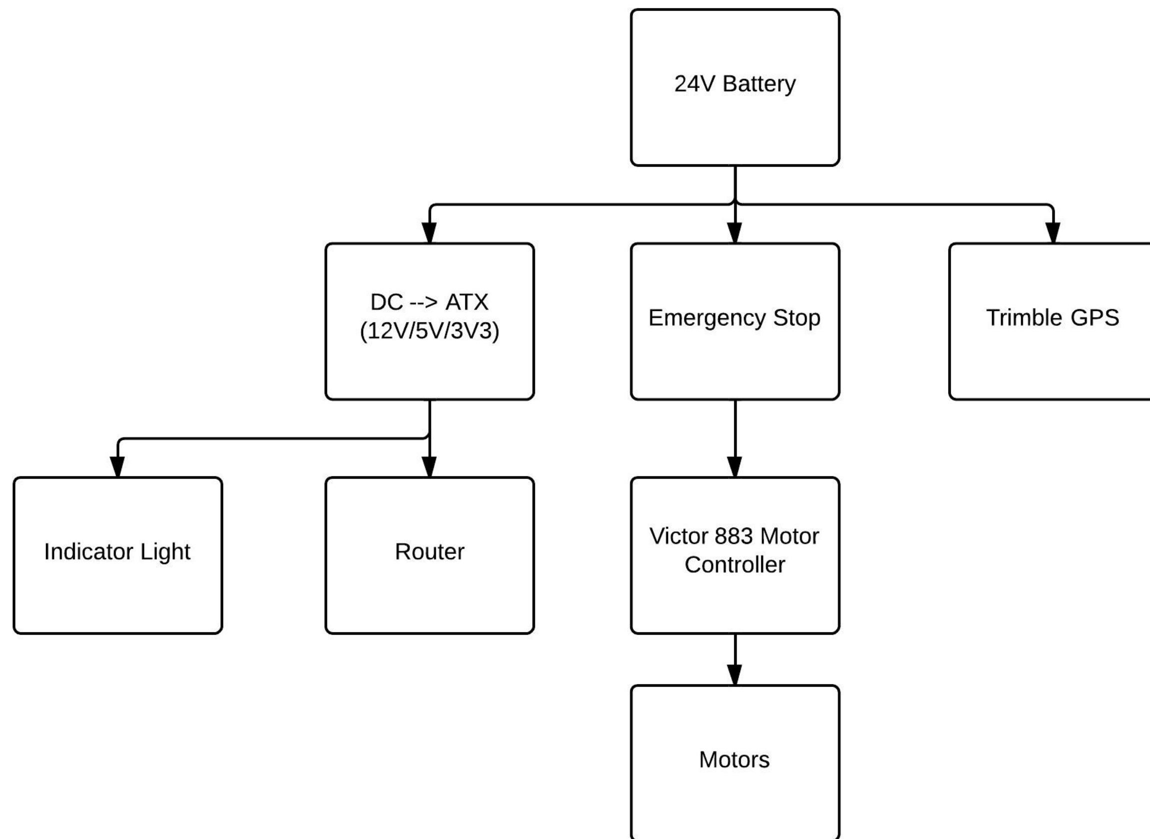


Figure 2: Power distribution diagram with emergency stops.

Batteries and Power

The battery system consists of two 12V sealed lead universal battery in series. In the current state, the batteries are charged separately by an AC to DC charger at 500W. We have searched for a more powerful charger that will provide more than 1kW power in order to charge both batteries at the same time, but due to budget restraint we are not able to afford one. The battery will supply power to the driving motors, router, and GPS module. The router has a Vin of 12 volts, and GPS module has Vin at 19V. Therefore a DC to DC converter regulator was used to convert 24V to 19V. The regulator module's maximum output is 190W, which will be sufficient for providing power to router and GPS module. Further study will be done in order to determine if this module will also provide additional charge to the laptop.

Emergency Stops and Motors

There are two driving motor on the base of the robot, each driven by a Victor 883 motor controller which is opto-isolated from the its PWM inputs. The power of the motors is provided by the battery after passing through the emergency stop module. The emergency stop module contains two methods

that will cut off the power supply to the motors: the onboard kill switch, and the wireless kill switch. The onboard kill switch is mounted in the back of the robot, easily accessible during any type of situation. The wireless kill switch is independent from the rest of the wireless module, making it more reliable. When emergency stop is activated, the control signal will still be received by the controller, and the motor will not have power to operate.

SOFTWARE SYSTEM

In this section, we describe the software running on both the computer and the microcontroller. RASPutin's main processing is done on a laptop running Ubuntu 12.04 with the Robotic Operating System (ROS) version Groovy. ROS is a meta operating system built on top of Unix that creates a framework for robotics application. A microcontroller, connected to the laptop via USB, runs custom code to control and interface with RASPutin's motors, encoders, and safety light. Figure 3 shows a high-level data flow diagram of all computation done between receiving sensor data and sending motor commands.

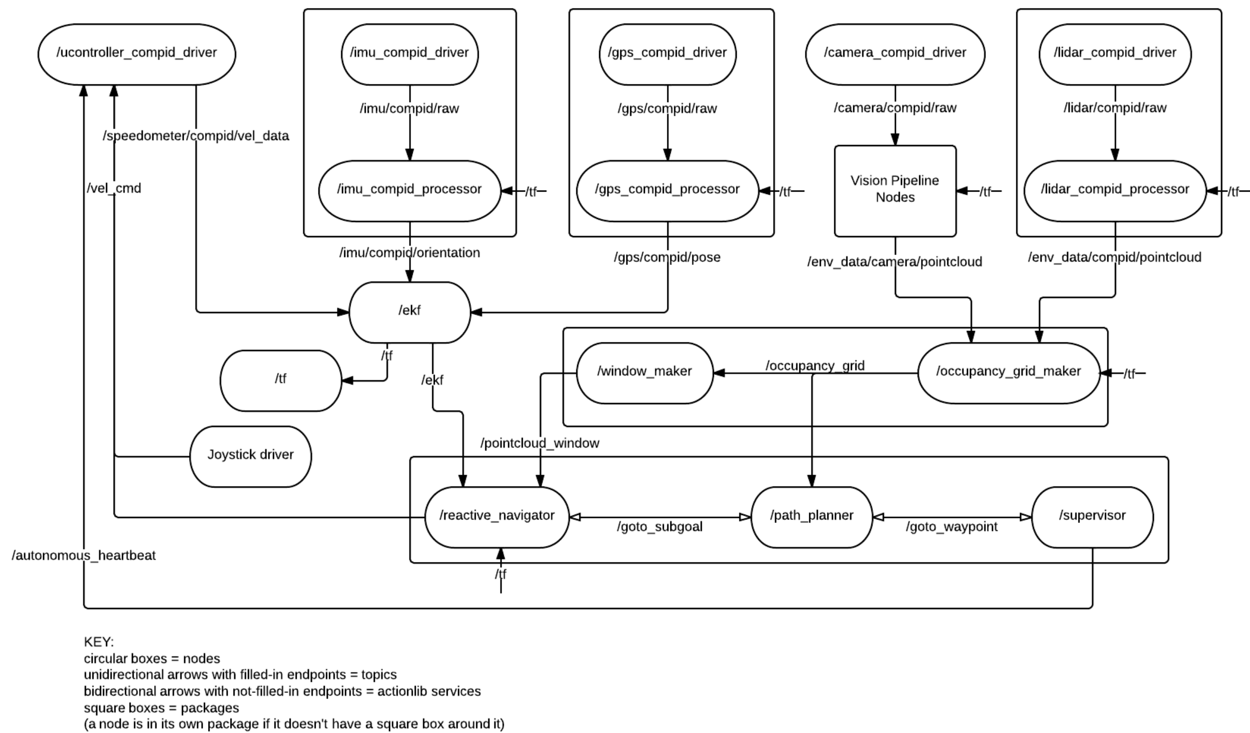


Figure 3: ROS Data Flow Diagram

Component Drivers

The ROS ecosystem provides several drivers for common components. We used the ROS usb_cam webcam driver to receive data from both our F100 cameras. We also ran ROS's joystick driver for the controller used during testing. RASPutin's predecessor, DoloRAS, used the VN 200 as a GPS and IMU sensor, so we reused its driver. We also wrote new custom drivers for the BD982 GPS receiver and the LM4F microcontroller.

Localization and Sensor Fusion

Like DoloRAS, RASPutin uses an Extended Kalman Filter in order to determine its location in the world. A Kalman filter is a linear estimator that uses updates from various sensors and confidence levels in each sensor to produce a state estimation vector. It uses a linear transition of robot's state from one prediction to the next, while at the same time maintaining a matrix of variances, or confidences, corresponding to each element in the state vector. This state state estimation process is robust because it can tolerate errors or even failures in a set of sensors and still provide a reasonable state and error estimation. An Extended Kalman Filter (EKF) approximates a nonlinear state transition function, which is more suited for a differential drive robot like RASPutin. For more information on Kalman filters and EKFs, see [1].

Rather than using an existing implementation, we have written our own EKF to have full flexibility with how state estimates are computed and updates are used. Our EKF estimates a state vector that consists of position (in meters), heading, linear and angular velocity, linear acceleration, roll, and pitch. We are using the BD982 GPS receiver for global position updates (using x/y meter updates computed from GPS coordinates using a local reference point), the VN 200's accelerometer and magnetometer for roll/pitch/yaw updates, and the motor's encoders for angular and linear velocity updates.

Obstacle Detection

RASputin uses the data from two 120 degree webcams to detect obstacles. The two cameras are arranged such that their fields of view overlap ~40 degrees in front of the robot, which leaves a total field of view of ~200 degrees.

Obstacles are defined as the inverse of the image space that is classified as drivable. Drivable space classification is performed primarily through color segmentation of the ground. To perform this segmentation, colors that are within one of K rectangular prisms ("color regions") in HSV space are considered drivable. These color regions are determined automatically from a part of the image that is known to always correspond to the ground ("ground region") under all normal circumstances. First, a color palette is generated by collecting the colors of all the pixels in the ground region. Next the color palette is clustered into the K tightest-fitting color regions by using OpenCV's K-means algorithm and finding the minimum and maximum values of all the pixels in each cluster. Finally, the pixels in the entire original image that match one of the color regions are classified as ground and the other pixels are classified as obstacles. The automatic classification process can be slow, so the color regions are only updated at 1hz, while segmentation still occurs at the full 30hz.

The obstacles in the segmented image are then mapped via a perspective transform to a grid aligned to the ground, which is assumed to be reasonably flat. This grid is interpreted as a point cloud, and statistical outliers are removed using the Point Cloud Library (PCL) to decrease the probability of false-positives. The filtered point cloud is then converted into a LIDAR scan for compatibility with existing ROS nodes.

Mapping and Navigation

Mapping and navigation are performed as a single unified unit to decrease code complexity. We found that a more modular, isolated approach required extra software complexity. This complexity was caused by additional overhead by one node estimating or reading the state of the other. Mapping and navigation are both performed with ROS libraries using an occupancy grid and A* pathfinding

respectively. Collisions are avoided with reactive navigation. Performance of our mapping and navigation system is improved by a Local Dynamic Window.

Mapping uses a occupancy grid which is the default implementation provided by ROS through an included package. During testing, we found that an occupancy grid outperformed both a direct point-cloud and particle filter while providing a simple model that was easy to reason about.

Navigation was also performed through provided ROS libraries and is implemented with A* path planning through the provided occupancy grid. Because we combined mapping and navigation into one unit, we were able to make assumptions about the dimensions and resolution of the map to optimize our throughput and memory consumption. To prevent collisions with obstacles, we used a reactive navigator that responds to flattened obstacles provided by vision processing. The reactive navigator performs simple ray-tracing on each possible path available to the robot.

To further increase our data flow rate and lower the processing load on our system, we added a Local Dynamic Window to our navigation pipeline. The Dynamic Window limits the quantity of data directed to the reactive portion of our navigation to increase the update speed and response time. The response time of the reactive navigator limits the overall speed of the robot as it is the only component responsible for preventing collisions with obstacles.

Microcontroller code

The embedded system design was broken down into two layers: the ROS node that communicates with the Microcontroller (MCU), and the code running on the MCU. The code on the MCU is further divided in two by purpose: the protocol code, and the velocity control code.

The LM4F ROS node, running on the central computer, communicates to the MCU over a serial link that uses JSON for serialization. JSON was chosen to standardize communication for reusability. This node accepts commands from the joystick and reactive navigation through the ROS topic /vel_cmd that it subscribes to. The data that is received on the /vel_cmd topic is translated by the LM4F ROS node to JSON and sent to the MCU. The LM4F node also receives odometry information from the MCU as JSON and converts it into a message that is sent to the EKF. Mapping and driving logic is performed on an abstracted basis where world interactions are calculated in terms of linear and angular velocity. This allows us to create the software core on a separate thread from the embedded system design. The ROS Twist message format was chosen as the message type for the velocity command node towards this end. Three axis velocity components in terms of linear and angular directions can be passed on to the microcontroller ROS node.

The protocol code on the MCU uses a C library for parsing JSON known as Jsmn . This library lexes and parses the JSON format. The protocol code on the MCU sends the content from the JSON parser to the velocity control code. RASLib as mentioned previously is a set of functions designed to streamline peripheral interaction on the LM4F120-EK using StellarisWare, a TI software suite to facilitate programming on ARM based embedded systems by abstracting register and memory mapped I/O interaction with function calls. A simplified call graph can be seen in figure 4. A set of functions are tied to each command, which when received, calls the corresponding handler (See Figure 6 for a data flow graph for the ROS node an embedded system).

The MCU velocity control code maintains velocity using a set of encoders on each motor to provide feedback in a Proportional-Integral-Derivative (PID) Controller. The basic design of the velocity control code can be seen in Figure 5. The output of the PID controller is measured in delta

encoder ticks, which are converted to a duty cycle to forward as a Pulse Width Modulated (PWM) signal to the motor controllers.. This allows us to map the rate of change of the encoder readings to revolutions of the output shaft to the speed the wheels. Each motor is individually controlled within the PID loop to ensure that each motor is receiving the correct power in relation to the other motor.

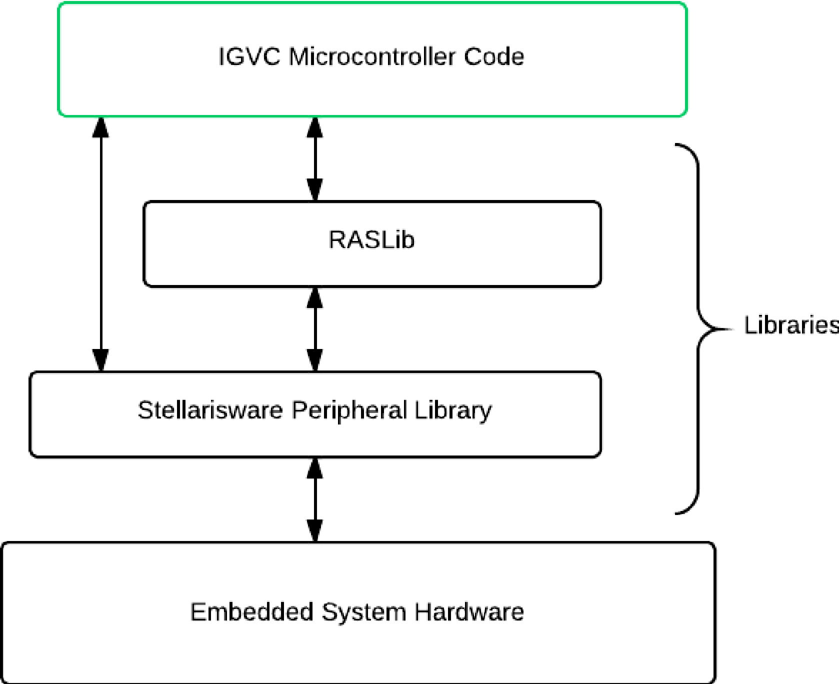
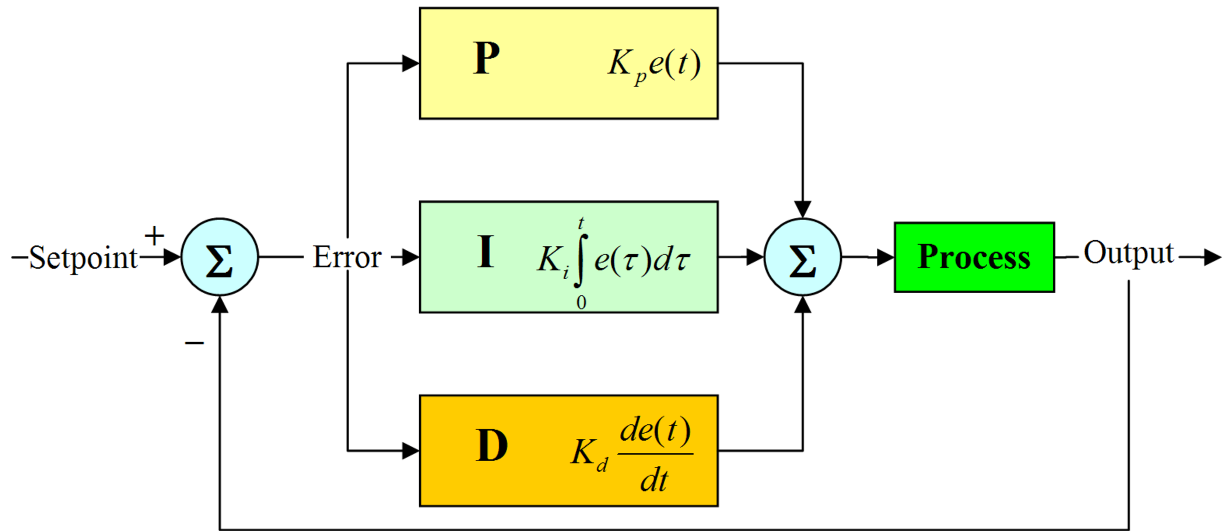


Figure 4: Hardware to Software Level Library Call Graph



http://radhesh.files.wordpress.com/2008/05/pid.jpg

Figure 5: PID control diagram

Command	Name	Operation
SPLM	Set Power Left Motor	Sets the raw power of the left motor (percentage duty cycle)
SPRM	Set Power Right Motor	Set the raw power of the right motor (percentage duty cycle)
SVLX	Set Linear Velocity	Sets the goal for the linear velocity based off of the center of the robot. (Meters/Second)
SVAX	Set Angular Velocity	Sets the goal for the angular velocity based of the center of rotation for robot. (Radians/Second)
RSTE	Reset Encoder	Resets the encoder values

Table 1: JSON command formats

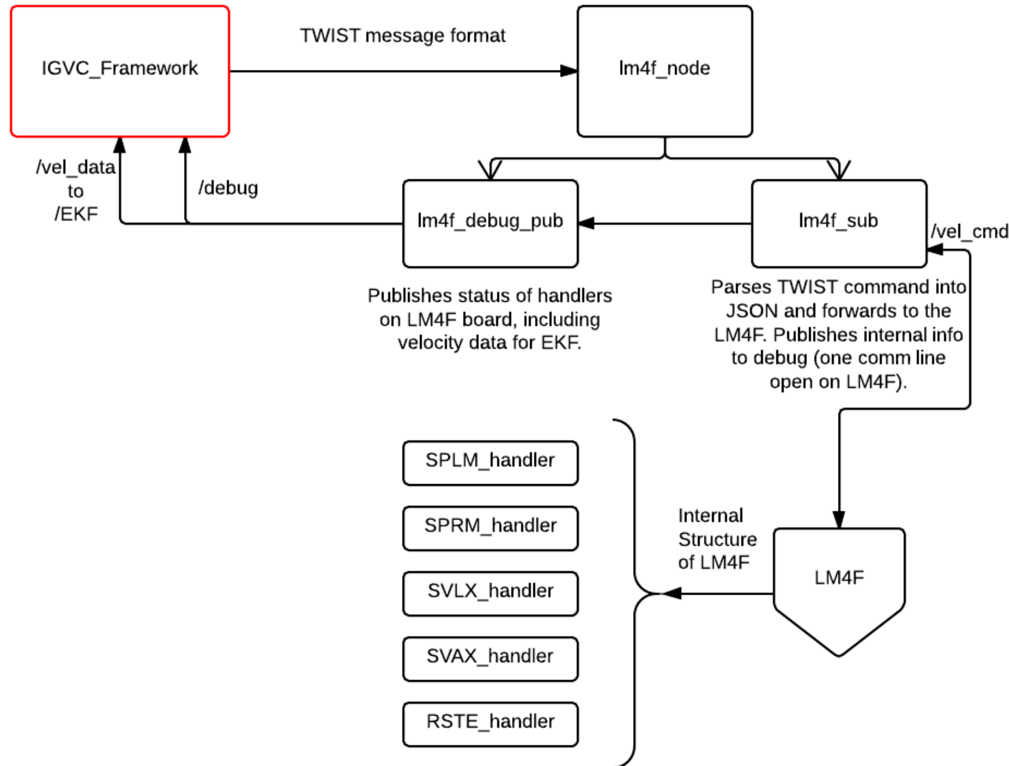


Figure 6: Data Flow ROS Node

ANALYSIS OF EXPECTED PERFORMANCE

We have not fully tested RASPutin in the field; however, we can make estimations based on observations of how DoloRAS performed at last year's IGVC competition, testing on individual components, and from partial demonstrations we have done with RASPutin in the last month.

The wheelchair that both DoloRAS and RASPutin are built on was made to be a reliable medical device capable of carrying a 300 lbs person at a speed of up to five mph. During testing this year, we found that RASPutin could achieve a maximum speed of about five mph. In the past, we have seen the same chassis and motors carry a weight as least as much as RASPutin's up a ramp of 15 degree inclination when going about four mph. From observations during demonstrations, we have found that the wheelchair batteries can last for several hours without needing to be charged.

We have not yet tested navigation software on a real field, but we have observed ROS mapping and navigation code running in a simulated environment with good results. Because a full map is being used to plan paths, we do not expect complex obstacles like switchbacks, center islands, or dead ends to make any negative impact on RASPutin's ability to reach waypoints.

Sensing: Using our dual-camera-based detection method, obstacles are registered when they are 5m away, within a 200 degree range centered on the front of the robot. From testing, we have found that the sensors used to localize are highly accurate (detailed in the Components section above), so the

accuracy of waypoint navigation will not be limited by hardware. Velocity control at the embedded system level has also proved to be accurate to within 0.5 meters/second during testing.

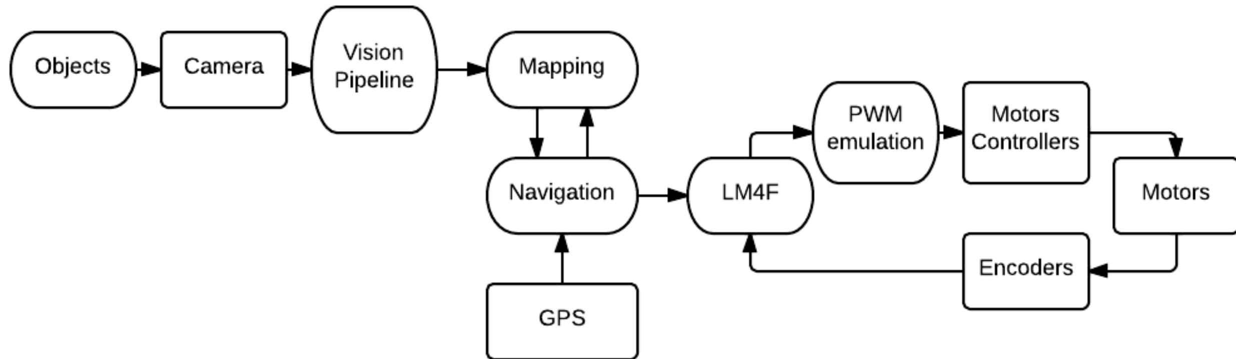


Figure 7: Data Flow

CONCLUSION

During our design iterations, we focused on three core concepts:

1. Simplicity
2. Modularity
3. Reusability

We diverged from previous years by largely simplifying our design. We replaced the central computer with a laptop, rather than the motherboard based system that we employed from our 2013 IGVC entry. This not only removed the larger footprint of the computer components itself, but removes the DC-AC converter and larger power supply unit we had as well. The wiring was hugely cut down by the use of the laptop and the modularity increased as the core computer can be easily swapped out. The number of sensors was reduced as well to only include two cameras, encoders, GPS, and an IMU. This reduces the data flow pipeline, reduces power consumption, and still allows for redundancies in data collection to improve our probabilistic model in the EKF.

Modularity was the next design core we considered. ROS was a large help in this feature. It allowed us to design data flow irrespective of specific hardware. The specific cameras, motor controllers, microcontroller, encoders, GPS, etc could be changed or swapped out with little to no impact on other nodes. This leads to the next major point, reusability. To allow us to improve in the future, documentation and version control through Git was extensively used. In addition, because we used ROS nodes components and can be changed and updated without rewriting the entire code base.

Our current design is intended to be reusable and robust, providing an advanced starting point for next year's iteration in our design.

ACKNOWLEDGEMENTS

VectorNav, TI, Trimble, OmniSTAR, UT ECE Department, Dr. Valvano

REFERENCES

[1] Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. Probabilistic Robotics. Cambridge, MA: MIT, 2005. Print.

APPENDIX A: TEAM AND HOURS WORKED

Total Estimated Hours: 211

Person	Department
Robby Nevels	CS, ECE
Chris Haster	CS, ECE
Josh James	ME, ECE
Jimmy Brisson	ECE
Kevin Gilbert	ECE
Gilberto Rodriguez III	ECE
Joshua Bryant	ECE

APPENDIX B: COMPONENT COSTS

Component	Cost
VectorNav 200 IMU	$3,200 (donated)
Trimble BD982 GPS Receiver and Dual Antenn	$13,500 (donated)
WideCam F100 Cameras (2)	$80
Lenovo ThinkPad Edge E431 Laptop	$700
Encoders (2)	$24
Jet 3 Power wheelchair (including motors)	$40
Router	$37
LM4F Microcontroller	$10
Frame materials	$160
Total	**$17751**