INVESTIGATOR
UNMANNED GROUND VEHICLE WITH NATURAL HUMAN INTERFACE FOR SURVEILLANCE AND RECONNAISSANCE

By

Kevin French

A THESIS PRESENTED TO THE DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR OF SCIENCE WITH HONORARY SUMMA CUM LAUDE

UNIVERSITY OF FLORIDA

2015

Abstract

InvestiGator is a two wheel drive surveillance and reconnaissance unmanned ground vehicle (UGV) that can continually run for upwards of twelve hours. After twelve hours the robots power supply can be hot swapped so that there is never any down time. The InvestiGator is controlled via a natural human interface over a Wi-Fi network. A Leap Motion is used to implement the natural human interface by tracking the user's hand. There are two modes of operation direct remote control and waypoint navigation. Direct remote control allows the user to directly control the motion of the UGV. Waypoint navigation allows the user to place waypoints in a virtual environment. The robot will then move to the corresponding location of the waypoint in the real world. To generate the virtual environment a Microsoft Kinect is used in conjunction with a visualization software known as RVIZ. The Kinect provides 4 possible modalities to view the world in, giving the user large amounts of information about the current situation even in complete darkness.

Table of Contents

Table of Figures

Introduction

The InvestiGator is an unmanned ground vehicle (UGV) designed for exploration, surveillance, and reconnaissance. It is approximately one foot in diameter and uses two driven wheels for differential steering and one caster for balance. Equipped with the Microsoft Kinect and a sonar array, InvestiGator can view the world in 3 modalities; visible light, infrared light, and sound waves. The three modalities can be combined to create two more derived vision spaces, point clouds and disparity images. Robot Operating System has a visualization tool called RVIZ which allows a user to view all 5 of the different modalities at the same time in a single three dimensional space, providing the user with a much greater understanding of the environment then any one modality could on its own. The strengths of each modality can be capitalized on by the user. In the absence of light the infrared, sonar, point clouds, and disparity images act as night vision. In the presence of harsh sunlight the infrared camera will become susceptible to noise but the sonars and the visible light camera will be unaffected. The UGV sends its visual data to a user's Robot Operating System enabled device over a Wi-Fi network. This allows a user to control the UGV while it is in Wi-Fi range. To control the UGV a natural human interface is employed using the Leap Motion. The Leap Motion uses infrared cameras to track the hands of the user. Two control modes have been implemented, direct remote control and a waypoint navigation system. To use direct remote control a user simple places one hand above the Leap Motion and angles it in the direction of desired motion. In waypoint mode a user grabs a waypoint with a pinching motion moves the waypoint to the desired location as visualized by RVIZ places the waypoint and accepts it with a swipe of the hand. Once a waypoint has been confirmed the robot uses a smooth motion control algorithm to generate a smooth path between its current pose (position and orientation) and its final pose.

Features

Below is a list of features of InvestiGator

- The ability to view the environment in three dimensions so that the user can have a better understanding of the environment
- Natural human interface to make the system easy to use
    - Direct remote control to maneuver with fine control
    - Waypoint control to semi-autonomously move the UGV
- Wi-Fi connectivity to operate over a wireless network

- Ability to operate in any lighting condition so that the UGV can operate 24 hours a day
- Power oring so that multiple power sources can be used at the same time
- Power source hot swap so that the system doesn't have to go down when batteries need to be replaced
- One hour minimum runtime so that the UGV has long mission lifetime
- Simple power monitoring to know when to switch the batteries

Structure

Figure 1: CAD rendering of InvestiGator

The InvestiGator is based off of a common small robot design; a circular platform with two wheels and a caster. A custom design structure was used due to the needs of the payload. Most off the shelve platforms are too small to support the Kinect and or prohibitively expensive.

Payload

The structure was designed to carry two motors attached to wheels, one caster, four sonars, one Kinect, and the electrical hardware. To hold the sonars four mounts were added. Tabs where used to hold the wooden mounts in place. Since the InvestiGator moves forward most of the time three of the four sonars were placed in the front to allow for higher sonar definition in the front while only one was placed in the back. On the same platform the two motors were attached near the front so that only one caster would be needed and so that the caster would be pulled instead of pushed. A raised platform was added to support the Kinect and provided additional surface area for electrical hardware. The Kinect

works best when it is at least six inches of the ground, with the wheel height and the four inch spacers for the raised platform the desired six inches was achieved.

Locomotion

The InvestiGator employs two motors to perform differential steering. Differential steering allows for simple control mechanisms which are discussed in the inverse kinematics section of software. Sixty millimeter radius off road foam RC car wheels were used for the wheels. These wheels are large enough to allow the InvestiGator to drive over small obstacles and have superb traction. Pololu 75:1 low power metal gear motors were chosen for their ample torque, low power requirements, and included encoder.

Electrical

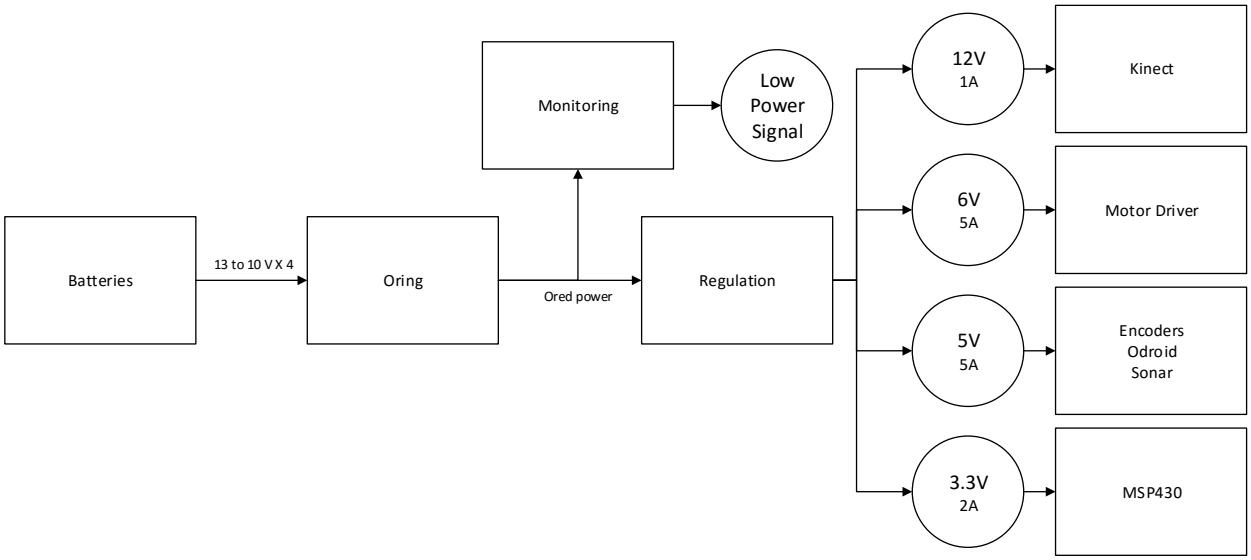


Figure 2: Electrical system overview

The electrical system is composed of a power oring board, a power regulating board, a control board, an embedded computer, two encoders, two motors, a dual motor driver, four sonar, and a Kinect. Up to four power sources between 8.5 and 13 volts can be supplied to the power oring board at the same time. The power supplies are ored together and then passed to the regulation and monitoring board. The regulated power is then distributed to the devices or boards that require the particular voltage. The control board is the middle man between the embedded computer and the electrical hardware. The control board takes commands from the embedded computer and performs the desired operation.

Sensor Selection

To acquire the ability to see in three dimensions a depth sensor was required. The Kinect is a commercially available and cheap RGB-D sensor where RGB-D stands for red green blue depth. The Kinect can actually see more than what was originally intended for the project. The Kinect has plenty of community support and multiple software development kits and libraries. The price, community support, and additional data provided, made it the ideal sensor for the InvestiGator. The Kinect does constrain operation too indoors during the day since the technology uses structured infrared light which the sun interferes with. Being constrained to the indoors was not a large concern since the mechanical design isn't well suited to the outdoors anyways. The Kinect also has a minimum range for depth sensing.

To counteract the minimum range four sonars that work in the dead zone of the Kinect were added. In addition to providing close range object detection the sonars can operate outdoors with sunlight.

Encoders are needed to perform state estimation based on forward kinematic equations of motion. A Hall Effect quadrature encoder with 896 counts per revolution was chosen. These encoders are used to measure the speed of revolution of the motors.

Battery Selection

Three cell lipo batteries were selected as the power source. Lipo batteries have very high energy density and so without adding much weight to the UGV the UGV obtains a significant amount of energy. Three cell batteries were chosen because the voltage regulation circuitry is limited to a maximum voltage below that of the maximum four cell lipo voltage. Running the system under normal conditions with one 2200 mAh lipo battery gives approximately three hours of runtime which is well above the one hour minimum runtime.

Power Oring

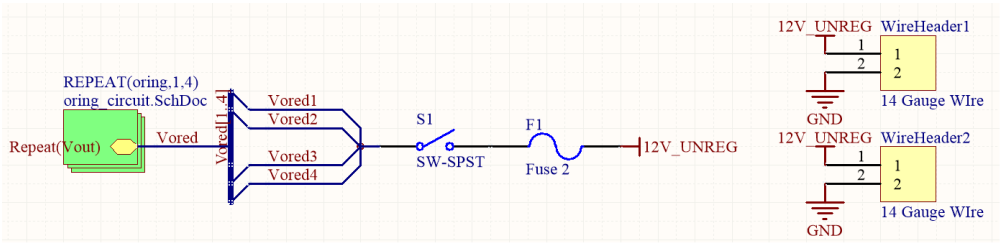


Figure 3: Top level oring circuit

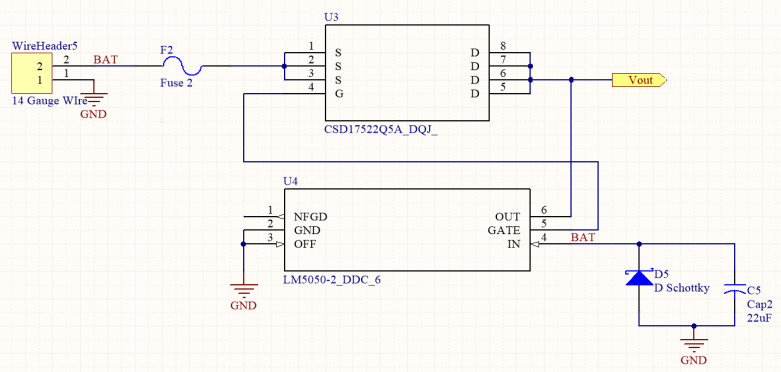


Figure 4: Repeated internal circuit for oring

The power oring board gives two very useful features to the robot, the ability to use multiple power sources and the ability to hot swap power sources. Four identical circuits are used to provide four input channels to the ored voltage. Each circuit acts as an ideal diode only sourcing current when the line voltage on the output is less than the input voltage. When more than one voltage sources are attached to the inputs only the highest voltage will have current drawn from it. Once the sources are approximately equal the voltage sources will have current drawn equally from them. To get an ideal diode behavior, the Texas Instruments LM5050-2 chip drives a power mosfet.

Hot swapping is achieved by leaving at least on voltage source on an input then placing the fresh source on a different input at which point the old source can be removed.

Each input has a schottky diode for reverse voltage protection. In addition a ten amp fuse is placed between each input and the mosfet. After the voltage sources are ored together they are passed through a switch which is the off on for the UGV. After the switch there is another 10 amp fuse.

Power Regulation

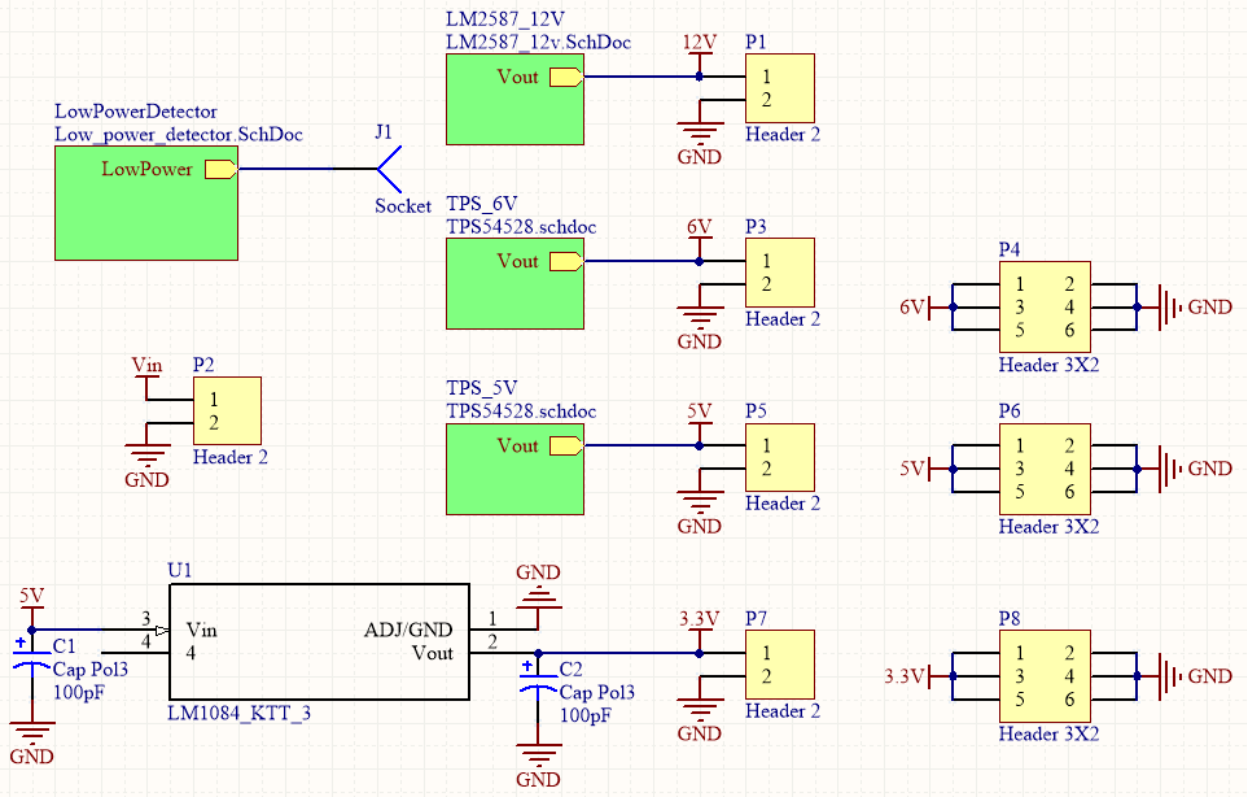
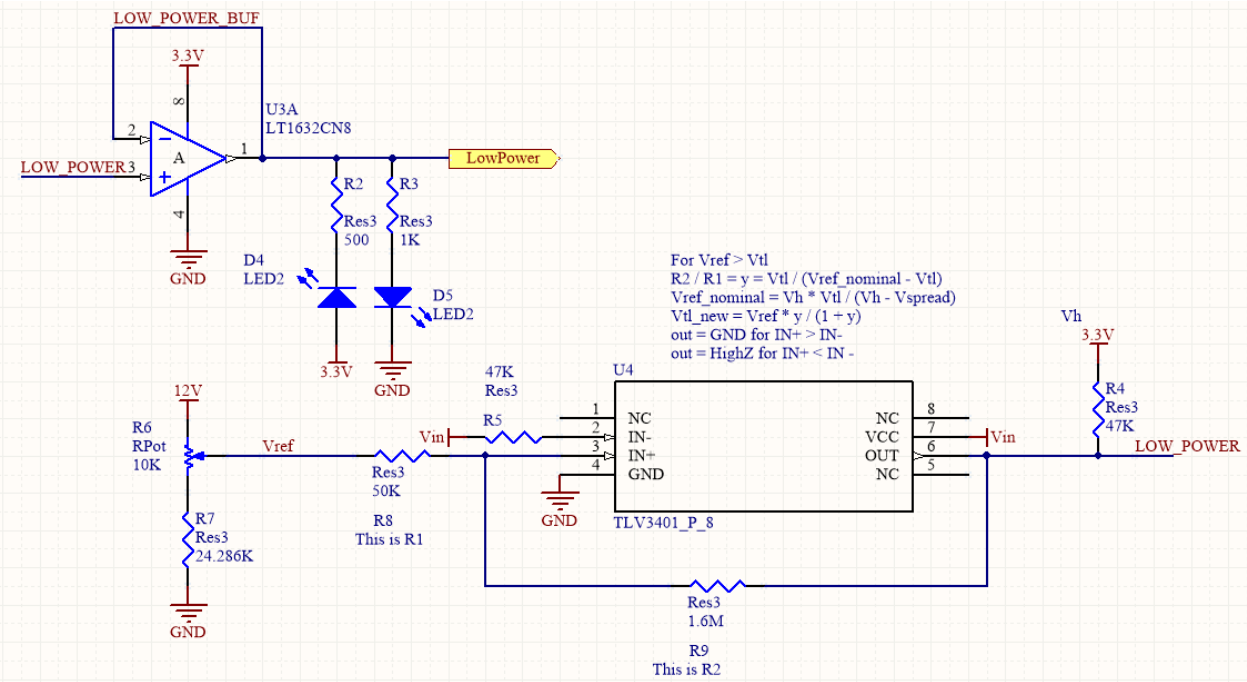


Figure 5: Top level power regulation circuit

Three switching regulators and one linear regulator are used to provide the four voltage levels required of the UGV. A 1A flyback regulator is used to regulate the 12V output since at lower input voltages the voltage will need to be boosted up to 12V. Two nearly identical 5 amp buck switching regulators are used to produce 5 and 6 V voltage rails from raw input. Each of the feedback circuits has a different value resistor providing the two different voltage levels. A 5A linear regulator fed from the 5V switching regulator generates the 3.3 V voltage rail. The switching regulator circuits can be found in Appendix A. Additionally a low power indicator circuit is provided.

Low Power Indicator



The low power indicator uses a comparator as a Schmitt Trigger to provide the low power signal. The input to the comparator is the unregulated ored power from the oring board. The voltage reference comes from an adjustable voltage divider with the regulated 12V source as its high voltage. The Schmitt Trigger is set to have about 0.1 V gap between the low and the high voltage threshold. The comparator is not designed to source current and is railed between ground and the unregulated input voltage so an op-amp is used as a voltage buffer that rails at 3.3 V and ground. The buffered voltage is fed to an active high and an active low led to indicate the status of power and it is fed to an output pin so that the control board can read the status of low power.

Control Board

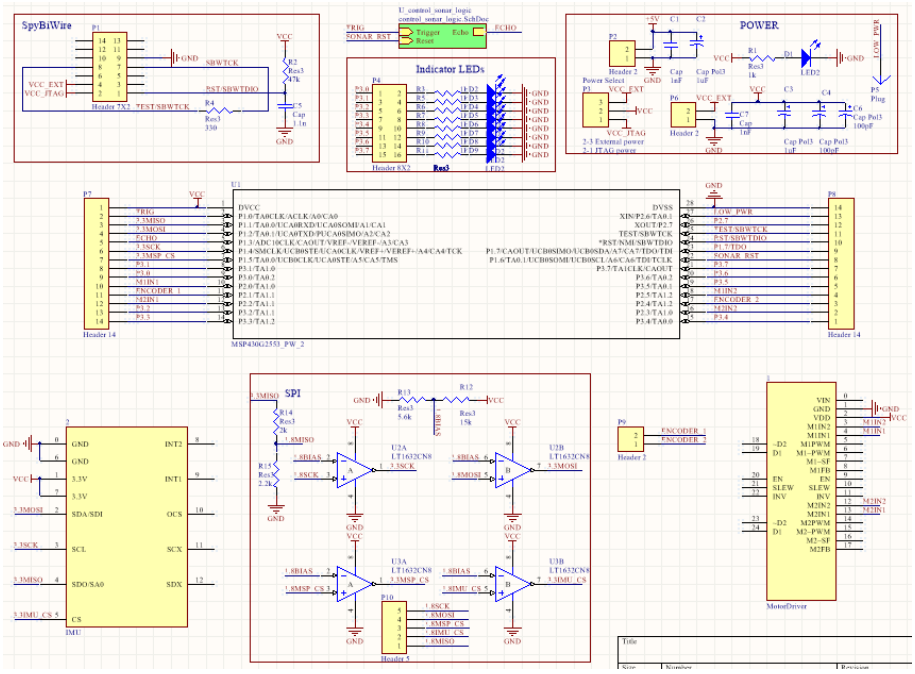


Figure 6: Top level control board

The control board handles the hardware interface of InvestiGator. There are eight major components of the control board hardware; microprocessor, SpyBiWire, sonar array, status LEDs, power, SPI, motor controller and encoders.

Microprocessor

There are a vast multitude of microprocessors to choose from. For InvestiGator an MSP430G2553 was selected for the following reasons:

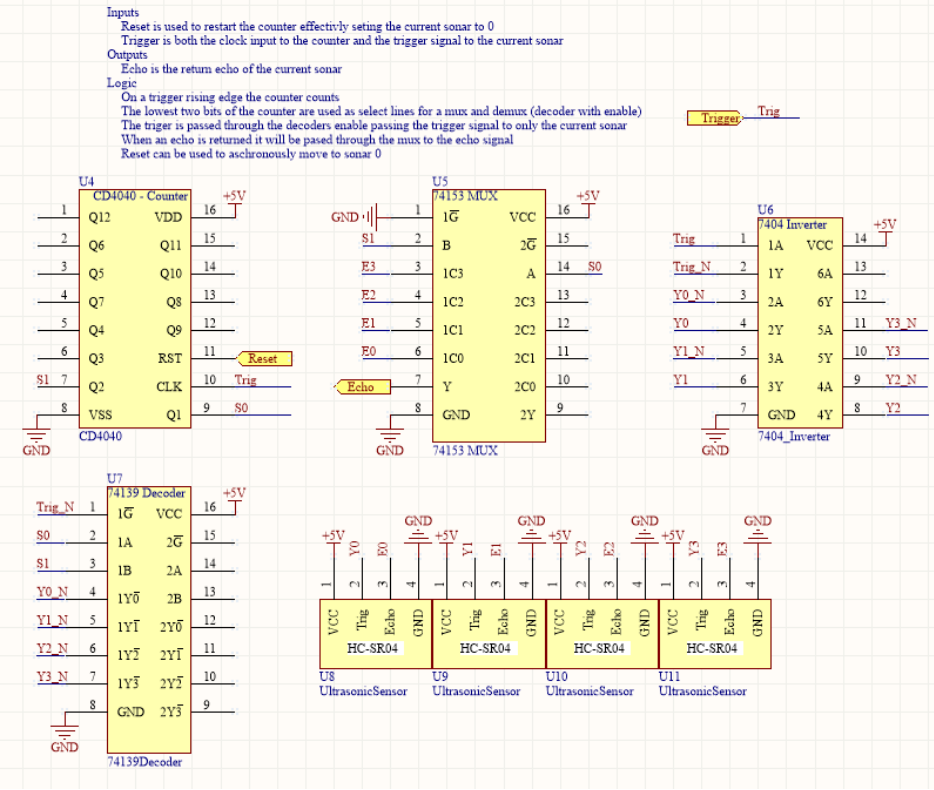
- It is very low power
- It has all the peripherals and functionality necessary to accomplish the following tasks
  - Communicate with the embedded computer
  - Produce PWM signals for the motor controller
  - Display status on status LEDs
  - Grab sonar data
  - Grab encoder data
- It's operating frequency is plenty high enough
- Low complexity of board design

The MSP430G2553 provides an economical solution to all the needs of InvestiGator at the cost of additional firmware development time. The MSP430G2553 is prohibitive in that it only has two timers, has limited GPIO, and several interrupts are shared between peripherals. All of these difficulties are overcome in the firmware.

SpyBiWire

The SpyBiWire interface provides a minimalistic programing interface to the MSP430. SpyBiWire doesn't use any of the GPIO pins which was important since the other components required several GPIO pins. A standard JTAG programmer header was still broken out since the SpyBiWire programmers typically are combined with a JTAG programmer. The extra JTAG pins on the connector were left as no connects.

Sonar Array



Each sonar requires four connections. Two connections are power and therefore do not consume GPIO but the trigger and echo pin both require GPIO pins. With four sonars that is a total of 8 GPIO required. Using an MSP430 greatly limited the amount of GPIO pins available so a method was devised to reduce any number of sonars down to a three pin interface using a counter, de-multiplexer, and multiplexer of size greater than or equal to the number of sonars. A trigger signal from the microprocessor is sent to the counter clock and the de-mux. The echo signals from the sonars are sent to a mux and the mux's

output goes to an echo line on the microprocessor. The counter outputs are sent to the select lines of the de-mux and the mux. When the microprocessor outputs a trigger the rising edge triggers the counter to change state. Any trigger signal that gets through before the change of counter state is less than the minimum pulse with required by the sonars and does not trigger the previous sonar. After the state change the rest of the trigger signal is sent to the sonar selected by the state of the counter. Once the sonar returns its echo signal it will be passed through on the mux for the microprocessor to capture. To obtain a known initial state a third pin on the microprocessor is attached to the reset of the counter. At any time the counter can be reset to return to the initial state.

Status LEDs

Due to the massive reduction in required number of pins because of the sonar array logic gates a total of eight extra GPIO were available. The eight extra GPIO can be used for latter expansion but are currently used to control eight status LEDs. The LED's indicate heart beats, fault conditions, warnings, and other useful user information.

Power

Power can either be supplied from the power regulation board or from the programmer based on the position of a jumper. It is useful to use the programmer power when the entire system doesn't need to be turned on while programing the MSP430. There are several bypass caps for different components across the board such as the logic gates and MSP430. 3.3V is provided for the MSP430 and 5V is supplied for the logic gates. An LED exist to indicate that the MSP430 is powered. A pin is provided to connect the low power signal from the regulation board.

SPI

To communicate with the embedded computer SPI was chosen. The Odroid XU3, the chosen embedded computer, had three options for serial communications, USB, SPI, or I2C. To use USB would have required a conversion circuit such as an FTDI chip. An FTDI chip would have added unnecessary complexity since neither I2C nor SPI require additional hardware, aside from level conversion. So the faster of the two serial protocols, SPI, was selected. The speed is required to achieve acceptable feedback rates for control loops in the embedded computer. The embedded computer uses 1.8 V logic and the MSP430 uses 3.3 V logic. To convert 1.8 V logic to 3.3 V logic high speed op-amps were used as comparators with a 0.9 V bias such that it would rail between 3.3 V and ground. To convert 3.3 V logic to 1.8 V logic a simple resistor divider was used.

Motor controller

A breakout for a dual MC33926 motor driver is provided. The motor driver is set up such that it can be interfaced by two pins per channel. By setting one of the pwm pins to a constant and applying a pwm to the other pin both voltage level and polarity can be controlled.

Encoders

Each encoder has a single pin that the MSP430 attaches to. The MSP430 performs input capture on these channels to measure the rising or falling edges produced by the encoder.

Embedded Computer Selection

To be able to efficiently interface with the Kinect, use Wi-Fi communication, and run advanced algorithms an embedded computer was required. Hard Kernel produces a line of embedded computers called Odroids. Odroids were originally designed to emulate android phones so that developers didn't have to buy a phone to develop on. Hobbyist started using them for other projects making them popular for hobby projects. The community for Odroids has grown significantly in the past years making it easier to work with Odroids. For InvestiGator an Odroid XU3 lite was selected. It can run Ubuntu 14.04 with ROS Indigo. The XU3 specifications can be found at http://www.hardkernel.com/main/products/prdt_info.php?g_code=G141351880955&tab_idx=2. The important features are the Samsung Exynos-5422, USB Hub, IO ports, and DC input. The Odroid XU3 lite provides the horsepower required to run intensive processing such as Kinect data and has the IO required to communicate with a microprocessor.

Firmware

Firmware for this project was defined as any code written on the microprocessor. While the MSP430G2553 is a fast power friendly microprocessor, it is limited in what it can do. Several considerations had to be made in developing the firmware such as GPIO usage, how to best use timers, how to share interrupts, and more.

Communications

Probably the most important job the MSP430 has is to communicate with the embedded computer. To do so a protocol was designed. It was made to mimic how many SPI devices expose their data. The embedded computer is the master and so the MSP430 just waits for a signal from the master. The master initiates a communication by writing a 7 bit address where the 8^{th} bit (the most significant bit)

represents weather the master is trying to read or a write. Then if the master is trying to read the MSP430 serializes data and places it in a transmit buffer. The master continues to send more data which is just ignored by the MSP430 until the transmit buffer is empty and the master has finished the read. If the master instead is trying to write the MSP430 sets a counter to the amount of expected data for the address requested by the master. The master continues to write the data which the MSP430 puts into a receive buffer. Once the transmission is complete the MSP430 de-serializes the data and converts it to the appropriate data types and calls any necessary function calls.

The MSP430 has 6 addresses:

- Sonar - latest data from the sonars
- Encoder speed – latest speed reading from the encoders
- Encoder position – latest position reading from the encoders
- Motors – will set the motor voltage when written to
- Set status – will set the status when written to
- Get status – latest status information

Motor Interface

The motor interface produces pwm signals for the motor controller. A set pwm function is used to set the output voltage and direction of each motor. A single timer is used to generate both pwms by using three output compare channels. One is for framing the pwm essentially setting the period. The other two are for the pulse width length.

Status Indicators

When a communication for the set status register is set the status LEDs will reflect the change. Status LEDs are also used to show fault conditions such as loss of communication with the embedded computer.

Encoders

The encoders use the second timer's input capture channels to stamp rising edges on the encoder pins with the timer time. Two of the three of the timers capture compare registers were used. When the motors don't move there will be no readings. In a scenario where there are no readings the last read value would be returned which gives unacceptable error. To counteract this an overflow interrupt is

added which, if it goes off twice before an update, sets the encoder speed to 0. Position is also tracked by incrementing or decrementing a 32 bit integer based on direction.

Sonar Array

The three pin sonar array is the only system that does not use timer interrupts for its timing. Since it is the final system it can sit in an infinite loop and poll timers. The sonar produces periodic trigger signals and waits for the echo. If an echo is not returned within some amount of time the next trigger is sent and the corresponding sonars distance is set to zero.

Software

All of the code written for user interface and to be run on the Odroid XU-3 lite is considered software. To speed development time Robot Operating System (ROS) was used for its extensive tool set and communication protocols. ROS can be programed in either C++ or Python. Python was used since it is much faster to develop in, however; where speed was very important as in the case of processing data from the Kinect C++ was used. The robot is primarily remote controlled and while a minor degree of autonomy could be added easily it was not implemented for the sake of time.

Robot Operating System

Robot Operating System ROS is an extensive set of tools and conventions designed from the bottom up to encourage collaboration and reuse of code. It has an ever growing user base and new versions are actively being developed. It allows for quick development of robotic systems. ROS was used on InvestiGator for this reason. Drivers for the Kinect have already been made, inter-process communications have been abstracted away, networking is seamless, and more.

In ROS a standalone unit of code is called a Node. Each node can communicate with other nodes over topics. There are multiple modes of communication, the two used for InvestiGator are messages and services. Messages are sent out on a topic for anyone to choose to listen to, there is no guarantee of delivery. A message can be sent out by a publisher and it can be listened to by a subscriber. Services are sent out on a topic to a service provider. The service provider performs an operation and returns results to the request sender.

ROS uses tf, short for transformations, to perform translation and rotation between reference frames. A reference frame is a 3D orthonormal basis, for short they are just called frames. There are several

frames on the UGV, there is one for each sonar, each Kinect camera, the center of the robot, and a frame for the world.

Hardware Interface

To be able to communicate with the MSP430 a Node was developed that would convert, serialize, and send data over SPI. This Node, comms, has three operations continuous read, single read, and single write. When a continuous read is added comms adds it to a list of reads that it will read as fast as possible and publish the data out on a topic. Continuous reads can be removed if necessary. Single reads and single writes get placed in a queue where they wait to be serviced. After one iteration of continuous reads, comms checks if there are any single reads and or writes. If there are single reads and or writes then they will be serviced at that time, otherwise the next iteration of continuous reads begins. This method ensures about 50 Hz update rate of data from the MSP430.

Another node is responsible for dealing with all of the MSP430 specifics such as motor commands and encoder readings. It adds all the continuous reads to the comms node, and initiates single reads and writes when necessary. The MSP430 node also converts raw data from the MSP430 into usable data for other Nodes.

Motion Control

To control the motion of the UGV two wheel differential steer kinematics are used. A lead controller is used to control the motors speed. Waypoint control comes from the University of Michigan's Intelligent Robotics Lab [1].

Inverse Kinematics

As the robot is a differential steer robot constrained to a 2D plane the Kinematic equations are relatively simple. The equations of motion were obtained from Computational Principles of Mobile Robotics [2].

$$\omega_r = \left(\frac{1}{r_{wheel}}\right)\frac{V + wheelbase * \omega}{2}$$

$$\omega_l = \left(\frac{1}{r_{wheel}}\right)\frac{V - wheelbase * \omega}{2}$$

Where ω_x is the corresponding wheels angular velocity, r_{wheel} is the radius of the wheels, wheelbase is the distance the wheels are apart, V is the desired forward velocity and ω is the desired angular velocity about the instantaneous center of rotation.

A user supplies a twist, a linear and angular velocity vector, which the inverse kinematics node converts into motor commands. V comes from the x component of the twist and ω comes from the z component of the angular velocity. The motor commands are then sent to the wheel speed controller.

Motor Control

To control the motors a model of the motors was designed. Measuring the impulse response, a transfer function for the motor was obtained. From the data a steady state gain and time constant was derived. Modeling the motor as an active low pass filter, a lead compensator was designed in Matlab to improve the phase margin of the closed loop system and ensure a quick time domain response. See Appendix B for the Matlab code that looks for an optimal controller.

Waypoint Control

Waypoint control takes in a desired Pose, position and orientation, in the world frame and employs control laws from reference [1]. The UGV has successfully implemented the design in [1] and can be shown to have similar results as the paper. See Appendix B for Matlab simulations.

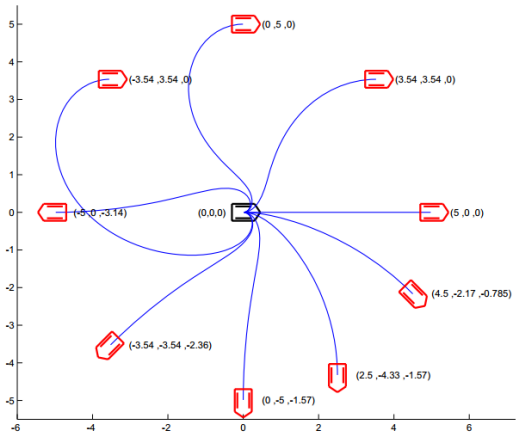


Fig. 5. Example trajectories of a unicycle with the proposed control law, with constants $k_1 = 1$ and $k_2 = 10$. Note that the vehicle converge to the slow manifold more quickly with higher value of k_2.

Figure 7: Smooth paths as illustrated by [1]

Percept Generators

Percepts are what the UGV senses or perceives. The Kinect is the major contributor to percepts but the encoders are also very important.

Forward Kinematics

Forward kinematics uses equations from Computational Principles of Mobile Robotics [2] to convert encoder readings into state estimation.

$$V = \frac{V_R + V_L}{2} \qquad \omega = \frac{(V_R - V_L)}{wheelbase}$$

$$R = \frac{wheelbase * V}{(V_R - V_L)}$$

$$d\theta = dt * \omega \qquad dx = R * \sin d\theta \qquad dy = R * (1 - \cos d\theta)$$

The differential quantities are added to the current values to update to the new state of the robot. A Simple two point moving average is applied to R and ω instead of using instantaneous values.

Kinect

Due to the limited computational power of the Odroid XU3 lite and the network lag the Kinect point clouds are significantly filtered. Point clouds are arrays of data containing positional data and possibly color data. The first step in the point cloud filter is to remove all points that the UGV does not care about. All values corresponding to points significantly above or below the UGV are remove entirely. Next a voxel, volume pixel, filter is introduced. The voxel filter intelligently averages points in the point cloud such that only one point exist per voxel. The averaging decimates the data by a factor of 10. This data can then be used for algorithms and viewing by the user over the network. In addition to the point cloud the Kinect also produces depth, ir, and rgb images. These are made available to the user over the network.

User Interface

The user interface has two main components the natural human interface which is implemented with a Leap motion and the visualizer, implemented with ROS's RVIZ software.

RVIZ

RVIZ is one of the many tools provided with ROS. RVIZ allows a user to visualize ROS messages. In figure 8 the rgb image from the Kinect is shown on the upper half of the laptop and the top down view with point cloud is displayed on the bottom half. The user is placing a waypoint by making a pinching motion. RVIZ can be used to visualize several other data types as well.

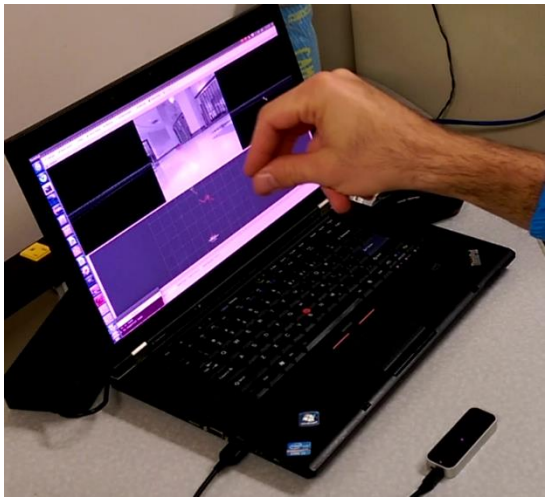


Figure 8: RVIZ displaying UGV in world

Leap Motion

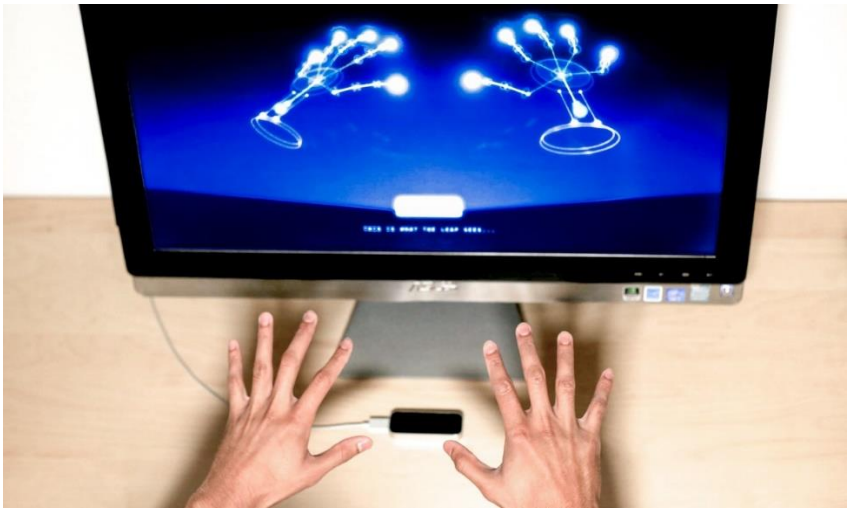


Figure 9: Leap Motion visualizer

A Leap Motion is used to provide a natural human interface. The Leap Motion is similar to the Kinect in that it measures depth with infrared light but it is designed for a significantly smaller scale. The Leap Motion is designed for use with ones hands as can be seen in figure 9. Two modes of operation have been implemented with the Leap Motion, direct remote control, and waypoint navigation.

In direct remote control mode the angle of the users hand directly controls the motion of the UGV. Pitching the hand forward moves the UGV forward and pitching backwards moves the UGV backwards. Rolling the hand left or right causes the robot to turn counter clockwise or clockwise respectively.

To switch back and forth between waypoint control and direct remote control the user places both hands in the field of view of the leap motion and makes a large circle with one hand.

In waypoint mode a user grabs a pose, position and orientation, and places it in the world as visualized through RVIZ. The UGV will then use the control laws developed in [1] to move itself to the goal position. The procedure for placing a waypoint is as follows:

1. Pinch with one hand in view of the Leap Motion to grab a waypoint
2. While maintaining pinch move the waypoint to desired location
3. While maintaining pinch rotate hand left and right to change the orientation of the pose
4. Let go of pinch
5. Wait half a second
6. Swipe down to confirm the waypoint

References

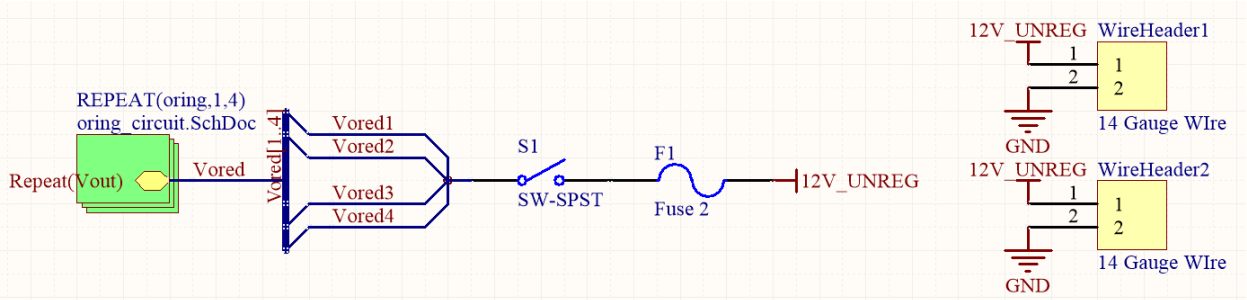
[1] J. J. Park and B. Kuipers, "A Smooth Control Law for Graceful Motion of Differential Wheeled Mobile Robots in 2D Environment," in *IEEE Int. Conf. on Robotics and Automation*, Shanghai, China, 2011.

[2] G. Dudek and M. Jenkin, Computational Principles of Mobile Robotics, Toronto: Cambridge University Press, 2010.

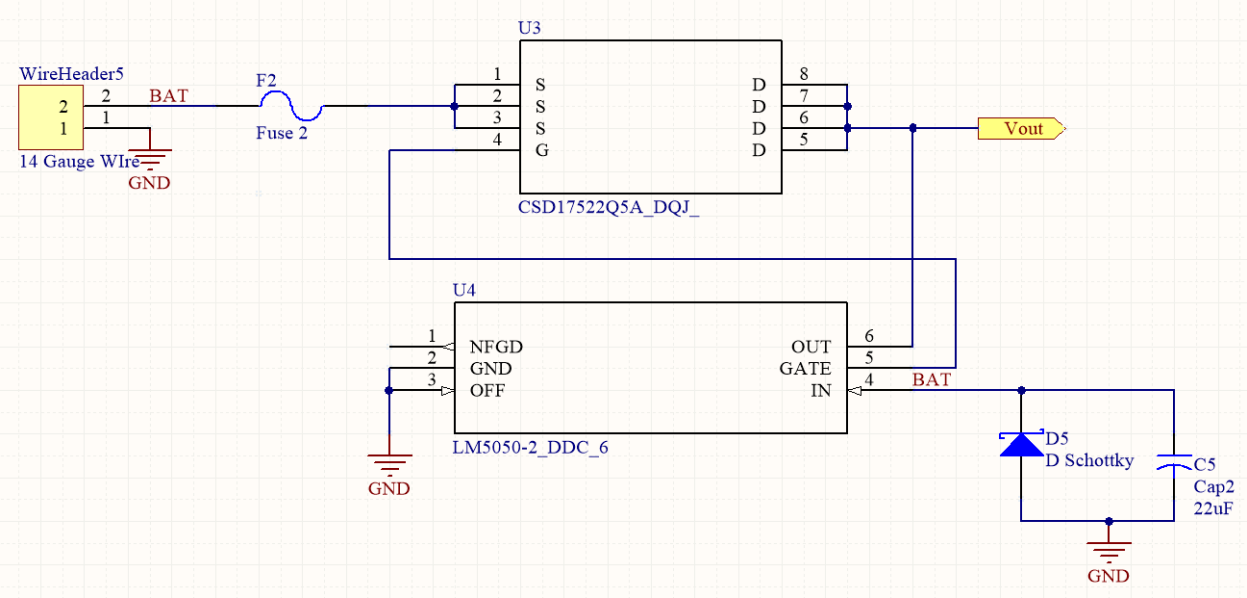
To view the code, and Altium files visit fnivek's github, https://github.com/fnivek.

Appendix A

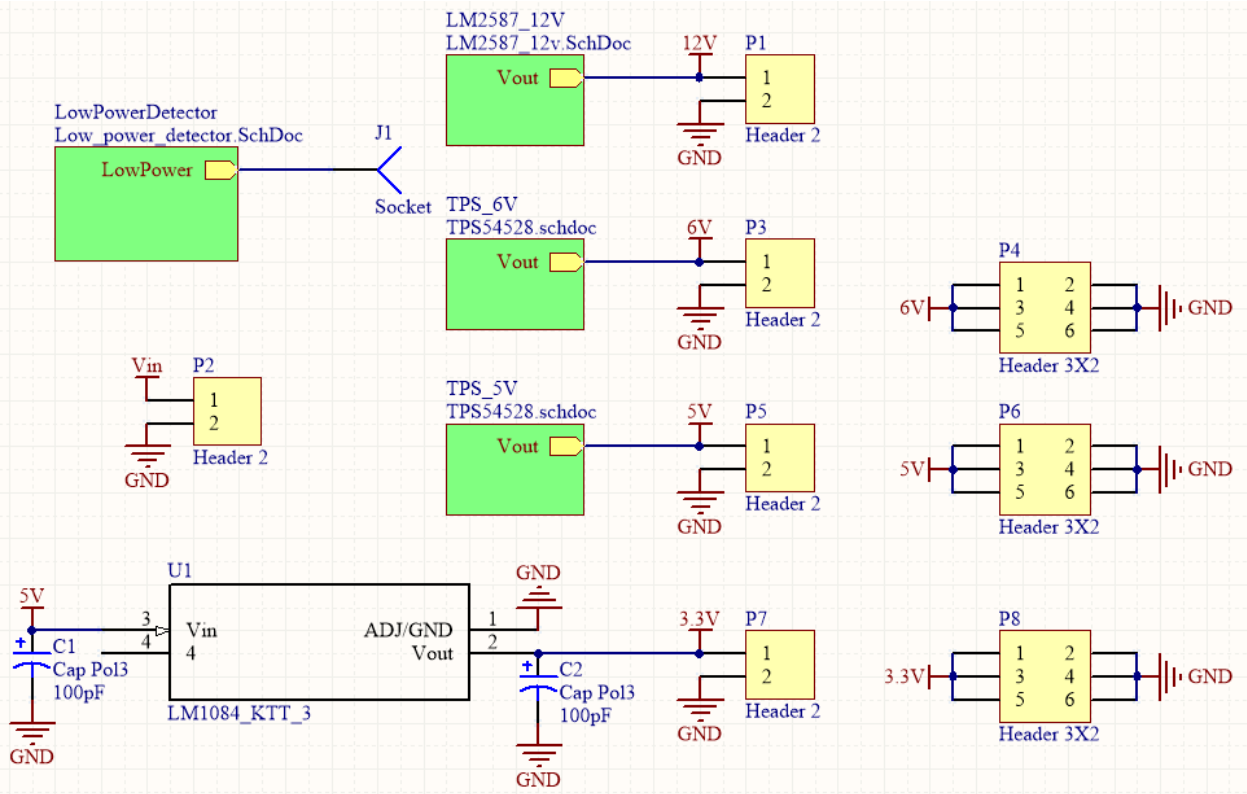
Power oring top level circuit:



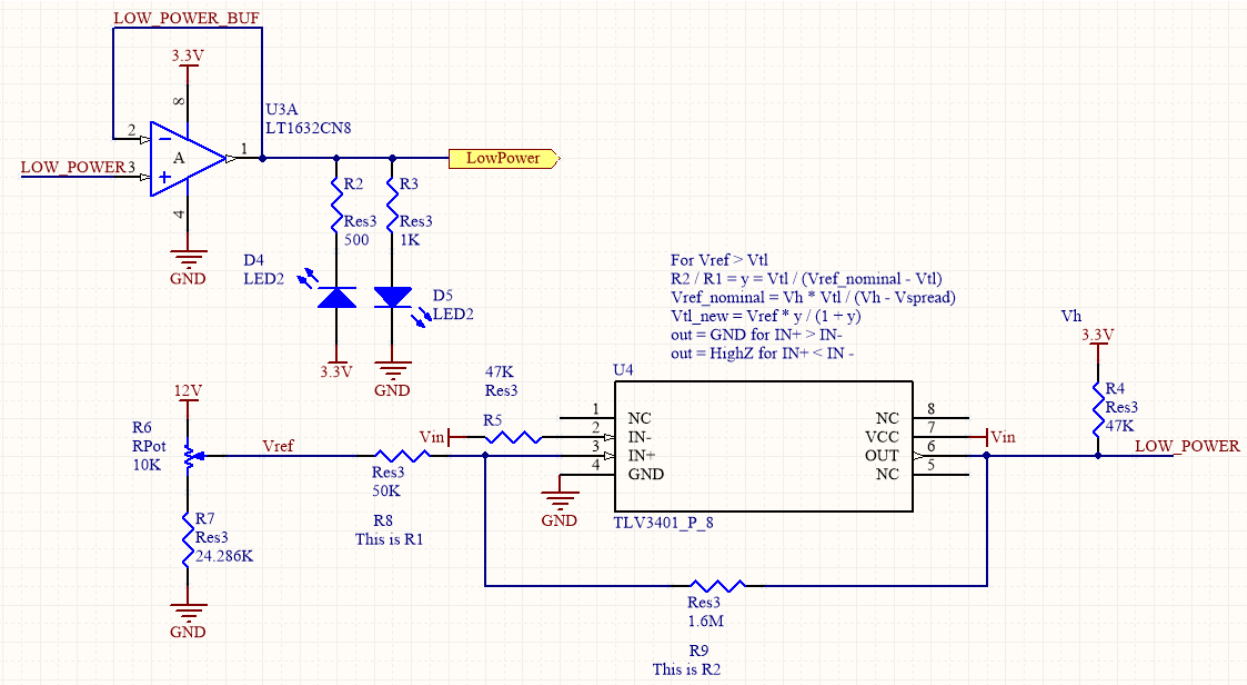
Power oring internal repeated circuit



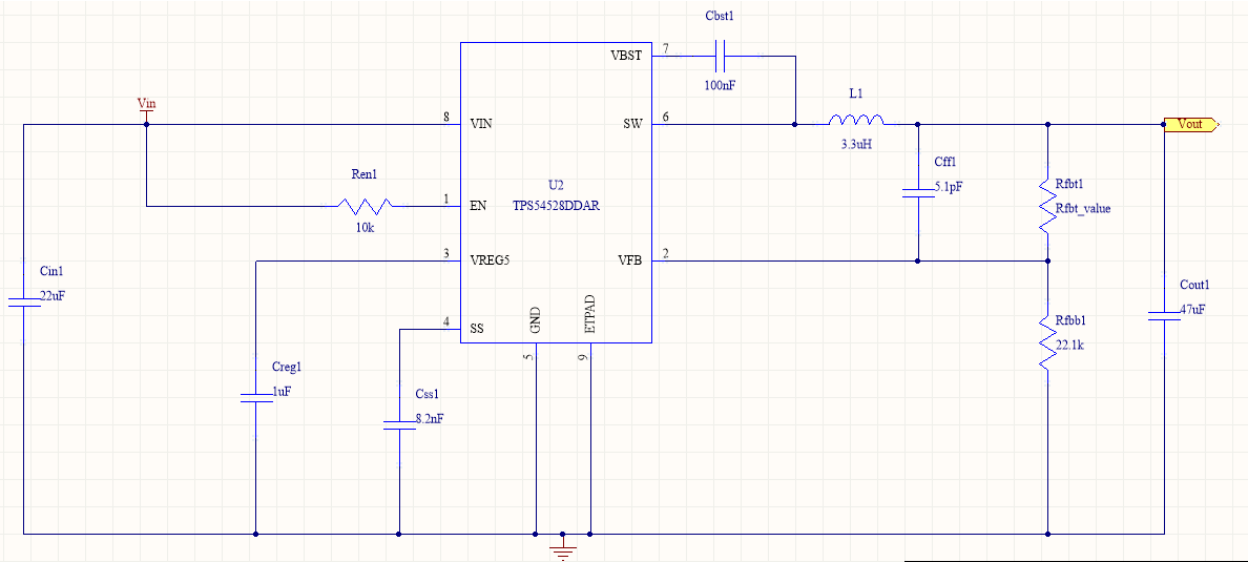
Power regulation top circuit



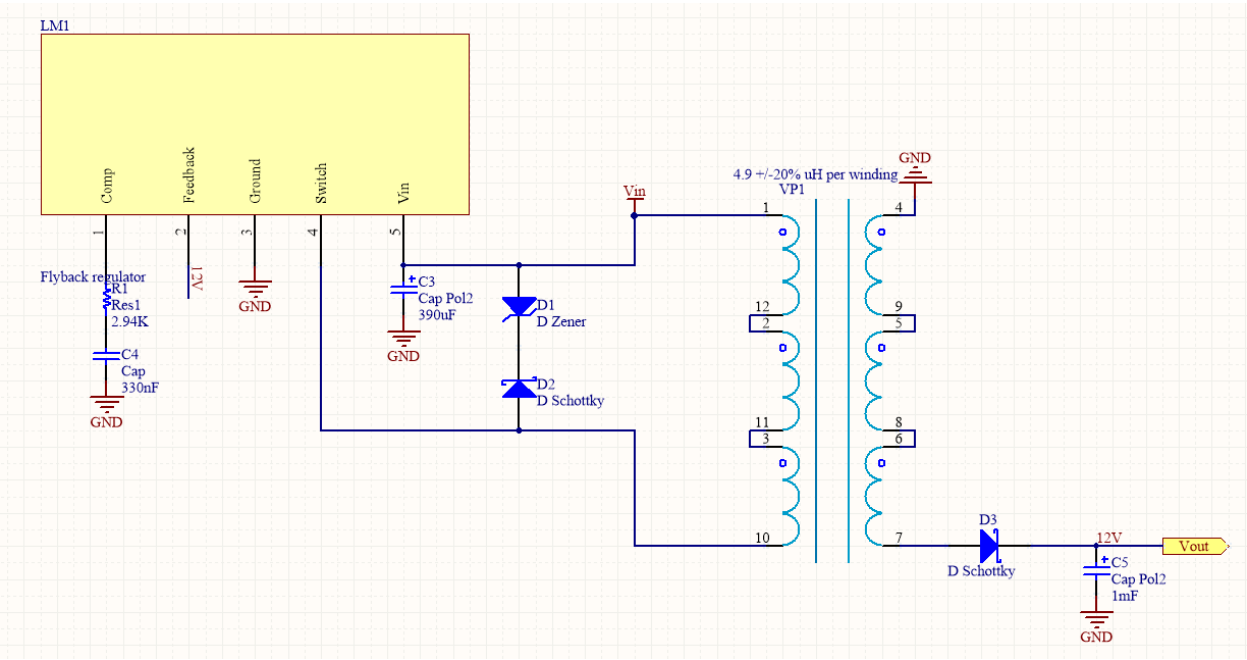
Power regulation low power indicator



Power regulation 5 and 6 V repeated circuit



Power regulation 12 V



Appendix B

Finds best lead controller for a given model of a motor

```
function [z p gain] = bestcontroller( zstart,zstop,zres,pstart,pstop,pres,kstart,kstop, kres,trans)

polesnzeros =
CombVec(complexlist(zstart,zstop,zres),complexlist(pstart,pstop,pres),complexlist(kstart,kstop,kres));
s=tf('s');
best=100000000;
%[Gm,Pm,Wgm,Wpm] = margin(sys);
%Finish function by checking for stability before setting new best C
for k=1:length(polesnzeros)
   %THE CONTROLLER ADDS AN INTEGRATOR
   %THIS WAS MADE FOR SPEED CONTROL FOR A DC MOTOR
    C = (polesnzeros(3,k)*(s+ polesnzeros(1,k)))/(s^2+s*polesnzeros(2,k));
    T = C*trans/(1+C*trans);
    [yy,tt,xx] = step(T);
    S = stepinfo(yy,tt);
    if((S.PeakTime < 0.1) && (S.PeakTime > 0) && (yy(length(yy),1) > 0.95))
    prod = S.Overshoot * S.SettlingTime;
   % B = bandwidth(T*((1/dcgain(T))));
    if((prod<best)&& (prod > 0))
       gain=polesnzeros(3,k);
       z = polesnzeros(1,k);
       p = polesnzeros(2,k);
       best = prod;
    end
    end
end
end
```

Generates a Waypoint command sequence

```
state = [30;0;0]; % This is where you enter in the distance, final heading, and angle between origin and
final destination (r,theta,delta)
%The robot starts at rest at the "origin"
%Origin is in quotes because r theta and delta are w.r.t. the robot
dt = 0.01;% I simulated at 100Hz changing this won't do much in simulation; there's no feedback or real
dynamics
rdot = 0;
thetadot = 0;   %Robot initially at rest
deltadot = 0;
kv = 0.5;
k2 = 10;       %Gains from the umich paper but I selected kv
k1 =1;
```

```
v = 0;         %Initial velocity
omega = 0;     %Initial rot. rate
vrecord = 0;    %This is how I saved the velocity and omega as they evolved over time
omegarecord = 0;%Don't judge me ;)
% r = state(1);
% theta = state(2);
% delta = state(3);
for i = 0:10001
   rdot = -v*cos(state(3));
   thetadot = (v/state(1))*sin(state(3));            %These first three lines are the kinematic equations from
the paper
   deltadot = (v/state(1))*sin(state(3)) + omega;
   state(1) = state(1) + dt*rdot;
   state(2) = state(2) + dt*thetadot;               %Update the states
   state(3) = state(3) + dt*deltadot;
   v = kv * state(1);                              %Velocity is a linear function of distance
   omega = -kv*(k2*(state(3) - atan(-k1*state(2))) + (1 + (k1/(1
+(k1*state(2))^2))*sin(state(3))));%Omega calculation from the paper
   %The calculated v and omega would be fed to the inverse kinematics engine and then to the speed
controller

   vrecord = [vrecord , v];
   omegarecord = [omegarecord , omega];    %Keeping track of the v and omega over time

end
```

Simulates the UGV following waypoint path

```
%%%%%  Run this simulation after waypt.m  %%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%The simulation takes the v and omega commands and plots the path of the
%robot if those commands were instantly effective
x = 0;
y = 0;
dt = 0.01;     %This should be the same as in waypt.m
theta = 0;    %Assume the robot is at the origin
xrecord = [];
yrecord = [];
thetarecord = [];
for i=1:length(vrecord)
   theta = theta + omegarecord(i)*dt;  %Integrate omega to get theta
   x = x + vrecord(i)*cos(theta)*dt;
   y = y + vrecord(i)*sin(theta)*dt;   %Update the robot position
   xrecord = [xrecord , x];
   yrecord = [yrecord , y];          %Store the position and angle
   thetarecord = [thetarecord, theta];
```

```
end

subplot(4,1,1)
plot(xrecord, yrecord, '.')

subplot(4,1,2)
plot(0:10002, vrecord)

subplot(4,1,3)
plot(0:10002, omegarecord * 180 / pi)

subplot(4,1,4)
plot(0:10002, thetarecord * 180 / pi)
```