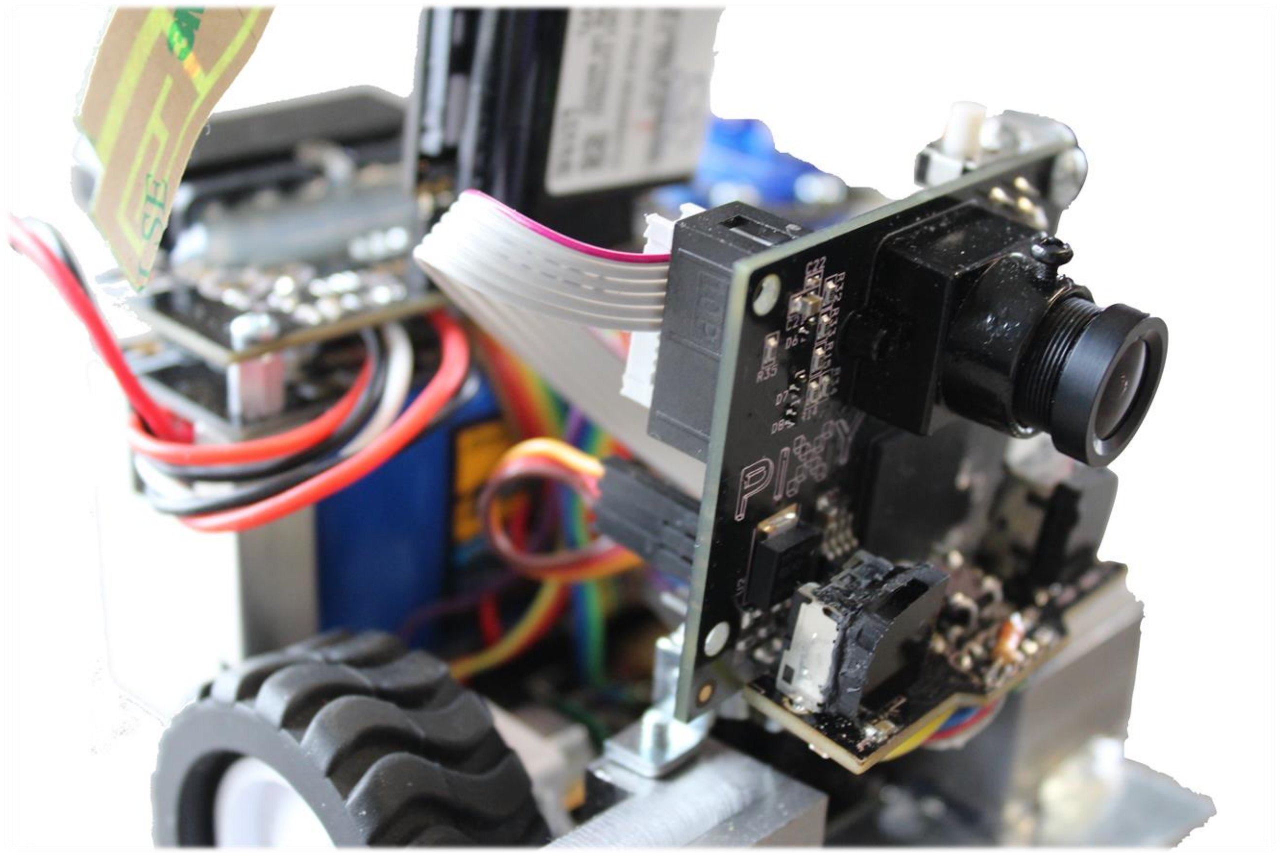
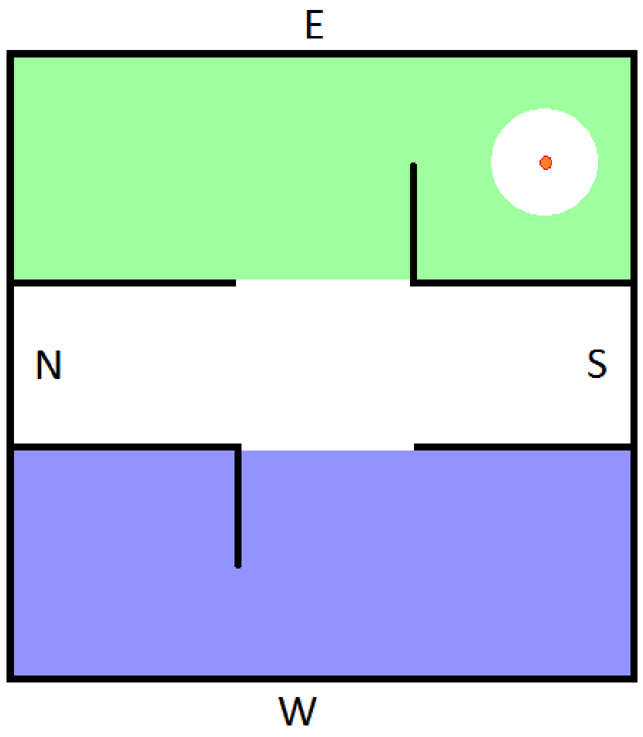
Microprocessors II Final Project Report
Ladder 42

By: Broderick Carlin and Tyler Holmes



Abstract

The goal of this project was to, starting on either the north or the south side, be activated by one of two distinct tones. Then, based on which tone was played, the robot would navigate to either the east (highlighted green for illustration purposes) or west room (in blue) where the candle will have been placed. The robot then looks for the candle, drives to it until it detects the white circle under the candle. The robot finally then texts us that it has found the robot and how long it took. We then put out the candle ourselves by blowing it out.



The maze as laid out by the project guidelines. The candle can be placed in any location in the east or west rooms.

In order to accomplish this goal, we used motors with encoder feedback, a color and luminosity sensor pointed downwards, two wiimote IR camera's, a FONA GSM module, a microphone with amplifier, and a PIXY camera for object avoidance. Unfortunately, due to time constraints and other issues, the PIXY camera, along with the compasses and mechanical whiskers that were originally included, were not implemented for reasons discussed later.

Scope of Project

The objective of this project was to design and produce a fully autonomous robot that is able to distinguish audio tones, navigate a maze, locate a candle, and extinguish it. The robot must be based around a microprocessor from the PIC18F family and be able to fit entirely within a 6”x6”x6” box. It originally had to avoid obstacles as well, but that was removed as a requirement before the competition.

In order to accomplish this task, we used motors with encoder feedback for absolute position (as long as our wheels did not slip), a color and luminosity sensor pointed downwards, two Wii mote IR cameras, a FONA GSM module, a microphone with amplifier, and a PIXY camera for object avoidance.

Motors and Encoders

In the end, the high precision and high power of the motors as well as the feedback given by the encoders were the most important part of the control algorithm. Once the obstacles were removed as a consideration, we were able to hardcode distances and angles into the robots drive algorithm to get to specific parts of the rooms. Once at a good vantage point, we could go into a searching algorithm, looking for IR sources on the Wii cameras.

The motor drives and encoders formed a simple control loop that allowed accurate control over the rotation of the each motor individually, while also having the rotation of the two motors being interdependent on the others angle of rotation. There were three control algorithms that were responsible for controlling the movement of the wheels at separate times. The most simple of these was the algorithm that was responsible for driving the robot in a single straight line. This algorithm took in a single value representative of the linear distance of travel that was desired. By knowing the circumference of the wheels as well as the number of encoder ‘ticks’ per revolution, a simple ratio can be formed to find the number of encoder ‘ticks’ needed to reach our target distance. Simply monitoring and counting encoder ‘ticks’ is not enough however, as there is a high chance that our two motors will have slightly difference characteristics which would cause our robot to slowly turn to one side. To combat this a difference between the angle of rotation in the two motors was taken and this value was used as a dynamic proportional gain that was added to the slower side, and taken away from the faster side. By adding in this interdependence it is possible to create a simple algorithm to drive in a straight line without the worry for slowly drifting to one side or the other. Driving straight and checking if we have reached our target goal proves to also be an inaccurate way to achieve a repeatable distance traveled measurement. To increase accuracy, a second proportional gain was added to each side that was directly proportional to the distance left to travel. This means that as the robot approached its target position it would begin to slow down. Without this feature added problem such as overshooting the target or even flipping over were encountered. With that additional of this proportional gain we were able to achieve repeatable results that accurately moved the robot to within a couple mm of its target position.

The second feedback loop used to control the motors was that which was used for turning the robot. Similar to driving, an angel was sent to the algorithm and based upon the known circumference of the wheels, the number of encoder 'ticks' per revolution, and the robots wheelbase, a relationship was able to be derived to relate the angle of rotation with the number of encoder 'ticks'. Using this relationship an algorithm was developed using the same techniques for interdependence and error compensation to allow for accurate and repeatable rotation.

The final feedback loop used to control the motors does not take advantage of the encoders but rather uses cameras as its feedback. The motors are driven straight and by checking the location of a known key point in the camera frame, the rotation of the two motors is compensated in an attempt to keep the key point in the same location of the frame continuously. This method of feedback is useful for when you want to drive towards a known object, and in our scenario this happened to be the candle. By not having to worry about obstacles we are able to lock onto the candle using a single camera and then switch into this mode of control, effectively driving straight towards the camera repeatedly.

GSM cellular connectivity

Our robot takes advantage of T-Mobile's 2G cellular network to communicate with our cell phones through texting to inform us when it locates a candle, as well as other important information, such as how long it took the robot to find the candle and the status of our sensors. The specific module we used was the FONA module sold by Adafruit industries. It used a SIM800L cell module chip that communicated of UART. By sending it a string of specific commands and waiting for certain responses, texting was implemented with relative ease. The major issue we encountered was the speed of the GSM network, which is sometimes nearly instantaneous, and sometimes takes minutes.

IR cameras

The wii IR cameras have been thoroughly reverse engineered by the Wii homebrew community, making them very easy to interface with. They communicate over I2C and have a very specific initialization sequence involving commands, delays, and settings. Once fully setup, it is very easy to get (x,y) coordinates as well as the size of the four IR blobs each camera sees. After retrieving that data, we are able to look at both sets of data and determine if we see a candle. We can know whether or not both cameras see a candle by looking at the y coordinates of each IR blob. If both cameras detect a blob at the same y height, we know that it isn't a spurious error, but rather a physical IR source, most likely the candle.

Once we had detected a candle, we then attempted to triangulate its position using trigonometry. This relied on knowing the field of view of the Wii cameras as well as having a stable IR source to reference. We would sometimes get very accurate measurements, but other times get crazy numbers, such as 200 cm behind us. This was due to quantization error. Sometimes, the triangles formed by the candle would become quantized into impossible triangles of greater than 180 degrees, forcing the math to compute that the candle was behind us rather than in front of us. This is mostly due to how close together the sensors were to each other. Instead of triangulating the location, we instead looked until the candle was in the center of our field of view and just drove straight at the candle. This was made possible by the elimination of obstacles.

Some of the issues we ran into with this approach was erroneous readings from the reflection of the candle and trouble finding the candle due to the very narrow field of view of the Wii cameras. Sometimes, especially before we took off the plastic covering the walls, the bot would lock on to the reflection of the candle rather than the candle itself, causing the robot to careen into the wall (because it never ran into the white circle, so it never stopped). This was mainly remedied by the removal of the plastic coating. Further improvements could have included having a size threshold. If the IR source was smaller than a certain threshold, it would not register as a successful detection of the candle. This was not implemented due to time constraints. The other major issue was the narrow field of view. The Wii cameras only have a field of view of 33 degrees horizontally and 23 degrees vertically, which when you're looking at both cameras to have a match makes for a pretty narrow detection area. This could have been remedied by a constant rotational speed while continually polling the Wii cameras or by having a smaller turning angle.

Audio interface

This is the task of designing the circuit to amplify and filter the audio from an electret mic so it can be fed into an analog input on the PIC processor. This task also includes interpreting the incoming audio using an FHT algorithm to determine the dominant frequency so as to detect the fire alarms. The hardware used for this was inspired by an electret mic breakout board offered by Sparkfun, while the FHT algorithm was a derivation of the original software released in the late 80's. The FHT algorithm operated at an update rate of ~150Hz and had a resolution of 300Hz. This resolution, although very poor, was more than enough to suite our needs.

Color Sensor

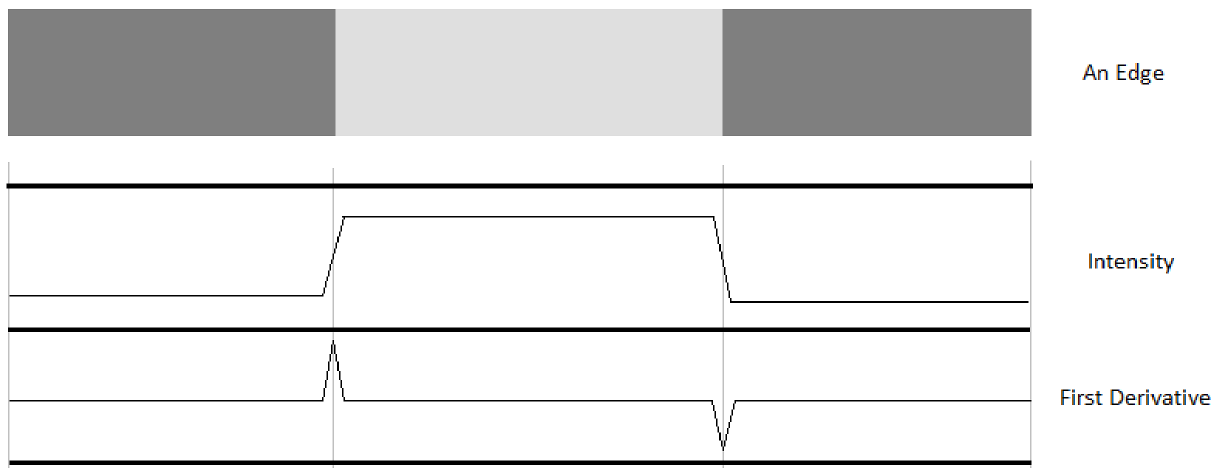
The color sensor was very straight forward and easy to work with. The datasheet was crystal clear in its operation and accessing of its data. This sensor also ran on I2C communication protocol, which is quite a robust and reliable protocol. There may be some overhead in the starting, stopping, addressing slaves, and acknowledges, but once it starts working it is reliable and simple to understand.

The color sensor is able to detect intensity in the red, blue, green, and clear color spectrums. For our use, we only looked at the clear spectrum, but were able to get the other three working reliably as well. Once the sensor was set up properly to take a measurement, detection was as simple as sending a few commands and reading back some bytes from the sensor. Black carpet returned a value of around 200, whereas white returned a value of about 2,000. The sensor even offered different multiplication factors, so it was a very flexible and easy sensor to work with.

Full color camera

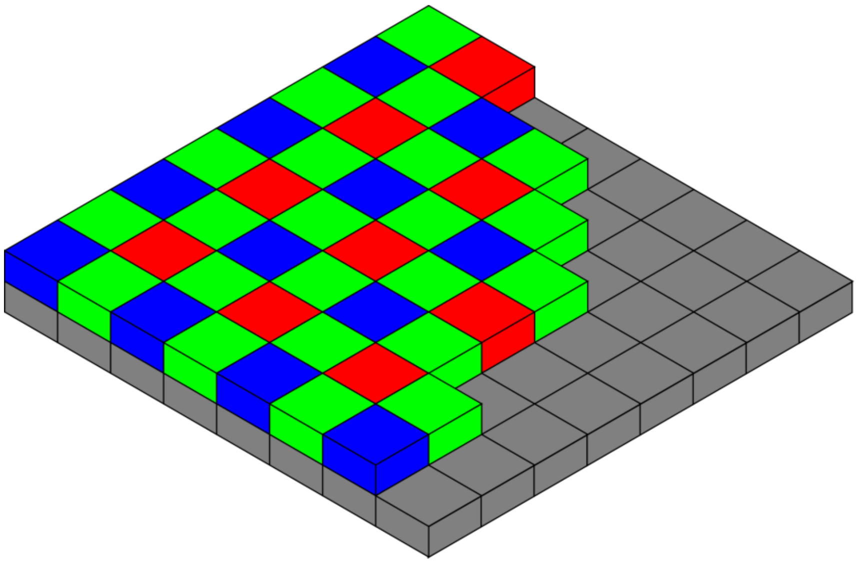
The Pixy CMUcam5 took up the majority of Tyler's time, which is unfortunate considering the Pixy was not officially implemented in the final design. Due to the potential for erroneous readings of objects, such as lightly colored particles on a black surface and the fact that object avoidance was no longer a factor, it was taken out the day of the competition. Despite this, I'm still going to cover all of the technical details of what I did, mainly so I feel better about the month I spent working on it.

The camera's main object avoidance technique was edge detection. This was done through the implementation of a Sobel filter and thresholding. This approximates the derivative of the pixel's intensity in both the x and y direction. Once we have this, we can approximate a gradient by adding the absolute value of the x and y components together. A true gradient is the square root of the sum of the squares, but the simple addition of the absolute value is often used as a very accurate approximation.



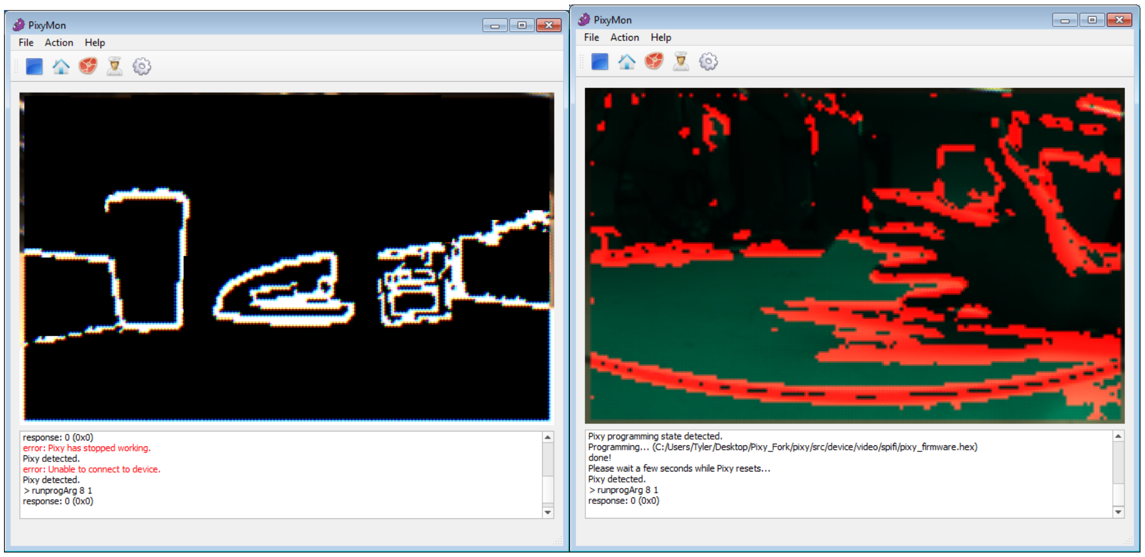
How simple edge detection works

In order to calculate the gradient, we first must convert RGB Bayer image data to intensity. A Bayer filter image sensor has a grid of either red, green, or blue sensors at each pixel location. This leaves two colors missing at each pixel location.



http://commons.wikimedia.org/wiki/File%3ABayer_pattern_on_sensor.svg
Bayer filter technology. Each pixel only has either red, green, or blue data.
25% red, 25% blue, and 50% green.

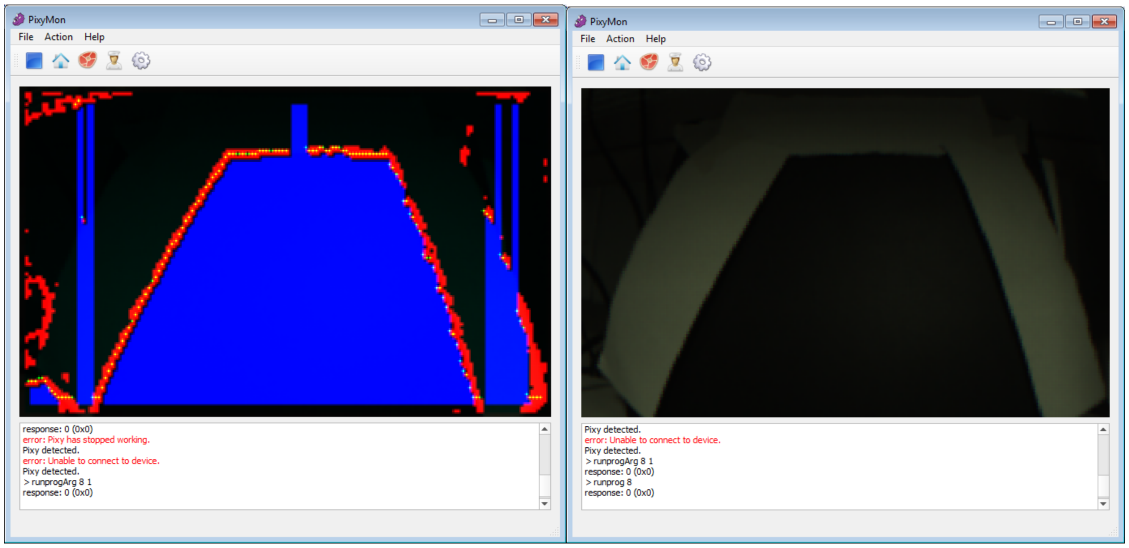
Instead of interpolating the missing data, I instead grouped four pixels together and calculated the effective intensity of the group. The benefits of this method is speed and accuracy, which are important, but the downsides are decreasing the resolution by a factor of four. The effective resolution after this process is 160 by 100 pixels, which is not very high resolution. This turned out to be a significant limitation. The result of this process turned out looked very promising at first, detecting edges quite well.



```
response: 0 (0x0)
error: Pixy has stopped working.
Pixy detected.
error: Unable to connect to device.
Pixy detected.
> runprogArg 8 1
response: 0 (0x0)
```

```
Pixy programming state detected.
Programming... (C:/Users/Tyler/Desktop/Pixy_Fork/pixy/src/device/video/spifi/pixy_firmware.hex)
done!
Please wait a few seconds while Pixy resets...
Pixy detected.
> runprogArg 8 1
response: 0 (0x0)
```

left: course testing with a coffee cup, stapler, box, and two walls.
right: noisier environment with a hand



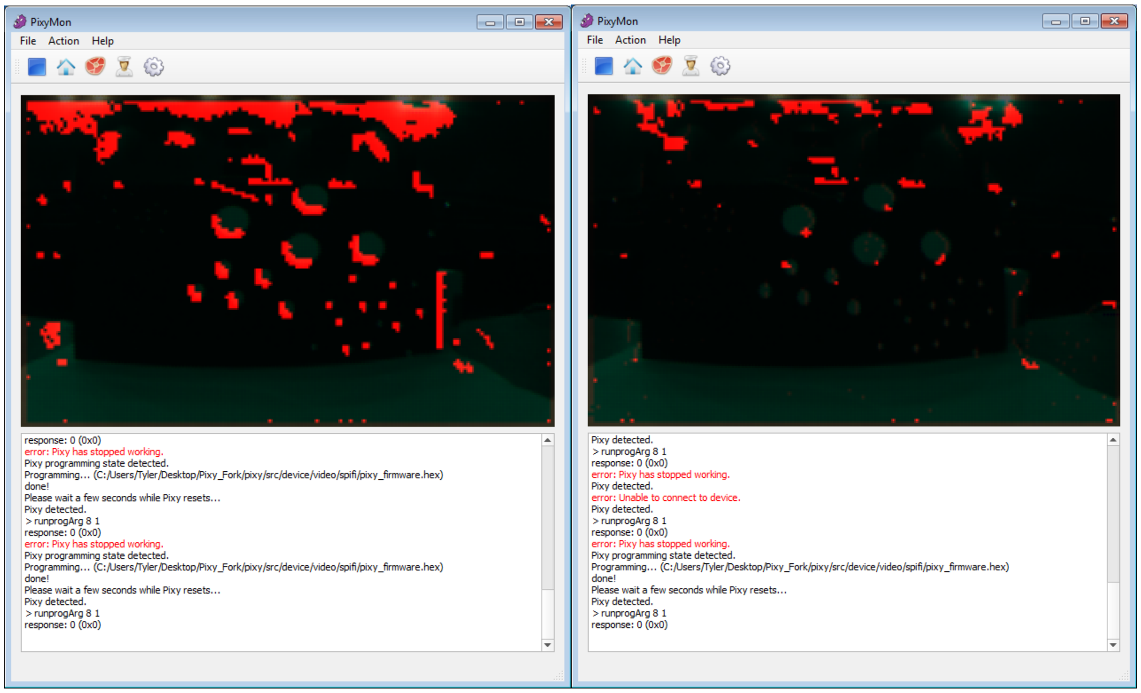
```
response: 0 (0x0)
error: Pixy has stopped working.
Pixy detected.
error: Unable to connect to device.
Pixy detected.
> runprogArg 8 1
response: 0 (0x0)
```

```
Pixy detected.
error: Unable to connect to device.
Pixy detected.
> runprogArg 8 1
response: 0 (0x0)
> runprog 8
response: 0 (0x0)
```

left: edge detection with drivable floor flood fill.
right: raw image of the floor detection

The major issues that held us back was erroneous detection, gaps in edges, and tuning the camera's exposure and brightness settings to get quality edges. As I decreased the threshold for an edge, erroneous detection increased, and vice versa. To overcome these issues, things like edge extension, which does an angle calculation on the edge and extends the edge across the gap where the threshold is too low, and smart filtering of noise is needed. Unfortunately I did not have enough time to implement these improvements.

We also had to deal with white specs all over the black carpet. I was able to overcome this issue by doing smart blob filtering, specifically looking for small specs and removing them from the edge array. When entering the spec, it goes from a black to white intensity, which the derivative of which is positive. A length counter is then initiated. Once we go back from white to black, which the derivative of which is negative, if the length is in the range of blobs, the previous length of edges is deleted.



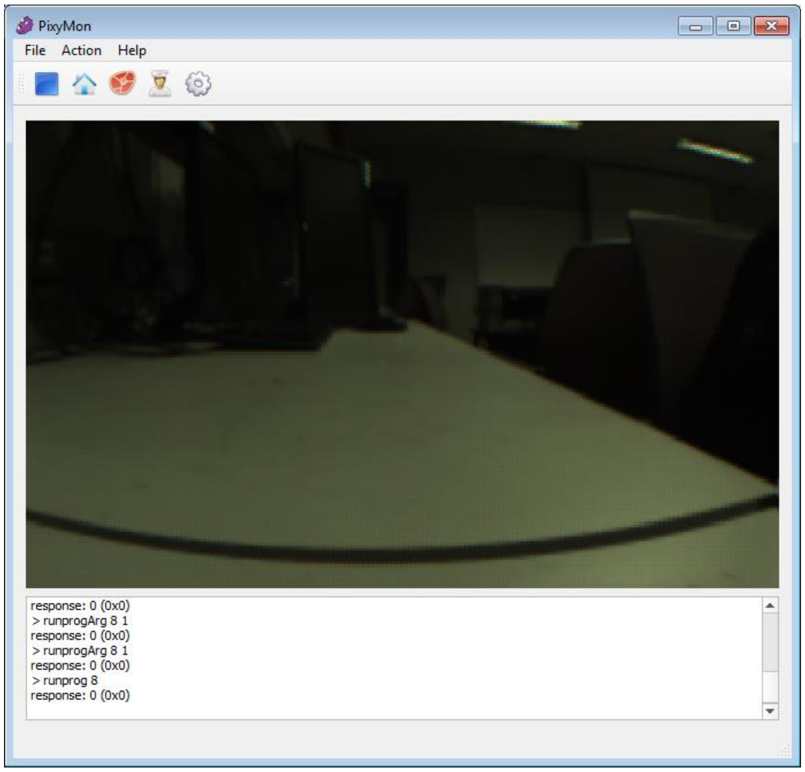
```
response: 0 (0x0)
error: Pixy has stopped working.
Pixy programming state detected.
Programming... (C:/Users/Tyler/Desktop/Pixy_Fork/pixy/src/device/video/spifi/pixy_firmware.hex)
done!
Please wait a few seconds while Pixy resets...
Pixy detected.
> runprogArg 8 1
response: 0 (0x0)
error: Pixy has stopped working.
Pixy programming state detected.
Programming... (C:/Users/Tyler/Desktop/Pixy_Fork/pixy/src/device/video/spifi/pixy_firmware.hex)
done!
Please wait a few seconds while Pixy resets...
Pixy detected.
> runprogArg 8 1
response: 0 (0x0)
```

```
Pixy detected.
> runprogArg 8 1
response: 0 (0x0)
error: Pixy has stopped working.
Pixy detected.
error: Unable to connect to device.
Pixy detected.
> runprogArg 8 1
response: 0 (0x0)
error: Pixy has stopped working.
Pixy programming state detected.
Programming... (C:/Users/Tyler/Desktop/Pixy_Fork/pixy/src/device/video/spifi/pixy_firmware.hex)
done!
Please wait a few seconds while Pixy resets...
Pixy detected.
> runprogArg 8 1
response: 0 (0x0)
```

left: no blob compensation pointed at a test sheet with many white blobs of different sizes
right: same shot with blob compensation in the x direction. As you can see,
most of the blobs are removed

We were able to implement object avoidance reliably by taking the front edge found by the floor detection algorithm, averaging adjacent points down by a factor of four, and stopping if any of the edges got within 20 pixels of the bottom of the screen. If we had more time, we would've been able to develop an algorithm to drive around an encountered obstacle, but we ran out of time.

Our original plan was to extrapolate a top-down floor map based on the detected floor. Unfortunately, while this was mathematically sound, the reality of quantization and a shallow camera angle caused us to be unable to implement the algorithm successfully. The camera was too close to the ground and the camera was too low of resolution. It also has a very distorting fisheye lens that throws off all of the calculations near the edge of the frame.



```
response: 0 (0x0)
> runprogArg 8 1
response: 0 (0x0)
> runprogArg 8 1
response: 0 (0x0)
> runprog 8
response: 0 (0x0)
```

That's supposed to be a straight line, but the fisheye lens distorts the image

Due to all these factors, the use of the Pixy image sensor was not fully implemented, however, object avoidance was successfully tested and with further work could be viable. Reliable and accurate distance extrapolation on the other hand is not viable at the height we had the Pixy or the lens we had. Theoretically, we could compensate for the warping, but it would just be easier to buy a different lens.

Digital Compass

One of the simpler devices is a digital compass that allows us to retrieve our absolution direction with high precision and nearly as frequently as we require. Sadly during construction of our circuits, both of our digital compasses were damaged and could not be used in the final implementation of our robot. These compasses would have been used in the control loop of the motors to allow a second check against the encoders to improve reliability and robustness of the system. Because encoders monitor relative location, there is the potential for errors to collect and over time cause errors in drive functionality. With the addition of a compass this could have been avoided as compasses are absolutely positioning devices, however this was never realized in the final design because of the failed soldering attempts with the small package.

Passive whiskers

Hardware was added to the robot to support two physical whiskers that would protrude from the front of the robot and be used as a last resort in terms of object avoidance.

Because of the high reliability of the control algorithm and speed at which the robot was able to execute movements, these whiskers were left off in the interest of time. Some preliminary tests was done with the whiskers that acted as a proof of concept and so the software aspect was able to be fully flushed out and tested. Mechanical whiskers were developed and kept on hand in case they needed to be retrofit, however because of the metallic structure of the whiskers there was also a fear of the whiskers getting caught on the felt flooring and bent under the robot causing an electrically hazardous situation, as was experienced during some initial trial runs.

Instructions for Use

In order to activate and use the robot, you must first set the desired settings on the settings DIP switch. When the LCD screen or UART terminal specifies the mode you are interested in, the small push button can be used to enter the selected mode. For our task we want the robot to be in competition mode. Once the robot has entered competition mode it becomes idle until it detects a tone signifying the robot to start. To send the tone, the user should play a 2.5 KHz or 3.8 KHz wave through a speaker system. Depending on the noise level in the room the volume of the tone may need to be adjusted, however the robot has been configured to be sensitive to very low volume tones in these regions.

Once the robot has detected the tone it will begin to search for the candle in the ‘maze’ and during this time no interference should be needed from the user. Once the robot has located the candle, it will approach the candle and notify the user through a SMS text message of the time is has taken to locate the candle. Once this has completed the robot will have successfully finished its task and the competition will complete.

Theory of Operation of Design

The final theory of operation was stripped down to a very simple collection of predetermined paths that could be sent to a control algorithm to move around the maze. Once the tone was determined, the bot would drive directly to the middle of the maze and decide whether to turn left or right based on the tone and the state of the setting switches, which indicated which side of the maze the robot was on. Once the turn was made, the robot would follow a predetermined path looking for any sources of IR light. It went about this by checking the IR cameras for a source of IR light within a predetermined threshold. Upon locating a source of IR light the robot would drive directly towards the light source in search of a white circle that encompassed the candle. Upon location of the white circle, a SMS message is sent to one or both of the team members with the time needed to complete the ‘maze’ and find the candle.

Lessons Learned

Many lessons have been learned in the course of this project. Both of us agree that this was the most edifying, exciting, enjoyable, and useful project we have yet to do here at St. Thomas. Many mistakes were made which will hopefully never be made as a result. From schematic mistakes to underestimating complexity, this project gave a great overview of how an entire project goes.

One main issue we ran into was board layout issues and schematic errors. Many traces had to be rerouted on the board, including all of the signal wires for both motor drives not being wired up originally. There were also multiple pad layouts that were interpreted upside down, resulting not being able to use our logic level shifters and the wii camera oscillator being wired upside down. These issues and others were a result of not paying enough attention to detail, not double checking work, and one member of the team being on spring break in Florida during the design of the board (which did not do good things for focus on design).

Another major stumbling block was the massive PIC datasheet we had to read through and interpret. The summary on the front page is much more of a marketing page than a technical summary of the processor. Once we were into the project a good ways, we found issue after issue and annoying nuance after nuance that was not spelled out properly. One big thing that we got lucky on was called PPS-LITE, which allows peripherals to be re-routed to nearly any pin. When we designed the board, we thought ANY peripheral could be re-routed to ANY PPS programmable pin. We were very fortunate that nearly every peripheral was routed to a pin that could be used properly.

As with any undergrad project, and most likely any project, estimating complexity and time required to complete a task was difficult. We assumed always that the next upcoming task would be relatively simple, which it almost always was, however, the scale of the task would always make it into a much longer endeavor than originally estimated. This lead to things like not fully implementing the Pixy as well as last minute issues with things like texting and the settings switches.

Design Documentation

Multi-level Costed BOM goes here

A full multi-level costed BOM (the sheet called final BOM) can be found at: https://docs.google.com/spreadsheets/d/1MLRK0OTAJ687aTI1OwpH8V0JPYjGa9OtciOKF_UKiFs/edit?usp=sharing

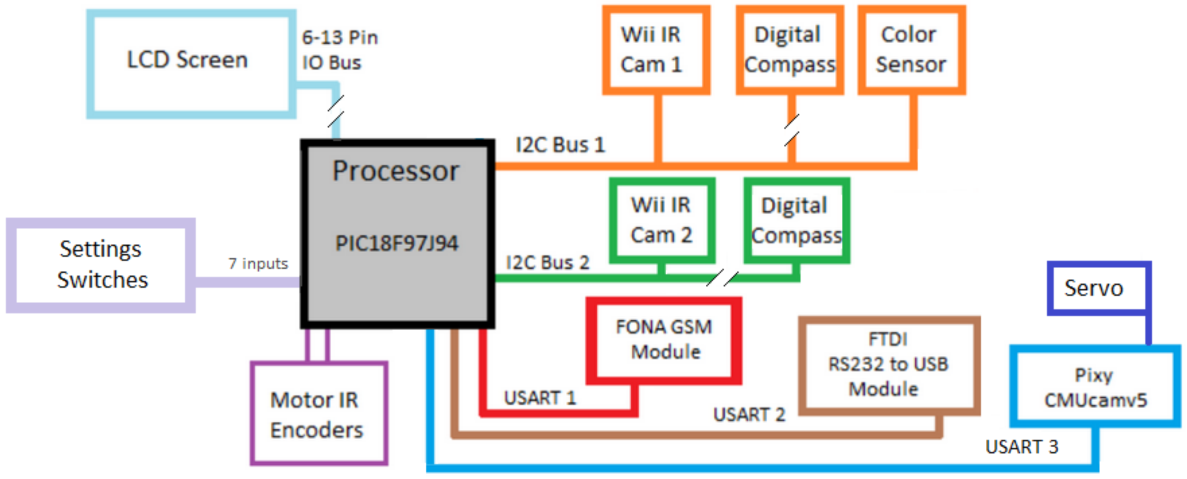
Below are pictures of it if you don’t want to go there.

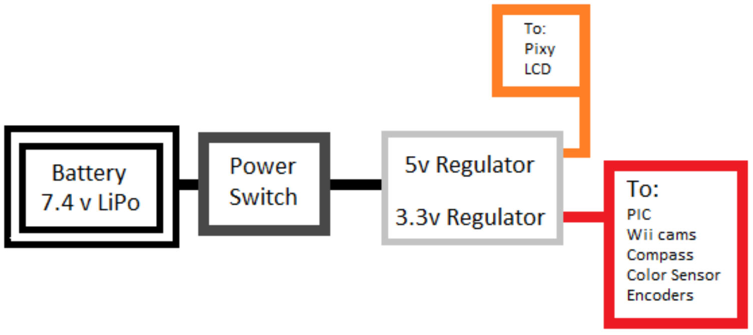
Digi-key				
1	White LED for ground color detection	http://www.digikey.com/pr	$0.41	$0.41
1	IC for ground color detection	http://www.digikey.com/pr	$2.77	$2.77
2	MC33926PNB H-bridge driver	http://www.digikey.com/pr	$3.44	$6.88
5	TXB0104RGYR logic level shifter (variable) & Tristator	http://www.digikey.com/pr	$1.69	$8.45
1	OPA334AIDBVT op-amp for electret mic amplifing circuit	http://www.digikey.com/pr	$3.14	$3.14
1	FT232RL-REEL FTDI to USB chip	http://www.digikey.com/pr	$4.50	$4.50
2	Heat sink for motor drivers	http://www.digikey.com/pr	$0.78	$1.56
1	Slide power switch, 8.5 A	http://www.digikey.com/pr	$1.13	$1.13
1	USB connector	http://www.digikey.com/pr	$0.78	$0.78
1	Powered Oscillator (25 MHz) For Wii Motes	http://www.digikey.com/pr	$1.35	$1.35
1	PIC18F97J94T - main processor	http://www.digikey.com/pr	$5.59	$5.59
1	HMC5883L compass sensor	http://www.digikey.com/pr	$3.30	$3.30
1	IC REG LDO 3.3V 1A SOT223	http://www.digikey.com/pr	$0.49	$0.49
1	IC REG LDO 5V 1A DPAK	http://www.digikey.com/pr	$0.48	$0.48
Capacitors				
5	Aluminum 20% 35v 10 uF 493-6419-1-ND	http://www.digikey.com/pr	$0.36	$1.80
10	ALUM 25V 20% SMD 100 uF 493-2190-1-ND	http://www.digikey.com/pr	$0.31	$3.10
50	Ceramic 50V 10% 0805 .1 uF 1276-1003-1-ND	http://www.digikey.com/pr	$0.03	$1.50
6	Ceramic 16V 10% 0805 4.7 uF 399-8101-1-ND	http://www.digikey.com/pr	$0.21	$1.26
3	Ceramic 16V 10% 0805 1 uF 399-1284-1-ND	http://www.digikey.com/pr	$0.12	$0.36
10	Ceramic 50V 5% 0805 12 pF 1276-1120-1-ND	http://www.digikey.com/pr	$0.05	$0.48
6	Ceramic 50V 10% 0805 33 nF 399-1165-1-ND	http://www.digikey.com/pr	$0.10	$0.60
3	Ceramic 16V 10% 0805 .22 uF 1276-1284-1-ND	http://www.digikey.com/pr	$0.12	$0.36
3	Ceramic 10V 10% 0805 10 uF 1276-2402-1-ND	http://www.digikey.com/pr	$0.20	$0.60
Semiconductors				
3	Shottkey Barrier Diode 350 mV @ 1 A RB161M-20CT-ND	http://www.digikey.com/pr	$0.44	$1.32
Conectors				

2	JST Power conn - slot B2B-PH-K-S(LF)(SN) 455-1704-ND	http://www.digikey.com/pr	$0.16	$0.32
2	JST Power conn - tab B2B-XH-A(LF)(SN) 455-2247-ND	http://www.digikey.com/pr	$0.14	$0.28
1	2x3 Pixy connector 30306-6002HB 3M15451-ND	http://www.digikey.com/pr	$0.64	$0.64
	Resistors			
10	1/8W 1% 0805 2.2k P2.20KCCT-ND	http://www.digikey.com/pr	$0.10	$1.00
20	1/8W 1% 0805 10k RMCF0805FT10K0CT-ND	http://www.digikey.com/pr	$0.10	$2.00
3	1/8W 1% 0805 1M P1.00MCCT-ND	http://www.digikey.com/pr	$0.10	$0.30
5	0.4W 1% 0805 100 RHM100AECT-ND	http://www.digikey.com/pr	$0.16	$0.80
5	1/8W 1% 0805 47 1276-5208-1-ND	http://www.digikey.com/pr	$0.10	$0.50
100	1/8W 0805 0 RHM0.0ARCT-ND	http://www.digikey.com/pr	$0.01	$0.79
5	1/8W 1% 0805 22k 1276-5356-1-ND	http://www.digikey.com/pr	$0.10	$0.50
5	.4W 5% 0805 4.7k RHM4.7KKCT-ND	http://www.digikey.com/pr	$0.10	$0.50
	Pololu			
1	LCD Screen	https://www.pololu.com/p	$8.95	$8.95
1	Pololu Micro Metal Gearmotor Bracket Extended Pair	https://www.pololu.com/p	$4.99	$4.99
2	Pololu Micro Metal Gearmotor 30:1 gearing, 1600mA draw	https://www.pololu.com/p	$6.00	$12.00
1	Pololu Rubber/Plastic wheel pair with hub	https://www.pololu.com/p	$6.98	$6.98
2	1/2" Metal Ball Coaster	https://www.pololu.com/p	$1.99	$3.98
1	Hall effect encoder pair	https://www.pololu.com/p	$8.95	$8.95
	Sparkfun			
5	specially sized micro angle bracket	https://www.sparkfun.co	$0.50	$2.50
	HobbyKing			
1	ZIPPY Flightmax 2100mAh 2S3P 7.4v	http://www.hobbyking.co	$12.70	$12.70
	MISC			
1	Microphone from Broderick		$0.00	$0.00
1	Micro Servo for Camera Apperatus Tilting from Tyler		$0.00	$0.00
1	Board from DirtyPCB (also to act as body/frame of robot)	http://dirtypcbs.com	$55.00	$55.00
1	FONA GSM Module from Tyler	http://www.adafruit.com/p	$0.00	$0.00
1	Pixy camera (salvage from personal project)	http://charmedlabs.com/d	$0.00	$0.00
1	Broken/old Wii mote to salvage camera from	http://www.craigslist.com	$0.00	$0.00
				$120.89

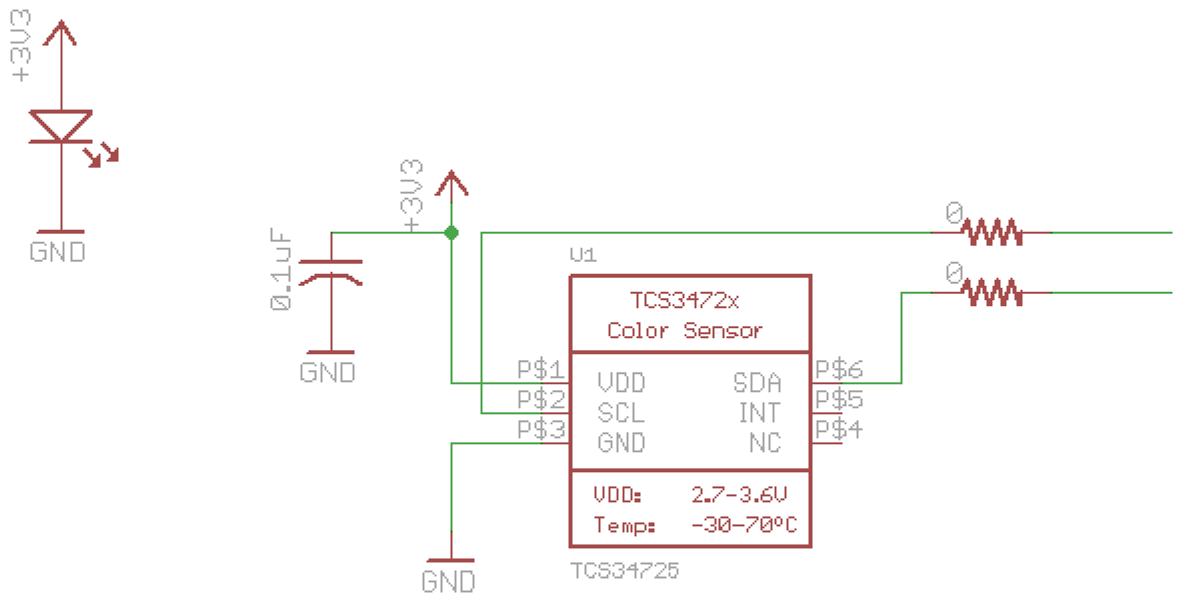
Drawing Package

System Block Diagram

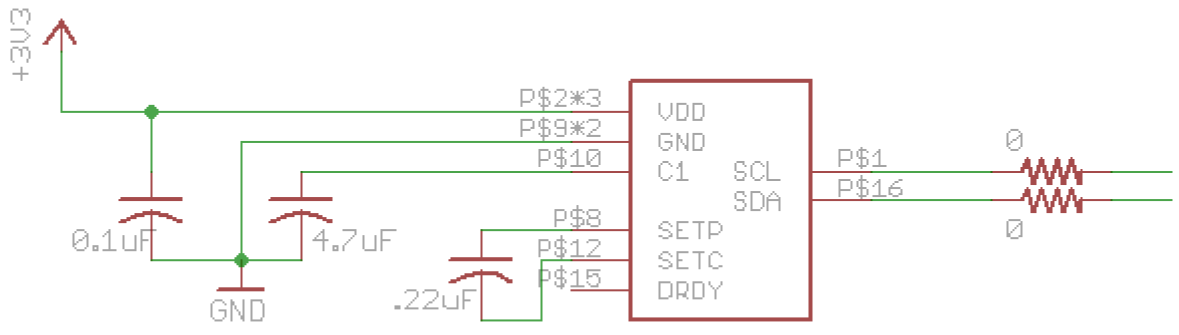




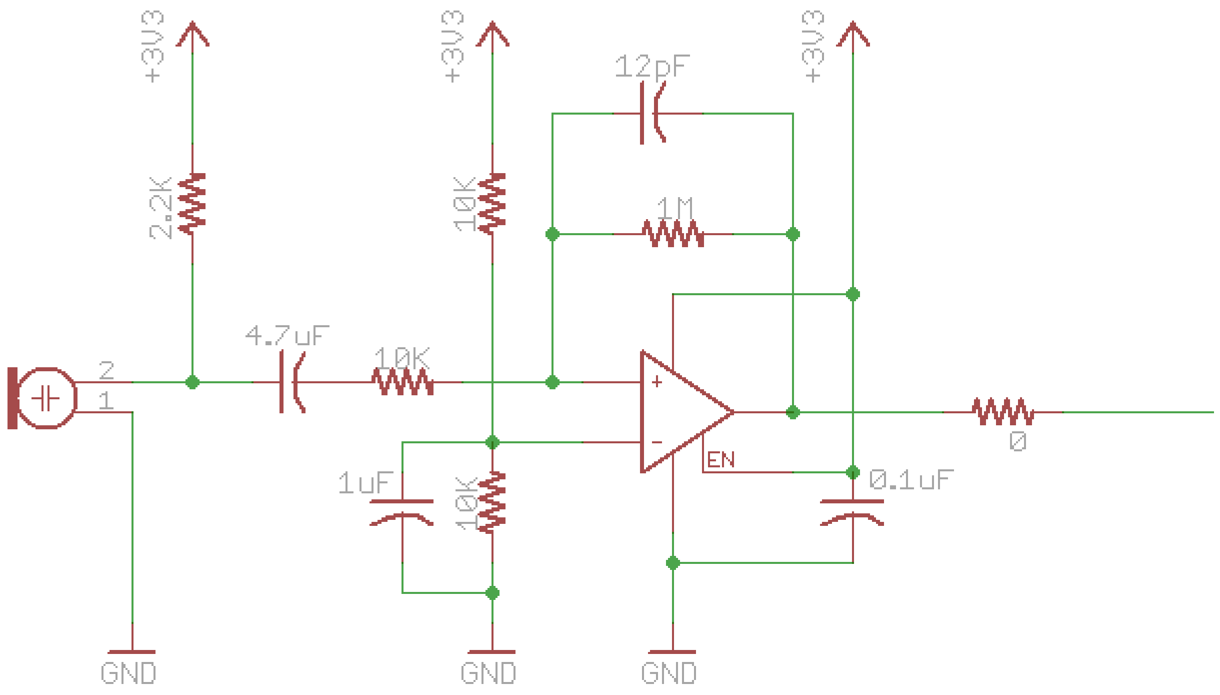
Schematics and Boards



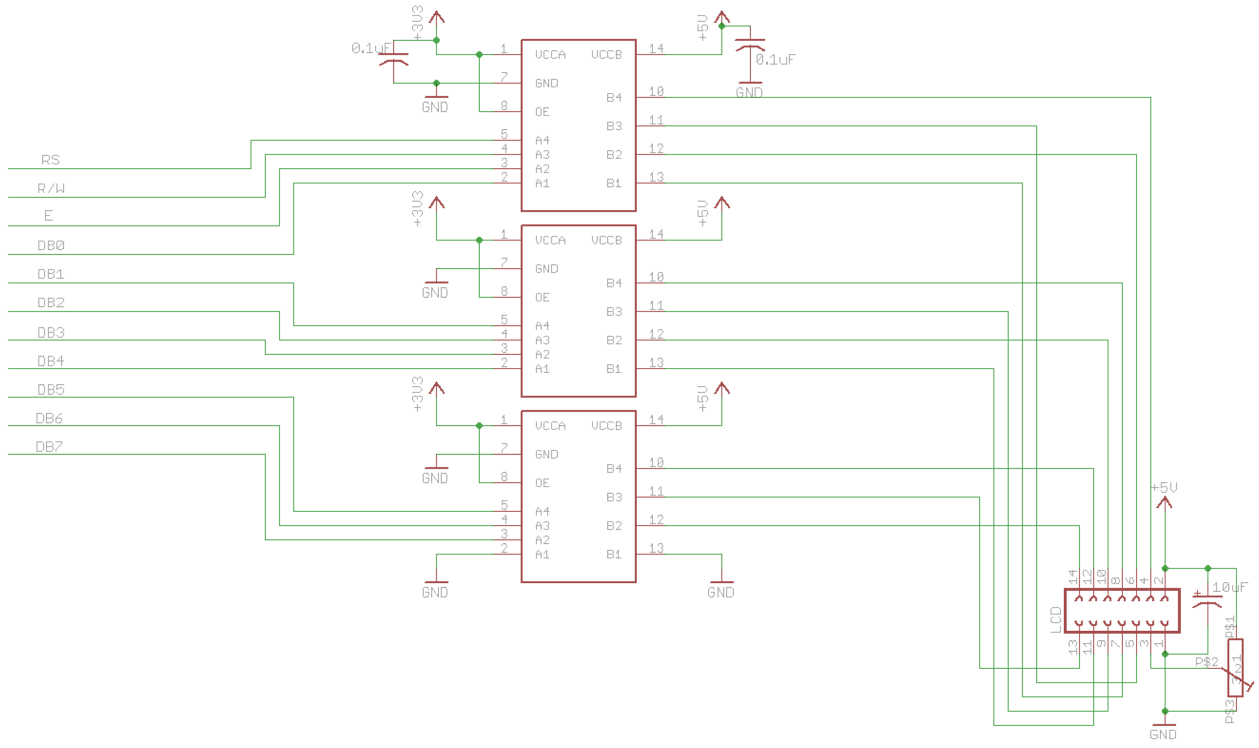
Color sensor and LED wiring



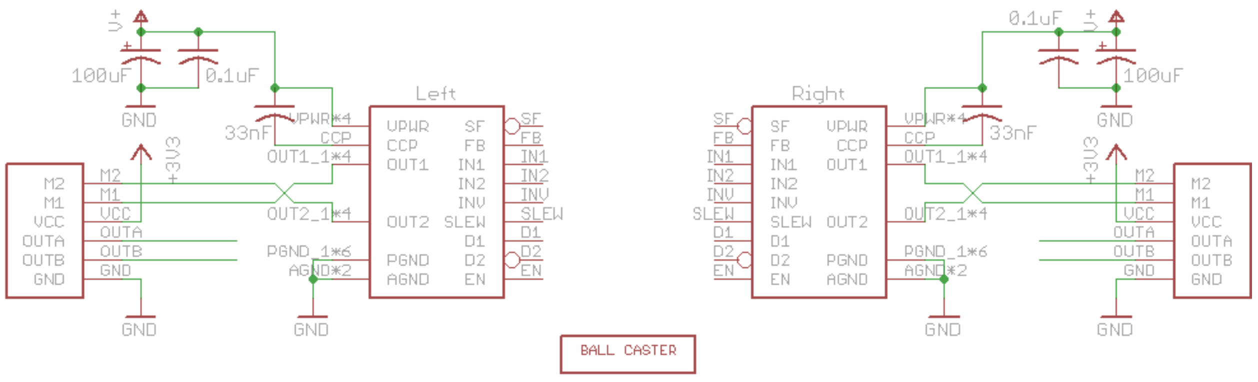
Compass sensor wiring



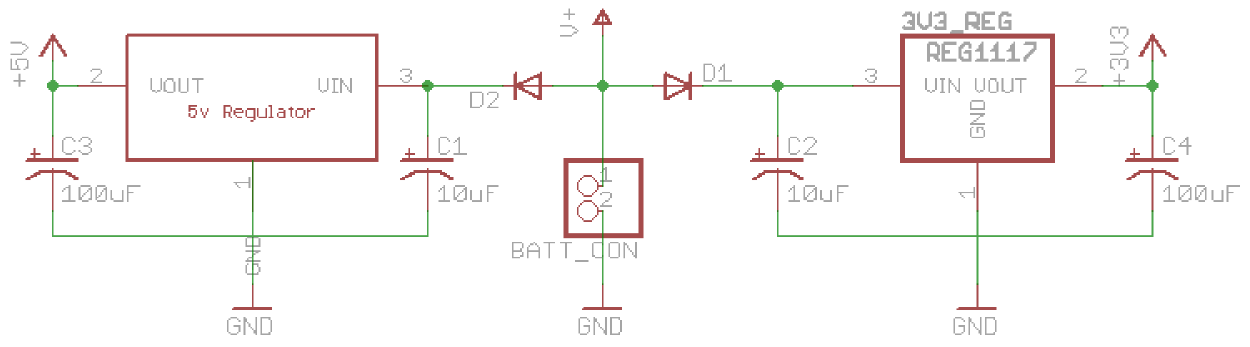
Electric mic conditioning circuit



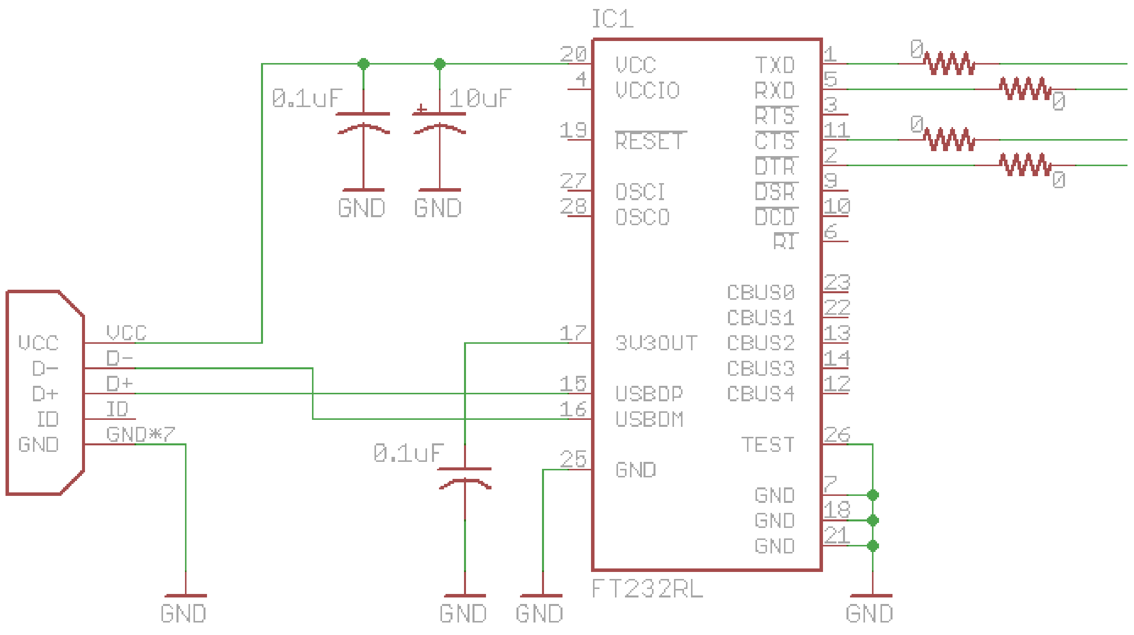
LCD wiring circuit



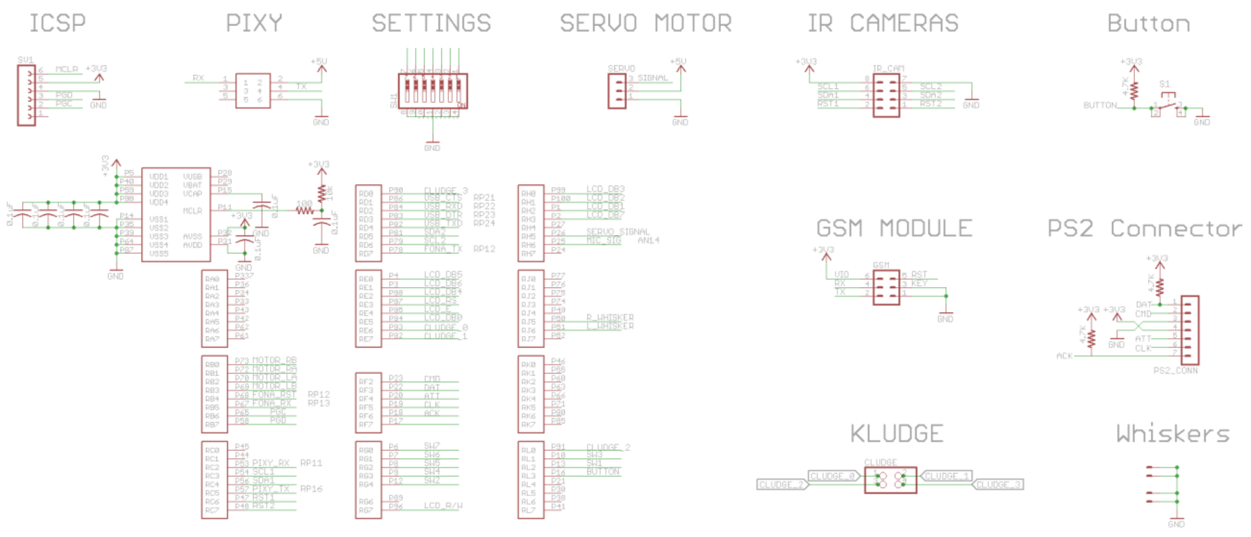
Motor drive wiring



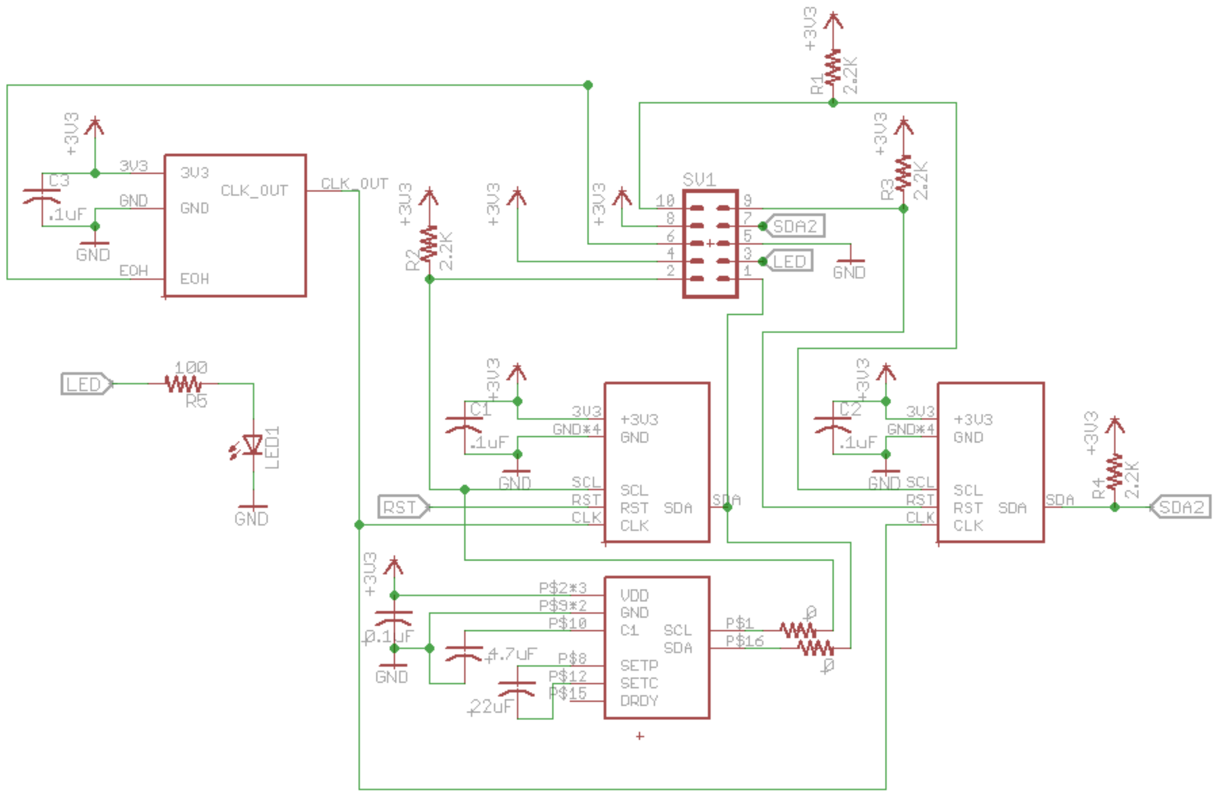
Power regulation



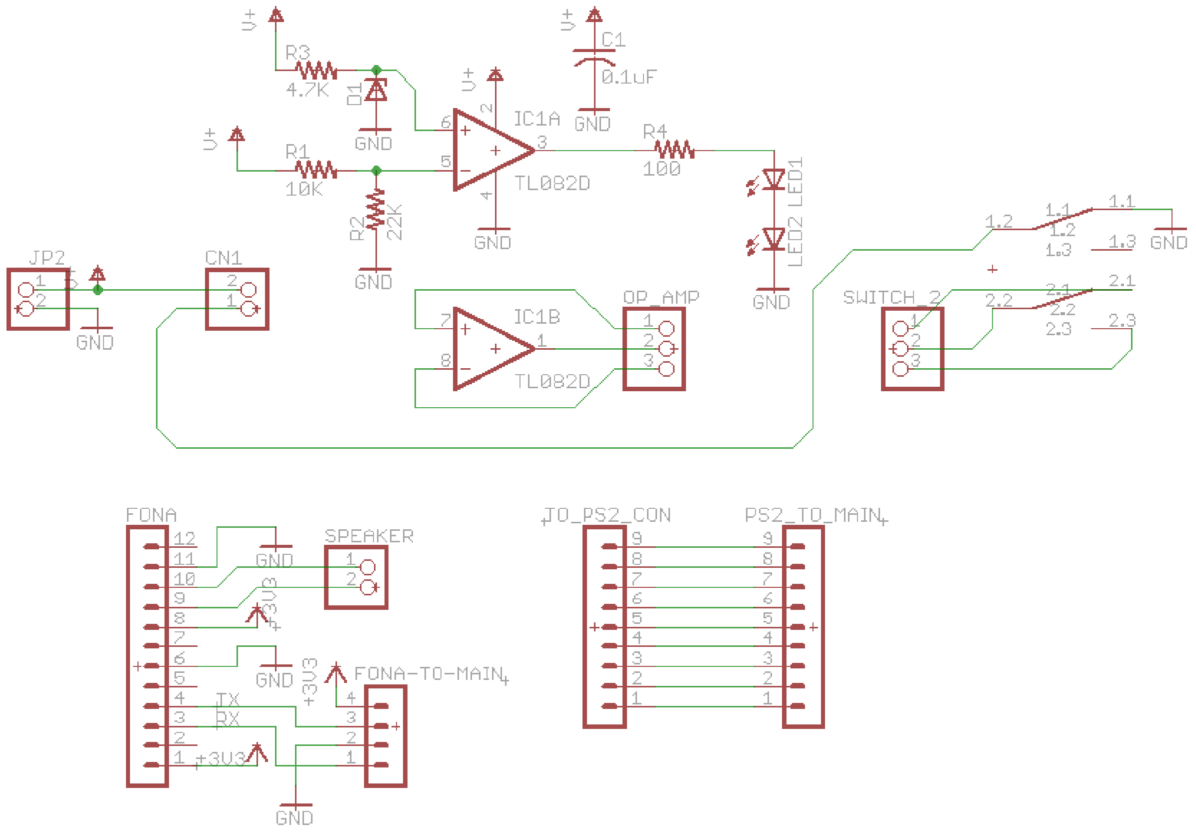
USB FTDI wiring



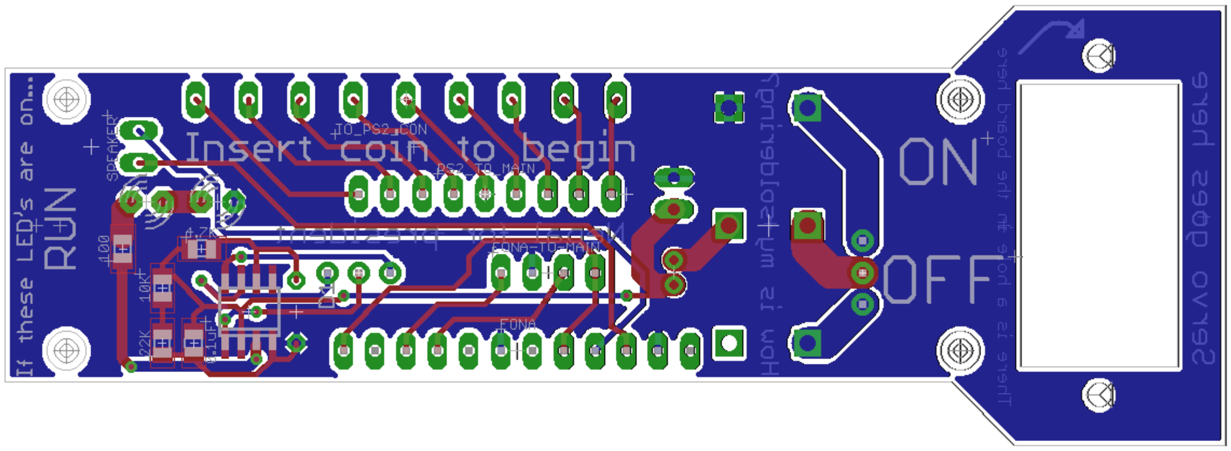
PIC18F97J94 wiring along with miscellaneous connectors going to other boards



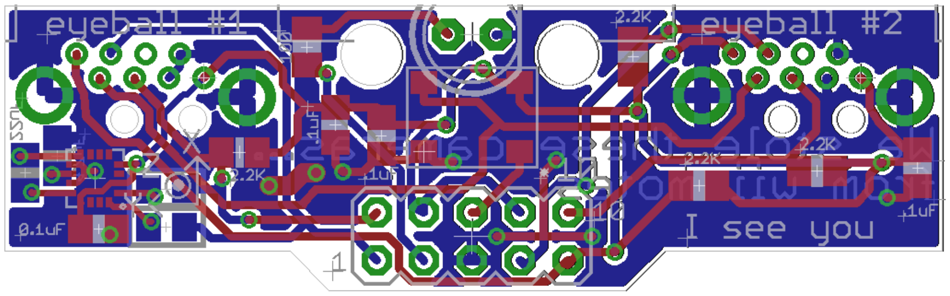
Wii-Pixy sub board schematic. Has two wii cameras, an oscillator, and a compass on it



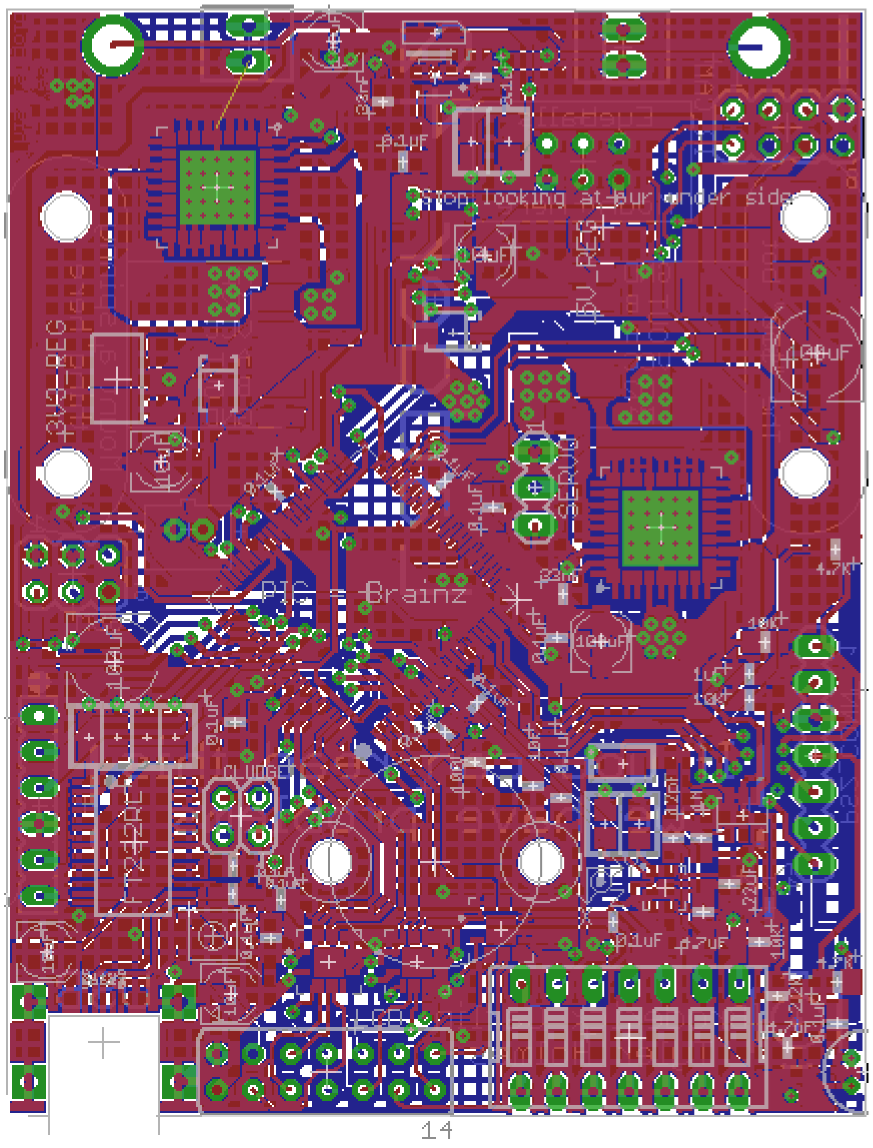
Fona, PS2, power switch, and low battery warning circuitry



Fona/PS2/Power companion board

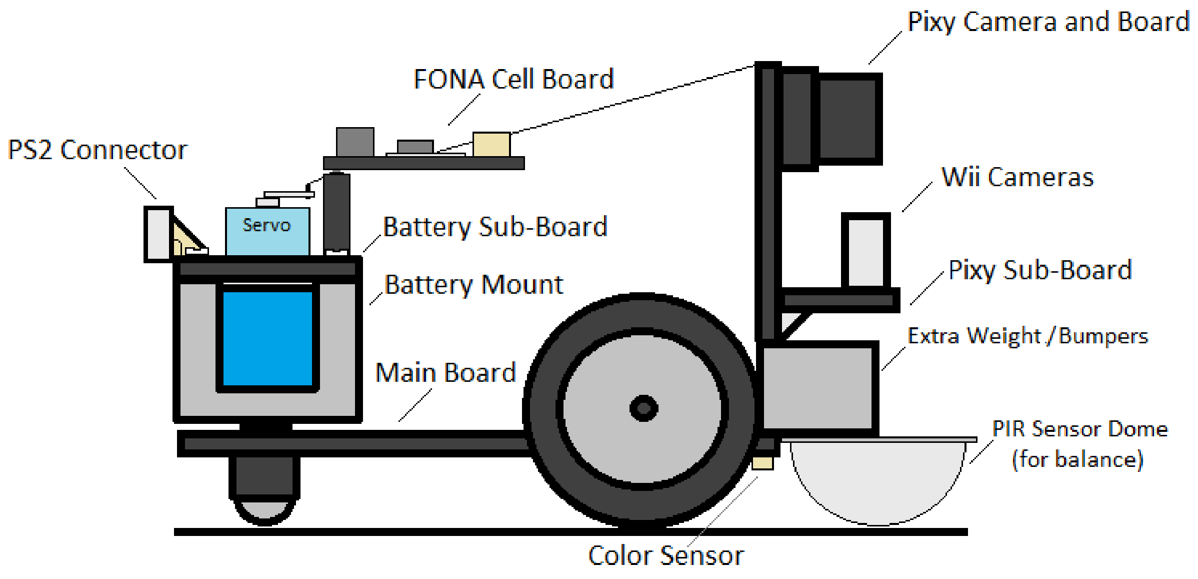


Wii-Pixy board



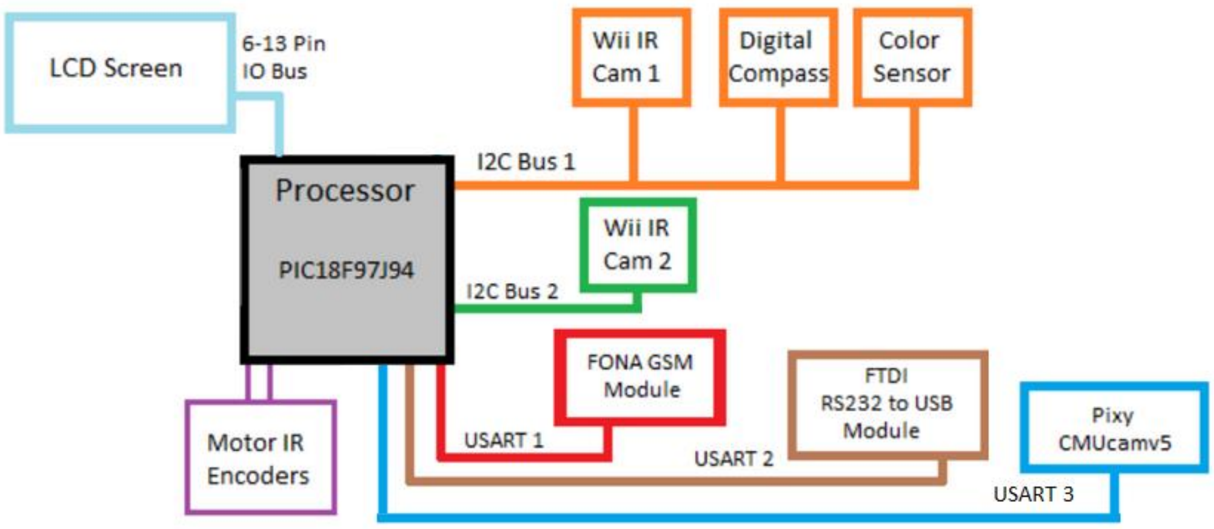
Main board

Mechanical Design



Source Code Package

Algorithms and Flowcharts



Annotated Index to Source Code Modules

PIC software:

NOTE: every file listed below has a corresponding header file in which a detailed description of each function resides.

Communication:

I2C.c
: custom functions for initializing and handling I2C communications.

UART.c
: custom functions for initializing and handling all of our UART communications, including custom functions much like the LCD code that allow you to print variables inside of strings.

Sensors:

colorSensor.c
: functions taking care of the initialization and reading from the color sensor.

compass.c
: functions taking care of the initialization and reading from the compass sensors.

wiiCams.c
: functions adapted from JOHNNY CHUNG LEE’s code for initializing and communicating with the Wii IR cameras.

encoders.c
: functions taking care of the initialization and keeping track of how far the motors have traveled.

fft.c
: functions adapted for use on our PIC that perform a DHT on the audio data to analyze the dominant frequencies, as well as custom analysis functions.

FONA.c
: functions based off of the Adafruit Arduino functions for interfacing with the FONA GSM module.

whiskers.c
: functions for initializing and reading the mechanical whiskers on the front of the robot. Never fully developed or implemented.

Utilities, Settings, Other:

main.c
: the main file where initialization occurs as well as the main logic for the entire robot.

GlobalDefines.h
: various defines, constants, and in-line functions.

config.h
: header file taking care of the fuse bits setting up the basic function of the PIC18F97J94.

delays.c
: custom delays functions for various lengths of times.

LCD.c
: custom functions for initializing and communicating with the LCD with print functions that behave like the printf function, with the ability to print an arbitrary number of variables inside of a string.

motorDrive.c
: functions for interfacing with the motor drives, controlling speeds of the individual motors.

PS2.c
: functions adapted from existing Arduino code which bit-bang a modified SPI protocol for communicating with PS2 remote controls.

settings.c
: functions for initializing and reading our seven settings on the DIP switch we have on our robot.

lines of code, c files only: 3,730
lines of code, including .h: 5,608

Custom Pixy Software:

edgedetect.cpp

this is where I did everything in regards to final competition mode edge detection.

main_m4.cpp

only modified this function to, after it has initialized everything, go into my function where it will run forever or into the debugging mode, where I can see what the pixy sees.

progvideo.cpp

modified version of the mode where the pixy streams what it sees as well as a “cooked mode” where I can see how it processes the image.

Documented Source Code

Fully commented source code can be found online at our GitHub pages:
https://github.com/beeedy/candleBot

https://github.com/TDHolmes/pixy/tree/master/src/device/video

Bibliography

Wii Camera Reverse Engineering

http://procrastineering.blogspot.com/2008/09/working-with-pixart-camera-directly.html

http://www.wiibrew.org/wiki/Wiimote

Edge Detection Book

Practical Algorithms for Image Analysis by Michael Seul, Lawrence O'Gorman, Michael J. Sammon