

# Table of Contents

1. Abstract Page 1

2. Introduction Page 2

3. Navigation System Page 4  
 i. GPS Page 4

ii. Magnetometer Page 5

iii. Encoders Page 6

4. Targeting System Page 7

5. Object Avoidance Page 10

6. Microcontroller Page 12

7. Motor Controller Page 13

8. Power Page 14

9. Hardware Page 15

10. Conclusion Page 16

11. Recommendations Page 19

12. References Page 20

13. Appendices Page 21

i. Equations Page 21

ii. Purchased Items Page 22

iii. Milestones Page 23

iv. Code Segments Page 24

# 

### Abstract

Capsule Corp is tasked with the role of designing an autonomous robot that is capable of detonating IED devices placed in tight quarters. At the end of the design process, the robot will take part in the Autonomous Payload Delivery Challenge. Since IEDs are dangerous to handle, beacons that are emitting an 80KHZ signal will be used instead. Utilizing a global positioning system (GPS) unit, a Magnetometer and wheel encoders; the robot will navigate to the beacon and drop a payload. The robot is also equipped with an ultrasonic sensor which detects obstacles it needs to avoid.

### Introduction

The goal of this project was to design a vehicle that will drop a payload at beacons randomly placed in a field. Dropping a payload at a beacon represents the success of detonating an IED in a real life scenario. The components that make the vehicle autonomous were chosen with care to ensure that they perform as speculated. Without confidence in the component performing to expectation, the device would have to be replaced, lengthening the build process. For the structure of the robot, a prebuilt chassis from Lynxmotion was chosen since it allowed the integration of a differential drive system. A Sabertooth motor controller controls the speed of each motor depending on the output of the STM32F0 microcontroller. The STM32F0 is the brain of the project since it would analyze the incoming signals of the sensors using I2C, USART, and GPIO inputs. After the signals are processed, the microcontroller would command the vehicle to move accordingly.

In order to travel to GPS coordinates, a navigation system was implemented. The navigation system on the vehicle consists of a GPS unit, a magnetometer, and wheel encoders. The GPS allows the vehicle to know exactly where it is located, so a vector can be calculated between its location and the location of the first beacon. Once the vector is calculated, the heading of the vehicle would be polled from the magnetometer to know the heading of the vehicle. When the heading of the vehicle is close to the calculated heading the vehicle drives towards that heading. The wheel encoders will keep track of the distance that the vehicle had traveled. It would also keep the car going relatively straight by correcting the speed of the wheel if any wheel slips should occur.

In a real situation, while going to a GPS location there would always be obstacles in the way. In order to circumvent these obstacles the vehicle would make use of an ultrasonic sensor that is attached to the front bumper. The object avoidance system would activate when the vehicle is 1 to 3 feet away from an obstacle. Once the obstacle had been detected the vehicle would execute a 90 degree turn to the right, then drive a foot and then execute another 90 degree turn to the left. It would then check to see if an obstacle is still in the way. If it is, the process would be repeated until there is no obstacle detected in the path of the calculated vector. After the obstacle is cleared, it would execute the targeting system. Even with ideal conditions the GPS is not 100% accurate, so a targeting system is used to further close in on the beacon when in range.

The targeting system makes use of two ferrite rod antennas that are custom built and tuned to 80 kHz. The antenna would detect a beacon emitting an 80 kHz from about 6 feet away. There are four parts to the antenna system. These parts would collect a signal, amplify it and regulate it to get a clear output. The scalar would then adjust the analog output based on proximity. This clean, amplified signal would be sent to an analog to digital converter in the microcontroller. Once a signal is detected on the antenna, the microcontroller would halt the motors and signal the Sabertooth to rotate on the spot to find the exact point where the two antennae pick up the same signal strength. This would mean that the beacon is directly in front of the vehicle indicating the vehicle can then drive until the signal has peaked and drop the payload.

In order to score points in the competition, the vehicle must drop payloads at the designated beacons and return to the starting point in under 2 minutes. Also, the vehicle build must not exceed a $750 budget in cost and be no larger than a 20”x20”x20” cube in size. The tasks mentioned above must be performed at each beacon location, ensuring that the payload is dropped relatively close to the ring of the beacon for full points.

### Navigation System

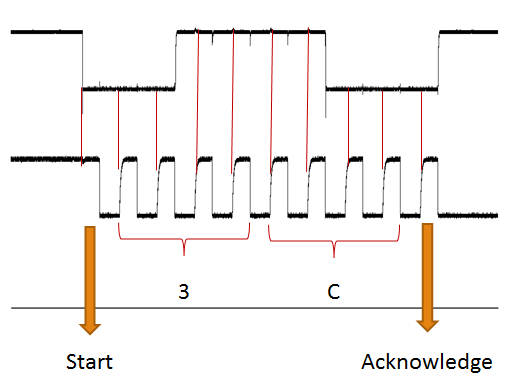
When it comes to the navigation system of the robot, a GPS unit, a magnetometer and wheel encoders were used. These sensors effectively put the robot on the correct path to the beacon. The GPS was used to inform the microcontroller of the current location of the robot, while the magnetometer was used to give the current cardinal bearing. The wheel encoders are used to counteract inaccuracies in the GPS, which can be of from 6 to 10 feet, by counting wheel rotations. The magnetometer used was a triple-axis magnetometer which would not get affected in the event the robot tilts due to uneven terrain.

### GPS

The Adafruit GPS communicates with the microcontroller using the USART serial communication protocol. The microcontroller is using an interrupt service routine to receive data from the transfer port of the GPS so it would not interfere with the inputs from other sensors. The GPGGA string is parsed to obtain the longitude and latitude position of the vehicle. Once the GPS coordinates of the beacons locations are inputted, an algorithm would calculate the vectors to those locations. To calculate the resultant vector, two GPS locations are needed: the current location of the vehicle and the location of the beacon. The data from the GPS is then converted to degrees and used to calculate the distance using the Pythagorean Theorem shown in the appendix. As for the heading, the atan2 function in the appendix is used to obtain the heading between point a and point b. Using this newly calculated information the vehicle knows how far it has to travel before it has reached the proposed location of the beacon.

### Magnetometer

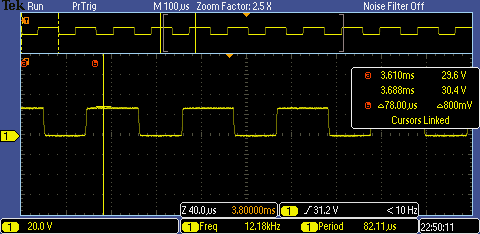
The purpose of the magnetometer is to know the current heading of the vehicle. Due to the magnetic field created from the motors and other components, the magnetometer was mounted on top. It was discovered that the write address and read address were 0x3C and 0x3D respectively. Since ST the manufacturer, has an I2C library for the STM32F0 board, which was used to set up the foundation of the communication process. Then, by calling certain commands from the library in the correct order, data can be pulled from registers. The microcontroller was set to constantly poll the magnetometer for information in an interrupt service routine. The magnetometer was set up to send out the average of 8 samples at a rate of 15 hz. Each time it sent 6 outputs, which are the MSBs and LSBs of the x,y,z registers. In order to calculate the heading of the vehicle, the atan2 function was used with the values polled from the magnetometer.



### Encoders

Due to uneven terrain and the grass field, wheel slips can occur and cause our robot to go off course. Even with equal voltage being given to each motor, they may not be balanced in performance. The motors may rotate faster than others, this being a side effect of differential drive systems. To counter this, wheel encoders were utilized as another measure to ensure our robot stayed true to its path. The wheel encoders gave a pulse width feedback that was viewed on an oscilloscope and any discrepancies were noted. If the microcontroller detected the robot veering off course, via code that compared actual wheel encoder value to calculated ones, it signals the motors to adjust its trajectory. The feedback from the encoder lets the vehicle know the distance it has travelled, as well.

In the example below, the feedback from the wheel encoders with friction is around 12 kHz. When the difference between the sides of the robot is outside a certain range, it will adjust itself. Using the difference in frequencies, rotational speed can be calculated. An algorithm was used to do this as shown here.



### Targeting System

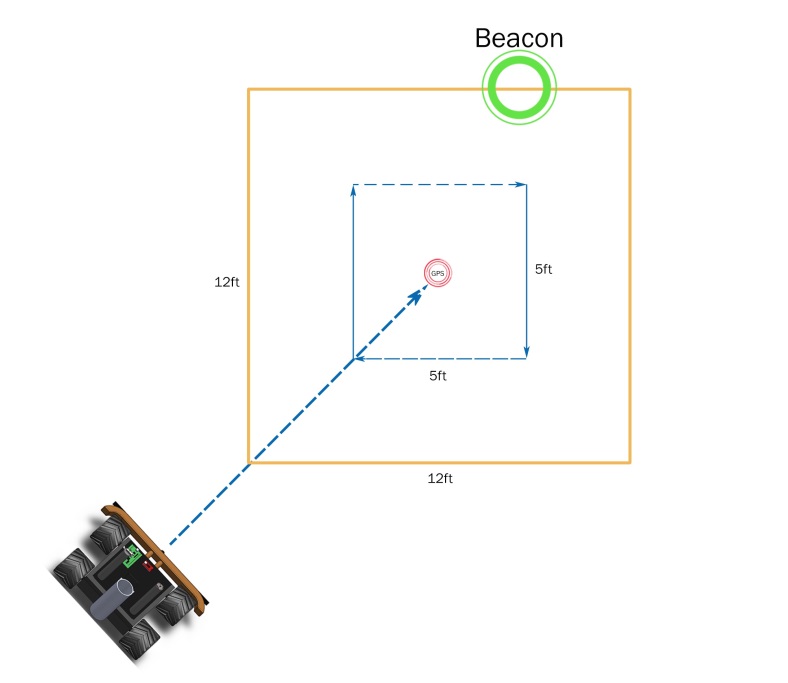
A major problem with commercial grade GPS modules is their inherent inaccuracy. Through extensive testing, the Adafruit Ultimate GPS module used on the robot, was shown to be accurate within six feet of our actual target which is a ring emitting an 80 kHz signal. The ring is approximately one foot in diameter needing a fix for the inaccuracy of the GPS. Capsule Corp’s solution was to take advantage of the fact that the target would be emitting the 80 kHz sinusoidal signal to further narrow in on the target. The plan was to build two beacon-detecting antenna circuits that would be mounted on opposing sides of the front end of the vehicle. In this way, the robot could determine, based on signal strength, its relative location to the beacon. With this information, the robot would be capable of navigating itself directly to the beacon to deliver the payload on target.

The first step in achieving this goal was to design a loop antenna capable of detecting the 80 kHz signal. Many experiments were made in regards to the antenna’s coil wire length and thickness. It was observed that either more windings about the ferrite core or the use of thinner gauge wire increased the inductance of the antenna. At a point of about 800 turns, the inductance became so great that the capacitance required to tune the antenna became too small and too sensitive, as well as the capacitance within breadboard. Choosing too thin of a wire had a similar effect. So, after some trial and error, the team settled on a wire gauge of 29 and 700 turns about the ferrite core. This yielded the desired signal pickup at the estimated minimum distance of three feet, without the sensitivity to capacitive variations.

With a properly tuned antenna, the next step was to amplify the signal so that it could be seen and measured at distances equal to or greater than three feet without the help of the oscilloscope’s internal amplification. In an effort to keep the circuit simple, an operational amplifier was used of this purpose. With setting the minimum voltage and maximum voltage into the operational amplifier to 5V and 0V respectively, the negative portion was automatically cut-off and only the positive portion was left. Effectively, in one step the signal was amplified and rectified. With this set-up the range from which the antenna could accurately detect the signal was about three feet.

In order to use the rectified signal with the microcontroller, it would be converted to a DC voltage. To accomplish this, a peak detector was used. The peak detector circuit was comprised of a capacitor, to hold the peak voltage value, and a resistor to allow the capacitor to slowly discharge to ground. A Schottky diode was also added to the output of the operational amplifier to ensure that the capacitor discharged through the resistor and not back through the amplifier. Finally, before the signal was passed to the Analog to Digital Converter on the microcontroller, a protective circuit was added. The STM32F0 Microcontroller that we used can only handle DC input voltages of less than 3.3 Volts, so a Bat54s was added between 3.3 Volts and Ground.

With the beacon detection circuits built, the next step was to develop a search routine for when the robot reached the GPS location. The search routine needed to ensure that the robot would get close enough to the beacon in order for either of the two front mounted antennae to pick up the signal. The search routine chosen, which is shown in the figure below, was simple, yet effective.



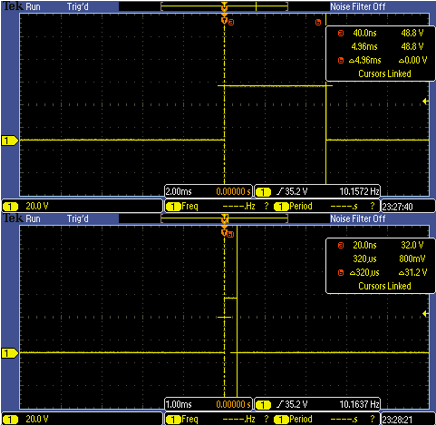
The robot would stop three feet short of GPS, execute a 45 degree right turn and then three 90 degree left turns, forming a 5 by 5 foot square and concluding where it started. The front mounted antennae are separated by one foot and the beacon antennae have a range of about three feet. The area searched is at least 12 square feet which is more than enough for the given inaccuracies of the GPS. Along this path, one or both of the antennae are predicted to detect the signal. If the voltage detected on one of the antennas is lower than the other, the robot will rotate in the direction of the stronger signal. It will keep rotating until the signals on picked up on each antenna produce an equal voltage. Once this is achieved, the beacon must be either directly in front of or directly behind the robot. To check, the robot will move one foot forward and reassess whether the signal strength has increased or decreased. If it has increased, it is heading in the right direction and proceeds forwards until it reaches a tested maximum signal strength at which point the robot will stop, drop its payload, and calculate a new vector to the next target.

The Payload Delivery mechanism is a simple, effective design. It consist of a 2.5 inch diameter plastic tube, oriented vertically, containing a horizontally mounted servo controlled ring at its bottom. The tube holds the three payloads stacked on top of a ring and on each 180 degree rotation of this ring, it will drop one payload and automatically load the next payload. On the next drop the ring needs to only rotate another 180 degrees in the opposite direction to drop the next payload. We used the HITEC HS-322HD servo which has a stall torque of 41.66 oz/inch. This was more than enough for payload delivery because with a total of three golf balls loaded it would be required to lift weighs only 1.62 oz. The servo was powered by the 5 Volt output of the Sabertooth and had a running current draw of about 200 mA. The servo was controlled using the microcontroller by sending pulses of varying widths corresponding to servo positions. A pulse width of 600 uS would rotate the servo all the way in one direction and a pulse width of 2400 uS would rotate it back the opposite way.

### Object avoidance

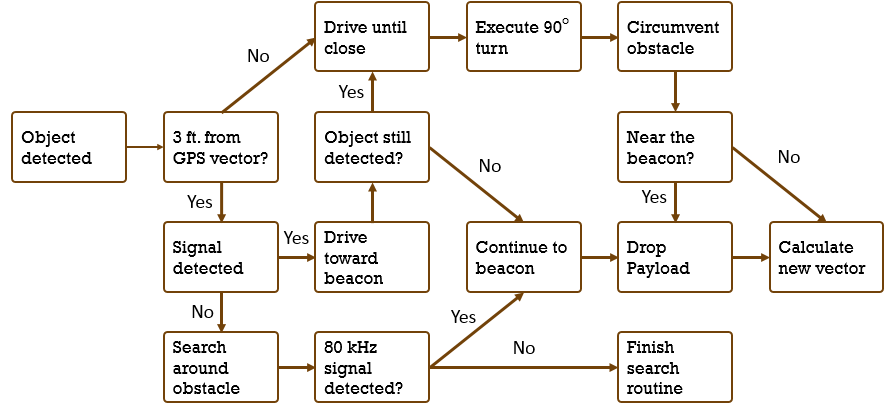
Obstacles will be scattered throughout the field of play, so some form of obstacle avoidance is needed. Capsule Corp’s object avoidance system consists of one ultrasonic sensor mounted on the front bumper of the chassis, since all objects detected will be encountered directly ahead of the car. Specifically, the ultrasonic sensor Maxbotix HRLV-EZ4 was chosen. This sensor fits perfectly within the requirements demanded: a low-cost ultrasonic range finder with object proximity detection from 1 millimeter to 5 meters and a resolution of 1 millimeter.

The first step to accomplish an object avoidance system that met the requirements demanded was to test the ultrasonic sensor and learn its functions. An ultrasonic sensor is a device that emits ultrasonic sound waves that bounce back when hitting objects, thus the time measured from the generation of sound to receiving it back judges distance. The output waveform changes were tested using an oscilloscope when an object is moved closer or farther away from the sensor. The data points were recorded in correlation to the distance at which the object is at. The output waveforms obtained are shown in this figure.



From the testing phase, it was found that the Maxbotix HRLV-EZ4, a pulse-width of one microsecond equals one millimeter, as specified by the manufacturer. The top figure shows object detection from 16 feet and the bottom figure shows one of 1 foot, which is the distance considered from where the robot is close enough to the object. As shown in the figure, for 1 foot the wave pulse-width is 320 us, since 1 foot equals to 320 mm.

The next step was to develop the object avoidance algorithm. Some algorithms were conceived before the final one was chosen. An important question involving the conception of the algorithm was the necessity to overlap both object avoidance and targeting systems. Since presumably the obstacles will be placed nearby the predetermined locations, the data from the antennae into the object avoidance algorithm was needed. The final object avoidance algorithm implemented is shown here.



Object Avoidance Algorithm

When one of the obstacles are encountered, the microcontroller check whether or not the robot is on the targeting zone, 3 feet from the GPS vector. If this condition is satisfied, the robot starts a search routine and if not it circumvents the obstacle, driving towards the beacon whenever the 80 kHz signal is detected. If, finally, the car is near the beacon it will drop the payload and if not, the microcontroller will reassess the GPS coordinates to calculate a new vector to the target, plotting a new course.

### Microcontroller

All of the signal processing and data calculation were handled by the STM32F0 Discovery board (microcontroller), made by STMicroelectronics. For this project, the STM32F0 board utilized one UART port, one I2C port, and eight GPIO ports for communication with peripheral components.

GPS uses UART to transmit coordinates in a data string labeled GPGGA, which the microcontroller parsed into latitude and longitude information for distance and bearing calculations. Magnetometer passes information via I2C in three dimensions. The microcontroller discarded the z component and receives the information as heading from a magnetic compass. Five of the eight GPIO ports are used to receive sensor information. DC voltage from each of the two antennas, sampled via analog digital conversion (ADC), was correlated directly to distance from signal strength. Pulse width modulated (PWM) signals from both wheel encoders provided wheel RPM feedback, as well as distance traveled by the robot as a function of time due to linear motion. Ultrasonic range finder returns a distance value was used to as a part of the object avoidance algorithm.

The remaining three GPIO ports output were used to output commands. One was to give PWM signal to the payload delivery system, commanding the servo motor to turn when the robot has reached its location. The last two GPIO channels sent PWM signals to the motor controller, giving separate commands to each side of the vehicle to achieve differential drive. The microcontroller also provides 3.3VDC output to components that cannot operate on 5VDC, while able to take in analog and digital inputs between 0V and 3.3VDC.

### Motor Controller

The propulsion of the robot was provided by Sabertooth 2x12 motor controller, made by Dimension Engineering and provided as a part of the chassis kit. The motor controller can provide up to 12A of current to two separate DC brushed motors. To achieve the differential drive configuration, two motors on each side were connected in parallel to draw power from each of the motor controller outputs. The motor controller operated in analog mode, with each side receiving PWM signals to control motor speed. Specific pulses widths correspond with motion, with 1ms pulse for full reverse, 1.5ms for no rotation, and 2ms for full forward. A combination of predetermined pulse widths, combined with encoder feedback, allows precise robot locomotion with straight line, rotate (turn in place), and arc paths.

The motor controller also proved a 1A 5VDC output, to provide power to various components. All of the sensor components drew power from the 5VDC output or the microcontroller, while the motors drew power from the battery directly. This eliminated the need for a separate transformer and greatly simplified the design, as well as reduced noise output to interfere with sensors.

### Power

All components were powered by a 7.2VDC 3800mAh flat NiMH battery pack. The four motors drew power directly, while the microcontroller and peripherals that uses 5VDC drew power from the motor controller 1A 5VDC output, and the components that required 3.3VDC were powered by the microcontroller. The total power consumption by the various components totaled 789.3mA, as detailed in the table below.

|  |  |  |
| --- | --- | --- |
| Component | Voltage Requirement (VDC) | Current Draw (mA) |
| STM32F0 | 5 | 11 |
| GPS | 3.3 | 56 |
| Antenna (2) | 5 | 3.1 x 2 |
| Ultrasonic | 3.3 | 100 |
| Magnetometer | 3.3 | 0.1 |
| Payload Delivery | 5V | Up to 700 (stall) |
| Encoder (2) | 5V | 8 x 2 |
| Total |  | 789.3 |

### Hardware

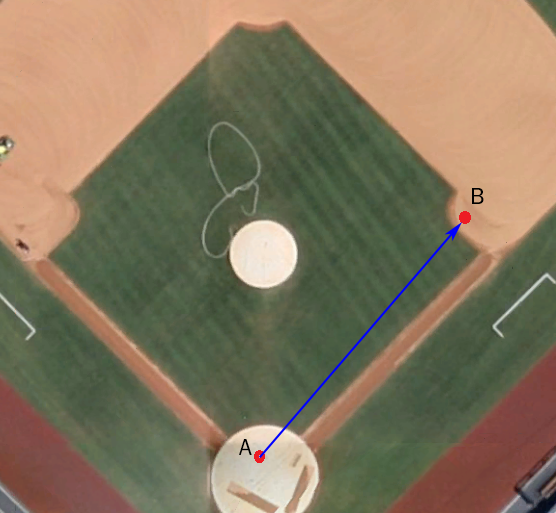
The chassis, motor, motor controller, and wheels were purchased as a kit from Lynxmotion. The chassis is aluminum in construction, with fiberglass plates covering to house the motor controller, battery, and internal wiring. Four identical motors were used, with one encoder attached to each rear motor for feedback (for specifications please view the Appendix). The microcontroller is mounted on top of the robot for ease of programming. Mounted in the front of the vehicle on a bumper are two antennas on each, and ultrasonic sensor directly in the center. The left side has a PVC pipe elevating the magnetometer approximately six inches from the motors to avoid interference, while the right side contains the main switch and the motor power switch. The payload delivery system is mounted in the rear, which provides clearance to the payload once deployed.

A single printed circuit board (PCB) centralized all signal routing. The microcontroller is mounted directly to the PCB, while all peripherals have their power, ground, and signal pins laid out on the perimeter of the PCB. All wires are routed to the inside of the chassis, and connected to the PCB in a bundle through an adjacent opening. Two additional PCBs, one for each antenna, grouped all the analog circuitry for 80 kHz beacon signal processing, as well as served as a physical mount for the antenna windings.

### Conclusions

The goal of this build process was to create an autonomous vehicle that can locate and dispose of improvised explosive devices. While the use of the systems and sensors to complete this task would have worked perfectly in theory, there were some pitfalls that were unforeseen.

In terms of Navigation, the system worked as intended. The most difficult part of implementing this system was trying to implement the GPS and magnetometer systems. The STM32F0 microcontroller required some knowledge of both USART and I2C to set up the input ports so that they could be implemented. Once the systems were interfaced, the algorithms to pull data of the modules needed to be written. The GPS would be constantly outputting data that was irrelevant as well as useful statements. Due to this, a parsing algorithm was written in an interrupt. This interrupt would sample data from the GPS. After valid data was parsed from the module, the vector was calculated using this data and predefined locations. We found that these vector plots were very accurate as shown here:



The vector plotted in this picture is the distance between home plate and first base at Petco Park. The distance was calculated to be 90.32 feet while the actual distance is 90 feet. This was a very good tolerance of accuracy as it is within a foot of error.

The magnetometer also performed to spec. The actual bearing received was very accurate, as leaving the robot facing a direction dialed the magnetometer to the exact bearing from true north. The update rate was fairly slow at 15Hz. This was more of an issue for keeping the robot going true. After creating numerous “go straight” functions, it was found that the slow nature of polling the magnetometer for data often could not keep up with how fast the robot moved. This was countered by developing code that would stop after driving a certain distance and recalculate a vector to the GPS location. Using this system, the robot would need to get a magnetometer update only when stationary. The wheel encoders would then handle the robot staying relatively true in lieu of the magnetometer.

The targeting process was not implemented in the final design. The primary reason for this was the lack of experience with creating custom antennas. After winding up a ferrite rod and attempting to rectify and amplify the signal, it was found that combining the systems on a PCB resulted in all the parts being altered. This alteration was so vast that retuning the antenna and reconfiguring the amplification and rectifying stages was not feasible in the time before competition day. It was thus decided that the entire system would be removed from the robot. The algorithm itself was written in pseudo code, however. The robot would just drop a payload at the end of the GPS location vector. The payload delivery worked as designed.

The object avoidance system worked very well. The ultrasonic sensor, as previously discussed, had outputted distances very accurately. The only issue that it had was that, occasionally, there would be a “ghost obstacle” issue. Here the ultrasonic would either briefly ignore a current object being detected or detect an obstacle that was not there. This was simple to overcome with the object avoidance code in the appendices. The code will delay sampling to confirm that any object stopped for was indeed an object or a ghost image.

The object avoidance algorithm that ended up on the STM32 was only for objects out of beacon range. Given the fact that the targeting algorithm was scrapped, the focus was only on obstacles on the journey to a beacon. The process of dodging the obstacle using 90 degree turns was capable and worked well with stationary obstacles.

The primary issues came with the hardware and STM32 microcontroller. Once learned, the coding process for the STM32 was not too difficult but the learning process was long. The device had multiple initialization registers that were vital to input and output ports being set up. The process of learning the I2C and USART protocols took weeks. Once all ports were initialized, however, the algorithms programming took very little time to create. A major issue occurred with the motors. After repeated usage throughout the design process and testing, couple of the motors failed the night before design day. This forced us to use our remaining motors and a caster wheel to retain our ability to compete. After numerous rewritings of the code and adjustments to the design, it was found that the loss of torque of the motors was too large a deficiency to overcome.

While the motor controller performed as specified by the manufacturer, it was coded using pulse widths due to lack of USART ports. The controller work perfectly when sent the correct pulses for each side of the motor. The primary reason for not using mixed mode, a format similar to an RC car but utilizing differential drive was its odd complexity. Sending signals to the microcontroller and expecting a turn or full straight resulted in the motors acting erratically.

### Recommendations

Given the fact that the robot had a catastrophic failure, costing Capsule Corp a chance at competing, the primary recommendation involves the motors. Larger and more powerful motors are a must for those that wish to use a differential drive system. The grass proved to be such a large drain for the torque of the motors that if any fail the robot would not be able to move. Moreover, the reason for the motors failing in the first place was excess current draw from motor stalls. With the 90 degree turns the robot executed, the strain from trying to negotiate the rough terrain increased the chance of stalls drastically.

Another recommendation would be to research a more precise antenna. The design that was to be implemented was far too unstable to be used when all of the amplification and rectifying stages were applied to the antenna together. Research in this field would have been more vital than in any other part.

While the STM32 performed admirably for the task that was given, an upgrade to the STM32F4 would have been a better option. The extra USART ports would have allowed us to use serial communication with the motor controller. Being faster and having more ports would have been an added bonus.

### Reference

1. *STM32F0DISCOVERY Discovery kit for STM32F051 line - with STM32F051R8 MCU.* STMicroelectronics. Web. http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF253215

2. *Sabertooth 2X12 regenerative dual motor driver*. Dimension Engineering. Web. http://www.dimensionengineering.com/products/sabertooth2x12

3. *Lynxmotion - A4WD1 No Electronics.* Lynxmotion. Web. http://www.lynxmotion.com/c-111-a4wd1-no-electronics.aspx

4. *Triple Axis Magnetometer Breakout - HMC5883L*. Sparkfun. Web.

<https://www.sparkfun.com/products/10530>

# 5. *Ultrasonic Range Finder* - Maxbotix HRLV-EZ4. Sparkfun. Web.

# https://www.sparkfun.com/products/11309

# 6. *Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates*. Adafruit. Web.

<http://www.adafruit.com/products/746>

# 7. *Hitec 33322S HS-322HD Standard Deluxe Karbonite Gear Servo*. [Hitec RCD Inc.](http://www.amazon.com/s/ref=bl_sr_toys-and-games?ie=UTF8&field-brandtextbin=Hitec+RCD+Inc.&node=165793011) Web.

<http://www.amazon.com/Hitec-33322S-HS-322HD-Standard-Karbonite/dp/>

B0006O3XEA/ref=sr\_1\_1?ie

### Appendix

### Equations

= Latitude2 – Latitude1

= Longitude2 – Longitude1

Bearing =

### Wheel and Motor Specifications

|  |  |
| --- | --- |
| Voltage | 7.2VDC |
| RPM | 291 |
| Reduction | 30:1 |
| Torque | 11 oz-in continuous  54.3oz-in stall |
| Shaft | 6mm |
| Wheel Diameter | 123.5 mm |

### Ultrasonic sensor characteristics

|  |  |
| --- | --- |
| Operating voltage | 2.5 V – 5.5 V |
| Nominal current draw | 2.5 mA – 3.1 mA |
| Operating Temperature | -15 °C to 65 °C |
| Resolution | 1 mm |
| Distance range | 30 cm – 5 m |
| Object proximity detection range | 1 mm – 5 m |

### Purchased Items



MILESTONES

During the project development, there are some key achievements that are essential for the success of the project:

March 29, 2014

• Integrate the Sabertooth motor controller and encoders with the STM32F0 microcontroller to have the vehicle drive straight autonomously for 10ft.

April 4, 2014

• Design a simple antenna system that is able to pick up an 80 kHZ signal from a certain distance

April 16, 2014

• Interface the GPS and Magnetometer with the microcontroller to allow the vehicle to go to an inputted GPS location with the heading from the magnetometer.

April 30, 2014

• Vehicle should be able to circumvent an obstacle and calculate a new vector towards

### Obstacle Avoidance

void OADODGE(void)

{

Stop(); //Stop the car

Turn(-90); //Turn Left

GoDistance(0.5); //Go half a meter

Turn(90); //Turn Right

if(GoSonic() < 1500 && GoSonic() != 0)

// Check to see if there is an object ~2.5 feet away and avoid again

{

Turn(-90); //Turn Left

GoDistance(0.5); //Go half a meter

Turn(90); //Turn Right

}

GoDistance(0.5); //Go half a meter

}

void OASTOP(uint32\_t sonic)

{

if(sonic < 1500 && sonic != 0){

//Dodge the obstacle until obstacle is no longer detected

OADODGE();

}

GoStraight(); //Drive straight

}

}

### Magnetometer Code

double receiveHead(void)

{

I2C\_NumberOfBytesConfig(I2C1, 1);

I2C\_SlaveAddressConfig(I2C1, 0x3C); //Write Command

I2C\_MasterRequestConfig(I2C1, I2C\_Direction\_Transmitter);

I2C\_GenerateSTART(I2C1, ENABLE);

delay2(1);

I2C\_SendData(I2C1, 0x03);//(A) it initializes the mag to send data.

delay2(100);

I2C\_NumberOfBytesConfig(I2C1, 6);

I2C\_SlaveAddressConfig(I2C1, 0x3D); //Read Command

I2C\_MasterRequestConfig(sEE\_I2C, I2C\_Direction\_Receiver);

I2C\_GenerateSTART(sEE\_I2C, ENABLE);

delay2(2);

temporXH = I2C\_ReadRegister(I2C1, I2C\_Register\_RXDR); //Read from the registers

delay2(2);

temporXL = I2C\_ReadRegister(I2C1, I2C\_Register\_RXDR);

delay2(2);

temporZH = I2C\_ReadRegister(I2C1, I2C\_Register\_RXDR);

delay2(2);

temporZL = I2C\_ReadRegister(I2C1, I2C\_Register\_RXDR);

delay2(2);

temporYH = I2C\_ReadRegister(I2C1, I2C\_Register\_RXDR);

delay2(2);

temporYL = I2C\_ReadRegister(I2C1, I2C\_Register\_RXDR);

delay2(2);

I2C\_GenerateSTOP(I2C1, ENABLE);

FullHeadingX = ((temporXH << 8) | temporXL); // Concatenate the MSB and LSB

FullHeadingZ = ((temporZH << 8) | temporZL);

FullHeadingY = ((temporYH << 8) | temporYL);

head = (atan2(FullHeadingY, FullHeadingX)) \* 180 / PI; // Calculate the heading

if (head < 0) // Make the heading positive

head += 360;

return head;

}

### Go Straight

void GoStraight(void)

{

startHeading = receiveHead();

if (stopFlag == 0)

{

updateCheck = 501;

do // ramp up wheel speed

{

leftSpeed += 25;

if (leftSpeed > 8800)

leftSpeed = 8800;

TIM\_OCInitStructure.TIM\_Pulse = leftSpeed; //LEFT SIDE

TIM\_OC4Init(TIM1, &TIM\_OCInitStructure);

rightSpeed += 25;

if (rightSpeed > 8800)

rightSpeed = 8800;

TIM\_OCInitStructure.TIM\_Pulse = rightSpeed; //RIGHT SIDE

TIM\_OC3Init(TIM1, &TIM\_OCInitStructure);

} while ((leftSpeed < 8600) && (rightSpeed < 8600));

stopFlag = 1;

}

while(1) // go straight forever

{

updateCheck = 502;

if(msTicks %100 == 0)

{

diffHeading = receiveHead() - startHeading;

encoderValue1 = GetEncoder1(); // get encoder

encoderValue2 = GetEncoder2();

updateCheck = 503;

if(abs(diffHeading) < 2)

{

updateCheck = 701;

if (encoderValue2 < 155000)

{

updateCheck = 7011;

rightSpeed += 100;

if (rightSpeed > 8800)

rightSpeed = 8800;

TIM\_OCInitStructure.TIM\_Pulse = rightSpeed; //RIGHT SIDE

TIM\_OC3Init(TIM1, &TIM\_OCInitStructure);

}

if (encoderValue2 > 165000)

{

updateCheck = 7012;

rightSpeed -= 100;

if (rightSpeed < 6800)

rightSpeed = 6800;

TIM\_OCInitStructure.TIM\_Pulse = rightSpeed; //RIGHT SIDE

TIM\_OC3Init(TIM1, &TIM\_OCInitStructure);

}

if (encoderValue1 < 155000)

{

updateCheck = 7013;

leftSpeed += 100;

if (leftSpeed > 8800)

leftSpeed = 8800;

TIM\_OCInitStructure.TIM\_Pulse = leftSpeed; //LEFT SIDE

TIM\_OC4Init(TIM1, &TIM\_OCInitStructure);

}

if (encoderValue1 > 165000)

{

updateCheck = 7014;

leftSpeed -= 100;

if (leftSpeed < 6800)

leftSpeed = 6800;

TIM\_OCInitStructure.TIM\_Pulse = leftSpeed; //LEFT SIDE

TIM\_OC4Init(TIM1, &TIM\_OCInitStructure);

}

}

else if (diffHeading > 2)

{

updateCheck = 702;

leftSpeed = 7800;

TIM\_OCInitStructure.TIM\_Pulse = leftSpeed; //LEFT SIDE

TIM\_OC4Init(TIM1, &TIM\_OCInitStructure);

rightSpeed = 8800;

TIM\_OCInitStructure.TIM\_Pulse = rightSpeed; //RIGHT SIDE

TIM\_OC3Init(TIM1, &TIM\_OCInitStructure);

}

else if (diffHeading < -2)

{

updateCheck = 703;

leftSpeed = 8800;

TIM\_OCInitStructure.TIM\_Pulse = leftSpeed; //LEFT SIDE

TIM\_OC4Init(TIM1, &TIM\_OCInitStructure);

rightSpeed = 7800;

TIM\_OCInitStructure.TIM\_Pulse = rightSpeed; //RIGHT SIDE

TIM\_OC3Init(TIM1, &TIM\_OCInitStructure);

}

}

}

}

### Parsing GPS

void GetGPS(void)

{

k = 0;

while(ModeEins != 1) //Checks to see if the GPS is valid

{

//...............................v i - 1................v

// ....., x x x x . x x x x \0 N ,x x x x x .x x x x \0 W ,..............

//......^i - 1..............^insert...................^insert............

if(pie[k] == 'N' || pie[k] == 'W')//If it is the latittude, longitude

{

BackTrack = 1;

if(pie[k]== 'N')

pie[k] = 'n';

k--;

pie[k] = '0';

}

if(BackTrack == 0)

{

k++;

}

else

{

k--;

if(pie[k-1] ==',')

{

if(pie[k-2] == 'n')

{

Deg[0] = pie[k]; //Get the Deg of the longitude

Deg[1] = pie[k + 1];

Deg[2] = pie[k + 2];

Deg[3] = '\0';

LngDeg = atof(Deg); //Convert the string to a float

Min[0] = pie[k + 3]; //Get the remmaing bytes, minutes

Min[1] = pie[k + 4];

Min[2] = pie[k + 5];

Min[3] = pie[k + 6];

Min[4] = pie[k + 7];

Min[5] = pie[k + 8];

Min[6] = pie[k + 9];

Min[7] = '\0';

LngMin = atof(Min); //Convert the string to a float

LngMin = LngMin / 60 \* 100; //Convert the DM to DD

LngDeg = LngDeg + LngMin/100; //Concatenate the deg and decimal

BackTrack = 0;

ModeEins = 1;

}

else if(pie[k] == '3')

{

Deg[0] = '0';

Deg[1] = pie[k];//Get the degree of the latitude

Deg[2] = pie[k + 1];

Deg[3] = '\0';

LatDeg = atof(Deg);//Convert the string to float

temp = LatDeg;

Min[0] = pie[k + 2];//Get the remaing bytes, minutes

Min[1] = pie[k + 3];

Min[2] = pie[k + 4];

Min[3] = pie[k + 5];

Min[4] = pie[k + 6];

Min[5] = pie[k + 7];

Min[6] = pie[k + 8];

Min[7] = '\0';

LatMin = atof(Min); //Convert the string to float

LatMin = LatMin / 60 \* 100;

LatDeg = LatDeg + LatMin/100; //Concatenate the deg and decimal

BackTrack = 0;

}

}

}

}

}