March 2018

Sensors for Autonomous Systems

Ching-Hsiang Chen
Worcester Polytechnic Institute

Tin T. Bui
Worcester Polytechnic Institute

Zachary Joseph Peasha
Worcester Polytechnic Institute

Sensors for Autonomous Systems

SJB-MQP-2A17

Authors:
Ching-Hsiang Chen
Tin Bui
Zachary Peasha

Advisor:
Professor Stephen Bitar

Sponsor:
Allegro Microsystems LLC

Date of Submission:
March 2, 2018

Acknowledgements

We acknowledge with gratitude to Professor Stephen Bitar for his continuous support and supervision in every step of this project. Without his help, the project would not have taken off in the first place.

We also acknowledge Professor John A. McNeill for his selecting us to be part of the NECAMSID Lab Major Qualifying Projects.

Abstract

This Major Qualifying Project seeks to develop a functional scale model of a fully autonomous ferry. Using sensory systems commonly found in autonomous vehicles, we developed an autonomous surface vehicle capable of avoiding obstacles while traveling between docks as well as docking without any human input. We believe that the development of a fully autonomous ferry could help the marine industry as a whole move towards a safer and more autonomous future.

Executive Summary

Problem

Labor wages have been rising all over the world. In the United States, according to the report from Reuters Business News, the U.S labor department indicates that the wage has lifted on the year-on-year rate of increase to 2.6 percent.[1] This is the largest increase since the first quarter of 2015, from 2.5 percent in the third quarter. Raising wages to retain and attract workers and offering other benefit has become a bigger issue for the employers.

[1] U.S. labor costs increase solidly in the fourth quarter.

Applying autonomous technology has become a trend in twenty-one centuries. However, debates about machine taking jobs away from human has become a big concern. Thus, it has become a challenge for the society to find the balance of how much and what are the duties that should be replaced by machines and what should be remain to be executed by humans.

Goals and Objective

In this project, our purpose of applying autonomous technology was not just to save wages, but to replace dull and repeated duties with machines and computers, leaving the more interesting, critical and meaningful operation to the labors. In this project, we focused on applying autonomous technologies to public ferry that takes passengers from one fixed location to another fixed location.

Our two objectives were to allow the boat to transport autonomously from one location to another and perform autonomous docking once it reaches the dock. Our goal was to make the captain and other crew members' job more than just taking passengers from one location to another. Perhaps while working, they could have more attention to interact with the passengers, could anticipate the potential accident before it occurs and maybe would not be exhausted by the dull duties, which may cause them to be unhappy with their job or lack of energy to figure out other important matters in their individual life.

Background

In the United States, public ferry transportation system like the NYC public ferry that operates around Manhattan Island and other nearby area could possibly be benefited by operating autonomously. These ferries operate approximately 16 hours per day, 7 days a week and arrives every 20 minutes[2]. Driving public ferry is a professional job, so employer cannot just

[2] *NYC Ferry Homepage*

pay these employee minimum wages. However, steering the ferry on the open sea does not require as much attention as driving a public bus in a busy downtown where a bus driver has to constantly dodge and break hard to prevent accidents. Since these ferries are moving only from one fixed location to another and the path are pretty much preserve only for the ferry, it would be a good idea to apply autonomous technology into operating these ferries.

Another example is the airplane. Modern day airplanes are all equipped with autopilot system. The autopilot system enables an aircraft to fly on its own after take-off. This allows pilots to take some rest during the long flight and only requires pilots to take over when severe turbulence happens or the aircraft is about to land.

The similarity between public ferry and airplanes is that they both operate in a very large open space and it does not likely to require constant dodging and breaking during moving on its path.

If autopilot system has been operating in airplane for more than a decade, perhaps it's time for the ferry to apply autonomous system.

Methodologies

To travel from one location to another, we applied sensors including GPS module, digital compass, motor control module and ultrasonic sensors.

We used the GPS module to fetch the ferry's current GPS coordinate and then compare to the destination's coordinate. By applying the "haversine" formula, we calculated the shortest distance of two coordinates over the earth's surface and thus determined which direction to approach and the distance between two coordinates.

The digital compass, which included a three-axis magnetometer sensor and a three-axis accelerometer sensor, allowed us to know the ferry's current heading direction. Once we knew the target's heading direction and the ferry's current heading direction, we could then know if the

ferry was moving toward the correct direction. If not, we could apply the motor control modules to control the motors to move the ferry to point to the target's direction.

If there were any obstacles while the ferry was approaching its target location, the ultrasonic sensors on the front of the boat would detect the position of the obstacle and determine whether the boat should stop or which direction to dodge.

To dock the boat, we installed five infrared sensors on the dock to locate the real-time location of the boat. Once the boat was close to the dock, each infrared sensor would start measuring the distance between the boat and each sensor. These measured data were then sent into our algorithm to determine the current location of the boat relatively to the dock. The microcontroller then sent commands through Bluetooth to the boat and guided the boat to move to the desired position.

However, if an emergency incident were to have happen, the boat could have been switched back to manual mode to allow human to take over the control. In our project, if we had needed to take over the control from the autonomous system, the Bluetooth module on the dock would have sent a shutdown command to the boat. After that, the boat would have been control by its original RC remote controller.

Recommendations

Our autonomous system can be applied to any type of public ferry that takes passengers from one fixed location to another fixed location: for example, NYC Ferry and the public ferry that operates between Cape Cod to Martha's Vineyard. This could significantly save the cost for the employer, and perhaps we can allow the crew members to be less stressed by the repeated or dull duties and focus more on the critical and more interesting part of the work.

Due to the limited budget, time and skills, we are only able to choose sensors that are cheap and easy to interface with. We used infrared sensors. However, for future projects that aim to perform testing this system on a full-scale ferry, we recommend that infrared sensor be

replaced by laser or LIDAR sensors to perform a larger range and more accurate measurements that won't be affected by other factors easily.

Table of Contents

Table of Figures

Table of Equations

1. Introduction

When the topic of autonomous vehicles is brought up, most people picture sitting in a car that chauffeurs you to any location entered in a GPS. While it is expected this will be the future of the automotive industry, there are many roadblocks in making this a reality. As with most new technologies, the change is implemented slowly as additional safety or comfort systems are added to vehicles. For example, many modern automobiles have optional lane assist, which adjusts the vehicle's steering angle to maintain position in the lane. Lane assist demonstrates merely a partially autonomous system.

While much of the focus on autonomous vehicles is on automobiles, there is another area where we may see fully autonomous systems sooner. In the marine industry, many companies have started work toward developing fully autonomous ships. These ships would be able to navigate from one location to another, while avoiding objects as well as other ships in the ocean. Much research has gone into the development of unmanned cargo ships, but it seems that may still be away off. Surprisingly, the autonomous ferry may be one of the first fully autonomous ships to set sail.

Ferries are ships designed to transport people, and sometimes vehicles, between locations. It would be ideal to implement as autonomous transportation since they are traveling short distances as well as docking in the same locations. This means if there were any malfunctions in the system, another boat could reach the ferry quickly and manually control it back to the dock. The ferry will travel between the same docks and dock in the same position to load and unload passengers.

While the autonomous ferry may not be as exciting for companies as the idea of an autonomous cargo ship, it may be a strong argument in the implementation of autonomous ships. Like the automotive industry, the integration of autonomous marine vehicles will happen

gradually. If the future of transportation, either by land, air, or water, is by fully autonomous vehicles, the public will need to begin to trust these systems. The autonomous ferry would not only provide a service to build that trust with the public, but also provide the platform to prove the safety and efficiency of this emerging technology.

For this Major Qualifying Project (MQP), a scale ship is used. This model ship is able to travel across water between one docking position to another. While traveling through the water, it could adjust its heading and if necessary, come to a complete stop to avoid collisions. Once the model ship has reached its destination, it began to slow to a docking speed and position itself to allow “passengers” to board. Once the boarding process is completed the boat will receive an input telling it to travel to the next docking location.

All of this was not a simple task. The major hurdles for this project was sensing the environment around the ship and knowing its location. The ship's heading as well as speed will be adjusted based on the position and destination information given to the ship. While following the direction given to the ship, it will also need to adjust to the environment. If the ship detects the potential for a collision, it would adjust speed and heading to avoid the object without straying too far from the original path.

2. Background

2.1 Autonomous Surface Vehicles

Safety is the most important factor when considering the use of Autonomous Surface Vehicles (ASVs). While the efficiency of ASVs provides an operating cost reduction as well as removing the crew from the ships and danger while at sea, the ASV still has to interact with its environment. While it seems like removing the crew would remove any human error from the

navigation of the ship, this is not true. The human error now lies in the software.[3] Those algorithms developed to control the ship can and will have some degree of error. Anticipating every single potential event at sea is almost impossible, and even the best algorithm may create the wrong response to a certain event. This doesn't mean that a manned ship wouldn't make the same mistake. When designing an ASV, the ship must be considered to be at least as safe, if not safer than a manned ship.

Currently, autonomous cargo ships seem to be getting the most attention. Having unmanned cargo ships would allow more room for cargo, since there would be no need for room for the crew. Any human error in navigating could be removed and the ship could be monitored safely from shore. While these ships are currently expensive, they are expected to cut operating costs by 90% annually. It seems that the technology for these ships is ready to be implemented, but the International Maritime Organization doesn't expect legislation governing crewless ships to be in place before 2020.[4] However, a Norwegian company that has plans on implementing the autonomous ship in stages. First the ship will have a manned crew and operate similarly to a conventional cargo ship. Then the ship will be crewless, but be remotely operated from a location on shore. Lastly, once legislation is passed, the ship will be able to operate fully autonomously.

Another area being researched for autonomous watercraft is autonomous sailboats. These sailboats would be ideal for long term usage at sea. Since they only need to adjust the angle of the sail and rudder to control movement, the battery consumption is less than that of a motorized boat. Boats with conventional gas or diesel engines would require periodic refueling. These sailboats could rely on renewable energy sources, such as wind or solar power. Like most autonomous vehicles, these ships can navigate to locations and adjust heading based off current position. While moving to a predetermined location, they need to sense their

[3]The Human Element and Autonomous Ships
[4]Norway Takes Lead in Race to Build Autonomous Cargo Ships

environment for potential collisions, and adjust heading when necessary. These sailboats would be ideal for conducting marine research that doesn't require a crew to stay on a ship for extended periods of time.[5]

Autonomous ships even have an application in the Navy. Currently Rolls-Royce is developing an autonomous ship for use by the Navy.[6] The purpose of this ship would be to detect mines, perform patrols, and surveillance missions. This ship would be smaller than typical naval ships, but would be designed to work alongside them. This autonomous ship has the benefit of making Naval operations safer. Patrol would be safer since there are no humans on this ship. By alerting the ships when mines are detected, the autonomous ship makes the manned ships safer. The main concern with the autonomous ship is the reliability. If there is a malfunction, there will be no crew aboard to diagnose and repair the ship. This means any autonomous ship must be made as reliable as possible.

2.2 Autonomous Sensing Technology

In order to design an autonomous vehicle, we needed to select sensors that would work well in our operating conditions. The sensing technology for autonomous operation can be broken down into two categories: sensing location and sensing environment. In order to pick a method to accomplish each of these tasks, multiple autonomous systems were researched. These included robots, automobiles, boats, and drones. Since the application was not important we were able to focus on what was used to sense the environment, location, and how well it met the system's needs. The research focused on the sensors commonly found in autonomous systems.

[5]Toward an Autonomous Sailing Boat
[6]Rolls-Royce Planning Autonomous Naval Ship for Patrol, Surveillance and Mine Detection

2.2.1 Infrared Sensors

Infrared distance measuring sensors use a beam of infrared light to measure the distance from the sensor to the object the light reflects off of. The distance between the sensor and the object can be detected using different methods. The sensor can measure the intensity of the light that reflected off the object to detect the distance. This method is prone to error in measurements due to the changing amounts of IR light in the operating environment. A more effective technique it to measure the change in position of the returning beam.

The manufacturer “Sharp” produces one of the most commonly used IR sensors in DIY applications. The sensor is equipped with lens which emits narrow light beam. After reflecting from the object, the beam will be directed through the second lens on a position-sensitive photodetector (PSD). The conductivity of this PSD depends on the position where the beam falls. The conductivity then converted into an analog voltage. The result is a changing analog voltage based on changing distance measurements. The route of beams reflecting from different distance presented as below in figure 1:

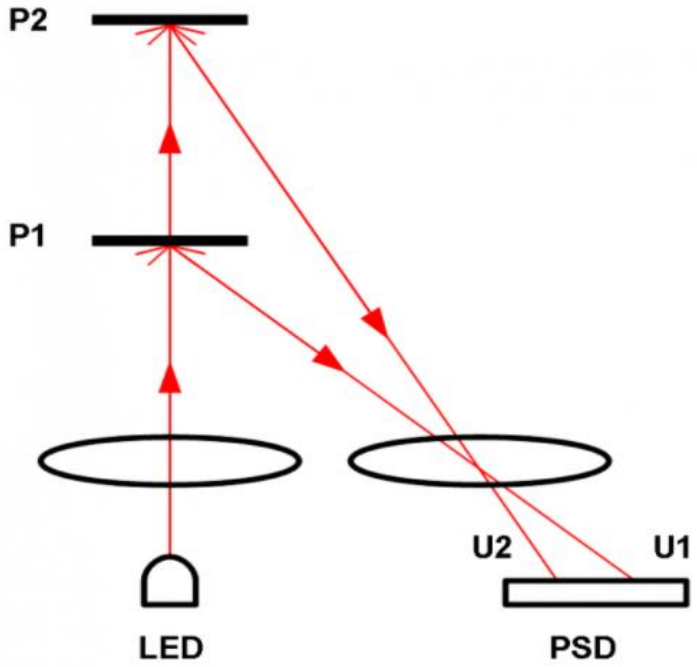


Figure 1: IR Sensor Distance Measurement

The output voltage of the distance sensor is inversely proportional to the distance measured. This means that when the distance increases, the output voltage decreases. All sensors have the specific measuring range where the measured results are credible, and this range depends on the type of the sensor. Maximum measured distance is restricted by two aspects: the amount of the reflected light decreases with increased distance, and the inability of the PSD to register a small change in location of the reflected ray. When measuring an object that is too far away, the output remains approximately the same as if it were measuring the objects at the maximum distance. Minimum distance is restricted due to the peculiarity of the sensor, meaning the output starts to decrease sharply at the minimum distance specification. This problem can be avoided by ensuring the object measured stays within the manufacturer's specified operating range. The diagram of the voltage versus distance of the sensor make clearly the previous point:

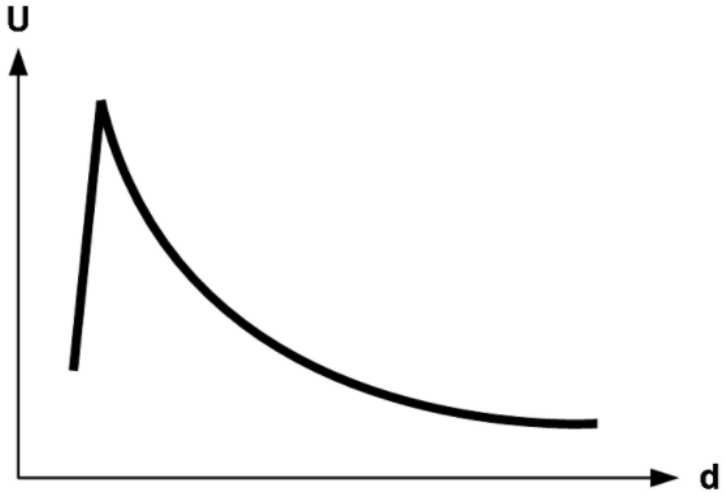


Figure 2: IR Sensor Voltage-Distance Curve

The graph shows the relationship between its output voltage and measured distance. This graph is not linear; however, the graph of inverse values of output voltage and distance can be approximated using a linear relationship. A linear approximation of the inverse values can be seen in figure 3.

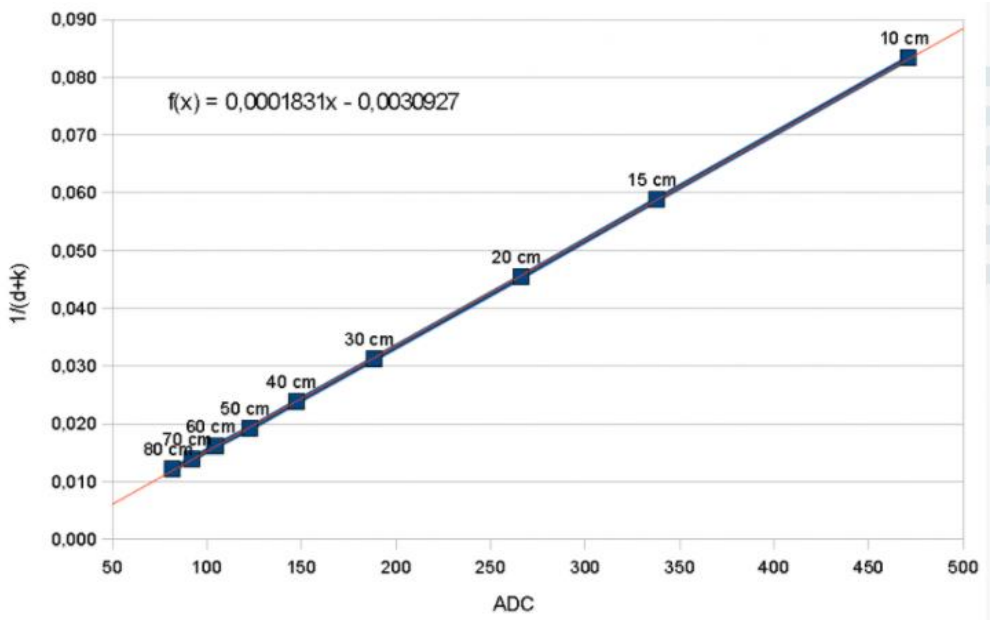


Figure 3: Linearizing ADC Value and Distance

As seen on the graph, the trend line overlaps quite precisely with the points of the graph. The overlapping is achieved by using the help of the correction constant. This correction constant was discovered by using the trial-and-error method. The graph is very similar to the linear trend line and a generalization can be made and say that the relation between the voltage and measured distance can be seen as following:

$$\frac{1}{d+k} = \mathrm{a} * \mathrm{ADC} + \mathrm{b}$$

Equation 1

d is the distance

k is the correction constant

ADC is digitized value of voltage

a is the linear member (determined by the trend line equation)

b is the free member (determined by the trend line equation)

Distance can be expressed from the formula:

$$d = \frac{\frac{1}{a}}{ADC + \frac{b}{a}} - k$$

Equation 2

With this formula, the measured distance of the objects can thus be calculated.[7] An example of a commercially available IR distance sensor can be seen in figure 4.

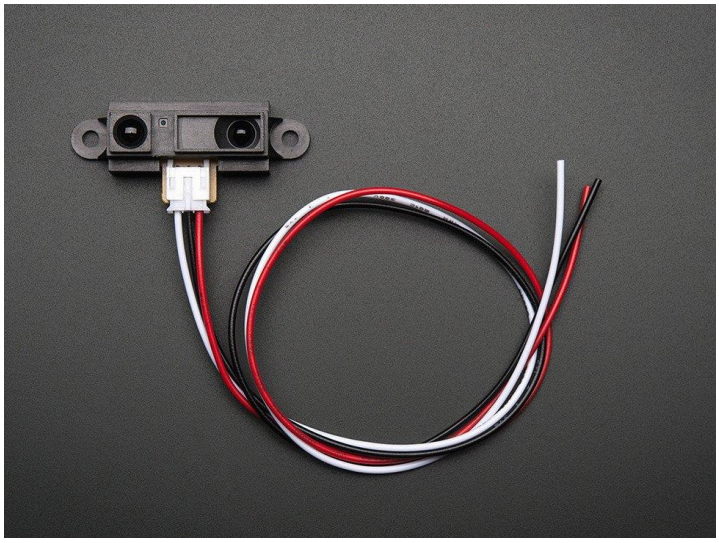


Figure 4: IR Distance Sensor (GP2Y0A21YK0F)

While infrared sensors are commonly used in distance measurement for autonomous land vehicles, few examples have been found on ASVs. One reason might be the interference effected by the external IR sources. In outdoor applications, the IR waves from the sun can cause fluctuations, and thus effect the accuracy of the measurement. Many IR sensors have a typical operating range of 1 meter. For a land vehicle, even moving at moderate speeds, the trajectory of the vehicle can be adjusted in time to avoid collision. However, ASVs are less agile. Even with only moving at moderate speeds, which moves between 46.3 km/h to 64.82km/h, a ship may still cause a collision because of insufficient reaction time and the inertia propelling the

[7]Infrared Distance Sensor

ship forward. While this sensor may not be ideal for avoiding collisions, it may still be useful for measuring the distance between the ASV and the dock.

2.2.2 Accelerometers

An accelerometer is a sensor that measures a force resulting on the sensor due to an acceleration. Accelerometers typically use either two capacitive plates or piezoelectric materials to measure changes in forces. For the capacitive plate system, the plate moves when a force is applied. This results in a change in capacitance, which can then be quantified into a force measurement. Piezoelectric crystals would output a voltage proportional to an applied force. This voltage can then be used to quantify the amount of force measured by the sensor. Figure 5[8] shows an image of a 3-axis accelerometer IC.

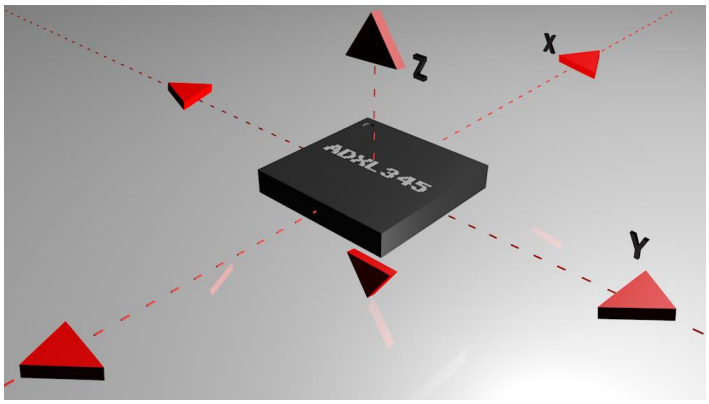


Figure 5: Three - Axis Accelerometer

Accelerometers can be used when detecting velocity or dead reckoning on an ASV. Dead reckoning is the process of estimating your current position based of your last position and velocity. Accelerometers usually include a gyro sensor inside one integrated module. The problem with using this sensor to measure position is errors would accumulate. Each time the current position is calculated, the error in that position estimation is added to the error in the previous position. Another issue on the ASVs is the noise effecting the gyro sensor. On the

[8]Accelerometer Basics

water the waves sway the boat creating “noise” that the gyro sensor picks up. This noise causes errors in the sensors readings that will vary based on wave intensity and ship size. The error of these sensors positions and velocity measurements may seem like a concern, but these errors would be reduced when paired with GPS.

2.2.3 Global Positioning System

The global positioning system (GPS) is the system that provides geolocation and the time information to a receiver anywhere on the earth. This requires an unobstructed signal from four or more GPS satellites (three for location and one for time). The GPS receiver does not transmit any data, it operates independently like any telephonic or internet reception. The GPS module is based on the time and the known position of specialized GPS satellites. These satellites carry atomic clocks that can synchronize with the ground. Any drift from the true time maintained on the ground is corrected daily. The GPS satellites continuously transmit the data of their current time and position. A GPS receiver will monitor multiples satellites and solve equations to determine the precise position of the receiver and its deviation from the true time. A commercial GPS receiver can be seen in figure 6.

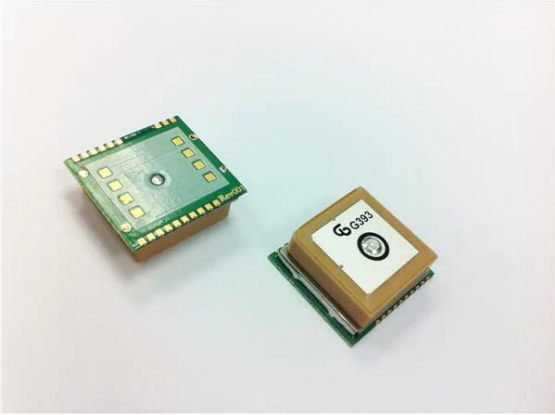


Figure 6: GPS Receiver

GPS is the most commonly used navigation aid in marine applications. A commercial system has been developed for port operation which uses GPS as the sole navigation sensor.

The boat, or ferry, position can be determined if at least four satellites are in direct line of sight of the receiving antenna. 0.20-0.02 meter accuracy can be obtained with five or six satellites transmitting data to the receiver.

Instead of the direct path from satellites to the boat or ferry, the signal could bounce of some surface and a delay will incurred in that particular signal path. This may lead to errors in the estimated object's position, which costs several meters of inaccuracy. With an error range of under 5 meters, the need for GPS is necessary in this application. The ability of GPS will allow the boat to follow a predefined path accurately. The position estimate is usually only available at low frequency from most commercial GPS systems with the maximum update rate at 4 Hz. The solution to the low update rate is to fuse the GPS data with a sensor suited to measure at higher frequency. The GPS/compass was utilized to measure the boat moving along different locations.

2.2.4 Millimeter Wave Radar

Autonomous systems are beginning to utilize millimeter wave radar, which is a special class of radar technology that uses short wavelength electromagnetic waves. Millimeter Wave radar transmits signals to the object and captures the reflected signals. Millimeter Wave systems consist of transmit (TX) and receive (RX) radio frequency components; analog components such as clocking; and digital components such as ADC, microcontroller (MCUs) and digital signal processors (DSPs). The synthesizer generates the signal called a chirp and then transmits it with the TX antenna. The reflected signal will be received by the RX antenna and then mixed with the chirp to produce the intermediate frequency signal. The differences in time delay and phase shift will determine the range, velocity, and angle of the objects. An advantage of this system is it has a large bandwidth to distinguish multiple nearby objects. An example of a mmWave sensor is Texas Instruments mmWave Sensor, as can be seen in figure 7.

Figure 7: Texas Instruments mmWave Radar Sensors

The ability of mmWave sensors to perform in all conditions could contribute to properly aligning the prototype boat to the dock. mmWave radar has low absolute accuracy, but it is less sensitive to extreme operating conditions. This is because the 4mm wavelength is less affected by small atmospheric particles and it exhibits lower absorption by water in the atmosphere due to a transmission window in the electromagnetic spectrum at 77 GHz.

The mmWave band is generally considered to span from 40 GHz to 300 GHz. The principal attributes of the system operating at these frequencies include small antenna size and high bandwidth. The short wavelengths are able to narrow beam widths, which are highly directional, with small components. This is important for the vehicle applications as the radar unit must be prominently mounted, often in minimal space. The high bandwidth means high resolutions are achievable. Many objects have features of comparable dimension to the mmWave signal. They often return more signal power in the millimeter rather than the microwave band, facilitating more accurate measurements of the environment. The range of the detection can go up to hundreds of meter and can respond to the object's speed up to 300km/h. Due to these advanced features of the mmWave, many applications use them as integral components.

The mmWave uses the frequency modulated carrier wave (FMCW) principle to measure distance. Alternative methods of measuring distance are based on a time of flight principle,

shown in equation 3, which measure the time between transmission and reception of a pulse modulated signal.

$$Distance = \frac{T * c}{2}$$

C is speed of radiation of electromagnetic waves

T is the time between transmitting and receiving the signal

Equation 3

This gives a single distance measurement to the first reflector that satisfies threshold detection criteria. In a FMCW, the transmitted and the received signal are combined in a microwave mixer to generate the intermediate frequency Δf which can be determined using equation 4:

$$\Delta f = \frac{2 * Range * fs}{cTs}$$

Ts is the sweep time

Fs is the sweep frequency

Equation 4

Thus, the range could be found according to equation 5 shown below:

$$Range = \frac{c * Ts * \Delta f}{2fs}$$

Equation 5

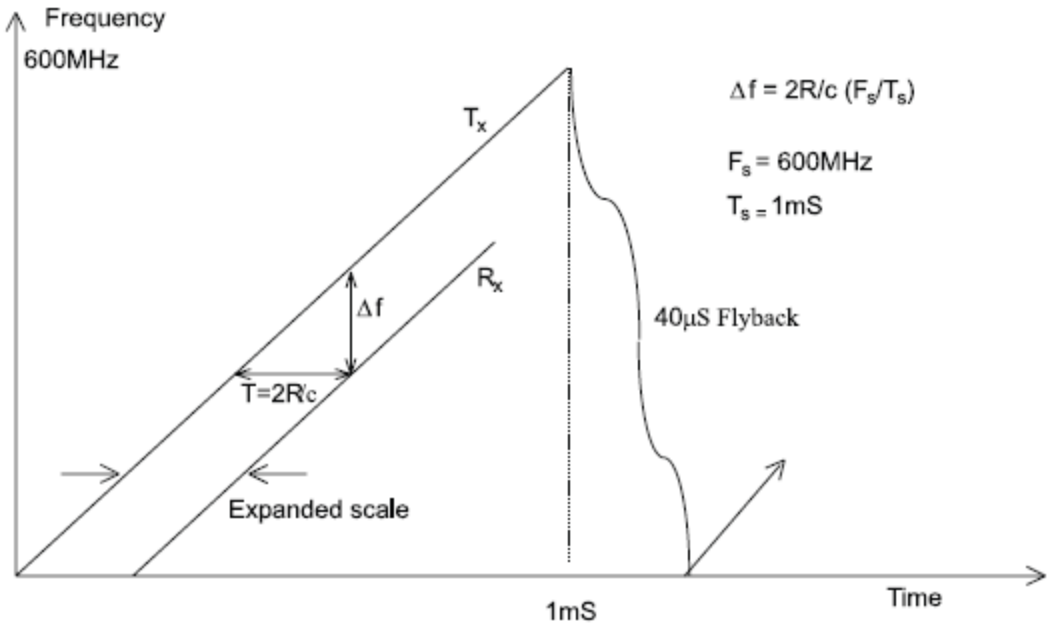


Figure 8: Radar Chirp TX and RX Over 600 MHz in 1ms

This is an example of how the distance and the range of the object will be measured in mmWave. Due to the facts that the price of mmWave are decreasing, and more advanced features will be integrated on mmWave, it will become one of the best sensors using for navigation vehicles.

2.2.5 Laser Range Sensor

The range laser sensor is a sensor, such as LIDAR, which uses a laser beam to determine the distance of the object. It can count, trigger, scan, map, profile and guide as well as verify levels, proximities and distance to the objects. The range laser sensor operates on the time of flight principle by sending the laser pulse in a narrow beam towards the objects and measuring the time taken by the pulse to be reflected off the target.

Figure 9: Laser Rangefinder (LIDAR)

Lidar consists of transmitter, which illuminates the target with the laser beam, and a receiver capable of detecting the components of light which is essentially coaxial with the transmitted beam. The receiver sensor calculates a distance based on the time needed for the light to reach the target and return. A mechanical mechanism with the mirror sweeps the light beam to cover the required area in a plane or even in three dimensions, using a rotating nodding mirror.

The sensor measures the phase shift between the transmitted and reflected signals. The wavelength of the modulating signal obeys the equation:

$$c = f * T$$

Equation 6

Where c is the speed of light and f is the modulating frequency and T is the known modulating wavelength. The picture below shows how the technique can be used to measure the distance.

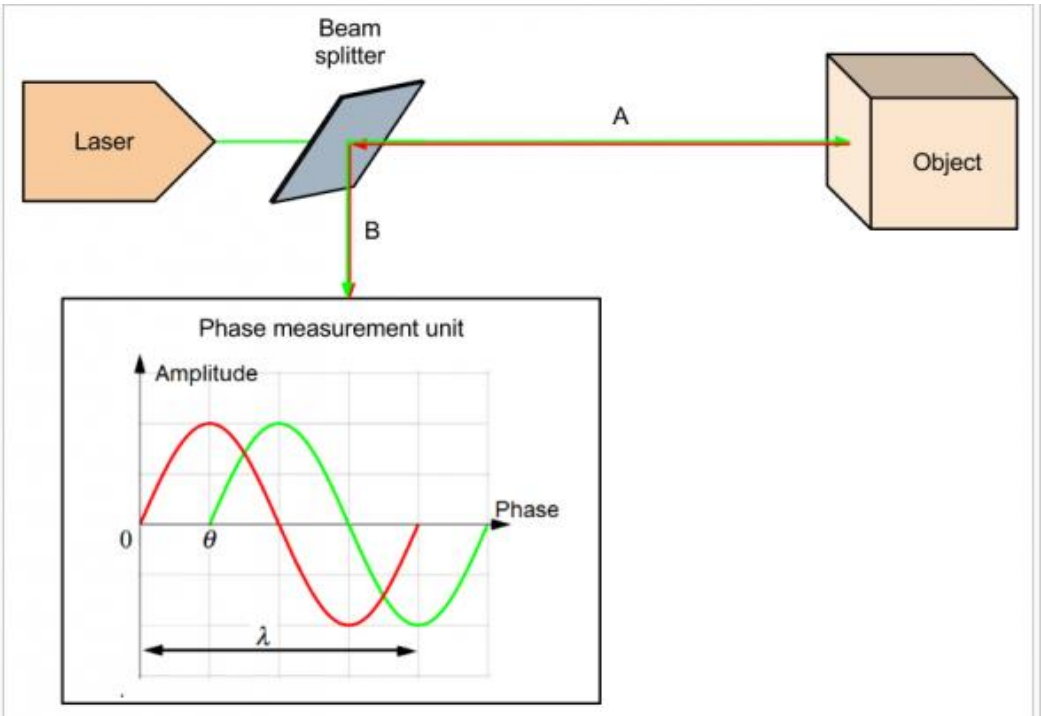


Figure 10: Phase Shift Measurements

It can be shown that the range is inversely proportional to the square of the received signal amplitude, directly affecting the sensor's accuracy.

Despite the laser beam being narrow, it will eventually spread over the long distances due to the divergence of the laser beam caused by the presence of heat bubbles in the air. These bubbles act as lenses ranging in size from microscopic to roughly half the height of the laser beam's path above the earth. These atmospheric distortions coupled with the divergence of the laser itself and with transverse winds that serve to push the atmosphere heat bubbles laterally may combine to make it difficult to get an accurate reading of distance of an object. Regardless to this disadvantage feature of laser sensor, it still is one of the most attractive sensors for boat or ferry navigation due to their accuracy and low cost. Most laser sensors provide range and information with sub degree resolution and the accuracies in the order of 1-10 centimeter in the 10-50-meter ranges. Nowadays, most laser sensors are non-contact, allowing the user to measure the distance to the target or material without the need of reflective

material. The laser sensor can be aligned to the target from any distance or angle within its range. There are so many advantage features of the laser sensor that makes it suitable for the docking boat system including fast data rate, small beam footprint, and the ability to scan multiple target detection.

2.2.6 Ultrasonic Sensor

Since the boat or ferry must detect the objects around its path to avoid collision, the use of the ultrasonic sensors would be effective at sensing the environment. The ultrasonic sensor has two membranes. One membrane produces sound, another catches the reflected echo. Basically, they act as a speaker and microphone. The sound generator generates short (the length is a couple of periods) ultrasonic impulses and triggers the timer. Second membrane registers the arrival of the sound impulse and stop the timer. From the time measured it is possible to calculate the distance traveled by sound. The distance to the object is half of the distance traveled by the sound wave.

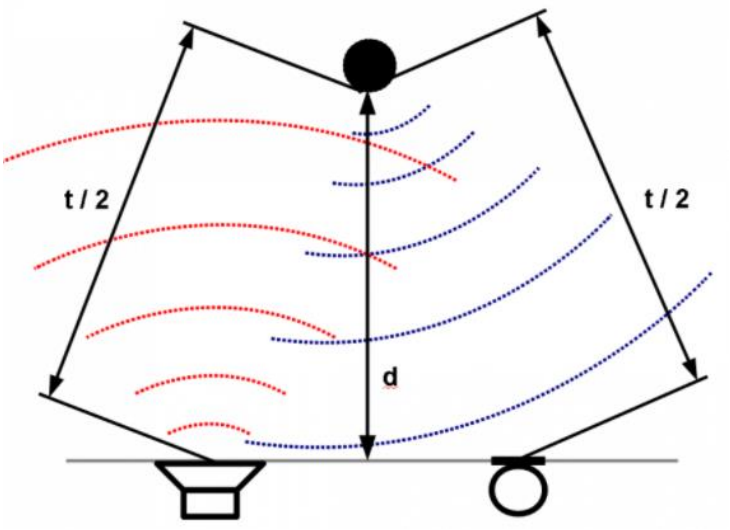


Figure 11: Principle of the Ultrasonic Distance Sensor

The ultrasonic sensors have many applications in everyday life. They are used to replace measuring tapes in construction sites. Some cars are equipped with ultrasonic parking sensors to alert the driver when they are approaching an object. Besides measuring distance, they can just register the presence of the object within the measuring range, for example in danger zones of working machines ultrasonic sensor can detect a person inside the danger zone and bring the machine to a stop. If the transmitter and receiver are separated, the flowing speed of the substance between them can be measured, because the sound wave travels slower upstream and vice versa.

Figure 12: Ultrasonic Sensor

Ultrasonic sensors convert electrical energy to sound to send the pulse, then the sound received back is converted back to electricity, which is what the brick (the device used to read the data from ultrasonic sensor) can understand. Ultrasonic sensors can measure the objects from small to large distances with the range is between 0.02 meters to 4 meters, which the precision in the order of 3 cm. The sensor works well and gives good readings in sensing large-sized objects with hard surfaces. Sometimes, the reflection from soft fabrics, curved objects or very small objects can be a challenge for the sensor to response the accurate readings.

In this project, the sensor SRF04/SRF05 was employed. This sensor does not give directional information about the distance. Besides the power supply pins, the sensor has also a

triggering pin and an echo pin. One difference of the SRF04 and SRF05 is that it is possible to use the single physical connector pins for the trigger and echo signals. This allows the sensor to use the standard three wire connector (power, ground and signal). When triggering pin is set high, the sensor generates a 40 KHz ultrasonic wave, which has a period of eight seconds. At that moment, the echo pin becomes high and remains high until the reflected sound is reached back to the sensor. So the echo signal reflects basically the time during which the sound reaches to the object and come back from the object. By measuring that time and multiplying it by the speed of sound and then dividing it by two the distance from the object can be calculated.

Besides some dominant features of the ultrasonic sensor using for the detection the objects, there are still remaining some disadvantages. The ultrasonic sensor can be sensitive to other ultrasonic sensor and acoustic noise in the area. If the readings are jumping down to 0 inches even though the objects are placed few inches away from the sensor, the signal measured could be from interference caused by other ultrasonic sensors and the environment. If more ultrasonic sensors need to be used for the desired purpose, only one sensor should be active at one time.

3. Problem Statement

Labor wages rise annually by 2.6 percent, in United States. With decreasing profits, employers are all seeking for solution to minimize the quantity of employees, while maximizing the effect that they can contribute. In some industries, employers are seeking autonomous or automation technologies to replace repetitive, or dangerous, tasks with machines and let labors focus on other more critical matters.

According to the 2016 accident statistics of the recreational boating statistics from the US Coast Guard, there are 144 reported accidents while docking or undocking, 5 deaths

reported and 30 Injuries reported.[9] Lack of experiences and unclear vision are two of the most common causes. Parking a car is easy because drivers can easily see their surroundings with the side mirror, rear mirror or through the windows. However, docking a boat requires more than that. Larger size boats or yachts are commonly equipped when cameras to help the captain see their surroundings. For small boat, the captain often has to rely on experiences and their crew who can help guiding on the dock.

According to the 2016 accident statistics of the recreational boating statistics from the US Coast Guard, there are 2565 reported accidents, 207 deaths and 1554 injuries while cruising. Many of these happen because the operator could not see an obstacle. Some of these happen because they are not paying attention while cruising and some of these happen because they couldn't see what is underneath the water of the boat.

Many of these accidents could be avoid if they had sensors like object avoidance sensors or obstacle distance detecting sensors installed. These use to be common accidents for cars in public roads, however, many automobile manufactures like Volvo, Tesla and Mercedes-Benz are equipping their new released vehicles with sensors like object avoidance sensors, which will stop the vehicle when it approaches too close to an obstacle in front, parking camera, autopilot parking, and so on. These technologies have started to make impact on the road and have urged more automobile company to equip these sensor technologies into their latest product to prevent accidents.

This project tested the functionality of an autonomous system in a marine application. The platform of this project is a ferry, which would most likely be the first boat to be integrated with autonomous control. Successful implementation in the ferry could help push for autonomous systems to be placed in more watercraft for commercial and recreational purposes.

[9] *2016 Recreational Boating Statistics*

This could help lessen the accidents in boats used for recreational purposes as well as decreasing cost of commercial shipping applications.

4. Methodology

4.1 System Functionality

The basic functionality requirement of the system was to navigate between two locations. The model ship needed to know its current position, next waypoint, and heading. To accomplish this a GPS system paired with a compass was used. The GPS provided information about current position and the next waypoint to reach. The compass provided data giving the current heading of the ship. The expected heading given by the GPS was compared to the actual heading given by the compass. Any difference in these values will show an error in the direction the ship is traveling. The ship then adjusts the heading to minimize the difference in the values to maintain course.

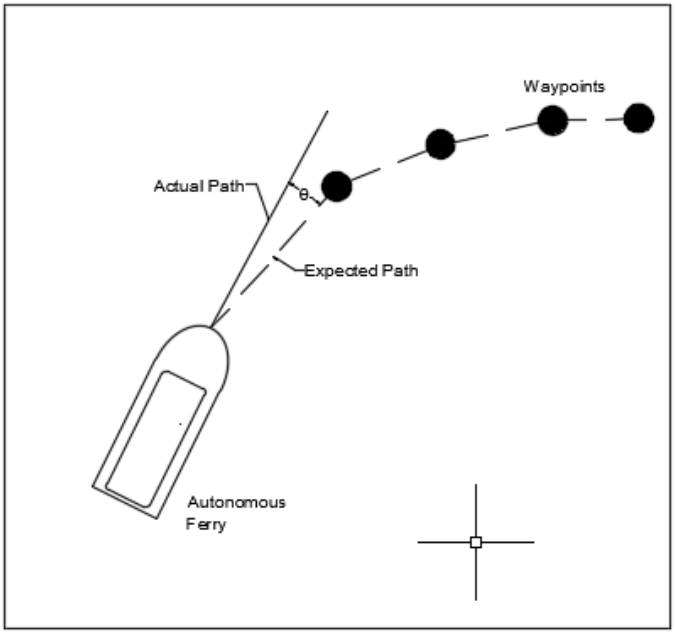


Figure 13: Autonomous Navigation Using Waypoints

The amount of compensation needed to be adjusted during the testing phase of this project. This allowed the compensation to be “dialed in” to prevent erratic steering. If the boat turned too far it would continuously steer left and right. The average direction would be correct, but the steering would be erratic. If the boat doesn’t compensate enough it will drift off track. The solution to this was to take frequent measurements and adjust whenever possible. This prevented the boat from getting far off the path between measurements. Since the GPS was able to give position related data every second, the ship was able to adjust its heading every second.

In order to detect potential collisions, an ultrasonic sensor was needed on each side of the boat. Unfortunately, the time constraints prevented the integration of this system in the final prototype. The multiple sensors would have helped detect objects in the boat’s path, objects that may come into the path, and objects that may collide with the boat. When an object is detected outside a specified range of the sensors the boat would continue along its path as normal. If that object came within the range the boat would decelerate to a “no wake” speed. This would allow for safer maneuvering of the boat until the object was outside of this range. The object would be necessary for a truly autonomous system.

One issue faced was how to approach the problem of avoiding an object in the water. There can be almost an infinite number of different scenarios which the boat will have to know to react to. Depending on where the object is and where the object is heading could change how the autonomous ship should avoid collision. The most likely scenario for collision would be the autonomous boat traveling toward an object. If this happens the autonomous boat will steer right, as with typical right of way for boating[10]. If the object is a ship it’s customary to turn enough to show intention to steer right away from the ship. Since we may not be able to distinguish what an object is, whether ship or static object, we assumed the safest maneuver is

[10]Boating Rules

to steer right. One issue was if the autonomous boat was crossing paths with another boat. Typically the boat on the right has the right of way. The concern lies within the other ship not yielding to the autonomous ship. To maintain safety the autonomous ship should slow to it's no wake speed allowing for the ship to pass. Once the other ship was out of the possible collision zone, the autonomous boat should return to cruising speed.

Safely docking the autonomous boat is complex task. The navigation system needed to be accurate enough to get close to the dock so the dock is able to sense the position of the boat. The ultrasonic sensors on the autonomous boat help avoid collision, but could not distinguish a dock from another boat. This is why the ferries dock provides the instructions for the ferry to dock. Once the ferry is near the dock it connects wirelessly to a Bluetooth module on the dock. The module receives data from an array of IR sensors placed on the dock. The sensors measure the distance from the dock to the boat as well as the angle of the boat relative to the dock. The dock module then uses that information to instruct the boat how to move to place itself parallel with the dock at a distance suitable for loading and unloading of passengers. We assume the loading and unloading of passengers will be done on the same side of the boat at each dock. Figure 14 shows a diagram of the dock measuring the boats position.

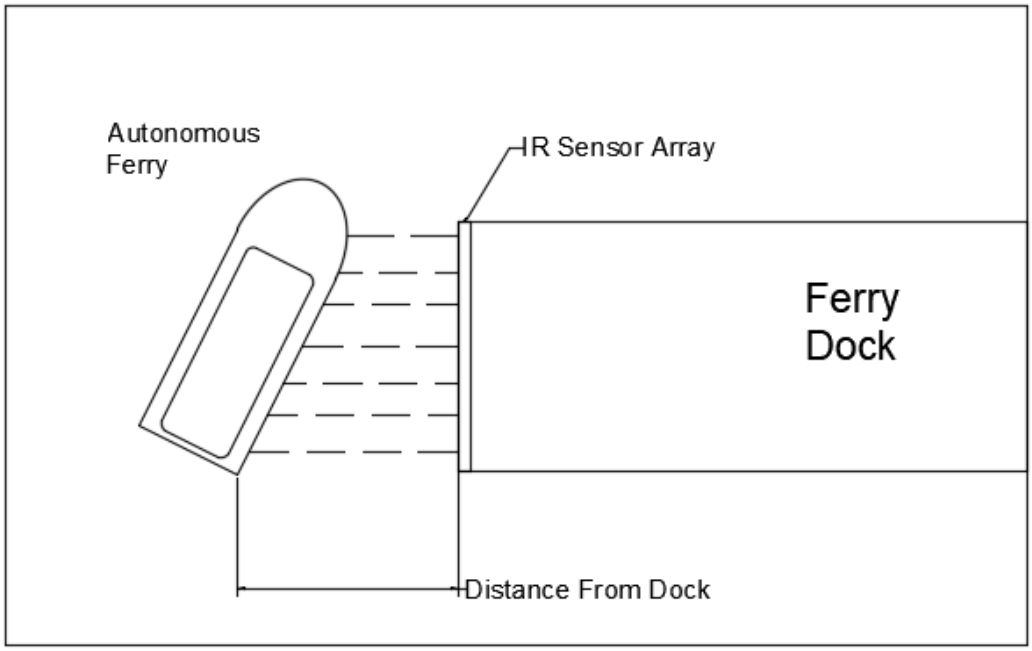


Figure 14: Docking Autonomous Ferry

Due to the complexity of this project to maintain organization development and testing phases were broken into separate tasks. The fact that our autonomous vehicle is a boat poses additional issues. If any error or malfunction happened while in the water, it would be difficult to retrieve the boat. For this reason we initially be implemented our autonomous control system on a RC car. This will allowed us to confirm the operation of the system before placing the autonomous vehicle in the water.

The first system that was implemented is the navigation system. We started by getting the RC car to travel from one point to another. This was the simplest movement required of our system. Once this functionality was confirmed, the vehicle then followed a predefined path between a starting and finishing point using multiple waypoints. Unfortunately, performing these tests on a flat surface is different from the effects of waves and wind while on the water. Once the navigation system was confirmed to be working, the next system was implemented.

Since movement in the water is so different than that on land, this part of the testing was done on the RC boat. First all of the previous systems that were tested on the RC car needed to

be tested again on the RC boat. With the navigation and object avoidance systems confirmed to be working, an array of IR sensors were mounted on a dock. First the distances measured by the sensors were confirmed. Next the data was sent wirelessly to the boat. Once the data was accurately transmitted, the ability for the boat to place itself next to the dock without making contact was confirmed.

4.2 Project Specification

For this project the boat used had to be scaled down from an actual ferry. For this reason the distances and speeds of our boat had to be scaled down as well. The typical size of a ferry can range from 30 feet to 600 feet.[11] For our calculations we assumed to be scaling down from the median of this range. The average cruising speed of a ferry is between 46.3km/h to 64.8km/h. Again the median of this range was taken to scale down to our model. Since the boat is 30 inches long, an approximation of 3 feet was used to scale relative speeds down.

When the autonomous boat began docking, the Bluetooth module controls the speed of the boat. Since average docking speed of a ferry could not be found we assume the boat should travel at a “no wake” speed given for personal watercraft. This is roughly listed as 8 km/h. For the scaled down model, while within the no wake zone, the speed should be to 0.42 km/h. This keeps operating speed low in higher traffic areas. The no wake zone is considered to be within 61 meters of a dock.[12] The scaled down model remains at a no wake zone while within 3.2 meters of the dock.

Once the boat is outside the no wake zone the navigation controls cruising speed. For full sized ferries, the cruising speed is approximately 55.6 km/h. The scaled down models cruising speed should be 2.43 km/h. This speed should be maintained while outside of no wake

[11]Ferry Operations for the 21st Century
[12]Proper Speed and Distance on a PWC

zones. The GPS lets the boat know when it is within the no wake zone at its dock. While traveling between the locations the boat needs to detect objects that are potential collisions.

While traveling, a full size boat needs to stay 30 meters away from other ships and 60 meters away from any swimmers.[13] For added safety we assume a minimum safe distance from any object will be 60 meters for a full size ship. The scaled down model then needed to maintain a distance of 3.15 meters from potential collisions. If an object is detected within this range the scale model ship returns to a no wake speed of 0.42 km/h. While this decreases the efficiency of the autonomous ferry, it greatly increases the safety and maneuverability of the ship. Once the object has left the 3.15 meter range of the autonomous boat, the boat can return to its cruising speed. Once the ferry enters the no wake zone for the next dock it returns to its no wake speed.

For this ship to maintain a distance of 3.15 meter the sensor that could aid in object avoidance needed to have a minimum range of 3.15 meters. For the scale model ferry HC-SR04 ultrasonic sensors was be used. These sensors provide a range of 4 meters of distance measurements. This means the ship is be able to detect objects before they are in the unsafe range for cruising speeds. Obviously this range would be extremely unsafe for full size ferries, but there are ultrasonic sensor that have a range of up to 60 meters. For the IR sensors on the dock the GP2Y0A02YK0F only has a range of 30 cm. Again this would be impractical for full scale use but these sensors could be replaced with long range IR sensors which can have a range of 5 meters, plenty for aligning next to a dock.[14] The GPS, however, would be perfect for full scale used since it is accurate within 2 meters. If the GPS transceiver is centered in the ferry the location may be off center but still on the ferry. The 2 meter accuracy has more of a negative

[13]Know the Rules
[14]GP2Y0A710K0F Distance Measuring Sensor Unit Measuring distance: 100 to 550 cm Analog output type

effect on the scale model but with the aid of the compass, the position and heading of the boat is still be relatively accurate.

4.3 Module Functionality

There are two basic functionalities for autonomous vehicles. The vehicle needs to know its location and destination as well as sense the environment. A GPS and a compass combination was selected to sense the coordinate and the direction of the destination relatively to the boat. This was used to navigate the boat to the dock. In order to do this, the data from the navigation system is used to control the motors to steer the boat to the correct direction. To detect objects around it, ultrasonic sensors can be mounted to the sides of the boat. Lastly, to communicate with the dock, the boat was equipped with a Bluetooth module. The block diagram of the autonomous system can be seen in figure 15.

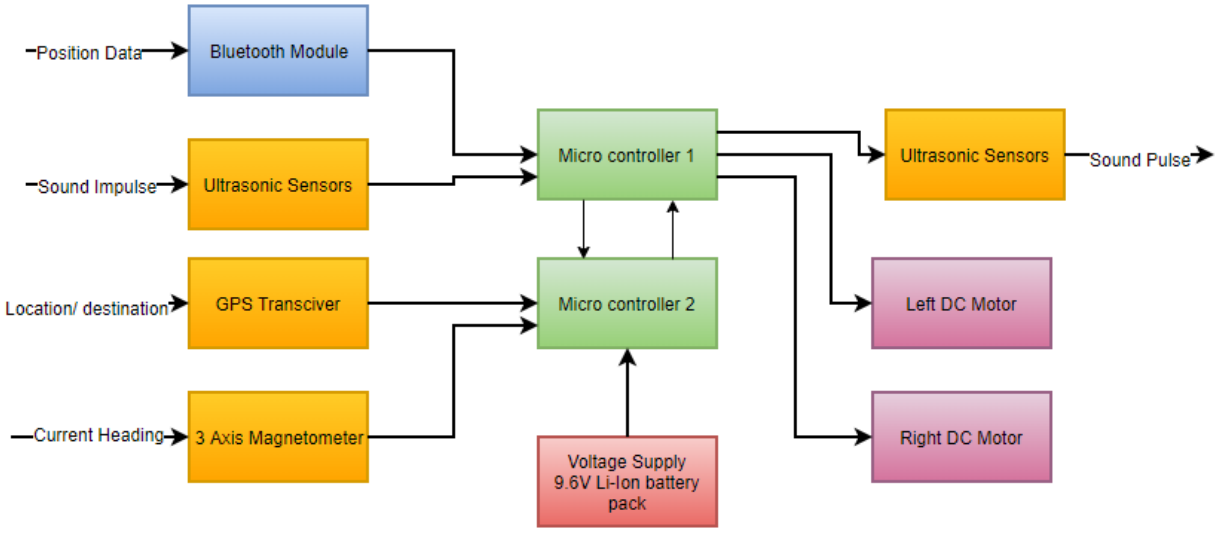


Figure 15: Autonomous Boat Block Diagram

4.3.1 Motor Control

To control those motors' speed, a L293N H-bridge motor control module was applied. This module included a huge heatsink to dissipate the heat produced in the h-bridge. To power

the h-bridge, a power supply and ground wires were soldered to the remote control's microcontroller. This allows the connection of the battery to the car without needing additional modification. A ground wire was also used to create a common ground between an Arduino and the h-bridge. This is needed since the Arduino controls the h-bridge with 5V and PWM signals.

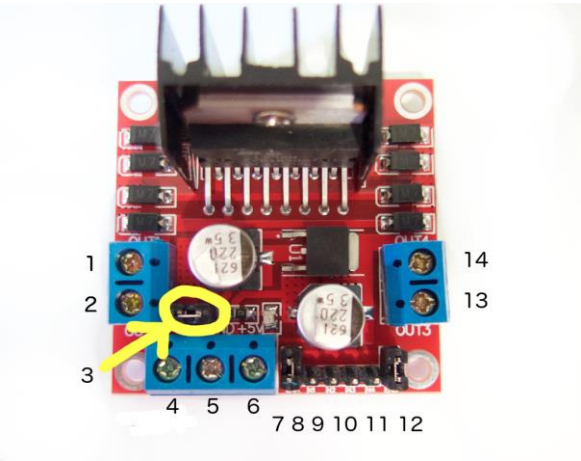


Figure 16: LN298 H-Bridge

With the power and ground supplies connected, we then connect the microcontroller's signal and those motors to the H-bridge module. The OUT3 and OUT4 pin of the h-bridge are connected directly to motor. Pins 7 and 3 of the Arduino were connected to IN3 and IN4. The pins are set to outputs that control the direction of the motor. Pin 6 of the Arduino should be connected to ENB pin of the H-bridge module with supplying 5V. This will then enable the H-bridge to activate the motor.

The steering motor was wired in a similar manner. The supply wires to the steering motor was connected to OUT1 and OUT2 of the h-bridge. This supplies the power to the motor. Pins 2 and 4 of the Arduino were connected to IN1 and IN2. These pins are set to outputs and are used to control the direction of the motor. Pin 5 was connected to the ENA pin of the h-bridge. When ENA is supplied with 5V, the motor is enabled

After wiring the module, the program to control the motors could then be written. To organize this program, each command for the motors control was written as an individual

function. This allows the main loop of the program to call each function when needed to activate the motors.

The first function is used to control the car to move the forward. To set the direction, the pin connected to IN3 is set to 5V and the pin for IN4 is connected to ground. With the direction set the speed of the motor is set by pulse width modulating the ENB connection on the h-bridge. Using the analogWrite() function, the PWM signal can be set to a value between 0 (0% max speed) and 255 (100% max speed). For testing purposes, the ENB was pulsed at 75, which is about 30% max speed.

The next function performs the opposite command. This function reverses the direction of the motor, allowing the car to drive backwards. To change the direction of the motor, the voltages at IN3 and IN4 needed to be reversed. This function sets IN3 to 0V and IN4 to 5V. The ENB pin is again pulsed at 75, which is about 30% max speed.

The motor stop function was added for two purposes. The first purpose is to handle an error condition. If the microcontroller ever sends an invalid command, the motor stop function would be called, bringing the car to a complete stop to prevent accident. The second purpose is to bring the vehicle to a complete stop before reversing the motor. This prevents reversing the voltages of the motor while it is still moving in the opposite direction. To stop the motor, the ENB is pulsed at 255, or 100% max speed. This is equivalent to a constant 5V signal. Next IN3 and IN4 are connected to the ground. This shorts the motors wires to ground, which helps “brake” the motor to prevent any rolling on an uneven surface.

The first function used to control the steering is the function to steer right. The ENA is pulsed at a rate of 50, or 20% max speed. Next pin IN1 is set to 5V and pin IN2 is set to 0V. Next, the function has a delay that keeps the PWM active for a set time. While this active the motor, it will continue to turn right. The function takes a value named “steeringAngle.” This variable sets the length of the delay the enable signal is active and effectively how large of a steering angle is used. Also, the angle of steering take a values named “navigationSteering”.

This variable allows different steering angles to be set based off the need to turn sharper or not. Using the switch case for every motion of steering, the motor performed successfully the smooth steering. Currently, only one steering angle is set but more will be added. This delay sets the wheels at about half way between the mechanical stops.

The next function operates similarly to the last but turns the car to the left. The delay feature and ENA pulse are set to the same value of the previous function. The main difference is the supply voltages of IN1 and IN2 are reversed. IN1 is now set to 0V and IN2 is set to 5V.

The next function is designed to stop the steering motor and hold it in place. Since the previous steering functions only turn the wheels, this function is intended to keep the wheels in the desired position. The ENA pin is pulsed at 255, or 100% max speed. IN1 and IN2 is connected to ground to add resistance to the motor turning. This helps maintain the desired steering and keeps the wheels straight when not steering command is received.

Testing the motor control circuits focused on two topics: can the motors move in the correct directions, and how fast the motors will move for the specified PWM signal. To perform the testing, a small test code was written. This program cycled through each of the motor functions, one at a time. Function was run for three seconds then stopped immediately. A delay of two seconds was set between each function call. This ensured the microcontroller to focus on one specific function at a time. Once the functions were confirmed to work, tests speeds were set at increments of about 10% and increased until the speeds became excessive. We found that a provide a 30% PWM signal to the motor was a good testing speed. These are not the final operating speeds since they must increase for carrying more weight on the car after mounting the Arduinos and other sensors.

4.3.2 Ultrasonic Sensing

Since the ultrasonic sensing was not added to the autonomous platform, the wiring was not being finalized. The sensor was connected in a temporary configuration just for testing. This

consisted of power and ground supplies, an activation signal, and a return signal. While the original plan was to include these sensors in the project, project deadlines prevented these sensors from being integrated into the autonomous system.

Figure 17: DIYMore Ultrasonic Sensor

To measure the distance between the sensor and the object, we applied ultrasonic proximity sensor. On the sensor, the trigger pin fired an 8-cycle sonic burst and the echo pin receives the sonic burst. Since we know ultrasonic waves travel at the speed of sound, which is 340m/s. Thus, by measuring the time it takes for that 8-cycle sonic burst to return back to the sensor, we calculated the distance by the following formula.

$$distance = speed * \frac{time}{2}$$

Equation 7

The reason that we had to divide the time by 2 is because the time that we measured are the time of the sonic burst traveling round trip. Thus, since we only need the distance between the two objects, we will have to divide the time by 2 to get the time it takes in one way.

To test our code, we placed an obstacle on the right and held our ultrasonic sensor on the left. We first started to test it by placing the obstacle at around 25 cm apart from our ultrasonic sensor, like figure 18. The measured value was then displayed on our laptop using a

serial terminal. The measured data was between 23~26cm (The distance observed: 24 24 25 25 23 23 24 24 26 25) which is pretty close to the expected 25cm.

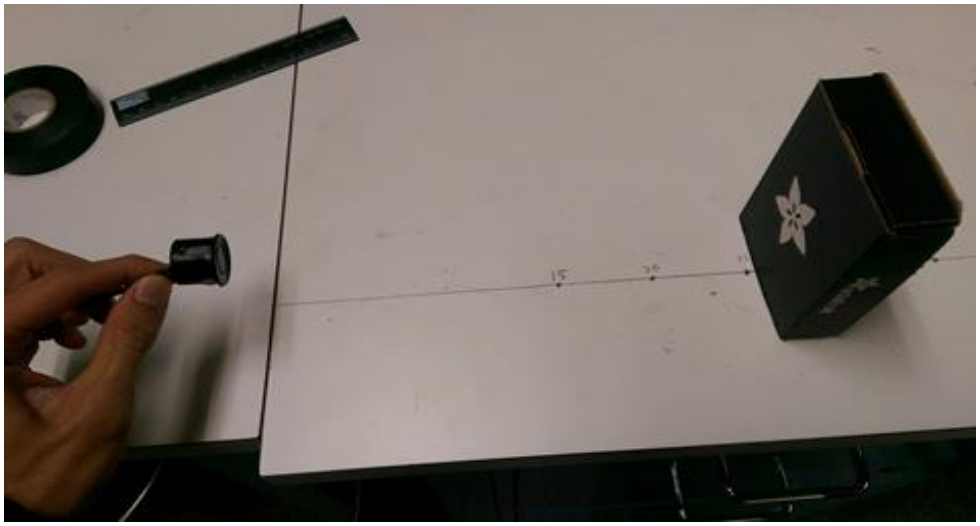


Figure 18: Ultrasonic Testing

We then moved the obstacle on the right to 40 cm apart from the ultrasonic sensor. As the measured values displayed on the laptop shown, it's around 38~40 cm (the observed distance: 39 39 39 39 40 38 38 38 38 38) which was accurate enough to detect objects near the boat.

Figure 19: Ultrasonic Testing

4.3.3 GPS

Since this project used a GPS shell, no wiring needed to be done. The shell required pins to be soldered to the board. With the pins on the GPS shell the GPS shell is attached directly on top of the Arduino Uno. The GPS shell allows access to the pins of the Arduino so the I/O can still be used.

Figure 20: Adafruit GPS Logger Shield

The purpose of using the GPS is to know our real-time position in terms of coordinate system. Once the boat knows its current position, it can then compare it with the target

coordinate and thus get the remaining optimal distance and our heading toward the target in terms of radians.

To get our current position, we applied the fetch GPS function in the Adafruit GPS library. As long as the GPS module can find a satellite to connect, the module returned the latitude and longitude coordinate in terms of degrees. The fetched coordinate was verified by typing the coordinates into Google map to double check.

To calculate the distance, we applied the “haversine” formula, which calculates the shortest distance of two coordinates over the earth’s surface. The “haversine” formula is a very commonly used formula in navigation systems. It can measure the great-circle distance between two points on a sphere given their longitudes and latitudes. This is generally used in spherical trigonometry. The following are the equations used:

$$a = \sin^2(\frac{\Delta\varphi}{2}) + \cos(\varphi_1) * \cos(\varphi_2) * \sin^2(\frac{\Delta\gamma}{2})$$

Equation 8

$$c = 2 * \tan^{-1} 2\frac{\sqrt{a}}{1-a}$$

Equation 9

$$Distance = R * c$$

Equation 10

φ1= Current Latitude, γ_1 = Current Longitude

φ2 = Current Latitude, γ_2 = Current Longitude

a = the square of half the chord length between the points

c = the angular distance in radians, R = radius of earth = 6,371 km

To calculate the headings, we used the inverse trigonometric functions to obtain the angle in radians of the two points. The equations used are shown below:

$$x = \sin(\frac{\Delta\gamma}{2} * \cos(\varphi_2))$$

Equation 11

$$y = \cos(\varphi_1 * \sin(\varphi_2) - (\sin(\varphi_1) * \cos(\varphi_2) * \cos(\frac{\Delta\varphi}{2})))$$

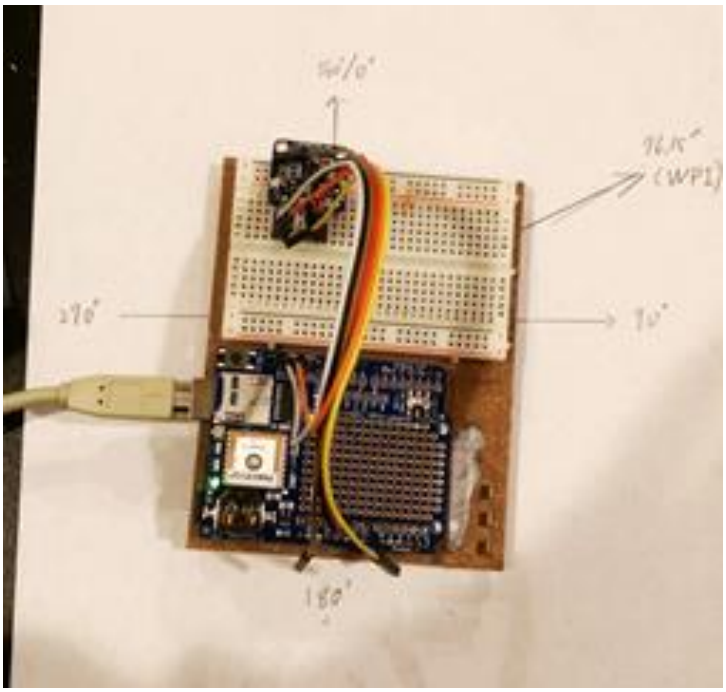
Equation 12

$$Heading = \tan^{-1}\frac{y}{x}$$

Equation 13

The calculated heading was in radians. This was then being compared to the heading that the Magnetometer has detected. After that, we determined the difference between actual heading and the target's heading and thus adjust further if needed.

For testing, we used WPI Higgins lab as our target location and 5 Newton avenue as our current starting location. As figure 21 shown, our laptops monitor printed out our current location coordinate and our target locations coordinate. It then calculated the distance between these two coordinates and display it on the monitor. In figure 21, desired heading meaning that the direction that we should be aiming for and current compass heading means the current direction that we were aiming. The current compass heading was measured by the compass. Our system was based on the difference in the desired heading and current compass heading and indicates that we should turned right in order to point to the correct location.



Time: 2:44:40.0

Date: 11/12/2017

Fix: 1 quality: 1

---------- Target and Current GPS location ---------

Current location: 42.270786, -71.825408

Target location: 42.273841, -71.807929

Distance to next coordinate: 1477.14 meter

Desired heading: 76.71

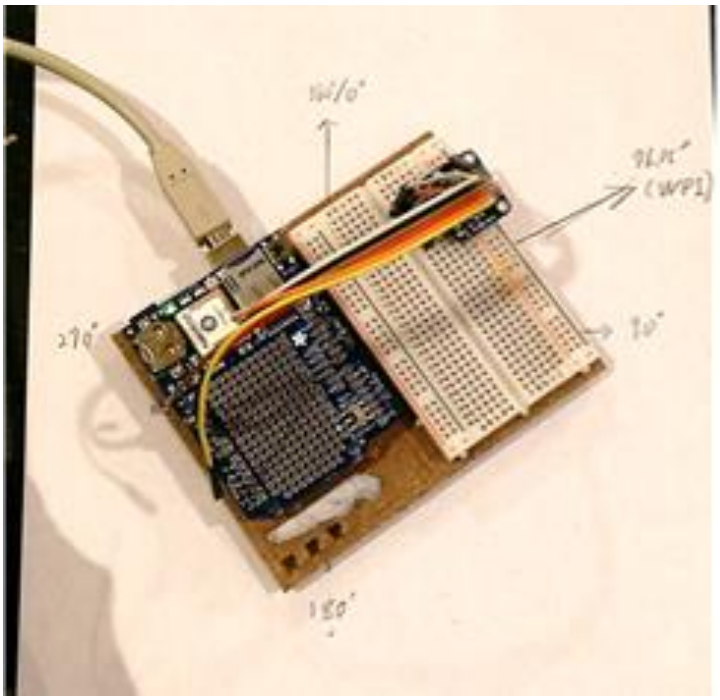
---------- Compass basic ---------

Current compass heading: 4.18

Adjusting toward to the right

Figure 21: GPS Setup and Testing

Once we have turned the device's heading toward right, as figure 21 shown, the current compass heading display in figure 22 was very close to our desired heading and thus it is no longer displaying turn right but displayed on track.



GPS basis:

Time: 2:55:51.0

Date: 11/12/2017

Fix: 1 quality: 1

---------- Target and Current GPS location ---------

Current location: 42.270744, -71.825508

Target location: 42.273841, -71.807929

Distance to next coordinate: 1467.76 meter

Desired heading: 76.60

---------- Compass basic ---------

Current compass heading: 76.87

Figure 22: GPS Setup and Testing

4.3.4 Compass

Using I2C communication, we needed to connect Vcc of the Compass to 5V input voltage of Microcontroller. The SCL pin 5 and SDA pin 4 of the compass needed to connect with the SCL and SDA pin of the Microcontroller, respectively. Also, the ground for both needed to be connected as well.

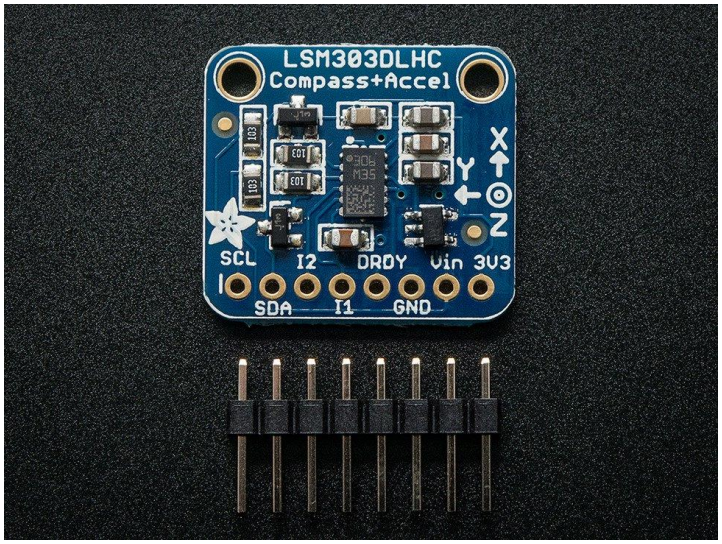


Figure 23:LSM303DLHC Compass & Accelerometer

To achieve a tilt compensated electronic compass system, we needed the measured data from both 3-axis Magnetometer sensor and a 3-axis accelerometer sensor. The 3-axis Magnetometer sensor is used to obtain the earth's magnetic field. We then used the three-axis's data to determine the heading angle with respect to the magnetic north. The 3-axis accelerometer is used to obtain the tilt angles of yaw, pitch and roll to compensate the measured heading angle of the 3-axis Magnetometer sensor. To obtain the magnetometer and the accelerometer data, we had to go to the register where the latest measured data is stored and copy these values to the local variable to our microcontroller through I2C communication protocol.

In the following are the example of reading the accelerometer value of each axis from the assigned register.

```
ACC_Data[0] = Read(0x18, 0x28);//read OUT_X_L_A (MSB)
ACC_Data[1] = Read(0x18, 0x29);//read OUT_X_H_A (LSB)
ACC_Data[2] = Read(0x18, 0x2A);//read OUT_Y_L_A (MSB)
ACC_Data[3] = Read(0x18, 0x2B);//read OUT_Y_H_A (LSB)
ACC_Data[4] = Read(0x18, 0x2C);//read OUT_Z_L_A (MSB)
ACC_Data[5] = Read(0x18, 0x2D);//read OUT_Z_H_A (LSB)
Ax = (int) (ACC_Data[0] << 8) + ACC_Data[1];
```

Ay = (int) (ACC_Data[2] << 8) + ACC_Data[3];

Az = (int) (ACC_Data[4] << 8) + ACC_Data[5];

After we had all the raw values from the magnetometer and the accelerometer, we can then apply the following formula shown in the equations below.

$$roll = \sin^{-1}(\frac{Accy}{\cos(pitch)})$$

Equation 14

$$mx = magx * \cos(pitch) + magz * \sin(pitch)$$

Equation 15

$$my = magx * \sin(roll) * \sin(pitch) + magy * \cos(roll) - magz * \sin(roll) * \cos(pitch)$$

Equation 16

$$Heading = 180 * \tan^{-1}(\frac{my}{mx})$$

Equation 17

Accy = measured acceleration data in the y axis (g/m^2)

magx = measured magnetic data in x axis (μTesla)

magy = measured magnetic data in y axis (μTesla)

magz = measured magnetic data in z axis (μTesla)

To test if the compass is now tilt compensated, the device was tilted it along its top and bottom edge slowly. If the compass output needed to a difference of ± 2º. This meant the compass is tilt compensated

To see if the tilt-compensate algorithm is working or not, we first placed our devices flat on the table and measured our current heading value as figure 23 shown. We can see that our

current heading is around 67 radians. Compass heading measured values were 66.52, 67.62, 66.40, 67.31, 65.67, 67.62, 66.91, 67.15, 66.80, 66.80, 67.55, 67.15, 67.31, and 67.04

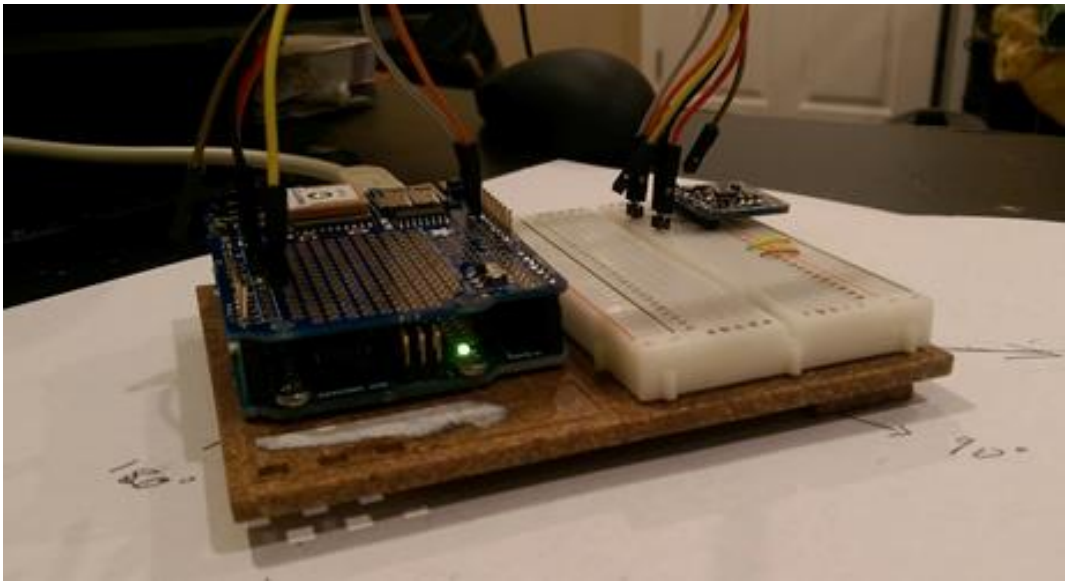


Figure 24: Compass Testing

We then slightly tilted our device to one side and measured the heading result again. Without tilt-compensated algorithm, the heading values are no longer around 67 radian it has changed to around 82 radian (the compass heading: 82.11, 82.59, 82.46, 82.11, 82.34, 82.32, 82.67, 82.27, 82.88, 82.69, 82.16, 83.06, 82.00, 82.90). The results show the heading value barely changes from the measured result that we obtain when it is flat on the table surface. The heading are 69.28, 69.15, 68.87, 68.69, 68.66, 68.89, 68.79, 68.61, 69.05, 69.06, 69.62, 68.49, and 67.54.

4.3.5 SPI

In this system, we used two microcontrollers which can interact to each other to control the components of the overall system more effectively. In order to make them communicate to each other, the SLK, MISO, MOSI, SS, GND pins of the first microcontroller needed to be connected to the same pins of the second microcontroller, respectively.

Our two microcontrollers are connected and communicate through SPI. The microcontroller that is in charge of processing the GPS and electronic compass's data (Master) will send the desired control information to the microcontroller that is in charge of the motor control and the obstacles detection (Slave).

The microcontroller (Master), based on the difference of the angle between actually heading and the desired heading, calculates which direction the steering has to move. This data is transferred through SPI to another microcontroller (slave).

To transfer the data, the microcontroller (Master) has to send a logic low through the chip select line (ss) to the microcontroller (slave) in order to let the microcontroller (slave) knows that the data that appears on the data line (MOSI) is for it to receive. After transmitting the data, microcontroller (Master) then sends a logic high to let the microcontroller (slave) know that it can stop reading the data from the data line (MOSI)

Once the microcontroller (slave) detected that there are new data coming into its 8 bit register. The code immediately stops at whatever it is currently processing and check what the new data is. Inside this SPI interrupt function, we update the motor's steering control to make sure it is steered to the desired direction.

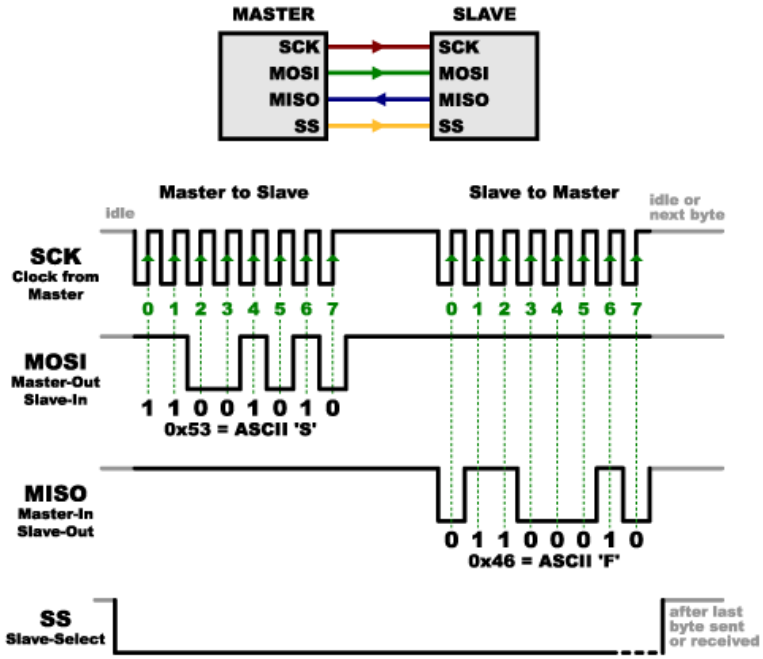


Figure 25: SPI Communication Timing Diagram

To test the functionality of the SPI the Arduinos were wired together. Test characters were sent from one controller to the next. Using the serialPrint() function, the characters received were printed out on the terminal. The delay between sending and receiving the characters was found to be more than expected. By removing some of the test codes from the sending Arduino, we were able to send and receive data in under a second.

4.3.6 IR Range Sensing

The IR distance sensors used are 3 wire sensors. Each sensor required a voltage supply and ground. The third wire is an analog output that is connected to one of the analog pins on the Arduino Nano. The 5V supply and ground pins of the Arduino Nano supplied the power to the sensors. An array of five IR sensors were connected to the analog pins A0-A4.

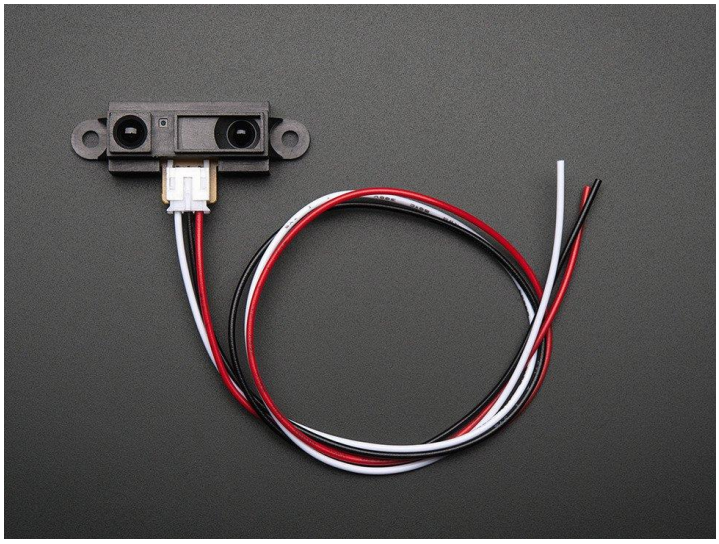


Figure 26: Infrared Sensor

The analog voltage output from the sensor is an analog value between 0V and 5V. To convert the voltage into a distance, the datasheet provided an equation to convert the voltage into a distance in centimeters. The equations is shown below.

$$Distance = 10650.08 * Voltage^{-0.935} - 10$$

Equation 18

Since the sensors to have a limited operating range, we needed to add logic to detect when the sensors get readings outside the range. Since we don't need to take measurements near the 1.5m limit of the sensor, once the readings are at 1m, we no longer use the measurement. This prevents incorrect distances from being used.

For testing the measurement accuracy of the sensor, a single IR sensor was connected to the Arduino Nano. Using the serialPrint() function, the distances measured were displayed on the computers screen. Measurements were taken at 5cm intervals up to 50cm. When under 15cm, the measurements become erratic and are no longer accurate enough to use. With our operating range tested, we connected an array of five IR sensors to the Arduino Nano. The measurements of each sensor were printed on the terminal. This was intended to test the relative accuracy of the sensors. With an object placed perpendicular to the sensor array, the

actual distance was 20cm. Each sensor measured +/-2cm from the actual distance. This was accurate enough for our dock.

4.3.7 Bluetooth Modules

In order to communicate with the dock, the boat and the dock both need a Bluetooth module to send and receive data. For Bluetooth communication the HC-06 and HC-05 modules were selected. These two modules are almost identical except that the HC-05 is a “master” module, while the HC-06 is a “slave” module. Since only the master module can initiate the connection between the devices, the dock will look to make the connection and the boat will wait until a connection is made.

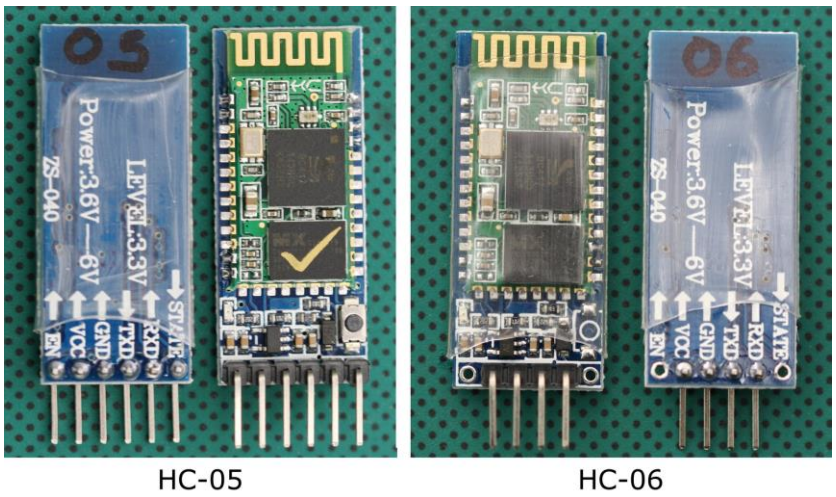


Figure 27: HC-05 & HC-06 Bluetooth Modules

The wiring for the boat and dock is basically the same. Each BT module requires a 5V and ground reference connection. The modules use 3.3V logic on the TX and RX pins. This is fine for the TX pin since the Arduino can accept a 3.3V signal on the RX pin but the Arduino outputs 5V logic. This voltage could damage the RX pin of the Bluetooth module. To bring this voltage down to 3.3V, a voltage divider is connected to the RX pin on the Bluetooth module. This brings the Arduinos 5V signal down to 3.3V.

The modules required some set up to function the way we wanted. The master module can be put in full AT mode, which allows the modules to be configured using a serial terminal. The HC-05 was configured to look for the HC-06 on the boat. Once the HC-06 is detected, the Bluetooth modules are automatically paired and begin sending data. The AT command LINK allowed the modules to only pair with each other. We were also able to set a password for the devices that is check by each module before pairing. This prevents other devices from linking to the Bluetooth modules and preventing data from being sent. The actual program waits until the Bluetooth connection is confirmed. Once the connection is confirmed, the program begins sending and receiving data via Bluetooth. The dock sends characters telling the boat how to move when in close proximity to the dock. The commands are basic and tell the boat to move forward, backward, left, or right, depending on the measurements taken by the IR sensors.

In order to confirm operation before integrating the system on the boat, an Arduino Nano was connected to the HC-05 and an Arduino Uno was connected to the HC-06. A PC was connected to each of the microcontrollers and a test program was uploaded that allows the serial terminal on the PC to send and receive strings. Each module was confirmed to be able to send and receive data while connected to the PC. The next phase is testing the movement commands when integrated with the dock and IR sensors.

5. Design

5.1 Autonomous Vehicle Design

The initial approach was to mount the sensors and modules on an RC car. This allowed quick access to the vehicle during initial set up and testing of the system. The RC car used was a Foxx S911, a roughly one-foot long RC truck. The plastic body was removed and a temporary mounting surface was attached to the frame of the truck. The microcontrollers were powered by

the USB cable connected to a PC. This allowed the car to be quickly disconnected in case of any errors in testing. The wiring is almost identical to the RC boat and will be covered later in the section. This platform was used to simply test the ability of the system to adjust its heading while traveling to a programmed location. The RC car platform can be seen in figure 27.

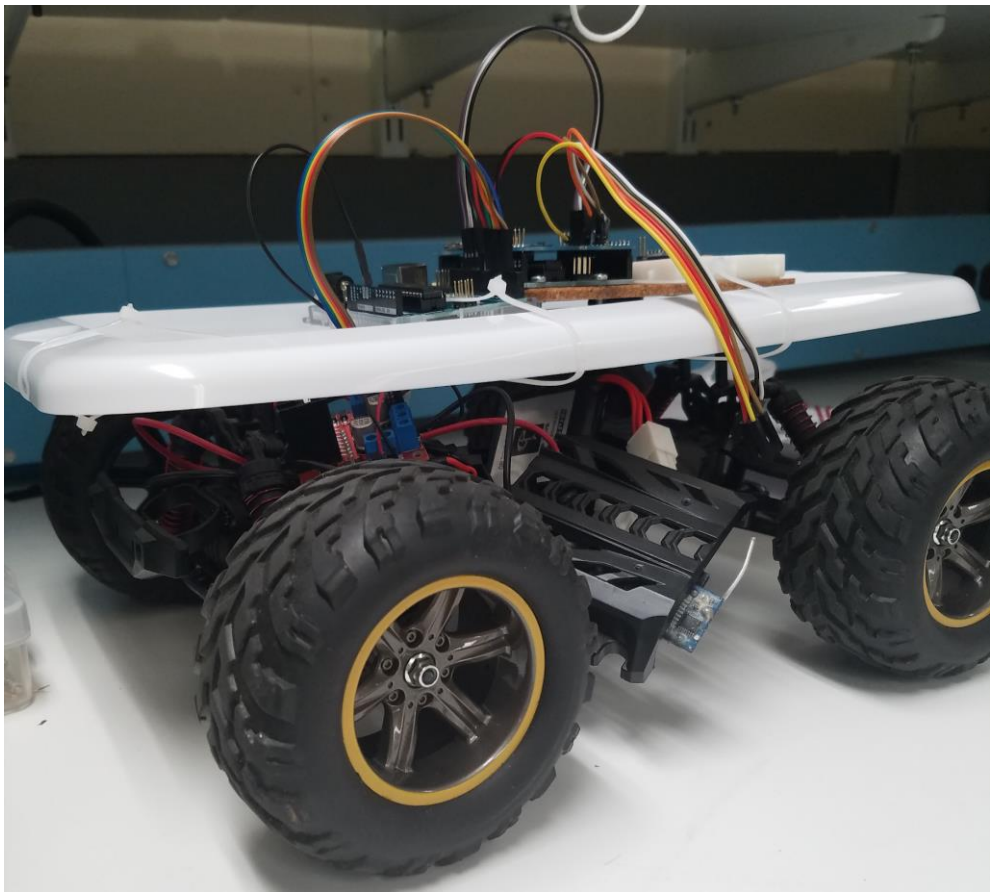


Figure 28: RC Car Platform

A 30” RC battleship was selected as the final platform for the prototype. The boat is long and flat providing an ideal surface for mounting the electronics needed for autonomous control. The length of the boat helps prevent the boat from tilting forward and backward when moving, which would decrease the accuracy of the compass measurements. The boat also has two propellers instead of a single propeller and rudder. This gives the ship high maneuverability and allows for more precise movement.

In order to control the boat some modifications needed to be made. Initially, all electronics for the boat were stored inside the boat. Since we wanted to be able to easily access the electronics, we decided to mount everything to the top of the boat. This meant we needed to relocate the boats battery to the top of the ship. Since the application of the system is in the water, we needed to protect the electronics from the water. An air tight Tupperware container was used to seal in the electronics. The top of the container was mounted to the top of the ship providing a flat surface for mounting the modules as well as protecting the circuits from the water. The wires controlling the motor and connecting to the RC board needed to be routed into this sealed unit. Holes were drilled into the cover that allowed wiring to run between the container and the boats interior. These holes were then resealed using a silicon-based sealant.

With the added components added to the boat, stability became a concern. With the added weight to the boat as well as the higher center of gravity, the boat was more likely to flip over during hard turns. To compensate for this instability, foam tubes were mounted to either side of the boat. This helped improve the buoyancy of the boat and made the boat wider to help prevent any tipping. Figure 28 shows the modified boat.

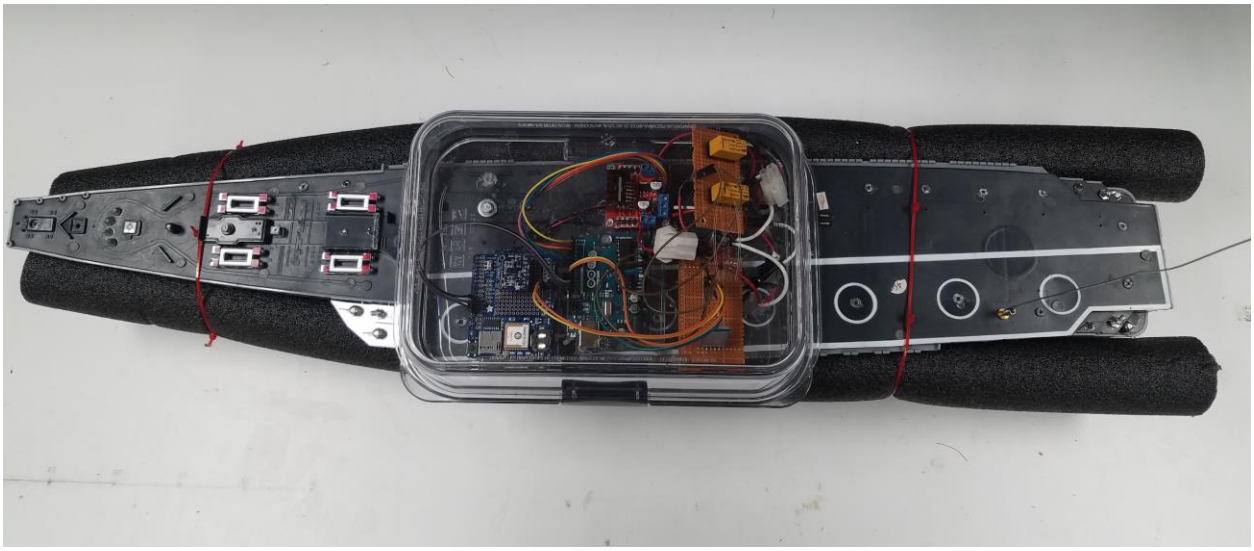


Figure 29: Autonomous Boat Prototype

For the boat to move autonomously, each module tested was integrated into one larger system. This system is similar to that of the RC car, but with the addition of a few components. Two relays were added to the motor control circuit and a Bluetooth module was added to the Arduino controlling the motors. The circuit diagram for the boat can be seen in figure 29 below.

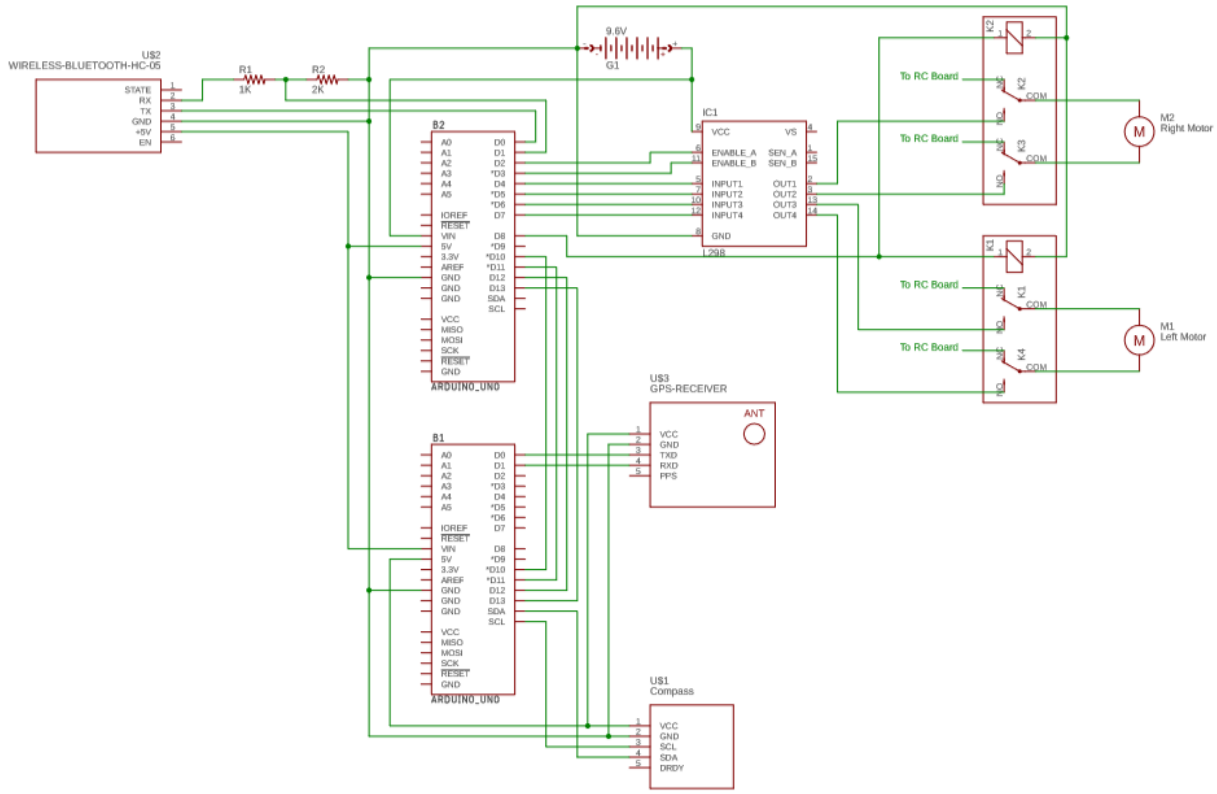


Figure 30: Autonomous Boat Schematic

The relays were added to control the switching between manual control, and autonomous control. This is a necessary function in any autonomous system in case of emergency or malfunction. These relays connect each motor to either the RC board or the H-bridge. The RC board is connected to the normally closed (NC) contact, while the H-bridge is connected to the normally open (NO) contact. This allows the boat to switch between autonomous control and RC control while protecting the H-bridge as well as the RC board. There are two possible ways to switch between modes. While the boat is in range of the dock, the Bluetooth modules are sending data between the dock and boat. The dock can send a

character ‘s’ to the boat which indicates it to switch the relay supply voltage from 5V to 0V. The RC board output is then connected to the motors, allowing the boat to be controlled by remote. The other method to switch is a simple timer using the millis() function. This function measures the amount of time elapsed since the program began. This is in case there is a need to switch to manual control, but the boat is out of range of the Bluetooth connection. Once the time has reached a certain value, the relay will again be switched to 0V allowing manual control of the boat. The motor control circuit can be seen in figure 30.

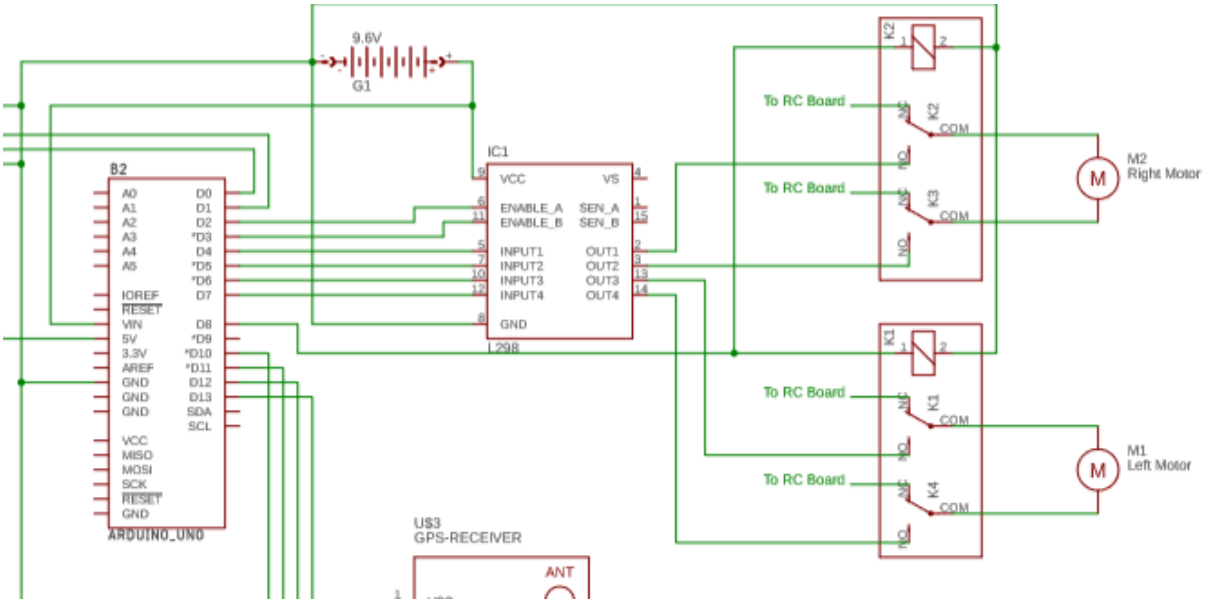


Figure 31: Relay Circuit

As mentioned earlier, a Bluetooth module was added to the circuit. This module can send and receive data from another Bluetooth module located on the dock. The dock is able to measure the position of the boat and sends instructions on how to maneuver to reach a docking position. The characters received from the dock tell the boat to either move left, right, forward, backward, or to stop. The module used is a HC-06. This module is considered a “slave.” In the context of Bluetooth, a slave device can be connected to by another device but is unable to initiate the connection. This module operates using 3.3V logic, which is an issue since Arduinos use 5V logic. Arduinos are able to receive 3.3V logic without issue but sending 5V logic data to

the RX pin of the HC-06 will damage the module. To correct this issue, a voltage divider consisting of a 2kΩ and 1kΩ resistor was used to bring the arduinos 5V output logic down to 3.3V for the HC-06. The boat's Bluetooth circuit can be seen below in figure 31.

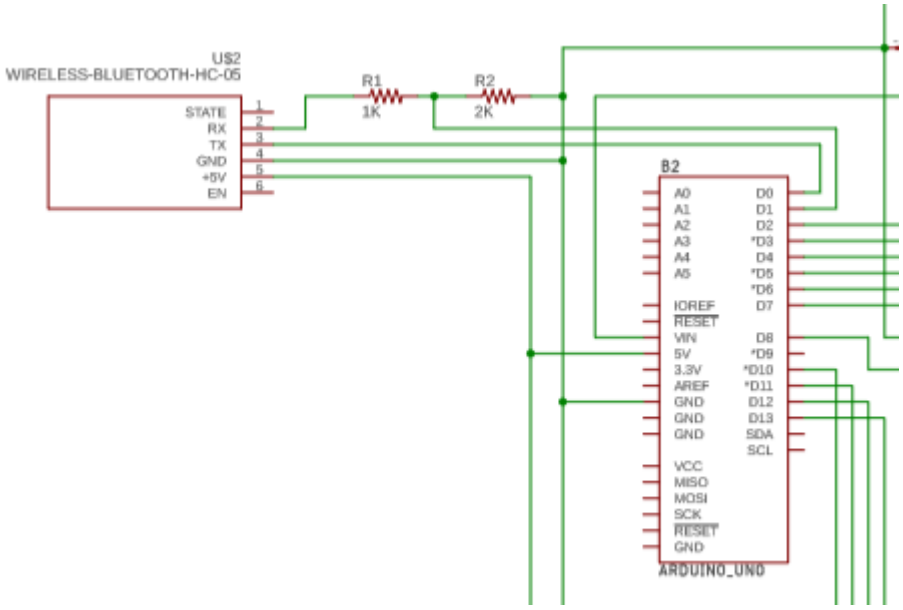


Figure 32: Boat Bluetooth Circuit

The circuit to control navigation in the RC car is the same as in the boat. One Arduino is connected to the GPS and compass. This Arduino decides how the boat needs to move to go to the programmed GPS location. Once it calculates the direction it then sends a character to the Arduino controlling the motors. The motors are then controlled by the H-bridge to execute the desired maneuver. The Arduino connected to the GPS will send different characters depending on the error in heading. The larger the error in direction, the faster the motors will be PWM. This increases the speed the boat turns when facing the wrong direction allowing the boat to quickly get back on track. This has the added bonus of making small adjustments when needed to prevent the boat from overcompensating during small adjustments. The navigation circuit can be seen in figure 32.

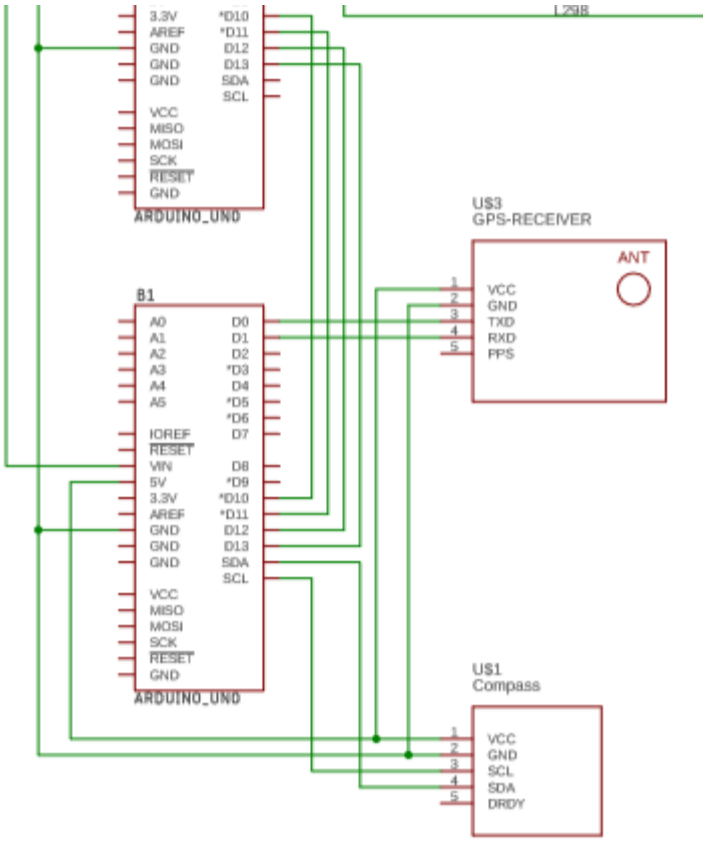


Figure 33: Navigation Circuit

5.2 Dock Design

In order to test the boats ability to dock a model dock was needed. The dock was constructed out of thin plywood as well as insulating foam to allow the dock to float in the water. Plywood was mounted to the top of the dock that allowed another piece to be mounted to the front of the dock. This provided a location for the IR sensors to be mounted. The plywood attached to the front of the dock was cut with tall enough to leave room to adjust the sensors vertically if needed. The prototype dock can be seen in figure 33.

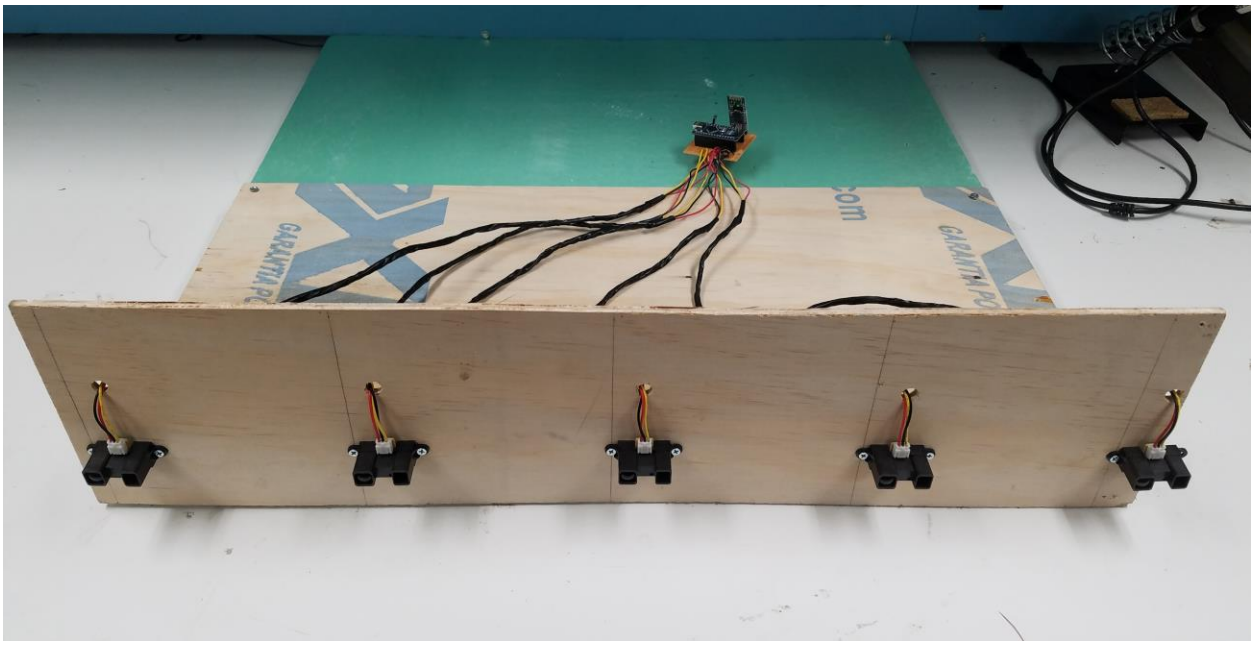


Figure 34: Prototype Dock

The main function of the dock is to guide the boat into a docking position. The docks circuit is made up of an Arduino Nano, a Bluetooth module, a toggle switch, and IR sensors. The circuit for the dock can be seen in figure 34.

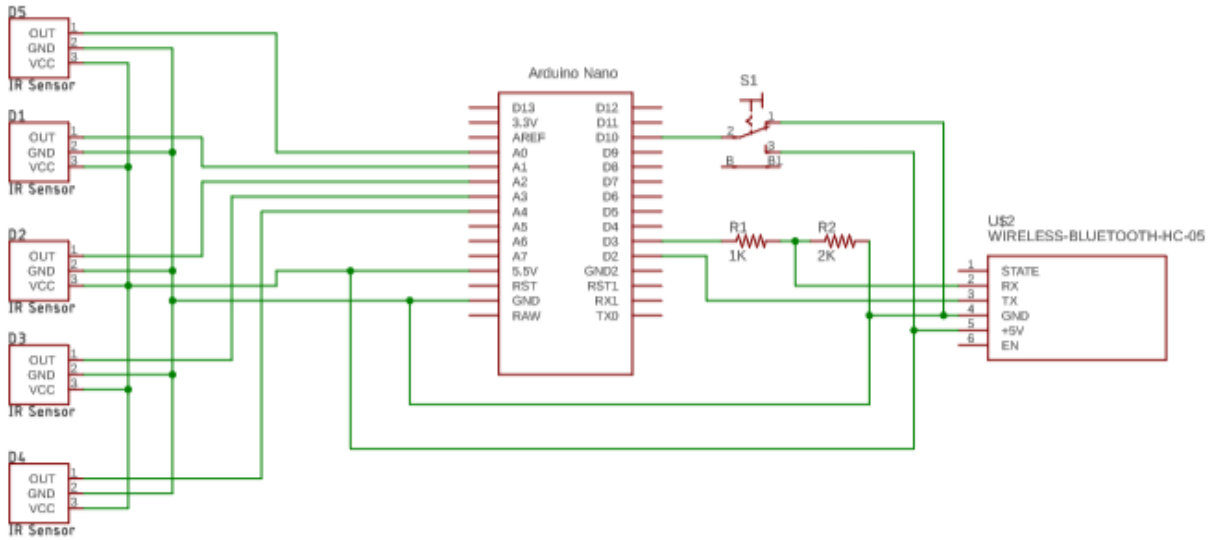


Figure 35: Dock Circuit

In order to measure the boat's position relative to the dock, five IR sensors are mounted to the front of the dock. Using the distance from the dock gathered from the sensor, the dock can decide how to move the boat in order to reach the desired position. For this application, the desired position is all sensors reading approximately equal distances from the boat. When each IR sensor detects the boat, that means the boat is in front of the dock but may be at an angle relative to the dock. When all sensors are approximately equal in value, the boat will be parallel with the IR sensors, and be in the desired docking position. An error of 2 centimeter is allowed between distance measurements since IR sensor do have a degree of error depending on the operating environment. Depending on the measured position of the boat, the dock decides if the boat needs to move forward, backward, left, right, or stop.

Once the Arduino selects the needed adjustment, the command needs to be sent to the boat. This is done using the Bluetooth connection between the dock and boat. The dock uses a HC-05 Bluetooth module. This module can operate as either a slave or master. For our application this module is set up as a master. This means it will initiate the connection once the other Bluetooth module is in range. To set this up, the AT commands were used to program the settings of the Bluetooth module. These are commands that are run in a serial window and operate outside of the Arduinos program. Using the AT commands, the passwords for each device were set. Next the devices were paired and linked using their addresses. This means that they will only pair with each other and continue to attempt to pair until a connection is made. Once that connection is made the dock will send characters via Bluetooth that tell the boat how to move to the desired location. When the connection is made, the navigation data is ignored and only the data sent over Bluetooth is used.

The last component of the circuit is the switch. This is used to switch between autonomous and manual mode when within range of the dock. When the toggle switch is low, the dock will send data to the boat as usual. If the toggle switch is high the character 's' is sent to the boat. When this is received, the relays are supplied with 0V, returning the boat to manual

control. This is intended to be used in any case when the boat is not operating as expected and the autonomous control needs to be overridden.

6. Analysis and Results

6.1 RC Car Prototype

With all modules tested, the motor control and GPS/compass modules were integrated into the RC platform. While testing we found that the microcontroller connected to the GPS was not sending data fast enough. This cause significant errors in direction and made the vehicle head in the wrong direction, then overcompensate to correct it. During testing the speeds of the motors also became an issue. With the PWM of the motors low the turning and acceleration prevented the car from navigating in the desired way. When the PWM was increased, the vehicle was able to overcome the friction and move toward the desired destination. The downside was with the increased PWM the error in steering angle increased. Most cases resulted in the vehicle oversteering past the desired heading. This combined with the increased speed of the drive motor caused the vehicle to swerve back and forth significantly on the way to the destination. Since the motor control is significantly different on the boat, it was decided that the project would move forward to the final platform.

6.2 Navigation

The first test was the boat's ability to navigate from one location to another. Since the test requires a GPS signal, the navigation system required outdoor testing. This meant that during testing the boat was subjected to environmental effects such as wind and current. For the

testing Salisbury Pond in Institute Park was selected as the test location. This location is close to WPI's campus and was one of the first bodies of water to thaw.

To set the course the GPS on the boat was used to measure the boats current position. The boat was placed at the destination and the location was recorded. That location was then programmed into the Arduino as the destination for the boat. The boat was then placed in the water down the shoreline. The start location was selected due to its ease of access to the water as well as the fact the boat would travel close to the shoreline in case of malfunction. The start and destination locations of the test can be seen in figure 35, with the boat traveling south toward Salisbury St.

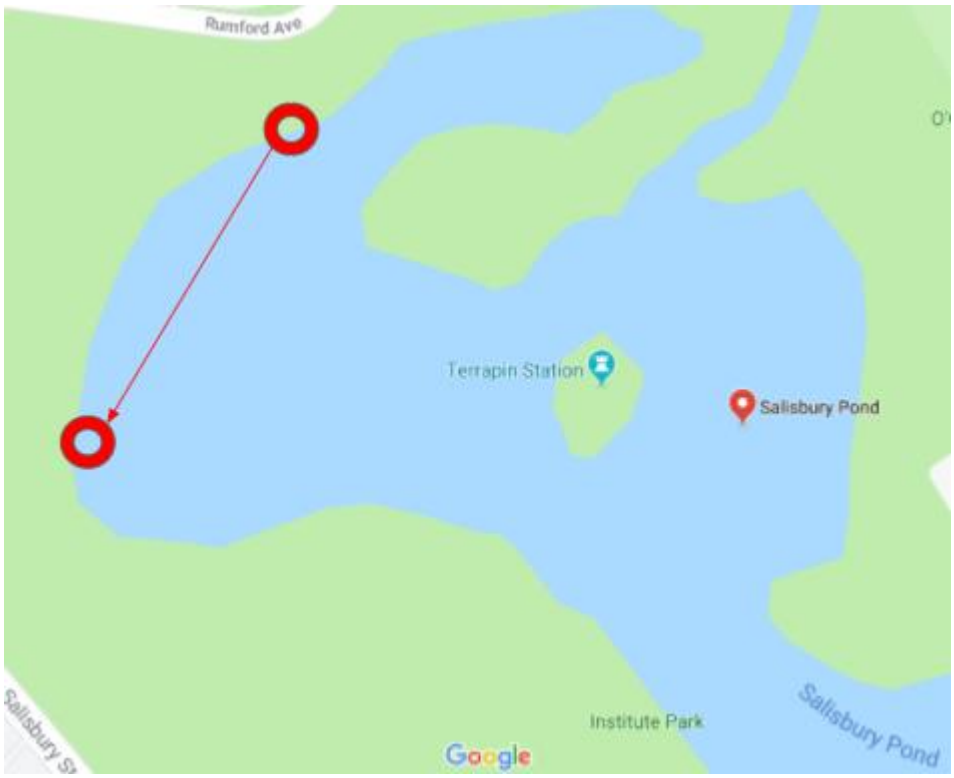


Figure 36: Navigation Test Path

During initial testing a major issue arose. When the boat attempted to move forward it would consistently pull to the right. This occurred when the boat was in autonomous mode and when it was in manual mode. In autonomous mode the boat was unable to correct its direction and was unable to reach the destination. We found the issue to be due to the left motor

overpowering the right motor. Both motor circuits were tested for improper connections but both circuits were error free. The issue is the motor itself. While the dimensions of the boat where ideal for the test platform, the motors are lower quality and unreliable.

While ideally the motors would be replaced before continuing, due to parts availability and time constraints we decided to continue testing with the motors currently installed. In order to balance those motors, the PWM values to the H-bridge where adjusted. Since the right motor was the weaker, the PWM was set as high as possible, 255. The left motor PWM was set to a lower value, 100. This value balances those motors almost perfectly but created other issues. The new PWM values caused the boat to move much slower than desired. When the wind or current of the water began pushing the boat off course, the boat had difficulty correcting its heading. To fix this issue, the PWM was increased when turning left and right. The higher motor speed while turning helped give the boat more momentum while correcting the boats direction faster. This momentum was carried while the boat attempted to go straight. While straight is a lower motor speed it seemed to be able to maintain the boats speed. While the motors imbalance caused issues with the testing, we were able to successfully navigate from one location to another.

6.3 Docking

With the navigation system operating, the docking system was tested. This test again was conducted in Salisbury Pond. For the test the dock was placed in the water and the boat was placed in the beginning of the docking position. This meant that one of the IR sensors was detecting the front of the boat. For this test we assumed the boat would be approaching from the same direction each time it docked.

Figure 37: Testing the Autonomous Docking

Since the testing was done outdoors, the boat was subjected to various environmental changes. The first one to note was the effect of wind and current on the boat. While testing the wind and current was much stronger than when testing the navigation system. This caused issues with placing the boat in a starting position, as well as the boat maintaining its current position once docked. Even the dock was pushed by the current. The second major issue was the measurements by the IR sensors. The IR sensors would produce random error values causing the boats measurements not to be taken. This happened randomly while docking and appeared to be due to saturation of the sensors from the sunlight outside and the reflection off the water.

The motor speed required frequent adjustment while testing. Initially, the same PWM values that were used in the navigation system. This became an issue since the boat would

move forward too quickly and the boat would pass the dock before the dock could send the reverse command. With the motors turned down the boat approached the dock at a more desirable speed. When the boat passed the dock, the reverse command was set but the motors were unable to overcome the momentum of the boat. The reverse motor speed was increased and the boat was then able to align itself with the dock. While docking the boats angle to the dock was changed. In some cases, the boat was able to turn left or right against the current but many times the current caused the boats angle to the dock to exceed the angle that the dock could read.

One of the more successful aspects of the docking system was caused by the high currents. Once the boat was in the docking position it was able to come to a stop. The current then began to push the boat past the dock again. Once the first IR sensor no longer detected the boat, the reverse command was again sent bringing the boat back into the docking position. This shows that the boat was able to position itself in front of the dock and, in most cases, maintain that position.

7. Recommendations

While the goals of this project were met, many changes could be made to improve the system. The boat platform provided a good base for the project, but also provided limitations. Some research was initially done into building the boat for the project. Building the boat would allow us to make the boat larger and design around the electronics used. The larger boat would also help lessen the effects of current on the boat helping the boat's ability to dock constantly. Since the boat was built, the motors an RC controls would need to be purchased. This would allow us to purchase higher quality motors to alleviate the imbalance of motor power discovered during testing.

The IR sensors also created some issues while docking. The environmental effects on the sensors makes their readings erratic. The IR sensors could be replaced with LiDAR to measure the distance of the boat from the dock. This would be an expensive change and would require more research into possible sensors to use. This would help improve the reliability of the commands sent by the dock. This could also require a redesign of the dock due to the mounting and added weight.

Lastly, we had hoped to include ultrasonic sensing into the project. A truly autonomous vehicle would need a means of detecting objects in its environment. Unfortunately, we were unable to add the sensors to the boat. The work involved in wiring, mounting, programming, and testing would be extremely time consuming. Including this would have significantly pushed this project beyond the deadline.

Appendix

A. Bill of Material

Bill of Materials					
Item	Part Number	Price	Quantity	Total	Source
Infrared Proximity Sensor	GP2Y0A02YK0F	14.95	7	104.65	https://www.sparkfun.com/products/8958
Infrared Sensor Jumper Wire	SEN-08733	1.50	7	10.50	https://www.sparkfun.com/products/8733
Triple Axis Magnetometer	MAG3110	14.95	2	29.90	https://www.sparkfun.com/products/12670
Triple Axis Accelerometer Breakout	MMA8452Q	9.95	2	19.90	https://www.sparkfun.com/products/12756
Triple-Axis Gyro Breakout	ITG-3200	24.95	2	49.90	https://www.sparkfun.com/products/11977
GPS Receiver	EM-506	39.95	1	39.95	https://www.sparkfun.com/products/12751
Ultrasonic Sensor	HC-SR04	3.95	4	15.80	https://www.sparkfun.com/products/13959
Microcontroller - Arduino Uno	R3	24.95	1	24.95	https://www.sparkfun.com/products/11021

B. Bibliography

[1] S. Ahvenjärvi, "The Human Element and Autonomous Ships," *TransNav: International Journal on Marine Navigation and Safety of Sea Transportation*, vol. 10, *(3)*, pp. 517-521, 2016. Available: https://doaj.org.

[2] C. Paris, "Norway Takes Lead in Race to Build Autonomous Cargo Ships," *Wall Street Journal*, 2017. Available: https://www.wsj.com/articles/norway-takes-lead-in-race-to-build-autonomous-cargo-ships-1500721202.

[3] F. Plumet *et al*, "Toward an Autonomous Sailing Boat," *IEEE Journal of Oceanic Engineering*, vol. 40, *(2)*, pp. 397-407, 2015. . DOI: 10.1109/JOE.2014.2321714.

[4] J. Mannes, "Rolls-Royce planning autonomous naval ship for patrol, surveillance and mine detection," Available: http://social.techcrunch.com/2017/09/12/rolls-royce-planning-autonomous-naval-ship-for-patrol-surveillance-and-mine-detection/.

[5] (). *Infrared distance sensor*. Available: http://home.roboticlab.eu/en/examples/sensor/ir_distance.

[6] (). *Accelerometer Basics*. Available: https://learn.sparkfun.com/tutorials/accelerometer-basics.

[7] (). *Boating Rules*. Available: https://maritime.college/Boating-Rules.php.

[8] (). *Ferry Operations for the 21st Century*. Available: http://www.maritimelawcenter.com/html/ferry_operations.html.

[9] (). *Proper Speed and Distance on a PWC*. Available: https://www.boat-ed.com/oregon/studyGuide/Proper-Speed-and-Distance-on-a-PWC/10103801_700143117/.

[10] Roads and Maritime Services, N S W. (). *Know the rules*. Available: http://www.rms.nsw.gov.au/maritime/safety-rules/rules-regulations/know-the-rules.html.

[11] (). *GP2Y0A710K0F Distance Measuring Sensor Unit Measuring distance: 100 to 550 cm*

*Analog output type *. Available: https://cdn-shop.adafruit.com/datasheets/gp2y0a710k.pdf.

[12] Tao Wang *, Nanning Zheng, Jingmin Xin and Zheng Ma, "Integrating Millimeter Wave Radar with a Monocular Vision Sensor for On-Road Obstacle Detection Applications," 2011. Available: http://www.mdpi.com/1424-8220/11/9/8992/htm.

[13] Karthik Ramasubramanian and Brian Ginsburg, "WR1243 sensor:

Highly integrated

76–81-GHz radar front-end

for emerging ADAS applications," 2017. Available: http://www.ti.com/lit/wp/spyy003/spyy003.pdf.

[14] Cesar Iovescu and Sandeep Rao, "The fundamentals of

millimeter wave sensors," 2017. Available: http://www.ti.com/lit/wp/spyy005/spyy005.pdf.

[15] J. Spange, "Autonomous Docking for Marine Vessels Using a Lidar and Proximity Sensors." 2016. Available: https://brage.bibsys.no/xmlui/handle/11250/2440566.

[16] (). *Laser Technology*. Available: http://www.lasertech.com/Laser-Sensors.aspx.

[17] Guivant Jose, Nebot Eduardo and Baiker Stephan, "Autonomous Navigation and Map building Using Laser Range Sensors in Outdoor

Applications " 2006. Available: https://pdfs.semanticscholar.org/ddc6/fb419680bd13e2a8ce2fc4f4abfb9063f85b.pdf.

[18] (). *IR Break beam Sensors*. Available: https://learn.adafruit.com/ir-breakbeam-sensors/overview?view=all.

[19] (9/9/16). *How Do PIRs Work*. Available: https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/how-pirs-work.

[20] (). *IR Sensor What is Different About the Sharp IR Sensor?*. Available: http://education.rec.ri.cmu.edu/content/electronics/boe/ir_sensor/4.html.

[21] Anonymous "24 GHz Radar For Non-Contact Industrial Sensors," Available: http://www.microwavejournal.com/articles/28982-ghz-radar-for-non-contact-industrial-sensors.

[22] (). *Infrared Proximity Sensor*. Available: http://www.g9toengineering.com/AllSaints/infraredproximity.htm.

[23] (11/5/14). *Ultrasonic ranger library and demo code for arduino*. Available: https://www.elecrow.com/wiki/index.php?title=File:Ultrosonic_ranger_module_Library_and_demo_code_for_arduino_10.zip.

[24] (6/18/15). *Ultrasonic Ranging Sensor Module*. Available: https://www.elecrow.com/wiki/index.php?title=Ultrasonic_Ranging_Sensor_Module#Introduction.

[25] (8/17/13). *Ultrasonic Sensor*. Available: https://www.scribd.com/document/166448734/Ultrasonic-Sensor-Wikipedia-The-Free-Encyclopedia.

[26] Anonymous "Sonar," 2017. Available: https://en.wikipedia.org/w/index.php?title=Sonar&oldid=800644267.

[27] T. Wang *et al*, "Integrating Millimeter Wave Radar with a Monocular Vision Sensor for On-Road Obstacle Detection Applications," *Sensors*, vol. 11, *(9)*, pp. 8992-9008, 2011. Available: http://www.mdpi.com/1424-8220/11/9/8992. DOI: 10.3390/s110908992.

[28] J. Guivant, E. Nebot and S. Baiker, "High accuracy navigation using laser range sensors in outdoor applications," *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. no.00CH37065)*, vol. 4, pp. 3822 vol.4, 2000. . DOI: 10.1109/ROBOT.2000.845326.

[29] Anonymous "Laser rangefinder," 2017. Available: https://en.wikipedia.org/w/index.php?title=Laser_rangefinder&oldid=793789776.

[30] B. Cook and K. Cohen, "Multi-source sensor fusion for small unmanned aircraft systems using fuzzy logic," *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 1-6, 2017. . DOI: 10.1109/FUZZ-IEEE.2017.8015593.

[31] Anonymous (). *Ultrasonic Sensor - Wikipedia, The Free Encyclopedia | Ultrasound | Waves*. Available: https://www.scribd.com/document/166448734/Ultrasonic-Sensor-Wikipedia-The-Free-Encyclopedia.

[32] Anonymous ().U.S. labor costs increase solidly in the fourth quarter. Available: https://www.reuters.com/article/us-usa-economy-costs/u-s-labor-costs-increase-solidly-in-the-fourth-quarter-idUSKBN1FK1XR , Jan. 31, 2018[Mar. 17, 2018]

[33] Anonymous (). *NYC Ferry Homepage* Available: https://www.ferry.nyc/

[34] Anonymous (). - *2016 Recreational Boating Statistics.*Available: http://www.uscgboating.org/library/accident-statistics/Recreational-Boating-Statistics-2016.pdf

C. Code

Program for Navigating the boat

```
// Test code for Adafruit GPS modules using MTK3329/MTK3339 driver
//
// This code shows how to listen to the GPS module in an interrupt
// which allows the program to have more 'freedom' - just parse
// when a new NMEA sentence is available! Then access data when
// desired.
//
// Tested and works great with the Adafruit Ultimate GPS module
// using MTK33x9 chipset
//    ------> http://www.adafruit.com/products/746
// Pick one up today at the Adafruit electronics shop
// and help support open source hardware & software! -ada
// GPS coordinate distance measure wesite:
// https://andrew.hedges.name/experiments/haversine/
#include <stdlib.h>
#include <SPI.h>
#include <Adafruit_GPS.h>
#include <SoftwareSerial.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM303_U.h>

----------------------------*/
#define SCK_PIN  13
#define MISO_PIN 12
#define MOSI_PIN 11
#define SS_PIN   10
```

```
SoftwareSerial mySerial(8, 7);
Adafruit_GPS GPS(&mySerial);
#define GPSECHO  true
/*////////////////////////////////////////////////////////////////*/

// this keeps track of whether we're using the interrupt
// off by default!
boolean usingInterrupt = false;
void useInterrupt(boolean); // Func prototype keeps Arduino 0023 happy
/*////////////////////////////////////////////////////////////////*/

/** ----------------------------  Set target's GPS coordinate ------------------------------------**/


/*Target corrdinates*/
float targetLatArray[] = {42.277854, 42.277187, 42.27384};
float targetLongArray[] = {-71.808235,-71.807617,-71.807933};

float currentLat, currentLong, targetLat, targetLong, desiredHeading;
//CurrentTarget keeps track of the target in the array
//headingThreshold is the amount of degrees the vehicle can be off its desired heading before adjusting the course
//distanceThreshold is the amount of meters away from target coordinate, that when exceeded it will continue towards the next coordinate
int currentTarget = 0, headingThreshold = 4, distanceThreshold = 3;
//When checking for additional coordinates, the vehicle when stop when currentTarget is equal to noOfCoords
int noOfCoords = 3;

int arrive = 0; // 0 = Not arrive to the location, 1 = arrive to the location

/* Assign a unique ID to this sensor at the same time */
Adafruit_LSM303_Mag_Unified mag = Adafruit_LSM303_Mag_Unified(12345);
float heading = 0;
float distance = 0;
float error = 0;

char orient;

void setup()
```

```
{
  pinMode(MOSI_PIN, OUTPUT);
  pinMode(MISO_PIN, INPUT);
  pinMode(SCK_PIN, OUTPUT);
  pinMode(SS_PIN, OUTPUT);

  digitalWrite(SS_PIN, HIGH);

  SPI.begin();
  // connect at 115200 so we can read the GPS fast enough and echo without dropping
chars
  // also spit it out
  Serial.begin(115200);
  Serial.println("Adafruit GPS library basic test!");

  // 9600 NMEA is the default baud rate for Adafruit MTK GPS's- some use 4800
  GPS.begin(9600);

  // uncomment this line to turn on RMC (recommended minimum) and GGA (fix data)
including altitude
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
  // uncomment this line to turn on only the "minimum recommended" data
  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY);
  // For parsing data, we don't suggest using anything but either RMC only or
RMC+GGA since
  // the parser doesn't care about other sentences at this time

  // Set the update rate
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);   // 1 Hz update rate
  // For the parsing code to work nicely and have time to sort thru the data, and
  // print it out we don't suggest using anything higher than 1 Hz

  // Request updates on antenna status, comment out to keep quiet
  GPS.sendCommand(PGCMD_ANTENNA);

  // the nice thing about this code is you can have a timer0 interrupt go off
  // every 1 millisecond, and read data from the GPS for you. that makes the
  // loop code a heck of a lot easier!
  useInterrupt(true);
```

```
  delay(1000);
  // Ask for firmware version
 // mySerial.println(PMTK_Q_RELEASE);

    Serial.println("Magnetometer Test"); Serial.println("");

  /* Initialise the sensor */
  if(!mag.begin())
  {
    /* There was a problem detecting the LSM303 ... check your connections */
    Serial.println("Ooops, no LSM303 detected ... Check your wiring!");
    while(1);
  }
}


// Interrupt is called once a millisecond, looks for any new GPS data, and stores it
SIGNAL(TIMER0_COMPA_vect) {
  char c = GPS.read();
  // if you want to debug, this is a good time to do it!
#ifdef UDR0
  if (GPSECHO)
    if (c) UDR0 = c;
    // writing direct to UDR0 is much much faster than Serial.print
    // but only one character can be written at a time.
#endif
}

void useInterrupt(boolean v) {
  if (v) {
    // Timer0 is already used for millis() - we'll just interrupt somewhere
    // in the middle and call the "Compare A" function above
    OCR0A = 0xAF;
    TIMSK0 |= _BV(OCIE0A);
    usingInterrupt = true;
  } else {
    // do not call the interrupt function COMPA anymore
    TIMSK0 &= ~_BV(OCIE0A);
    usingInterrupt = false;
  }
```

```
}

uint32_t timer = millis();
void loop()                     // run over and over again
{
  // in case you are not using the interrupt above, you'll
  // need to 'hand query' the GPS, not suggested :(
  if (! usingInterrupt) {
    // read data from the GPS in the 'main loop'
    char c = GPS.read();
    // if you want to debug, this is a good time to do it!
 //   if (GPSECHO)
   //   if (c) Serial.print(c);
  }

  // if a sentence is received, we can check the checksum, parse it...
  if (GPS.newNMEAreceived()) {
    // a tricky thing here is if we print the NMEA sentence, or data
    // we end up not listening and catching other sentences!
    // so be very wary if using OUTPUT_ALLDATA and trytng to print out data
    //Serial.println(GPS.lastNMEA());   // this also sets the newNMEAreceived() flag to
false

    if (!GPS.parse(GPS.lastNMEA()))   // this also sets the newNMEAreceived() flag to
false
      return;  // we can fail to parse a sentence in which case we should just wait for
another
  }

  // if millis() or timer wraps around, we'll just reset it
  if (timer > millis())  timer = millis();

  // approximately every 1 seconds or so, print out the current stats
  if (millis() - timer > 1000) {           //  how fast can we use
    timer = millis(); // reset the timer
  getHeading();
  getGPS();
  //printData();
  adjustCourse();
  sendDate(orient);
```

```
  delay(10);
  }
}

/*////////////////////////////////////////////////////////////////////////////////////////////////////////////*/

/*-------------------------------------Get GPS information ---------------------------------------------
--------*/
void getGPS(){

    if (GPS.fix) {
      currentLat = GPS.latitudeDegrees;
      currentLong = GPS.longitudeDegrees;
    }

  }
/*////////////////////////////////////////////////////////////////////////////////////////////////////////////*/

/*---------------------------------- Function to calculate the desired heading ----------------------
------*/
float calculateHeading(float currentLat, float currentLong, float targetLat, float
targetLong){

  //Calculating the x and y arguments needed for the atan2 trig function
  //The heading is based on the target coordinates and the current coordinates
  float x = sin(radians(targetLong-currentLong)) * cos(radians(targetLat));
  float y = cos(radians(currentLat)) * sin(radians(targetLat)) - (sin(radians(currentLat)) *
cos(radians(targetLat)) * cos(radians(targetLong-currentLong)));
  float heading = atan2(x, y);
  //If the heading is negative it can't be used, this can be corrected by adding two pi.
  if (heading < 0.0){
    heading += 2*PI;
  }

  return degrees(heading);
}

/*////////////////////////////////////////////////////////////////////////////////////////////////////////////*/
```

```
/*---------------------------------- Calculate the distance between two latitudes and longitudes
----------------------------*/
/* Uses the haversine formula to calculate the distance between two points.
(http://www.movable-type.co.uk/scripts/latlong.html a variation is found on this page)
    The distance is returned in meters
*/
float calculateDistance(float currentLat, float currentLong, float targetLat, float
targetLong) {

    float a = sq((sin((radians(currentLat - targetLat)) / 2.0))) +
            (cos(radians(currentLat))*cos(radians(targetLat)) * sq((sin((radians(currentLong
- targetLong)) / 2.0))));
    float distance = 2.0 * 6371000 * asin (sqrt(a));
    return distance;
 }
/*////////////////////////////////////////////////////////////////////////////////////////////////////////////*/

/*---------------------------------- Adjusting the course of the vehicle based on the current
heading and desired heading ----------------------------*/
void adjustCourse(){
  //The distance to the current target coordinate is calculated
  distance = calculateDistance(currentLat, currentLong, targetLatArray[currentTarget],
targetLongArray[currentTarget]);
  //Stops the vehicle when there are no coordinates left
  //If the distance threshold is exceeded, move on to the next target location in the array
  if(currentTarget == noOfCoords){
    Serial.println("No coordinates left in the array.");
 //   drive(motor_pins, halt);
    return;
  }
  if(distance < distanceThreshold){
    Serial.println("Arrived at the target location.");
    delay(2000); //  stop for 2 second to indicate arrival
    Serial.println("If there are additional coordinates it will continue");
   if(currentTarget < 4){
    currentTarget++;
   }
   else{
    arrive = 1;
   }
```

```
  }

  //Calculates the desired/"optimal" heading based on the current location and target
location
  desiredHeading = calculateHeading(currentLat, currentLong,
targetLatArray[currentTarget], targetLongArray[currentTarget]);
  //Find the difference between the current heading and the desired heading
  //And does some correction to find the correct error
  error = desiredHeading - heading;
  if(error < -180) error += 360;
  if(error > 180) error -= 360;
  Serial.print(" - Error: ");
  Serial.print(error);
  //Depending on the error, the vehicle will then adjust its course
  //Small delays for the left and right turns are added to ensure it doesn't oversteer.
  if(abs(error) <= headingThreshold){
  //  Serial.println(" - On course");
    orient = 'd';
       Serial.println("----------- sending d ----------------------");
  }
  else if(error < 0 && error >= -45){
   orient = 'a';
   Serial.println("----------- sending a ----------------------");
  }

  else if(error < -45 && error >= -90){
   orient = 'b';
      Serial.println("----------- sending b ----------------------");
  }

  else if(error < -90){
   orient = 'c';
      Serial.println("----------- sending c ----------------------");
  }

  else if (error > 0 && error <= 45){
    orient = 'e';
       Serial.println("----------- sending e ----------------------");
  }
```

```
  else if (error > 45 && error <= 90){
    orient = 'f';
       Serial.println("----------- sending f ----------------------");
  }

   else if (error > 90){
    orient = 'g';
       Serial.println("----------- sending g ----------------------");
  }

  else{

  }
}

/*////////////////////////////////////////////////////////////////////////////////////////////////////////////*/
/*---------------------------------------- Get Compass Date ----------------------------------------
-----*/

void getHeading(){
    /* Get a new sensor event */
  sensors_event_t event;
  mag.getEvent(&event);

  float Pi = 3.14159;

  // Calculate the angle of the vector y,x
  heading = (atan2(event.magnetic.y,event.magnetic.x) * 180) / Pi;

  // Normalize to 0-360
  if (heading < 0)
  {
    heading = 360 + heading;
  }
//  Serial.print("Compass Heading: ");
//  Serial.println(heading);
  }

/*////////////////////////////////////////////////////////////////////////////////////////////////////////////*/
```

```
/*---------------------------------------- Get Compass Date ----------------------------------------*/
void printData(){

   Serial.println("xxxxxxxxxxxxxxxxxxxxxxxxx-- GPS basic --xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx");
   Serial.print("Time: ");
   Serial.print(GPS.hour, DEC); Serial.print(':');
   Serial.print(GPS.minute, DEC); Serial.print(':');
   Serial.print(GPS.seconds, DEC); Serial.print('.');
   Serial.println(GPS.milliseconds);
   Serial.print("Date: ");
   Serial.print(GPS.day, DEC); Serial.print('/');
   Serial.print(GPS.month, DEC); Serial.print("/20");
   Serial.println(GPS.year, DEC);
   Serial.print("Fix: "); Serial.print((int)GPS.fix);
   Serial.print(" quality: "); Serial.println((int)GPS.fixquality);

   Serial.println("xxxxxxxxxxxxxxxxxxxxxxxxx-- Target and Current GPS location --xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx");

   Serial.print("Current Location: ");
   Serial.print(GPS.latitudeDegrees, 6);
   Serial.print(", ");
   Serial.println(GPS.longitudeDegrees, 6);

   // if not arrive to the final destination, print the target location
  if(arrive != 1){
   Serial.print("Target Location(");Serial.print(currentTarget);Serial.print("): ");
   Serial.print(targetLatArray[currentTarget],6);
   Serial.print(", ");
   Serial.println(targetLongArray[currentTarget],6);
   Serial.print("Distance to next coordinate: ");
   Serial.print(distance);
   Serial.println(" meter");
   Serial.print("Desired heading: ");
   Serial.println(desiredHeading);

    }
```

```
  Serial.println("xxxxxxxxxxxxxxxxxxxxxxxxx-- Compass basic --
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx");
  Serial.print("Current Compass Heading: ");
  Serial.println(heading);
  if (orient == 'a' ||orient == 'b' || orient == 'c' ){
    Serial.println(" - Adjusting towards the left");
    }
    else if(orient == 'd'){
      Serial.println(" - On Track");
      }
      else if (orient == orient == 'a' ||orient == 'b' || orient == 'c'){
         Serial.println(" - Adjusting towards the right");
        }
        else{}
          Serial.println("-------------------------------------------");
    }

 /*////////////////////////////////////////////////////////////////////////////////////////////////////////////////*/

 void sendDate(char data){
  digitalWrite(SS, LOW);
  SPI.transfer(data);
  delayMicroseconds(10);
  digitalWrite(SS, HIGH);
  }
```

Program for motor control arduino on boat

```
#include <stdlib.h>
#include <SPI.h>

//Motor Control Pins
#define enA 5
#define enB 3
#define in1 4
#define in2 2
#define in3 6
#define in4 7
#define relay A0

//SPI Data Pins
#define SCK_PIN  13
```

```
#define MISO_PIN 12
#define MOSI_PIN 11
#define SS_PIN   10

//Global variables for steering commands
char steer;
char dock;
int driveSpeed = 150;
int dockSpeed = 100;

//Global variable for object detection
int objectDetected = 0;

#include <SoftwareSerial.h>
SoftwareSerial BTSerial(8, 9); // RX | TX

void setup() {
  Serial.begin (9600);
  Serial.println("Arduino with HC-06 is ready");
  // HC-06 default baud rate is 9600
  BTSerial.begin(9600);
  Serial.println("BTserial started at 9600");
  pinMode(MOSI_PIN, INPUT);
  pinMode(MISO_PIN, OUTPUT);
  pinMode(SCK_PIN, INPUT);
  pinMode(SS_PIN, INPUT);
  SPCR |= _BV(SPE);   // turn on SPI in slave mode
  SPCR != _BV(SPIE);

  SPI.attachInterrupt(); // turn on interrupt

  pinMode(enA, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(enB, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
  pinMode(relay,OUTPUT);
  delay(20000);
}

ISR (SPI_STC_vect)
{
/* receiving the integer, ranging from 0~255*/
```

```
//int number = SPDR;  // grab byte from SPI Data Register

/* receiving the char, ranging from 0~255*/
steer = SPDR;  // grab byte from SPI Data Register
}

void loop()
{
  while (BTSerial.available())
    {
      dock = BTSerial.read();
      //Serial.println(dock);
      if (dock == 's')
      {
       analogWrite(relay, 0);
       Serial.println("Manual");
      }
      else
      {
      analogWrite(relay, 255);
      dockingControl(dock);
      }
    }
  navigationSteering(steer);
   if (millis() > 180000)
    analogWrite(relay,0);
  else
    analogWrite(relay,255);

}


/*
 * ---------------------------------------------
 * This function turns both motors in the same
 * direction moving the boat backwards.
 * ---------------------------------------------
 */
void driveReverse( int D_PWM)
{
  analogWrite(enA, 220);
  analogWrite(enB, 220);
  digitalWrite(in1 ,LOW);
  digitalWrite(in2 ,HIGH);
```

```
  digitalWrite(in3, HIGH);
  digitalWrite(in4, LOW);
}

/*
 * ---------------------------------------------
 * This function turns both motors in the same
 * direction moving the boat forwards.
 * ---------------------------------------------
 */
void driveForward( int D_PWM)
{
  analogWrite(enA, 255);
  analogWrite(enB, 100);
  digitalWrite(in1 ,HIGH);
  digitalWrite(in2 ,LOW);
  digitalWrite(in3, LOW);
  digitalWrite(in4, HIGH);
}

/*
 * ---------------------------------------------
 * This function brings both motors to a stop.
 * ---------------------------------------------
 */
void driveStop()
{
  analogWrite(enA, 255);
  analogWrite(enB, 255);
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
  digitalWrite(in3, LOW);
  digitalWrite(in4, LOW);
}

/*
 * ---------------------------------------------
 * This function turns the boat to the right by
 * turning the motors in opposite directions.
 * ---------------------------------------------
 */
void rightTurn(int D_PWM)
{
  analogWrite(enA,255);
```

```
  analogWrite(enB, 125);
  digitalWrite(in1 ,HIGH);
  digitalWrite(in2 ,LOW);
  digitalWrite(in3, HIGH);
  digitalWrite(in4, LOW);
}

/*
 * ---------------------------------------------
 * This function turns the boat to the left by
 * turning the motors in opposite directions.
 * ---------------------------------------------
 */
void leftTurn(int D_PWM)
{
  analogWrite(enA, 125);
  analogWrite(enB, 255);
  digitalWrite(in1 ,LOW);
  digitalWrite(in2 ,HIGH);
  digitalWrite(in3, LOW);
  digitalWrite(in4, HIGH);
}

/*
 * ---------------------------------------------
 * This function controls the docking maneuvers
 * based on characters received via the bluetooth
 * module.
 * ---------------------------------------------
 */
void dockingControl(char nav)
{
  switch (nav)
  {
    //The boat moves in the forward direction
    case 'p':
    {
      Serial.println("Stop");
      driveStop();
      delay(5000);
      break;
    }
    case 'f':
    {
```

```
      Serial.println("Forwards");
      driveForward(dockSpeed);
      break;
    }
    //The boat moves in the reverse direction
    case 'b':
    {
      Serial.println("Backwards");
      driveReverse(dockSpeed);
      break;
    }
    //The boat adjusts to the right
    case 'r':
    {
      Serial.println("Right");
      rightTurn(dockSpeed);
      break;
    }
    //The boat adjusts to the right
    case 'l':
    {
      Serial.println("Left");
      leftTurn(dockSpeed);
      break;
    }
    //Maneuvering stops if an invalid command is received
    default:
    {
      driveStop();
      Serial.println("Default");
      break;
    }
  }
}

/*
 * ------------------------------------------------------
 * This function controls the ships nagivation when
 * there is a valid GPS signal and destination. Based
 * off commands from the other arduino, the ship will
 * manuver in the desired direction.
 * ------------------------------------------------------
 */
void navigationSteering(char turn)
```

```
{
   switch (turn)
  {

   //The boat makes a slight turn left
   case 'a':
   {
     Serial.println("Left");
     leftTurn(driveSpeed);
     break;
   }

   //The boat makes a medium turn left
   case 'b':
   {
     Serial.println("Left");
     leftTurn(driveSpeed);
     break;
   }

   //The boat makes a hard turn left
   case 'c':
   {
     Serial.println("Left");
     leftTurn(driveSpeed);
     break;
   }

   //The boat makes a slight turn right
   case 'e':
   {
     Serial.println("Right");
    rightTurn(driveSpeed);
     break;
   }

   //The boat makes a medium turn right
   case 'f':
   {
     Serial.println("Right");
    rightTurn(driveSpeed);
     break;
   }
```

```
    //The boat makes a hard turn right
    case 'g':
    {
      Serial.println("Right");
     rightTurn(driveSpeed);
      break;
    }

    //The boat continues straight on
    case 'd':
    {
      Serial.println("Forwards");
      driveForward(driveSpeed);
      break;
    }

   //Stops boat if invalid command is received
    default:
    {
      //Stop vehicle if error in navigation input
      driveStop();
      break;
    }
  }
}
```

Program on the dock for detecting the position of the boat and guide the boat to correct position.

```
// Basic Bluetooth sketch HC-05_03_9600
// Connect the HC-05 module and communicate using the serial monitor
//
// The HC-05 defaults to commincation mode when first powered on.
// Needs to be placed in to AT mode
// After a factory reset the default baud rate for communication mode is 38400
//
//
//  Pins
//  BT VCC to Arduino 5V out.
//  BT GND to GND
//  BT RX to Arduino pin 3 (through a voltage divider)
//  BT TX to Arduino pin 2 (no need voltage divider)
```

```
#include <SoftwareSerial.h>
SoftwareSerial BTserial(2, 3); // RX | TX
// Connect the HC-05 TX to Arduino pin 2 RX.
// Connect the HC-05 RX to Arduino pin 3 TX through a voltage divider.
int readMode = 10;   //  read toggle to switch mode
//int switchModePin = 10 ; // switch the relay
int mode = 0;

int sensorpin_a = A0; // pin that is connected to the IR sensor_A
int sensorpin_b = A1; // pin that is connected to the IR sensor_B
int sensorpin_c = A2; // pin that is connected to the IR sensor_C
int sensorpin_d = A3; // pin that is connected to the IR sensor_D
int sensorpin_e = A4; // pin that is connected to the IR sensor_E

int bondary_1 = 23;
int bondary_2 = 70;                                          ;

int trigValue =0;
int sns_A ; // sensor A on the dock, hold the distance value
int sns_B ; // sensor B on the dock, hold the distance value
int sns_C ; // sensor C on the dock, hold the distance value
int sns_D ; // sensor D on the dock, hold the distance value
int sns_E ; // sensor E on the dock, hold the distance value

int aa =0 ;   // determine if each sensor is triggered
int bb =0;
int cc =0;
int dd =0;
int ee =0;

char commend;
int B_position;

int error = 3;
int error_p = 6;

void setup()
{
    // start the serial communication with the host computer
    Serial.begin(9600);
    Serial.println("Arduino with HC-05 is ready");
   pinMode(readMode, INPUT);
 //  pinMode(switchModePin, OUTPUT);
    // start communication with the HC-05 using 9600
```

```
    BTserial.begin(9600);
    Serial.println("BTserial started at 9600");
}

void loop()
{

  mode = digitalRead(readMode);     // read the input pin
  if (mode == HIGH){
        BTserial.write('s');
        Serial.println("s");
        delay(500);
    }

 else {

   getIR_Value();
   sensorTrig();
   trigValue = aa + bb + cc + dd + ee;
   sendCommend(trigValue);
   /*---------------------------------------- For testing ---------------------------------------- */
 //  printActualDistance();    // for testing distance of each sensor
  // printTrigVal();            // for testing if each sensor is reading values
    printPosistion();         //
   /*-------------------------------------------------------------------------------------------- */
       Serial.print("trigValue is: ");
       Serial.println(trigValue);
       delay(1000);

 }
}

void getIR_Value(void){

  float sensorvalue_a = analogRead(sensorpin_a);
  float sensorvalue_b = analogRead(sensorpin_b);
  float sensorvalue_c = analogRead(sensorpin_c);
  float sensorvalue_d = analogRead(sensorpin_d);
  float sensorvalue_e = analogRead(sensorpin_e);

  float sensorvalue_a2 = analogRead(sensorpin_a);
  float sensorvalue_b2 = analogRead(sensorpin_b);
  float sensorvalue_c2 = analogRead(sensorpin_c);
  float sensorvalue_d2 = analogRead(sensorpin_d);
```

```
  float sensorvalue_e2 = analogRead(sensorpin_e);

  float sensorvalue_a3 = analogRead(sensorpin_a);
  float sensorvalue_b3 = analogRead(sensorpin_b);
  float sensorvalue_c3 = analogRead(sensorpin_c);
  float sensorvalue_d3 = analogRead(sensorpin_d);
  float sensorvalue_e3 = analogRead(sensorpin_e);

  sns_A = convertToDistance(sensorvalue_a,sensorvalue_a2,sensorvalue_a3 );
  sns_B = convertToDistance(sensorvalue_b,sensorvalue_b2, sensorvalue_b3);
  sns_C = convertToDistance(sensorvalue_c, sensorvalue_c2, sensorvalue_c3);
  sns_D = convertToDistance(sensorvalue_d, sensorvalue_d2, sensorvalue_d3);
  sns_E = convertToDistance(sensorvalue_e, sensorvalue_e2, sensorvalue_e3);
  }

int convertToDistance(float value, float value2, float value3 ){
    float cm1 = 10650.08*pow(value,-0.935) -10;
    int distance1 = roundf(cm1);
    float cm2 = 10650.08*pow(value2,-0.935) -10;
    int distance2 = roundf(cm2);
    float cm3 = 10650.08*pow(value3,-0.935) -10;
    int distance3 = roundf(cm3);
    int distance = (distance1 + distance2 + distance3 )/3;
     return distance;

    }

int sendCommend(int trigVal){
  int temp =0;
  switch (trigVal){

    case 0:  //  0 0 0 0 0
        BTserial.write('n');    // go forward
        commend = 'N';
        B_position = 0;
       break;

    case 16:    //     1 0 0 0 0
        BTserial.write('f');    // go forward
        commend = 'f';
        B_position = 16;
      break;

     case 24:    //     1 1 0 0 0
```

```
  temp = sns_A - sns_B;
  if (abs(temp) <= error){
  BTserial.write('f');    // go forward
  commend = 'f';
  B_position = 24;
  }
  else if(abs(temp) > error && temp > 0){   // turn right
    BTserial.write('r');
  commend = 'r';
  B_position = 24;
    }
  else if(abs(temp) > error && temp < 0){
    BTserial.write('l');     // turn left
    commend = 'l';
  B_position = 24;
  }
 break;

case 28:    //     1 1 1 0 0
  temp = sns_A - sns_C;
  if (abs(temp) <= error){
  BTserial.write('f');    // go forward
  commend = 'f';
  B_position = 28;
  }
  else if(abs(temp) > error && temp > 0){   // turn right
    BTserial.write('r');
    commend = 'r';
  B_position = 28;
    }
  else if(abs(temp) > error && temp < 0){
    BTserial.write('l');     // turn left
    commend = 'l';
  B_position = 28;
  }
 break;

 case 30:    //     1 1 1 1 0
  temp = sns_A - sns_D;
  if (abs(temp) <= error){
  BTserial.write('p');    // go forward
  commend = 'p';
  B_position = 30;
  }
```

```
  else if(abs(temp) > error && temp > 0){   // turn right
    BTserial.write('r');
   commend = 'r';
  B_position = 30;
    }
  else if(abs(temp) > error && temp < 0){
    BTserial.write('l');     // turn left
  commend = 'l';
  B_position = 30;
  }

break;

 case 31:    //     1 1 1 1 1
  temp = sns_A - sns_E;
  if (abs(temp) <= error_p){
    BTserial.write('p');    // go forward
    commend = 'P';
  B_position = 31;
  }
  else if(abs(temp) > error_p && temp > 0){   // turn right
    BTserial.write('p');
   commend = 'p';
  B_position = 31;
    }
  else if(abs(temp) > error_p && temp < 0){
    BTserial.write('p');     // turn left
   commend = 'p';
  B_position = 31;
  }
break;

  case 15:    //     0 1 1 1 1
  temp = sns_B - sns_E;
  if (abs(temp) <= error){
  BTserial.write('p');    // go backward
  commend = 'p';
  B_position = 15;
  }
  else if(abs(temp) > error && temp > 0){   // turn right
    BTserial.write('r');
  commend = 'r';
  B_position = 15;
    }
```

```
  else if(abs(temp) > error && temp < 0){
    BTserial.write('l');     // turn left
  commend = 'l';
  B_position = 15;
  }
break;

  case 7:    //     0 0 1 1 1
  temp = sns_C - sns_E;
  if (abs(temp) <= error){
  BTserial.write('b');    // go backward
   commend = 'b';
  B_position = 7;
  }
  else if(abs(temp) > error && temp > 0){   // turn right
    BTserial.write('r');
  commend = 'r';
  B_position = 7;
    }
  else if(abs(temp) > error && temp < 0){
   BTserial.write('l');     // turn left
  commend = 'l';
  B_position = 7;
  }
break;

   case 3:    //     0 0 0 1 1
  temp = sns_D - sns_E;
  if (abs(temp) <= error){
  BTserial.write('b');    // go backward
  commend = 'b';
  B_position = 3;
  }
  else if(abs(temp) > error && temp > 0){   // turn right
    BTserial.write('r');
    commend = 'r';
  B_position = 3;
    }
  else  if(abs(temp) > error && temp < 0){
    BTserial.write('l');     // turn left
  commend = 'l';
  B_position = 3;
  }
break;
```

```
      case 1:    //     0 0 0 0 1
       BTserial.write('b');    // go backward
              commend = 'b';
       B_position = 1;
      break;

    default:
     break;

  }

  }

void sensorTrig(void){
  if ((sns_A > bondary_1) && (sns_A < bondary_2)){
    aa = 16; // 1 0 0 0 0
    }
    else
    aa =0;

      if ((sns_B > bondary_1) && (sns_B < bondary_2)){
        bb = 8; // 0 1 0 0 0
        }
         else
        bb =0;

          if ((sns_C > bondary_1) && (sns_C < bondary_2)){
           cc = 4; // 0 0 1 0 0
           }
           else
           cc =0;

            if ((sns_D > bondary_1) && (sns_D < bondary_2)){
             dd = 2;  // 0 0 0 1 0
              }
               else
                dd =0;

               if ((sns_E > bondary_1) && (sns_E < bondary_2)){
                ee = 1; // 0 0 0 0 1
                 }
                 else
                 ee =0;
```

```
 }

void printActualDistance(void){
      Serial.print("Value of A is: ");
      Serial.println(sns_A);
      Serial.print("Value of B is: ");
      Serial.println(sns_B);
      Serial.print("Value of C is: ");
      Serial.println(sns_C);
      Serial.print("Value of D is: ");
      Serial.println(sns_D);
      Serial.print("Value of E is: ");
      Serial.println(sns_E);
      Serial.println("////////////////////////////////////////////////////////////////");
        Serial.println("                                                              ");

 }

void printTrigVal(void){
      Serial.print("Value of aa is: ");
      Serial.println(aa);
      Serial.print("Value of bb is: ");
      Serial.println(bb);
      Serial.print("Value of cc is: ");
      Serial.println(cc);
      Serial.print("Value of dd is: ");
      Serial.println(dd);
      Serial.print("Value of ee is: ");
      Serial.println(ee);
      Serial.println("////////////////////////////////////////////////////////////////");
        Serial.println("                                                              ");

 }

void printPosistion(void){
      Serial.print("Current Posistion =  ");
      Serial.println(B_position,BIN);
      Serial.print("Commend to Blutooth =  ");
      Serial.println(commend);
      Serial.println("////////////////////////////////////////////////////////////////");
      Serial.println("                                                            ");

 }
```