## Contract No:

This document was prepared in conjunction with work accomplished under Contract No. DE-AC09-08SR22470 with the U.S. Department of Energy (DOE) Office of Environmental Management (EM).

## Disclaimer:

This work was prepared under an agreement with and funded by the U.S. Government. Neither the U. S. Government or its employees, nor any of its contractors, subcontractors or their employees, makes any express or implied:

1 ) warranty or assumes any legal liability for the accuracy, completeness, or for the use or results of such use of any information, product, or process disclosed; or

2 ) representation that such use or results of such use would not infringe privately owned rights; or

3) endorsement or recommendation of any specifically identified commercial product, process, or service.

Any views and opinions of authors expressed in this work do not necessarily state or reflect those of the United States Government, or its contractors, or subcontractors.

We put science to work.™

Savannah River National Laboratory™

OPERATED BY SAVANNAH RIVER NUCLEAR SOLUTIONS

A U.S. DEPARTMENT OF ENERGY NATIONAL LABORATORY • SAVANNAH RIVER SITE • AIKEN, SC

# Autonomous Sampling Platform Development

## Radiological Contamination Mapping at SRS

Nicholas Moya

July 2016

SRNL-STI-2016-00422, Revision 0

SRNL.DOE.GOV

## DISCLAIMER

This work was prepared under an agreement with and funded by the U.S. Government. Neither the U.S. Government or its employees, nor any of its contractors, subcontractors or their employees, makes any express or implied:

1. warranty or assumes any legal liability for the accuracy, completeness, or for the use or results of such use of any information, product, or process disclosed; or
2. representation that such use or results of such use would not infringe privately owned rights; or
3. endorsement or recommendation of any specifically identified commercial product, process, or service.

Any views and opinions of authors expressed in this work do not necessarily state or reflect those of the United States Government, or its contractors, or subcontractors.

**Printed in the United States of America**

**Prepared for**
**U.S. Department of Energy**

**Keywords:** *Boat Drone, Contamination, PID, Plant, Controller, Stability,*

**Retention:** *Permanent*

# Autonomous Sampling Platform Development

## Radiological Contamination Mapping at SRS

Nicholas Moya

July 2016

Prepared for the U.S. Department of Energy under contract number DE-AC09-08SR22470.

Savannah River National Laboratory™
OPERATED BY SAVANNAH RIVER NUCLEAR SOLUTIONS

# REVIEWS AND APPROVALS

AUTHORS:

Nicholas Moya, SRNL Intern Date

Dr. Tad Whiteside, Principle Investigator Date

APPROVAL:

David Crowley, Manager Date

# ACKNOWLEDGEMENTS

This project would not have been done were it not for the mathematical model of the boat by Mosharrof Hossain Sarker, whom I congratulate and thank. I would also like to extend a thank you to one of my professors at UNLV, Dr. Brenan Morris who recommended me to this internship opportunity. He is truly one of the finest professors at UNLV because of his diligent dedication to help elevate every student he can. This paper is complemented by the work of David Nasol whose work on measuring radioactive contaminants gave rise to the very necessity of a boat drone. I would also like to acknowledge SRNL and all the people I've met here who have welcomed me and helped to make SRNL feel like a second home. Lastly, but most significantly, I would like to extend a sincere thank you to my mentor in this project, Dr. Tad Whiteside. Dr. Whiteside has aided me when I was stuck, advised me when I thought I was done, and encouraged me all along the way with his friendly, practical, and genuine demeanor. It is because he took a chance on me, that I have been able to forward my career, advance my education, and improve my future.

# ABSTRACT

From 1961 to 1964, radioactive elements were released from the Savannah River Site into local bodies of water via cooling water charges from the reactors on site. In 1983, the extent of the radioactive contamination was first studied and elements such as $^{137}$Cs, $^{90}$Sr, $^{238}$Pu, $^{241}$Am, $^{244}$Cm, and tritium were found to have seeped from local bodies of water into sediment and the surrounding flora and fauna. The current method of tracking and monitoring radioactive contamination at the SRS is to gather samples and conduct measurements in a laboratory. A cheaper, and safer, method to conduct such measurements would be to automate the process by using an autonomous boat that can travel to locations, conduct measurements, and return home all without human intervention. To introduce this idea, the construction of an autonomous boat prototype was completed to demonstrate the practicality and feasibility of such an idea. The prototype travels to a set of waypoints, stops at each waypoint, and returns when all waypoints have been reached. It does this by employing a simple battery-powered boat with an Arduino controller that steers the boat using a steering algorithm incorporated into a Proportional Integral Derivative (PID) function. A total of three tests were conducted at two different bodies of water and after working out some hardware problems, the boat drone was able to successfully steer and reach all programmed waypoints. With the prototype complete, the next steps to realizing the final product of the boat drone will include adopting a processing unit with higher bit architecture, using a bigger boat with a more powerful trolling motor, and incorporating a solar panel for continuous power and round-the-clock performance.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF ABBREVIATIONS

| | |
|---|---|
| SRNL | Savannah River National Laboratory |
| PWM | Pulse Width Modulation |
| DC | Direct Current |
| PID | Proportional Integral Derivative |
| MOSFET | Metal Oxide Semiconductor Field Effect Transistor |
| AWG | American Wire Gauge |

# 1.0 Introduction

1.1 Problem Identified

The Savannah River Site was built in the 1950's for the refinement of nuclear materials to be used in the deployment of nuclear weapons. From 1961 to 1964, radioactive elements were released from the cooling water charges of the reactors into local bodies of water. The effects of this radioactive contamination were first studied in the 1983 where it was found that the contamination had spread from the bodies of water into the local flora and fauna and even seeped into the sediment. It became apparent that the contamination had to be monitored so that its spread would be limited and thus its damage mitigated.

1.2 Current Solution

Current monitoring of radioactive contamination involves sampling water and sediment in the bodies of water of interest. A grid system and shoreline perimeter is established so sampling locations can be randomized without bias. However, the task of sampling so many locations is time consuming, expensive, and laborious. Sample scheduling is limited by work days, good weather, and personal availability. Human error too becomes an issue as long hours in arduous work conditions can lead to safety concerns and sampling errors. Thus, this current solution for contamination monitoring and sampling is found to be costly and inconvenient.

1.3 New solution

A new solution to this problem would be to replace the human with a simple, low cost, autonomous boat that monitors a body of water for radioactive contaminants. We will call this autonomous boat our Boat Drone. The Boat Drone would be preprogrammed by a user to travel to a sequence of locations (waypoints), once at a waypoint it would conduct measurements, or collect samples, of the body of water, or sediment, for radioactive contamination. After it has accomplished its desired task at a waypoint, it would travel to the next waypoint until all measurements have been concluded. With its objectives completed, it would return to its preprogrammed base-station.

By automating the entire processes, scheduled monitoring becomes very flexible; a body of water can be monitored anytime, and for however long, as opposed to when a person is available. This means that monitoring times can be expanded to include nights, weekends, and other off-hour times. Not only that, but the Boat Drone will be much more cost efficient; instead of paying someone to monitor every time it is needed, the Boat Drone can be equipped with a solar panel so it can monitor continuously with fuel costs severely mitigated. With the exception of first time assembly costs, the Boat Drone will soon pay for itself in scheduling flexibility and negligible fuel expense. This makes a Boat Drone solution not only reliable but sustainable and practical as well.

The end product of this Boat Drone will be about the size of a John boat, use a trolling motor for speed and steering, and come equipped with a solar panel for mobile power. However, the aim of this project is not to produce a fully equipped Boat Drone, but instead of smaller a simpler Boat Drone prototype to prove that the idea is practical and achievable. As such, the aim of this prototype is to autonomously travel to a set of waypoints preprogrammed in by a user. With the success of this prototype, the idea can be refined and scaled up to match the desired end product of the Boat Drone.

## 2.0 System Structure

The task of autonomously steering and traveling to multiple waypoints in a sequence is accomplished through three primary modules; sensors, processors, and actuators. Each of these modules has multiple parts within themselves necessary for operation. What follows is a list of the components of each module which includes their name, function, and effect on the behavior of the Boat Drone.

### 2.1 Sensors

Sensors collect data about a systems environment that can be sent to the processors to do calculations and make decisions. For the Boat Drone system, two pieces of data must be collected; its current position, and its current bearing.

To determine its current position, it uses the Adafruit Ultimate GPS sensor which can link to multiple satellites to get a fix on its current location. Once a fix is obtained, it can relay information about its current position (latitude and longitude), its elevation, nautical speed (knots), Greenwich Time, and other info. However, the Boat Drone only needs data about its current location in latitude and longitude so that is all that is parsed and sent to the processor. The processor uses this data to determine how far from a waypoint it is and what direction it should go to reach that waypoint. A picture of this GPS sensor can be seen in figure 1.

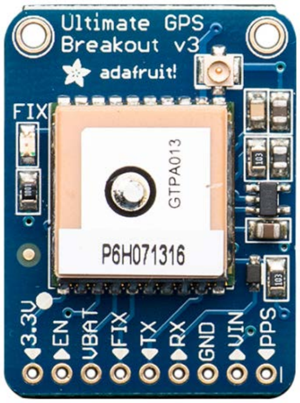


**Fig. 1 – The GPS Sensor**

In order to detect its current bearing, the Boat Drone uses the Adafruit HMC5883L Breakout - Triple-Axis Magnetometer/Compass Sensor. This compass sensor returns information about the x, y, and z position to the Boat Drone. This information is then fed to the processor where it is used to determine its current bearing and help it steer until it faces the waypoint. A picture of the compass sensor can be seen in figure 2.

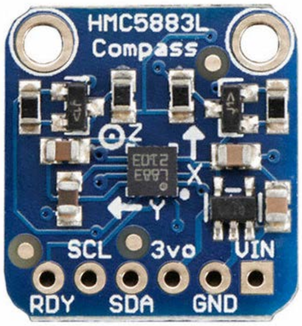


**Fig. 2 – The Compass Sensor**

2.2 Processor

Processors use data collected by the sensors to conduct calculations and make decisions. For the Boat Drone, a simple Arduino Uno R3 was all that was needed to provide more than enough processing power. The Arduino Uno R3 is really just a development board that houses the real processing power; the ATmega328P IC. The ATmega328P IC has adequate processing speed and multiple analog, digital, and Pulse Width Modulation (PWM) pins that make it more than sufficient for the Boat Drone. A list of specs can be seen below, as well as a picture of the Arduino Uno R3 housing the ATmega328P IC in figure 3.

- 32K Flash Program Memory
- 1K EEPROM
- 13 digital pins (6 of which are also PWM pins)
- 6 analog pins
- Vin pin for alternative power
- 5v pin to power other devices
- 20MHz processing speed

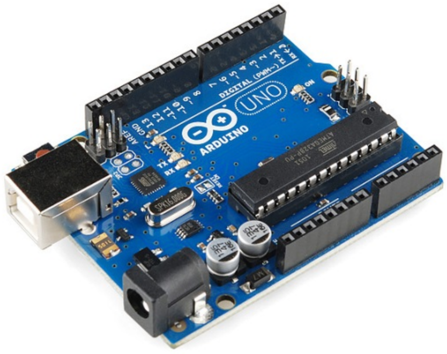


**Fig. 3 – Arduino Uno R3**

The Arduino takes in measurements by the sensors and uses that data to calculate the bearing, heading to the next waypoint, and distance to the next waypoint. Lastly, it sends PWM commands to the motor driver circuit to control the DC motors so that the boat steers and travels to the next waypoint.

2.3 Actuator

The actuator is the part of the system which acts on the environment to produce the desired behavior of the system. In the Boat Drone, the actuators are twin propellers, attached to DC motors, which in turn are controlled by a motor driver circuit. The motor driver circuit uses a PWM command from the Arduino to set the angular velocity of the DC motor which is translated to the linear velocity of the Boat Drone. By changing the angular velocity of one DC motor, with respect to the other DC motor, the Boat Drone can be turned and steered toward a desired heading. Thus the DC motors act on the system to both propel it forward and turn it from side to side. A picture of the motor driver circuit, the DC motors, and the propellers are provided in figures 4, 5, and 6, respectively.

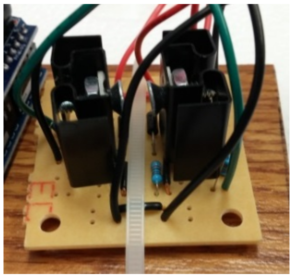


**Fig. 4 – Motor Driver Circuit**

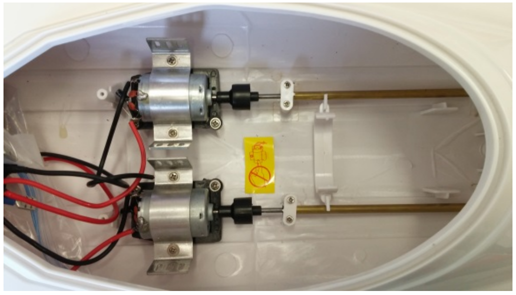


**Fig. 5 – DC Motors**

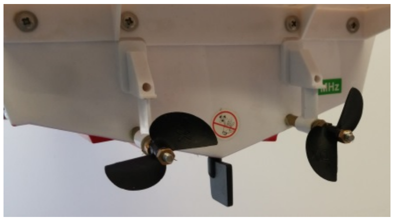


**Fig. 6 – Propellers**

Of course, a power source and chassis is also required for the Boat Drone. This prototype only used a simple RC boat as its chassis, with all of the internal electronics gutted out. The power supply is provided by a 7.2v 1700mAh NiCd battery. However this only yields 6 to 10 minutes of run time so a battery with more milli Amp Hour capacity is recommended for field deployment. A picture of the RC boat and battery are provided in figures 7 and 8, respectively.

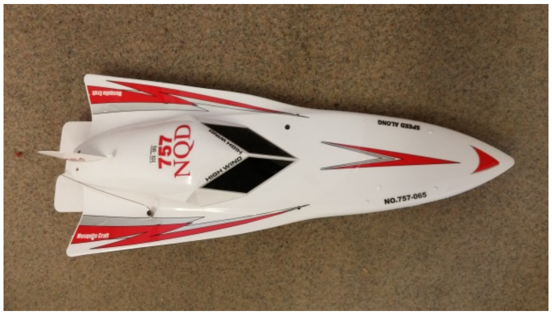


**Fig. 7 – RC Boat Chassis**

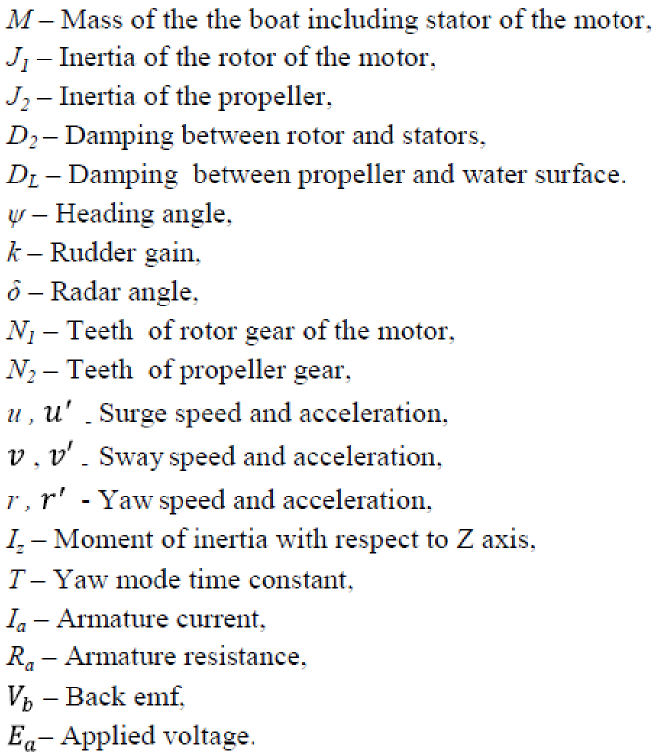
**Fig. 8 - Battery**

## 3.0 Mathematical Model

It's relatively straight forward to derive algorithms to program into the Arduino to control how the Boat Drone steers. However, the algorithms have to be imbedded into a controller which in turn must act on a simulated, or modeled, representation of the Boat Drone. Thus a mathematical model of the boat must first be derived before we can design and program the controller which will act on it. The following model is derived under the following assumptions:

1. The boat is small
2. The waves have a negligible effect on the boat's sideways motion (sway)
3. The boat's up and down motion (heave) is not considered
4. Only small rudder angles are considered
5. Acceleration is constant

This derivation is provided by the paper *Modeling and Analysis of Control System for a Multi-Robotic System* by Md. Mosharrof Hossain Sarker, et al., and is listed in the reference page of this report. Figure 9 lists all of the symbols and variables that will be used in the derived transfer equations.



$M$ – Mass of the the boat including stator of the motor,
$J_1$ – Inertia of the rotor of the motor,
$J_2$ – Inertia of the propeller,
$D_2$ – Damping between rotor and stators,
$D_L$ – Damping between propeller and water surface.
$\psi$ – Heading angle,
$k$ – Rudder gain,
$\delta$ – Radar angle,
$N_1$ – Teeth of rotor gear of the motor,
$N_2$ – Teeth of propeller gear,
$u$ , $u'$ - Surge speed and acceleration,
$v$ , $v'$ - Sway speed and acceleration,
$r$ , $r'$ - Yaw speed and acceleration,
$I_z$ – Moment of inertia with respect to Z axis,
$T$ – Yaw mode time constant,
$I_a$ – Armature current,
$R_a$ – Armature resistance,
$V_b$ – Back emf,
$E_a$– Applied voltage.

**Fig. 9 – Symbols and their meaning**

3.1 Motion Analysis

Looking at figure 10, it can be seen that a boat has six degrees of freedom, however, we will only consider three; surge, sway, and yaw. Figure 11 provides the coordinate system from which we can reference our variables and derive transfer equations.

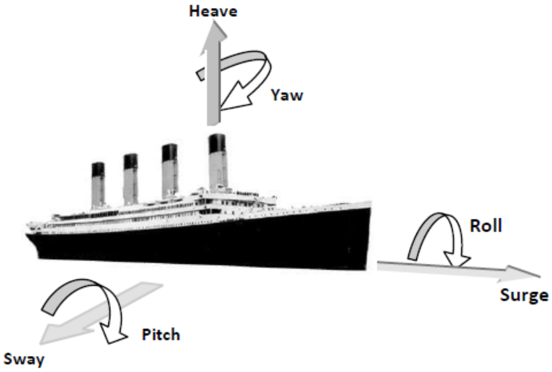


**Fig. 10 – Degrees of Freedom**

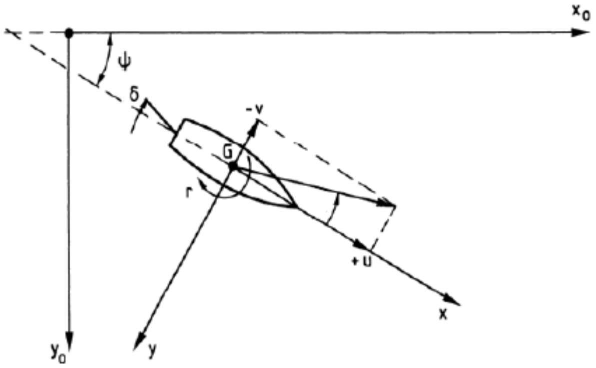


**Fig. 11 – Coordinate System**

One such transfer equation derived by Sarker relates a boats position in the x, y reference frame to the rudder angle. In actuality, it is more accurate to describe this as a turning angle since we do not use a rudder to steer the Boat Drone but instead use different speeds of the two DC motors.

$$\frac{(x, y)}{\delta} = \frac{k(v, u)}{s^2(1 + Ts)}$$

3.2 Mechanical Analysis

The interaction of forces between the boat and the water can be modeled as a mechanical system as shown in figure 12. From this model, a transfer function is derived by Sarker which relates input of force with the output of linear displacement.

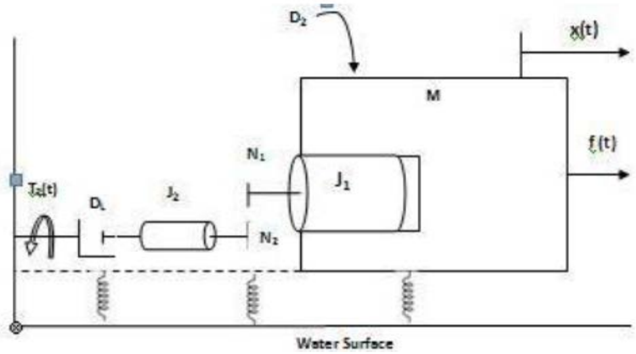


**Fig. 12 – Mechanical model of boat in water**

Setting

$$J_{eq} = J_1 + J_2 \left(\frac{N_1}{N_2}\right)^2$$

$$T_1 = T_2 (\frac{N_1}{N_2})$$

$$D_{eq} = D_L (\frac{N_1}{N_2})^2$$

We can derive the following transfer functions for force related to position and torque related to angular position:

$$\frac{x(s)}{F_a(s)} = \frac{-\left(\frac{J_{eq}}{kr}s^2 + \frac{(D_2 + D_{eq})}{kr}s\right) + \frac{D_2 s}{r}}{M\frac{J_{eq}}{kr}s^4 + D_2\frac{J_{eq}}{kr}s^3 + \frac{k_{eq}J_{eq}}{kr}s^2 + \left(\frac{D_2 + D_{eq}}{Kr}\right)Ms^3 + \left(\frac{D_2 + D_{eq}}{Kr}\right)D_2 s^2 + \left(\frac{D_2 + D_{eq}}{Kr}\right)k_{eq}s - \frac{D_2^2}{kr}s^2}$$

3.3 Electro-Mechanical Analysis

Lastly, the electro-mechanical part of the Boat Drone system is represented by the DC motors which relate an input of voltage to an output of angular velocity. Modeling DC motors is an established academic practice and as such has a classically derived transfer function. Instead of using this transfer function, we will instead use the Laplace transforms of the equations that are used to derive the transfer function of a DC motor and substitute our own variables from the mechanical transfer function. This will yield the complete transfer function which we can use to model the Boat Drone.

From the previous equations:

$$X(s) = \frac{\theta(s)[D_2 s] + F(s)}{Ms^2 + D_2 s + K_{eq}} = \frac{\theta(s)[D_2 s] - \frac{T_1}{r\,cos\,\theta_1}}{Ms^2 + D_2 s + K_{eq}}$$

And

$$\theta(s)[J_{eq}s^2 + (D_2 + D_{eq})s] - \frac{\theta(s)[D_2 s]^2 - \frac{T_1}{r\,cos\,\theta_1}[D_2 s]}{Ms^2 + D_2 s + K_{eq}} = T_1(s)$$

$$\theta(s)\left[\frac{(J_{eq}s^2 + (D_2 + D_{eq})s)(Ms^2 + D_2 s + K_{eq}) - (D_2 s)^2}{Ms^2 + D_2 s\left(1 + \frac{1}{K}\right) + K_{eq}}\right] = T_1(s)$$

Now, using the transfer functions for DC motors:

$$R_a I_a(s) + L_a s I_a(s) + V_b(s) = E_a(s)$$

$$\frac{(R_a + L_a s)T_1(s)}{K_t} + K_b s\theta(s) = E_a(s)$$

$$T_1(s) = \frac{K_t(E_a(s) - K_b s\theta(s))}{R_a}$$

Putting this all together, we arrive at the transfer function which we will use to represent the Boat Drone.

$$\frac{\Theta(s)}{E_a(s)} = \frac{Ms^2 + D_2 s(1 + \frac{1}{K}) + K_{eq}}{\frac{R_a}{K_t}[\{J_{eq}s^2 + (D_2 + D_{eq}s)\}\{(M - D_2)s^2 + D_2 s + K_{eq}\}] + K_b s}$$

### 3.4 Stability Response

Before a controller can be designed, it is important to look at the stability of the open loop system, or, the system with only the plant itself. For our purposes, this means looking at the stability of the Boat Drone without a controller. The stability of the plant is of interest because it determines how we design and tune the controller. To look at the open loop stability of the system, we choose reasonable values for the parameters in the electro-mechanical transfer function and program it into simulation software. In this case, we will program it into Simulink so that we can generate various plots to determine its stability. These plots are provided in the figures below.

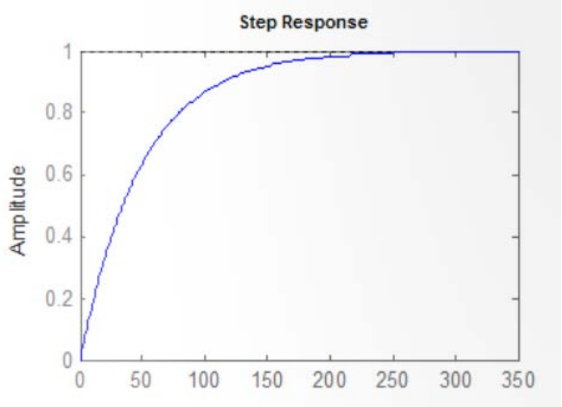


**Fig. 13 – Step Response**

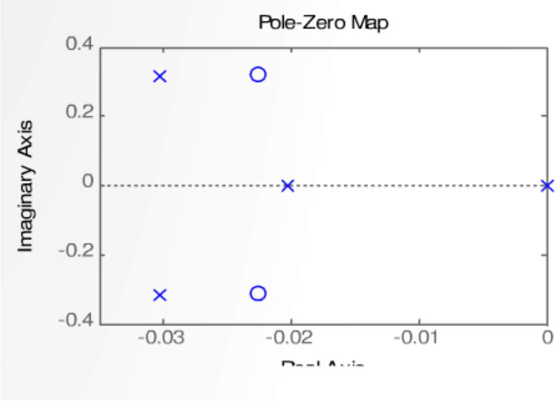


**Fig. 14 – Pole-Zero Map**

We can see that the step response has no overshoot, oscillations, nor steady state error but what is evident is the enormous rise time. The bottom axis is time in seconds.

It can be seen that every pole is in the left hand plane, making the plant stable. The bottom axis is Real Axis.

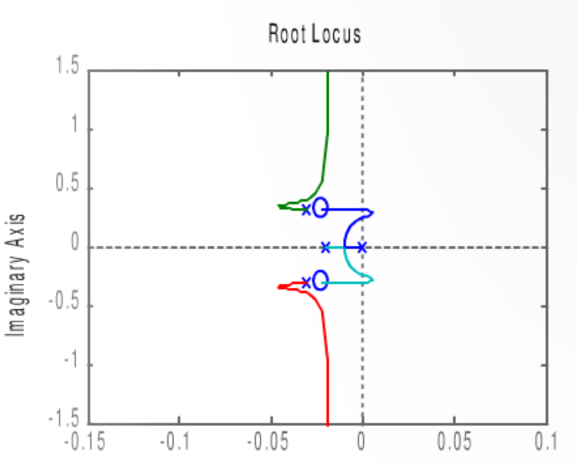


**Fig. 15 – Root Locus**

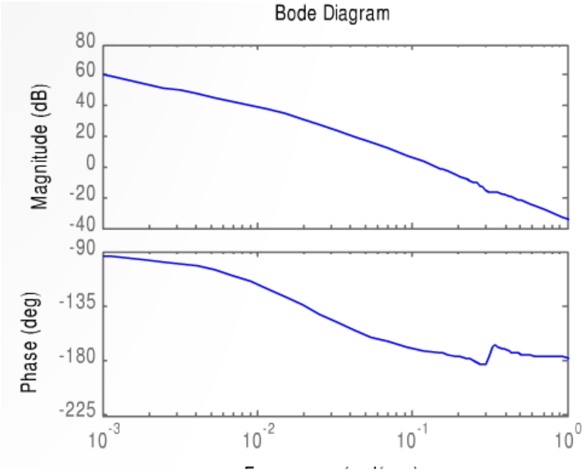


**Fig. 16 – Bode Diagram**

Here we see that increasing the gain can result in unstable behavior but the majority of gain values remain in the stable region. The bottom axis is Real Axis.

Here too, we find a reasonable gain margin for stability; about -20dB The phase margin, however, is dangerously close to 0°. The bottom axis is frequency in Hertz.

So, even with a large rise time in the step response and a phase margin dangerously close to 0° in the Bode plot, the open loop system remains stable. This means that we can use a PID controller, tuned with the Nichols Ziegler method, to control the Boat Drone.

## 4.0 Controller Design

The controller will act on the plant so that the system as a whole produces the desired behavior, which is measured by the step response of the system. Once the plant is modeled, a controller is designed, tuned and implemented into the system. Lastly, the new response of the system is simulated to verify that the controller behaves as expected.

### 4.1 Plant

As previously noted, the plant is stable, which means we are free to design almost any controller for the Boat Drone system. The one we will choose to use is called a PID controller. This controller is chosen because it’s simple to program and easy to tune. However, before we can start designing the controller, the plant must be changed from continuous time domain to discrete time domain. This is necessary because the sensors, namely the GPS, are not continuous and run off of a sampling time. In the case of the GPS, the data is parsed 10 times every second meaning that it has a sampling period of 0.1. A transfer function can be transformed from continuous time to discrete time by several methods but the one used here is called the bilinear transform, also known as Tustin’s method. The transformation is given below:

$$s = \frac{2}{T}\frac{(z-1)}{(z+1)}$$

Using this transformation, the plant, and thus the system, changes from continuous time to discrete time, this in turn allows a discrete controller to be designed.

### 4.2 Controller

A PID controller is based off of an equation which can be split into an algorithm and easily programmed. The equation for a PID controller is given below.

$$u[k] = u[k-1] + K_1 e[k] + K_2 e[k-1] + K_3 e[k-2]$$

Where

u[k] – Control Variable
u[k-1] – Previous Control Variable
$K_1 = K_p + K_i + K_d$
$K_2 = -K_p -2*K_d$
$K_3 = K_d$
e[k] – Error
e[k-1] – Previous Error
e[k-2] – Error Two Discrete Units of Time Ago
K – Discrete Unit of Time

Error is defined as the difference between the ideal value and the actual value, which for the Boat Drone means the difference between the heading and the bearing. The control variable is the value fed back into the system to make the error closer to zero. In the Boat Drone, u acts as the variable that helps set the speed of the motors which in turn help change its bearing to match its heading. So, the only unknown parameters are the $K_p$, $K_i$, and $K_d$ terms (PID parameters) which have to be tuned to match the plant. One such way of tuning PID parameters is called the Ziegler–Nichols method.

4.3 Tuning

The Ziegler–Nichols method is a reliable method for tuning PID parameters and can be applied in two scenarios; the step response is stable (called the step-response method), or the step response is astable (called the ultimate sensitivity method). In this case, the step response is stable so we will use the step-response method. This method is very straightforward; first we look at the step response of the open loop system (the system with just the plant and no controller). Then the time before it starts to rise is measured (L in figure 17) and the slope R is measured. Lastly, setting a = LR, we plug our values into the table in figure 18 to determine the PID parameters. When this is done for the Boat Drone, the following PID parameters are calculated. These are the values programmed into the controller algorithm which controls the steering behavior of the Boat Drone.

$$K_p = 1.292$$
$$K_i = 1.113$$
$$K_d = -0.282$$

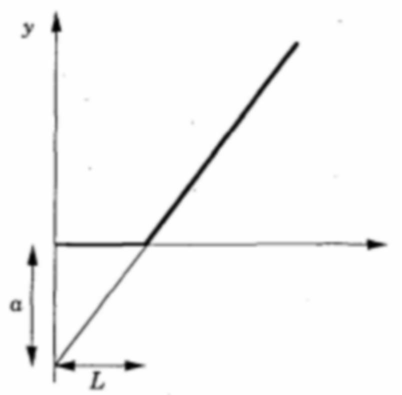
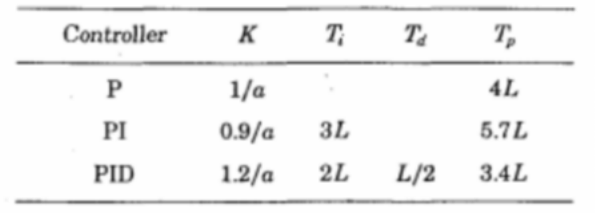


**Fig. 17 – Step Response Measurements**



| Controller | $K$ | $T_i$ | $T_d$ | $T_p$ |
|---|---|---|---|---|
| P | $1/a$ | | | $4L$ |
| PI | $0.9/a$ | $3L$ | | $5.7L$ |
| PID | $1.2/a$ | $2L$ | $L/2$ | $3.4L$ |

**Fig. 18 – PID Tuning Chart**

4.4 Stability Response

With the controller tuned to the plant, the system is complete. The last, but most important, objective is to check the step response of the closed loop system to determine how the step response has improved. Using Simulink, we can simulate the system and plot the step response. The closed loop system is given in figure 19 and the new step response is plotted in figure 20.

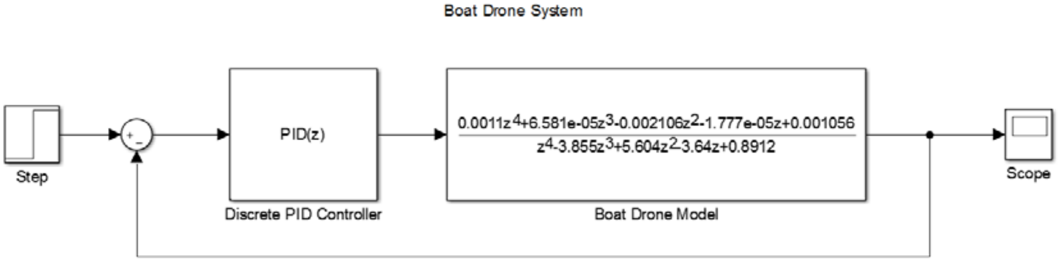

Boat Drone System
Step
+
-
PID(z)
Discrete PID Controller
$\frac{0.0011z^4+6.581e\text{-}05z^3-0.002106z^2-1.777e\text{-}05z+0.001056}{z^4-3.855z^3+5.604z^2-3.64z+0.8912}$
Boat Drone Model
Scope

**Fig. 19 – Closed Loop Boat Drone System**

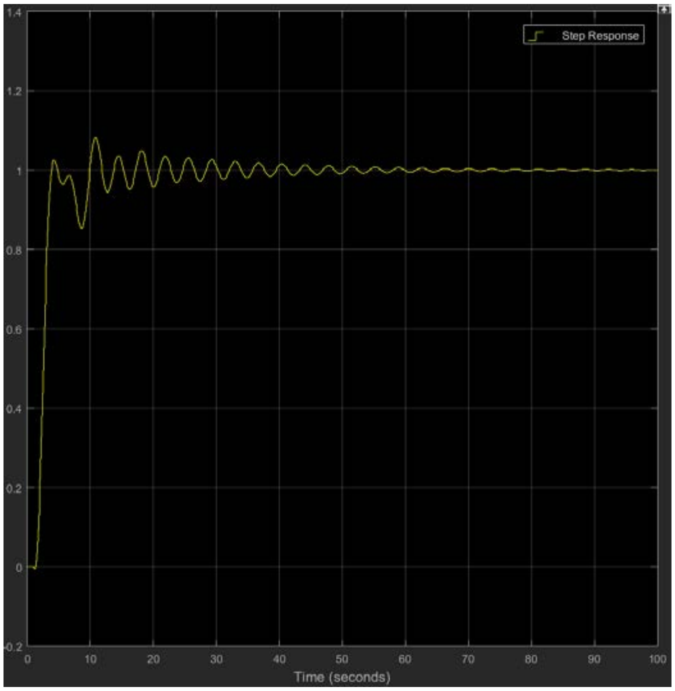


**Fig. 20 – Step Response of the Boat Drone System**

As seen in the step response, the rising time is dramatically improved and at the cost of minimal overshoot and negligible steady state error. Armed with the stable simulation results, it is now time to experimentally test the Boat Drone in a body of water so as to determine the true effectiveness of the Boat Drone prototype.

## 5.0 Results

Once all the parts had been assembled and tested, both individually and when integrated, the boat drone was ready to be tested in the water. We wanted to test if the boat could successfully steer itself from one waypoint to the next, to demonstrate that the controller was working as designed. A total of three tests we conducted; two at Gem Lakes, Aiken, SC and the other at Langley Pond, Aiken, SC. In the first test, the boat steered perfectly to its first waypoint, but after a couple of seconds one of the wires melted off, cutting the connection to the battery and stopping the boat. After soldering the motor controller circuit with thicker, solid gauge wires, we tried to test again but one of the propellers fell off during the second test, making us unable to continue. After waiting several days for another boat to arrive, we finally conducted the third and last test. Here the boat worked as intended; it steered to the first waypoint, stopped, and steered to the other waypoint. It actually overshot its last waypoint because one of the propellers became tangled in some seaweed, slightly diminishing its steering precision.

The results show that for each test, the controller worked just as designed and any failures present were caused by the hardware of the cheap RC boats, not the circuity. During these tests, the boat was able to reach all waypoints with the accuracy of one or two meters; much more accurate than the 11 meters accuracy that was estimated. Because we were testing the ability of the boat to reach multiple waypoints, this prototype can be considered to meet all expectations as it was able to steer and travel to multiple waypoints.

## 6.0 Future Work

With the success of this prototype, we look to making improvements on the original design and expanding to satisfy the requirements of the finished product. In the original prototype, an Arduino is programmed as the controller of the boat, and while the Arduino is an uncomplicated tool for programming, it is limited by its 8 bit architecture, which in turn limits the precision of the GPS to 6 decimal spaces, equating to an accuracy of 11 meters on the ground. Switch the Arduino for a computationally stronger device with higher architecture, such as a BeagleBone, would improve the accuracy of the boat drone. Another improvement will be to use a bigger boat with a stronger motor. To this effort, we have already acquired a Jon Boat sized vessel which will have a trolling motor attached to provide stronger propulsion in future seaweed filled bodies of water. Lastly, a solar panel and charger system will be incorporated so that the boat drone will be able to conduct measurements and collect samples all day and night until it needs to recharge. It will be the future work of those who continue this project to implement these changes so that the boat drone can be fully realized.

## 7.0 References

Åström, Karl J. and Hagglund, T. *PID Controllers, 2nd ed*. International Society of Automation, Triangle, NC. 1995.

Åström, Karl J. and Wittenmark, Björn. *Computer-Controlled Systems: Theory and Design, 3rd Ed*. Dover Publications, Inc. Mineola, New York. 2011.

Sarker, Mosharrof Hossain et al. *Modeling and Analysis of Control System for a Multi-Robotic System.* International Journal of Intelligent Control and Systems, Vol. 14, No. 4, Pg. 221-227. December 2009.

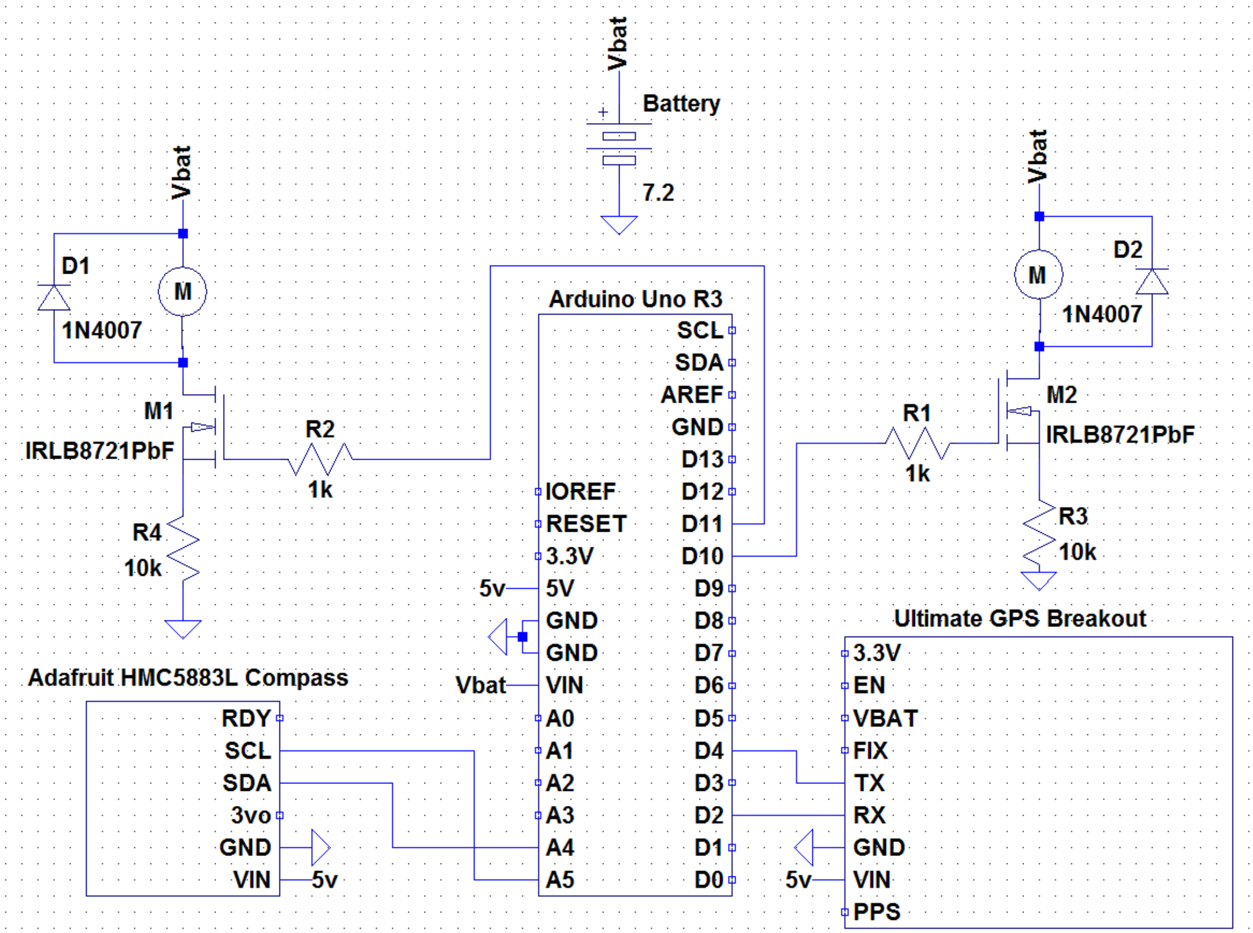
*Savannah River Site Environmental Report 2014*. Savannah River Nuclear Solutions, LLC. 2014. Savannah River Site Aiken, South Carolina, USA.

Vlad Pomogaev. "Boat Autopilot." Instructables. May 25th, 2015. Autodesk Inc. June 6th, 2016. <http://www.instructables.com/id/Boat-Autopilot/>.

Whicker, F. Ward et al. *Distribution of Long-Lived Radionuclides in an Abandoned Reactor Cooling Reservoir*. Ecological Monographs by the Ecological Society of America, Pg. 471-496. Washington DC. 1990.

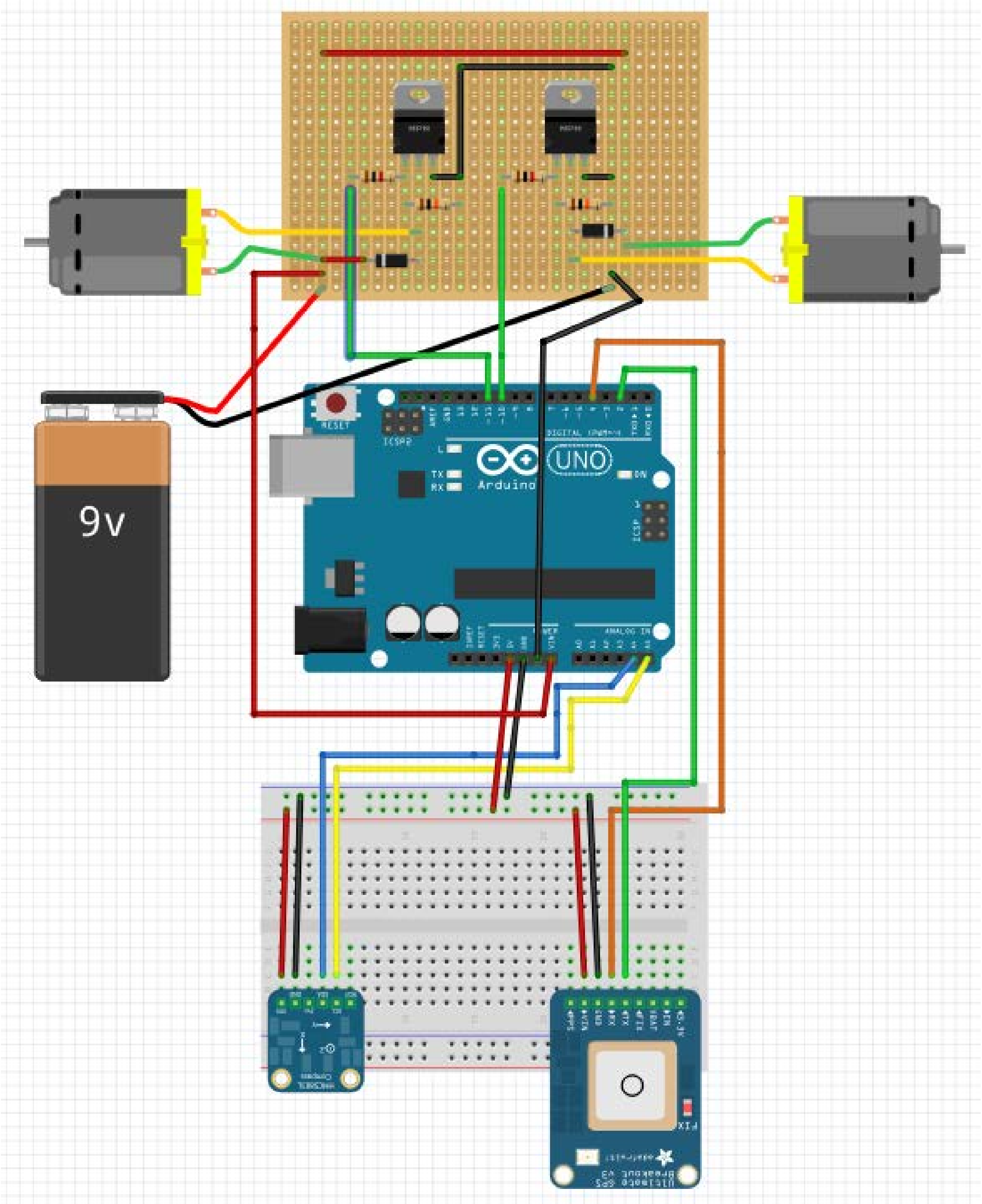
## Appendix A – Circuit Schematic and Layout

Circuit Schematic in LTSpice IV



Layout in Fritzing

Note: The 9v battery represents the 7.2v 17000mAh NiCd battery that was actually used.



# Appendix B – Code

main.ino

```
/*  Nicholas Moya
 *  SRNL - Boat Drone
 *  Dr. Tad Whiteside
 *  7/4/2016
 *
 *  BOAT DRONE - MAIN
 *  ------------------------------
 *  This code steers a boat toward multiple waypoints, starting with the
first
 *  one in a sequence and ending with the last one in the sequence. It
 *  accomplishes this by using a GPS module, a Compass module, two DC motors,
 *  and the BoatDrone library. The code works in these steps:
 *
 *    1. Determine its current location (GPS)
 *    2. Determine its current heading (Compass)
 *    3. Calculate the angle, direction, and distance between its current
position and the waypoint (BoatDrone)
 *    4. Using the DC motors, steer the boat to the waypoint until the
waypoint is reached (DC Motors)
 *    5. Once the waypoint is reached, steer in the direction of the next
waypoint
 *    6. Once all waypoints have been reached, turn off the DC motors.
 *
 *  This will be done continuously, meaning that if the current position
changes,
 *  the speeds of the DC motors will change so that it continues to steer the
boat to the next waypoint.
 *
 *  CONNECTIONS:
 *  Compass:  SDA -> A4, SCL -> A5
 *  GPS:      RX -> D2, TX -> D4
 *  DC Motor: Left PWN -> D11, Right PWM -> D10
 *  Battery;  High -> VIN, Low -> GND
 */



// GPS LIBRARIES
#include <Adafruit_GPS.h>
#include <SoftwareSerial.h>

// COMPASS LIBRARIES
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_HMC5883_U.h>

// BOATDRONE LIBRARY
#include <BoatDrone.h>



// OBJECTS
SoftwareSerial mySerial(4, 2); // Pins digital 4 (TX) and digital 2 (RX)
```

```
Adafruit_GPS GPS(&mySerial);
// Assign a unique ID to this sensor at the same time
Adafruit_HMC5883_Unified mag = Adafruit_HMC5883_Unified(12345);
BoatDrone prototype;



// GPS DECLARATIONS
// Set GPSECHO to 'false' to turn off echoing the GPS data to the Serial
console
// Set to 'true' if you want to debug and listen to the raw GPS sentences.
#define GPSECHO  false
// 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



// MY DECLARATIONS
//                                 {latitude, longitude}
//                  accurate to XX.XXXX____,  XX.XXXX____ places
const double waypoint[4][2] = {{33.34361936, -81.74101174},   // waypoint 1
                               {33.34399486, -81.7403841},    // waypoint 2
                               {33.34445751, -81.73954993},   // waypoint 3
                               {33.3439711, -81.73915431}};   // waypoint 4
// NOTE: The last waypoint must be where you are so that the boat comes back
to you
const int number_of_waypoints = 4;
const double declinationAngle = 0.11222467; // constant, based on location,
used in compass
double displacement; // distance from waypoint
double deltaY; // used to determine the desired heading
double deltaX; // used to determine the desired heading
int bearing; // compass heading (in degrees)
int heading; // The direction to the waypoint (in degrees)
bool reached_waypoint = false;
int i = 0;  // counter for waypoints

// PID declarations
int error3 = 0;
int error2 = 0;
int error1 = 0;
int u1 = 0;
int u = 0;

// DC motor declarations
const int base_percent_speed = 50;  // steady state speed
const int maxSpeed = 100;           // turning speed
const int RightMotorPin = 10;
const int LeftMotorPin = 11;
int right_motor_speed_percent = 0;
int left_motor_speed_percent = 0;



void setup()
```

```
{

  // delay for 60 secs to make time for disconnecting the cable, closing the
lid, and putting the boat in the water
  delay(60000);



  // **************************
  // *** INITIALIZE THE GPS ***
  // **************************
  GPS.begin(9600);
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);
  GPS.sendCommand(PGCMD_ANTENNA);
  useInterrupt(true);
  delay(1000);
  mySerial.println(PMTK_Q_RELEASE);



  // ******************************
  // *** INITIALIZE THE COMPASS ***
  // ******************************
  if(!mag.begin())
  {
    while(1);
  }
}



// *******************************
// *** GPS: INTERUPT FUNCTIONS ***
// *******************************
SIGNAL(TIMER0_COMPA_vect)
{
  char c = GPS.read();
#ifdef UDR0
  if (GPSECHO)
    if (c) UDR0 = c;
#endif
}

void useInterrupt(boolean v)
{
  if (v)
  {
    OCR0A = 0xAF;
    TIMSK0 |= _BV(OCIE0A);
    usingInterrupt = true;
  } else
  {
    TIMSK0 &= ~_BV(OCIE0A);
    usingInterrupt = false;
  }
}
```

```
void loop()
{

  while(i < number_of_waypoints)
  {
    do
    {



        // ******************************
        // *** GPS: PARSE COORDINATES ***
        // ******************************
        if (GPS.newNMEAreceived())
        {
          if (!GPS.parse(GPS.lastNMEA()))
            return;
        }



        // ****************************
        // *** COMPASS: GET BEARING ***
        // ****************************
        sensors_event_t event;
        mag.getEvent(&event);
        bearing = prototype.GetHeading(event.magnetic.y, event.magnetic.x,
declinationAngle);



        // **********************
        // *** GO TO WAYPOINT ***
        // **********************
        if (GPS.fix)
        {
          // get heading and distance to waypoint
          deltaY = waypoint[i][1] - GPS.longitudeDegrees;
          deltaX = waypoint[i][0] - GPS.latitudeDegrees;
          heading = prototype.GetHeading(deltaY, deltaX, declinationAngle);
          displacement = prototype.Haversine(GPS.latitudeDegrees,
GPS.longitudeDegrees, waypoint[i][0], waypoint[i][1]);

          // use heading and bearing to get u from PID function
          prototype.PID(heading, bearing, &error1, &error2, &error3, &u1,
&u);

          // use u to set speed of motors
          left_motor_speed_percent = base_percent_speed + u;
          right_motor_speed_percent = base_percent_speed - u;

          // implement speed of motors to steer boat
          prototype.DCmotor(left_motor_speed_percent, maxSpeed,
LeftMotorPin);
```

```
          prototype.DCmotor(right_motor_speed_percent, maxSpeed,
RightMotorPin);
        }
        else
        {
          // stay still while fix is acquired
          analogWrite(LeftMotorPin, 0);
          analogWrite(RightMotorPin, 0);
        }

        // has the waypoint been reached?
        if (displacement < 1.0)
          reached_waypoint = true;
        else
          reached_waypoint = false;

    }
    while(reached_waypoint == false);

    // pause for 1 sec between waypoints
    analogWrite(LeftMotorPin, 0);
    analogWrite(RightMotorPin, 0);
    delay(1000);

    // queue up next waypoint
    i++;

  }

  // all waypoints have been reached
  done();
}

// stay still after all waypoints have been reached
void done()
{
  while(1)
  {
    analogWrite(LeftMotorPin, 0);
    analogWrite(RightMotorPin, 0);
  }
}
```

BoatDrone.h

```
/*

Nicholas Moya
SRNL - Boat Drone
Dr. Tad Whiteside
7/4/2016

BOAT DRONE - SOURCE CODE (.h)
---------------------------------------------------------------------------
This is the source code .h file for the Boat Drone main arduino .ino file.
Five functions, and several variables, are declared here, wrapped in the
class BoatDrone, and referenced in the main code. The functions are:

        DisplayCoordinates - prints coordinates to the serial monitor
        Haversine - uses the haversine formula to calculate and return the
                                 distance between two coordinates
        GetHeading - determines the current heading
        PID - calculates the control variable, u, needed to steer the boat
        DCmotor - sets the speed of a DC motor

These functions are explicitly defined in the .cpp source code file.

*/

#ifndef BoatDrone_H
#define BoatDrone_H

class BoatDrone
{
                // PID variables shared with main
                int error3;
                int error2;
                int error1;
                int u1;
                int u;

        public:
                void DisplayCoordinates(double lat, double lon);
                double Haversine(double lat_start, double lon_start,
                                                   double lat_end, double lon_end);
                int GetHeading(double y, double x, double declinationAngle);
                void PID(int heading, int bearing, int *e1, int *e2,
                                          int *e3, int *u1, int *u);
                void DCmotor(int motor_speed_percent, int maxSpeed, int motorPin);

        private:
                const double r = 6371000.0; // radius of the earth in meters
                const double pi = 3.1416;

                // PID Parameters
                const int uMAX =  50; // This value must be such that this holds:
                const int uMIN = -50; // 100 = base_percent_speed +/- uMax/uMin.
                const double kp = 1.292;
```

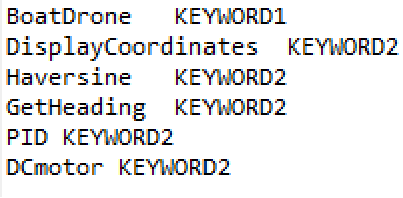
```
                        const double ki = 1.113;
                        const double kd = -0.282;
                        double k1 = kp + ki + kd;
                        double k2 = -kp - 2*kd;
                        double k3 = kd;
                        int opp_bearing;

                        // DC Motor variable
                        int motor_real_speed;
};

#endif
```

BoatDrone.cpp

```
/*

Nicholas Moya
SRNL - Boat Drone
Dr. Tad Whiteside
7/4/2016

BOAT DRONE - SOURCE CODE (.cpp)
---------------------------------------------------------------------------
This is the source code .cpp file for the Boat Drone main arduino .ino file.
Five functions, and several variables, are explicitly defined here and
referenced in the main code. The functions are:

        DisplayCoordinates - prints coordinates to the serial monitor
        Haversine - uses the haversine formula to calculate and return the
                     distance between two coordinates
        GetHeading - determines the current heading
        PID - calculates the control variable, u, needed to steer the boat
        DCmotor - sets the speed of a DC motor

These functions are referenced in the .h source code file.

*/

#include "Arduino.h"
#include "BoatDrone.h"

void BoatDrone::DisplayCoordinates(double lat, double lon)
{
  if (lat > 0)
  {
    Serial.print(lat, 4);
    Serial.print(" N, ");
  }
  else
  {
    Serial.print(lat*(-1), 4);
    Serial.print(" S, ");
  }

  if (lon > 0)
  {
    Serial.print(lon, 4);
    Serial.println(" E.");
  }
  else
  {
    Serial.print(lon*(-1), 4);
    Serial.println(" W.");
  }
}
```

```
double  BoatDrone::Haversine(double  lat_start,  double  lon_start,  double  lat_end,
double lon_end)
{
        // convert coordinates from degrees to radians
        double lat_start_rad = lat_start*pi/180.0;
        double lon_start_rad = lon_start*pi/180.0;
        double lat_end_rad = lat_end*pi/180.0;
        double lon_end_rad = lon_end*pi/180.0;

        // haversine formula
        double     haversine     =         2.0*r*asin(sqrt(sq(sin((lat_end_rad     -
lat_start_rad)/2.0))+cos(lat_start_rad)*cos(lat_end_rad)*sq(sin((lon_end_rad       -
lon_start_rad)/2.0)))) - 5;
        if (haversine <= 0.0) haversine = 0.0;

  return haversine;
}

int BoatDrone::GetHeading(double y, double x, double declinationAngle)
{
        double heading = atan2(y, x);
        heading += declinationAngle;

        // Correct for when signs are reversed.
         if(heading < 0)
        heading += 2*pi;

        // Check for wrap due to addition of declination.
        if(heading > 2*pi)
                heading -= 2*pi;

        // Convert radians to degrees for readability.
        double headingDegrees = heading*180.0/pi;

        return (int)headingDegrees;
}

void BoatDrone::PID(int heading, int bearing, int *e1, int *e2, int *e3, int *u1, int
*u)
{
        // update variables
        *e3 = *e2;
        *e2 = *e1;
        *u1 = *u;

        // steering algorithm
        if (bearing < 180)
        {
                opp_bearing = bearing + 180;
                if (heading - opp_bearing < 0)
                        *e1 = heading - bearing;
                else
                        *e1 = -1*(360-heading+bearing);
        }
        else
        {
                opp_bearing = bearing - 180;
                if (heading - opp_bearing < 0)
                        *e1 = 360 - bearing + heading;
                else
                        *e1 = heading - bearing;
        }

        // PID calculation
        *u = k1*(*e1) + k2*(*e2) + k3*(*e3);
        // original: *u = *u1 + k1*(*e1) + k2*(*e2) + k3*(*e3);
        // removed u1 to improve steady state error and overshoot
        // but at the cost of rise time

        // bound u
        if (*u > uMAX) *u = uMAX;
        if (*u < uMIN) *u = uMIN;

        // new values are passed by reference
}

void BoatDrone::DCmotor(int motor_speed_percent, int maxSpeed, int motorPin)
{
        // bound motor speed percent
        if (motor_speed_percent > maxSpeed) motor_speed_percent = maxSpeed;
        if (motor_speed_percent < 0) motor_speed_percent = 0;
        //Serial.print(motor_speed_percent); Serial.println(" %");

        // write speed to motor
        motor_real_speed = map(motor_speed_percent, 0, maxSpeed, 0, 255);
        analogWrite(motorPin, motor_real_speed);
}
```

keywords.txt



```
BoatDrone   KEYWORD1
DisplayCoordinates  KEYWORD2
Haversine   KEYWORD2
GetHeading  KEYWORD2
PID KEYWORD2
DCmotor KEYWORD2
```

update_boat_drone.cmd

```
@echo off

rem - Nicholas Moya
rem - Boat Drone Prototype
rem - Dr. Tad Whiteside
rem - 7/15/2016
rem
rem
rem - update_boat_drone.cmd
rem
rem - This code allows the user to simply double click this .cmd file and
the
rem - waypoint lat/lon values from the waypoints.txt file will be written
rem - into the arduino boat drone sketch AND the sketch will be uploaded.


rem - use lat/lon values from waypoints text file to set lat/lon values
for
rem        waypoints in arduino file
cd C:\Users\N8691\Documents\visual_basic
cscript Set_Waypoints.vbs

rem - upload arduino sketch with newly reset waypoints
cd C:\Program Files (x86)\Arduino
arduino.exe --upload
C:\Users\N8691\Documents\Arduino\arduino_demo\arduino_demo.ino

echo.
echo         Boat Drone code have been successfully uploaded.
echo.

PAUSE
```

Set_Waypoints.vbs

```
' ######################################################################
' Nicholas Moya
' SRNL - Boat Drone Prototype
' Dr. Tad Whiteside
' 7/15/2016
'
'
'
' Set_Waypoints.vbs
'
' This code opens a text file containing the number of waypoints and
' the waypoints, stores those values temporarily and then opens the
' arduino file and replaces the placeholders with the real values
' and closes it. Please see the README for details on how to use this
' code and write in the waypoints.txt file.
' ######################################################################



Const arduinoFilePath =
"C:\Users\N8691\Documents\Arduino\arduino_demo\arduino_demo.ino"
Const waypointsFilePath = "C:\Users\N8691\Documents\waypoints.txt"
Const ForReading = 1
Const ForWriting = 2
Const numOfWaypoints_placeholder = "numOfWay"
Const waypoints_placeholder = "waypoints"
Dim numOfWaypoints_value
Dim waypoints_value



'
##########################################################################
#######
' OPEN TEXT FILE, READ IN NUMBER OF WAYPOINTS AND WAYPOINTS, STORE THEM,
CLOSE FILE
'
##########################################################################
#######

' creates text file as object and opens it
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set waypointsFile = objFSO.OpenTextFile(waypointsFilePath,1)

' read in and store number of waypoints
numOfWaypoints_value = waypointsFile.ReadLine()

' read in and store waypoints
waypoints_value = waypointsFile.ReadLine()

' close file
waypointsFile.Close
```

```
Set waypointsFile = Nothing



'
##########################################################################
##################################
' OPEN ARDUINO FILE, REPLACE PLACEHOLDER FOR NUMBER OF WAYPOINTS WITH THE
REAL NUMBER OF WAYPOINTS, CLOSE FILE
'
##########################################################################
##################################

' sets text file as object, opens it
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFile = objFSO.OpenTextFile(arduinoFilePath, ForReading)

' read all lines from old file, close it
strText = objFile.ReadAll
objFile.Close

' replace the numOfWaypoints_placeholder with the numOfWaypoints_value
numOfWaypoints_value = Replace(strText, numOfWaypoints_placeholder,
numOfWaypoints_value)

' open text file, writes content to it, closes it
Set objFile = objFSO.OpenTextFile(arduinoFilePath, ForWriting)
objFile.WriteLine numOfWaypoints_value
objFile.Close



'
##########################################################################
##############
' OPEN ARDUINO FILE, REPLACE PLACEHOLDER FOR WAYPOINTS WITH THE REAL
WAYPOINTS, CLOSE FILE
'
##########################################################################
##############

' sets text file as object, opens it
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFile = objFSO.OpenTextFile(arduinoFilePath, ForReading)

' read all lines from old file, close it
strText = objFile.ReadAll
objFile.Close

' replace the waypoints_placeholder with the waypoints_value
waypoints_value = Replace(strText, waypoints_placeholder, waypoints_value)

' open text file, writes content to it, closes it
Set objFile = objFSO.OpenTextFile(arduinoFilePath, ForWriting)
objFile.WriteLine waypoints_value
objFile.Close


Wscript.Echo "          Waypoints have been successfully reset.          "
```

reset_placeholders.txt

```
const double waypoint[numOfWay][2] = {waypoints};
```

update_boat_drone_README.txt

```
INSTRUCTIONS TO RESET WAYPOINTS LAT/LON VALUES



Set up/Reset
-------------------------------------------------------------------------
1. In Set_Waypoints.vbs, change arduinoFilePath so it matches the path
      of the arduino file. Look in Set_Waypoints for an example.

2. Likewise, change waypointsFilePath so that it too matches the path
      of the text file that has the waypoints in it.

3. Make sure that the lat/lon values in the waypoints.txt file are
      written in the following way:



number_of_waypoints
{latValue1, lonValue1}, {latValue2, lonValue2}, ... {latValuen, lonValuen}



(see the waypoints.txt file for an example)

4. Replace the old actual lat and lon values in the arduino code with the
      placeholders. Ex:



const double waypoint[4][2] = {{33.34361936, -81.74101174},
                               {33.34399486, -81.7403841},
                               {33.34445751, -81.73954993},
                               {33.3439711, -81.73915431}};



becomes:



const double waypoint[numOfWay][2] = {waypoints};



(see reset_placeholders.txt for an example)

5. Change the directory in update_boat_drone.cmd to match the directory
      of the Set_Waypoints.vbs file.

6. Change the path in update_boat_drone.cmd to match the path
      of the arduino sketch file.
```

```
Action
--------------------------------------------------------------------------

Double click the update_boat_drone.cmd batch file and you should see two
      messages saying that the waypoints have been successfully reset
      and that the boat drone sketch has been uploaded.
```

# Appendix C – Bill of Materials

| Quantity | Part | Retailer Link | Cost |
|---|---|---|---|
| 1 | Triple-axis Magnetometer (Compass) Board - HMC5883L | https://www.adafruit.com/product/1746 | $9.95 |
| 1 | Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates - Version 3 | https://www.adafruit.com/product/746 | $39.95 |
| 2 | N-channel power MOSFET - 30V / 60A | https://www.adafruit.com/products/355 | $3.50 |
| 1 | Resistor 1K Ohm 1/4th Watt PTH - 20 pack | https://www.sparkfun.com/products/13760 | $0.95 |
| 1 | Through-Hole Resistors - 10K ohm 5% 1/4W - Pack of 25 | https://www.adafruit.com/products/2784 | $0.75 |
| 1 | Arduino Uno R3 (Atmega328 - assembled) | https://www.adafruit.com/products/50 | $24.95 |
| 1 | Half-size breadboard | https://www.adafruit.com/products/64 | $5.00 |
| 1 | 1N4001 Diode - 10 pack | https://www.adafruit.com/products/755 | $1.50 |
| 1 | Solid-Core Wire Spool - 25ft - 22AWG | https://www.adafruit.com/products/289 | $2.95 |
|  |  |  |  |
|  |  | **TOTAL** | $89.50 |