The FAMU-FSU College of Engineering
FSU Panama City, Panama City FL 32405
Electrical and Mechanical Engineering Undergraduate Department

TO: Senior Design Professors for EML 4552/EEL 4915C

FROM: RoboBoat Team

Subject: Test Readiness Review

Dear Dr. Damion Dunlap and Dr. Geoffery Brooks,

Contained in this document is the current testing plan and readiness review as it stands. As our project draws to a close, our team plans on implementing several testing procedures in order to ensure that the project will be a viable system when finished. As you read the document below, please know we are extremely appreciative for your time.

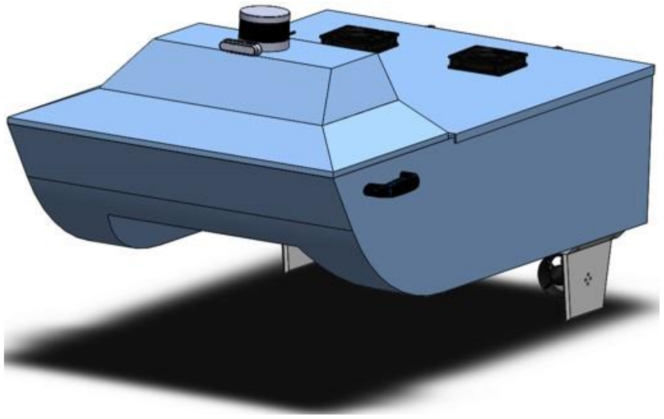
Sincerely,

Brandon Bascetta
Courtney Cumberland
Mark Hartzog
Madison Penney
Peter Oakes
Toni Weaver
FSU Panama City Mechanical and Electrical Engineering Seniors

FAMU-FSU
College of
Engineering

EEL 4911C/EML 4552

Test Readiness Review



RoboBoat Development Team

FSU Panama City Mechanical & Electrical Engineering Seniors:
Brandon Bascetta
Courtney Cumberland
Mark Hartzog
Madison Penney
Peter Oakes
Toni Weaver

Professors: Dr. Damion Dunlap & Dr. Geoffrey Brooks
10 July 2020

EEL 4911C/EML 4552
Test Readiness Review

TABLE OF CONTENTS

I. Summary of TRR Report

A. Advisor Contact Information:

ECE Senior Design Coordinator:

Dr. Geoffrey Brooks
(850) 770-2247
gbrooks@pc.fsu.edu

MEE Senior Design Coordinator:

Dr. Damion Dunlap
(850) 770-2204
ddunlap@fsu.edu

RoboBoat Technical Advisor:

Dr. Joshua Weaver
jnweaver@fsu.edu

B. RoboBoat - Development Team

Mechanical Design Lead - Brandon Bascetta
Manufacturing Lead - Courtney Cumberland
Software Lead - Mark Hartzog
Software/Hardware Integrator - Peter Oakes
Hardware Developer - Madison Penney
Systems Lead - Toni Weaver

C. Project Summary

The overall objective of this project is to develop and manufacture a working boat complete with sensors and basic software that is capable of competing in the RoboBoat competition. This goal will be achieved by completing three different subprojects. These include software development, hardware development, boat design and manufacturing. Figure 1, shown below, displays the functional decomposition of the project. This project will focus primarily on the left three branches of the image.

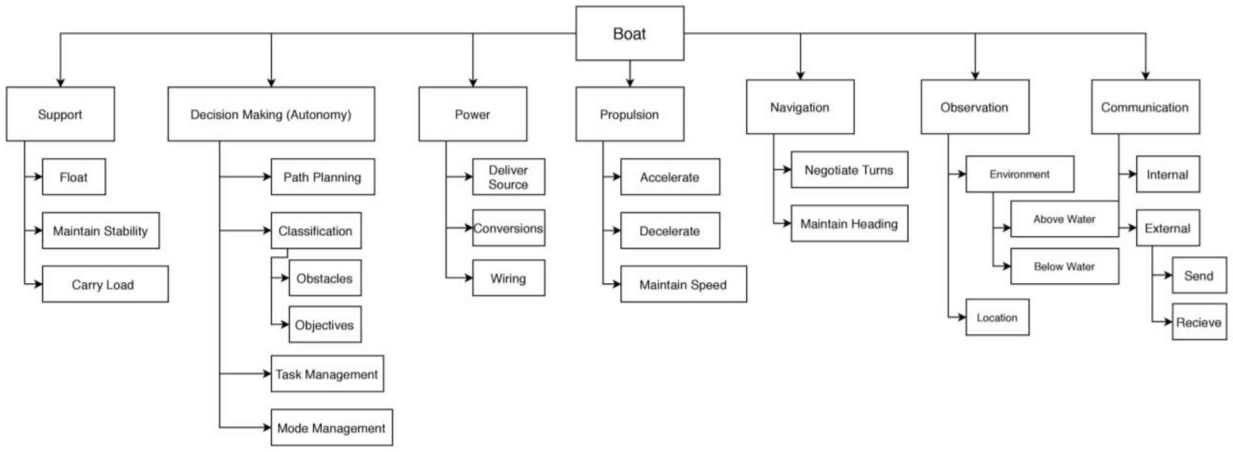


Figure 1. Functional Decomposition of the project.

i. Software Development

The software team is responsible for bringing life to the hardware components in order to make them serve a functional purpose. Using the Robot Operating System (ROS), as our middleware platform, we can tap into pre-existing algorithms that are often tailor made for our sensors by the sensor's creators themselves. Using these algorithms and the tools in the lab we can protype our own software, written by us, to create a functional system with each piece of software working in tandem to create a large and unique data set making the vehicle mobile.

ii. Hardware Development

The hardware design will essentially take each respective sensor and will wire and place it in the most optimal position of the vehicle. Because of the nature of some of the sensors, it is imperative that they are calibrated and placed in strategic locations in order to be implemented properly so that they may generate helpful data.

iii. Boat Design and Manufacturing

Utilizing the engineering design methods to meet the customer's needs, a larger, more stable boat was designed for this year's competition. Therefore, a new boat will be constructed. A fiberglass and epoxy resin composite was chosen as the primary material thus, the hand lay-up method of construction will be implemented to manufacture the hull of the boat as well as the lids. The final CAD design of the boat was created in Solidworks. The overall size and weight of the vessel is constrained by the RoboBoat rules. This boat will have a length of 50", width of 30" and height of 30" and an estimated weight of approximately 22.67 lbs excluding the electrical components.

D. Project Motivation

The RoboBoat competition is an international robotics competition that focuses on allowing young engineering students to create solutions for some of the most difficult and challenging electrical, and computer engineering challenges. The tasks themselves include using custom algorithms to allow the boat to autonomously solve puzzles. For example, some of the tasks include navigating a channel of buoys, finding a path through a field of obstacles, or performing a speed test to exhibit the vehicle's power. The specific duties of the software team are to take the powered sensors and setup their respective firmware, and drivers, in addition to wiring them and using their data sets to produce logic solving with algorithms. These algorithms, as mentioned previously, will allow the tasks required by the competition to be solved autonomously. Using the experiences from last year, the team will optimize the algorithms to enhance their performance which will allow the vehicle to exhibit better run times. Algorithms, data processing and publishing will be implemented primarily through the ROS environment.

E. RoboBoat Development Team Goals:

- Properly setup the drivers and various sensors and modules on the vehicle
- Create a functional data set generated by the various sensors
- Send the generated data to ROS (Robotic Operating System)
- Create data connections in ROS so the sensors can communicate to one another
- Import the data to custom executables and scripts to create logic solutions and data manipulation
- Create algorithms consisting of the modified data set
- Give the motors commands based upon the logic and algorithms being implemented

F. RoboBoat Project Stages

i. Spring 2020 - Previous Work

a. Setup and integrate hardware using the PE's power box. This included driver installation, manufacturer packages and firmware.
b. After the first step was complete, the sensors data generation methods were calibrated, and the heat displaced by them regulated by placing them in strategic positions.
c. The IMU was placed in an area that caused the least magnetic interference on the test boat.
d. The LiDAR was placed on the top of the test boat to maximize the visible areas and ranges.
e. The camera was placed in the front of the test boat to maximize obstacles detection.
f. After these steps were completed, the data was further calibrated and optimized and then sent into ROS once more.
g. The boat hull design was finalized in CAD.
h. The boat size was finalized at 30" X 50" x 25".
i. The boat hull mold was finished using 1" and ½" foam, spray foam, modeling clay and packing tape.
j. A modular fin design was created to attach the thrusters to and mount on the bottom of the pontoons.
k. Software was developed to drive the boat using motor mixing.
l. Software was developed to allow the boat to be driven using the RC controller.

ii. Summer 2020 - Current Work

m. The power system will be tested to ensure all voltages are outputting correctly.
n. Each necessary sensor will be connected to the power system.
o. Each sensor that is used will generate data.
p. The data collected in the first stage will be imported into ROS executable (nodes).
q. The LiDAR (Light Detection and Ranging sensor) and IMU (Inertial Measurement Unit) sensors will be tested and integrated into the ROS environment.
r. Code will be created to complete the mandatory navigation channel task.
s. Sensor data will be combined with navigation algorithms to allow the boat to perform basic obstacle avoidance.
t. The boat hull will be manufactured using hand laid fiberglass.
u. Sensor mounts will be created using CAD and manufactured using rapid prototyping.
v. The boat software, sensors and hull will be tested in water.

G. Proposed Testing Plan

Below is a tabulated list of the testable requirements that the team is planning to achieve during the proposed test.

Requirement	Testing Method	What is Success?	Passed (Y/N)
Hull			
Hull Floats	Place completed hull in a swimming pool.	The hull does not sink, it floats.	
Hull Carries 15 lbs	While in the swimming pool, dive weights will be added incrementally until 15 lbs is reached (dive weights are 3 lbs each).	The hull will carry 15 lbs with the pontoons only be submerged less than 4 inches.	
Hull weighs <25 lbs	Place hull on scale and read weight.	Weight is < 25 lbs.	
Hull doesn’t leak	Place hull in pool carrying 15 lbs for a minimum of 30 minutes.	Hull has no water in the interior.	
Minimal Deflection	Place 9 lbs on the center section and measure deflection with a ruler.	The measured deflection will be less than ⅛”.	

Hardware/Wiring (Components Not Connected)			
Power output for the Ouster OS1-16 LiDAR (not connected)	Using a multimeter, measure the voltage output from the power source to the Ouster OS1-16.	The voltage is within the range of 22-26 V, optimally at 24 V.	
Power output for the two ESCs (not connected)	Using a multimeter, measure the voltage output from the power source to the two ESCs.	The voltage, for each ESC, is within the range of 7-26 V, optimally at 16 V.	
Power output for the kill switch Arduino Mega (not connected)	Using a multimeter, measure the voltage output from the power source to the kill switch Arduino Mega.	The voltage is within the range of 7-12 V, optimally at 9 V.	
Power output for the PID Arduino Mega (not connected)	Using a multimeter, measure the voltage output from the power source to the PID Arduino Mega (from the Simply NUC).	The voltage is 5 V.	
Power output for the USB Hub (not connected)	Using a multimeter, measure the voltage output from the power source to the USB Hub.	The voltage is within the range of 5-12 V.	
Power output for the NETGEAR N900 Wireless Router (not connected)	Using a multimeter, measure the voltage output from the power source to the NETGEAR N900 Wireless Router.	The voltage is within the range of 12-19 V. Should be closer to 19 V due to how the power source was made.	
Power output for the Jetson Xavier (not connected)	Using a multimeter, measure the voltage output from the power source to the Jetson Xavier.	The voltage is within the range of 9-20 V.	
Power output for the Simply NUC (not connected)	Using a multimeter, measure the voltage output from the power source to the Simply NUC.	The voltage is within the range of 12-19 V.	
Hardware/Wiring (Components Connected and ON)			
Power output connection to the Ouster OS1-16 LiDAR (connected)	Using a multimeter, measure the voltage output and current draw to the Ouster OS1-16. After measuring the voltage, divide the maximum allowed power by this measured voltage to calculate the maximum allowed current.	The voltage is within the range of 22-26 V, optimally at 24 V. The power is within the range of 14-20 W (peak 22 W at startup).	

Power output connection to the two ESCs (connected)	Using a multimeter, measure the voltage output and current draw to the two ESCs.	The voltage, for each ESC, is within the range of 7-26 V, optimally at 16 V. The max current (constant), for each ESC, is 30 A.	
Power output connection to the kill switch Arduino Mega (connected)	Using a multimeter, measure the voltage output and current draw to the kill switch Arduino Mega.	The voltage is within the range of 7-12 V, optimally at 9 V.	
Power output for the PID Arduino Mega (connected)	Using a multimeter, measure the voltage output and current draw to the PID Arduino Mega (from the Simply NUC).	The voltage is 5 V.	
Power output connection to the USB Hub (connected)	Using a multimeter, measure the voltage output and current draw to the USB Hub.	The voltage is within the range of 5-12 V. The current does not exceed 4 A.	
Power output connection to the NETGEAR N900 Wireless Router (connected)	Using a multimeter, measure the voltage output and current draw to the NETGEAR N900 Wireless Router.	The voltage is within the range of 12-19 V, will likely be closer to 19 V. The current does not exceed 2.5 A.	
Power output connection to the Jetson Xavier (connected)	Using a multimeter, measure the voltage output to the Jetson Xavier.	The voltage is within the range of 9-20 V.	
Power output connection to the Simply NUC (connected)	Using a multimeter, measure the voltage output and current draw to the Simply NUC.	The voltage is within the range of 12-19 V. The current must not exceed 3 A	
Power output connection to the Ouster OS1-16 LiDAR (connected)	The LiDAR will be turned on and observed for 3 minutes.	The LiDAR runs smoothly without any brownouts, shutting off, malfunctioning, or overheating.	
Power output connection to the two ESCs (connected)	The two ESCs will be turned on and observed for 3 minutes.	The two ESCs run smoothly without any brownouts, shutting off, malfunctioning, or overheating.	
Power output connection to the kill switch Arduino Mega (connected)	The kill switch Arduino Mega will be turned on and observed for 3 minutes.	The kill switch Arduino Mega runs smoothly without any brownouts, shutting off, malfunctioning, or overheating.	

Power output connection to the PID Arduino Mega (connected)	The PID Arduino Mega will be turned on and observed for 3 minutes.	The PID Arduino Mega runs smoothly without any brownouts, shutting off, malfunctioning, or overheating.	
Power output connection to the NETGEAR N900 Wireless Router (connected)	The NETGEAR N900 will be turned on and observed for 3 minutes.	The NETGEAR N900 runs smoothly without any brownouts, shutting off, malfunctioning, or overheating.	
Power output connection to the Jetson Xavier (connected)	The Jetson Xavier will be turned on and observed for 3 minutes.	The Jetson Xavier runs smoothly without any brownouts, shutting off, malfunctioning, or overheating.	
Power output connection to the Simply NUC (connected)	The Simply NUC will be turned on and observed for 3 minutes.	The Simply NUC runs smoothly without any brownouts, shutting off, malfunctioning, or overheating.	
ESCs and Thrusters	Run the thrusters, which are connected to the ESCs, to max power. Measure the voltage and the current.	The voltage does not exceed 26 V, and the current does not exceed 30 amps.	
Turnigy High Capacity 10000mAh 4S LiPo Batteries	During testing, check the voltage output from the batteries.	The voltage range is maintained at 14.8-16.3 V.	
Sensor Design			
Sensor mounts articulate	Sensors will be placed on the mount and the angle will be adjusted by raising and lowering the mount.	Mount is able to adjust to different angles.	
Sensor mount will be adaptable	Mounts created will be modular to fit onto two 80/20 rails.	Mount will fit on the 80/20 rail showing that the sizing is correct and other mounts can be made using these sizings.	
Mounts are easily replaceable	The mounts will be 3D printed and spares will be made.	Print can be made on most 3D printer beds with common filament (PLA or PETG).	

Software			
Boat detects obstacles	Obstacles will be introduced in a controlled manner and the data will be logged.	Software accurately and repeatedly identifies obstacles.	
PID controller is capable of creating smooth continuous motion.	System will be driven using PID controller.	System moves in a smooth and continuous manner.	
Boat Localized	System will be traveled around a specific path several times and the data logged.	The data points gathered at each point will agree with each other (within a 10% margin of error).	
Basic Waypoint Navigation Completed	System will be tasked with a waypoint within ROS.	System arrives at the waypoint within a reasonable amount of time.	

These complied requirements will serve as a checklist for everything needed to stay efficient, productive and successful during the initial test of the vehicle.

II. Test Readiness Plan

This testing serves to allow the team to render experimental data to determine how the vehicle behaves and operates in the water. The data needed from the testing includes information regarding the buoyancy of the vessel, the stability of the vehicle in water, the effectiveness of the control system and the remote control commands, the stability of network connection from the vehicle to the computer, the resolution and quality of the sensor data in the physical environment and the software produced vehicle commands. These tests will confirm many of the project goals and milestones that date back to the onset of this semester. These milestones include a functional hull, working thrusters using a control system and remote control, functional integrated hardware and software tasking for the physical system.

A. Subsystem Testing

The testing plan will be concentrated on three different areas, boat hull performance, hardware and power correctness and software ability. These three areas each have a set of subsystem testing requirements.

EEL 4911C/EML 4552
Test Readiness Review

To begin, our team will be testing the manufactured boat hull. These tests will ensure that the hull will remain positively buoyant and have a stable moment of inertia. This will be done by performing test in the following areas:

- Hull ability to remain positively buoyant with ~15 pounds while not leaking after 30 minutes.
- Hull deflection remains under ⅛" in the center section.
- Hull weighs less than 25 pounds

The next stage of testing will concentrate on the electrical/hardware subsystem of the project. This stage will focus on ensuring that the electrical components of the system are maintained with the correct voltage and power requirements and the mounting of the hardware itself. This stage will also ensure that the power of the system will be maintained in a steady way and the hardware mounts will be sufficient. This stage will be focused on the following specific areas:

- Testing the outputs and connections for all electrical and hardware components.
- Testing the wiring of each component to ensure stable and accurate connections.
- Testing the adjustability of the sensor mounts.
- Testing the modularity of the mounts.
- Testing the future proof of the mounts.

The final stage of the testing phases will focus on the software programming of the project. This stage will focus on the affectability of the software to perform the prescribed duties of the project.

- The boat is able to localize in its environment.
- The boat is capable of obtaining basic waypoint navigation.

B. Testing Resources

The resources needed to conduct the test include two persons to manage the vehicle while it is floating in the water in the case that there is a situation where the team loses control of the vessel, the boat takes on water, capsizes or likewise failures. In addition to physical aid, the team will also be prepared with the supporting documentation of the sensors and a wiring diagram.

C. Testing Risks

As with any project, there are certain risks associated with any testing plan. These risks must be mitigated in order for the testing of the project to properly take place. For each risk associated with the testing plan, an acceptable replacement has been made.

The first, and most obvious risk involves the availability of a finished boat hull. This project covers several different areas with the boat hull acting as the shell to hold all the components of the project. However, if the boat hull for the project is not completed in time, a previous testing boat will be used in its place. This boat is known to be sound and positively buoyant.

Even if a finished hull is produced in time for testing, the risk for testing the hull is that it will sink, therefore, the testing should be done in the shallow end of a swimming pool rather than open water and a rescue swimmer should be on hand to salvage the hull.

The project requires a nerve system of wired components, which must be powered using several lipo batteries. One of the original goals of the project was to create a smart power box system to aid with the powering of components. If the power box for the project cannot be completed on time, however, the boat will be wired up without using it. Previous testing has been completed using a powered layout that did not contain a smart power box.

III. Conclusion

A comprehensive and well-defined testing plan is necessary in any major project. This project is a collaborative effort between manufacturing, hardware and software teams. With the current testing plan presented in this paper, the project goal of creating a working boat capable of waypoint navigation and basic entry level autonomy for the competition.

Appendix

The PID Node Executable

```
/*************************************************
 *   Mark Hartzog  <markthartzog@gmail.com>     *
 *************************************************/

#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
#include "controller/Drive.h"
#include "stdio.h"
#include "pid.h"

// Define Global Variables

float linear_vel;
float angular_vel;
float process_var_x = 0.0;
float process_var_z = 0.0;
float previous_error_x = 0.0;
float previous_error_z = 0.0;

// Define callback to unfiltered cmd_vel
void cmdvelCallback(const geometry_msgs::Twist vel){

// Set the x and z equal to data published by an unfiltered cmd_vel
linear_vel = vel.linear.x;
angular_vel = vel.angular.z;

}


int main(int argc, char **argv) {

// Initialize ROS node
   ros::init(argc, argv, "pid");

   ros::NodeHandle nh;

// Subsrcribe to unfiltered cmd_vel
   ros::Subscriber sub = nh.subscribe("/cmd_vel", 1, cmdvelCallback);
```

```
// Define publisher for filtered cmd_vel
    ros::Publisher controlled = nh.advertise<controller::Drive>("/controlled_velocities", 1);

// Define a loop rate to prevent overflow of data to the thread
    ros::Rate loop_rate(25);

// Define a handler for the PID class
    PID pid;

    // Define the PID gains and feed them into the Class
    // In Order: Kd, Ki, Kd, dt

    float pgain_x = 0.025;
    float igain_x = 0.0028;
    float dgain_x = 0.0066;
    float dt_x = 0.052;
    float max_x = 2.0;
    float min_x = -2.0;

    // Define the PID gains and feed them into the Class
    // In Order: Kd, Ki, Kd, dt

    float pgain_z = 0.025;
    float igain_z = 0.0033;
    float dgain_z = 0.0062;
    float dt_z = 0.052;
    float max_z = 1.0;
    float min_z = -1.0;


    // Define an object Twist
    geometry_msgs::Twist vel;

    // Define an object Twist
    controller::Drive drive;

    // Take in the X the goals
    pid.valueslinear(pgain_x, igain_x, dgain_x, dt_x, max_x, min_x);
    // Take in the Z the goals
    pid.valuesangular(pgain_z, igain_z, dgain_z, dt_z, max_z, min_z);

    // Define the increment variables
    float increment_x = 0.0;
    float increment_z = 0.0;

    ROS_INFO("The PID controller is on...");

while (ros::ok()) {

    // Checks the linear input to create limiter

    if (linear_vel > max_x){
        ROS_WARN("\nThe incoming linear cmd_vel exceeds limits. Setpoint being set to ([%f]):",
max_x);
        linear_vel = max_x;
    } else if (linear_vel < min_x){
        linear_vel = min_x;
        }
    // Checks the angular input to create limiter
    if (angular_vel > max_z){
    ROS_WARN("\nThe incoming angular cmd_vel exceeds limits. Setpoint being set to ([%f]):", min_x);
        angular_vel = max_z;
    } else if (angular_vel < min_z){
        angular_vel = min_z;
        }

    // Call the control function of the linear x
    increment_x = pid.controllinear(linear_vel, process_var_x, previous_error_x);
    // Call the control function of the angular z
    increment_z = pid.controlangular(angular_vel, process_var_z, previous_error_z);

    // Feed in previous error
    previous_error_x = linear_vel - process_var_x;
    previous_error_z = angular_vel - process_var_z;

    // Add new increment contribution to the previous process variable
    process_var_x += increment_x;
    process_var_z += increment_z;

    // Set the velocities equal to the publisher data
    drive.forward = process_var_x;
    drive.turn = process_var_z;

    // Publish
    controlled.publish(drive);
    // Spin and sleep
    ros::spinOnce();
    loop_rate.sleep();

}

    return 0;

}
```

The PID Header File

```
/*
Mark Hartzog <markthartzog@gmail.com>
Special thanks to Bradley J. Snyder <snyder.bradleyj@gmail.com>
*/

#include "ros/ros.h"
#include "cmath"

class PID {

public:

    // Define all varibles for linear
    float KP_X;
    float KD_X;
    float KI_X;
    float dt_X;
    float max_X;
    float min_X;
    float error_X;
    float integral_X;
    float derivative_X;
    float previous_error_X;

    // Define all varibles for angular
    float KP_Z;
    float KD_Z;
    float KI_Z;
    float dt_Z;
    float max_Z;
    float min_Z;
    float error_Z;
    float integral_Z;
    float derivative_Z;
    float previous_error_Z;

    // The PID function prototype which allows the transfer of values from the main exe for the linear
control
    void valueslinear(float KP_X, float KI_X, float KD_X, float dt_X, float max_X, float min_X);

    // The PID function prototype which allows the transfer of values from the main exe for the angular
control
    void valuesangular(float KP_Z, float KI_Z, float KD_Z, float dt_Z, float max_Z, float min_Z);

    // Defines the control loop function feeds in setpoint variable and process variable
    float controllinear(float SP_X, float PV_X, float prev_err_x);

     // Defines the control loop function feeds in setpoint variable and process variable
    float controlangular(float SP_Z, float PV_Z, float prev_err_z);

    // Define an error return for the derivative path
    float feedbackerror(float pre_x_er);

};


// The PID function definition
void PID::valueslinear(float gk, float gi, float gd, float delt, float h, float l){

// Delete records of the past and clear old errors

previous_error_X = 0.0;
integral_X = 0.0;

// Set variables equal to variables fed in from the main.
KP_X = gk;
KI_X = gi;
KD_X = gd;
dt_X = delt;
max_X = h;
min_X = l;

}
```

```
// The PID function definition
void PID::valuesangular(float gk, float gi, float gd, float delt, float h, float l){

// Delete records of the past and clear old errors

previous_error_Z = 0.0;
integral_Z = 0.0;

// Set variables equal to to variables fed in from the main.
KP_Z = gk;
KI_Z = gi;
KD_Z = gd;
dt_Z = delt;
max_Z = h;
min_Z = l;

}

float feedbackerror(float pre_x_er){

}

float PID::controllinear(float SP_X, float PV_X, float prev_err_x){

    // Define the loop variables used for processing
    float proportional_output;
    float integral_output;
    float derivative_output;
    //Redefine total output at 0
    float total_output;

    //ROS_INFO("The derivative gain: ([%f])", KD_X);
    // Define the error between the setpoint and the process variable
    error_X = (SP_X - PV_X);

    // Multiply by the proportion amount and define the output
    proportional_output = (KP_X * error_X);

    // Define the integrator summer
    integral_X += (error_X * dt_X);

    // Define the integrator output
    integral_output = (KI_X * integral_X);
    // Define the differentiator
    derivative_X = (error_X - prev_err_x) / dt_X;

    // Define the differential output
    derivative_output = (KD_X * derivative_X);

    // Define the total output
    total_output = proportional_output + integral_output + derivative_output;

    return total_output;

}

float PID::controlangular(float SP_Z, float PV_Z, float prev_err_z){

    // Define the loop variables used for processing
    float proportional_output = 0.0;
    float integral_output = 0.0;
    float derivative_output = 0.0;
    float integral_Z = 0.0;
    float derivative_Z =0.0;

    // Redefine output as 0
    float total_output = 0.0;

    // Define the error between the setpoint and the process variable
    error_Z = (SP_Z - PV_Z);

    // Multiply by the proportion amount and define the output
    proportional_output = (KP_Z * error_Z);

    // Define the integrator summer
    integral_Z += (error_Z * dt_Z);

    // Define the integrator output
    integral_output = (KI_Z * integral_Z);

    // Define the differentiator
    derivative_Z = (error_Z - prev_err_z) / dt_Z;

    // Define the differential output
    derivative_output = (KD_Z * derivative_Z);
    // Define the total output
    total_output = proportional_output + integral_output + derivative_output;

    //Save error record
    previous_error_Z = error_Z;

    return total_output;

}
```

The Waypoint Solver Algorithm

```
/*****************************
 *   2020 by Mark Hartzog    *
 *   and Michael Kirke       *
 *   markthartzog@gmail.com  *
 *   kirkeml1997@gmail.com   *
 *                           *
 *****************************/

#include "ros/ros.h"
#include "ros/time.h"
#include "std_msgs/String.h"
#include "std_msgs/String.h"
#include "geometry_msgs/Pose.h"
#include "geometry_msgs/Twist.h"
#include <costmap_converter/ObstacleArrayMsg.h>
#include <move_base_msgs/MoveBaseAction.h>
#include <actionlib/client/simple_action_client.h>
#include <iostream>
#include <array>
#include <cmath>
#include <math.h>

// Define global variables

bool first_bouy_reached = false;
bool second_waypoint_reached = false;
float PI = 3.14159265;
bool detection = false;

class Task{
```

```
public:

Task get;

void vectormath(float ly, float ry, float lx, float rx, float scale, float &wpx, float &wpy){

    float u_y = ly - ry;
    float u_x = lx - rx;
    ROS_INFO("Vector component for x: ([%lf])", u_x);
    ROS_INFO("Vector component for y: ([%lf])", u_y);
    // Call the normalize method
    float normalized_u_x = (u_x / (sqrt((pow(u_x, 2.0)) + (pow(u_y, 2.0)))));
    float normalized_u_y = (u_y / (sqrt((pow(u_x, 2.0)) + (pow(u_y, 2.0)))));
    ROS_INFO("The normalized vector component for x: ([%lf])", normalized_u_x);
    ROS_INFO("The normalized vector component for y: ([%lf])", normalized_u_y);

    float angle_between_buoys = (atan2(u_y, u_x));
    float magnitude_u = sqrt(pow(u_x, 2.0) + pow(u_y, 2.0));

    // Perform the 90 deg rotation
    float sx = 0.0;
    float sy = 0.0;
    sx = normalized_u_x;
    sy = normalized_u_y;
    normalized_u_x = sy;
    normalized_u_y = -1 * sx;

    //Scale up the vector

    wpx = normalized_u_x * scale;
    wpy = normalized_u_y * scale;

    ROS_INFO("The scaled rotated vector component for x: ([%lf])", wpx);
    ROS_INFO("The scaled rotated vector component for y: ([%lf])", wpy);

}

bool navgoal(float x, float y){

    bool flag = false;

    typedef actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction> MoveBaseClient;
```

```
// Tell the action client that we want to spin a thread by default
MoveBaseClient ac("move_base", true);

// Wait for the action server to come up
while(!ac.waitForServer(ros::Duration(5.0))){
  ROS_INFO("Waiting for the move_base action server to come up");
}
  move_base_msgs::MoveBaseGoal goal;

  ROS_INFO("Setting x Waypoint to: ([%lf])", x);
  ROS_INFO("Setting y Waypoint to: ([%lf])", y);

  // Send a goal to the robot to move towards the first set of buoys
  goal.target_pose.header.frame_id = "map";
  goal.target_pose.header.stamp = ros::Time::now();

  goal.target_pose.pose.position.x = x;
  goal.target_pose.pose.position.y = y;

  //Need to fix this to be a dynamic quaternion. Not hardcoded to 1.0.
  //geometry_msgs::Pose orient;

  goal.target_pose.pose.orientation.w = 1.0;

  ROS_INFO("Sending goal");
  ac.sendGoal(goal);

  ac.waitForResult();

  if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED){
    ROS_INFO("The first set of bouys were reached");
    flag = true;
    //ros::shutdown();
  }

  else{
    ROS_INFO("The rover failed to move for some reason");
    ros::shutdown();
  }
```

```
    }
}

class Buoy{

    Buoy buoyLeft;
    Buoy buoyRight;

    public:

    float point1_x;
    float point2_x;
    float point3_x;
    float point1_y;
    float point2_y;
    float point3_y;
    float angle;


    float average_x(){
        float calcX = (point1_x + point2_x + point3_x) / 3;
        //ROS_INFO("The x position: [%lf]", calcX);
        return calcX;
    }

    float average_y(){
        float calcY = (point1_y + point2_y + point3_y) / 3;
        //ROS_INFO("The y position: [%lf]", calcY);
        return calcY;
    }

    float anglefinder(float y, float x){

        float angle = (atan2(y, x));
        return angle;
    }

    float midpoint_locator(float p1, float p2){

        float midpoint = ((p1 + p2) / 2);
        return midpoint;
    }
};
```

```
// Defines the position callpack function

void positionCallback(const costmap_converter::ObstacleArrayMsg pos){

    buoyLeft.point1_x = pos.obstacles[0].polygon.points[0].x;
    //ROS_INFO("The x points: [%lf]", buoyLeft.point1_x);
    buoyLeft.point2_x = pos.obstacles[0].polygon.points[1].x;
    buoyLeft.point3_x = pos.obstacles[0].polygon.points[2].x;

    buoyLeft.point1_y = pos.obstacles[0].polygon.points[0].y;
    buoyLeft.point2_y = pos.obstacles[0].polygon.points[1].y;
    buoyLeft.point3_y = pos.obstacles[0].polygon.points[2].y;

    buoyRight.point1_x = pos.obstacles[1].polygon.points[0].x;
    //ROS_INFO("The x points: [%lf]", buoyRight.point1_x);
    buoyRight.point2_x = pos.obstacles[1].polygon.points[1].x;
    buoyRight.point3_x = pos.obstacles[1].polygon.points[2].x;

    buoyRight.point1_y = pos.obstacles[1].polygon.points[0].y;
    //ROS_INFO("The y points: [%lf]", buoyRight.point1_y);
    buoyRight.point2_y = pos.obstacles[1].polygon.points[1].y;
    buoyRight.point3_y = pos.obstacles[1].polygon.points[2].y;

    if ((buoyRight.point1_x != 0) || (buoyRight.point2_x != 0) || (buoyRight.point3_x != 0) ||
(buoyRight.point1_y != 0) || (buoyRight.point2_y != 0) || (buoyRight.point3_y != 0)){
        detection = true;
    }
    if ((buoyLeft.point1_x != 0) || (buoyLeft.point2_x != 0) || (buoyLeft.point3_x != 0) ||
(buoyLeft.point1_y != 0) || (buoyLeft.point2_y != 0) || (buoyLeft.point3_y != 0)){
        detection = true;
    }
}

int main(int argc, char **argv){
    ros::init(argc, argv, "straight_line_task");

    // Declares and defines node object
    ros::NodeHandle nh;

    // Subscribes to the the obstacle detection package to gather position data
    ros::Subscriber sub = nh.subscribe("/costmap_converter/costmap_obstacles", 10000, positionCallback);

    while (ros::ok()) {
```

```
ros::spinOnce();

// Calculates midpoint between the two.

if (detection == true){

     if(first_bouy_reached == false){
        float midpoint_x = buoyRight.midpoint_locator(buoyLeft.average_x(), buoyRight.average_x());
        float midpoint_y = buoyRight.midpoint_locator(buoyLeft.average_y(), buoyRight.average_y());
        first_bouy_reached = get.navgoal(midpoint_x, midpoint_y);
    }
}

// Define the straight line action
/*if(first_bouy_reached == true){
    typedef actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction> MoveBaseClient;
    //tell the action client that we want to spin a thread by default
    MoveBaseClient ac("move_base", true);
    //wait for the action server to come up
    while(!ac.waitForServer(ros::Duration(5.0))){
        ROS_INFO("Waiting for the move_base action server to come up");
    }
        move_base_msgs::MoveBaseGoal goal;
        float px = key.pointpublisher_x();
        float py = key.pointpublisher_y();
        ROS_INFO("Setting the next x Waypoint to: ([%lf])", key.average_x());
        ROS_INFO("Setting the next y Waypoint to: ([%lf])", py);
        //we'll send a goal to the robot to move towards the first set of buoys
        goal.target_pose.header.frame_id = "map";
        goal.target_pose.header.stamp = ros::Time::now();
        goal.target_pose.pose.position.x = px;
        goal.target_pose.pose.position.y = py;
        //Need to fix this to be a dynamic quaternion. Not hardcoded to 1.0.
        //geometry_msgs::Pose orient;
        goal.target_pose.pose.orientation.w = 1.0;
        ROS_INFO("Sending goal");
        ac.sendGoal(goal);
        ac.waitForResult();

        if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED){
            ROS_INFO("The first set of buoys were reached");
            second_waypoint_reached = true;
```

```
                //ros::shutdown();
            }
            else{
                ROS_INFO("The rover failed to move for some reason");
                //ros::shutdown();
            }

        } */

    }
        ROS_INFO("I REACHED THE END OF THE NODE");
return 0;

}
```

Arduino Motor Mixing Code and Visual Feedback

```
//*********************************//
// Brandon Bascetta <brandonbascetta@gmail.com>
// Toni Weaver <tfs32413@gmail.com>
//*********************************//

//Include Libraries
#include "ros.h"
#include "std_msgs/Int16.h"
#include "Servo.h"
#include "FastLED.h"

//Function Prototypes

//Autonomous control
void cmd_control(int duty_l, int duty_r);

//Manual RC
void esc_control_manual();

//Read in rc input
void rc_read_in();

//Light Control
void lightboi(int light_mode);
```

```
//Define pins and such
#define CH1 3
#define CH2 4
#define CH5 5
#define CH6 6
#define CH8 7
#define ESCL 10
#define ESCR 11
#define LED_PIN 8
#define NUM_LEDS 256

//Led panel control object
CRGB leds[NUM_LEDS];

//Servo objects for esc writing
Servo escl;
Servo escr;

//ros node handler
ros::NodeHandle nh;

//Some global variables
int left_duty = 0, right_duty = 0;
unsigned long ch1 = 0;
unsigned long ch2 = 0;
unsigned long ch5 = 0;
unsigned long ch6 = 0;
unsigned long ch8 = 0;

//mode for lights
int mode = 1;
//1 = manual
//2 = autonomous
//3 = kill

//Toni's variables
//Variables for the code
long thrusterL = 0;
long thrusterR = 0;

//linear value x
int linx = 0;
```

```
//angular value w
int omega = 0;

//Velocity map values
int minV = -10;
int maxV = 10;

//angular
int minA = -10;
int maxA = 10;

//These values represent the output velocities of the thrusters
int escMin = 1100;
int escMed = 1500;
int escMax = 1900;
long rcescL = 0;
long rcescR = 0;

//
bool manual = false;
bool lockEngaged = true;
bool horn = false;


//These values reflect general values of the rc transmitter may not be exact numbers
int rcMed = 1500;
int rcLow = 980;
int rcHigh = 2000;

//Calback Functions
void duty_input_left( const std_msgs::Int16& vall)
{
  left_duty = vall.data;
}

void duty_input_right( const std_msgs::Int16& valr)
{
  right_duty = valr.data;
}
//Setting up ros subscribers
ros::Subscriber<std_msgs::Int16> sub1("drive_cmd_left" , duty_input_left);
ros::Subscriber<std_msgs::Int16> sub2("drive_cmd_right" , duty_input_right);
```

```
void setup() {

  //Initialize Pin I/O's
  FastLED.addLeds<WS2812B, LED_PIN, GRB>(leds, NUM_LEDS);
  pinMode(CH1, INPUT);
  pinMode(CH2, INPUT);
  pinMode(CH5, INPUT);
  pinMode(CH6, INPUT);
  pinMode(CH8, INPUT);

  //initialize node and topic subscriptions
  nh.initNode();
  nh.subscribe(sub1);
  nh.subscribe(sub2);

  //default esc signal to 1500 ms
  left_duty = 1500;
  right_duty = 1500;

  //Startup for lights
  for (int i = 0; i < NUM_LEDS; i++) {
    leds[i] = CRGB(0, 10, 10);
    FastLED.show();
  }

  for (int i = 0; i < NUM_LEDS; i++) {
    leds[i] = CRGB(10, 10, 0);
    FastLED.show();
  }

  for (int i = 0; i < NUM_LEDS; i++) {
    leds[i] = CRGB(10, 0, 0);
    FastLED.show();
  }

  //Attach onjecy to esc pin and set min and max output
  escl.attach(ESCL, 1100, 1900);
  escr.attach(ESCR, 1100, 1900);
}

void loop() {
  //check callbacks
  nh.spinOnce();

  //check rc
  rc_read_in();

  if (lockEngaged == false) {

    //for autonomous control
    if (manual == false) {
      nh.loginfo("Autonomous!");
      cmd_control(left_duty, right_duty);

    }
    //for manual mode
    if (manual == true) {
      nh.loginfo("Manual!");
      esc_control_manual();
    }

  }
  lightboi(mode);

}








void cmd_control(int duty_l, int duty_r) {

  //write esc commands from node
  escl.writeMicroseconds(duty_l);
  escr.writeMicroseconds(duty_r);
}

void rc_read_in() {
  ch1 = pulseIn(CH1, HIGH);
  ch2 = pulseIn(CH2, HIGH);
  ch5 = pulseIn(CH5, HIGH);
  ch6 = pulseIn(CH6, HIGH);
  ch8 = pulseIn(CH8, HIGH);

  if (ch8 < 1500) {
    horn = true;
  }
  else {
    horn = false;
  }

  if (ch5 > 1500 || ch5 < 900)
  {
    lockEngaged = true;
    //nh.loginfo("Lock Engaged!");
    mode = 3;
    escl.writeMicroseconds(1500);
    escr.writeMicroseconds(1500);
    nh.loginfo("Killed!!");

  }

  else
  {
    lockEngaged = false;
    nh.loginfo("Lock Disbaled!");

    //Manual/auto switch
    if (ch6 > 1500)
    {
      manual = true;
      mode = 1;
    }
    else
    {
      manual = false;
      mode = 2;
    }
  }
}
void esc_control_manual()
{

  //input from ch1 for linear velocity and ch2 for angular velocity
  linx = map(ch1, rcLow, rcHigh, minV, maxV);
  omega = map(ch2, rcLow, rcHigh, minA, maxA);


  //convert to driving each motor
  thrusterL = linx - omega;
  thrusterR = (linx + omega) * 0.75;


  //convert to pwm for esc
  rcescL = map(thrusterL, minV, maxV, escMin, escMax);
  rcescR = map(thrusterR, minV, maxV, escMin, escMax);

  //send command to esc
  escl.writeMicroseconds(rcescL);
  escr.writeMicroseconds(rcescR);


}


void lightboi(int light_mode) {

  //Manual
  if (light_mode == 1) {
    for (int i = 0; i < NUM_LEDS; i++) {
      leds[i] = CRGB(10, 10, 0);
    }

  }

  //Autonomous
  if (light_mode == 2) {

    for (int i = 0; i < NUM_LEDS; i++) {
      leds[i] = CRGB(0, 10, 10);
    }

  }

  //Killed
  if (light_mode == 3) {

    for (int i = 0; i < NUM_LEDS; i++) {
      leds[i] = CRGB(10, 0, 0);
    }

  }

  FastLED.show();
}
```

References

Figures provided by the references below:

Mistry, Siddharth, et al. “Design of HMI Based on PID Control of Temperature.” *Research Gate*, May 2017, www.researchgate.net/publication/316709017_Design_of_HMI_Based_on_PID_Control_of_Temperature.

Pebrianti, Dwi. “Exploration of Unknown Environment with Ackerman Mobile Robot Using Robot Operating System (ROS).” *Research Gate*, Dec. 2015, www.researchgate.net/publication/289882516_Exploration_of_unknown_environment_with_Ackerman_mobile_robot_using_robot_operating_system_ROS/figures?lo=1.