MAE 434W/435 Capstone Project:

Development of An Underwater Autonomous Vehicle

Faculty Advisors:

Dr. Krishnanand Kaipa, Dr. Cong Wei

Project Team Members:

William Buhrig, Justin Day, Nick Eoff, Tom Herlihy, Matthew Laverty, Colin Sizemore

Table of Contents

Table of Contents....................2
List of Figures....................2
Abstract....................3
Introduction....................3
Methods....................4
    Completed Methods....................4
    Proposed Methods....................5
Preliminary Results....................6
Discussion....................8
Appendices....................9
References....................12

List of Figures

A1.1: Python code for the AUV Basestation Control System....................10
A1.2: Python code for the AUV Drone Control System....................11
A2.1: Project Budget....................12
A3.1: GANTT Chart for MAE 434W Discussions and Project Work....................12
A3.2: GANTT Chart for MAE 435 Project Work....................13

Abstract

This report will be about the MAE 434W/435 senior design project titled “Development of an Underwater Autonomous Vehicle.” This project deals with an existing underwater autonomous vehicle, or AUV that is 15.5 inches high, and 5 inches wide. It is propelled by four thrusters powered by brushless DC motors and controlled by eight potentiometers. The scope of the project is to test and modify the existing AUV design. Initial tests were done to ensure that the motors operate correctly and to determine where water infiltration occurs. Completed results show that the motors are fully functional. However, the design will be modified so that the AUV will be operated by a single PlayStation 3 controller. A small amount of water infiltrates the AUV when it is submerged in water. Consequently, the design will be modified to create a completely waterproof design. Additional testing and modifications will be performed to allow the AUV to operate untethered and to achieve neutral buoyancy.

Introduction

The automation of technology has been of noted importance to engineers. Automated technologies have been utilized in many industries, for example in the automotive, aviation, and manufacturing. Underwater exploration has utilized automation as well through the use of autonomous underwater vehicles, or AUVs. An example of a use case for autonomous underwater vehicles is for “exploring the diverse oceanography of Spain” [1]. Another example of the use of autonomous underwater vehicles was “monitoring undersea cables for damage by using the electromagnetic field from the cables to direct the AUV” [2]. Controlling AUVs also greatly varies between models, as one reference showed “the use of a fuzzy PID controllers in AUVs have been adopted widely by the industry” [3]. Another design for autonomous control is “combining sonar and real-time path planning to allow the AUV to operate in unknown environments" [4]. The future of AUV technology has been foretold to be "RaspberryPi based computer navigation systems, energy harvesting, and high-speed underwater communication" [5]. Thus, the need for a low-cost AUV has been established.

The purpose of the project is to find a low-cost solution to the development of autonomous underwater vehicles. Projects like Bluefin for example "utilize sophisticated navigation, control, and power systems" which are costly to develop [5]. If one is lost the time to build another would take a long period of time. Utilizing a low-cost design would allow for more AUVs to be utilized in applications like "search and rescue, salvage operations, oceanographic mapping" [5].

Methods

Completed Methods

To provide a visual representation of the new AUV control system, a python-based interface was developed. The system takes input from a PlayStation 3 (Sony Interactive Entertainment, San Mateo, CA) controller and depicts the outputs in a visual format. The output for the thrusters is represented by a vertical bar whose height increases as thruster output increases. The position of the front of the drone is represented by a circular dot inside of a square, with the dot moving inside of the square based upon control input. Drone depth is also represented in a similar manner to thruster output, with a vertical bar whose height decreases as depth decreases. The code for this method has been included in Appendix A.

To ensure the AUV is appropriately ballasted two approaches were taken. The first was a theoretical calculation based on the internal volume of air inside the AUV and the volume of the

AUV’s resin hull and applying those to the buoyant force formula: $\frac{V_{Sub}}{V_{Total}} = \frac{\rho_{avg,body}}{\rho_{fluid}}$. This would be compared to the density of water, which is 997 kilograms per meter cubed, and the resulting displacement of the AUV came out to five and a half pounds of water. This simple calculation will be compared to a computer scan of the AUV to verify its displacement. Water testing was also performed, which involved immersing it in forty gallons of water in a closed environment. When unladen the AUV would not sink, however when five pounds of lead diving weight and the approximately three pound battery were loaded into the AUV it sank reliably.

Proposed Methods

To address problems with potential water infiltration both CAD analysis and physical testing will be used. The previously mentioned Inventor files will be used to analyze how the O-ring water seal between the nose and main sections of the drone prevents water from flowing inside the drone. The physical testing will involve immersing the front end of the AUV under a depth of water such that the O-ring will be submerged, and the section will remain submerged for up to an hour, which periodically checks to see if water has bypassed the O-ring. If the O-ring fails it will be replaced with a more fitting O-ring based on its dimensions and the elasticity of the material the O-ring is made of, for example 3M silicone or natural latex rubber. An additional measure to improve this seal is to add a Latex gasket on both sides of the rubber gasket to improve its elastic properties.

To help guide the design process and parameters for the AUV, several standards will be utilized. The American Society of Mechanical Engineers (ASME) Boiler and Pressure Vessel Code will be utilized to ensure the AUV does not fail due to water pressures which increase with depth. ANSI/ABYC E-13-2022 will be relevant if lithium-ion batteries are used to power the drone if a change of battery type is decided for better buoyancy or otherwise. While currently under development, ISO/AWI 20682 Autonomous Underwater Vehicles - Risk and Reliability will give additional guidance on general testing the electronic systems of the AUV. Finally, to provide guidance for waterproofing testing, the Ingress Protection Code (IP Code) will be utilized with the drone needing to be of an IP Code rating of 9 for zero water infiltration.

Due to the discovery of the hull slowly deteriorating when immersed in water, the AUV will be coated with a protective layer of sealant to prevent material deterioration when submerged in water. This material type that we have chosen is an Epoxy coating that will be applied from a pneumatic spray gun thinned by a lacquer. The specific formula of resin will be chosen by two criteria: the coating must properly bond to the acrylic hull without compromising its structural integrity, additionally the layer must be stable under saline, brackish, and freshwater conditions.

To finalize the control system loop, the local compute unit on the drone still needs to be extensively coded to be capable of receiving the serial connection from the control cable and translate those commands into actions done by the drone. The current goal is to get the drone to act as a RC drone that can then be abstracted to generalized commands, thereby easing the control system into its final state over time.

Preliminary Results

Waterproofing the AUV is still one of the major focuses for this project. In all three of the tests that were conducted water leaked into the craft. On the second round of testing it was determined that a leak point for the craft was coming from the top section where the AUV can be opened into two separate pieces. It was determined that the O-ring that had been placed to seal the two sections was not creating a sufficient seal. To eliminate this issue different O-rings that were made from a more mailable and thicker material were ordered to take the place of the existing o-ring to create a better formed seal around the opening. However, when the O-rings came from the manufacturer, it was found that they did not fit the craft. Dr. Kaipa suggested that sections from latex gloves could be placed in conjunction with the existing O-ring to create a seal. This would ensure that when the top section of the craft is screwed down onto the body that the latex would be squeezed tightly enough between the two pieces to create a waterproof seal. Since then, latex material has been acquired and further waterproofing tests have been planned.

While conducting the third test on craft a new issue regarding the resin material the craft was discovered. The resin that the craft was made from felt spongey as if it had absorbed water. After running one of our fingernails around the thicker portion on the bottom of the craft a considerable amount of resin was able to be scraped off the craft leading to the conclusion that the material was deteriorating from being placed in water. To combat this issue, it was decided that a water proof epoxy would be used to coat the shell, completely sealing the shell from outside water. In this third test it was also observed that air bubbles had permeated out from the gaskets on the motor mounts. Seeing these air bubbles raised questions regarding the seal around the mounts so it was decided that when the shell is coated with the epoxy, the mounts themselves will also be coated to eliminate them as a possible leak point.

The control system as of this point has not been fully implemented; however, we have been able to program the foundations to the remote control system in Python. This foundation is a telecommunications console, running on a laptop, that allows any Human Interface Device, Flight-Stick or Wireless Controller, to be used as an input. Once this input signal is received the program will translate the input signal to a corresponding output on the display and eventually to a serial link between the remote control station and the drone.

Additionally, to allow the drone to operate, a Raspberry Pi (Raspberry Pi Foundation, Caldecote, UK) computer will receive commands via a control cable. The computer will take the signals generated from the control system interface, a laptop computer, via a human interface device, parse the signal, then send it to the AUV which will in turn receive the signal and move according to the user input. These results will serve as a foundation for making the control systems of the drone architecture capable of supporting a radio interface device for future progress.

Discussion

As the drone has developed there have been major improvements to the control systems of the drone. The control system has gone from eight different potentiometers to a human interface device, PlayStation 3 or HOTAS controller, and a remote computer system to local computation device to the drone. The water resistance of the drone has been improved; however, the drone still has a leakage issue through the o-ring and or the motor mounts. To improve the o-ring section we will in addition to the new o-rings an additional Latex gasket will be added to the seal. For the motor mounts the gaskets that are currently installed will be replaced with a silicone caulking to remove possible ingress locations of water. In addition to these modifications to make the AUV neutrally buoyant lead weights have been obtained which will be distributed throughout the hull of the drone.

Appendices

```
from ast import AsyncFunctionDef
from asyncio.windows_events import NULL
from cmath import isclose
from pickle import FALSE
import sys
from winreg import KEY_CREATE_SUB_KEY
import pygame
import serial
from pygame.locals import *

def draw_Screen():
    # Control Display
    pygame.draw.rect(screen,x_y_grid_color, x_y_grid)
    pygame.draw.rect(screen,thrust_grid_color, thrust_grid)
    pygame.draw.rect(screen,motor1_grid_color, motor1_grid)
    pygame.draw.rect(screen,motor2_grid_color, motor2_grid)
    pygame.draw.rect(screen,motor3_grid_color, motor3_grid)
    pygame.draw.rect(screen,motor4_grid_color, motor4_grid)

    # Steering Window
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(margin_size-line_size,margin_size-line_size,line_size,bar_height+2*line_size))    # Left
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(margin_size-line_size,margin_size-line_size,bar_height+2*line_size,line_size))    # Top
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(margin_size-line_size,margin_size+bar_height,bar_height+2*line_size,line_size))   # Bottom
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(margin_size+bar_height,margin_size-line_size,line_size,bar_height+2*line_size))   # Right
    screen.blit(pygame.font.SysFont(FONT,int(bar_height/10)).render('S',False,(255,255,255)),(margin_size,margin_size))
    # Thrust Window
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(2*margin_size+bar_height-line_size,margin_size-line_size,line_size,bar_height+2*line_size))          # Left
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(2*margin_size+bar_height+bar_width,margin_size-line_size,line_size,bar_height+2*line_size))          # Right
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(2*margin_size+bar_height-line_size,margin_size-line_size,bar_width+line_size,line_size))             # Top
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(2*margin_size+bar_height-line_size,margin_size+bar_height,bar_width+line_size,line_size))            # Bottom
    #pygame.draw.rect(screen,[255,255,255],pygame.Rect(2*margin_size+bar_height,margin_size+bar_height/2,bar_width,line_size))                           # Center Line
    screen.blit(pygame.font.SysFont(FONT,int(bar_height/10)).render('T',False,(255,255,255)),(2*margin_size+margin_size/4+bar_height,margin_size))

    #Motor Number 1
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(3*margin_size+bar_height+bar_width-line_size,margin_size-line_size,line_size,bar_height/2+2*line_size-margin_size/2))     # Left
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(3*margin_size+bar_height+2*bar_width,margin_size-line_size,line_size,bar_height/2+2*line_size-margin_size/2))   # Right
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(3*margin_size+bar_height+bar_width-line_size,margin_size-line_size,bar_width+line_size,line_size))                       # Top
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(3*margin_size+bar_height+bar_width-line_size,margin_size+bar_height/2-margin_size/2,bar_width+line_size,line_size))       # Bottom
    screen.blit(pygame.font.SysFont(FONT,int(bar_height/10)).render('1',False,(255,255,255)),(3*margin_size+margin_size/4+bar_height+bar_width,margin_size))
    #Motor Number 2
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(3*margin_size+bar_height+bar_width-line_size,3*margin_size/2-line_size+bar_height/2,line_size,bar_height/2+2*line_size-margin_size/2))     # Left
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(3*margin_size+bar_height+2*bar_width,3*margin_size/2-line_size+bar_height/2,line_size,bar_height/2+2*line_size-margin_size/2))   # Right
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(3*margin_size+bar_height+bar_width-line_size,3*margin_size/2-line_size+bar_height/2,bar_width+line_size,line_size))                       # Top
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(3*margin_size+bar_height+bar_width-line_size,margin_size+bar_height,bar_width+line_size,line_size))       # Bottom
    screen.blit(pygame.font.SysFont(FONT,int(bar_height/10)).render('2',False,(255,255,255)),(3*margin_size+margin_size/4+bar_height+bar_width,3*margin_size/2+bar_height/2))
    #Motor Number 3
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(4*margin_size+bar_height+2*bar_width-line_size,margin_size-line_size,line_size,bar_height/2+2*line_size-margin_size/2))     # Left
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(4*margin_size+bar_height+3*bar_width,margin_size-line_size,line_size,bar_height/2+2*line_size-margin_size/2))   # Right
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(4*margin_size+bar_height+2*bar_width-line_size,margin_size-line_size,bar_width+line_size,line_size))                       # Top
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(4*margin_size+bar_height+2*bar_width-line_size,margin_size+bar_height/2-margin_size/2,bar_width+line_size,line_size))       # Bottom
    screen.blit(pygame.font.SysFont(FONT,int(bar_height/10)).render('3',False,(255,255,255)),(4*margin_size+margin_size/4+bar_height+2*bar_width,margin_size))
    #Motor Number 4
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(4*margin_size+bar_height+2*bar_width-line_size,3*margin_size/2-line_size+bar_height/2,line_size,bar_height/2+2*line_size-margin_size/2))     # Left
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(4*margin_size+bar_height+3*bar_width,3*margin_size/2-line_size+bar_height/2,line_size,bar_height/2+2*line_size-margin_size/2))   # Right
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(4*margin_size+bar_height+2*bar_width-line_size,3*margin_size/2-line_size+bar_height/2,bar_width+line_size,line_size))                       # Top
    pygame.draw.rect(screen,[255,255,255],pygame.Rect(4*margin_size+bar_height+2*bar_width-line_size,margin_size+bar_height,bar_width+line_size,line_size))       # Bottom
    screen.blit(pygame.font.SysFont(FONT,int(bar_height/10)).render('4',False,(255,255,255)),(4*margin_size+margin_size/4+bar_height+2*bar_width,3*margin_size/2+bar_height/2))

#serialconnection = serial.Serial('COM7',9600)
pygame.init()
pygame.display.set_caption('game base')
STATE_CONTROLLER = "PS3"
FONT = "Times New Roman"
scale_factor = 3
line_size = 1 #Pixels
margin_size = 3 * scale_factor
bar_width = 30 * scale_factor
bar_height = 100 * scale_factor
small_bar_height = (bar_height - 10 )*scale_factor
screen_size_x = (5*margin_size+3*bar_width+bar_height)
screen_size_y = (bar_height+2*margin_size)
cursor_size = 2*scale_factor
screen = pygame.display.set_mode((screen_size_x,screen_size_y), 0, 32)
clock = pygame.time.Clock()

pygame.joystick.init()
joystick = [pygame.joystick.Joystick(i) for i in range(pygame.joystick.get_count())]

initalization_value = 0
x_y_grid = pygame.Rect(initalization_value,initalization_value,cursor_size,cursor_size)
x_y_grid_color = (initalization_value,initalization_value,initalization_value)
thrust_grid = pygame.Rect(2*margin_size+bar_height,initalization_value,bar_width,initalization_value)
thrust_grid_color = (initalization_value,initalization_value,initalization_value)
motion = [initalization_value,initalization_value,initalization_value,initalization_value,initalization_value]
dead_zone = 0.025
motor1 = initalization_value
motor1_grid = pygame.Rect(3*margin_size+bar_height+bar_width,initalization_value,bar_width,initalization_value)
motor1_grid_color = (initalization_value,initalization_value,initalization_value)
motor2 = initalization_value
motor2_grid = pygame.Rect(3*margin_size+bar_height+bar_width,initalization_value,bar_width,initalization_value)
motor2_grid_color = (initalization_value,initalization_value,initalization_value)
motor3 = initalization_value
motor3_grid = pygame.Rect(4*margin_size+bar_height+2*bar_width,initalization_value,bar_width,initalization_value)
motor3_grid_color = (initalization_value,initalization_value,initalization_value)
motor4 = initalization_value
motor4_grid = pygame.Rect(4*margin_size+bar_height+2*bar_width,initalization_value,bar_width,initalization_value)
motor4_grid_color = (initalization_value,initalization_value,initalization_value)
thrust_vector = initalization_value
#Axis 0: Up    (+) Down  (-)
#Axis 1: Left  (+) Right (-)
#Axis 2: Front (-) Back  (+)
PAUSE = False
port = serial.Serial()
port.port = "COM3"
port.baudrate = 9600
while True:
    screen.fill((0,0,0)) #Wipes Screen
    draw_Screen()
    pygame.display.update()
    if PAUSE:
        for i in range(len(motion)):
            motion[i] = 0
        pygame.display.update()
    while PAUSE:
        for event in pygame.event.get():
            if event.type == KEYDOWN:
                if event.key == K_p:
                    PAUSE = False
    if port.is_open == False:
        port.open()
    print(port.readline())
    #data = serialconnection.readline().decode().strip()
    #print(data)
    if STATE_CONTROLLER == "X56":
        NULL
    elif STATE_CONTROLLER == "PS3":
        motion[2] = motion[4]-motion[3]

    x_y_grid.x = motion[0]*(bar_height-cursor_size)/2+(margin_size+bar_height/2)-cursor_size/2
    x_y_grid.y = motion[1]*(bar_height-cursor_size)/2+(margin_size+bar_height/2)-cursor_size/2
    x_y_grid_color = ((motion[2]+1)*255/2,0,abs(motion[2]-1)*255/2)
```

```
if motion[2] > 0:
    thrust_grid.y = margin_size+bar_height/2-bar_height*motion[2]/2
    thrust_grid.height = bar_height*motion[2]/2
    thrust_grid_color = (0,0,(motion[2]+1)*150/2+50)
elif motion[2] < 0:
    thrust_grid.y = margin_size+bar_height/2
    thrust_grid.height = bar_height*abs(motion[2])/2
    thrust_grid_color = (abs(motion[2]-1)*150/2+50,0,0)
else:
    thrust_grid.y = 0
    thrust_grid.height = 0
    thrust_grid_color = (127,0,127)

if motor1 > 0.5:
    motor1_grid.y = margin_size+bar_height/4-bar_height*(motor1-0.5)/2
    motor1_grid.height = bar_height*(motor1-0.5)/2-margin_size/4
    motor1_grid_color = (0,0,150*(motor1-0.5)*2+50)
elif motor1 < 0.5:
    motor1_grid.y = margin_size+bar_height/4-margin_size/4
    motor1_grid.height = bar_height*(0.5 - motor1)/2-margin_size/4
    motor1_grid_color = (150*(1-motor1*2)+50,0,0)
else:
    motor1_grid.y = 0
    motor1_grid.height = 0

if motor2 > 0.5:
    motor2_grid.y = 3*margin_size/2+bar_height/2+bar_height/4-bar_height*(motor2-0.5)/2
    motor2_grid.height = bar_height*(motor2-0.5)/2-margin_size/2
    motor2_grid_color = (0,0,150*(motor2-0.5)*2+50)
elif motor2 < 0.5:
    motor2_grid.y = 2*margin_size+bar_height/2+bar_height/4-margin_size/2
    motor2_grid.height = bar_height*(0.5 - motor2)/2-margin_size/2
    motor2_grid_color = (150*(1-motor2*2)+50,0,0)
else:
    motor2_grid.y = 0
    motor2_grid.height = 0

if motor3 > 0.5:
    motor3_grid.y = margin_size+bar_height/4-bar_height*(motor3-0.5)/2
    motor3_grid.height = bar_height*(motor3-0.5)/2-margin_size/4
    motor3_grid_color = (0,0,150*(motor3-0.5)*2+50)
elif motor3 < 0.5:
    motor3_grid.y = margin_size+bar_height/4-margin_size/4
    motor3_grid.height = bar_height*(0.5 - motor3)/2-margin_size/4
    motor3_grid_color = (150*(1-motor3*2)+50,0,0)
else:
    motor3_grid.y = 0
    motor3_grid.height = 0

if motor4 > 0.5:
    motor4_grid.y = 3*margin_size/2+bar_height/2+bar_height/4-bar_height*(motor4-0.5)/2
    motor4_grid.height = bar_height*(motor4-0.5)/2-margin_size/2
    motor4_grid_color = (0,0,150*(motor4-0.5)*2+50)
elif motor4 < 0.5:
    motor4_grid.y = 2*margin_size+bar_height/2+bar_height/4-margin_size/2
    motor4_grid.height = bar_height*(0.5 - motor4)/2-margin_size/2
    motor4_grid_color = (150*(1-motor4*2)+50,0,0)
else:
    motor4_grid.y = 0
    motor4_grid.height = 0

motor1 = ((motion[0]+1)/2)
motor2 = (abs(motion[0]-1)/2)
motor3 = (abs(motion[1]-1)/2)
motor4 = ((motion[1]+1)/2)
#print("Motor 1: ",motor1,"Motor 2: ",motor2,"Motor 3: ",motor3,"Motor 4: ",motor4)

for event in pygame.event.get():
    #if event.type == JOYBUTTONDOWN:
        #if event.button == 1: #Eventually this Will be the Kill Drone Button
    if event.type == JOYAXISMOTION:
        if STATE_CONTROLLER == "X56":
            if event.axis == 0:
                motion[0] = event.value
            if event.axis == 1:
                motion[1] = event.value
            if event.axis == 4:
                motion[2] = event.value
        elif STATE_CONTROLLER == "PS3":
            #if event.axis == 0: #LEFT/RIGHT Left Stick
            #if event.axis == 1: #UP/DOWN Left Stick
            if event.axis == 2: #LEFT/RIGHT Right Stick
                motion[0] = event.value
            if event.axis == 3: #UP/DOWN Right Stick
                motion[1] = event.value
            if event.axis == 4: #Left Bumper
                motion[3] = (event.value+1)/2
            if event.axis == 5: #Right Bumper
                motion[4] = (event.value+1)/2
    if event.type == QUIT:
        pygame.quit()
        sys.exit()
    if event.type == KEYDOWN:
        if event.key == K_k:
            if STATE_CONTROLLER == "X56":
                STATE_CONTROLLER == "PS3"
                print("Switching to Controller")
            elif STATE_CONTROLLER == "PS3":
                STATE_CONTROLLER == "X56"
                print("Switching to Joystick")
        if event.key == K_p:
            PAUSE = True
        if event.key == K_ESCAPE:
            pygame.quit()
            sys.exit()

for i in range(len(motion)):
    if abs(motion[i]) < dead_zone:
            motion[i] = 0
    elif abs(motion[i]) > 1 - dead_zone:
        if motion[i] > 0:
            motion[i] = 1
        else:
            motion[i] = -1
    elif abs(motion[i]) < dead_zone-1:
        if motion[i] > 0:
            motion[i] = 1
        else:
            motion[i] = -1
port.close()
clock.tick(60)
```

A1.1: Python code for the AUV Basestation Control System

```
import time
import RPi.GPIO as GPIO
#wiringpi.wiringPiSetup()
#serial = wiringpi.serialOpen('/dev/ttyAMA0',9600)
GPIO.setmode(GPIO.BOARD)
motor1_gpio = 4             #pin
motor2_gpio = 13            #pin
motor3_gpio = 14            #pin
motor4_gpio = 27            #pin
startup_duty_cycle = 0      #%
startup_motor_PWM_Hz = 667  #hz 661-672 are Zero thrust
motor_duty_cycle = 10       #%
motor_max_update_rate = 400 #hz
GPIO.setup(motor1_gpio, GPIO.OUT)
GPIO.setup(motor2_gpio, GPIO.OUT)
GPIO.setup(motor3_gpio, GPIO.OUT)
GPIO.setup(motor4_gpio, GPIO.OUT)
motor1 = GPIO.PWM(motor1_gpio, startup_motor_PWM_Hz)
motor2 = GPIO.PWM(motor2_gpio, motor_PWM_Hz)
motor3 = GPIO.PWM(motor3_gpio, motor_PWM_Hz)
motor4 = GPIO.PWM(motor4_gpio, motor_PWM_Hz)
motor1.start(startup_duty_cycle)
motor2.start(startup_duty_cycle)
motor3.start(startup_duty_cycle)
motor4.start(startup_duty_cycle)
try:
    motor1.ChangeDutyCycle(motor_duty_cycle)
    while True:
        for freq in range(667, 640, -1):
            print("Motor 1 at ", freq, "Hz")
            motor1.ChangeFrequency(freq)
            time.sleep(1.0/motor_max_update_rate)
except KeyboardInterrupt:
    pass
motor1.stop()
motor2.stop()
motor3.stop()
motor4.stop()
GPIO.cleanup()
```

A1.2: Python code for the AUV Drone Control System

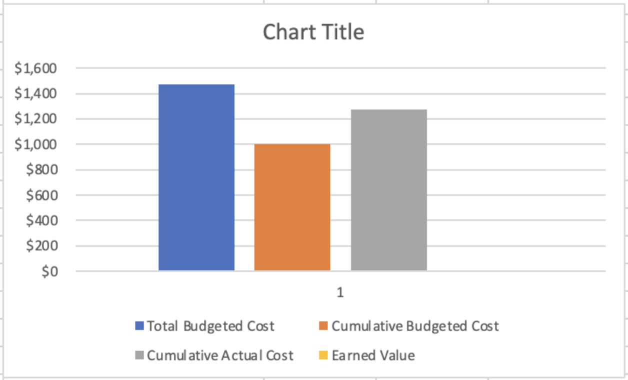
Wage Costs (so far)			
Project Members:	Hours Worked	Cost per Hour	Total Cost per Member
Nick	8	$25	$200
Justin	8	$25	$200
Matt	8	$25	$200
Will	11	$25	$275
Colin	8	$25	$200
Tom	8	$25	$200
		Total Cost:	**$1,275**

Matierals Cost (to be purchased)	
Raspberry Pi	$50
Weights	$30
O-rings and Latex	$30
PBC Piping	$20
Testing Trash can	$30
Epoxy	$40
Total Cost:	**$200**

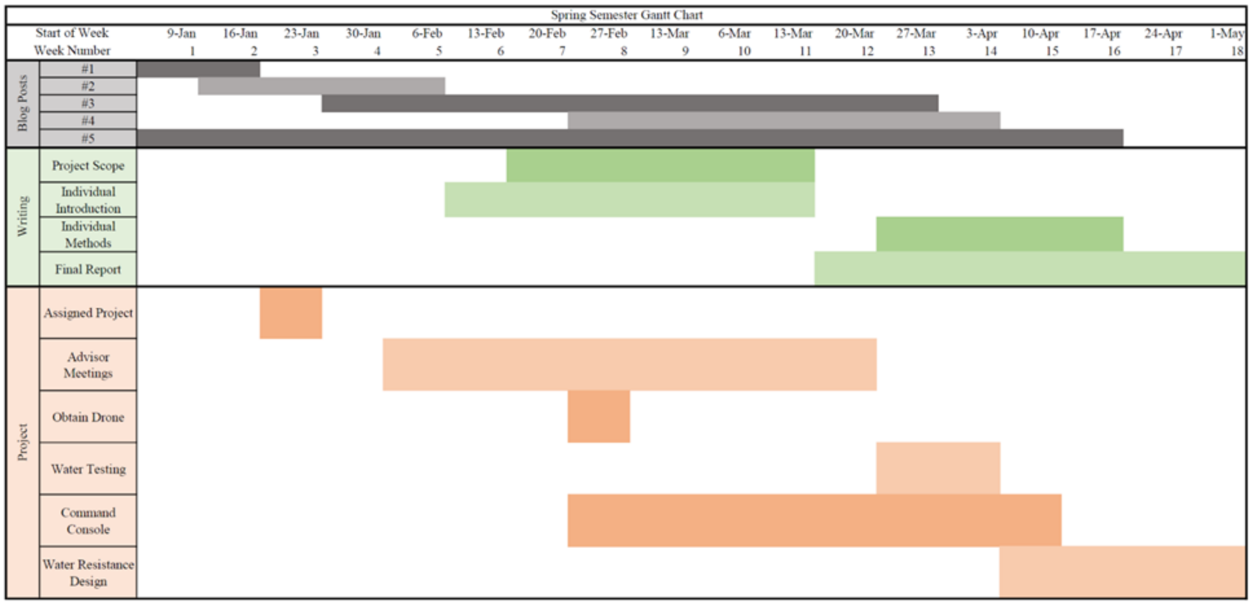
Total Budgeted Cost	$1,475	
Cumulative Budgeted Cost	$1,000	
Cumulative Actual Cost	$1,275	
Cumulative Earned Value	0% completed therefore no earned value	$0
Cost Performance Index	1.15686275	
Cost Variance	13.559322	

Equipment and Software Cost	
Python	$0
Auto CAD	$0
Laptops	$0
Excell	$0
Total Cost:	**$0**

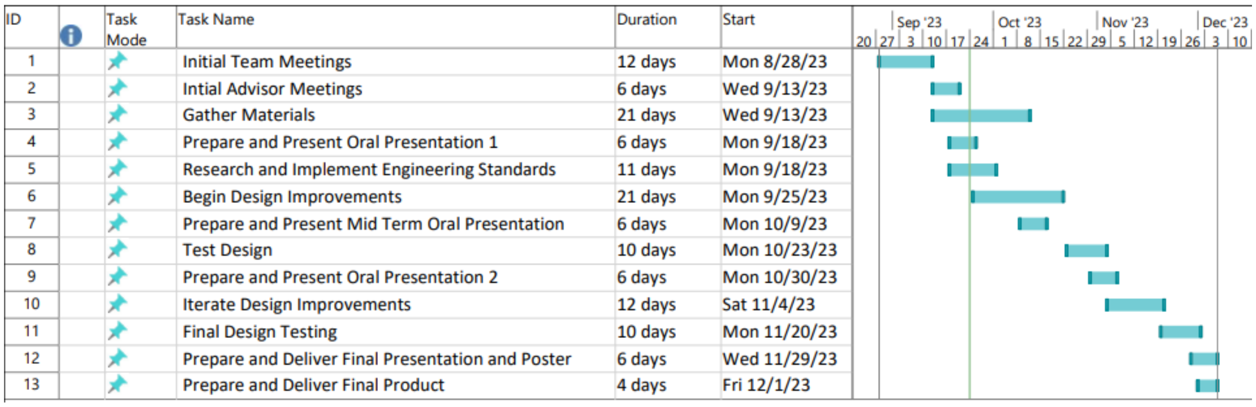
Facilities Cost	
Dr. Kaipa's Pool	$0
Dr. Kaipa's Lab	$0
Total Cost:	**$0**



A2.1: Project Budget



A3.1: GANTT Chart for MAE 434W Discussions and Project Work



ID	ⓘ	Task Mode	Task Name	Duration	Start
1		📌	Initial Team Meetings	12 days	Mon 8/28/23
2		📌	Intial Advisor Meetings	6 days	Wed 9/13/23
3		📌	Gather Materials	21 days	Wed 9/13/23
4		📌	Prepare and Present Oral Presentation 1	6 days	Mon 9/18/23
5		📌	Research and Implement Engineering Standards	11 days	Mon 9/18/23
6		📌	Begin Design Improvements	21 days	Mon 9/25/23
7		📌	Prepare and Present Mid Term Oral Presentation	6 days	Mon 10/9/23
8		📌	Test Design	10 days	Mon 10/23/23
9		📌	Prepare and Present Oral Presentation 2	6 days	Mon 10/30/23
10		📌	Iterate Design Improvements	12 days	Sat 11/4/23
11		📌	Final Design Testing	10 days	Mon 11/20/23
12		📌	Prepare and Deliver Final Presentation and Poster	6 days	Wed 11/29/23
13		📌	Prepare and Deliver Final Product	4 days	Fri 12/1/23

A3.2: GANTT Chart for MAE 435 Project Work

References

[1] Busquets, J., Busquets, J. V., Tudela, D., Perez, F., Busquets-Carbonell, J., Barbera, A., Rodriguez, C., Garcia, A. J., Gilabert J., 2012, “Low-cost AUV based on Arduino open source microcontroller board for oceanographic research applications in a collaborative long term deployment missions and suitable for combining with an USV as autonomous automatic recharging platform,” Proceedings of the IEEE Symposium on Autonomous Underwater Vehicles, Southampton, UK, 24-27 September, 2012. https://doi.org/10.1109/AUV.2012.6380720

[2] Xiang X., Yu C., Niu Z., Zhang Q., 2016, “Subsea Cable Tracking by Autonomous Underwater Vehicle with Magnetic Sensing Guidance,” *Sensors* 16(3): 1-23. DOI:10.3390/s16081335

[3] Khodaryari, M. H., Bolachain, S., 2015, “Modeling and control of autonomous underwater vehicle (AUV) in heading and depth attitude via self-adaptive fuzzy PID controller,” *ASME J Mar. Technol.* 20(3), pp 559-578. DOI:10.1007/s00773-015-0312-7

[4] Li, J.-H., Lee, M.-J., Park, S.-H., Kim, J.-G., 2012, “Real time path planning for a class of torpedo-type AUVs in unknown environment,” Proceedings of the IEEE Symposium on Autonomous Underwater Vehicles, Southampton, UK, 24-27 September, 2012. https://doi.org/10.1109/AUV.2012.6380728

[5] Sahoo, A., Dwivdey, S. K., Robi, P.S., 2019, “Advancements in the field of underwater autonomous vehicles”, *Ocean Engineering* 18: pp 145-160, April 3, 2019. https://doi.org/10.1016/j.oceaneng.2019.04.011