MAE 434W/435 Capstone Project:

Development of An Underwater Autonomous Vehicle

OLD DOMINION UNIVERSITY

Frank Batten College of Engineering & Technology

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

List of Figures

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 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 was to test and modify the existing AUV design. Initial tests were done to ensure that the motors operated correctly and to determine where water infiltration occurred. Although the motors were fully functional, the design was modified so that the AUV is now operated by a single PlayStation 3 controller. Initially, a small amount of water infiltrated the AUV when it was submerged in water. Consequently, the design was modified to create a completely waterproof design. Finally, a properly calculated ballast was installed 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.

Due to the discovery of the hull slowly deteriorating when immersed in water, the AUV was coated with a protective layer of sealant to prevent material deterioration when submerged in water. This material type that we have chosen is contact cement that was applied by hand using a brush. The coating was checked to ensure the solvents used in its formulation did not attack the acrylic plastic which the AUV hull is constructed from. It was also tested during a long term submersion test prior to application where a section of the same plastic was coated and immersed in water for twenty-four hours. At the end of testing it was found that the coating had protected the section. The coating was applied in two coats by hand. In all subsequent immersion tests the outermost layers of acrylic plastic did not become soft or wear away.

To adequately waterproof the AUV and protect it from water ingression several different methods were tested. To understand the original state of waterproofing the single o-ring between a top and bottom grove was tested and water was found inside the hull after ten minutes of testing. At the suggestion of Dr. Kaipa, it was decided to test using rubber gloves and balloons as material to fill any gaps in the grooves which are not occupied by the o-ring. The latex gloves had a thickness of approximately 0.005in and the balloons had a thickness of approximately 0.01 inch. Water was found inside the AUV after testing with the latex gloves but not with the balloons. Despite the promising results of this early test any subsequent test was not able to

replicate the results of the first balloon test. As no permanent modifications could be made to the AUV hull itself, it was decided clay would be used to seal the top cap to the hull as its malleability would allow the grooves where the o-ring is placed to be fully sealed. Before that test was performed it was decided to ensure each cable which penetrated the hull had its mountings secured. After testing with the clay seal no water was found inside the AUV. This result was found repeatedly with subsequent tests.

The control system has been massively changed from its original design. Originally the drone was controlled using four potentiometers connected to a PWM signal generator. The PWM was sent to the ESC driver boards which commanded each motor. This system was not ergonomic nor was it optimal for future automation. This system was discarded in favor of a Raspberry Pi which outputs PWM signals from a Python-based software generator using the library RPi.GPIO. The AUV was changed to a joystick based game console controller which utilized a Python code to generate control inputs. These control inputs were communicated to the AUV using a laptop computer and wireless internet router on the same network, where the router would receive the control inputs and communicate them to the AUV using internet socket communication. Total wireless communication was tested using the Raspberry Pi's wireless communication capabilities however they were unsuccessful.

Proposed Methods

For future proposed methods for the sister AUV team, full autonomy should not be too far from accomplishment. The control system can be connected to a wireless wifi router which connects to the raspberry pi. A PWM generator board will be used in conjunction with the Raspberry Pi to generate more precise and stable PWM signals. A terminal block hat for the Raspberry Pi will be used as a more permanent connection system to the AUV hull. A chassis

system will be implemented so the Raspberry Pi, battery pack, and other navigational and propulsion equipment can be easily retrieved and inserted into the AUV to improve modularity, mission capabilities, and packing density.

While the ballast system has improved from its previous state, active ballasting should be pursued as the implemented system rather than the passive ballasting currently used. The proposed system for active ballasting is to utilize the ballasting systems used by naval submarines. This system would use volumes of air in syringes or bottles which can be displaced with water when the AUV needs to dive and replaced with air when the AUV needs to ascend. The initial ballast system should still remain to help the diving capability of the AUV when an active system is implemented.

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.

Results

Waterproofing the AUV was one of the major facets of the project and after trying various methods and many tests, we have accomplished complete waterproofness. We found that clay provided an adequate water sealant for the AUV. After several tests at various depths and temperatures, we found that the clay completely stopped water infiltration.

The issue of resin deterioration was also fixed. After coating the entire outer surface of the AUV with a waterproof epoxy, we found that there has been no resin breakdown. We have continued to monitor the AUV for resin breakdown, and after almost two months since applying the epoxy, there has been no more resin deterioration.

Furthermore, we have implemented a correctly calculated ballast system. Using buoyancy force calculations, we determined that five and a half pounds of weight need to be added to the AUV to overcome the buoyancy force. To do this, we mounted four PVC pipes filled with lead shot to the exterior of the AUV. After testing with these added weights we found that the AUV is now neutrally buoyant.

Finally, the improved control system design has been fully implemented. While before the AUV's propellers were controlled by turning four different potentiometers, they are now operated by a hand-held PlayStaion3 controller. 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 receives commands via a control cable. The computer takes the signals generated from the control system interface, a laptop computer, via a human interface device, parses the signal, then sends it to the AUV. The AUV will in turn receive the signal and move according to the user input. These results serve as a foundation for making the control systems of the drone architecture capable of supporting a radio interface device for future progress.

Discussion

It is recommended that for future work that a number of improvements be made. The current AUV team has set up multiple routes for improvements in the future. The main improvement being to develop autonomy. For developing autonomy, the code written to fully operate the thrusters can also be used for autonomous purposes. Once the ethernet cord is plugged into the wifi router, the router can then connect to the raspberry pi, hence running wirelessly. The kinks that need to be worked out include slight modifications to the current code, a system where the wifi router can connect to the raspberry pi above water. If a future team is able to find a way to keep the raspberry pi above water and run a control system underwater to the AUV, this would be the best option for autonomy.

It is believed that some sort of buoy system that is above water is the best option for autonomy. If the AUV were able to be tethered to the buoy, this would be ideal. Over time, the current AUV team believes this is the closest to full autonomy that this project can come. The main reason being is because there are not many, if any, known ways to get communication commands through the water to something wirelessly (within amateur bands).

Conclusion

Over the course of the AUV project, multiple milestones have been accomplished. The AUV has developed complete waterproofing, become neutrally buoyant, and zero degradation of exterior material. The control system was drastically improved; it initially took eight different potentiometers to control and has been upgraded to one PlayStation controller. There are still improvements needed in the code to further control the thrusters power, nothing else in the code needs further improvement. Overall, the AUV has made significant progress over the past year and many discoveries were made that would be advantageous for the next group moving forward.

Appendices

```
from ast import AsyncFunctionDef
from asyncio.windows_events import NULL
from cmath import isclose
from math import trunc
from pickle import FALSE
import sys
from unicodedata import digit
from winreg import KEY_CREATE_SUB_KEY
import pygame
import socket
import time
from pygame.locals import *

def digitToString(number):
    if number < 10:
        return('  '+str(number))
    elif number < 100:
        return(' '+str(number))
    elif number < 1000:
        return(str(number))
    else:
        return '000'

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))

s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.bind((socket.gethostname(),1234))
s.listen(1)
clientsocket, address = s.accept()
print(f"Connection from {address} has been established!")
pygame.init()
pygame.display.set_caption('game base')
STATE_CONTROLLER = "XBOX"
FONT = "Times New Roman"
scale_factor = 10
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,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
motor_max_update_rate = 60 #hz
time.sleep(2)
#Axis 0: Up    (+) Down  (-)
#Axis 1: Left  (+) Right (-)
#Axis 2: Front (-) Back  (+)
PAUSE = False
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 STATE_CONTROLLER == "X56":
    NULL
elif STATE_CONTROLLER == "PS3":
    motion[2] = motion[4]-motion[3]
elif STATE_CONTROLLER == "XBOX":
    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)
motor1factor = (motion[2]*motor1+1)/2
motor2factor = (motion[2]*motor2+1)/2
motor3factor = (motion[2]*motor3+1)/2
motor4factor = (motion[2]*motor4+1)/2
##print("Motor 1: ",motor1factor,"Motor 2: ",motor2factor,"Motor 3: ",motor3factor,"Motor 4: ",motor4factor)
motor1data = trunc(motor1factor*1000)
motor2data = trunc(motor2factor*1000)
motor3data = trunc(motor3factor*1000)
motor4data = trunc(motor4factor*1000)

#motordata = digitToString(motor1data)+digitToString(motor2data)+digitToString(motor3data)+digitToString(motor4data)

max_transmission_frequency = 60 #hz
clientsocket.send(bytes(str(motor1data),"utf-8"))
time.sleep(1/(max_transmission_frequency*4))
clientsocket.send(bytes(str(motor2data),"utf-8"))
time.sleep(1/(max_transmission_frequency*4))
clientsocket.send(bytes(str(motor3data),"utf-8"))
time.sleep(1/(max_transmission_frequency*4))
clientsocket.send(bytes(str(motor4data),"utf-8"))
time.sleep(1/(max_transmission_frequency*4))
time.sleep(1/motor_max_update_rate)


#clientsocket.send(bytes(motordata,"utf-8"))
#time.sleep(1/(max_transmission_frequency))
#time.sleep(1/motor_max_update_rate)

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
        elif STATE_CONTROLLER == "XBOX":
            #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 == "XBOX"
                    print("Switching to Joystick")
                elif STATE_CONTROLLER == "XBOX":
                    STATE_CONTROLLER == "X56"
                    print("Switching to Joystick")
            if event.key == K_p:
                PAUSE = True
            if event.key == K_ESCAPE:
                pygame.quit()
                sys.exit()
                socket.close()

    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
    clock.tick(60)
```

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

```
import time
import math
import socket
import RPi.GPIO as GPIO
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ipv4_default = '10.161.80.254'
ipv4 = input('What what is the IPv4 Address of the Local Host: ')
if ipv4=='localhost':
    s.connect((socket.gethostname(),1234))
else:
    s.connect((ipv4,1234))
print('Connection Established with '+str(ipv4)+'!')
def thrustclamp(thrustvalue): #Clamps values between -1 and 1
    if thrustvalue>1:
        return 1
    elif thrustvalue<-1:
        return -1
    return thrustvalue
pwmFrequency = 50.75 #hz
def thrustToDuty(thrustvalue):
    thrustvalue=(thrustclamp(thrustvalue)+1)/2
    return thrustvalue*5+4.5
GPIO.setmode(GPIO.BOARD)
motor1_gpio = 11            #pin
motor2_gpio = 13            #pin
motor3_gpio = 15            #pin
motor4_gpio = 16            #pin
startup_duty_cycle = 7      #%
#motor_max_update_rate = 16 #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, pwmFrequency)
motor2 = GPIO.PWM(motor2_gpio, pwmFrequency)
motor3 = GPIO.PWM(motor3_gpio, pwmFrequency)
motor4 = GPIO.PWM(motor4_gpio, pwmFrequency)
motor1.start(startup_duty_cycle)
motor2.start(startup_duty_cycle)
motor3.start(startup_duty_cycle)
motor4.start(startup_duty_cycle)
motor_statup_period = 2 #seconds
print('Motors Initizlizing! Waiting '+str(motor_statup_period)+'second(s)!')
time.sleep(motor_statup_period)
print('Motor Startup Completed Receiving Signals from Remote.')
max_transmission_frequency = 25
motor_max_update_rate = 400
try:
    while True:
        s.send(bytes(str('1'),'utf-8'))
        motor1data = int(s.recv(32).decode("utf-8"))/1000*2-1
        time.sleep(1/(motor_max_update_rate*4))
        motor2data = int(s.recv(32).decode("utf-8"))/1000*2-1
        time.sleep(1/(motor_max_update_rate*4))
        motor3data = int(s.recv(32).decode("utf-8"))/1000*2-1
        time.sleep(1/(motor_max_update_rate*4))
        motor4data = int(s.recv(32).decode("utf-8"))/1000*2-1
        time.sleep(1/(motor_max_update_rate*4))
        #print("Motor 1: ",motor1data,"Motor 2: ",motor2data,"Motor 3: ",motor3data,"Motor 4: ",motor4data)
        motor1.ChangeDutyCycle(thrustToDuty(motor1data))
        motor2.ChangeDutyCycle(thrustToDuty(motor2data))
        motor3.ChangeDutyCycle(thrustToDuty(motor3data))
        motor4.ChangeDutyCycle(thrustToDuty(motor4data))
        #rawmotordata = s.recv(32).decode("utf-8")
        #print(rawmotordata)
        #motordata = int(rawmotordata)
        #print(motordata)
        #motor4 = int(motordata % 1000)
        #motor3 = int((motordata % 1000000 - motor4)/1000)
        #motor2 = int((motordata % 1000000000 - (motor3+motor4))/1000000)
        #motor1 = int((motordata % 1000000000000 - (motor4+motor2+motor3))/1000000000)
        #print("Motor 1: ",motor1,"Motor 2: ",motor2,"Motor 3: ",motor3,"Motor 4: ",motor4)
        #thrust4 = thrustclamp((motor4-500)/1000)
        #thrust3 = thrustclamp((motor3-500)/1000)
        #thrust2 = thrustclamp((motor2-500)/1000)
        #thrust1 = thrustclamp((motor1-500)/1000)
        #print("Motor 1: ",thrust1,"Motor 2: ",thrust2,"Motor 3: ",thrust3,"Motor 4: ",thrust4)
        #motor1.ChangeDutyCycle(thrustToDuty(thrust1))
        #motor2.ChangeDutyCycle(thrustToDuty(thrust2))
        #motor3.ChangeDutyCycle(thrustToDuty(thrust3))
        #motor4.ChangeDutyCycle(thrustToDuty(thrust4))
        time.sleep(1/(max_transmission_frequency))
except KeyboardInterrupt:
    pass
motor1.stop()
motor2.stop()
motor3.stop()
motor4.stop()
GPIO.cleanup()
s.shutdown(socket.SHUT_RDWR)
s.close()
```

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

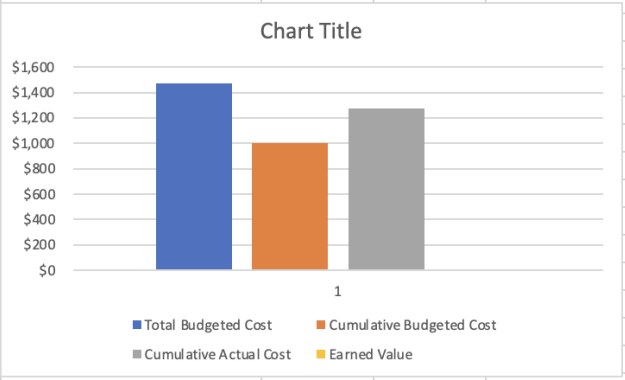
Wage Costs (so far)			
Project Members:	**Hours Worked**	**Cost per Hour**	**Total Cost per Member**
Nick	8	$25	$200
Justin	8	$25	$200
Matt	8	$25	$200
Will	11	$25	$275
Colin	8	$25	$200
Tom	8	$25	$200
		Total Cost:	**$1,275**

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**

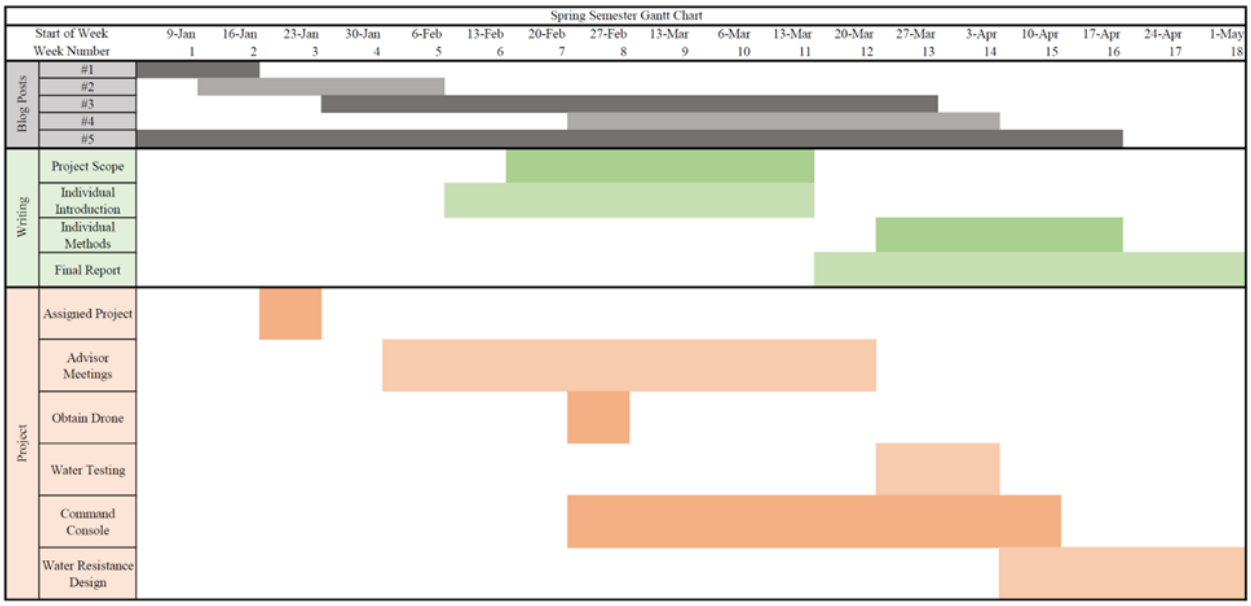
Equipment and Software Cost	
Python	$0
Auto CAD	$0
Laptops	$0
Excell	$0
Total Cost:	**$0**

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

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	

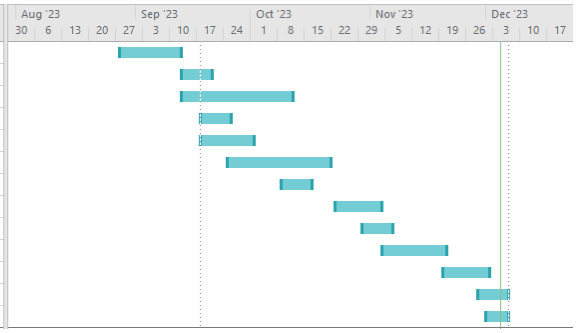


A2.1: Project Budget



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

	Task Mode	Task Name	Duration	Start	Finish
1		Initial Team Meetings	12 days	Mon 8/28/23	Tue 9/12/23
2		Intial Advisor Meetings	6 days	Wed 9/13/23	Wed 9/20/23
3		Gather Materials	21 days	Wed 9/13/23	Wed 10/11/23
4		Prepare and Present Oral Presentation 1	6 days	Mon 9/18/23	Mon 9/25/23
5		Research and Implement Engineering Standards	11 days	Mon 9/18/23	Sun 10/1/23
6		Begin Design Improvements	21 days	Mon 9/25/23	Sat 10/21/23
7		Prepare and Present Mid Term Oral Presentation	6 days	Mon 10/9/23	Mon 10/16/23
8		Test Design	10 days	Mon 10/23/23	Fri 11/3/23
9		Prepare and Present Oral Presentation 2	6 days	Mon 10/30/23	Mon 11/6/23
10		Iterate Design Improvements	12 days	Sat 11/4/23	Mon 11/20/23
11		Final Design Testing	10 days	Mon 11/20/23	Fri 12/1/23
12		Prepare and Deliver Final Presentation and Poster	6 days	Wed 11/29/23	Wed 12/6/23
13		Prepare and Deliver Final Product	4 days	Fri 12/1/23	Wed 12/6/23



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