Northrop Grumman Collaboration Project
California Polytechnic University, Pomona
Unmanned Underwater Vehicle

Overview

The Unmanned Underwater Vehicle (UUV) was designed by Andrew Lawson in the 2015-2016 school year at Cal Poly Pomona. In the original design, it featured seven thrusters- four Blue Robotics T-100 Thrusters on each corner, two Blue Robotics T-200 Thrusters on the sides, and an unknown Blue Robotics Thruster underneath the vehicle pointed left-to-right. The vehicle was tethered via a large Ethernet cable capable of communication with the vehicle's computer systems and another cable that provided power. The computer systems onboard are unknown. The team was composed entirely of seniors and did not return for another year.

In the second year of its life (2016-2017), the vehicle team was led by Harjot Singh. While the team was small and lacked a full software division, they were able to create two battery boxes to power the vehicle without the need of a power tether. They also were able to program a basic "figure 8"-like maneuver. The communications team also demonstrated the ability to communicate from the bottom of a pool to the surface via ham radios. All members of this team did not return for a second year.

At one point in the vehicle's life, a torpedo, ballast system, and claw were designed and tested, but never implemented. The date at which these occurred is unknown but suspected to be during the 2016-2017 year.

Now in the 2017-2018 year, the vehicle team is led by Dominic Holloman. This team is solely composed of sophomores/second year students. The team did not have knowledge of the vehicle's status, composition, or progress until they first saw the disassembled vehicle in the third week of winter quarter. The team was not allowed access to the vehicle for total 16 weeks of the 27 allotted weeks. Regardless, a claw, ballast system, autonomous navigation, and expanded battery life were successfully demoed on May 19, 2018. The main issues faced by the team have been a lack of documentation, explanation, and, most critically, knowledge.

This book is to aid future teams in an understanding of what transpired this year (2017-2018) and how the vehicle is organized. Should anything be changed, rearranged, or removed, it is expected that this document is to be kept up-to-date by the reigning team.

Good luck in your endeavors and take great care of Andrew Lawson's project.

Capsule

General:

The main capsule is made of acrylic, flanked by two stainless steel faceplates and held in place by aluminum extrusions. The front denoted with a strip of white electrical tape wrapped around the handle, as well as googly eyes placed on the metal. On the front plate, nine holes are drilled into the faceplate, allowing for external cables to pass through without allowing water into the vehicle. On the back there are six similar holes. Each hole is guarded with a set of complex nuts and cord grips, rubber caps, and plastic inserts which, when tightened, allow for each hole to be waterproofed. Only the top hole on the front face plate does not have any of these protections. It instead is meant solely for the Blue Robotics Bar30 High-Resolution 300m Depth/Pressure Sensor. Once this is inserted and its nut tightened on the inside portion of the faceplate, the hole is water-tight. This depth sensor was damaged and must be re-ordered; its wires are easy to tear.

Waterproofing:

To water seal any cable too small for the core grips, slice a "(NSF -51) PUR ATE -6m (4 x 6) MAX W.P. 180 PSI/1.3MPA" cable down one side and split it open like a hot dog bun. Insert the cable and cover with heat shrink. Slide this into one of the ports on the faceplate and tighten the cord grip ON the heat-shrinked cable. If the core grip is tightened on a region before or after the heat-shrinked cable, water will leak into the capsule.

Electrical

Power:

The UUV is powered by two sets of batteries. The first set (two rechargeable $LiFePO_4$ 4 cell, 12.8V, 32,400mAh (32.4Ah), student-made batteries) powers the thrusters. These batteries were made to fit within the battery boxes attached underneath the UUV. During testing, however, we found Battery Box 1 to leak. In retrospect, we believe this could have been due to the power cables exiting the battery boxes not being fully tightened; as such, we regarded this battery to be water-damaged. Regardless, we did not want to risk water flowing inside the vehicle and we needed two extra ports in one of the faceplates. As such, we removed all components from Battery Box 1(instead using it to hold weights) and decided to solely use Battery Box 2 to house the batteries. This means only one of the two batteries is being used.

There are 12 extra battery cells in the lab, contained in small, white, cardboard boxes. These 3.2V 5400mAh batteries were purchased on eBay from “athomemarket” for a price of $79.99 for 60 Fullriver batteries.

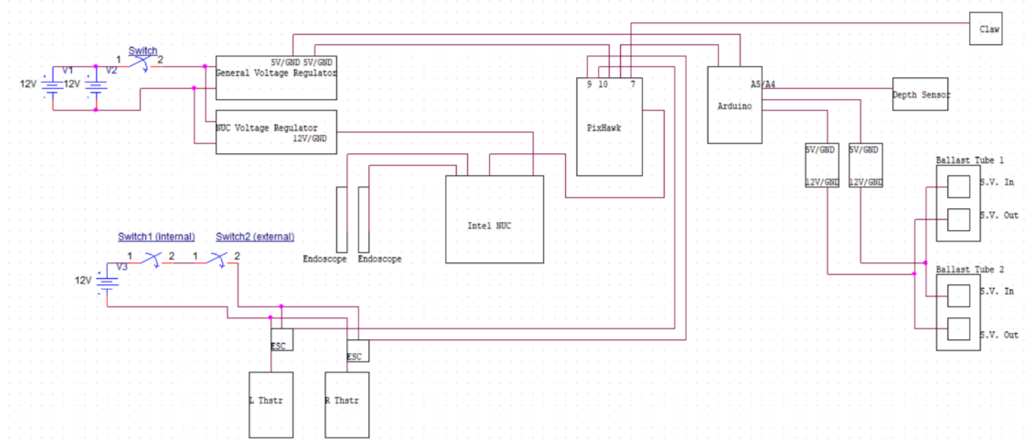


Figure 3.1 General overview of UUV electronics

The next two batteries are Stark Power 12V 9Ah Deep Cycle $LiFePO_4$ batteries. These are used to power all the computing devices/electronics. The two mounted on the vehicle are new as of May 11, 2018. The one original battery in the lab was bloating, bulging, and not holding its charge as long as it ought to- signs of a decaying battery. They are wired in parallel; the set provides 12V with a 18Ah capacity. The current set up is overkill, certainly, however, this does ensure the vehicle can last for quite a long time in the water. The exact time it can last has not been tested. *Regardless, if the batteries are well kept, then power should never be an issue*.

Should power mysteriously cut out, <u>pull the vehicle out of the water</u>. This may be a sign of water leaking into the vehicle.

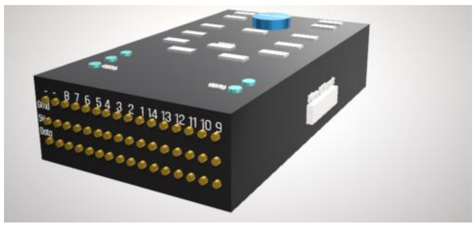
Power from these batteries is routed through specific devices before they power their respective components. The massive 32.4 Ah batteries feed in through the front faceplate and connect to a bar underneath the main electronics tray. This distributes the power to whichever thrusters are connected. As of the 2017-2018 demo, only the two T-200 Thrusters on the sides of the vehicle were connected to power. Power from the thrusters is controlled by the heavy flick switch and a second “safety” switch mounted on the exterior. Both must be “on” for power to flow to the thrusters.

Power from the two 12V 9Ah batteries is routed through two voltage regulators- a purple board underneath the top tray which powers the Arduino controlling the ballast system and the power controlling the data pins on the PixHawk, and a green board mounted on top the top tray which powers the Intel NUC.

<u>Computers:</u>

Now on to the computing devices- the Intel NUC runs Linux (Ubuntu to be specific) and manages the object detection using HSV (Hue, Saturation, Value). Plugged into its USB ports are the PixHawk and two endoscopes, one protruding from each face plate. When the main electronic switch is flipped, the NUC will be powered but will not be turned on. This is necessary since, as of writing, the PixHawk and the endoscopes must be plugged into the vehicle in a specific order before pressing the power switch on the NUC. The current code assigns a label to each item based on the order in which they wrere inserted, not by which port. *Because of this, it must be followed exactly*. The method is outlined in the software chapter.

The PixHawk is a complex microcontroller with an integrated accelerometer, array of input pins, and running convoluted software called "MavLink." While the software will be discussed in detail in the software chapter, the pin layout and subsequent requirements will be described here. There are three rows of pins on the PixHawk with 16 columns. The top row is ground, the middle is 5V input, and the bottom is data. From left to right, column 1 and 2 are not usable, columns 3-9 are labelled in the software as pin column 8-1, and columns 10-16 are labelled as 14-9. The right thruster is connected to pin 16 (pin 9 in the software) and the left thruster is pin 15 (pin 10). The claw is connected to pin 3 (pin 8). 5V and ground must be connected somewhere along the columns- the exact position is not important as of now so long as it is not columns 1 or 2. The 5V are being supplied by the purple voltage regulator along the bottom of the top tray which receives its power from the two 12V, 18Ah batteries. Be careful with the ports on the purple voltage regulator- they are easy to break, and the board may be custom made. During our testing, we broke two of the 5V/GND plugs. Re-soldering the pieces proved to be fruitless.



The Arduino controls the ballast system. The ballast system uses four solenoid valves- two per tube. In each tube, one valve controls CO_2 into the balloons and the other controls CO_2 out of the balloons and into the water. The filling and releasing of CO_2 in the balloons will allow the vehicle to rise and dive in the water. These solenoid valves require 12V and are polarity protected. However, they do not have any data input- they are naturally closed and open only when power is applied. Since we do not have any computers capable of outputting 12V, we use an AND gate, two step-up voltage circuits, and the power/ground from the PixHawk to control the solenoid valves. The Arduino is coded to interpret data from the depth sensor and, based on its position, send a current through one of two digital pins to the AND gate. This

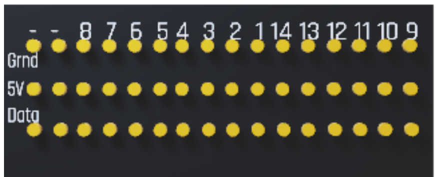


Figure 3.2 3D model of PixHawk
(if viewing electronically, it can be manipulated)

current is AND'ed with the 5V from the PixHawk and its result is then stepped up to 12V and supplied to either the "in" solenoid valves or the "out" solenoid valves. The PixHawk can also send a signal to the Arduino once the mission is complete, causing the vehicle to surface. While we were not able to test this full system, the code and digital logic work. The limiting factor in this system is the current output from the step-up circuits. Since the Arduino cannot supply ample amperage, we switched our power source to the PixHawk (and therefore the 12V, 18Ah batteries). We have not tested since this switch.

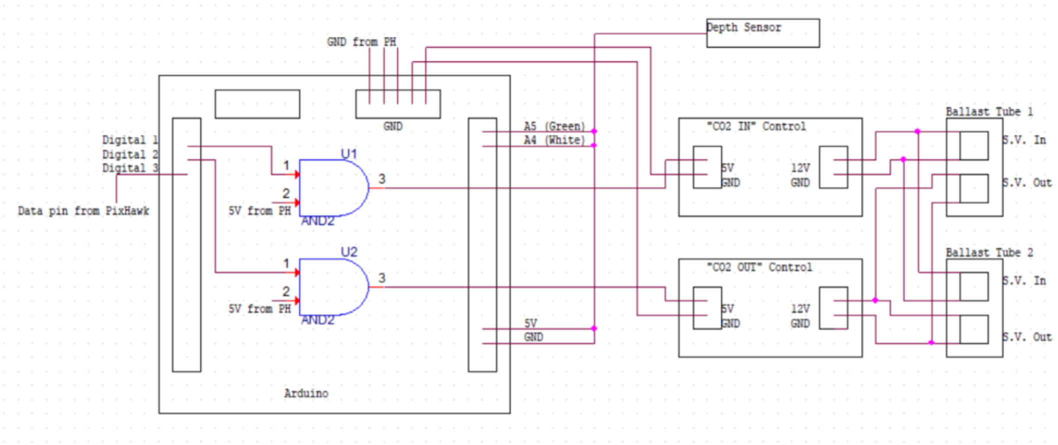


Figure 3.3 Arduino/Ballast system control

This system could be upgraded to work solely off the PixHawk- connect the depth sensor to the PixHawk then use the data pins from the PixHawk. The PixHawk could be coded to maintain a specific depth and send a current through one of two data lines to control the solenoid valves. The data line may not be able to provide the voltage or current required for the solenoid valves, so the PixHawk 5V power rail should be AND'ed with data line. The result can then be stepped up to 12V.

Sensors:

The endoscopes on either end of the vehicle were chosen due to their size- the diameter of the cameras is exactly that of the cord grips on the faceplates, allowing them to easily be moved. If a camera of any other size is used, neither the camera nor its plug can fit through the cord grips, meaning if the camera is to be inserted and water-sealed, it must be cut and re-soldered inside the vehicle's capsule. This would then prevent the camera from being moved again without cutting and soldering it once more. We felt this was too big of a risk.

The endoscopes are low resolution, peaking at 720p. The camera itself is fragile and prone to breaking if pulled too hard. If the metal cap is pulled free from the camera and the camera continues to work, it can be repaired with the marine heat shrink. Keep in mind, this may make removing the camera more difficult.

The endoscope cable is one meter long with a potentiometer near the end which controls the brightness of the lights which surround the camera lens. On the end of the cable is USB-C and can be connected to a smartphone with a USB-C port (assuming the phone software is downloaded). Should one break, they are available to purchase from Amazon for roughly $16.50 and are eligible for Amazon Prime shipping.

The depth sensor is approximately $70 from Blue Robotics with multiple libraries available to download from their website. These libraries allow the depth sensor to be used on a variety of devices including Arduino and PixHawk ArduSub. Four cables protrude from the sensor and are glued to the inside of the sensor. This means the cables are stiff and resist movement. Because of this, they are prone to tearing. We tore the cables to our depth sensor due to this. Be careful.

We started the year with the Imaginex 852-00-140 Sonar. We were only able to retrieve depth data, not echo locational data. After many weeks, we have determined this unit might have a manufacturing defect or was broken in prior years. As such, we did not utilize the device. Dr. Boskovich currently possesses the unit so that he may test it as well.

The Blue Robotics T-200 Thrusters require 12V and have a maximum of 16A draw each. Two are installed on the vehicle and are mounted on either side. In the current design, only these are being used to direct the vehicle. Four T-100 Thrusters are mounted on each corner with the intention of moving the vehicle up and down. These are currently unused.

These motors are controlled by small electronic speed controllers (ESC's) which can be manipulated with various pulse width modulations (PWM's). This allows us to control the thrusters as though they were servos. It is from these ESC's we power the thrusters- the power strip mounted on the bottom tray connects to these devices. These ESC's also need power and have a basic power/data/ground plug extending from the board to the PixHawk. When we received the vehicle, these had been wired so that each data cable was separate but all ESC's shared a ground cable and the 5V input cable had been cut. We repaired only two sets of these cables to use with the T-200 thrusters. Currently, the each of the two ESC's we are using are soldered to the power/data/ground cables from each T-200 Thruster. The boards then connect to the power strip below the cable, sending the 12V across the thrusters. From there, the brown/yellow/red cables connect to the PixHawk in either pin 9 or 10, depending on the side.

The thrusters must be fully submerged in water when powered. If not, they will fry and the thrusters will be ruined. There is no lubricant in the thrusters- they require water. These thrusters are around $200 each.

When the thrusters are running, they must be kept fully submerged and around 3 to 4 inches underneath the water. When they are too close to the surface, a vortex begins to form. Air will funnel into the motors. While we never tested to see if this damaged the motors, we took no chances. We feel this ought not to be tested unless spares are affordable.

Claw:

The claw is controlled by a simple, waterproof servo. Its power/data/ground wires connect to the PixHawk and are controlled via PWM's. One improvement would be to purchase a servo with greater gripping strength and more range of motion.

Switches:

There are three main, hardware switches installed on the UUV. One controls power from the Stark Power 12V batteries to the electronics and the other two control power from the custom batteries to the ESC's/thrusters. One of these two switches (Switch 2 on the schematic) is redundant, purposefully, for safety. It is currently uninstalled due to the wires connecting the switch to the power distribution board being too thin of a gauge. Since there is a maximum of 8A flowing through that circuit, the small, 22 gauge wire **must** be replaced with the thicker 14 gauge wire. If not, these wires could melt if the max current is met. Once the wire is exchanged, the second, safety kill switch can be added. The switch controlling power to the electronics (mounted next to the NUC) and Switch 1 on the schematic (mounted on the middle tray) are all internal. Switch 2 is external, placed on the back faceplate. This switch is waterproof.

Software

______Working on the UUV has allowed me to learn so much more about teamwork, cooperation, and most importantly dedication. This was by far one of the most intriguing projects I have ever worked on in my college career. I can honestly say that this robot would not be at this stage without the hard work of every single individual on this team. Coordination between both the hardware and the software team allowed the vision of Andrew Lawson to become an even better reality. What you put into this project really reflects what you get out of it. Do not let the challenges get the best of you; maintain a positive mindset every step of the way. Persevere and tackle any hardships that arise, whether it be a locked door or a lock screen. Only time will tell how deep this UUV can go. Best of luck team!

~ Varun Venkitachalam (Software Lead)

Mavlink_control:

This file takes functions that are defined from autopilot_interface and allows them to be executed. It is the most important file for the UUV since it is the primary file that allows it to move. In this file there are many different sections that show the breakdown of mavlink. The primary section that is required for the UUV to move is known as "serial_port.start()" and "autopilot_interface.start()". These mavlink defined methods allow the compiler to know when to start reading the threads and look for commands.

Starting from the very top of the file, there are different files that have been included for the execution of this c++ code. It is important to make sure that these files are available in the mavlink_control.cpp folder. DO NOT DELETE any file even if it seems unnecessary in this folder since it is required for the code to compile properly. One example of a file that may seem useless is "stdafx.h". This file is required for opencv and object detection to work properly.

Video capture is how our webcams are used to detect objects. capWebcam is webcam 0 and webcam2 is webcam 1. capWebcam represents the front facing camera on the UUV and webcam2 represents the back. The rest of the code in this initial section continues to initialize our webcams by setting their resolution to a fixed value followed by a fixed height and width. Our HSV image is also initialized in this section and is explained thoroughly in the HSV section of this document.

This year mavlink was used to control the thrusters on the UUV as servo inputs. Based on the pulse width modulation (PWM value) the speed at which the thrusters moved was controlled. Here is a link to our thruster documentation explaining the significance of each PWM value in relation to thrust speed: here. We recommend testing the functionality of the code on continuous servos first before testing it on the actual thrusters. **IMPORTANT**: the thrusters will NOT move if they are not initialized with a **PWM value of 1500** when starting the code. Here is an example from the blue robotics website. This was one major problem we personally experienced and hope that no other team ever experiences it again. It is also very important to make sure not to run the thrusters above water. This can immediately fry the thrusters rendering them completely useless.

In order to compile your changes to mavlink_control, you must open the folder in terminal and simply type in “make”. Cmake is not necessary for local changes. Once it has compiled with no errors, run the executable file as any normal file is run through linux. “./mavlinkControl “ The most efficient way we tested our code was through servos directly plugged into the pixhawk. As a reminder, the folder that we used was titled “c_uart_interface_exampleOPENCV “ which is located on the desktop. Make sure to select the proper folder, since there are others with similar names.

Autopilot_interface:

All functions must be made in the autopilot_interface.cpp and autopilot_interface.h files before they can be used in mavlink_control.cpp. The files already contain several functions. The functions that we use are only arm_pixhawk, disarm_pixhawk, and write_set_servo.

The arm_pixhawk will start the pixhawk and ready it to receive commands. The pixhawk itself will display a solid blue light in the middle to indicate its armed status.

The disarm_pixhawk will halt the mission and the pixhawk will no longer receive commands. The pixhawk itself will display a blinking blue light in the middle to indicate its disarmed status.

The write_set_servo function controls the sub’s propellers. The function has two parameters: “const int &servo” and “const int &pwm.” The servo parameter selects which propeller to control. The servo number depends on which pin the propeller is connected to in the Pixhawk. The pwm parameter controls the thrust speed of the propeller.

Movement Algorithm / Motor Control:

The code that allows the UUV to move is found in the mavlink_control.cpp file, starting from “autopilot_interface.start()”. In order to understand what this code is doing, the general search algorithm must first be understood. The vision for the UUV was seen in 5 different vertical section. These sections were labeled as L left, left, middle, right, and R right. L left being section 1 and R right being section 5. When the UUV is searching for the object, the vision software is continuously checking if it sees an object in any of these sections.

If the object is right in the middle, the left thruster and the right thruster will move forward, allowing the uuv to thrust forward. If the object is in the left section, the left thruster will stop and the right thruster will slowly thrust allowing the uuv to turn left. L left section is a hard left, right is a soft right, and R right is a hard right. This vision algorithm will continuously search and home towards the object until it has centered the object. Once the object is in grabbing distance based off of the dArea value, the UUV slightly reverses to see if the object has been grabbed. If it has not been grabbed, the claw opens again and it continues searching. Once the object has been successfully grabbed, the vision turns off Camera A and switches to Camera b, the back camera.

Understand that Camera B is on the backside of the UUV and in a vision perspective, the thrusters have switched sides. This means that if the vision detects the object in the L left section, the left thruster must move in the reverse direction. Similarly if it detects in the R right section, the right thruster moves in reverse making a hard right from a vision perspective. This pattern is followed until the home object has been centered and reached a certain dArea. Once this is achieved, the code stops executing and the pixhawk disarms (starts to blink blue).

In regards to the code in mavlink_control.cpp, servo port 9 is always the right thruster. Servo port 10 is the left thruster and servo port 7 is the claw. Do not use the draw contour section that has been commented out, as this often causes the execution to close unexpectedly due to the amount of processing that is required.

Starting from the first while loop, grab is 0 when the object has not been grabbed yet; foundObj is to compare the found dArea with the current dArea when the UUV is reversing to confirm that it has grabbed the object. The grab value is incremented when the object is in the center section and the dArea is equal to the proper grabbing distance. If it is not in the grabbing distance, then the code checks if the object is straight on. If it is not straight on it checks from L left to R right. Once the object has been grabbed, the code goes to camera B, which is the second while loop. An important note is that the units for the dArea is unknown, and the values in the code have been obtained from trial and error.

Camera A is closed at this point and the while loop continuously executes until homeReach does not equal 0. If homeReach equals 0, that means it has not reached home (the starting position). Once we have confirmed that the uuv has reached home, the motors are set to a pwm of 1500, which means to stop, the claw is set to a pwm of 1700, which means to open, and homereach is incremented, which finally breaks the loop. Once the loop has been broken, the mission is complete so the pixhawk is disarmed and the serial_port() is stopped.

Cmake / Execution commands:

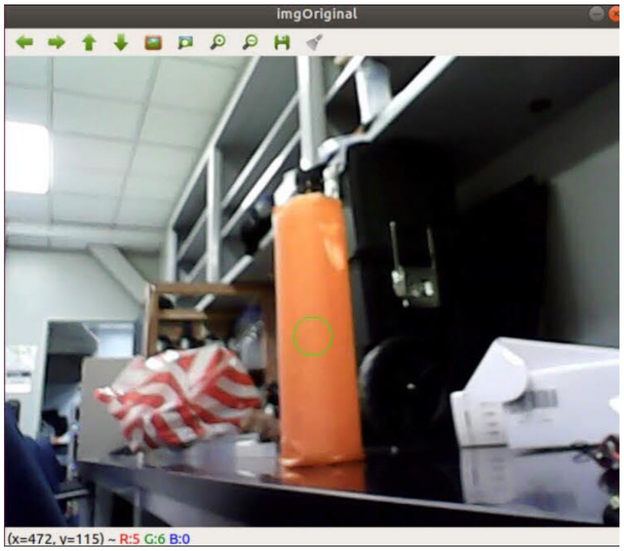
The first step that we took was download and compile the Mavlink code. Mavlink allows scripts on the Nuc to send commands to the Pixhawk so that the Pixhawk can control the sub. The complete guide to the installation can be found in the UUV Drive > 2017-2018 > Team #4 Software Team > Mavlink > Mavlink Tutorial (direct link: tutorial). The package will contain necessary folders such as autopilot_interface, mavlink_control, CMakeLists.txt. The purpose of autopilot_interface and mavlink_control will be discussed in the next sections of the documentation. CMakeLists.txt contains all the possible messages/commands that can be used to create new functions. The steps to create new functions are also in the Mavlink Tutorial.

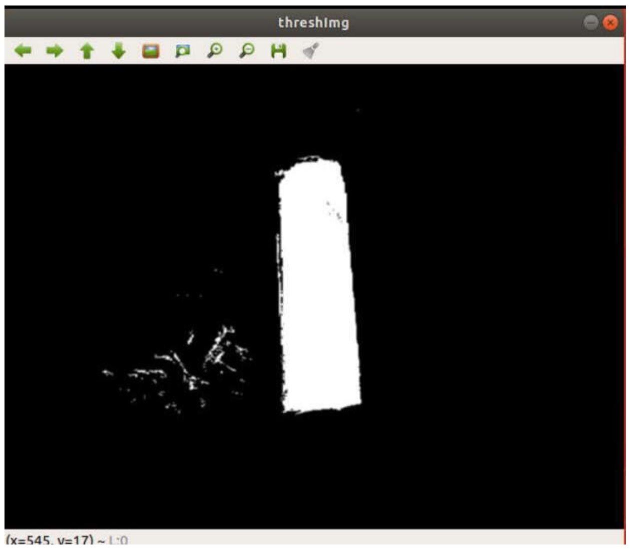
OPENCV / HSV:

The computer vision that has been used for object detection is done through OPENCV. OPENCV is the platform that allows our webcams to actively detect objects. Some of the code that is in mavlink_control.cpp is included specifically for the opencv libraries in order to map an x and y coordinate of the detected object. The vision was also broken into 5 different sections based off of the available x coordinates from our vision scope.

One problem that was experienced when using opencv was to make sure that the proper version was being used. Initially, we had somehow acquired version 4.0 of opencv, which was not even released this year, resulting in the webcams not being detected. We then downgraded our version using git checkout and went to the proper branch. All opencv versions are available on github and can be changed by changing the branch. Opencv must also be included and set to required in cMakeLists.txt in order for it to be detected in mavlink_control.cpp. If for some reason opencv is uninstalled, it must be relinked to the project and the you have to run cmake again after including it in the cMakeLists.txt.

HSV is the method we used to detect color underwater. The reason why HSV is better for detecting color than RGB is because HSV searches for one particular color, identifies it, sets that color to white, and makes everything else in the image black. This means that there is less room for failure since the intended color has been distinguished from its background.





In order to accurately get the HSV value for your object, point the webcam at the screen and check what color is detected from this webs. Based on the changes in HSV determine whether the Hue, Saturation, or value needs to be changed. Then change it in the commented sections that have a number for hue high, hue low, etc. Once your object has been clearly outlined in white in the hsv image, adjust your dArea value for the most optimal results to know when the object should be grabbed.

ArduSub / QGroundControl:

Ardusub is the software that allows the submarine to be detected and allows the mavlink code to directly interface with the vehicle. In other cases, QgroundControl is ideally used by planes and drones in order to set waypoints for a way to navigate to different areas. The issue in using waypoints for the UUV is that there is no GPS underwater, so there is no way to get the coordinates to the specific location; thereby making waypoint navigation impossible. Instead, for the purposes of the UUV, QGroundControl was used to set limits on the PWM values that were used by the servo input.

Since navigation was primarily done through the two motors, importing custom configurations for our specific UUV was not required. We had initially believed that QgroundControl was not required for us anymore, because of controlling the motors as servos; however, eventually realized through trial and error that the webcam would begin to lag severely every time we had restarted our intel nuc and ran mavlinkControl without opening QGroundControl. In order to prevent this lag, QgroundControl must be opened EVERY TIME the nuc is restarted. An important disclaimer is that mavlinkControl cannot be running at the same time as QGroundControl since the pixhawk transfers data from the same port. The workaround we discovered was to open Qgroundcontrol, let it transfer the required data for a few seconds, close QgroundControl, and then run the mavlink command. This is demonstrated in the script used for autonomous navigation. (The script is found in the Dummy folder in the desktop). Overall, if QgroundControl is not opened, your output feed from the webcam is guaranteed to be laggy.

Pixhawk:

The Pixhawk model that we used was the 2.4.8 model. The Pixhawk should be connected to the Nuc via the USB cable. The safety switch button should be inserted in the port labeled “switch” in the middle of the Pixhawk. The light on the switch should be solid red so that Pixhawk can run. On the side of the Pixhawk, there should be pins with number labels. The numbers should match the numbers used for the servo parameter in the write_set_servo function in mavlink_control.cpp. The ordering of the pin starts with servo 1 at the pins labeled “1” at the middle (7th set of pins from the right) and the order should move leftward. The “SB” pins should be servo 9 and “RC” pins should be servo 10. Servo 11 should start at the right most pins that are also confusingly labeled “1”. The servo ordering continues from 11 at those rightmost pins to servo 16 at the pins labeled “6” (6th set of pins from the right, not the 5th set of pins from the left). Servo 7 should be for the claw, servo 8 should be for the battery, servo 9 should be for the right propeller, and servo 10 should be for the left propeller. If the claw or propellers are inserted in other ports, the write_set_servo functions in the mavlink_control.cpp file must be adjusted to use the correct servo ports

There is also a red switch that is attached to the pixhawk which allows the servo ports to be accessed. If the switch is blinking red, that means that it is off and must be turned on. In order to turn on the switch, grab the red light with your thumb and press it gently until the color changes from blinking red to a solid red color. Once this color is solid, your pwm changes will be transferred over to the servo input.

The Pixhawk also has LED lights slightly below the center that indicates the status of the Pixhawk. Flashing blue indicates that the sub is not armed yet and will not move at all. Solid blue indicates that the sub is armed and will begin its mission. Note that once the solid blue appears, the sub will immediately begin moving if a target is spotted. The Pixhawk LED will occasionally show double flashing yellow. In this case, ArduSub must be reinstalled/updated in QGroundControl.

Lockscreen:

One of the biggest hurdles the software team experienced was understanding why the UUV would not work when testing in the pool. In the drive there is a video of us testing the movement and vision of the UUV by putting the thrusters in a bucket and checking whether they were reacting in the intended way. These tests worked perfectly without fail; yet when testing in the water we never heard the motors initialize. After debugging the situation, we discovered that the lockscreen would appear every time the nuc was powered without an hdmi cable attached to a monitor. This was the reason why the motors were not initializing, since the script was never being run.

The solution was to bypass the lockscreen by going directly into the boot files through the terminal and removing it. The reason why the simple lockscreen removal button didn't work is because our nuc is dual booted, resulting in the nuc to believe that a lockscreen is required every single time. The moral of this section is to let every person know to never bring that lockscreen back in this nuc. The password that was required to sign in was NorthropGrumman2018 , but it was removed in an attempt to bypass the lockscreen.

One way to ensure that the lock screen is gone is to power the nuc without plugging it into a monitor, waiting about 2 to 3 minutes to let it boot up, and then plugging an hdmi cable into the monitor. If the script is executing, then the lockscreen has been bypassed. If the lock screen is still visible, then it still needs to be fixed. As far this year, we have successfully bypassed the lockscreen for the script to be executed effectively.

Port Ordering:

The specific order of plugging in the ports must be done every time the sub needs to do its mission. The order is available here: order. These steps are necessary since Ubuntu’s device detection is dependent on the order of devices inserted, not by the port locations. For example, if the camera is inserted first then it is considered as “device1.” It does not matter if it is plugged in the front or back port, as long as it was plugged in first, it will always be considered “device 1.” The order of detected devices will be gone once the Nuc is unplugged. If any of the steps are not strictly followed, the sub may not properly arm or the camera directions may be reversed.

File Searcher:

The most important folder that contains all of the contents for the execution of the UUV is titled c_uart_interface_exampleOPENCV. This folder has mavlink up to date and has integrated OPENCV for object detection. It is located on the desktop of the nuc and can be accessed right at boot up. In order to run this folder right click it and select, "open in terminal". Then type "make" to compile your code and then type " ./mavlinkControl" to execute your code. The code must be executed with the pixhawk plugged in and the two webcams attached to the nuc or it will display an error.

The other folder that is important is titled "Dummy" and is also located on the desktop of the nuc. This folder contains the script that allows the UUV to work autonomously. **IMPORTANT:** When setting the initial 10 minute delay for set up, always make sure to set the delay on the script and not on the mavlink_control.cpp file. This guarantees that the delay will actually be processed and the motors will not accidently move when above water.

In order to run the script you must search "startup applications" on the search bar and open it up. This allows the user to select the scripts they choose to run at bootup. There should already be a path to the Dummy folder in startup applications.

The port for the pixhawk should be TTYACM0, for camera A it should be video0, and for camera B it should be video1. Search up how to check active ports on linux to confirm that these ports are accurately listed.

Ballast System (more insight):

The hardware team did an amazing job of assembling and implementing the huge cylindrical tubes around the UUV, each connected with a series of solenoid valves and pipes that control the flow of air. The four corner tubes each hold a balloon attached to the end of the valve pipe that inflates or deflates based on software commands to allow the UUV to submerge or ascend in water. The front and rear cylinders were mainly used to add weight to the UUV and provide a safer cushion so that the vehicle does not get damaged as easily when it runs into walls or obstacles. Unfortunately, the 2017-2018 team was unable to fully test and finalize this portion of the project but significant research and progress has been made and recorded for both hardware and software.

The basic idea behind the need for this functionality in the UUV is the following. Because of how buoyant the vehicle was (not heavy enough to fully submerge in the water), there was a need to find a method to add more weight to the vehicle. Rather than adding on more electronics (which would not have worked considering how packed the inside of the capsule was) or hanging dumbbells to the frame of the vehicle (this method was attempted but failed due to the imbalance it would cause), a ballast system approach was taken to capture a more realistic replication of what a submarine would do.

The ballast system solenoid valves control the flow of air upon receiving an electrical current signal. Depending on the programming of these valves, the valves are able to be "opened" which allows air to enter to raise the vehicle above water (using the CO_2 canister). It can also be “closed” which release the air in the valves and submerge the UUV. (Refer to the Google Drive team folder: Software -> Ballast System).

The ballast system ultimately did not get tested because there needed to be a solution to delivering the correct amount of current from the arduino microcontroller to the solenoid valves. Since an arduino board is only capable of outputting 5V (volts) maximum while the solenoid valves needed much more than that (12V), it was not possible to directly wire the two to make the components of the ballast system work with the software logic. A “step-up” device and an AND gate logic was used as a potential solution but these ultimately did not transfer sufficient voltage from the arduino board to the solenoid valves.

Overall, if a working solution is found to allow for effective current transfer between the arduino microcontroller and the solenoid valves, then the ballast system should most definitely be in working state. In terms of software, every code for the ballast system was recorded based on changes made to reflect how it was programmed to get to the final code.