nscc
Ivany Campus

OCEANS TECHNOLOGY

OTAD 5009 – OCEANS TECHNOLOGY PROJECT II

AQUATIC MINI OBSERVATION SYSTEM WINCH FOR SENSOR DEPLOYMENT

2020-04-20

FOR: STEPHANE KIRCHHOFF, ALFRED WHITE. INSTRUCTORS. NSCC.
FROM: MAYA ANURAJ, BROOKE MOORE, RAJ PATEL, CHUCK TAYLOR. STUDENTS. NSCC.

Short Abstract

This report details the activities involved in the AMOS Winch project for the Oceans Technology Project course in the NSCC Oceans Technology program. Discussion includes collaboration with industry partner *InNatureRobotics* and instructors at the college, the design process, task delegation, prototyping, and a reflection on lessons learned.

Table of Contents

1 Table of Figures

AQUATIC MINI OBSERVATION SYSTEM WINCH FOR SENSOR DEPLOYMENT

1 Introduction

There is a standing need to gather information about the ocean's key physical properties. Datasets that include features such as temperature, conductivity, salinity, dissolved oxygen, and pH are constantly sought after by scientists and researchers alike. Capturing ocean data is crucial to improving the global collective understanding of our ocean's processes. Measurements in varying depths and different environments are required, some of which may be too dangerous for human exploration. An alternative way to obtain this data is to introduce an instrument capable of traversing the water column.

As is similar in many fields, one of the primary goals of modern innovation is automation. Automation allows for cost savings and is often a safer approach given it usually removes most or all of the human labour. Automation is helping progress society from the former craft system to mass production, providing better work, higher wages, more jobs, and better living standards (Pollack, Fitzpayne, McKay, 2019).

One method for measuring ocean properties that is becoming more sought after in the field recently is using Autonomous Surface Vehicles (ASVs). ASVs are robotic vehicles that operate on the water surface collecting oceanographic data (National Oceanography Centre, 2020)

The biggest limitations with current ASVs is a lack of depth profiling capability (National Oceanography Centre, 2020). This challenge is currently at the forefront of ASV research and development.

InNatureRobotics is a start-up company in Fredericton, New Brunswick. Part of the work Murray Lowery-Simpson is doing with his ASV (the Aquatic Mini Observation System (AMOS)) is to create a lightweight, low cost system with a small automated winch capable of deploying sensors down through the water column. Murray provided this project idea to the NSCC Oceans Technology program as a potential capstone project.

This report outlines the worked involved in developing an automated winch system which could potentially be used on the deck of an ASV.

1.1 Objective

The initial aim of this project is to engineer an automated winch that would meet the parameters requested by *InNatureRobotics* for AMOS. AMOS' technical parameters were the following:

- Maximum weight of system: Approximately 1 kilogram
- Weight of sensor payload: Approximately 5 kilograms
- Cable length (in water): 30 metres
- Able to rotate 180° in order to secure sensors over the vessel or on the deck

After performing some research, initial testing, and close discussions with NSCC faculty regarding the mechanical potential of a winch system with the technical parameters of the AMOS system, it was decided that it was not practical to engineer a system weighing approximately 1kg that could execute the aforementioned functions with a payload of 5kg using the resources available to the group. While a system could be digitally modeled and 3D printed using industrial plastics, a 5kg payload would exceed the system's mechanical capabilities.

Given that a system superstructure constructed with aluminum would be lightweight yet magnitudes stronger than a plastic system, this became the new approach for the final design. Aluminum would also be better suited to enduring the marine environment. An aluminum structure might in the end be too heavy for the current version of AMOS, given it structure is engineered using ultra-lightweight materials such as Styrofoam and plastics, however an aluminum design could be used with future iterations of the AMOS system as well as have broader applications for the ASV market as a whole. A lighter payload was also considered in the final design (approximately 1kg).

2 Technical Approach and Considerations

2.1 Project Schedule

SMART WINCH PROJECT SCHEDULE - SEMESTER 1			
Month	**Date**	**Activity**	**Progress**
September	25th	Formalizing of groups	General discussion
October	2nd	Skype meeting with Murray Simpson	Discussion
	9th	Discussion on parts	Brainstorm ideas
	16th	Electromechanical system planning	Made goals
	23rd	Meet on moving forward	Solidified plans
	24th	Placing order for stepper motor/Shield	Ordered initial parts
	25th	Skype meeting with Murray	Reassessed constraints
	29th	Meeting with Murray at COVE	AMOS winch prototype demo
	30th	Meeting with Alfred/Stephane	Made Letter of Intent
November	1st	Researching ideas - Arduino & Superstructure	Picked up servo motor
	5th	Meeting with Dan Bolivar	Suggestion to use Mechanical Actuator
	13th	Meeting with Dan Bolivar	Collected Aluminum Prototyping Kit
	20th	Meeting with Peter Oster	Soldered Motor Shield
	NA	Group work on presentation and technical reports	Completed Journal, Project Report
December			

SMART WINCH PROJECT SCHEDULE - SEMESTER 2

Month	Date	Activity	Progress
January	15th	Assessment of Reports	Divided work in sub-teams
	24th	Discussion on superstructure	Materials selected
	26th	Prototype Discussion	Made wooden parts
February	2nd	Modelling in Inventor 3-D	Took dimensions for parts
	3rd	Finding parts - screws, nuts, bolts etc.	Found screws from Peter
	4th	Got Fishing Reel - Planned for programming	Brainstormed the main structure
	5th	Produced drawings for superstructure	Final dimensions were made
	6th	Meeting with Luke to print 3-D parts	Modelled 3-D parts in Inventor
	9th	Discussed placement of parts on circular base	Files sent for 3-D printing
	11th	Collected parts and machined circular base	Exact parts are being collected
	13th	Brainstormed to find more parts	Found fasteners and hardware for structure
	14th	Attempt to make L-brackets from Aluminum strips	Found screws and made support angles
	15th	Work on Bluetooth module	Found baud-rate and learnt IMME setup
	16th	Work on stepper motor	Connected stepper motor to motor-shield
	17th	Met with Peter to troubleshoot Stepper motor	Resolved issue of changing library
	18th	Brainstormed ordering of parts	Discussion with faculty to order future parts
	19th	Work on mechanical super-structure	Drilled holes and made support shafts
	20th	Worked towards making longer support shafts	Made support shafts to perfection
	23rd	Testing of weight capacity of servo motor	Weight capacity was sufficient to lift all parts
	25th	Meeting with Dan - assemby of parts started	Drilled holes onto both support shafts
	26th	Received joystick and Bluetooth module	Worked to connect it to the Arduino
	29th	Drilled holes onto base and assembled slender arms	Assembled slender arms and tested weight on spool
March	4th	Troubleshooting Arduino and physical hardware	Connection between two Arduino boards established
	5th	Made Inventor 3-D file for Motor Stand	Collected it in next two days from Luke
	6th	Decided screw size and pulley size to be used	Assembled motor onto motor stand and bought pulley
	7th	Worked on Arduino Programming	Troubleshooted serial connection
	14th	Worked on Bluetooth module	Connected Arduino Uno and Mega
	15th	Aim to finish superstructure	Worked to assemble final Aluminum superstructure
	16th	Last day to work in OTAD workshop	Assembled remaining parts onto superstructure
	20th	Resolving issues for Bluetooth module	Bluetooth module connected successfully
	22nd	Make Arduino work with Module & Joystick	Successful Connection
April	2nd	Meeting on Microsoft Teams with Faculty	Discussed plans during COVID-19 Pandemic
	7th	Group Meeting on Microsoft Teams	Delegated tasks on reports and presentation

Figure 1 Project schedule indicating activities and tasks completed with their respective dates.

2.2 Concept Design

The project design process began with preliminary conceptualization of a small automated ASV winch system. This was done as a group through discussion and guidance from NSCC faculty and Murray Simpson.

The following sections break down the design process involved in the preliminary design phase, and then breaks down the mechanical and electronic aspect of the process from start to finish individually.

2.2.1 Preliminary Design

The preliminary design drawing (Figure 2) was developed to align with the requirements provided by Murray Simpson and the recommendations by the NSCC faculty.

Due to the time constraints, a decision was made to develop a mechanical system rather than an electromechanical system. The structure requirements involved:

- supporting a 4-5 kg sensor payload;
- the ability to deploy 30 meters of cable; and
- detecting the payout distance using a magnetic or optical encoder.

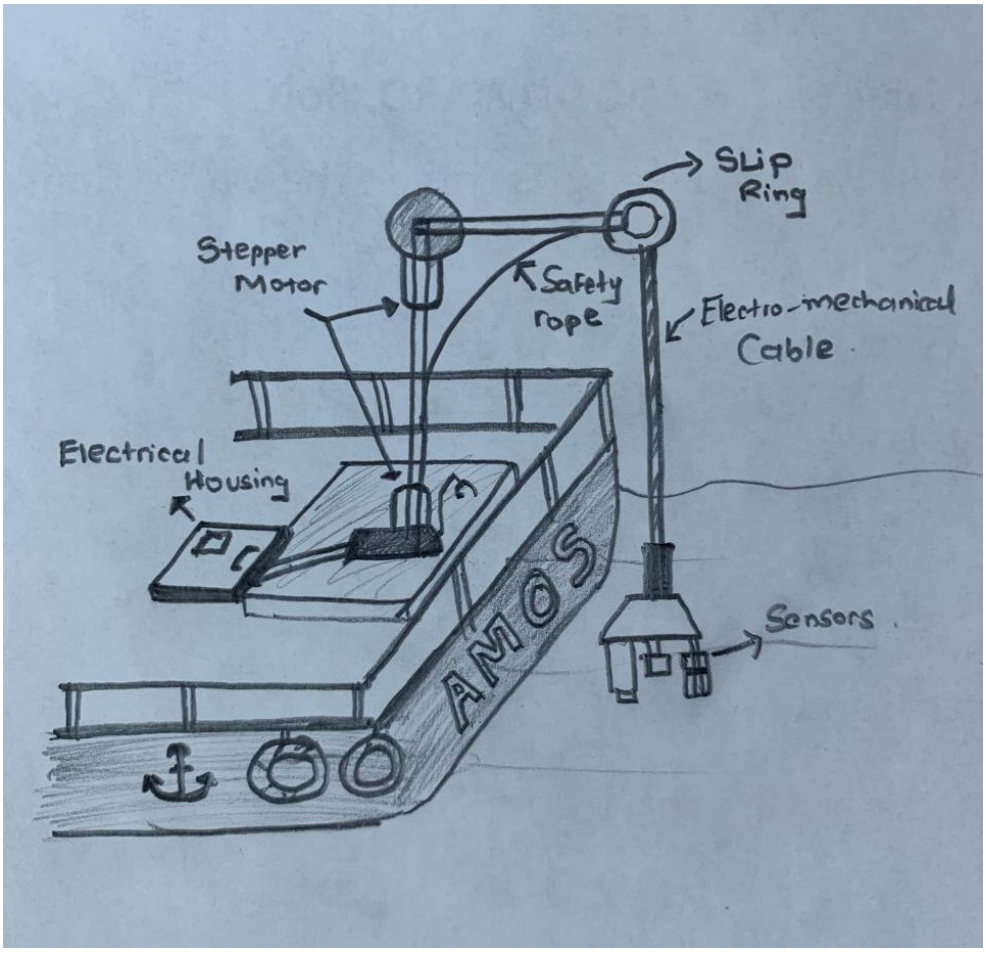

Figure 2 Preliminary design sketch produced by Raj Patel.

Final Concept Design Drawing

Winch system parameters were re-evaluated after additional recommendations were provided by the NSCC faculty and Murray Simpson. Reducing the weight constraint and removing the slipring requirement from the design were the main points of discussion.

Another concept design drawing (Figure 3) was developed that implemented the recommendations as well as incorporated a fish reel which will be used as the drum. The drum will hold the cable and rotate through the actuation of a stepper motor.

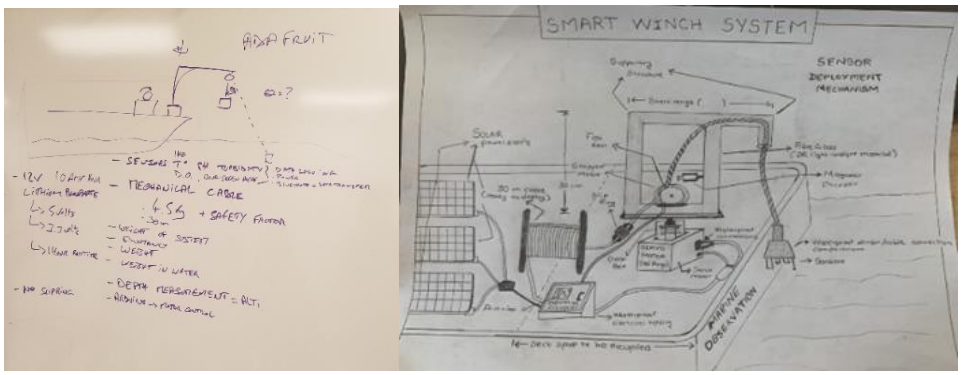

Figure 3 Second concept design drawing that was developed implementing the recommendations and incorporation of a fish reel to be used as the drum based on discussion with Murray Simpson and the NSCC faculty.

The following points were discussed:

- battery backup for real-time data backup
- determine how much torque is needed to lift the components
- use of magnetic encoder to determine the length of the cable deployed
- winch system should rotate 180° in order to house the sensors on the deck
- the system should have an automated failsafe emergency stop system in case of entanglement
- two motors will drive the system (servo motor for rotational movement, and stepper motor for cable deployment and retraction)
- waterproofing components
- Arduino for the motor control
- how to model it without a slipring (non-electromechanical system (i.e., mechanical))

2.2.2 System Constraints

The following points were taken into consideration while developing the winch system:

- small, lightweight crane design
- stepper motor shaft insertion to replace the arm of the fish spool (drum)
- slipring if the final prototype is electromechanical
- the second motor can be a gearbox if more torque is required
    - gearboxes are used to increase torque while reducing the speed of a prime mover output shaft (i.e. a motor crankshaft)
    - the output shaft of a gearbox rotates at a slower rate than the input shaft, and this reduction in speed produces a mechanical advantage which increases torque
- should be able to be fully deploy 30 m of cable within 60 seconds.
- winch system will be exposed to marine, wind, and collision conditions
- servo motor has constant torque while the stepper motor has a higher holding torque, however, as the speed of the motor increases, torque reduces
- stepper motor draws full current (either in the case of sitting still or running), and as it is constantly drawing current, it can produce a heating effect
- the servo motor only draws as much as current it requires and as demanded by the application, which allows it to remain cool
- stepper motor: lesser rpm, 2000 rpm or less, when a lot of torque is required at the low end
- servo motor: higher speed, 2000 rpm and higher, with higher dynamic acceleration

2.3 Mechanical Design Methodology

2.3.1 Objective

The development of the winch system superstructure was one of the preliminary design tasks in order to achieve the main goal of the project. The superstructure should:

- be a stable mechanical crane-like structure which will support the sensor payload;
- account for the torsional effect of the motor; and
- account for the bending caused by the pulling of the cable installed.

2.3.2 Prototype 1

The team collaboratively tried to develop an aluminum-based prototype in which the hardware and aluminum members were obtained through a mechanical prototyping kit provided by Dan Bolivar. The pre-tapped members were fastened in the shop. The structure shown in Figure 4 depicts the preliminary design in which the drum is installed such that the cable aligns with the vertical top member. Two sheaves were used to reduce cable tension.

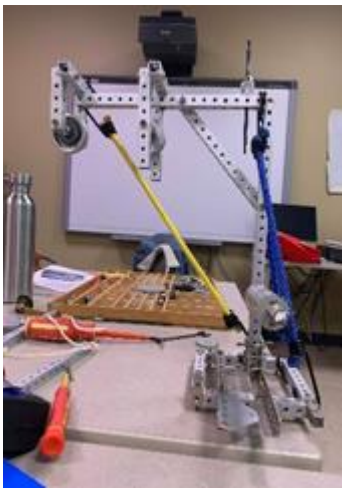

Figure 4 Prototype 1 using the mechanical prototyping kit which depicts the preliminary design in which the drum is installed such that the cable aligns with the vertical top member. Two sheaves were used to reduce cable tension.

2.3.3 Prototype 2

The prototype assembled during the first semester provided the team with a foundation to build upon. The winch system superstructure design going forward consisted of:

- a rectangular plastic base in which the electronic hardware would be installed;
- the servo motor being mounted underneath the base in order to actuate the superstructure; and
- the crane members, spool, and stepper motor being mounted on top of the base.

The mechanical parts were obtained. The team attempted to utilize any hardware that was available in the college to reduce cost. For example, the rectangular and circular base was composed of Delrin plastic and was obtained from the mechanical workshop. The fish spool was provided by NSCC faculty. The fish spool is a spinning reel for known for its longer cast and smooth retrieval.

Figure 5 Mechanical members (a) circular plate (b) rectangular plate (c) spinning fishing reel (d) arms put together.

Crane components

The crane members were modelled using Autodesk Inventor (Figure 6). The vertical arm was 20 cm long while the lateral members were 15 cm long. The designs were sent to the NSCC mechanical shop faculty and were then 3D printed. To ensure a lightweight design, a honeycomb lattice material was used rather than solid plastic material. The 3D print material is composed of Delrin plastic.

The members were fastened together with aluminum brackets (Figure 5d) to increase strength, reducing the stress and bending of the arms.

The drill bit used was from the Renard series sequence, M20, for flat headed screws of size 10-24 thread size, with 2" length.

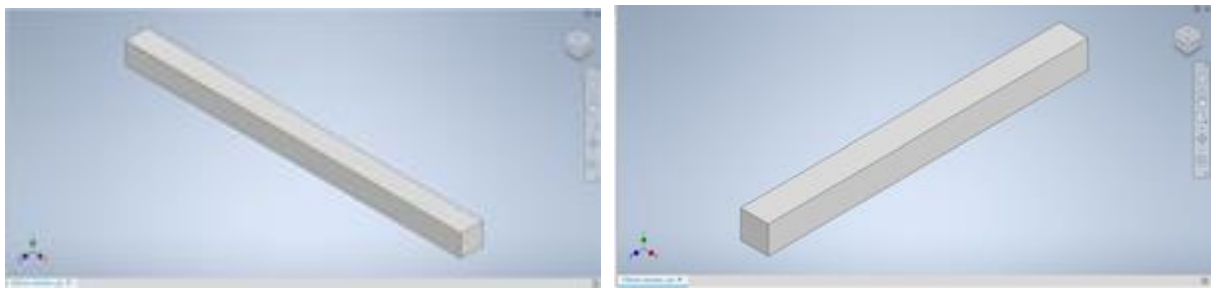

Figure 6 3D modelled crane members.

The aluminum brackets used were cut from an aluminum angular shaft scrap material by using a bandsaw. The brackets were cut to the fit the dimensions of the crane members.

After assembling the arm structure and fastening the structure to the circular base, the team tested the reel by positioning the spool where it was intended to be fastened.

The arms were supposed to support a sensor payload of at least 2 kg but there was high stress on the members indicated by the bending of the material. It was decided to use aluminum arms rather than 3D printed plastic arms for the next prototype.

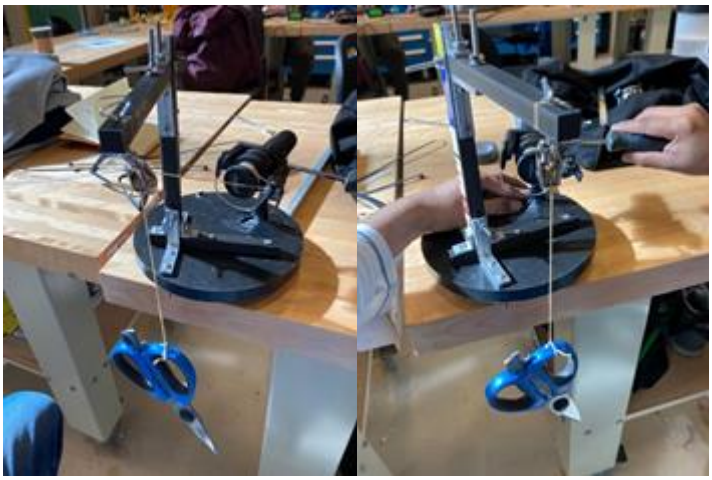

Figure 7 Testing the ability of the winch superstructure to hold the sensor payload weight.

A motor base stand was 3D modelled and printed (Figure 8). The base would create the height necessary to connect the shaft of the servo motor to the fish spool. The shaft of the servo motor was machined to suit the dimensions of the fish spool hole in which the shaft is to be inserted.

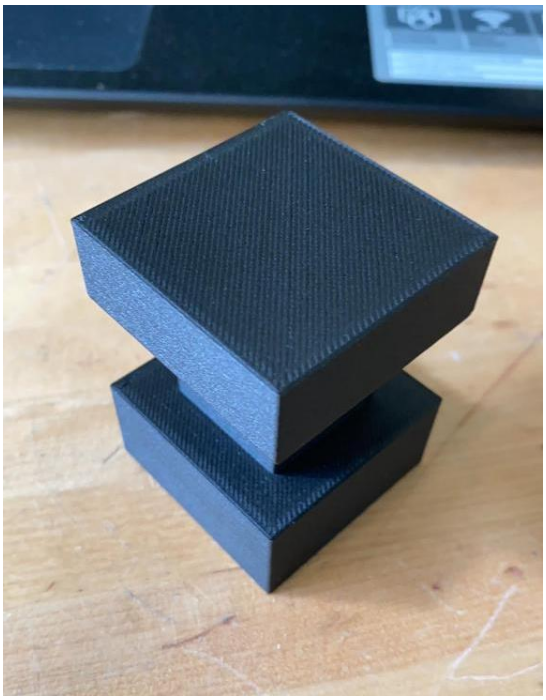

Figure 8 3D printed motor stand base.

2.3.4 Prototype 3

The third prototype was developed to be identical to prototype 2 (Figure 9). However, the 3D printed plastic arms and aluminum brackets were not strong enough for the purpose of our project. In using aluminum-based arms, the stress on the arms had been reduced significantly. The assembly process went as such:

- we used a bandsaw to cut a hollow aluminum piece into three arm parts using the dimensions from prototype 2;
- used a drill press to drill the holes into the pieces using the dimensions from prototype 2;
- used majority of the aluminum support brackets from prototype 2 but cut new ones as well because we needed to drill new holes for better fastening; and
- we used fasteners to secure the superstructure.

Figure 9 Aluminum-based arm members for prototype 3. Fastened to the same circular base.

Due to COVID-19 and the limitation of access to workshop tools, Gorilla Epoxy glue was used to bond:

- the stepper motor to the motor base stand which was bonded to the circular base in a position such that the motor shaft could be fitted to replaced the arm of the fish spool;
- the circular base to the servo motor which was glued to the cardboard – wood pallet platform; and
- the swivel-eye pulleys to the upper arm.

Electrical tape was used to increase the surface area in which the gorilla epoxy glue could better adhere to. Fish line was taken from an old fish spool and lined onto the relevant fish spool (drum). The winch structure when fully assembled had shown to be sturdy (figure 10), however, the glue was not able to bond the servo motor to the circular base as well as the other components. This would have not been an issue if we had access to appropriate tools to secure the winch system.

Figure 10 Final assembly of the structure

2.4 Electronic Design Methodology

2.4.1 Electronic Concept Design

Main objective

To develop a winch system that is capable of the following:

1. 180° rotation from starting position – actuated by a servo motor.
2. Deploying a cable (fishing line) via a spinning fish reel – actuated by a stepper motor.

Solution

Use an Arduino UNO microcontroller and its associated motor shield to drive the two motors which in turn will induce the 180° rotation and the deployment of cable tasks.

Use another Arduino microcontroller to control the motors remotely via Bluetooth modules.

Research & Development

Hardware

- Arduino UNO Rev3 (3)
- Arduino MEGA 2560 Rev3
- Adafruit Motor Shield v1 (2)
- NEMA 17 Bipolar Stepper Motor (2)
- DSSERVO DS3218 PRO 20 kg Digital Servo Motor (2)
- DSD TECH HM-19 Bluetooth Module (2)
- Arduino Joystick

Microcontrollers

The Arduino UNO is an 8-bit microcontroller board based on the ATmega328P (microchip). It has 14 digital input/ output pins, 6 analog inputs, and a 16 MHz ceramic resonator.

The Arduino MEGA 2560 Rev3 is an 8-bit microcontroller board based on the ATmega2560 (microchip). It has 54 digital input/ output pins, 16 analog inputs, 4 hardware serial ports (UARTs), and a 16 MHz crystal oscillator. The MEGA is compatible with the motor shield that is designed for the UNO.

Both microcontrollers possess a USB port and a power adapter port for either AC to DC power or battery supply.

The Adafruit Motor Shield v1 has 2 connections for 5V servo motors, connections for up to 4 bi-directional DC motors, and has 2 terminal blocks for linking up to 2 unipolar or bipolar stepper motors. It also has a 2-pin terminal block to connect external power and a configurable jumper.

Motors

The NEMA 17 is a 4-wire bipolar stepper motor with a 1.8° per step and 200 steps per revolution rating. It is known for executing steady torque and is compatible with Arduino motor shields. It has a 12V rating.

The DSSERVO DS3218 PRO 20 kg is a high torque digital servo motor. It has an adjustable servo arm that can be removed. It has a 4.8 - 6.8V rating.

DSD TECH HM-19 Bluetooth modules

The HM-19 Bluetooth 5.0 is the latest version of the Bluetooth wireless communication standard which is known for its higher data transfer rate and its 8-bit MTU (maximum transmission unit). It is comprising a 6-pin base board which leads to VCC, GND, TX, RX, STATE, and EN, making it convenient for Arduino projects. The manual is used to configure the modules; however, it lacks resourceful information in incorporating the modules within the Arduino code.

Obtaining hardware

After careful consideration and understanding of the concept design requirements, the hardware was ordered via Adafruit and Amazon. The Arduino MEGA 2560 Rev3, HM-19 Bluetooth modules, and one of the servo motors were obtained through NSCC faculty.

Independent component behavior with microcontroller

Stepper motor

The stepper motors' 4-wires are to be connected to one of the H-bridge blocks located on the motor shield which is attached to the UNO. The two H-bridge blocks available on the motor shield read M1 and M2 (first H-bridge) and M3 and M4 (second H-bridge). Due to the stepper motor being bipolar, wires are inserted and secured by screws (Figure 11). The wires are for power and message transmission.

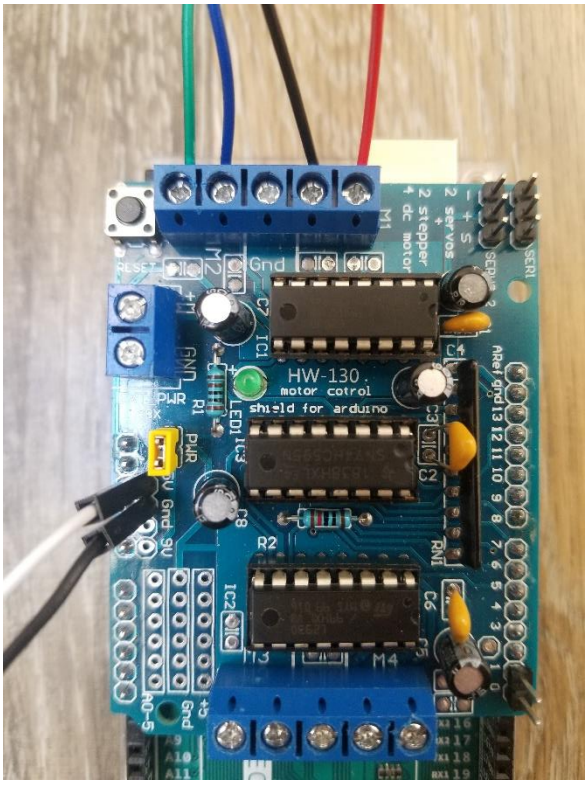

Figure 11 Arduino motor shield v1. Green, blue, black, and red wires indicate the H-bridge in which the stepper motor is connected. The two 3-pin units to the right of the H-bridge terminal are for servo motor connection.

Following the initial setup of wires, the USB cable is then connected to the USB port of the microcontroller board. The ~5V received from the USB connection suffices enough power to cause the stepper motor to make one 1.8° step.

Code from the Arduino IDE software platforms' library was used as a preliminary step in understanding how the stepper motor behaves via the microcontroller. The "AFMotor.h" library was used to communicate with the stepper motor. The code in Figure 12 actuates the stepper motor using four different modes.

```
#include <AFMotor.h>

// Number of steps per output rotation
// Change this as per your motor's specification
const int stepsPerRevolution = 200;
int motorSpeed = 30;

// connect motor to port #2 (M3 and M4)
AF_Stepper motor(stepsPerRevolution, 1);

void setup() {
  Serial.begin(9600);
  Serial.println("Stepper test!");

  motor.setSpeed(motorSpeed);
}

void loop() {

    //  Serial.println("Single coil steps");
    motor.step(1, FORWARD, SINGLE);
    motor.step(25, FORWARD, SINGLE);

    //  Serial.println("Double coil steps");
    motor.step(100, FORWARD, DOUBLE);
    motor.step(100, FORWARD, DOUBLE);

    //  Serial.println("Interleave coil steps");
    motor.step(100, FORWARD, INTERLEAVE);
    motor.step(100, FORWARD, INTERLEAVE);

    //  Serial.println("Micrsostep steps");
    motor.step(100, FORWARD, MICROSTEP);
    motor.step(100, FORWARD, MICROSTEP);
}
```

Figure 12 The code from "AFMotor.h" library was used to communicate with the stepper motor which actuates the motor using four different modes.

To actuate full revolutions, an external 12V power supply was inserted into the power adapter port of the microcontroller board.

With both the USB and external battery powering the board and stepper motor, it had not been realized that the 'power jumper' on the motor shield should have been removed (figure 11). The power jumper should be removed when supplying the board with 12V or more external power.

Servo motor

The servo motor's 3-pin connector is inserted directly on to the motor shield (S1 or S2). Either S1 or S2 can be used and just adjusted accordingly via the code. Like the stepper motor, a USB and/ or external power supply was used; however, the USB connection suffices enough power to cause the servo motor to initiate its 180° movement with adequate torque.

Code from the Arduino IDE software platforms' library was used as a preliminary step in understanding how the servo motor behaves via the microcontroller (figure 13). Arduino's "Servo.h" library was utilized to

communicate with the servo. The servo can be actuated by providing a position value to the servo variable from 0 to 360 degrees.

```
#include <Servo.h>
//
Servo myservo;  // create servo object to control a servo
// twelve servo objects can be created on most boards

int pos = 0;    // variable to store the servo position

void setup() {
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
}

void loop() {
  delay(15);

  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
    Serial.println(pos);
  }
  for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
}
```

Figure 13 Code from Arduino's "Servo.h" library that was utilized to communicate with the servo. The servo can be actuated by providing a position value to the servo variable from 0 to 360 degrees.

In having both the stepper and servo motor connected to the motor shield when supplying 5V USB and 12V external power, the microcontroller board, motor shield, and both motors were fried beyond repair. Therefore, hardware with identical specifications were then ordered.

System Component Integration

Bluetooth Integrated Motor Control

For the two motors to work by command, HM-19 Bluetooth modules (BTM) were implemented for wireless communication. In working with the BTM, it was realized that the motor shield prevented the BTM from being connected to the serial hardware ports 0 and 1. As such, an Arduino MEGA was utilized in order to provide additional serial hardware ports to connect the BTM.

The Arduino UNO was originally designated as the winch controller, intended to actuate specific motors based on external commands received from a second controller (remote controller). The motor shield however, obstructed ports 0 and 1 on the Arduino UNO - the only serial hardware ports available for serial communication. As such, the design was modified to utilize the Arduino MEGA as the winch controller as the additional serial ports made it possible to simultaneously connect the Bluetooth Module alongside the motor shield.

For the remote controller, the Arduino UNO was utilized as it has all the pins available that are required for the joystick and the BTM pin connectors.

Bluetooth Communication

The two HM-19 Bluetooth modules used to provide the system with communication functionality were configured as one module being the 'Master' module and the second being the 'Slave' module.

The 'Master' and 'Slave' modules are independent in that they can be used interchangeably between microcontrollers. The 'Master' module was designated to the winch controller, and the 'Slave' module was designated to the remote controller (figure 14).

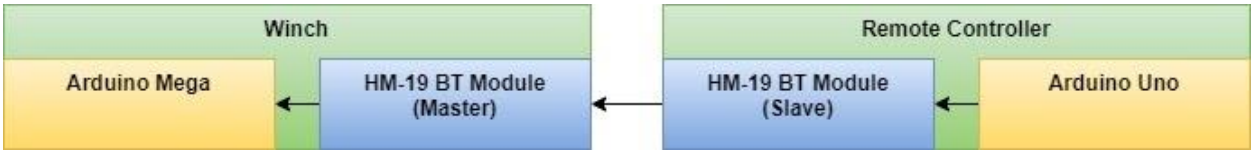

Figure 14 Schematic representation of the relationship between the winch (master) controller and the remote (slave) controller.

In configuring these Bluetooth modules, it is worth noting that the modules TX (transmission) pin is inserted into the TX port of the board and the RX (receiver) pin is inserted to the RX port of the board.

'Master' configuration

The Bluetooth module was connected to the Arduino UNO boards' serial ports 0 (TX) and 1 (RX). The serial monitor of the Arduino IDE platform is where the modules are configured. The parameters to be configured when using HM-19 Bluetooth modules are presented in table 1.

Table 1 HM-19 Bluetooth module parameters to be configured within the serial monitor of the Arduino IDE platform.

Parameter	HM-19 BT Module (Master)	HM-19 BT Module (Slave)
Baud	9600	9600
Role	1 - Central	0 - Peripheral
Work Type (IMME)	0 - Start pairing process immediately	1 - Wait until pairing request is received

'Slave' configuration

The Bluetooth module was connected to the Arduino MEGA boards' serial ports 18 (TX) and 19 (RX). The parameters to be configured when using HM-19 Bluetooth modules are presented in table 1.

Messaging Function

Bluetooth modules only send data as string values; therefore, in order to ensure the proper message was sent an encoding/decoding strategy was used.

By providing a messaging identifier, the winch controller can identify which sensor and/ or actuator the command is meant for.

The following (table 2) describes which identifiers correlate to which function:

Table 2 Message IDs for the Arduino joystick.

Message ID	Operation
A	Handle Joystick X Data
B	Handle Joystick Y Data
C	Actuate Servo

Additional message IDs can be implemented freely in this architecture to allow for additional capabilities.

Encoder

The Encoder function receives two parameters and prints a standard string to the serial port (figure 15).

```
void send_bluetooth_message(String msg_id, uint8_t value){
  //Packing the Serial message
  Serial.print("U*"); //A header
  Serial.print(msg_id);  //a token to indicate the message payload
  Serial.print(value);
  Serial.println("");
}
```

Figure 15 Encoder function

Decoder

The Decoder software function read the data from the Serial port and translated the string value into a specific operation (figure 16).

```
void select_operation(){
  //Figure out where your header is by getting the index of U*
  int offset = data.indexOf("U*"); //This is the header (0x552A)

  //Make sure the offset doesn't return -1 (which means it didn't find (U*) in
  //the message
  if (offset >= 0){

    //Extract the ID and Value from your message by getting everything
    // in the string between U* and the \n character

    //This is your Value (for example a joystick position Ex. 125)
    payload = data.substring(offset + 3, data.indexOf('\n'));

     //This is your ID - For example A, B, C, ..., Z
    char value = data.charAt (offset + 2);

    //Selects action based on message
    //Figure out what you want to do now that you have an action ID (for example - rotate winch) and a value with it
    switch (value){

      //Store Joystick x data
      case 'A':
        handle_joystick_x_data();
        break;
      //Store Joystick y Data
      case 'B':
        handle_joystick_y_data();
        break;
      //
      case 'C':
        set_servo_target_pos();
        break;
    //Add more case as the project grow
    }

    data = "0";
  }
}
```

Figure 16 Decoder software function

Joystick

Arduino's analog joystick was used to provide a Human-Machine Interface to the system. The joystick provides the following data shown in table 3.

Table 3 Data provided by joystick that can be read via the serial monitor of the Arduino IDE platform. Values change according to which direction the joystick button is pushed.

Data Type	Range	Midpoint	Description
Analog X Data	[0, 1023]	512	Motion of joystick's x-axis
Analog Y Data	[0, 1023]	512	Motion of joystick's y-axis
Digital Data	[0, 1]	Nil	Joystick Button

Reading Data

Analog Input (Joystick Input)

Data from the joystick's analog stick was read, processed, and encoded as a message over Bluetooth using the two functions below (figure 17):

```
/* Send joystick x data using a formatted message
 * Return: void
 * Input: nil
 */
void send_joystick_x_data(){
  // reading from the sensors
  x_value = map(analogRead(js_x_pin),0 ,1023, 0, 255);

  //Check to see if there was a change in x_value
  if((x_value <= (old_x_value + 1)) && (x_value >= (old_x_value - 1))){
    return;
  }

  // Convert the x_value to a string so it can be passed as a parameter
  // to the function
  send_bluetooth_message("A", x_value);

  //Store the value to check against in next function call
  old_x_value = x_value;

  return;

}
```

```
/* Send joystick y data using a formatted message
 * Return: void
 * Input: nil
 */
void send_joystick_y_data(){

  y_value = map(analogRead(js_y_pin),0 ,1023, 0, 255);

  //Check to see if there was a change in y_value
  if((y_value <= (old_y_value + 1)) && (y_value >= (old_y_value - 1))){
    return;
  }

  // Convert the x_value to a string so it can be passed as a parameter
  // to the function
  send_bluetooth_message("B", y_value);

  //Store the value to check against in next function call
  old_y_value = y_value;

  return;

}
```

Figure 17 Joystick analog input code.

Digital Input (Push Button)

In order to provide a level of parallelism for the remote controller an Interrupt Service Routine (ISR) was used to handle the joystick button press. This allowed the Arduino Uno to immediately deal with a button press rather than wait for the code execution to check the button state.

Interrupt Service Routine Setup

The ISR required three steps to setup:

1. Calling Arduino's "attachInterrupt" function - enabling the interrupt on a specific pin (figure 16).
2. Providing a function to call when the interrupt is triggered - this is the js_interrupt_handler.
3. Providing a value for when the interrupt should be triggered - FALLING caused the interrupt to be triggered when the button switched from a logic high (not pressed) to a logic low (pressed).

```
void setup() {
  // Init serial where data will be passed to - this is
  // the port on the Uno (serial port 0 and 1) which is
  // connected to the bluetooth module
  Serial.begin(9600);
  delay(500); // Half second delay - so you don't overwhelm the micro processor

  //Init built in LED - DEBUG
  pinMode(LED_BUILTIN, OUTPUT);

  // -- NOTES --
  //  Serial.write("AT+CONA4DA32677BE0");


  //Setup Arduino Uno pin to receive joystick input [Interrupt replaces this]
  //  pinMode(js_push_btn, INPUT);

  digitalWrite(js_intterupt_pin, HIGH);

  // Setup the interrupt for the joystick button
  // parameters
  //  - js_interrupt_pin: the connected pin
  //  - js_interrupt_handler: the function to run when the interrupt is triggered
  //  - FALLING: how the interrupt will be triggered (see interrupt docs)
  attachInterrupt(digitalPinToInterrupt(js_intterupt_pin), js_interrupt_handler, FALLING);

  Serial.println("Setup Complete");
}
```

Figure 18 Interrupt Service Routine Arduino code.

Debouncing

One issue with the Arduino joystick is "Switch Bouncing". Below visualizes what happens as a switch is pressed (figure 19). When this happens, a single button press can trigger a function multiple times causing undesired behavior in the system.

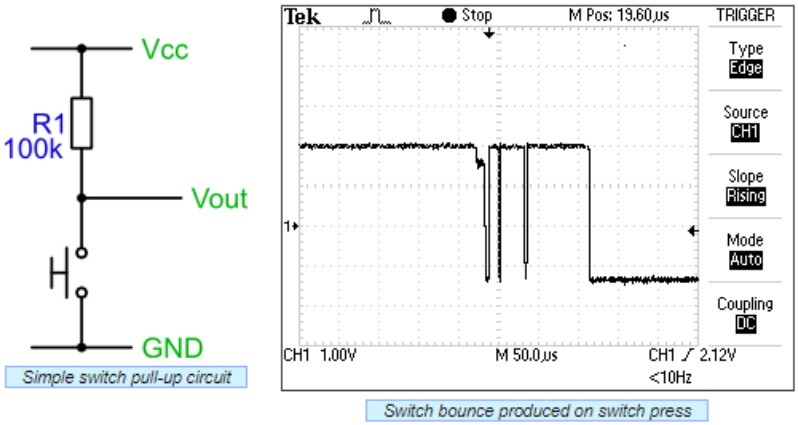

Figure 19 Switch bouncing.

In order to avoid this, Debouncing was used in the function that handled the interrupt. The function ensured that an adequate amount of time elapsed before a new message was sent.

```
/* Joystick Interrupt Handler
 * Input: Nil
 * Output: Nil
 * Desc: Handles the intterupt for the joystick button press.
 */
void js_interrupt_handler(){
  static unsigned long last_interrupt_time = 0;
  unsigned long interrupt_time = millis();

  // If interrupts come faster than 300ms, assume it's a bounce and ignore
  if (interrupt_time - last_interrupt_time > 300){
    send_bluetooth_message("C", 0);
  }

  last_interrupt_time = interrupt_time;
}
```

Figure 20 Joystick Debouncing Interrupt Code

2.5 Discussion

The initial goal of delivering a product based on the parameters provided by *InNatureRobotics* was unable to be met. Handling a 5kg payload with an ultra-lightweight material such as the 3D printed plastics available at the school weighing 1kg was deemed to be impractical. With preliminary testing, including Prototype 2, it was concluded that a system weighing a total of approximately one kilogram would be insufficiently strong enough to handle the payload required using the resources available. Prototype 2 was engineered by modeling the structure on Autodesk Inventor and then 3D printed using plastics. Even in the stable open environment of the OTAD shop setting (in contrast to the marine environment of the deck of an ASV), the system showed mechanical weakness. Specifically, the extended arm of the winch bent under the weight of the test payload.

The aluminum structural design was chosen over the available 3D printable plastics for two main purposes: the aluminum is significantly stronger material while still being relatively lightweight, and is more environmentally durable. The former is a very important consideration for any ocean technology given the harshness of the marine environment. Aluminum does corrode but it does not rust and is often used as a material for marine settings. While the plastic neither corrodes nor rusts, it is a semipermeable material which will breakdown structurally faster than the solid metallic aluminum material.

The mechanical and electronics phases were near completion individually at the time when social distancing orders were issued and the school was closed. The group was unable to come together to assemble the final structure nor work in the OTAD shop. One group member took the initiative of doing their best to assemble the structure at home using the resources available to them. That said, the full potential of the final design was not fully realized. Enough of an assembly was conducted, however, to demonstrate that the mechanical structure was sufficiently strong for the target test payload weight of 1kg. More testing would have been done in the lab to give the system a maximum payload weight rating.

The electronics were successfully delivered. The goal was to have two physically separate Arduino microcontrollers: one master controller sending wireless Bluetooth commands to the slave controller. The master controller receives and sends signal inputs by a small joystick in a similar fashion to other winch systems. The slave controller is connected to the winch motors and controls, based on joystick input, the rotation of the system (servo motor) and deployment of the cable (stepper motor). The system was fully realized in this regard.

The final product delivered with this project could be considered a prototype that has the potential, with future iterations, to become a marketable final product. A formalized design with additional features such as an encoder to count the length of cable deployed or retrieved, and an electromechanical cable to give power to sensors and/or receive electrical signals, would increase its market value. As discussed in the introduction, ASVs are an increasing market in oceanography as a cost-effective way to obtain physical measurements, and a lightweight automated winch system for deploying sensors would presumably be valuable in this space.

2.6 Technical Considerations

2.6.1 Future improvements

Mechanical

- improve the system's watertight integrity
- machined drum vs. fishing reel
- bearings to reduce the load on the servo motor

Hardware

- improve wire management – shorten/zip tie/fasten
- improved insulation of electrical components
- optical or rotary encoder for counting cable length
- use a more stable power supply

Software

- Stepper motor holds back code execution - maybe use an ISR to side step that.
- Messaging system could be implemented on the winch controller to provide status and feedback to the remote controller.
- Messaging system could be expanded upon to include any additional sensor/actuator on board the AMOS.

Modelling

- finite element analysis for stress calculation – allows more rapid prototype testing at a reduced cost.
- develop an electromechanical prototype – using electromechanical cable and slip ring
- finite element analysis for stress calculation – allows more rapid prototype testing at a reduced cost.

2.6.2 Lessons learned

- Power jumper needs to be removed on the motor shield (attached to the Arduino board) when powered using a 12V power supply – reduces risk of damaging boards/motors.
- Pin connectors need to be inverted when configuring the Bluetooth modules.
- 9V battery does not provide enough power to the system to enable simultaneous functioning of stepper and servo motor control and actuation as well as Bluetooth communication.
- Spool constraints prevented proper cable deployment given it's manufactured shape.
- Servo motor arm provided an uneven weight distribution even with the mitigation of the circular platform.
- Adding aluminum support pieces to 3D printed superstructure compromise it's structural integrity given the amount of mounting points required.
- Modeling the whole structure in a 3D modeling software and analyzing the load carrying capacity by assigning different materials would have given a better insight of the load carrying capacity of the material to be used for the structure. It would have depicted a better understanding of the structural integrity using different fasteners and brackets. This would have helped the team members to change the design principles in the early stage of the project rather than making prototypes.
- Adding a diagonal member in-between the arms would have been better idea rather than the use of brackets which didn't provide enough strength at the weak points.

3 Conclusion

This report outlines the activities involved in the Aquatic Mini Observation System Winch capstone project.

The main purpose of the oceans technology capstone project course is to gain experience and skills in the processes involved in delivering some sort of field-related technology. The value in the course is in the learning process itself. This project was a chance to apply concepts learned in other courses in the program to a real-world example. Beyond applying skills acquired in the program, it was also a chance to learn some other concepts such as microcontrollers, programming, and small electronic components such as the motors.

The group is happy with the outcome given the circumstances related to the school closing. We would have enjoyed working together using the resources available at the school to finish the final assembly of the system and conduct a thorough testing phase, but feel we reached sufficient milestones to demonstrate a proof of concept.

The final assembly is something we consider to be a prototype for which future iterations could deliver a product with real market value in the field of ocean robotic systems (ASVs and perhaps other applications). Though we were not able to reach that final design given limited resources, it is something we can envision, and something we feel capable of being able to achieve having brought the project this far.

4 References

Automation and a Changing Economy: The Case for Action. (2019, April 2). Retrieved April 20, 2020, from https://www.aspeninstitute.org/publications/automation-and-a-changing-economy-the-case-for-action/

Autonomous Surface Vehicles. (2020). Retrieved April 20, 2020, from https://www.noc.ac.uk/facilities/marine-autonomous-robotic-systems/asv

Appendix A

4.1 Winch Controller Arduino Code

```
//#include <SoftwareSerial.h>
//SoftwareSerial HM19(2, 3);

//SERVO
#include <Servo.h>

Servo myservo;  // create servo object to control a servo
int pos = 0;    // variable to store the servo position

//SERIAL COMMUNICATION
String data; //Store each line of Serial communication
String pot1String = "0", pot2String = "0";
float pot1, pot2;

void setup() {
  // put your setup code here, to run once:
  //ARDUINO SERIAL
  Serial.begin(9600);
  delay(500);

  //SERVO PIN
  myservo.attach(10);  // attaches the servo on pin 10 to the servo object

  //BLUETOOTH SERIAL
  Serial1.begin(9600);
  delay(500);

//  pinMode(19, INPUT);
//  digitalWrite(19, HIGH);
//  delay(500);

  pinMode(LED_BUILTIN, OUTPUT);

  Serial1.write("AT+CONA4DA32677B8C");

  Serial.println("Device Pairing Complete");
  //read first package
  if (Serial1.available() > 0)
  {
    data = Serial1.readStringUntil('\n');
  }
  //Disregard the first reading
  data = "0";
}

String str;

void rotate_winch_clockwise(){
  // clockwise sweeping motion
  // goes from 0 degrees to 180 degrees
  for (pos = 0; pos <= 180; pos += 1) {

    // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
```

```
    Serial.println(pos);
  }
}
void rotate_winch_counterclockwise(){
  // counterclockwise sweeping motion
  // goes from 180 degrees to 0 degrees
  for (pos = 180; pos >= 0; pos -= 1) {
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
    Serial.println(pos);
  }
}

// 0 for cw
// 1 for ccw
int next_rotation = 0;
void loop() {

  // put your main code here, to run repeatedly:
  //Check if there's data on the Bluetooth pipeline (data has been sent by the master
   if (Serial1.available() > 0)
    {
      //readStringUntil end (reads the string until it sees a newline character (\n)
      //That will store (U*AValue) (Header*IDValue)
      data = Serial1.readStringUntil('\n');
    }
    //Unpacks your message and picks the correct function
    //Initialize an empty variable called payload
    String payload = "";
    //If we successfully read the data then unpack the message
    if (data != "0")
    {
      //Figure out where your header is by getting the index of U*
      int offset = data.indexOf("U*"); //This is the header (0x552A)
      //Make sure the offset doesn't return -1 (which means it didn't find (U*) in
      //the message
      if (offset >= 0)
      {
        //Extract the ID and Value from your message by getting everything
        // in the string between U* and the \n character
        payload = data.substring(offset + 3, data.indexOf('\n'));  //This is your Value (for example a joystick position Ex. 125)
        char value = data.charAt (offset + 2); //This is your ID - For example A or B or C
        //Selects action based on message
        //Figure out what you want to do now that you have an action ID (for example - rotate winch) and a value with it
        switch (value)
        {
          case 'A':
            pot1String = payload;
            break;
          case 'B':
            pot2String = payload;
            break;
          case 'C':
            if(next_rotation == 0){
              //rotate_winch_clockwise();
```

```
                Serial.println("Rotating Motor Clockwise");
                next_rotation = 1;
              }
              break;
            case 'D':
              if(next_rotation == 1){
                //rotate_winch_counterclockwise();
                Serial.println("Rotating Motor Counter Clockwise");
                next_rotation = 0;
              }
              break;
          //Add more case as the project grow
          }
        }
      }
      //convert strings to float
      pot1 = pot1String.toFloat();
      pot2 = pot2String.toFloat();
      Serial.print("X-Data: ");
      Serial.println(pot1);

      Serial.print("Y-Data: ");
      Serial.println(pot2);
      delay(500);
      //joystick_y - 255 - motor speed input
      // 255 is the joystick's max
      // which means you want motor max speed


}


//  if (Serial1.available()) {
//    int inByte = Serial1.read();
//
//    // Needs to be write instead of println -
//    // println only prints numbers for some reason
//    Serial.write(inByte);
//
//    if(inByte == '0'){
//      digitalWrite(LED_BUILTIN, LOW);
//    }
//    if(inByte == '1'){
//      digitalWrite(LED_BUILTIN, HIGH);
//    }
//  }

//  // read from port 0, send to port 1:
//  if (Serial.available()) {
//    int inByte = Serial.read();
//    Serial1.write(inByte);
//  }

//}
```

4.2 Remote Controller Arduino Code

```
/*
 * Arduino Uno - Remote Controller
 *
 * Bluetooth HM-19 - Serial Monitor Direct
 * Tx -> Tx
 * Rx -> Rx
 *
 * Bluetooth HM-19 - Sketch Upload
 * Tx -> Disconnected
 * Rx -> Disconnected
 *
 * Bluetooth HM-19 - Operational
 * Tx -> Rx
 * Rx -> Tx
 *
 * Bluetooh HM-19 Info
 * Bluetooth Module MAC: A4DA32677B8C
 * Bluetooth Module Role: SLAVES
 * Bluetooth Baud: 9600
 * Bluetooth Mode: 1
 *
 * ---------------------------------
 * Joystick
 * P2 - Push Button
 * A0 - x-axis
 * A1 - y-axis
 * ---------------------------------
 *
 */

// Setup Arduino pin variables
//const int js_push_btn = 2; // digital pin connected to switch output [replaced by js_interrupt_pin]
const int js_x_pin = 0; // analog pin connected to X output - A0
const int js_y_pin = 1; // analog pin connected to Y output - A1

// Interrupt Pin
const byte js_intterupt_pin = 2;

// Joystick related variables - used to capture joystick input
// and reduce overall noise in the readings
uint8_t old_y_value = 0;
uint8_t old_x_value = 0;
uint8_t y_value = 0;
uint8_t x_value = 0;

// No longer needed - logic replaced by interrupt mechanics
// winch_state 0 - first position (start)
// winch_state 1 - second position (deploy)
//int winch_state = 0;

void setup() {
  // Init serial where data will be passed to - this is
  // the port on the Uno (serial port 0 and 1) which is
  // connected to the bluetooth module
  Serial.begin(9600);
  delay(500); // Half second delay - so you don't overwhelm the micro processor
```

```
  //Init built in LED - DEBUG
  pinMode(LED_BUILTIN, OUTPUT);

  // -- NOTES --
  //  Serial.write("AT+CONA4DA32677BE0");


  //Setup Arduino Uno pin to receive joystick input [Interrupt replaces this]
  //  pinMode(js_push_btn, INPUT);

  digitalWrite(js_intterupt_pin, HIGH);

  // Setup the interrupt for the joystick button
  // parameters
  //  - js_interrupt_pin: the connected pin
  //  - js_interrupt_handler: the function to run when the interrupt is triggered
  //  - FALLING: how the interrupt will be triggered (see interrupt docs)
  attachInterrupt(digitalPinToInterrupt(js_intterupt_pin), js_interrupt_handler, FALLING);

  Serial.println("Setup Complete");
}

/* Joystick Interrupt Handler
 * Input: Nil
 * Output: Nil
 * Desc: Handles the intterupt for the joystick button press.
 */
void js_interrupt_handler(){
  static unsigned long last_interrupt_time = 0;
  unsigned long interrupt_time = millis();

  // If interrupts come faster than 300ms, assume it's a bounce and ignore
  if (interrupt_time - last_interrupt_time > 300){
    send_bluetooth_message("C", 0);
  }

  last_interrupt_time = interrupt_time;
}

void send_bluetooth_message(String msg_id, uint8_t value){
  //Packing the Serial message
  Serial.print("U*"); //A header
  Serial.print(msg_id);  //a token to indicate the message payload
  Serial.print(value);
  Serial.println("");
}


/* Send joystick x data using a formatted message
 * Return: void
 * Input: nil
 */
void send_joystick_x_data(){
  // reading from the sensors
  x_value = map(analogRead(js_x_pin),0 ,1023, 0, 255);
```

```
113
114   //Check to see if there was a change in x_value
115   if((x_value <= (old_x_value + 1)) && (x_value >= (old_x_value - 1))){
116     return;
117   }
118
119   // Convert the x_value to a string so it can be passed as a parameter
120   // to the function
121   send_bluetooth_message("A", x_value);
122
123   //Store the value to check against in next function call
124   old_x_value = x_value;
125
126   return;
127
128 }
129
130 /* Send joystick y data using a formatted message
131  * Return: void
132  * Input: nil
133  */
134 void send_joystick_y_data(){
135
136   y_value = map(analogRead(js_y_pin),0 ,1023, 0, 255);
137
138   //Check to see if there was a change in y_value
139   if((y_value <= (old_y_value + 1)) && (y_value >= (old_y_value - 1))){
140     return;
141   }
142
143   // Convert the x_value to a string so it can be passed as a parameter
144   // to the function
145   send_bluetooth_message("B", y_value);
146
147   //Store the value to check against in next function call
148   old_y_value = y_value;
149
150   return;
151
152 }
153
154 void loop() {
155
156   // Send Commands/Data
157   send_joystick_x_data();
158   send_joystick_y_data();
159
160 }
```