Engineering a Profiling System for a Robotic Oceanographic Surface Sampler

by
Robert J. Shannon

A THESIS

submitted to

Oregon State University

Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Electrical and Computer Engineering
(Honors Scholar)

Presented June 9, 2017
Commencement June 2017

AN ABSTRACT OF THE THESIS OF

Robert J. Shannon for the degree of Honors Baccalaureate of Science in Electrical and Computer Engineering presented on June 9, 2017. Title: Engineering a Profiling System for a Robotic Oceanographic Surface Sampler.

Abstract approved:____

Jonathan Nash

ROSS (Robotic Oceanographic Surface Sampler) is an autonomous boat, equipped with oceanographic sensors, used for studying the ocean's surface. One of its advantages over traditional research vessels is its ability to sample near glaciers, an area too dangerous for manned ships. In order to properly ascertain deep ocean heat's effect on glacial melt, it is necessary to take profiles to catalogue the water's various characteristics from the surface to the bottom of the glacier. My challenge was to design and construct an electronic solution to enable remote sampling.

Key Words: Autonomous, CTD, Microcontroller, Profiling, Robotic

Corresponding e-mail address: rj_shannon@comcast.net

Engineering a Profiling System for a Robotic Oceanographic Surface Sampler

by
Robert J. Shannon

A THESIS

submitted to

Oregon State University

Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Electrical and Computer Engineering
(Honors Scholar)

Presented June 9, 2017
Commencement June 2017

Honors Baccalaureate of Science in Electrical and Computer Engineering project of Robert J. Shannon presented on June 9, 2017.

APPROVED:

____

Jonathan Nash, Mentor, representing Earth, Ocean, and Atmospheric Sciences

____

Jonathan Hurst, Committee Member, representing Mechanical Engineering

____

Matthew Shuman, Committee Member, representing Electrical and Computer Engineering

____

Toni Doolen, Dean, Oregon State University Honors College

I understand that my project will become part of the permanent collection of Oregon State University, Honors College. My signature below authorizes release of my project to any reader upon request.

____

Robert J. Shannon, Author

Acknowledgements

I would like to thank my mentor, Dr. Jonathan Nash, for his guidance, feedback, and support and for welcoming me into his team. This opportunity and its responsibilities helped me grow as an engineer, a scholar, and a person.

I would also like to thank thesis committee members Dr. Jonathan Hurst and Matthew Shuman for investing their time and expressing genuine interest in my research.

I want to thank my teammates Nick McComb, June Marion, Jasmine Nahorniak, and Brendan (Onn Lim) Yong along with everyone else I collaborated with during my time working on ROSS for cultivating a welcoming and fulfilling work environment. It was a pleasure working with you.

Additionally, I would like to thank Nick McComb for reaching out and involving me with the project, Dr. Rebecca Jackson for teaching me the science behind our research, and Cam Mullins for creating Figures One and Four of this report. Lastly, I thank my parents, Michael and Nancy Shannon, for fostering my love of learning and passion for engineering.

Thank you all for involving yourselves in my undergraduate experience. You have made it unforgettable.

Table of Contents

1. Introduction

1.1 The Goal

How can one predict a glacier's melting rate given the surrounding water's properties and the characteristics of its subglacial discharge? Glacial melting is a function of the quantity of fresh water entering the ocean at the base of the glacier coupled with the surrounding water's attributes. Fresh water streams into the ocean via channels within the glacier. These streams conjoin and create a plume leading away from the glacier's terminus (glacier’s leading edge). The fresh water mixes with surrounding salt water, further driving the melting process. The extent to which the water mixes influences the melting rate. Increased mixing brings the warmer salt water into contact with the glacier, while less mixing results in an insulating layer of fresh water. This process primarily occurs deep below the ocean’s surface, thus its study requires a system capable of measuring far below the research platform.

1.2 Why ROSS?

Professor Jonathan Nash, Ph.D. in Physical Oceanography, along with his team including June Marion, Nick McComb, Jasmine Nahorniak, Brendan (Onn Lim) Yong, and myself, is developing robotic oceanographic surface samplers. These robots, or ROSSs, are autonomous jet powered kayaks equipped with oceanographic sensors. ROSS provides three primary advantages over traditional research vessels. First, ROSS can collect data in parallel with multiple research vessels and other ROSS platforms for a fraction of the cost of a fully manned ship. Second, ROSS, unlike large research vessels, minimally disturbs the ocean’s surface. This allows scientists to “obtain uncontaminated observations of the upper few meters of the ocean, a region that is challenging to sample [1]”. Finally, ROSS can operate in environments too dangerous to send manned vessels, such as a glacier face.

One of Dr. Nash’s fields of study is glacial melting and how its rate is affected by deep ocean heat. In August of 2016, the team traveled to Petersburg, Alaska to take data at the LeConte glacier. Manned boats cannot go within half a mile of the terminus, thus ROSS’s third advantage comes into play. To prepare for this and future deployments, I designed and implemented the electrical and electronic systems as well as wrote software and firmware with the goal of creating a profiling system capable of taking measurements at the bottom of the glacier face.

2. System Design

2.1 System Setup

The profiling system, diagrammed in Fig. 1, consists of an electronically powered winch equipped with a rotary encoder to measure sensor depth in revolutions of line. This line is fed through the A-frame, a component which serves as a safety measure and method for detecting when the oceanographic instrument returns to the surface. It provides stability and protection when the instrument is out of the water and guides the instrument back in when beginning a profile. The instrument is weighted with approximately 10 lbs. of lead to increase its speed of descent.

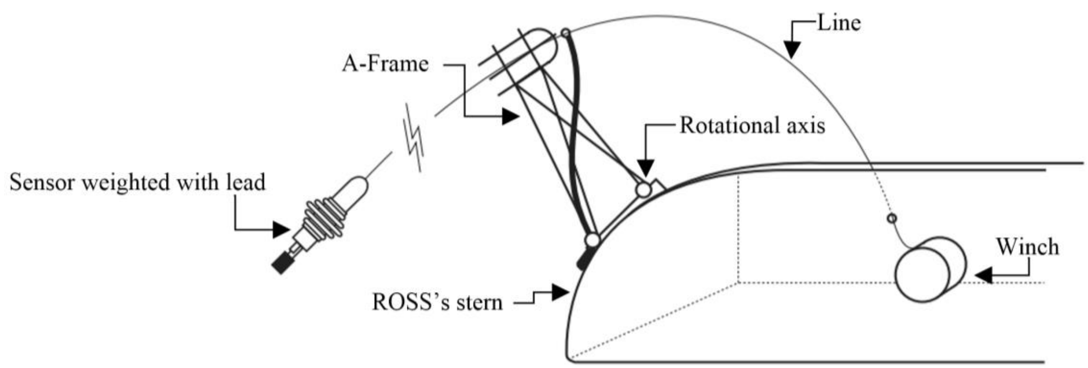

Fig. 1 – ROSS's profiling system monitors instrument position via sensors on the A-frame and Winch [5].

2.2 Design Goals

Given this setup, I designed a system capable of acting upon user supplied profiling commands, providing a method to control a profile's depth and speed in both directions. Under normal conditions, the winch begins a profile by slowly lowering the instrument into the water, accelerates to the user defined "out speed", stops at the desired number of revolutions (up to 65,535), and returns at the user defined "in speed" until it reaches a hard coded stopping distance. At this point, the winch slows to a software defined "slowing speed", preventing instrument damage. Once the instrument pulls the A-frame upright, its position is maintained and the data downloaded to ROSS's onboard computer.

It was crucial the manner in which this behavior was implemented could easily adapt to atypical commands. For example, in the case of an emergency, the system needs to interrupt its current process and respond to a stop command. The system would return to normal operation upon receiving new profile parameters.

Additionally, the code needed to be written such that it could easily be expanded upon.

3. The Technology

3.1 Electronic Speed Control

The driving force behind ROSS's profiling system is an electric Shimano Tiagra 130 fishing reel (winch). Its brushless motor is driven by an electronic speed control (ESC) which inverts DC power from two series 12V deep cycle batteries (24V) to three-phase AC power based on a pulse width modulation (PWM) signal from a microcontroller.

An inverter uses PWM to emulate lower voltages by varying the output voltage's duty cycle. Sinusoidally increasing and decreasing the duty cycle achieves a stepped sinusoid that appears as AC to a motor, as seen in Fig. 2. Manipulating the ESC's three sinusoidal outputs allows the user to control a motor's speed and direction.

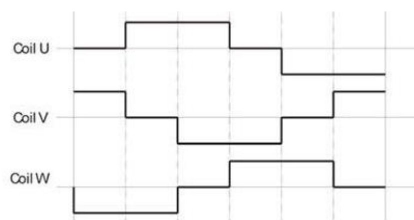

Fig. 2 – An ESC provides control over the winch's speed and direction by varying its output based on input received from a microcontroller [4].

3.2 Encoder

The winch is augmented with a Model 15H Accu-Coder quadrature encoder to determine how many revolutions of line have been let out. This type of encoder rotates along a code disk marked with the pattern shown in Fig. 3. The two bands of alternating black and white stripes are 90 degrees out of phase from one another, thus the sensor can derive direction based on which pattern's signal is leading. The number of times the encoder pulses can be converted into winch revolutions relative to the point at which the count was set to zero.

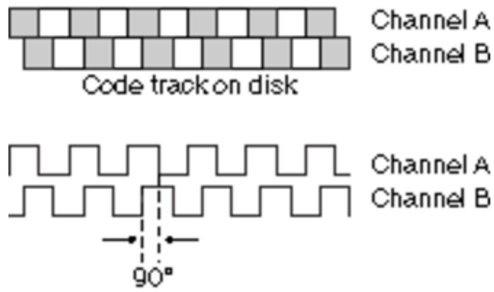

Fig. 3- The encoder's rotational motion generates a waveform from which the microcontroller can determine instrument position in revolutions [3].

While the encoder can serve as the stopping condition when lowering the sensor, it is unreliable when returning the sensor to the surface. The line may stretch under tension, requiring additional revolutions. For this reason I designed an independent system incorporating the Hall Effect as the primary stopping condition as discussed in section 4.1.

3.3 Oceanographic Instrument

While it can be adapted for other sensors, ROSS's profiling system is intended for use with RBR's Concerto CTD. This instrument measures the water's conductivity, temperature, and depth, providing a profile of characteristics necessary to determine the extent to which fresh and salt water mix along the glacier's face.

4. My Contributions

My contribution to the winch profiling system is comprised of four subsections: A-frame position detection, Python command script, winch control microcontroller program, and wireless sensor data transfer script. Together, these components provide the user full control over profile characteristics, such as line out speed, line in speed, and number of revolutions.

4.1 A-frame Detection

Magnets are positioned on the A-frame's rotational axis such that one of two Hall Effect sensors is activated when the frame is in the up or down position as seen in Fig. 4. These sensors are solid state and detect magnetic fields in their proximity. This means they will not produce a false positive in turbulent environments.

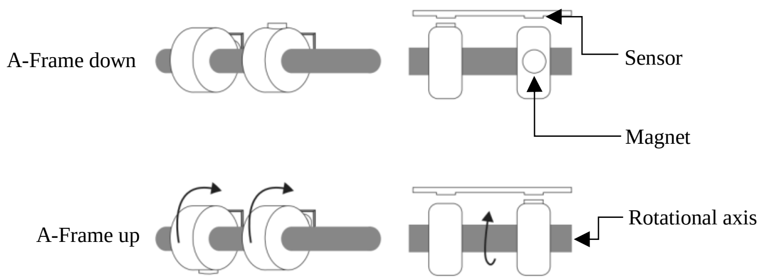

Fig. 4 – Hall Effect sensors detect A-frame position and serve as the primary inbound stopping condition [6].

I designed a PCB, seen in Fig. 5, to mount the sensor integrated circuit, as well as the necessary pull up resistors and decoupling capacitors. Their outputs are routed to the microcontroller and serve as the winch's primary inbound stopping condition.

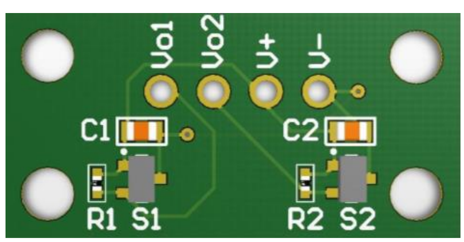

Fig. 5 – This PCB serves as the interface between the microcontroller and Hall Effect sensors.

4.2 Command Script

The Python script accepts four parameters at runtime: speed out, speed in, depth, and emergency stop. If the stop value is zero, the code then reads in the remaining arguments. Both speeds are represented as single byte where 0 indicates no movement and 254 represents maximum speed in the corresponding direction (255 is reserved). The depth is represented with two bytes and thus ranges between 0 and 65,535 revolutions. In its May 2017 deployment, ROSS's profiles averaged 400 revolutions (approximately 100 m).

The values are saved into an array and a checksum is derived by performing the "exclusive or" operation on all five bytes. A serial port is then opened between the computer and microcontroller and the character representing each byte is transmitted.

Emergency profile stops are represented using one of five integers. Three of these values interrupt the current profile and dictate the winch's behavior after stopping. These are stop and hold position, return at half speed, and return at full speed. The remaining two are used to initiate a motor start or stop, allowing the user to start and kill the boat's engine remotely. Fail-safes within the microcontroller code ensure the engine cannot start unintentionally and prevent motor damage due to accidental commands to start while already running.

To promote usability, script execution is abstracted using a graphical user interface (GUI), designed by Oregon State University senior faculty research assistant Jasmine Nahorniak, which allows the user to enter speed values as a percentage. The GUI can also initiate an emergency profile interruption and remotely activate or deactivate ROSS's engine by automatically sending the corresponding packet at the click of a button.

4.3 Microcontroller Program

I selected the Teensy 3.2 for the system's microcontroller because it provides several advantages over many alternatives. First, using the Teensyduino add-on, the microcontroller can be programmed in Arduino. This abstracted C based language provides access to a plethora of libraries, most notably, the Encoder Library which has been optimized for interfacing between a microcontroller and quadrature encoder. The Teensy's interrupt capable input pins allow the microcontroller to attain peak performance when reading encoder output.

Arduino code consists of two primary functions from which all additional functions are called: setup, which runs once when the microcontroller is first powered on, and loop, which executes repeatedly until the microcontroller loses power. I designed my loop function to behave as a three-state finite state machine as seen in Fig. 6. Each time loop executes, it runs the portion of code associated with the current state which includes the logic for determining the subsequent state.

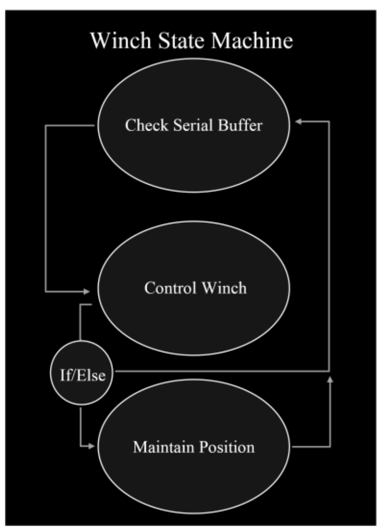

Fig. 6 - The microcontroller code continually iterates through three states, each of which is responsible for a different task related to winch control.

The initial state checks the serial buffer to see if a new packet has been fully received and, if so, updates the profile control variables. The following state interprets the stop byte from the most recent command packet to determine what kind of stop, if any, needs to be initiated or continued. It then manipulates the control logic variables and adjusts the speed as necessary.

Speed changes are handled by my “changeSpeed” function (Appendix B: lines 178 through 248) which abstracts the process of sinusoidally accelerating the winch from its current to the desired speed in the intended direction. Acceleration time can be set in the microcontroller code corresponding to line 56 of Appendix B. This acceleration pattern reduces motor wear, resulting in a more robust system.

Lastly, the FSM will either return to the initial state or first enter the "Maintain" state. This last state is used once the sensor has returned to the surface and the A-frame is in the upright position. In its current iteration, the profiling winch relies on electromagnetic braking to hold its position. This does not always provide enough torque to overcome the rotational force exerted by the A-frame. This state utilizes the Hall Effect sensors to detect if the A-frame sags and returns it to the upright position. This state’s necessity will decrease in future implementations supplemented by a mechanical winch brake.

In between each state, the microcontroller checks to see if it has been operating for more than one second since the last time it sent a status message. In such cases the microcontroller transmits the following via serial:

- Whether or not it has received a corrupted command packet as determined by comparing the calculated checksum to the transmitted one
- Whether or not the system is ready to begin a new profile
- The sensor's current direction of travel (up, down, or stationary)
- The sensor's distance from its starting position in revolutions

4.4 Wireless Data Download

Wirelessly downloading the instrument's data provides two major benefits: data backup and sensor protection. While the sensor itself is costly, even more valuable is the data stored on the device. If the line were to snap or the waterproof enclosure became compromised, all data from that deployment would be lost. Wirelessly downloading the data after each profile minimizes this risk.

The alternative to wireless downloading involves opening the waterproof casing and connecting to a laptop. This introduces opportunities for salt water to enter the sensor or the O-ring to become damaged.

The sensor's manufacturer, RBR, only supports wireless downloading in the form of a GUI on computers running a Windows operating system. ROSS's onboard computer runs Linux and thus requires a program that can be executed from the command line. RBR provided two C programs, one of which downloads the binary file storing the sensor's data. The second parses binary data into human readable characters when executed with the proper parameters.

After collaborating with RBR's software engineers, I was able to determine the conditions necessary for each program to operate. I constructed a testing apparatus to simulate a 3 meter profile (necessary for the sensor's Wi-Fi module to transmit) and wrote a BASH script to orchestrate the data download, parsing, and storage process.

The C program downloads all data stored on the sensor past a given memory offset. In order to isolate individual profiles and expedite the download, my BASH script calculates this offset based on the size of all previously downloaded data, saves the profile's binary data in a timestamped file, and appends the new data to a file containing the combined data from each of the deployment's profiles.

My script then passes the new binary file to the parsing program so the data can be read. This requires the user to run my script with the number of sensor data channels as the argument. In ROSS deployments, six data channels are utilized. These are the measured channels: conductivity, temperature, and pressure, and the derived channels: salinity, depth, and speed of sound.

5. Conclusion

Ultimately, I was successful in engineering a system capable of exhibiting desirable profiling behavior given user input dictating out speed, in speed, and depth. Its flexible design allowed for the addition and execution of non-profiling commands. This behavior was demonstrated in field tests conducted in the Willamette River and Yaquina Bay.

Although it is not yet implemented into ROSS's standard operation, I verified the data download and parsing capabilities using a testing apparatus I designed to imitate a 3 meter profile.

Despite these successes, the profiling system's initial iteration was unreliable. The ESC (Mamba Max Pro) was not rated to draw the current necessary to repeatedly lift the weighted CTD and overcome the force exerted by the A-Frame. In fact, the lack of airflow within the watertight housing further reduced its current handling capabilities.

In order to quantify the ESC's limitations, I conducted a series of tests in which I determined the maximum weight the winch could lift at various power levels. Even at 100 percent power, the ESC could not reliably lift 17.5 lbs., thus, it proved unable to meet our requirements.

I conducted the same tests with a replacement ESC (Mamba XL X) our supplier provided. The results indicated the new ESC could lift up to 40 lbs. at full power, suggesting it to be a viable replacement. ROSS's successful Alaska deployment in May 2017, using this ESC, corroborates this assertion.

Another shortcoming was the system's tendency to let line out too fast, causing it to tangle. This issue required the initial configuration be revised to meet mission requirements. ROSS's current iteration utilizes an augmented version of my profiling system which uses PID to alter the winch's speed in order to maintain line tension, preventing tangling. My original code was expanded upon to accommodate these changes, demonstrating its expandability. According to Jorian Bruslind, the engineer in charge of implementing PID control, "…it was good that there was a premade function that would take in an integer value and then ramp up the speed of the winch to said integer value…all we had to do was have the PID constantly monitor the resistance and then send the required speed to the ['changeSpeed'] function" [2]. Additionally, troubleshooting measures I included further facilitated the transition to PID control. Data taken using this new system can be seen below in Fig. 7, which displays conductivity, temperature, and depth with respect to time.

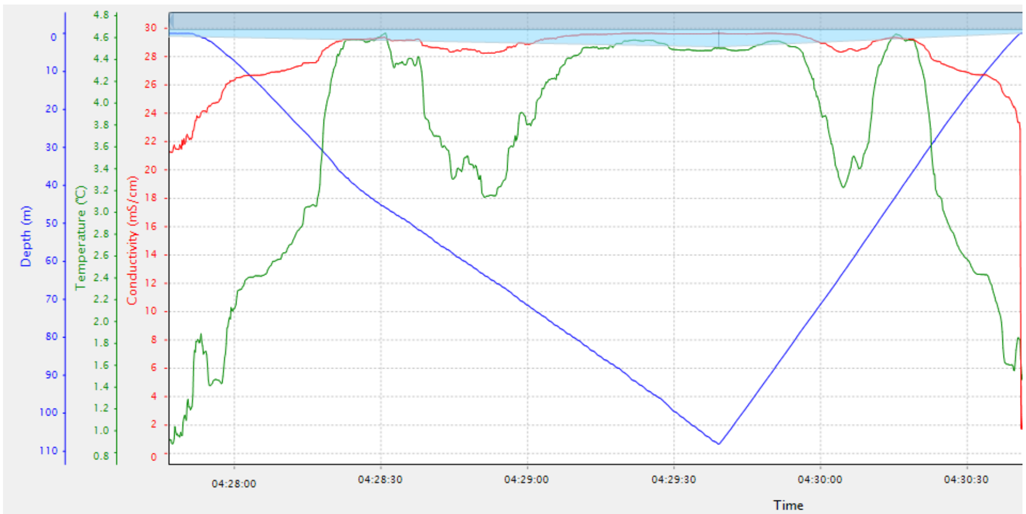

Fig. 7 – This CTD data, collected during ROSS's May 2017 Alaska deployment, illustrates the profiling system's ability to reach, and return from, depths of over 100 m at a steady rate.

I am proud to see ROSS's progress and success. Moving forward, I will continue to utilize not only the technical skills I developed, but also those in communication and project management. This experience has been one of the highlights of my undergraduate career and I am thankful to have had this opportunity.

Works Cited

[1] D. J. Nash, "ROSS: the Robotic Oceanographic Surface Sampler," Oregon State University College of Earth, Ocean, and Atmospheric Sciences, [Online]. Available: http://makani.coas.oregonstate.edu/ross/Why_ROSS.html. [Accessed 6 September 2016].

[2] J. Bruslind, Interviewee, [Interview]. 15 May 2017.

[3] "Quadrature Encoder Fundamentals," National Instruments, 14 March 2016. [Online]. Available: http://www.ni.com/white-paper/4763/en/. [Accessed 14 April 2017].

[4] Atmel Corporation, "AVR1607: Brushless DC Motor (BLDC) Control in Sensor Mode using ATxmega128A1 and ATAVRMC323," [Online]. Available: http://docs-europe.electrocomponents.com/webdocs/0fa2/0900766b80fa22a1.pdf. [Accessed 20 April 2017].

[5] C. Mullins, Artist, *Profiling System Layout.* [Art]. 2017.

[6] C. Mullins, Artist, *Magnet Position Relative to Hall Effect Sensor*. [Art]. 2017.

Appendices

Appendix A – Python Script

```python
from sys import argv
import serial
from time import sleep

def closeCheck():
    if ser.isOpen() == False:
        print "Port closed"
    elif ser.isOpen() == True:
        print "Port is still open"
        ser.close()
        closeCheck()

ser = serial.Serial(
    port='COM30',\
    baudrate=57600,\
    parity=serial.PARITY_NONE,\
    stopbits=serial.STOPBITS_ONE,\
    bytesize=serial.EIGHTBITS,\
    timeout= None) #Open serial port
ser.close()

parameters = [0,0,0,0,0,0,0]
stop = int(argv[4])

if stop == 0:
    header = 255
    speedOut = int(argv[1]) #Collect runtime parameters
    speedIn = int(argv[2])
    depth = int(argv[3])
    upperDepthByte = depth >> 8
    lowerDepthByte = depth & 0xFF
    checksum = (((speedOut ^ speedIn) ^ upperDepthByte) ^ lowerDepthByte) #XOR all variables to create checksum
    footer = 255
    parameters = [header , speedOut, speedIn, upperDepthByte, lowerDepthByte, checksum, footer] #Save parameters in array
elif stop == 1:
    for x in range(len(parameters)):
        parameters[x] = 0xAA
elif stop == 2:
    for x in range(len(parameters)):
        parameters[x] = 0xBB
elif stop == 4:
    for x in range(len(parameters)):
        parameters[x] = 0xCC
elif stop == 8:
    for x in range(len(parameters)):
        parameters[x] = 0xDD #remote start
elif stop == 16:
    for x in range(len(parameters)):
```

```
        parameters[x] = 0xEE #remote stop

print parameters

ser.open()
for x in parameters:
    x = chr(x) #cast to char to make single bite
    ser.write(x)
    sleep(0.1)

ser.close()
closeCheck()
```

Appendix B – Arduino Winch Control Program

```
//Winch control version 2.0
//#define DEBUG //Uncomment to print debugging information via serial
struct Winch_TYPE {
  uint8_t currentSpeed;
  uint8_t prevSpeed;
  uint8_t currentDir; // UP,DOWN,STOP defined in enum
  uint8_t prevDir;
  uint8_t prevDesiredDir;
  uint8_t prevDesiredSpeed;
} winch;

#include <Servo.h>
#include <Encoder.h>
#include <Timer.h>

Servo ESC; //Create ESC object
Encoder winchEncoder(3,2); //Create encoder object
Timer statusTimer; //Create a timer for sending status messages

enum{ //Assign integer values to each direction
  UP,
  DOWN,
  STOP
};

enum{  //Assign integer values to each state
  CHECK_BUFFER,
  CONTROL_WINCH,
  MAINTAIN
};

#define ENCODER_OPTIMIZE_INTERRUPTS
#define MAX_FORWARD 1910 //Maximum, minimum, and neutral pulse widths in microseconds
#define NEUTRAL 1479
#define MAX_REVERSE 1048
#define REV(x) 3936*x //Converts revolutions into encoder pings

//Speed constants
#define SLOW_DIST 2 //Distance in revolutions from full upright to begin changing winch speed in
#define LIFT_SPEED 65 //Speed for lifting the A-frame when maintaining or returning after a profile
#define FAST_IN_SPEED 65 //Speed for returning fast AND maintaining
#define SLOW_IN_SPEED 55 //Speed for returning slow AND maintaining

//Define remote start/stop pins
#define remoteStartPin 5
#define remoteStopPin 6
#define remoteStartLED 7
#define remoteStopLED 8
```

```
#define startTime 750 //How long remote start is held HIGH in milliseconds
//Define sensor pins
#define down 10//Hall Effect sensor indicating lowered position
#define downLED 13
#define up 11 //Hall Effect sensor indicating upright position
#define upLED 15

const double RAMP_TIME = 500; //Time it takes to change speed in milliseconds
const float pi = 3.14159;
uint64_t t0 = 0; //Beginning time for speed change
int16_t speedDifference = 0; //Difference between desired and current speed
int parameters[7];
int incomingByte = 0;

int state = CHECK_BUFFER;
int header;
int upperByte;
int lowerByte;
int checksum;
int buffSize = 0;
int speedOut;
int speedIn;
long long depth;
bool motorRunning = false;
bool depthReached = false;
bool halt = false;
bool returned = true;
bool dataCorrupted = false;
bool stopReturnFast = false;

void setup() {
  // put your setup code here, to run once:
  Serial1.begin(57600);
  #ifdef DEBUG
    Serial.begin(9600);
  #endif
  winch.currentSpeed = 100; //Initialize all struct values to stationary
  winch.prevSpeed = 100;
  winch.currentDir = STOP;
  winch.prevDir = STOP;
  winch.prevDesiredDir = STOP;
  winch.prevDesiredSpeed = 0;
  //Set pin modes
  pinMode(remoteStartPin, OUTPUT);
  pinMode(remoteStopPin, OUTPUT);
  pinMode(remoteStartLED, OUTPUT);
  pinMode(remoteStopLED, OUTPUT);
  pinMode(upLED, OUTPUT);
  pinMode(downLED, OUTPUT);
  pinMode(down, INPUT);
  pinMode(up, INPUT);
  //Initialize pin states
  digitalWrite(remoteStartPin, LOW);
```

```
  digitalWrite(remoteStopPin, HIGH);
  digitalWrite(remoteStartLED, LOW);
  digitalWrite(remoteStopLED, LOW);

  statusTimer.every(1000, sendStatus); //Send a status message every second

  ESC.attach(9, MAX_REVERSE, MAX_FORWARD); //Connect ESC with maximum and minimum puse width values
  ESC.writeMicroseconds(NEUTRAL); //Start the winch in neutral
  delay(5000); //Allow ESC to receive neutral signal for proper amount of time
}

void loop() {
  statusTimer.update();

  switch(state){

    case CHECK_BUFFER:
      if(buffSize == 7) //Update parameters if the serial buffer is full (new packet fully received)
         updateParameters();
      state = CONTROL_WINCH;
     break;

     case CONTROL_WINCH:
       if(header == 0xCC){ //STOP:Halt
         changeSpeed(0, STOP);
         halt = true;
         depthReached = true;
       }

       if(header == 0xAA){//STOP:Return at full speed
         changeSpeed(0, STOP);
         depthReached = true;
         if(!digitalRead(up) == false) //Return the winch to its upright position and maintian
           changeSpeed(FAST_IN_SPEED, UP);
         if(!digitalRead(up) == true){
           changeSpeed(0, STOP);
           winchEncoder.write(0);
         }
       }

       if(header == 0xBB){//STOP:Return slower
         changeSpeed(0, STOP);
         depthReached = true;
         if(!digitalRead(up) == false) //Return the winch to its upright position and maintian
           changeSpeed(SLOW_IN_SPEED, UP);
         if(!digitalRead(up) == true){
           changeSpeed(0, STOP);
           winchEncoder.write(0);
         }
       }

```

```
      if(header == 0xEE)//Stop the motor
        remoteStop();
      if(header == 0xDD)//Start the motor
        remoteStart();
      if(header == 255)//Take a profile - normal operation
        takeProfile();

      if(returned == true)
        state = MAINTAIN;
      else
      state = CHECK_BUFFER;
    break;

  case MAINTAIN:
    if(!digitalRead(up) == false)
      changeSpeed(LIFT_SPEED, UP);
    else{
      changeSpeed(0, STOP);
      winchEncoder.write(0);
    }
    state = CHECK_BUFFER;
  break;
  }
}

void changeSpeed(uint8_t newSpeed, uint8_t newDir){

  //If we want to go UP
  if(newDir == UP){
    newSpeed = 100 - newSpeed;
    //winch.currentDir = UP;
  }
  //If we want to go down
  else if(newDir == DOWN){
    newSpeed = 100 + newSpeed;
    //winch.currentDir = DOWN;
  }
  //Else we want to STOP
  else{
    newSpeed = 100;
    //winch.currentDir = STOP;
  }
  #ifdef DEBUG
    Serial1.print("[Desired: ");
    Serial1.print(newSpeed);
    Serial1.print(" Current: ");
    Serial1.print(winch.currentSpeed);
    Serial1.println("]");
  #endif


  //Check if no change is needed (if we are going the desired speed and direction)
  if(newDir == winch.currentDir && newSpeed == winch.currentSpeed){  //If the command is to continue moving the same speed and direction...
    winch.prevSpeed = winch.currentSpeed;
```

```
    winch.prevDesiredSpeed = newSpeed; //Next time we write a new speed we know it will be at t0, the beginning of a speed change
    winch.prevDesiredDir = newDir;
    #ifdef DEBUG
      Serial1.println("[REACHED END CASE]");
    #endif
    return;   //...return without altering speed
  }


  //This will catch if we have gotten to this spot while the previous call to the function was "no change requested"
  if(winch.prevDesiredSpeed != newSpeed && winch.prevDesiredDir != newDir){
    //If that's the case, then we want to initialize the acceleration
    t0 = millis();
    speedDifference = newSpeed - winch.prevSpeed;
    //Avoid errors comparing different data types. If the difference is a negative value, make it slightly more negative. Same for a positive speed difference.
    if(speedDifference < 0)
      speedDifference -= 1;
    else if (speedDifference > 0)
      speedDifference += 1;
  }

  uint64_t deltaT = millis() - t0;
  winch.currentSpeed = (double)winch.prevSpeed + (double)speedDifference*.5*(1-cos((pi*(double)deltaT)/RAMP_TIME)); //Accelerate sinusoidally
  constrain(winch.currentSpeed, 0, 200); //Do not write above or below the maximum pulse widths
  uint16_t speedToWrite = map(winch.currentSpeed, 0, 200, MAX_REVERSE, MAX_FORWARD); //Convert from sinusoid magnitude to pulse width
  ESC.writeMicroseconds(speedToWrite); //Write the scaled value
  #ifdef DEBUG
    Serial1.println(speedToWrite);
  #endif
  if(winch.currentSpeed == newSpeed)
    winch.prevSpeed = newSpeed; //Set the starting point for the next speed change

  if(winch.currentSpeed > 100)
    winch.currentDir = DOWN;
  else if(winch.currentSpeed < 100)
    winch.currentDir = UP;
  else if(winch.currentSpeed == 100)
    winch.currentDir = STOP;

  winch.prevDesiredSpeed = newSpeed; //Next time we write a new speed we know it will be at t0, the beginning of a speed change
  winch.prevDesiredDir = newDir;
}

void serialEvent1(){
```

```
  if(Serial1.available()){
    parameters[buffSize] = Serial1.read();
    buffSize++;
  }
}

void updateParameters(){
  header = parameters[0];
  speedOut = parameters[1]; //Save array contents to corresponding variables
  speedIn = parameters[2];
  upperByte = parameters[3];
  lowerByte = parameters[4];
  checksum = parameters[5];
  buffSize = 0; //Reset buffer size and control variables
  depthReached = false;
  halt = false;
  if(checksum == (((speedOut ^ speedIn) ^ upperByte) ^ lowerByte)){
    upperByte = upperByte << 8;
    depth = upperByte + lowerByte;
    depth = REV(depth); //pings/revolution
    speedOut = speedOut*100/254; //Scale from 0-254 to 0 - 100
    speedIn  = speedIn*100/254; //Scale from 0-254 to 0 - 100
    dataCorrupted = false;
  }
  else{
    depth = 0;
    speedOut = 0;
    speedIn = 0;
    dataCorrupted = true;
  }
}

void sendStatus(){
  Serial1.print("STATUS ");
  if(dataCorrupted == false){
    if(returned == true){
      Serial1.print("1  "); //Ready
    }
    else{
      Serial1.print("0  "); //Busy
    }
  }
  else{
    Serial1.print("3  "); //Data corrupted
    dataCorrupted = false;
  }
  Serial1.print("Dir ");
  if(winch.currentDir == UP)
    Serial1.print("up  ");
  else if(winch.currentDir == DOWN)
    Serial1.print("down  ");
  else if(winch.currentDir == STOP)
    Serial1.print("stationary  ");

  Serial1.print("Rev ");
```

```
  long long pingsFromSurface = winchEncoder.read();
  pingsFromSurface = pingsFromSurface/3936;
  long revsFromSurface = (long) pingsFromSurface;
  Serial1.println(revsFromSurface);
}

inline void remoteStart(){
  if (motorRunning == false){ //Prevent remote start from executing if motor already running
    digitalWrite(remoteStopPin, LOW);
    digitalWrite(remoteStartPin, HIGH);
    digitalWrite(remoteStartLED, HIGH);
    delay(startTime);
    digitalWrite(remoteStartPin, LOW);
    digitalWrite(remoteStartLED, LOW);
    motorRunning = true;
  }
}

inline void remoteStop(){
  if(motorRunning == true){
    digitalWrite(remoteStopPin, HIGH);
    digitalWrite(remoteStopLED, HIGH);
    motorRunning = false;
  }
}

void takeProfile(){
   if(depthReached == false){
     //Can't use switches with current version of Aux Board
    if(!digitalRead(down) == false){//Slowly let A-frame down from upright position
      changeSpeed(30, DOWN);
      returned = false;
    }
    else if((winchEncoder.read() < depth)){
      changeSpeed(speedOut, DOWN);
      returned = false;
    }
    else if(winchEncoder.read()>= depth){
      changeSpeed(0, STOP);
      depthReached = true;
      returned = false;
    }
  }
  else if(depthReached == true && halt == false){
    if(!digitalRead(up) == true){ //Stop when A-frame is in full upright position
      changeSpeed(0, STOP);
      winchEncoder.write(0); //Account for line stretching - reset after each cast
      returned = true;
    }
    else if(winchEncoder.read() > 20000){ //change back to else if
      changeSpeed(speedIn, UP);
      returned = false;
```

```
    }
    else if(winchEncoder.read() > 0){
      changeSpeed(LIFT_SPEED, UP);
      returned = false;
    }
    else if(!digitalRead(down) == false && !digitalRead(up) == false){ //Slow down when A-fram lifts up
      changeSpeed(LIFT_SPEED, UP);
      returned = false;
    }
    else if(winchEncoder.read() <= (-500*3936)){ //Winch will stop if line snaps and can't engage sensors
      changeSpeed(0, STOP);
      returned = true;
    }
  }
}
```

Appendix C – Wireless Data Download BASH Script

```bash
#!/bin/bash

profileSize=$(wc --bytes < /home/pi/CTD/LogFiles/combinedProfile.txt)
echo $profileSize

timeStamp=$(date +%Y%m%d%H%M%S)

./logger2wifidownloader -offset=$profileSize -length=all > /home/pi/CTD/LogFiles/placeHolder.txt

cat /home/pi/CTD/LogFiles/placeHolder.txt > /home/pi/CTD/LogFiles/Profile-$timeStamp.txt
cat /home/pi/CTD/LogFiles/placeHolder.txt >> /home/pi/CTD/LogFiles/combinedProfile.txt

cat /home/pi/CTD/LogFiles/combinedProfile.txt | /home/pi/CTD/ParseReader/a.out 6 > /home/pi/CTD/LogFiles/combinedParsedProfiles.txt
cat /home/pi/CTD/LogFiles/Profile-$timeStamp.txt | /home/pi/CTD/ParseReader/a.out 6 > /home/pi/CTD/LogFiles/parsedProfile-$timeStamp.txt
```