Project 2 Report

CSE 398

Syracuse University School of Engineering and Computer Science
Kyle Betten, Ibrahima Diallo

II. Table of Contents

*** Links in this document may not work on every platform. For easy navigation of the report visit: https://docs.google.com/document/d/1SbRTttv8ykdLLV_l5WkLMr5OEit0elnN3HSjy9brwBQ/edit?usp=sharing ***

III. Introduction

In project 2 our goal is to communicate data between the Arduino Nano and Raspberry Pi 2 zero W devices. One of the main learning outcomes of this project was learning how to send/receive packets of information between devices rather than just the data. This is because packets will keep the integrity of the data sent. Along with these devices the data was collected using a force sensor and an acceleration sensor. Using these sensors we have an example of real world applications, for example a car tracking potholes on its way to a destination. The device we used to detect the acceleration and bump traction was the BNO055 IMU. We used this device's code library written by Adafruit's computer engineers. We measured the acceleration as a float at 20 times per second. We used a specific formula to calculate the filtered acceleration.

IV. Design and Implementation Plan

We broke down our project into several demos to make progress incremental and give us the ability to verify that certain parts were working before moving on with the whole.

Breakdown of Day-to-Day Signatures:

Day 1:

On Day 1 we showed that we can properly interface the BNO055 sensor with the Arduino by printing Absolute Orientation Euler data to the Arduino Serial Monitor

Day 2:

On Day 2 we showed that we can properly detect a bump by continuously printing acceleration data to the serial monitor and printing "We hit a bump" when the acceleration passed a certain value.

Day 3:

The demonstration for Day 3 was to show that we can read and write to the RPi by writing 'A', 'B', and 'Cc' from Waveforms and printing to the RPi's console.

Day 4:

Our demonstration on Day 4 was to show that we can complete the full process of sending a packet by receiving acceleration packets on the RPi from the Arduino and verifying that the checksums matched on both sides.

Day 5:

On Day 5 we demonstrated that we can receive force sensor data, send that data in a packet, receive the data on the RPi, and print the contents of the packet on the RPi's console.

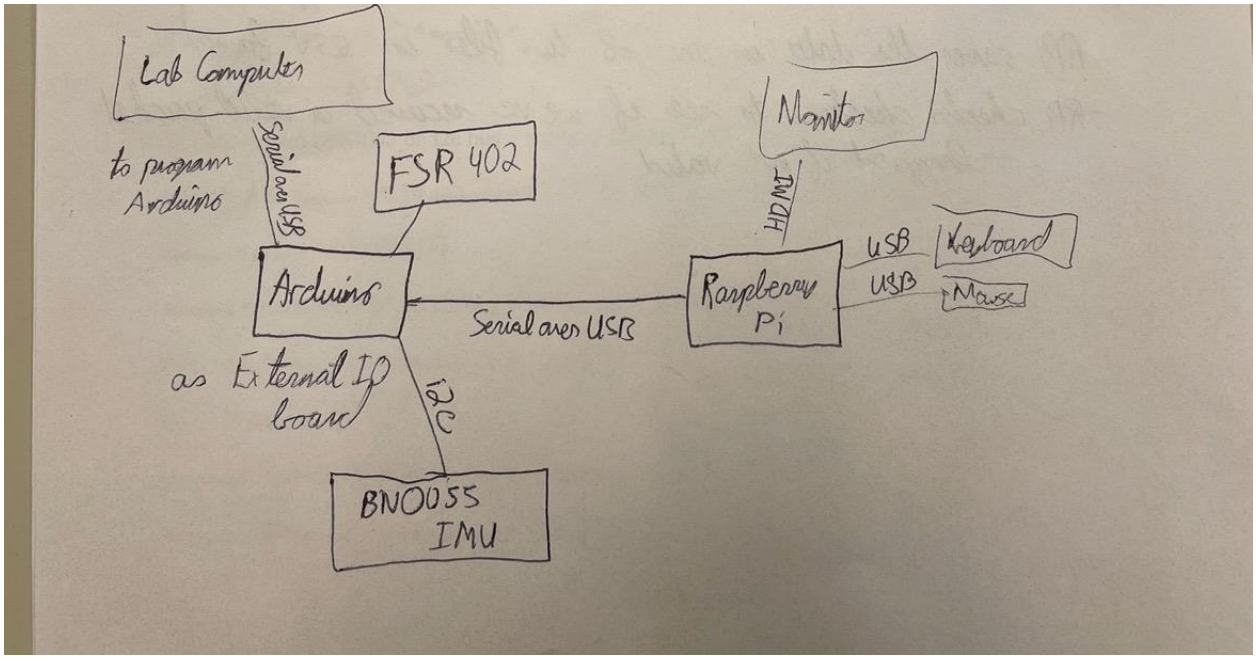
Day 6:

On Day 6, we planned to show complete implementation of the project by sending a force measurement packet and an acceleration packet

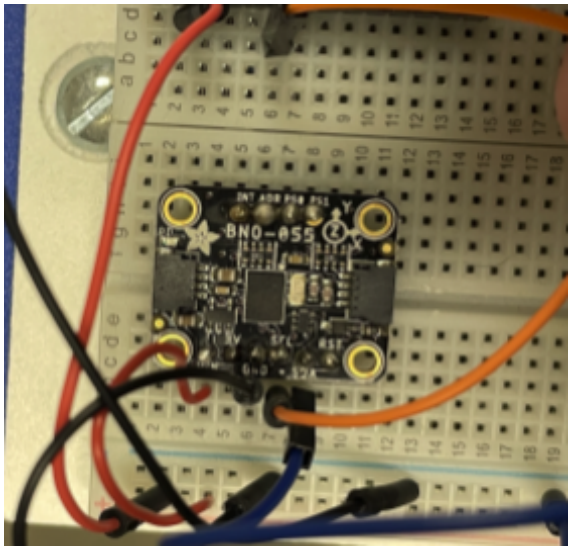
Summary of the contributions of the group members:
Kyle: Helped write a decent portion of the notebook. Wrote Arduino code once the group had to switch from Ibra's laptop to mine. Tried getting the packets to also send a date and time and convert the output to a .csv file.
Ibrahima: Helped wire every component with the group, created design and implementation plan (the signatures and demos), helped write the Arduino code
Tysean:

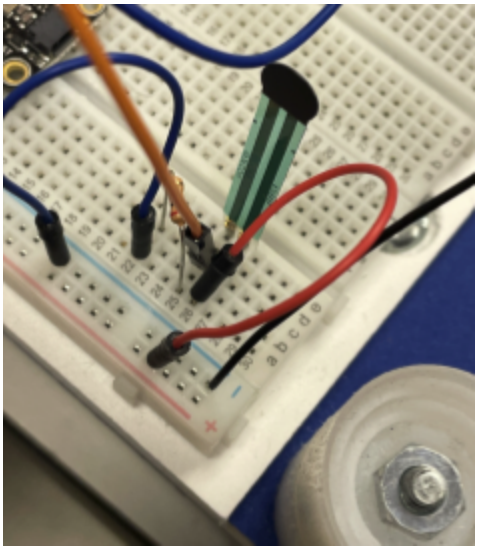
V. Description of Design Results

The setup of the hardware is as shown in this diagram:



We had the sensors attached to the Arduino (because it excels in real-time measurements). We wired the BNO055 to the Arduino with lines GND -> GND, VIN -> 3.3V, SCL-> SCL on the Arduino, SDA-> SDA on the Arduino. The Force Sensing Resistor (FSR) was also connected to the Arduino. The BNO055 and FSR look like this when properly interfaced:





We used both the Arduino and the Raspberry Pi in this project and had to program on both ends. Our Arduino code had one sketch and our Raspberry Pi code consisted of main.cpp, COBSdecoder.h, and COBSdecoder.cpp.

Arduino Code

This sketch handled everything that needed to be done on the Arduino. We combined the three acceleration measurements in the acc vector with the equation float a_measured = sqrt(pow(acc.x(),2) + pow(acc.y(),2) + pow(acc.z(),2));. We then set up an array of 40 floats to store the acceleration measurements to be used in the packet. We wrote a for loop to update index 19 in the array with the new measured acceleration every loop and shift indices . This was important so that whenever we did sense a bump we had access to the 20 acceleration measurements from the past second. We chose abs(a_measured) > (0.25*9.81) to be the condition for when a bump is detected. Once that happens we print "We hit a bump!!" to the serial monitor, send the first 20 float measurements into the packet, print the previous second into the monitor, take the next second of measurements, put the next second of measurements in the packet, print that next second to the monitor, and finally calculate the checksum. Instead of Serial.write(), we used myPacketSerial.send() to send the full contents of our packet to the RPI.

We used Arduino's analog read to quantify the force on the force sensor. Testing to send the force sensor data as a packet was very similar to sending packets of the acceleration. The value that we returned from the Arduino's ADC through Analog pin A1 was sent through our formula: float F = pow((((1024*20000)/sensorValue) - 20000)/(pow(10, 5.88)),(1.0/-0.8));
We decided that the condition for a button press should be abs(F) > 50. Once the condition was met we printed "We touched the butt...on" to the monitor, created the packet, added angles, added a different header (one for force packets now), printed the full 40 force measurements to the console, added them to the packet, and sent one.

Main.cpp

This file defines what should be in the packet struct. This file has the code for testing the COBS files.

This file also includes the code that we wrote for reading and writing with the RPi. We wired the TX and RX lines of the Arduino to the RX and TX pins of the RPi. We used the "Writing to Waveforms Test" section of our code in this part as well to send the characters 'A' 'B' and 'C' from the Arduino and print them to the console of the RPI.

In the final code (which is also in this file), we handled receiving force and acceleration packets. We wrote code in main to separate and print out the contents of our packet to the console. We also calculated a checksum on the RPi side of things and printed that checksum to the console. Our checksum ended up being an 8 bit integer on both ends. The formula that we used to calculate the checksum was to simply add up the ASCII value of the header and the 40 acceleration/force measurements.

In testing the force packets, when the packet was received on the RPi we decoded like we did earlier and printed all of the information to the console, verifying that it matched what was on the Arduino and that the numbers made sense in the context of our experiment.

COBSdecoder.h
Header file for COBSdecoder.h

COBSdecoder.cpp
File used to decode the packets sent with Consistent Overhead Byte Stuffing encoding.

VII. Assessment of Experimental Results

In the end, the group was able to communicate successfully between the devices without the date or time. The checksum values would match on both sides of the communication which would allow for the receiving side to properly get all of the information. Which in turn means that groups are able to send information in packets across devices such as Raspberry Pis and Arduinos. Packets are vital since it is the only measure that can make sure the information sent is correct and in order. Another outcome of this lab was getting information from multiple parts of the experiment. Having the two different sensors, acceleration and force sensors, showed how a project can have many different parts. Unfortunately, the group was not able to incorporate a date and time to the packets and output the data into a .csv file. Finishing this part of the lab would give the group insight on how companies could access the information easily after the

packet is sent. Overall, the experiment was incredibly useful for students to learn how to combine systems.

VIII. Appendix

Arduino Code:

```
#include <Wire.h>
#include <Adafruit_BNO055.h>
#include <Adafruit_Sensor.h>
#include <utility/imumaths.h>
#include <PacketSerial.h>

//using namespace LibSerial

Adafruit_BNO055 bno = Adafruit_BNO055(55);

float a_prev = 0; //Global because it is needed across loops

//Indicies 0-19 will be constantly updated
//Indicies 20-39 will be updates when we hit a bump
float previousSecondAcc[40];
float previousSecondForce[40];

PacketSerial myPacketSerial;

struct send_packet {
  char header;
  float measurement[40];
  float orient_yaw;
  float orient_pitch;
  float orient_roll;
  uint8_t checksum;
};

/************ SETUP ************ SETUP ************ SETUP ************/

void setup() {

  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial1.begin(9600);
  Serial.println("Setup called"); Serial1.println("");

  //Initialize the sensor
  if(!bno.begin()) {
    Serial.print("No BNO055 detected. Check wiring or I2C Addr.");
          while(1);
  }

  delay(1000);

  bno.setExtCrystalUse(true);
```

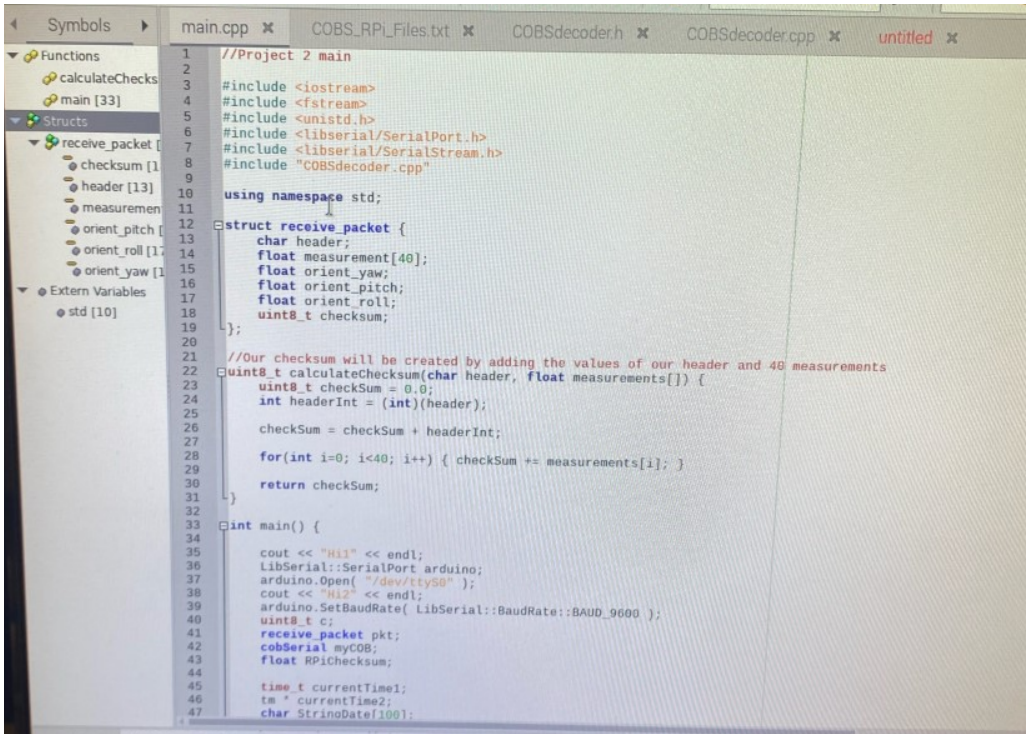
```
  myPacketSerial.setStream(&Serial);
  myPacketSerial.setStream(&Serial1);
}

/************ LOOP ************ LOOP ************ LOOP ************/

/*sending basic packet
send_packet testpkt;
testpkt.header = 'testeberger';
testpkt.checksum = 27;
myPacketSerial.send((uint8_t*)&testpkt, sizeof(testpkt)); */


void loop() {
  // put your main code here, to run repeatedly:



  //Euler angle
  imu::Vector<3> euler = bno.getVector(Adafruit_BNO055::VECTOR_EULER);
  //Accelation vector
  imu::Vector<3> acc = bno.getVector(Adafruit_BNO055::VECTOR_LINEARACCEL);

  int sensorPin = A1;
  int sensorValue = analogRead(sensorPin);
  float F = pow((((1024*20000)/sensorValue) - 20000)/(pow(10, 5.88)),(1.0/-0.8));

  /*
  // Orientation (euler angles)
  Serial.print("X: ");
  Serial.println(euler.x());
  Serial.print(" Y: ");
  Serial.println(euler.y());
  Serial.print(" Z: ");
  Serial.println(euler.z());

  // Acceleration
  Serial.print("X: ");
  Serial.println(acc.x());
  Serial.print(" Y: ");
  Serial.println(acc.y());
  Serial.print(" Z: ");
  Serial.println(acc.z());
  */

  //Combined Acceleration
  float a_measured = sqrt( pow(acc.x(),2) + pow(acc.y(),2) + pow(acc.z(),2) );

  /*
  //Filtered Acceleration
  float a_fil = (0.86*a_prev) + (0.14*a_measured);


  Serial.print("Measured Acceleration ");
  Serial.println(a_measured);
  */
```

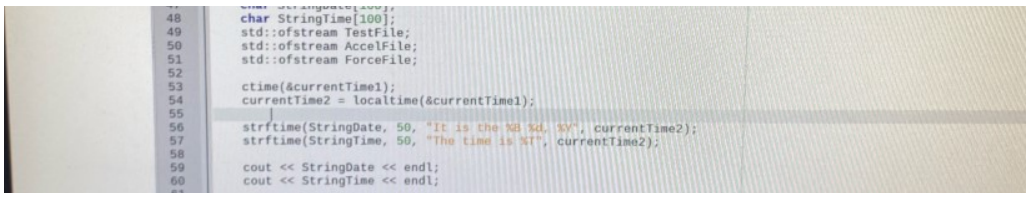
```
/*
Serial.print("Previous Acceleration ");
Serial.println(a_prev);
Serial.print("Filtered Acceleration ");
Serial.println(a_fil);
Serial.println(acc.z());
Serial.println("");
*/

//On every loop we need to update the array
for(int i=0; i<19; i++) {
  previousSecondAcc[i] = previousSecondAcc[i+1];
  previousSecondForce[i] = previousSecondForce[i+1];

}
previousSecondAcc[19] = a_measured;
previousSecondForce[19] = sensorValue;

// Button Pressed
if(abs(F) > 50){

  Serial.print("We touched the butt...on");

  send_packet pkt;
  pkt.header = 'm';                         // button header
  pkt.orient_yaw = euler.z();
  pkt.orient_pitch = euler.y();
  pkt.orient_roll = euler.x();

          for(int i=0; i<=19; i++) {
    pkt.measurement[i] = previousSecondForce[i];
          }

          //Print previous second array
          for(int i=0; i<=19; i++) {
    Serial.println(pkt.measurement[i]);
    Serial.print(" ");
          }
  Serial.println("");

          //Future second into the packet
          for(int i=20; i<40; i++) {
    sensorValue = analogRead(sensorPin);
          F = pow((((1024*20000)/sensorValue) - 20000)/(pow(10, 5.88)),(1.0/-0.8));
    pkt.measurement[i] = F;
          delay(50);
          }
  Serial.println("");
  Serial.println("Done Recording");

  pkt.checksum = 0;
          for (int i=0; i<40; i++){
    pkt.checksum += pkt.measurement[i];
          }
  pkt.checksum += (int)(pkt.header); //Calculates the checksum on Arduino's side
```

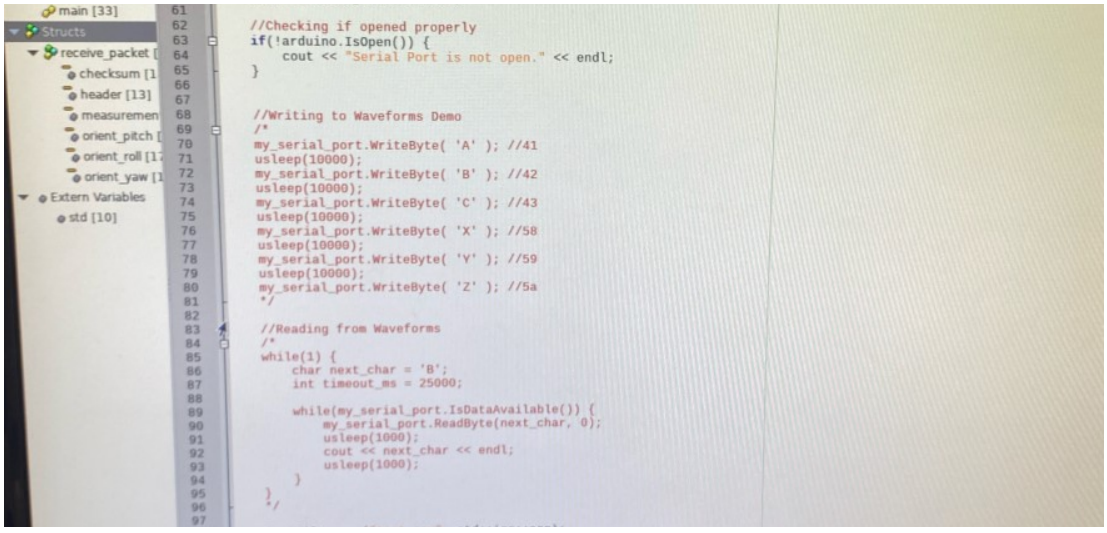
```
  myPacketSerial.send((uint8_t*)&pkt, sizeof(pkt));

}

// Bump detected
if(abs(a_measured) > (0.25*9.81))
{
  Serial.println("We hit a bump!!!");

  send_packet pkt;
  pkt.header = 'z';                         // bump header
  pkt.orient_yaw = euler.z();
  pkt.orient_pitch = euler.y();
  pkt.orient_roll = euler.x();


          //Previous second into the packet
          for(int i=0; i<=19; i++) {
    pkt.measurement[i] = previousSecondAcc[i];
          }

          //Print previous second array
          for(int i=0; i<=19; i++) {
    Serial.println(pkt.measurement[i]);
    Serial.print(" ");
          }
  Serial.println("");

          //Future second into the packet
          for(int i=20; i<40; i++) {

          acc = bno.getVector(Adafruit_BNO055::VECTOR_LINEARACCEL);
      a_measured = sqrt( pow(acc.x(),2) + pow(acc.y(),2) + pow(acc.z(),2) );

          //Serial.print("Measured Acceleration ");
          //Serial.println(a_measured);

          // /*
          //Gather the data for the packet
      pkt.measurement[i] = a_measured;
          //
      Serial.print(pkt.measurement[i]);
      Serial.print(" ");


          delay(50);
          }
  Serial.println("");
  Serial.println("Done Recording");

  pkt.checksum = 0;
          for (int i=0; i<40; i++){
    pkt.checksum += pkt.measurement[i];
          }
  pkt.checksum += (int)(pkt.header); //Calculates the checksum on Arduino's side
  myPacketSerial.send((uint8_t*)&pkt, sizeof(pkt));
```

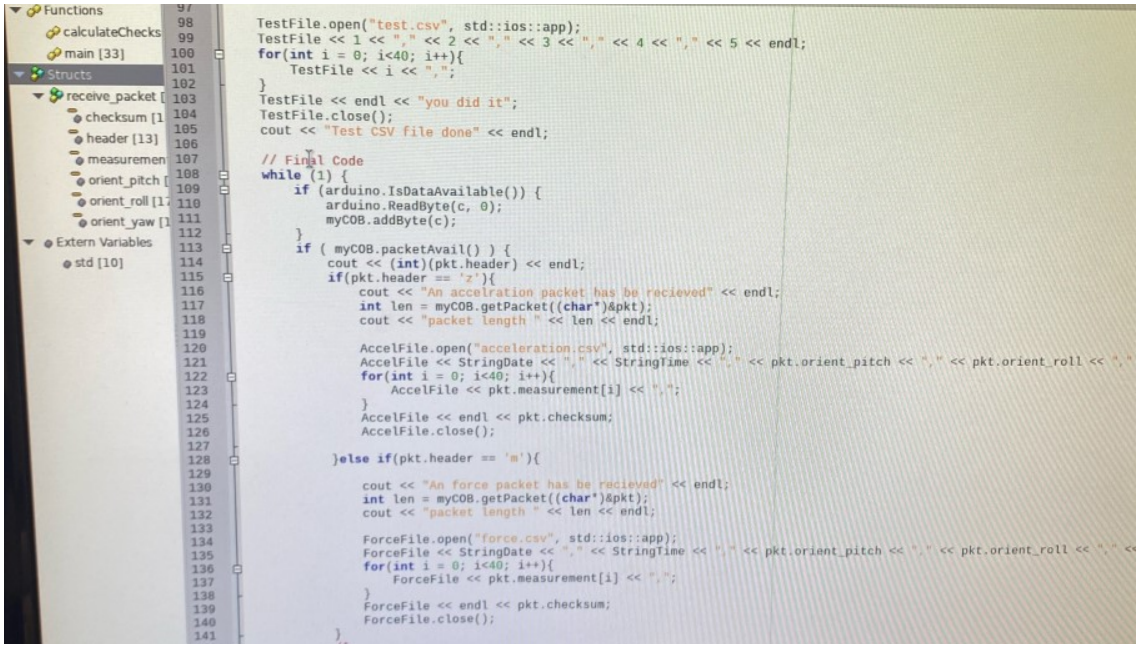
```
  }

  //Update previous
  //a_prev = a_fil;

  //Writing to Waveforms Test
  /*
  while(1) {
    Serial.write( 'A' );
    Serial.write( 'B' );
    Serial.write( 'C' );
    Serial.println("");
            delay(1000);
  }
  */

  //Reading from Waveforms Test
  /*
  if (Serial.available() > 0) {
            // get incoming byte:
    inByte = Serial.read();
    Serial.println(thirdSensor);
  }
  */

  delay(50);


}
```

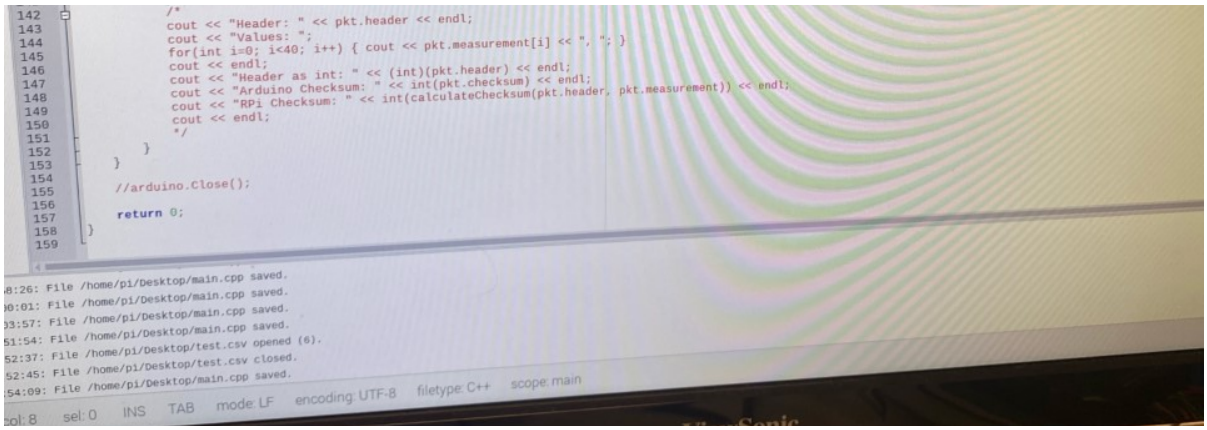
Main.cpp:


Symbols
main.cpp
COBS_RPi_Files.txt
COBSdecoder.h
COBSdecoder.cpp
untitled
Functions
calculateChecks
main [33]
Structs
receive_packet [
checksum [1
header [13]
measuremen
orient_pitch [
orient_roll [1
orient_yaw [1
Extern Variables
std [10]
//Project 2 main
#include <iostream>
#include <fstream>
#include <unistd.h>
#include <libserial/SerialPort.h>
#include <libserial/SerialStream.h>
#include "COBSdecoder.cpp"
using namespace std;
struct receive_packet {
char header;
float measurement[40];
float orient_yaw;
float orient_pitch;
float orient_roll;
uint8_t checksum;
};
//Our checksum will be created by adding the values of our header and 40 measurements
uint8_t calculateChecksum(char header, float measurements[]) {
uint8_t checkSum = 0.0;
int headerInt = (int)(header);
checkSum = checkSum + headerInt;
for(int i=0; i<40; i++) { checkSum += measurements[i]; }
return checkSum;
}
int main() {
cout << "Hi1" << endl;
LibSerial::SerialPort arduino;
arduino.Open("/dev/ttyS0");
cout << "Hi2" << endl;
arduino.SetBaudRate(LibSerial::BaudRate::BAUD_9600);
uint8_t c;
receive_packet pkt;
cobSerial myCOB;
float RPiChecksum;
time_t currentTime1;
tm * currentTime2;
char StringDate[100];


char StringTime[100];
std::ofstream TestFile;
std::ofstream AccelFile;
std::ofstream ForceFile;
ctime(&currentTime1);
currentTime2 = localtime(&currentTime1);
strftime(StringDate, 50, "It is the %B %d, %Y", currentTime2);
strftime(StringTime, 50, "The time is %T", currentTime2);
cout << StringDate << endl;
cout << StringTime << endl;


main [33]
Structs
receive_packet [
checksum [1
header [13]
measuremen
orient_pitch [
orient_roll [1
orient_yaw [1
Extern Variables
std [10]
//Checking if opened properly
if(!arduino.IsOpen()) {
cout << "Serial Port is not open." << endl;
}
//Writing to Waveforms Demo
/*
my_serial_port.WriteByte('A'); //41
usleep(10000);
my_serial_port.WriteByte('B'); //42
usleep(10000);
my_serial_port.WriteByte('C'); //43
usleep(10000);
my_serial_port.WriteByte('X'); //58
usleep(10000);
my_serial_port.WriteByte('Y'); //59
usleep(10000);
my_serial_port.WriteByte('Z'); //5a
*/
//Reading from Waveforms
/*
while(1) {
char next_char = 'B';
int timeout_ms = 25000;
while(my_serial_port.IsDataAvailable()) {
my_serial_port.ReadByte(next_char, 0);
usleep(1000);
cout << next_char << endl;
usleep(1000);
}
}
*/





COBSdecoder.h:

// The COBS decoder class header file

#include <iostream>

class cobSerial {
public:
cobSerial();
~cobSerial();
void addByte(uint8_t c);
bool packetAvail();
int getPacket(char* ptrOutput);
static const int constMaxPacketSize = 256;
private:
void decode();

```
		bool bAvail;
		char buff[constMaxPacketSize];
		int buffIdx;
		char decodedBuff[constMaxPacketSize];
		int decodeIdx;
};
```

COBSdecoder.cpp:

```
// The COBS decoder cpp file

#include "COBSdecoder.h"
#include <cstring>

using namespace std;

cobSerial::cobSerial() {
	buffIdx = 0;
	decodeIdx = 0;
	bAvail = false;
}

cobSerial::~cobSerial() {
}

void cobSerial::addByte(uint8_t c) {
	if ( c == 0 ) {
		decode();
		bAvail = true;
		buffIdx = 0;
	} else {
		buff[buffIdx] = c;
		buffIdx++;
	}
}

bool cobSerial::packetAvail() {
	return bAvail;
}
int cobSerial::getPacket(char* ptrOutput) {
	if (bAvail) {
		memcpy(ptrOutput, decodedBuff, decodeIdx);
		bAvail = false;
		return decodeIdx;
	} else {
		return -1;
	}
}
```

```
void cobSerial::decode() {
	int i = 0;
	int l = 0;
	decodeIdx = 0;
	while ( i < buffIdx) {
		l = i + buff[i];
		i++;
		while ( i < l ) {
			decodedBuff[decodeIdx] = buff[i];
			i++;
			decodeIdx++;
		}
		if ( l < buffIdx) {
			decodedBuff[decodeIdx] = 0;
			decodeIdx++;
		}
	}
}
```