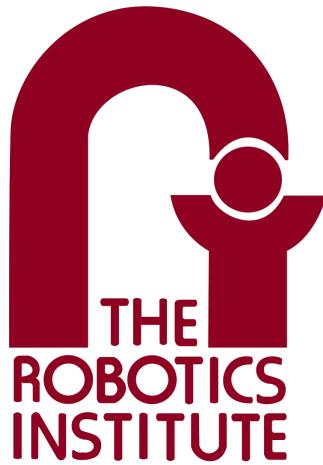

Individual Lab Report 1



Lunar ROADSTER

Team I

Author: **Boxiang (William) Fu**
Andrew ID: boxiangf
E-mail: boxiangf@andrew.cmu.edu

Teammate: **Deepam Ameria**
ID: dameria
E-mail: dameria@andrew.cmu.edu

Teammate: **Bhaswanth Ayapilla**
ID: bayapill
E-mail: bayapill@andrew.cmu.edu

Teammate: **Simson D'Souza**
ID: sjdsouza
E-mail: sjdsouza@andrew.cmu.edu

Teammate: **Ankit Aggarwal**
ID: ankitagg
E-mail: ankitagg@andrew.cmu.edu

Supervisor: **Dr. William “Red” Whittaker**
Department: Field Robotics Center
E-mail: red@cmu.edu

February 4, 2025

1 Individual Progress

1.1 Sensors and Motor Control Lab

I was responsible for developing and integrating a standalone graphical user interface (GUI) for the sensors and motor control lab. The GUI was done using the Qt Creator toolkit with customized packages ‘Q Custom Plot’ for plotting the transfer function and ‘Q Serial Port’ for interfacing with an Arduino Due via serial port. Figure 1 displays the layout of the GUI.

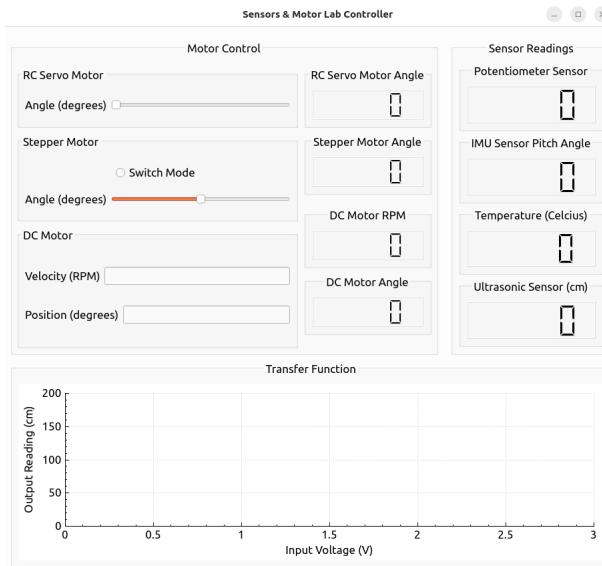


Figure 1: GUI Layout

The ‘Motor Control’ panel controls the 3 motors given out for the lab. The RC servo motor is controlled by a slider with a range between 0 to 180 degrees to control its desired angle. The stepper motor is controlled by a slider with a range between -180 to 180 degrees to control its desired angle. A switch button is used to change motor control between the GUI and the potentiometer. The DC motor is controlled by two text boxes. When velocity control is desired, the velocity text box is used to control the RPM of the DC motor within the range of -118 to 118 RPM. When position control is desired, the angle slider is used to control the angle in degrees of the DC motor within the range of -360 to 360 degrees from its current angle. Negative values switch the motor’s direction of rotation. The terminal values of each motor’s range is determined from the specification sheets of the motors given to us.

Once the GUI detects a change in the control values, it calls the `{Motor}_valueChanged` function. After converting the slider value to a predetermined format that is readable by the microcontroller, it is passed through the serial port to the Arduino Due. Once the command reaches the Arduino, it runs through the `handleCommand` function to determine which motor controller to send the command to. It then calls the appropriate motor controller function `{Motor}Controller` along with the passed value. The baton is then passed to my teammates to implement the inner workings of the motor controller function of their designated motor.

On the right side of the ‘Motor Control’ panel, the state of the motor is displayed via a LCD number. This display, along with the ‘Sensor Readings’ panel, are updated with a state callback function called at 5 Hz. Every 0.2 seconds, the GUI sends a request to the Arduino for its motor and sensor states. The Arduino then calls `{Motor}StateCallback`

for each motor and {Sensor}Callback for each sensor. The inner workings of each callback function was done by my teammates. Finally, the callback returns are concatenated and sent to the GUI in a predetermined format. The GUI back-end unpacks this and displays the values on the interface.

At the bottom of the GUI, a transfer function plot is displayed. It plots the input voltage of an ultrasonic range finder on the x -axis and the physical distance between the sensor and the detected object in centimeters on the y -axis. The plotter function was done by me using the ‘Q Custom Plot’ extension. The values feeding into the plot was completed by my teammate Bhaswanth. The completed physical system is shown in Figures 2 and 3.

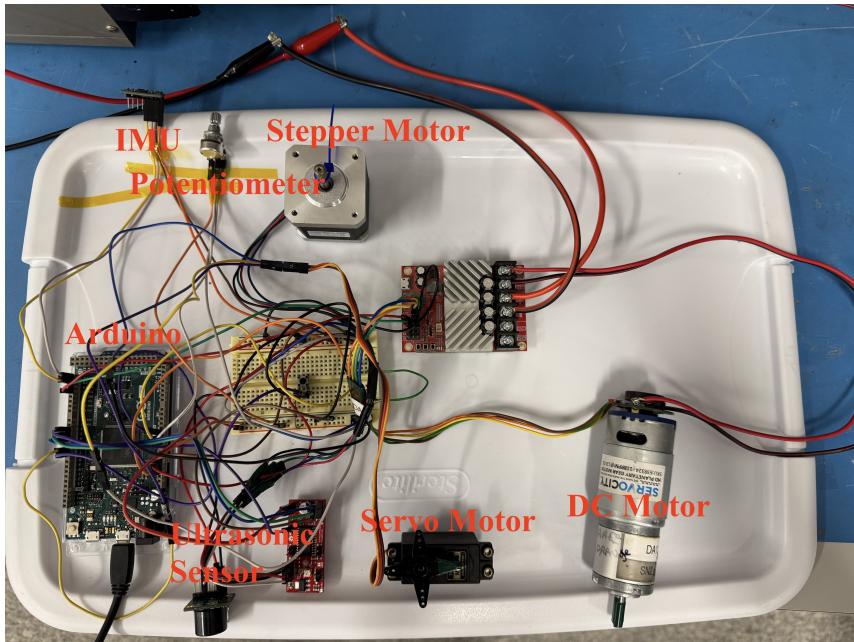


Figure 2: Sensors & Motor Control Lab Physical Setup

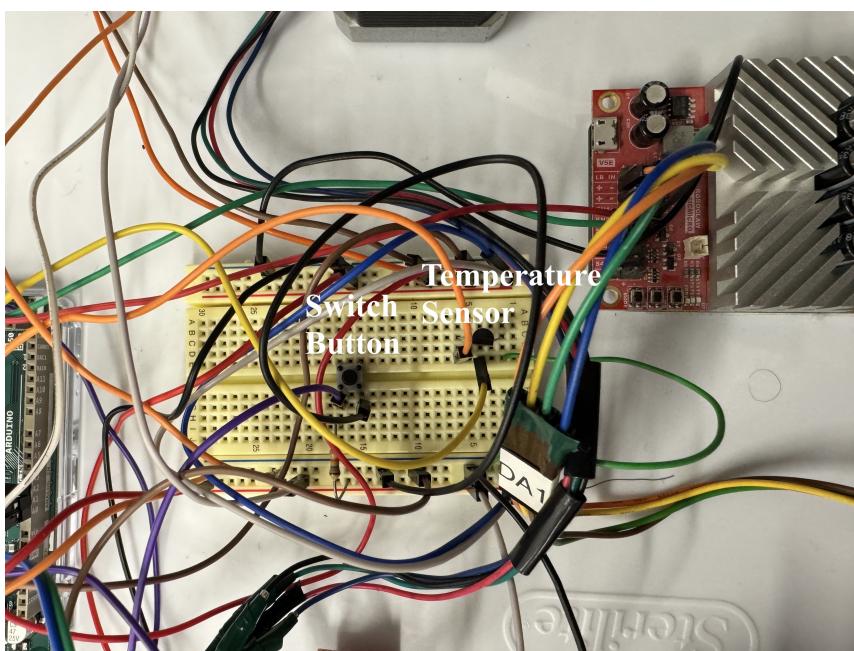


Figure 3: Close-up of Breadboard Setup

1.2 MRSD Project

Since the submission of the Conceptual Design Review Report, I have worked on the following tasks on the MRSD project:

1.2.1 Teleoperation of the Rover

The software stack left by the previous MRSD team Crater Grader already had a teleoperation stack to control the rover using a joystick. However, the documentation of how to use it is poor and outdated (e.g., the documentation specified to use the launch file `joystick_launch.py` where instead `motion_control_launch.py` should be used). I spent the majority of the winter break documenting Crater Grader's teleoperation stack and updating it to ROS2 Humble and Ubuntu 22.04. The updated documentation can be seen in Figures 4 and 5.

Lunar ROADSTER & Crater Grader Docker Bringup

The docker image is contained within the USB attached to the Jetson. The following procedure should be followed to obtain access to the Crater Grader docker container:

1. `sudo chmod 755 /usb /sd_card`
2. Type `df -h` to find mounting port of the USB drive. It should be `/dev/sda1` or `/dev/sdb1`
3. `sudo mount /dev/sda1 /usb` (replace with `/dev/sd{Port}1` if it is mounted elsewhere. You should see `CRATER_GRADER` under `ls`, if not `cd ..` and then `cd /usb` back)
4. `sudo mount /dev/mmcblk1p1 /sd_card`
5. Go inside `cd /sd_card`
6. `sudo systemctl restart docker` (you might need to rerun `sudo chmod 755 /usb /sd_card` if permissions are revoked)
7. `sudo docker start -i lunar_roadster_dev_jetson`
8. To open a new terminal: `sudo docker exec -it lunar_roadster_dev_jetson zsh`
9. For CraterGrader:
 - a. Go inside `/usb : docker-compose up -d cg-dev-hw`
 - b. Go inside `cd ./scripts`
 - c. `source set-udev-rules.sh`
 - d. `docker-compose exec cg-dev-hw zsh`
 - e. You should now be inside Crater Grader docker and see their logo
 - f. `source ./install/local_setup.zsh`
- g. Make sure the directory `/dev/cg_dev` is present and contains `tty_arduino`. This is the hardware interface from inside the docker to the Jetson.

Connecting Additional Terminals to the Docker

1. `cd /usb/CRATER_GRADER/`
2. `docker-compose exec cg-dev-hw zsh`
3. `source ./install/local_setup.zsh`
4. If it says `zsh: corrupt history file /root/.zsh_history` copy and paste Step 3 again

Figure 4: Documentation of Crater Grader's Docker

1.2.2 Moon Pit Crater Distribution

Adhering to advice by our supervisor, we adapted the crater generation code from the Moonshot Circumnavigation team to our needs. This involves reading in a dataset of the number of craters and its diameter. Fitting it to a Log-Log probability distribution, and sampling from the distribution for a 6 meter by 6 meter area that mimics the size of the moon pit in the Planetary Robotics Lab. The purpose of this task is to generate a landscape in the moon pit that has a crater distribution that is relatively lunar-realistic. This aids in guiding how we should implement our algorithms for path planning and tool planning. An example of the generated moon pit is shown in Figure 6.

Crater Grader Teleop Bringup

1. Open at least 2 terminals connected to the docker
2. `source ./install/local_setup.zsh` on all terminals
3. Check if the LEDs on the RoboClaws (RC1, RC2, and RC3) are flashing green. If not, press the `Reset` button on the Arduino
4. `colcon build` if not already built, then `source ./install/local_setup.zsh` again if needed
5. `ros2 launch motion_control motion_control.launch.py`
6. Use the other terminal and change the `mux_mode` to `teleop` using `ros2 topic pub /mux_mode cg_msgs/msg/MuxMode "{mode: 3}" --once`
7. You should be able to teleop the rover now. If not, check debugging section below

Checking if Gadgets and Links are Functioning Properly (for Teleop)

The following topics should be publishing messages if the links and gadgets are functioning properly:

- Joystick to Jetson: `ros2 topic echo /joy`
- Jetson to Arduino Interface: `ros2 topic echo /teleop_cmd`
- Arduino Interface to Arduino: `ros2 topic echo /arduino_cmd`
- Arduino to Jetson: `ros2 topic echo /arduino_feedback`

One common issue that we found was that the `micro_ros_agent` was not being initialized properly. A potential checklist to solve this is as follows:

1. `source install/local_setup.zsh`
2. `ros2 run micro_ros_setup create_agent_ws.sh`
3. `ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/cg_dev/tty_arduino -v6`
4. If the logger is not showing info, press the reset button on the Arduino. You should see `recv_message` and `send_message` if the `micro_ros_agent` is set up properly.

Figure 5: Documentation of Crater Grader's Teleoperation Stack

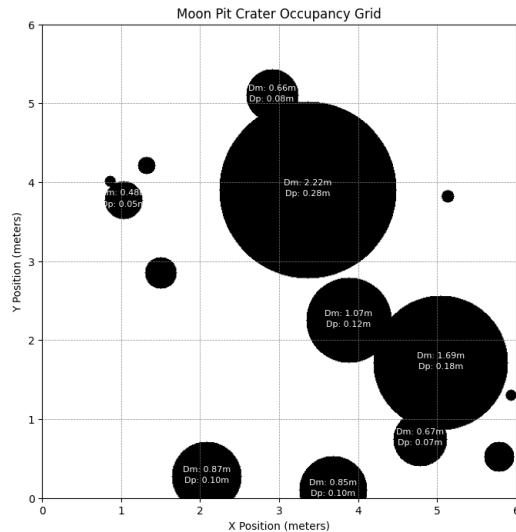


Figure 6: Moon Pit Crater Generation

1.2.3 ZED 2i Depth Camera Setup

With the help of my teammate Bhaswanth, I have set up the ZED 2i depth camera that our rover will use for localization and validation. This includes installing NVIDIA CUDA drivers and the ZED SDK drivers into our docker environment. It also involves interfacing the ZED camera and the docker container via serial port (i.e., so the data stream can be read inside the container), and converting the data stream to video using OpenCV. The video from the ZED camera will be used subsequently for the localization

and validation stacks later on in our project.

1.2.4 LAN Setup

I have set up a private LAN network for communications between devices. This is because prior MRSD teams had problems using the CMU-Secure network to transmit data during periods of high user traffic (e.g., SVD and FVD when 50+ people connect to the same WiFi port). The LAN network is connected to the internet via a wall outlet near our lab. We will be using the private network to establish communication between the rover's NVIDIA Jetson computer and the operations terminal on our laptops. Additionally, we will be using the private network to relay position information from the total station to the Jetson via a TX2 chip located beside the moon pit. Finally, static IPs were set on the private network so the connection port via `ssh` and Remmina does not change daily.

2 Challenges

2.1 Sensors and Motor Control Lab

One major challenge I faced when developing the GUI was that the serial port returns from the Arduino are best-effort delivery and not guaranteed delivery. This means that the return deliveries could be cut or that several returns could be bundled together in a single delivery. To fix this problem, I followed the solution from [this](#) repository. Essentially, it involves creating a serial buffer that waits until at least 2 returns are read. It discards the first return since it may be missing the front part of the message. The second message is used after checking that the message is of the correct length (so no lost packets occurred).

2.2 MRSD Project

The biggest challenge I faced for the MRSD project was getting Crater Grader's teleoperation stack to work. This is because the teleoperation README file that they provided was outdated and incorrect. I spent several days going through their code to understand how the ROS2 nodes were related. I found that there was a missing link between the `/teleop_cmd` topic from the joystick controller and the `/arduino_cmd` topic that was sending information to the Arduino and RoboClaw microcontrollers. The link was not inside the `src/teleop` directory but instead in the `src/motion_control` directory. After figuring this out, I was able to locate the correct launch file that links all the teleoperation nodes up. However, the rover was still not moving despite the `/arduino_cmd` topic pushing commands. It took some more time to figure out that the `mux_mode` of the rover needs to be changed from autonomous to teleoperation. This is done through the command `ros2 topic pub /mux_mode cg_msgs/msg/MuxMode "{mode: 3}" --once`. Only after this step was I able to get the rover to move based on teleoperation commands.

3 Teamwork

A breakdown of the contributions of each team member are tabulated below:

3.1 Sensors and Motor Control Lab

- **Ankit Aggarwal:** Implemented the stepper motor speed and direction controller using a potentiometer.
- **Deepam Ameria:** Interfacing IMU sensor and servo motor controller.

- **Bhaswanth Ayapilla:** Interfacing Ultrasonic Range Finder, implementing median/mode filter and transfer function, README file for the complete code.
- **Simson D’Souza:** Implemented DC motor control with encoder feedback and interfaced an IR sensor for distance measurement.

3.2 MRSD Project

- **Ankit Aggarwal:** Wheel design and printing, rover hardware setup/maintenance, circuit diagram design (in collaboration with Simson), VectorNav IMU interfacing, preliminary testing (in collaboration with team), project manager.
- **Deepam Ameria:** Prototype dozer development (in collaboration with Simson), preliminary tests for teleoperation and grading with prototype dozer (in collaboration with Simson), dozer blade and mechanism design, FARO scanner setup and Moon Pit scanning (in collaboration with team).
- **Bhaswanth Ayapilla:** NVIDIA Jetson setup, setup encoder drivers, setting up teleoperation (in collaboration with William), ZED Camera setup (in collaboration with William), setting up operations terminal, FARO scanner setup and Moon Pit scanning (in collaboration with team).
- **Simson D’Souza:** Prototype dozer development (in collaboration with Deepam), preliminary tests for teleoperation and grading with prototype dozer (in collaboration with Deepam), circuit diagram design (in collaboration with Ankit), FARO scanner setup and Moon Pit scanning (in collaboration with team), processing of point cloud data to generate an occupancy grid map for navigation.

4 Plans

4.1 Sensors and Motor Control Lab

Currently, there are no future plans for the sensors and motor control lab as it is an auxiliary assignment not directly related to our MRSD project. However, we swapped the motor driver and encoder given to us for the lab with a similar one that we are using for our rover. We also swapped the Arduino UNO with an Arduino Due as it is the microcontroller we plan on using for motor controls on our rover. We hope that the increased knowledge from the lab will be transferrable to our MRSD project and help us integrate our motor control stack.

4.2 MRSD Project

From now until the next lab demo, we hope to finish the hardware of our rover and complete the localization stack in the moon pit. My individual plan is to complete the localization stack with my teammate Bhaswanth. For positional localization, this involves setting up the total station, relaying the total station data via serial port to the TX2 chip beside the moon pit, and transmitting the data via our LAN network to the Jetson on the rover. For orientation localization, this involves reading the IMU data through serial port and extracting its roll, pitch, and yaw. Additionally, the depth camera may give additional data entries for stereo camera localization. The plan is to have the datasets fused together and the rover localized through an Extended Kalman Filter.

5 Code for Sensors & Motor Control Lab Interface

5.1 GUI

I was responsible for coding the entirety of this section except for the filtering of the ultrasonic sensor. The `dialog.ui` script is made with the help of Qt Creator.

5.1.1 main.cpp

```
1 #include "dialog.h"
2 #include "qcustomplot.h"
3
4 #include <QApplication>
5
6 int main(int argc, char *argv[])
7 {
8     QApplication a(argc, argv);
9     Dialog w;
10    w.setFixedSize(940, 850);
11    w.setWindowTitle("Sensors & Motor Lab Controller");
12    w.show();
13    return a.exec();
14 }
```

Listing 1: main.cpp

5.1.2 dialog.h

```
1 #ifndef DIALOG_H
2 #define DIALOG_H
3
4 #include <QDialog>
5 #include <QSerialPort>
6 #include <QSerialPortInfo>
7 #include <QDebug>
8 #include <QtWidgets>
9
10 QT_BEGIN_NAMESPACE
11 namespace Ui {
12 class Dialog;
13 }
14 QT_END_NAMESPACE
15
16 class Dialog : public QDialog
17 {
18     Q_OBJECT
19
20 public:
21     Dialog(QWidget *parent = nullptr);
22     ~Dialog();
23
24 private slots:
25
26     void updateArduino(QString);
```

```

28     void readSerial();
29
30     void on_servo_slider_valueChanged(int value);
31
32     void on stepper_slider_valueChanged(int value);
33
34     void on stepper_mode_button_toggled(bool checked);
35
36     void parseCallbackCommands(QString command);
37
38     void updatePlot();
39
40     int filterMean(std::vector<int>&);

41
42 private:
43     Ui::Dialog *ui;
44     QSerialPort *arduino;

45
46     // Arduino Due setup
47     static const quint16 arduino_due_vendor_id = 9025;
48     static const quint16 arduino_due_product_id = 61;
49     QString arduino_port_name;
50     bool arduino_is_available;

51
52     QByteArray serialData;
53     QString serialBuffer;
54     QString parsed_data;

55
56     QTimer *dataTimer;
57     QVector<double> xData, yData;

58
59     double transferFunctionInput;
60     double transferFunctionOutput;

61
62     void onDCPosCommentSubmitted();
63     void onDCVelCommentSubmitted();

64
65     std::vector<int> ultrasonicReadings;
66 };
67 #endif // DIALOG_H

```

Listing 2: dialog.h

5.1.3 dialog.cpp

```

1  /*
2  The following starter template is adapted from the following repos ,
3      which is under a GNU GPLv3.0 license:
4  https://github.com/vannevar-morgan/Qt-RGB-LED
5  https://github.com/vannevar-morgan/Qt-Temperature-Sensor
6  */
7
8 #include "dialog.h"
9 #include "ui_dialog.h"
# include "qcustomplot.h"

```

```

10
11 #include <QSerialPort>
12 #include <QSerialPortInfo>
13 #include <QDebug>
14 #include <QtWidgets>
15
16 #include <vector>
17
18 #include <iostream>
19
20 Dialog::Dialog(QWidget *parent)
21     : QDialog(parent)
22     , ui(new Ui::Dialog)
23 {
24     ui->setupUi(this);
25
26     arduino_is_available = false;
27     arduino_port_name = "";
28     arduino = new QSerialPort(this);
29
30     serialBuffer = "";
31     parsed_data = "";
32
33     transferFunctionInput = 0;
34     transferFunctionOutput = 0;
35
36     // Initialize transfer function plot
37     QCustomePlot *plot = ui->transferFunctionPlot;
38     plot->addGraph();
39
40     plot->xAxis->setLabel("Input Voltage (V)");
41     plot->yAxis->setLabel("Output Reading (cm)");
42
43     plot->xAxis->setRange(0, 5);
44     plot->yAxis->setRange(0, 1000);
45
46     dataTimer = new QTimer(this);
47     connect(dataTimer, &QTimer::timeout, this, &Dialog::updatePlot)
48         ;
49     dataTimer->start(100);
50
51     connect(ui->dc_angle_textbox, &QLineEdit::returnPressed, this,
52             &Dialog::onDCPosCommentSubmitted);
53     connect(ui->dc_vel_textbox, &QLineEdit::returnPressed, this, &
54             Dialog::onDCVelCommentSubmitted);
55
56     /*
57      // Finds product and vendor ID of Arduino Due
58      qDebug() << "Number of available ports: " << QSerialPortInfo::
59      availablePorts().length();
60      foreach(const QSerialPortInfo &serialPortInfo, QSerialPortInfo
61          ::availablePorts()){
62          qDebug() << "Has vendor ID: " << serialPortInfo.
63              hasVendorIdentifier();
64          if(serialPortInfo.hasVendorIdentifier()){

```

```

59         qDebug() << "Vendor ID: " << serialPortInfo.
60             vendorIdentifier();
61     }
62     qDebug() << "Has Product ID: " << serialPortInfo.
63             hasProductIdentifier();
64     if(serialPortInfo.hasProductIdentifier()){
65         qDebug() << "Product ID: " << serialPortInfo.
66             productIdentifier();
67     }
68 // Connects to Arduino via serial port
69 foreach(const QSerialPortInfo &serialPortInfo, QSerialPortInfo
70 ::availablePorts()){
71     if(serialPortInfo.hasVendorIdentifier() && serialPortInfo.
72         hasProductIdentifier()){
73         if(serialPortInfo.vendorIdentifier() ==
74             arduino_due_vendor_id){
75             if(serialPortInfo.productIdentifier() ==
76                 arduino_due_product_id){
77                 arduino_port_name = serialPortInfo.portName();
78                 arduino_is_available = true;
79             }
80         }
81     }
82     if(arduino_is_available){
83         // Open and configure the serialport
84         arduino->setPortName(arduino_port_name);
85         arduino->open(QSerialPort::ReadWrite);
86         arduino->setBaudRate(QSerialPort::Baud9600);
87         arduino->setDataBits(QSerialPort::Data8);
88         arduino->setParity(QSerialPort::NoParity);
89         arduino->setStopBits(QSerialPort::OneStop);
90         arduino->setFlowControl(QSerialPort::NoFlowControl);
91         QObject::connect(arduino, SIGNAL(readyRead()), this, SLOT(
92             readSerial()));
93         qDebug() << "Connected to Arduino!";
94     } else{
95         // Give error message if not available
96         QMessageBox::warning(this, "Port error", "Couldn't find the
97             Arduino!");
98     }
99 }
100 Dialog::~Dialog()
101 {
102     if(arduino->isOpen()){
103         arduino->close();
104     }
105     delete ui;
106 }
```

```

105
106 void Dialog::updateArduino(QString command)
107 {
108     if(arduino->isWritable()){
109         arduino->write(command.toStdString().c_str());
110     }else{
111         qDebug() << "Couldn't write to serial!";
112     }
113 }
114
115 void Dialog::readSerial()
116 {
117     QStringList buffer_split = serialBuffer.split(";");
118
119     if (buffer_split.length() < 3) {
120         serialData = arduino->readAll();
121         serialBuffer = serialBuffer + QString::fromStdString(
122             serialData.toStdString());
123         serialData.clear();
124     }
125     else{
126         serialBuffer = "";
127         parsed_data = buffer_split[1];
128         // qDebug() << parsed_data << "\n";
129
130         // If the parsed data has a $ sign at index 0, then
131         // recognize it as a callback command
132         if (parsed_data.at(0) == '$') {
133             parseCallbackCommands(parsed_data);
134         }
135         else {
136             // qDebug() << parsed_data << "\n";
137         }
138
139
140 void Dialog::updatePlot()
141 {
142     xData.prepend(transferFunctionInput);
143     yData.prepend(transferFunctionOutput);
144
145     while (xData.size() > 1000) {
146         xData.removeLast();
147         yData.removeLast();
148     }
149
150     QCustomPlot *plot = ui->transferFunctionPlot;
151     plot->graph(0)->setData(xData, yData);
152     plot->replot();
153 }
154
155
156 void Dialog::parseCallbackCommands(QString command)
157 {

```

```

158     QString processedCommand = command.mid(1);
159     qDebug() << "Processed command:" << processedCommand;
160
161     // Only update GUI if processed command has 9 elements
162     QStringList processedCommand_split = processedCommand.split(",\"");
163     // qDebug() << "Command length: " << processedCommand_split.length();
164
165     if (processedCommand_split.length() == 9) {
166
167         // Motor states
168         QString servoMotorState = processedCommand_split.at(0);
169         QString stepperMotorState = processedCommand_split.at(1);
170         QString velDCMotorState = processedCommand_split.at(2);
171         QString angleDCMotorState = processedCommand_split.at(3);
172
173         ui->servo_angle_state->display(servoMotorState);
174         ui->stepper_angle_state->display(stepperMotorState);
175         ui->dc_rpm_state->display(velDCMotorState);
176         ui->dc_angle_state->display(angleDCMotorState);
177
178         // Sensor states
179         QString potentiometerSensorState = processedCommand_split.
180             at(4);
181         QString imuSensorState = processedCommand_split.at(5);
182         QString temperatureSensorState = processedCommand_split.at
183             (6);
184         QString ultrasonicSensorState = processedCommand_split.at
185             (7);
186
187         // Filtering for ultrasonic sensor
188         int ultrasonicReading = ultrasonicSensorState.toInt();
189         if (ultrasonicReadings.size() >= 5) {
190             ultrasonicReadings.erase(ultrasonicReadings.begin());
191         }
192         ultrasonicReadings.push_back(ultrasonicReading);
193         int filteredUltrasonicValue = filterMean(ultrasonicReadings
194             );
195
196         ui->potentiometer_sensor_state->display(
197             potentiometerSensorState);
198         ui->IMU_sensor_state->display(imuSensorState);
199         ui->temperature_sensor_state->display(
200             temperatureSensorState);
201         ui->ultrasonic_sensor_state->display(
202             filteredUltrasonicValue);
203         // ui->ultrasonic_sensor_state->display(
204             ultrasonicSensorState);
205
206         // Transfer function
207         QString transferFunctionState = processedCommand_split.at
208             (8);
209         QStringList transferFunctionState_split =
210             transferFunctionState.split(":");

```

```

201
202     QString transferFunctionInputString =
203         transferFunctionState_split.at(0);
204     QString transferFunctionOutputString =
205         transferFunctionState_split.at(1);
206
207     transferFunctionInput = transferFunctionInputString.
208         toDouble();
209     transferFunctionOutput = transferFunctionOutputString.
210         toDouble();
211 }
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248

```

```

249     qDebug() << "Text_box_value:" << comment << "Command:" <<
250         command;
251     updateArduino(command);
252
253 }
254
255 int Dialog::filterMean(std::vector<int>& arr) {
256     if(arr.empty()) {
257         return 0;
258     }
259
260     int sum = 0;
261
262     for (size_t i = 1; i < arr.size(); i++) {
263         sum += arr[i];
264     }
265
266     return sum / arr.size();
267 }
```

Listing 3: dialog.cpp

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <ui version="4.0">
3      <class>Dialog</class>
4      <widget class="QDialog" name="Dialog">
5          <property name="geometry">
6              <rect>
7                  <x>0</x>
8                  <y>0</y>
9                  <width>1280</width>
10                 <height>1024</height>
11             </rect>
12         </property>
13         <property name="windowTitle">
14             <string>Dialog</string>
15         </property>
16         <property name="autoFillBackground">
17             <bool>false</bool>
18         </property>
19         <widget class="QGroupBox" name="groupBox">
20             <property name="geometry">
21                 <rect>
22                     <x>10</x>
23                     <y>20</y>
24                     <width>641</width>
25                     <height>471</height>
26                 </rect>
27             </property>
28             <property name="font">
29                 <font>
30                     <pointsize>13</pointsize>
31                 </font>
32             </property>
33             <property name="title">
```

```

34      <string>Motor Control</string>
35  </property>
36  <property name="alignment">
37    <set>Qt::AlignmentFlag::AlignCenter</set>
38  </property>
39  <widget class="QGroupBox" name="groupBox_2">
40    <property name="geometry">
41      <rect>
42        <x>10</x>
43        <y>40</y>
44        <width>421</width>
45        <height>91</height>
46      </rect>
47    </property>
48    <property name="title">
49      <string>RC Servo Motor</string>
50    </property>
51    <property name="alignment">
52      <set>Qt::AlignmentFlag::AlignLeading | Qt::AlignmentFlag::AlignLeft | Qt::AlignmentFlag::AlignVCenter</set>
53    </property>
54  <layout class="QHBoxLayout" name="horizontalLayout">
55    <item>
56      <widget class="QLabel" name="label">
57        <property name="text">
58          <string>Angle (degrees)</string>
59        </property>
60      </widget>
61    </item>
62    <item>
63      <widget class="QSlider" name="servo_slider">
64        <property name="minimum">
65          <number>0</number>
66        </property>
67        <property name="maximum">
68          <number>180</number>
69        </property>
70        <property name="orientation">
71          <enum>Qt::Orientation::Horizontal</enum>
72        </property>
73      </widget>
74    </item>
75  </layout>
76 </widget>
77 <widget class="QGroupBox" name="groupBox_4">
78   <property name="geometry">
79     <rect>
80       <x>440</x>
81       <y>40</y>
82       <width>191</width>
83       <height>91</height>
84     </rect>
85   </property>
86   <property name="title">
87     <string>RC Servo Motor Angle</string>

```

```

88 </property>
89 <property name="alignment">
90   <set>Qt::AlignmentFlag::AlignCenter</set>
91 </property>
92 <layout class="QHBoxLayout" name="horizontalLayout_3">
93   <item>
94     <widget class="QLCDNumber" name="servo_angle_state">
95       <property name="intValue" stdset="0">
96         <number>0</number>
97       </property>
98     </widget>
99   </item>
100 </layout>
101 </widget>
102 <widget class="QGroupBox" name="groupBox_5">
103   <property name="geometry">
104     <rect>
105       <x>440</x>
106       <y>140</y>
107       <width>191</width>
108       <height>91</height>
109     </rect>
110   </property>
111   <property name="title">
112     <string>Stepper Motor Angle</string>
113   </property>
114   <property name="alignment">
115     <set>Qt::AlignmentFlag::AlignCenter</set>
116   </property>
117   <layout class="QHBoxLayout" name="horizontalLayout_4">
118     <item>
119       <widget class="QLCDNumber" name="stepper_angle_state"/>
120     </item>
121   </layout>
122 </widget>
123 <widget class="QGroupBox" name="groupBox_6">
124   <property name="geometry">
125     <rect>
126       <x>10</x>
127       <y>280</y>
128       <width>421</width>
129       <height>181</height>
130     </rect>
131   </property>
132   <property name="title">
133     <string>DC Motor</string>
134   </property>
135   <property name="alignment">
136     <set>Qt::AlignmentFlag::AlignLeading | Qt::AlignmentFlag::AlignLeft | Qt::AlignmentFlag::AlignVCenter</set>
137   </property>
138   <layout class="QVBoxLayout" name="verticalLayout">
139     <item>
140       <layout class="QHBoxLayout" name="horizontalLayout_10">
141         <item>

```

```

142     <widget class="QLabel" name="label_3">
143         <property name="text">
144             <string>Velocity (RPM)</string>
145         </property>
146     </widget>
147 </item>
148 <item>
149     <widget class=" QLineEdit" name="dc_vel_textbox"/>
150 </item>
151 </layout>
152 </item>
153 <item>
154     <layout class="QHBoxLayout" name="horizontalLayout_11">
155         <item>
156             <widget class="QLabel" name="label_4">
157                 <property name="text">
158                     <string>Position (degrees)</string>
159                 </property>
160             </widget>
161         </item>
162         <item>
163             <widget class=" QLineEdit" name="dc_angle_textbox"/>
164         </item>
165     </layout>
166 </item>
167 </layout>
168 </widget>
169 <widget class="QGroupBox" name="groupBox_7">
170     <property name="geometry">
171         <rect>
172             <x>440</x>
173             <y>250</y>
174             <width>191</width>
175             <height>91</height>
176         </rect>
177     </property>
178     <property name="title">
179         <string>DC Motor RPM</string>
180     </property>
181     <property name="alignment">
182         <set>Qt::AlignmentFlag::AlignCenter</set>
183     </property>
184     <layout class="QHBoxLayout" name="horizontalLayout_12">
185         <item>
186             <widget class="QLCDNumber" name="dc_rpm_state"/>
187         </item>
188     </layout>
189 </widget>
190 <widget class="QGroupBox" name="groupBox_13">
191     <property name="geometry">
192         <rect>
193             <x>440</x>
194             <y>350</y>
195             <width>191</width>
196             <height>91</height>

```

```

197     </rect>
198   </property>
199   <property name="title">
200     <string>DC Motor Angle</string>
201   </property>
202   <property name="alignment">
203     <set>Qt::AlignmentFlag::AlignCenter</set>
204   </property>
205   <layout class="QHBoxLayout" name="horizontalLayout_13">
206     <item>
207       <widget class="QLCDNumber" name="dc_angle_state"/>
208     </item>
209   </layout>
210 </widget>
211 <widget class="QGroupBox" name="groupBox_12">
212   <property name="geometry">
213     <rect>
214       <x>10</x>
215       <y>140</y>
216       <width>421</width>
217       <height>131</height>
218     </rect>
219   </property>
220   <property name="title">
221     <string>Stepper Motor</string>
222   </property>
223   <property name="alignment">
224     <set>Qt::AlignmentFlag::AlignLeading | Qt::AlignmentFlag::AlignLeft | Qt::AlignmentFlag::AlignVCenter</set>
225   </property>
226   <layout class="QVBoxLayout" name="verticalLayout_2">
227     <item alignment="Qt::AlignmentFlag::AlignHCenter">
228       <widget class="QRadioButton" name="stepper_mode_button">
229         <property name="text">
230           <string>Switch Mode</string>
231         </property>
232       </widget>
233     </item>
234     <item>
235       <layout class="QHBoxLayout" name="horizontalLayout_16">
236         <item>
237           <widget class="QLabel" name="label_2">
238             <property name="text">
239               <string>Angle (degrees)</string>
240             </property>
241           </widget>
242         </item>
243         <item>
244           <widget class="QSlider" name="stepper_slider">
245             <property name="tabletTracking">
246               <bool>false</bool>
247             </property>
248             <property name="acceptDrops">
249               <bool>false</bool>
250             </property>

```

```

251     <property name="autoFillBackground">
252         <bool>false</bool>
253     </property>
254     <property name="minimum">
255         <number>-180</number>
256     </property>
257     <property name="maximum">
258         <number>180</number>
259     </property>
260     <property name="value">
261         <number>0</number>
262     </property>
263     <property name="sliderPosition">
264         <number>0</number>
265     </property>
266     <property name="orientation">
267         <enum>Qt::Orientation::Horizontal</enum>
268     </property>
269     </widget>
270   </item>
271 </layout>
272 </item>
273 </layout>
274 </widget>
275 </widget>
276 <widget class="QGroupBox" name="groupBox_14">
277     <property name="geometry">
278         <rect>
279             <x>670</x>
280             <y>20</y>
281             <width>241</width>
282             <height>471</height>
283         </rect>
284     </property>
285     <property name="font">
286         <font>
287             <pointsize>13</pointsize>
288         </font>
289     </property>
290     <property name="title">
291         <string>Sensor Readings</string>
292     </property>
293     <property name="alignment">
294         <set>Qt::AlignmentFlag::AlignCenter</set>
295     </property>
296     <layout class="QVBoxLayout" name="verticalLayout_3">
297         <item>
298             <widget class="QGroupBox" name="groupBox_10">
299                 <property name="title">
300                     <string>Potentiometer Sensor</string>
301                 </property>
302                 <property name="alignment">
303                     <set>Qt::AlignmentFlag::AlignCenter</set>
304                 </property>
305                 <layout class="QHBoxLayout" name="horizontalLayout_7">

```

```

306     <item>
307         <widget class="QLCDNumber" name="potentiometer_sensor_state">
308             <property name="intValue" stdset="0">
309                 <number>0</number>
310             </property>
311         </widget>
312     </item>
313     </layout>
314 </widget>
315 </item>
316 <item>
317     <widget class="QGroupBox" name="groupBox_11">
318         <property name="title">
319             <string>IMU Sensor Pitch Angle</string>
320         </property>
321         <property name="alignment">
322             <set>Qt::AlignmentFlag::AlignCenter</set>
323         </property>
324         <layout class="QHBoxLayout" name="horizontalLayout_8">
325             <item>
326                 <widget class="QLCDNumber" name="IMU_sensor_state">
327                     <property name="intValue" stdset="0">
328                         <number>0</number>
329                     </property>
330                 </widget>
331             </item>
332         </layout>
333     </widget>
334 </item>
335 <item>
336     <widget class="QGroupBox" name="groupBox_8">
337         <property name="title">
338             <string>Temperature (Celcius)</string>
339         </property>
340         <property name="alignment">
341             <set>Qt::AlignmentFlag::AlignCenter</set>
342         </property>
343         <layout class="QHBoxLayout" name="horizontalLayout_5">
344             <item>
345                 <widget class="QLCDNumber" name="temperature_sensor_state">
346                     <property name="intValue" stdset="0">
347                         <number>0</number>
348                     </property>
349                 </widget>
350             </item>
351         </layout>
352     </widget>
353 </item>
354 <item>
355     <widget class="QGroupBox" name="groupBox_9">
356         <property name="title">
357             <string>Ultrasonic Sensor (cm)</string>
358         </property>
359         <property name="alignment">

```

```

360         <set>Qt::AlignmentFlag::AlignCenter</set>
361     </property>
362     <layout class="QHBoxLayout" name="horizontalLayout_6">
363         <item>
364             <widget class="QLCDNumber" name="ultrasonic_sensor_state"/>
365         </item>
366     </layout>
367     </widget>
368 </item>
369 </layout>
370 </widget>
371 <widget class="QGroupBox" name="groupBox_22">
372     <property name="geometry">
373         <rect>
374             <x>10</x>
375             <y>500</y>
376             <width>901</width>
377             <height>311</height>
378         </rect>
379     </property>
380     <property name="font">
381         <font>
382             <pointsize>13</pointsize>
383         </font>
384     </property>
385     <property name="title">
386         <string>Transfer Function</string>
387     </property>
388     <property name="alignment">
389         <set>Qt::AlignmentFlag::AlignCenter</set>
390     </property>
391     <layout class="QVBoxLayout" name="verticalLayout_4">
392         <item>
393             <widget class="QCustomPlot" name="transferFunctionPlot" native
394                 ="true"/>
395         </item>
396     </layout>
397 </widget>
398 <customwidgets>
399 <customwidget>
400     <class>QCustomPlot</class>
401     <extends>QWidget</extends>
402     <header>qcustomplot.h</header>
403     <container>1</container>
404 </customwidget>
405 </customwidgets>
406 <resources/>
407 <connections/>
408 </ui>

```

Listing 4: dialog.ui

5.2 Arduino

I was responsible for coding the skeleton template of the Arduino code. My teammates were responsible for implementing their respective controllers and callbacks after each // TODO: IMPLEMENT FUNCTION BELOW comment.

```
1 #include <Adafruit_Sensor.h>
2 #include <Adafruit_MPU6050.h>
3 #include <Wire.h>
4 #include <Servo.h>
5 #include <SharpIR.h>
6 #include <RoboClaw.h>
7 #include <AccelStepper.h>
8
9 #define SERIAL_PORT Serial
10
11 #define ADDRESS 0x80
12
13 // Motor & Encoder parameters
14 #define ENCODER_TICKS_PER_REV 3416
15 #define DEGREE_TO_TICKS (ENCODER_TICKS_PER_REV / 360.0)
16
17 // Position PID Tuning
18 #define Kp_pos 2.0
19 #define Ki_pos 0.5
20 #define Kd_pos 1.0
21
22 // Velocity PID Tuning
23 #define Kp_vel 1.0
24 #define Ki_vel 0.5
25 #define Kd_vel 0.25
26
27 // Maximum speed in encoder counts per second
28 #define MAX_QPPS 6718
29
30 // Pins
31 #define EN_StepperDriver 2
32 #define Stp_StepperDriver 3
33 #define Dir_StepperDriver 4
34 #define servoPin 6
35 #define PushButtonPin 7
36 #define PotentiometerPin A0
37 #define temperaturePin A1
38 #define ultrasonicPin A2
39 #define PushbuttonPin 7
40
41 AccelStepper stepper(AccelStepper::DRIVER, Stp_StepperDriver,
42 Dir_StepperDriver);
43
44 // GLOBAL VARIABLES
45 // Note: GUI will execute commands sent by Arduino once every 2
46 // intervals as it discards every other command due to serial
47 // buffer
48 unsigned long previousMillis = 0;
49 const unsigned long interval = 100;
```

```

48
49 Adafruit_MPU6050 mpu;
50 Servo servo;
51 RoboClaw roboclaw(&Serial1, 10000);
52
53 int servoAngle = 0;
54 int dc_motor_speed = 0;
55 int dc_motor_angle = 0;
56 double cm = 0.0;
57
58 int PotControlFlag = 0;
59 volatile int PotVal = 0;
60 volatile int globalStepperValue = 0;
61 volatile int globalStepperAngle = 0;
62
63 // Debounce variables
64 volatile unsigned long lastDebounceTime = 0;
65 const unsigned long debounceDelay = 100;
66 const int incrementServo = 30;
67
68 bool isMoving = false;
69 uint32_t targetPosition = 0;
70
71 void setup() {
72   SERIAL_PORT.begin(9600);
73   Serial1.begin(38400);
74   while (!SERIAL_PORT) {
75     // Wait for the serial port to be ready
76   }
77
78 // IMU
79 if (!mpu.begin()) {
80   Serial.println("Failed to find MPU6050 chip");
81   while (1)
82     ;
83 }
84
85 // Servo Motor
86 servo.attach(servoPin);
87 servo.write(0);
88
89 // Push Button
90 attachInterrupt(digitalPinToInterrupt(PushbuttonPin), pushISR,
91                 FALLING);
92
93 // DC Motor
94 unsigned long startTime = millis();
95
96 // Stepper
97 pinMode(EN_StepperDriver, OUTPUT);
98 digitalWrite(EN_StepperDriver, LOW); // enable stepper(s)
99 stepper.setMaxSpeed(2000);
100 stepper.setAcceleration(1000);
101 stepper.setSpeed(0);

```

```

102 // // Initialize RoboClaw
103 // if (!roboclaw.ReadError(ADDRESS)) {
104 //   Serial.println("RoboClaw connected successfully.");
105 // } else {
106 //   Serial.println("Error detected in RoboClaw!");
107 // }
108
109 roboclaw.SetM1VelocityPID(ADDRESS, Kp_vel, Ki_vel, Kd_vel,
110   MAX_QPPS);
111 roboclaw.SetM1PositionPID(ADDRESS, Kp_pos, Ki_pos, Kd_pos, Kp_vel
112   , Ki_vel, Kd_vel, MAX_QPPS);
113
114 roboclaw.SpeedM1(ADDRESS, 0);
115 // Serial.println("Motor stopped at startup.");
116
117
118
119 void loop() {
120   if (isMoving) {
121     uint32_t currentPos = roboclaw.ReadEncM1(ADDRESS);
122     if (abs((int32_t)(currentPos - targetPosition)) <= 15) { // 
123       Position tolerance
124       roboclaw.SpeedM1(ADDRESS, 0); // 
125       Stop motor
126       // Serial.println("Target position reached.");
127       isMoving = false; // Reset movement flag
128     }
129   }
130
131   // Check if data is available to read from the serial port
132   if (SERIAL_PORT.available() > 0) {
133     // Read the incoming string
134     String receivedString = SERIAL_PORT.readStringUntil(';');
135
136     // // Echo the string back to the serial port
137     // SERIAL_PORT.print("Arduino received command: ");
138     // SERIAL_PORT.print(receivedString);
139     // SERIAL_PORT.print(";");
140     // SERIAL_PORT.flush();
141
142     if (receivedString.length() > 1) {
143       char commandType = receivedString.charAt(0);
144
145       String valueString = receivedString.substring(1);
146       if (isNumeric(valueString)) {
147         int commandValue = valueString.toInt();
148         handleCommand(commandType, commandValue);
149       }
150     }
151
152     unsigned long currentMillis = millis();
153     if (currentMillis - previousMillis >= interval) {

```

```

153     previousMillis = currentMillis;
154     timerCallback();
155 }
156
157 stepperCallback();
158 }
159
160 void timerCallback() {
161     // SERIAL_PORT.print("Timer callback executed at: ");
162     // SERIAL_PORT.print(previousMillis);
163     // SERIAL_PORT.print(";");
164
165     // A "$" is used to indicate the serial port return is a command
166     String serialReturn;
167     serialReturn.concat("$");
168
169     int servoMotorState = servoMotorStateCallback();
170     serialReturn.concat(servoMotorState);
171     serialReturn.concat(",");
172
173     int stepperMotorState = stepperMotorStateCallback();
174     serialReturn.concat(stepperMotorState);
175     serialReturn.concat(",");
176
177     int velDCMotorState = velDCMotorStateCallback();
178     serialReturn.concat(velDCMotorState);
179     serialReturn.concat(",");
180
181     int angleDCMotorState = angleDCMotorStateCallback();
182     serialReturn.concat(angleDCMotorState);
183     serialReturn.concat(",");
184
185     int potentiometerSensorState = potentiometerSensorCallback();
186     serialReturn.concat(potentiometerSensorState);
187     serialReturn.concat(",");
188
189     double imuSensorState = imuSensorCallback();
190     serialReturn.concat(imuSensorState);
191     serialReturn.concat(",");
192
193     double temperatureSensorState = temperatureSensorCallback();
194     serialReturn.concat(temperatureSensorState);
195     serialReturn.concat(",");
196
197     int ultrasonicSensorState = ultrasonicSensorCallback();
198     serialReturn.concat(ultrasonicSensorState);
199     serialReturn.concat(",");
200
201     double electricalInput = analogRead(ultrasonicPin);
202     double transferFunctionState = transferFunctionCallback(
203         electricalInput);
204     double electricalVoltage = analogRead(ultrasonicPin) * (5.0 /
205         1023.0);
206
207     serialReturn.concat(electricalVoltage);

```

```

206     serialReturn.concat(":");
207     serialReturn.concat(transferFunctionState);
208
209     SERIAL_PORT.print(serialReturn);
210     SERIAL_PORT.print(";");
211 }
212
213
214 void handleCommand(char commandType, int value) {
215     switch (commandType) {
216         case 'R':
217             servoMotorController(value);
218             break;
219         case 'S':
220             stepperMotorController(value);
221             break;
222         case 'V':
223             velDCMotorController(value);
224             break;
225         case 'A':
226             angleDCMotorController(value);
227             break;
228         case 'B':
229             buttonStepperMotorController(value);
230             break;
231         default:
232             break;
233     }
234 }
235
236
237 bool isNumeric(String str) {
238     if (str.length() == 0) return false;
239
240     int startIndex = 0;
241
242     if (str[0] == '_') {
243         if (str.length() == 1) return false;
244         startIndex = 1;
245     }
246
247     for (unsigned int i = startIndex; i < str.length(); i++) {
248         if (!isDigit(str[i])) {
249             return false;
250         }
251     }
252
253     return true;
254 }
255
256
257 // CONTROLLER FUNCTIONS
258
259 void servoMotorController(int control) {
260     /*

```

```

261 INPUT: Integer in min/max range of 0 to 180 corresponding to
262     desired angle
263 OUTPUT: Void
264 */
265 SERIAL_PORT.print("Servo\u2022motor\u2022controller\u2022received\u2022command:\u2022");
266 SERIAL_PORT.print(control);
267 SERIAL_PORT.print(";\u2022");
268
269 // TODO: IMPLEMENT FUNCTION BELOW
270 servoAngle = control;
271 servo.write(servoAngle);
272 }
273
274 void stepperMotorController(int control) {
275 /*
276 INPUT: Integer in min/max range of -180 to 180 corresponding to
277     desired angle
278 OUTPUT: Void
279 */
280 SERIAL_PORT.print("Stepper\u2022motor\u2022controller\u2022received\u2022command:\u2022");
281 SERIAL_PORT.print(control);
282 SERIAL_PORT.print(";\u2022");
283
284 // TODO: IMPLEMENT FUNCTION BELOW
285 if (PotControlFlag == 0) {
286     globalStepperValue = map(control, -180, 180, -1600, 1600);
287     globalStepperAngle = control;
288 }
289 }
290
291 void velDCMotorController(int control) {
292 /*
293 INPUT: Integer in min/max range of 0 to 118 corresponding to
294     desired RPM
295 OUTPUT: Void
296 */
297 SERIAL_PORT.print("Velocity\u2022DC\u2022motor\u2022controller\u2022received\u2022command:
298     \u2022");
299 SERIAL_PORT.print(control);
300 SERIAL_PORT.print(";\u2022");
301
302 // TODO: IMPLEMENT FUNCTION BELOW
303 // Set motor velocity (positive for forward, negative for reverse
304 // )
305 int dc_motor_speed = (control / 118.0) * 100.0;
306 int encoderSpeed = (dc_motor_speed * MAX_QPPS) / 100; // Scale
307     input speed (user enters 0-100%)
308 roboclaw.SpeedM1(ADDRESS, encoderSpeed);
309 }
310
311 void angleDCMotorController(int control) {
312 /*

```

```

310     INPUT: Integer in min/max range of 0 to 360 corresponding to
311         desired angle
312     OUTPUT: Void
313     */
314
315     // SERIAL_PORT.print("Angle DC motor controller received command:
316         ");
316     // SERIAL_PORT.print(control);
317     // SERIAL_PORT.print(";");
318
319     // TODO: IMPLEMENT FUNCTION BELOW
320     uint32_t currentPos = roboclaw.ReadEncM1(ADDRESS);
321     targetPosition = currentPos + (control * DEGREE_TO_TICKS);
322
323     // Serial.print("Moving motor to position: ");
324     // Serial.println(targetPosition);
325
326     roboclaw.SpeedAccelDecelPositionM1(ADDRESS, 10000, MAX_QPPS,
327         10000, targetPosition, 0);
328
329     isMoving = true; // Set flag for movement tracking
330 }
331
332 void buttonStepperMotorController(int control) {
333     /*
334     INPUT: Boolean with 1 indicating GUI control and 1 indicating
335         potentiometer control
336     OUTPUT: Void
337     */
338
339     SERIAL_PORT.print("Button controller received command:");
340     SERIAL_PORT.print(control);
341     SERIAL_PORT.print(";");
342
343
344
345     // CALLBACK FUNCTIONS
346     int servoMotorStateCallback() {
347         /*
348         INPUT: Void
349         OUTPUT: Integer in min/max range of 0 to 180 corresponding to
350             servo motor angle
351         */
352
353         // TODO: IMPLEMENT FUNCTION BELOW
354         return servoAngle;
355     }
356
357     int stepperMotorStateCallback() {
358         /*
359         INPUT: Void
360         OUTPUT: Integer in min/max range of -180 to 180 corresponding to

```

```

        stepper motor angle
360    */
361
362    // TODO: IMPLEMENT FUNCTION BELOW
363
364    return globalStepperAngle;
365}
366
367 int velDCMotorStateCallback() {
368    /*
369     INPUT: Void
370     OUTPUT: Integer in min/max range of -118 to 118 corresponding to
371         DC motor RPM
372    */
373
374    // TODO: IMPLEMENT FUNCTION BELOW
375
376    int speed = roboclaw.ReadSpeedM1(ADDRESS);
377    speed = speed * 118 / 6718;
378
379    return speed;
380}
381
382 int angleDCMotorStateCallback() {
383    /*
384     INPUT: Void
385     OUTPUT: Integer in min/max range of -360 to 360 corresponding to
386         DC motor angle
387    */
388
389    // TODO: IMPLEMENT FUNCTION BELOW
390
391    int enc = roboclaw.ReadEncM1(ADDRESS);
392    int angle = (enc * 360) / 3416;
393    angle = angle % 360;
394
395    return angle;
396}
397
398 int potentiometerSensorCallback() {
399    /*
400     INPUT: Void
401     OUTPUT: Integer corresponding to potentiometer reading
402    */
403
404    // TODO: IMPLEMENT FUNCTION BELOW
405    PotVal = analogRead(PotentiometerPin);
406
407    if (PotControlFlag == 1) {
408        globalStepperValue = map(PotVal, 0, 1022, -1600, 1600);
409        globalStepperAngle = map(globalStepperValue, -1600, 1600, -180,
410                                180);
411    }
412
413    return PotVal;

```

```

411 }
412
413 double imuSensorCallback() {
414 /*
415 INPUT: Void
416 OUTPUT: Double corresponding to sensed IMU pitch reading
417 */
418
419 // TODO: IMPLEMENT FUNCTION BELOW
420 sensors_event_t a, g, temp;
421 mpu.getEvent(&a, &g, &temp);
422
423 // Pitch using accel data
424 double pitchAccel = atan2(a.acceleration.x, a.acceleration.z) *
425 180 / PI;
426
427 return pitchAccel;
428 }
429
430 double temperatureSensorCallback() {
431 /*
432 INPUT: Void
433 OUTPUT: Double corresponding to Temperature reading (degree
434 celsius)
435 */
436
437 // TODO: IMPLEMENT FUNCTION BELOW
438 int reading = analogRead(temperaturePin);
439 double voltage = reading * 3.3;
440 voltage /= 1024.0;
441
442 double temperatureC = (voltage - 1.0) * 100;
443
444 return temperatureC;
445 }
446
447 int ultrasonicSensorCallback() {
448 /*
449 INPUT: Void
450 OUTPUT: Integer corresponding to ultrasonic reading (cm)
451 */
452
453 // TODO: IMPLEMENT FUNCTION BELOW
454 cm = analogRead(ultrasonicPin);
455 cm = transferFunctionCallback(cm);
456
457 return int(cm);
458 }
459
460 // OTHER FUNCTIONS
461 void pushISR() {
462 /*
463 INPUT: Void
464 OUTPUT: Void
465 */

```

```

464
465 // TODO: IMPLEMENT FUNCTION BELOW
466 unsigned long currentTime = millis();
467 if ((currentTime - lastDebounceTime) > debounceDelay) {
468     servoAngle += incrementServo;
469     if (servoAngle > 180) {
470         servoAngle = 0;
471     }
472     lastDebounceTime = currentTime;
473     servo.write(servoAngle);
474 }
475
476
477 double transferFunctionCallback(double electricalInput) {
478 /*
479 INPUT: Double corresponding to electrical input voltage
480 OUTPUT: Double corresponding to ultrasonic reading (cm)
481 */
482
483 // TODO: IMPLEMENT FUNCTION BELOW
484 double val = electricalInput * 0.498 * 2.54;
485
486 return val;
487 }
488
489 void stepperCallback() {
490 /*
491 INPUT: Void
492 OUTPUT: Void
493 */
494
495 // TODO: IMPLEMENT FUNCTION BELOW
496 stepper.moveTo(globalStepperValue);
497 stepper.run();
498 }
```

Listing 5: arduino_gui.ino

6 Sensors and Motor Control Lab Quiz

6.1 Q1: ADXL335 Accelerometer

6.1.1 What is the sensor's range?

The minimum range of the sensor is $[-3g, 3g]$ (although it has a typical range of $[-3.6g, 3.6g]$), where g is the gravity constant.

6.1.2 What is the sensor's dynamic range?

The typical dynamic range is $7.2g$, the minimum dynamic range is $6g$.

6.1.3 What is the purpose of the capacitor C_{DC} on the LHS of the functional block diagram on p. 1? How does it achieve this?

From the block diagram, the C_{DC} capacitor is placed at the ends of the power supply. This means that it is used to regulate the voltage of the power input and filter out any high frequency noise in the voltage that may cause erroneous sensor readings. The capacitor can do this as it opposes sudden changes in voltage. This is because it can act as a current source/sink to counterbalance sudden decreases/increases in voltage.

6.1.4 Write an equation for the sensor's transfer function.

$$V_{out} = 1.5V + (0.3V/g) \times a$$

Where a is acceleration, g is the gravity constant, and V is volts.

6.1.5 What is the largest expected nonlinearity error in g?

The largest expected nonlinearity error is $\pm 0.3\%$ of $7.2g$, which turns out to be $\pm 0.0216g$ (i.e. ± 0.0216 in g).

6.1.6 What is the sensor's bandwidth for the X- and Y-axes?

The bandwidth is 1600 Hz.

6.1.7 How much noise do you expect in the X- and Y-axis sensor signals when your measurement bandwidth is 25 Hz?

The noise density is given as $150 \mu g/\sqrt{Hz}$. So at 25 Hz, the noise is $150 \times \sqrt{25 \times 1.6} = 948.68 \mu g$ (the 1.6 multiplier is the filter constant given on p. 11). Note that μg refers to gravity, and not grams.

6.1.8 If you didn't have the datasheet, how would you determine the RMS noise experimentally? State any assumptions and list the steps you would take.

The following steps could be taken:

1. Take several measurements experimentally.
2. Calculate the error (by subtracting the mean).
3. Take the square of the errors.
4. Calculate the mean.

5. Take the square root.

Some assumptions are:

- The noise source is stationary.
- The sensor is not being disturbed by anything other than the white noise.
- The measurement bandwidth is well-defined.
- The sensor is calibrated.

6.2 Q2: Signal Conditioning

6.2.1 Filtering: Name at least two problems you might have in using a moving average filter.

- A moving average filter lags behind the signal as it averages over several past data points, creating delays in real-time responsiveness.
- A moving average filter is not able to distinguish closely spaced frequencies as it averages over the data points. This may create problems as important signals could be lost.

6.2.2 Filtering: Name at least two problems you might have in using a median filter.

- A median filter is relatively expensive to compute as it requires sorting and storing previous values.
- A median filter is not able to effectively filter out continuously distributed noise such as Gaussian noise. This is because Gaussian noise introduces small fluctuations about the true value. Taking the median value in a window leaves much of the Gaussian noise intact as the filter is more effective at filtering out impulse noise.

6.2.3 Opamps: In the following questions, you want to calibrate a linear sensor using the circuit in Fig. 1 so that its output range is 0 to 5V. Identify in each case: 1) which of V₁ and V₂ will be the input voltage and which the reference voltage; 2) the values of the ratio R_f/R_i and the reference voltage. If the calibration can't be done with this circuit, explain why.

Referring to the Circuits lecture (Jan 16) Slide 75, the basic op-amp equation is given by:

$$V_{out} = (V_2 - V) \frac{R_f}{R_i} + V_2$$

When setting one voltage as reference and the other as input, the equation becomes:

$$\begin{aligned} \text{V}_1 (\text{V}_{ref}) \text{ reference, } \text{V}_2 (\text{V}_{in}) \text{ varies: } V_{out} &= V_{in} \underbrace{\left(1 + \frac{R_f}{R_i}\right)}_{\text{Gain}} - V_{ref} \underbrace{\left(\frac{R_f}{R_i}\right)}_{\text{Offset}} \\ \text{V}_2 (\text{V}_{ref}) \text{ reference, } \text{V}_1 (\text{V}_{in}) \text{ varies: } V_{out} &= -V_{in} \underbrace{\left(\frac{R_f}{R_i}\right)}_{\text{Gain}} + V_{ref} \underbrace{\left(1 + \frac{R_f}{R_i}\right)}_{\text{Offset}} \end{aligned}$$

6.2.4 Your uncalibrated sensor has a range of -1.5 to 1.0V (-1.5V should give a 0V output and 1.0V should give a 5V output).

1. Identify V_2 as input voltage and V_1 as the reference voltage.
2. Let $\frac{R_F}{R_i} = 1$ and $V_{ref} = V_1 = -3$.

We obtain:

$$\begin{aligned} V_{out} &= V_{in}\left(1 + \frac{R_f}{R_i}\right) - V_{ref}\left(\frac{R_f}{R_i}\right) \\ V_{out} &= V_2(1 + 1) - (-3)(1) \\ V_{out} &= 2V_2 + 3 \end{aligned}$$

Plugging in $V_2 = -1.5$ gives $V_{out} = 0$ and $V_2 = 1$ gives $V_{out} = 5$.

6.2.5 Your uncalibrated sensor has a range of -2.5 to 2.5V (-2.5V should give a 0V output and 2.5V should give a 5V output).

The calibration cannot be done with this circuit. This is because for the input-output equations to hold, we would require a gain of 1 and an offset of 2.5.

If we identify V_1 as the reference voltage and V_2 as the input voltage, to get the required gain and offset, we would need to let $\frac{R_F}{R_i} = 0$. However, the offset will always be 0, and the 2.5 required cannot be achieved.

If we identify V_2 as the reference voltage and V_1 as the input voltage, to get the required gain and offset, we would need to let $\frac{R_F}{R_i} = -1$, which is physically impossible as resistances must be non-negative.

6.3 Q3: Control

6.3.1 If you want to control a DC motor to go to a desired position, describe how to form a digital input for each of the PID (Proportional, Integral, Derivative) terms.

- **Proportional term:** Set a desired position and read the current motor position through a motor encoder. Deduct the current motor position from the desired position to get the error in position. Multiply this error by a predefined K_D term to get the proportional digital input.
- **Integral term:** Add the error term from the current time-step with the summed errors from previous time-steps. Multiply this summed error by a predefined K_I term to get the integral digital input.
- **Derivative term:** Subtract the previous step's motor position from the current step's motor position and divide by the time-step. Multiply this term by a predefined K_D term to get the derivative digital input.

Add the proportional, integral, and derivative terms together to get the full digital input, and send the input to the motor controller.

6.3.2 If the system you want to control is sluggish, which PID term(s) will you use and why?

I would increase the P (proportional) term as it makes the system react more aggressively to errors, thus reducing sluggishness and decreasing the sluggishness of the system.

6.3.3 After applying the control in the previous question, if the system still has significant steady-state error, which PID term(s) will you use and why?

I would increase the I (integral) term as it accumulates past errors when there is a steady-state error. This would cause the integral term to keep increasing until the error is driven to zero.

6.3.4 After applying the control in the previous question, if the system still has overshoot, which PID term(s) will you apply and why?

I would increase the D (derivative) term as it acts as a damping term and makes the response less aggressive while still maintaining a reasonably fast reaction.