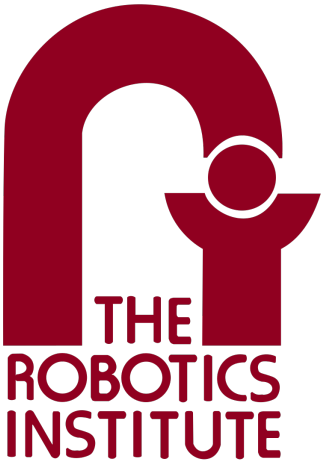
Individual Lab Report 1



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 14, 2025

Carnegie Mellon University
Robotics Institute

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.

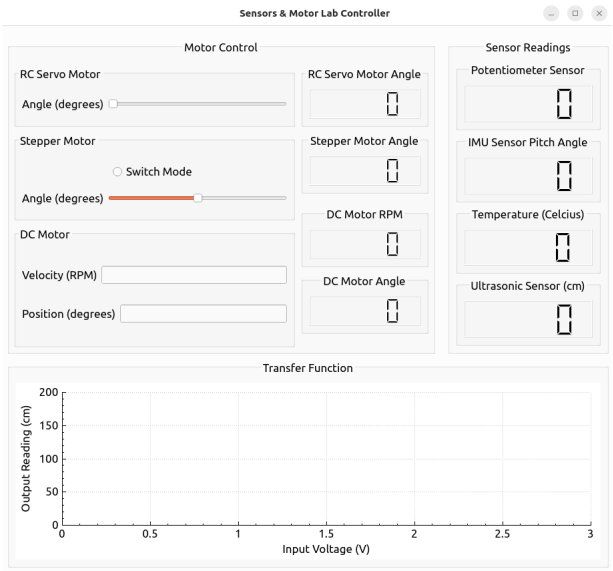


Figure 1: GUI Layout

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.

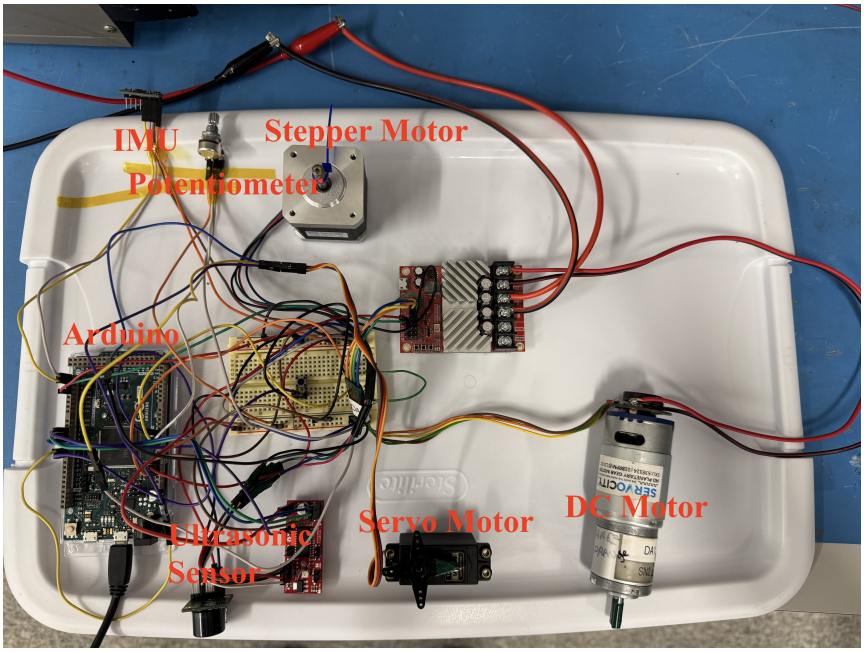


Figure 2: Sensors & Motor Control Lab Physical Setup

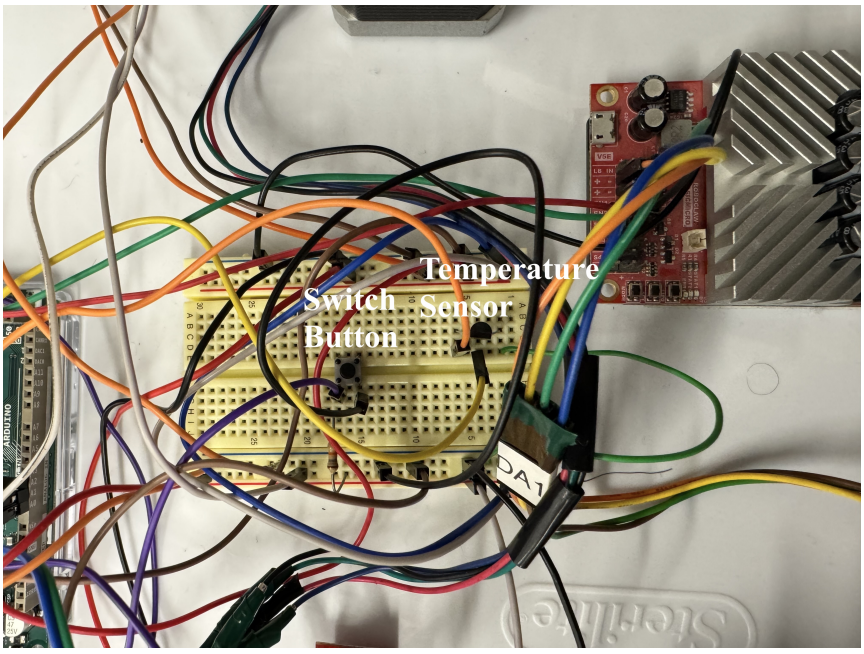


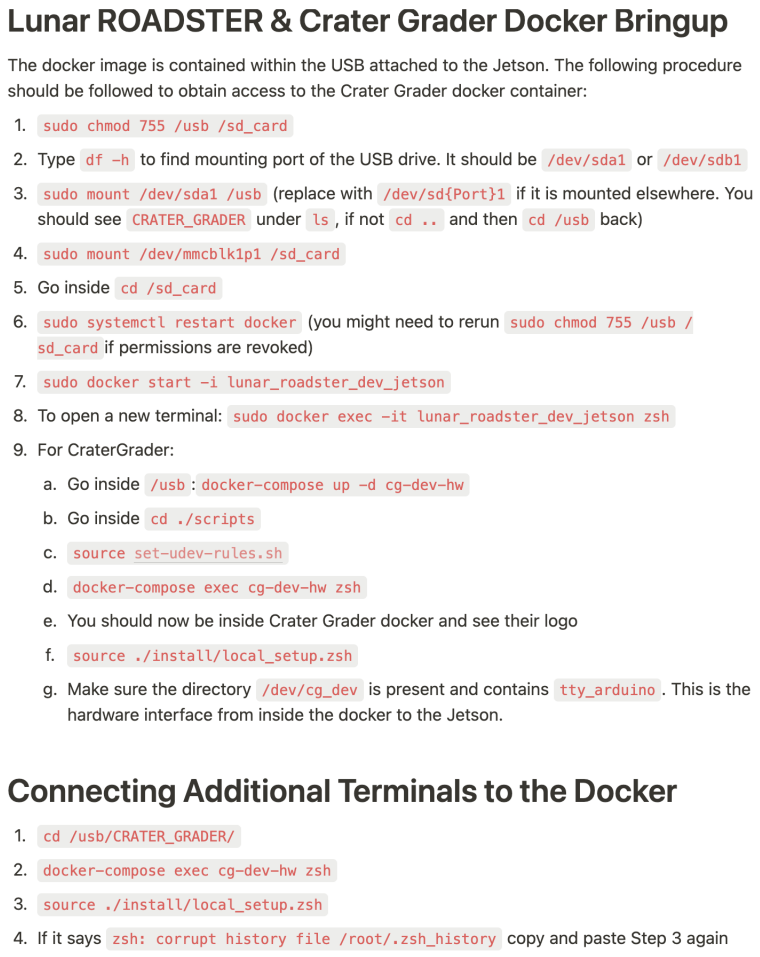
Figure 3: Close-up of Breadboard Setup

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 Circumnav 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.

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

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.

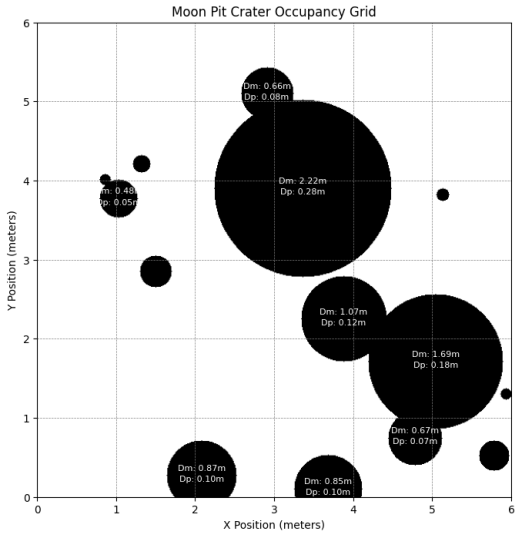


Figure 6: Moon Pit Crater Generation

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 collaberation 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 collaberation with Deepam), preliminary tests for teleoperation and grading with prototype dozer (in collaberation 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

```
#include "dialog.h"
#include "qcustomplot.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Dialog w;
    w.setFixedSize(940, 850);
    w.setWindowTitle("Sensors & Motor Lab Controller");
    w.show();
    return a.exec();
}
```

Listing 1: main.cpp

5.1.2 dialog.h

```
#ifndef DIALOG_H
#define DIALOG_H

#include <QDialog>
#include <QSerialPort>
#include <QSerialPortInfo>
#include <QDebug>
#include <QtWidgets>

QT_BEGIN_NAMESPACE
namespace Ui {
class Dialog;
}
QT_END_NAMESPACE

class Dialog : public QDialog
{
    Q_OBJECT

public:
    Dialog(QWidget *parent = nullptr);
    ~Dialog();

private slots:

    void updateArduino(QString);
```

```
27
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,
       which is under a GNU GPLv3.0 license:
3  https://github.com/vannevar-morgan/Qt-RGB-LED
4  https://github.com/vannevar-morgan/Qt-Temperature-Sensor
5  */
6
7  #include "dialog.h"
8  #include "ui_dialog.h"
```

```
9  #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     QCustomPlot *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     dataTimer->start(100);
49
50     connect(ui->dc_angle_textbox, &QLineEdit::returnPressed, this, &
           Dialog::onDCPosCommentSubmitted);
51     connect(ui->dc_vel_textbox, &QLineEdit::returnPressed, this, &
           Dialog::onDCVelCommentSubmitted);
52
53     /*
54     // Finds product and vendor ID of Arduino Due
55     qDebug() << "Number of available ports: " << QSerialPortInfo::
           availablePorts().length();
56     foreach(const QSerialPortInfo &serialPortInfo, QSerialPortInfo::
           availablePorts()){
57         qDebug() << "Has vendor ID: " << serialPortInfo.
               hasVendorIdentifier();
58         if(serialPortInfo.hasVendorIdentifier()){
```

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

```
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(
            serialData.toStdString());
122         serialData.clear();
123     }
124     else{
125         serialBuffer = "";
126         parsed_data = buffer_split[1];
127         // qDebug() << parsed_data << "\n";
128
129         // If the parsed data has a $ sign at index 0, then recognize
             it as a callback command
130         if (parsed_data.at(0) == '$') {
131             parseCallbackCommands(parsed_data);
132         }
133         else {
134             // qDebug() << parsed_data << "\n";
135         }
136     }
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.at
            (4);
180         QString imuSensorState = processedCommand_split.at(5);
181         QString temperatureSensorState = processedCommand_split.at(6)
            ;
182         QString ultrasonicSensorState = processedCommand_split.at(7);
183 
184         // Filtering for ultrasonic sensor
185         int ultrasonicReading = ultrasonicSensorState.toInt();
186         if (ultrasonicReadings.size() >= 5) {
187             ultrasonicReadings.erase(ultrasonicReadings.begin());
188         }
189         ultrasonicReadings.push_back(ultrasonicReading);
190         int filteredUltrasonicValue = filterMean(ultrasonicReadings);
191 
192         ui->potentiometer_sensor_state->display(
            potentiometerSensorState);
193         ui->IMU_sensor_state->display(imuSensorState);
194         ui->temperature_sensor_state->display(temperatureSensorState)
            ;
195         ui->ultrasonic_sensor_state->display(filteredUltrasonicValue)
            ;
196         // ui->ultrasonic_sensor_state->display(ultrasonicSensorState
            );
197 
198         // Transfer function
199         QString transferFunctionState = processedCommand_split.at(8);
200         QStringList transferFunctionState_split =
            transferFunctionState.split(":");
201 
202         QString transferFunctionInputString =
            transferFunctionState_split.at(0);
```

```
203         QString transferFunctionOutputString =
                transferFunctionState_split.at(1);
204
205         transferFunctionInput = transferFunctionInputString.toDouble
                ();
206         transferFunctionOutput = transferFunctionOutputString.
                toDouble();
207     }
208 }
209
210
211 void Dialog::on_servo_slider_valueChanged(int value)
212 {
213     QString command = QString("R%1;").arg(value);
214     qDebug() << "Slider value:" << value << "Command:" << command;
215     updateArduino(command);
216 }
217
218
219 void Dialog::on_stepper_slider_valueChanged(int value)
220 {
221     QString command = QString("S%1;").arg(value);
222     qDebug() << "Slider value:" << value << "Command:" << command;
223     updateArduino(command);
224 }
225
226 void Dialog::on_stepper_mode_button_toggled(bool checked)
227 {
228     QString command = QString("B%1;").arg(checked);
229     qDebug() << "Button value:" << checked << "Command:" << command;
230     updateArduino(command);
231 }
232
233 void Dialog::onDCVelCommentSubmitted()
234 {
235     QString comment = ui->dc_vel_textbox->text();
236
237     QString command = QString("V%1;").arg(comment);
238     qDebug() << "Text box value:" << comment << "Command:" << command
            ;
239     updateArduino(command);
240
241     ui->dc_vel_textbox->clear();
242 }
243
244 void Dialog::onDCPosCommentSubmitted()
245 {
246     QString comment = ui->dc_angle_textbox->text();
247
248     QString command = QString("A%1;").arg(comment);
249     qDebug() << "Text box value:" << comment << "Command:" << command
            ;
250     updateArduino(command);
251
252     ui->dc_angle_textbox->clear();
```

```
}

int Dialog::filterMean(std::vector<int>& arr) {
    if(arr.empty()) {
        return 0;
    }

    int sum = 0;

    for (size_t i = 1; i < arr.size(); i++) {
        sum += arr[i];
    }

    return sum / arr.size();
}
```

Listing 3: dialog.cpp

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>Dialog</class>
 <widget class="QDialog" name="Dialog">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>1280</width>
    <height>1024</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Dialog</string>
  </property>
  <property name="autoFillBackground">
   <bool>false</bool>
  </property>
  <widget class="QGroupBox" name="groupBox">
   <property name="geometry">
    <rect>
     <x>10</x>
     <y>20</y>
     <width>641</width>
     <height>471</height>
    </rect>
   </property>
   <property name="font">
    <font>
     <pointsize>13</pointsize>
    </font>
   </property>
   <property name="title">
    <string>Motor Control</string>
   </property>
   <property name="alignment">
    <set>Qt::AlignmentFlag::AlignCenter</set>
   </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">
```

```
     <item>
      <widget class="QLCDNumber" name="servo_angle_state">
       <property name="intValue" stdset="0">
        <number>0</number>
       </property>
      </widget>
     </item>
    </layout>
   </widget>
   <widget class="QGroupBox" name="groupBox_5">
    <property name="geometry">
     <rect>
      <x>440</x>
      <y>140</y>
      <width>191</width>
      <height>91</height>
     </rect>
    </property>
    <property name="title">
     <string>Stepper Motor Angle</string>
    </property>
    <property name="alignment">
     <set>Qt::AlignmentFlag::AlignCenter</set>
    </property>
    <layout class="QHBoxLayout" name="horizontalLayout_4">
     <item>
      <widget class="QLCDNumber" name="stepper_angle_state"/>
     </item>
    </layout>
   </widget>
   <widget class="QGroupBox" name="groupBox_6">
    <property name="geometry">
     <rect>
      <x>10</x>
      <y>280</y>
      <width>421</width>
      <height>181</height>
     </rect>
    </property>
    <property name="title">
     <string>DC Motor</string>
    </property>
    <property name="alignment">
     <set>Qt::AlignmentFlag::AlignLeading|Qt::AlignmentFlag::
        AlignLeft|Qt::AlignmentFlag::AlignVCenter</set>
    </property>
    <layout class="QVBoxLayout" name="verticalLayout">
     <item>
      <layout class="QHBoxLayout" name="horizontalLayout_10">
       <item>
        <widget class="QLabel" name="label_3">
         <property name="text">
          <string>Velocity (RPM)</string>
         </property>
        </widget>
```

```
      </item>
      <item>
       <widget class="QLineEdit" name="dc_vel_textbox"/>
      </item>
     </layout>
    </item>
    <item>
     <layout class="QHBoxLayout" name="horizontalLayout_11">
      <item>
       <widget class="QLabel" name="label_4">
        <property name="text">
         <string>Position (degrees)</string>
        </property>
       </widget>
      </item>
      <item>
       <widget class="QLineEdit" name="dc_angle_textbox"/>
      </item>
     </layout>
    </item>
   </layout>
  </widget>
  <widget class="QGroupBox" name="groupBox_7">
   <property name="geometry">
    <rect>
     <x>440</x>
     <y>250</y>
     <width>191</width>
     <height>91</height>
    </rect>
   </property>
   <property name="title">
    <string>DC Motor RPM</string>
   </property>
   <property name="alignment">
    <set>Qt::AlignmentFlag::AlignCenter</set>
   </property>
   <layout class="QHBoxLayout" name="horizontalLayout_12">
    <item>
     <widget class="QLCDNumber" name="dc_rpm_state"/>
    </item>
   </layout>
  </widget>
  <widget class="QGroupBox" name="groupBox_13">
   <property name="geometry">
    <rect>
     <x>440</x>
     <y>350</y>
     <width>191</width>
     <height>91</height>
    </rect>
   </property>
   <property name="title">
    <string>DC Motor Angle</string>
   </property>
```

```
    <property name="alignment">
     <set>Qt::AlignmentFlag::AlignCenter</set>
    </property>
    <layout class="QHBoxLayout" name="horizontalLayout_13">
     <item>
      <widget class="QLCDNumber" name="dc_angle_state"/>
     </item>
    </layout>
   </widget>
   <widget class="QGroupBox" name="groupBox_12">
    <property name="geometry">
     <rect>
      <x>10</x>
      <y>140</y>
      <width>421</width>
      <height>131</height>
     </rect>
    </property>
    <property name="title">
     <string>Stepper Motor</string>
    </property>
    <property name="alignment">
     <set>Qt::AlignmentFlag::AlignLeading|Qt::AlignmentFlag::
        AlignLeft|Qt::AlignmentFlag::AlignVCenter</set>
    </property>
    <layout class="QVBoxLayout" name="verticalLayout_2">
     <item alignment="Qt::AlignmentFlag::AlignHCenter">
      <widget class="QRadioButton" name="stepper_mode_button">
       <property name="text">
        <string>Switch Mode</string>
       </property>
      </widget>
     </item>
     <item>
      <layout class="QHBoxLayout" name="horizontalLayout_16">
       <item>
        <widget class="QLabel" name="label_2">
         <property name="text">
          <string>Angle (degrees)</string>
         </property>
        </widget>
       </item>
       <item>
        <widget class="QSlider" name="stepper_slider">
         <property name="tabletTracking">
          <bool>false</bool>
         </property>
         <property name="acceptDrops">
          <bool>false</bool>
         </property>
         <property name="autoFillBackground">
          <bool>false</bool>
         </property>
         <property name="minimum">
          <number>-180</number>
```

```
          </property>
          <property name="maximum">
           <number>180</number>
          </property>
          <property name="value">
           <number>0</number>
          </property>
          <property name="sliderPosition">
           <number>0</number>
          </property>
          <property name="orientation">
           <enum>Qt::Orientation::Horizontal</enum>
          </property>
         </widget>
        </item>
       </layout>
      </item>
     </layout>
    </widget>
   </widget>
   <widget class="QGroupBox" name="groupBox_14">
    <property name="geometry">
     <rect>
      <x>670</x>
      <y>20</y>
      <width>241</width>
      <height>471</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <pointsize>13</pointsize>
     </font>
    </property>
    <property name="title">
     <string>Sensor Readings</string>
    </property>
    <property name="alignment">
     <set>Qt::AlignmentFlag::AlignCenter</set>
    </property>
    <layout class="QVBoxLayout" name="verticalLayout_3">
     <item>
      <widget class="QGroupBox" name="groupBox_10">
       <property name="title">
        <string>Potentiometer Sensor</string>
       </property>
       <property name="alignment">
        <set>Qt::AlignmentFlag::AlignCenter</set>
       </property>
       <layout class="QHBoxLayout" name="horizontalLayout_7">
        <item>
         <widget class="QLCDNumber" name="potentiometer_sensor_state">
          <property name="intValue" stdset="0">
           <number>0</number>
          </property>
```

```
        </widget>
       </item>
      </layout>
     </widget>
    </item>
    <item>
     <widget class="QGroupBox" name="groupBox_11">
      <property name="title">
       <string>IMU Sensor Pitch Angle</string>
      </property>
      <property name="alignment">
       <set>Qt::AlignmentFlag::AlignCenter</set>
      </property>
      <layout class="QHBoxLayout" name="horizontalLayout_8">
       <item>
        <widget class="QLCDNumber" name="IMU_sensor_state">
         <property name="intValue" stdset="0">
          <number>0</number>
         </property>
        </widget>
       </item>
      </layout>
     </widget>
    </item>
    <item>
     <widget class="QGroupBox" name="groupBox_8">
      <property name="title">
       <string>Temperature (Celcius)</string>
      </property>
      <property name="alignment">
       <set>Qt::AlignmentFlag::AlignCenter</set>
      </property>
      <layout class="QHBoxLayout" name="horizontalLayout_5">
       <item>
        <widget class="QLCDNumber" name="temperature_sensor_state">
         <property name="intValue" stdset="0">
          <number>0</number>
         </property>
        </widget>
       </item>
      </layout>
     </widget>
    </item>
    <item>
     <widget class="QGroupBox" name="groupBox_9">
      <property name="title">
       <string>Ultrasonic Sensor (cm)</string>
      </property>
      <property name="alignment">
       <set>Qt::AlignmentFlag::AlignCenter</set>
      </property>
      <layout class="QHBoxLayout" name="horizontalLayout_6">
       <item>
        <widget class="QLCDNumber" name="ultrasonic_sensor_state"/>
       </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="
             true"/>
394       </item>
395      </layout>
396     </widget>
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>
```

```
#include <Wire.h>
#include <Servo.h>
#include <SharpIR.h>
#include <RoboClaw.h>
#include <AccelStepper.h>

#define SERIAL_PORT Serial

#define ADDRESS 0x80

// Motor & Encoder parameters
#define ENCODER_TICKS_PER_REV 3416
#define DEGREE_TO_TICKS (ENCODER_TICKS_PER_REV / 360.0)

// Position PID Tuning
#define Kp_pos 2.0
#define Ki_pos 0.5
#define Kd_pos 1.0

// Velocity PID Tuning
#define Kp_vel 1.0
#define Ki_vel 0.5
#define Kd_vel 0.25

// Maximum speed in encoder counts per second
#define MAX_QPPS 6718

// Pins
#define EN_StepperDriver 2
#define Stp_StepperDriver 3
#define Dir_StepperDriver 4
#define servoPin 6
#define PushButtonPin 7
#define PotentiometerPin A0
#define temperaturePin A1
#define ultrasonicPin A2
#define PushbuttonPin 7

AccelStepper stepper(AccelStepper::DRIVER, Stp_StepperDriver,
   Dir_StepperDriver);


// GLOBAL VARIABLES
// Note: GUI will execute commands sent by Arduino once every 2
   intervals as it discards every other command due to serial buffer
unsigned long previousMillis = 0;
const unsigned long interval = 100;

Adafruit_MPU6050 mpu;
Servo servo;
RoboClaw roboclaw(&Serial1, 10000);

int servoAngle = 0;
int dc_motor_speed = 0;
int dc_motor_angle = 0;
```

```
double cm = 0.0;

int PotControlFlag = 0;
volatile int PotVal = 0;
volatile int globalStepperValue = 0;
volatile int globalStepperAngle = 0;

// Debounce variables
volatile unsigned long lastDebounceTime = 0;
const unsigned long debounceDelay = 100;
const int incrementServo = 30;

bool isMoving = false;
uint32_t targetPosition = 0;

void setup() {
  SERIAL_PORT.begin(9600);
  Serial1.begin(38400);
  while (!SERIAL_PORT) {
    // Wait for the serial port to be ready
  }

  // IMU
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1)
      ;
  }

  // Servo Motor
  servo.attach(servoPin);
  servo.write(0);

  // Push Button
  attachInterrupt(digitalPinToInterrupt(PushbuttonPin), pushISR,
     FALLING);

  // DC Motor
  unsigned long startTime = millis();

  // Stepper
  pinMode(EN_StepperDriver, OUTPUT);
  digitalWrite(EN_StepperDriver, LOW);  // enable stepper(s)
  stepper.setMaxSpeed(2000);
  stepper.setAcceleration(1000);
  stepper.setSpeed(0);

  // // Initialize RoboClaw
  // if (!roboclaw.ReadError(ADDRESS)) {
  //   Serial.println("RoboClaw connected successfully.");
  // } else {
  //   Serial.println("Error detected in RoboClaw!");
  // }

  roboclaw.SetM1VelocityPID(ADDRESS, Kp_vel, Ki_vel, Kd_vel, MAX_QPPS
```

```
    );
  roboclaw.SetM1PositionPID(ADDRESS, Kp_pos, Ki_pos, Kd_pos, Kp_vel,
     Ki_vel, Kd_vel, MAX_QPPS);

  roboclaw.SpeedM1(ADDRESS, 0);
  // Serial.println("Motor stopped at startup.");

  SERIAL_PORT.print("Arduino Due Serial is ready!;");
}


void loop() {
  if (isMoving) {
    uint32_t currentPos = roboclaw.ReadEncM1(ADDRESS);
    if (abs((int32_t)(currentPos - targetPosition)) <= 15) {  //
       Position tolerance
      roboclaw.SpeedM1(ADDRESS, 0);                           // Stop
          motor
      // Serial.println("Target position reached.");
      isMoving = false;  // Reset movement flag
    }
  }

  // Check if data is available to read from the serial port
  if (SERIAL_PORT.available() > 0) {
    // Read the incoming string
    String receivedString = SERIAL_PORT.readStringUntil(';');

    // // Echo the string back to the serial port
    // SERIAL_PORT.print("Arduino received command: ");
    // SERIAL_PORT.print(receivedString);
    // SERIAL_PORT.print(";");
    // SERIAL_PORT.flush();

    if (receivedString.length() > 1) {
      char commandType = receivedString.charAt(0);

      String valueString = receivedString.substring(1);
      if (isNumeric(valueString)) {
        int commandValue = valueString.toInt();
        handleCommand(commandType, commandValue);
      }
    }
  }

  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    timerCallback();
  }

  stepperCallback();
}

void timerCallback() {
```

```
  // SERIAL_PORT.print("Timer callback executed at: ");
  // SERIAL_PORT.print(previousMillis);
  // SERIAL_PORT.print(";");

  // A "$" is used to indicate the serial port return is a command
  String serialReturn;
  serialReturn.concat("$");

  int servoMotorState = servoMotorStateCallback();
  serialReturn.concat(servoMotorState);
  serialReturn.concat(",");

  int stepperMotorState = stepperMotorStateCallback();
  serialReturn.concat(stepperMotorState);
  serialReturn.concat(",");

  int velDCMotorState = velDCMotorStateCallback();
  serialReturn.concat(velDCMotorState);
  serialReturn.concat(",");

  int angleDCMotorState = angleDCMotorStateCallback();
  serialReturn.concat(angleDCMotorState);
  serialReturn.concat(",");

  int potentiometerSensorState = potentiometerSensorCallback();
  serialReturn.concat(potentiometerSensorState);
  serialReturn.concat(",");

  double imuSensorState = imuSensorCallback();
  serialReturn.concat(imuSensorState);
  serialReturn.concat(",");

  double temperatureSensorState = temperatureSensorCallback();
  serialReturn.concat(temperatureSensorState);
  serialReturn.concat(",");

  int ultrasonicSensorState = ultrasonicSensorCallback();
  serialReturn.concat(ultrasonicSensorState);
  serialReturn.concat(",");

  double electricalInput = analogRead(ultrasonicPin);
  double transferFunctionState = transferFunctionCallback(
     electricalInput);
  double electricalVoltage = analogRead(ultrasonicPin) * (5.0 /
     1023.0);

  serialReturn.concat(electricalVoltage);
  serialReturn.concat(":");
  serialReturn.concat(transferFunctionState);

  SERIAL_PORT.print(serialReturn);
  SERIAL_PORT.print(";");
}
```

```
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
       desired angle
262   OUTPUT: Void
263   */
264
265   SERIAL_PORT.print("Servo motor controller received command: ");
266   SERIAL_PORT.print(control);
267   SERIAL_PORT.print(";");
```

```
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
         desired angle
277   OUTPUT: Void
278   */
279
280   SERIAL_PORT.print("Stepper motor controller received command: ");
281   SERIAL_PORT.print(control);
282   SERIAL_PORT.print(";");
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
         desired RPM
294   OUTPUT: Void
295   */
296
297   SERIAL_PORT.print("Velocity DC motor controller received command: "
         );
298   SERIAL_PORT.print(control);
299   SERIAL_PORT.print(";");
300
301   // TODO: IMPLEMENT FUNCTION BELOW
302   // Set motor velocity (positive for forward, negative for reverse)
303   int dc_motor_speed = (control / 118.0) * 100.0;
304   int encoderSpeed = (dc_motor_speed * MAX_QPPS) / 100;  // Scale
         input speed (user enters 0-100%)
305   roboclaw.SpeedM1(ADDRESS, encoderSpeed);
306 }
307
308 void angleDCMotorController(int control) {
309   /*
310   INPUT: Integer in min/max range of 0 to 360 corresponding to
         desired angle
311   OUTPUT: Void
312   */
313
314   // SERIAL_PORT.print("Angle DC motor controller received command:
         ");
315   // SERIAL_PORT.print(control);
316   // SERIAL_PORT.print(";");
```

```
317
318   // TODO: IMPLEMENT FUNCTION BELOW
319   uint32_t currentPos = roboclaw.ReadEncM1(ADDRESS);
320   targetPosition = currentPos + (control * DEGREE_TO_TICKS);
321
322   // Serial.print("Moving motor to position: ");
323   // Serial.println(targetPosition);
324
325   roboclaw.SpeedAccelDeccelPositionM1(ADDRESS, 10000, MAX_QPPS,
          10000, targetPosition, 0);
326
327   isMoving = true;  // Set flag for movement tracking
328 }
329
330 void buttonStepperMotorController(int control) {
331   /*
332   INPUT: Boolean with 1 indicating GUI control and 1 indicating
          potentiometer control
333   OUTPUT: Void
334   */
335
336   SERIAL_PORT.print("Button␣controller␣received␣command:␣");
337   SERIAL_PORT.print(control);
338   SERIAL_PORT.print(";");
339
340   // TODO: IMPLEMENT FUNCTION BELOW
341   PotControlFlag = control;
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 servo
          motor angle
350   */
351
352   // TODO: IMPLEMENT FUNCTION BELOW
353   return servoAngle;
354 }
355
356 int stepperMotorStateCallback() {
357   /*
358   INPUT: Void
359   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 DC
          motor RPM
371   */
372
373   // TODO: IMPLEMENT FUNCTION BELOW
374
375   int speed = roboclaw.ReadSpeedM1(ADDRESS);
376   speed = speed * 118 / 6718;
377
378   return speed;
379 }
380
381 int angleDCMotorStateCallback() {
382   /*
383   INPUT: Void
384   OUTPUT: Integer in min/max range of -360 to 360 corresponding to DC
          motor angle
385   */
386
387   // TODO: IMPLEMENT FUNCTION BELOW
388
389   int enc = roboclaw.ReadEncM1(ADDRESS);
390   int angle = (enc * 360) / 3416;
391   angle = angle % 360;
392
393   return angle;
394 }
395
396 int potentiometerSensorCallback() {
397   /*
398   INPUT: Void
399   OUTPUT: Integer corresponding to potentiometer reading
400   */
401
402   // TODO: IMPLEMENT FUNCTION BELOW
403   PotVal = analogRead(PotentiometerPin);
404
405   if (PotControlFlag == 1) {
406     globalStepperValue = map(PotVal, 0, 1022, -1600, 1600);
407     globalStepperAngle = map(globalStepperValue, -1600, 1600, -180,
          180);
408   }
409
410   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
```

```
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);

  // Pitch using accel data
  double pitchAccel = atan2(a.acceleration.x, a.acceleration.z) * 180
      / PI;

  return pitchAccel;
}

double temperatureSensorCallback() {
  /*
  INPUT: Void
  OUTPUT: Double corresponding to Temperature reading (degree celsius
     )
  */

  // TODO: IMPLEMENT FUNCTION BELOW
  int reading = analogRead(temperaturePin);
  double voltage = reading * 3.3;
  voltage /= 1024.0;

  double temperatureC = (voltage - 1.0) * 100;

  return temperatureC;
}

int ultrasonicSensorCallback() {
  /*
  INPUT: Void
  OUTPUT: Integer corresponding to ultrasonic reading (cm)
  */

  // TODO: IMPLEMENT FUNCTION BELOW
  cm = analogRead(ultrasonicPin);
  cm = transferFunctionCallback(cm);

  return int(cm);
}

// OTHER FUNCTIONS
void pushISR() {
  /*
  INPUT: Void
  OUTPUT: Void
  */

  // TODO: IMPLEMENT FUNCTION BELOW
  unsigned long currentTime = millis();
  if ((currentTime - lastDebounceTime) > debounceDelay) {
    servoAngle += incrementServo;
    if (servoAngle > 180) {
      servoAngle = 0;
    }
    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 Rf/Ri 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:

Gain Offset

V_1 (V_{ref}) reference, V_2 (V_{in}) varies: $V_{out} = V_{in}\left(1 + \frac{R_f}{R_i}\right) - V_{ref}\left(\frac{R_f}{R_i}\right)$

V_2 (V_{ref}) reference, V_1 (V_{in}) varies: $V_{out} = -V_{in}\left(\frac{R_f}{R_i}\right) + V_{ref}\left(1 + \frac{R_f}{R_i}\right)$

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:

$$V_{out} = V_{in}(1 + \frac{R_f}{R_i}) - V_{ref}(\frac{R_f}{R_i})$$
$$V_{out} = V_2(1 + 1) - (-3)(1)$$
$$V_{out} = 2V_2 + 3$$

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 has it acts as a damping term and makes the response less aggressive while still maintaining a reasonably fast reaction.