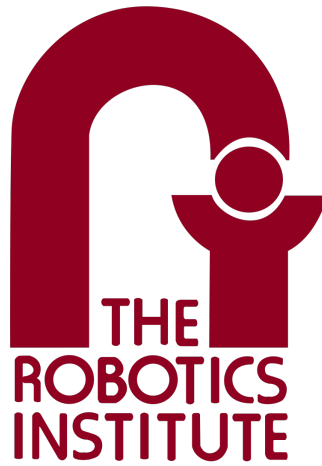

Individual Lab report



Lunar ROADSTER

Team I

Author: **Ayapilla Sri Bhaswanth**
Andrew ID: bayapill
E-mail: bayapill@andrew.cmu.edu

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

February 7, 2025

Contents

- 1 Individual Progress** **2**
 - 1.1 Sensors and Motors Lab 2
 - 1.1.1 Ultrasonic Range Finder 2
 - 1.1.2 Pushbutton 3
 - 1.1.3 Full Circuit 4
 - 1.2 MRSD Capstone Project: Lunar ROADSTER 5

- 2 Challenges** **7**
 - 2.1 Sensors and Motors Lab 7
 - 2.2 MRSD Capstone Project: Lunar ROADSTER 7

- 3 Teamwork** **7**
 - 3.1 Sensors and Motors Lab 7
 - 3.2 MRSD Capstone Project: Lunar ROADSTER 8

- 4 Plans** **8**
 - 4.1 Sensors and Motors Lab 8
 - 4.2 MRSD Capstone Project: Lunar ROADSTER 8

- 5 Code** **9**

1 Individual Progress

1.1 Sensors and Motors Lab

My responsibility for this lab was to develop the circuit and write code for the MB1000 LV-MaxSonar-EZ0 Ultrasonic Range Finder, implementing a mean filter and transfer function, and also to implement a pushbutton to control the servo motor.

1.1.1 Ultrasonic Range Finder

The Ultrasonic sensor has 7 pins, but I used only the following 3 pins:

- **Pin 3-AN:** Outputs analog voltage with a scaling factor of ($V_{cc}/512$) per inch. A supply of $5V$ yields $\sim 9.8mV/in.$ and $3.3V$ yields $\sim 6.4mV/in.$ The output is buffered and corresponds to the most recent range data.
- **Pin 6-+5V:** V_{cc} – Operates on $2.5V - 5.5V$. Recommended current capability of $3mA$ for $5V$, and $2mA$ for $3V$.
- **Pin 7-GND:** Return for the DC power supply

The sensor gives a range between 6-inches ($\sim 15.24cm$) to 20-inches ($\sim 645.16cm$), with 1-inch resolution ($\sim 2.54cm$).

The above information is taken from the [datasheet](#).

I utilized the analog pin of the ultrasonic sensor and connected it to an analog pin on the Arduino board. Doing a simple `analogRead` gives me the analog value, which can be converted into a physically interpretable distance value using the following transfer function:

$$\text{distance} = \text{analogVal} * 0.498 * 2.54$$

The above transfer function returns the distance in centimeters and was obtained by conducting experiments myself.

To get more stable readings, I decided to use a mean filter. It's a simple and fast way to smooth out the data since it doesn't require sorting like some other filtering methods..

```
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();
}
```

1.1.2 Pushbutton

I implemented a pushbutton for servo motor control. It essentially acts as an interrupt, overwriting all other functions to increment the servo angle by 30 degrees every time it is pressed. Once the servo reaches its maximum limit of 180 degrees, pressing the pushbutton again will move the servo to its 0 degree position.

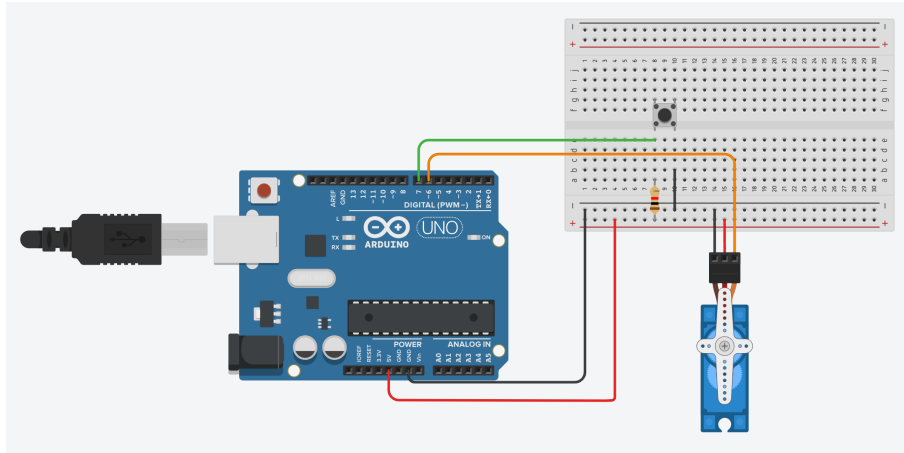


Figure 1: Pushbutton for Servo Control

```
void setup(){
  // Push Button
  attachInterrupt(digitalPinToInterrupt(PushbuttonPin), pushISR, FALLING);
}

void pushISR() {
  unsigned long currentTime = millis();
  if ((currentTime - lastDebounceTime) > debounceDelay) {
    servoAngle += incrementServo;
    if (servoAngle > 180) {
      servoAngle = 0;
    }
    lastDebounceTime = currentTime;
    servo.write(servoAngle);
  }
}
```

1.1.3 Full Circuit

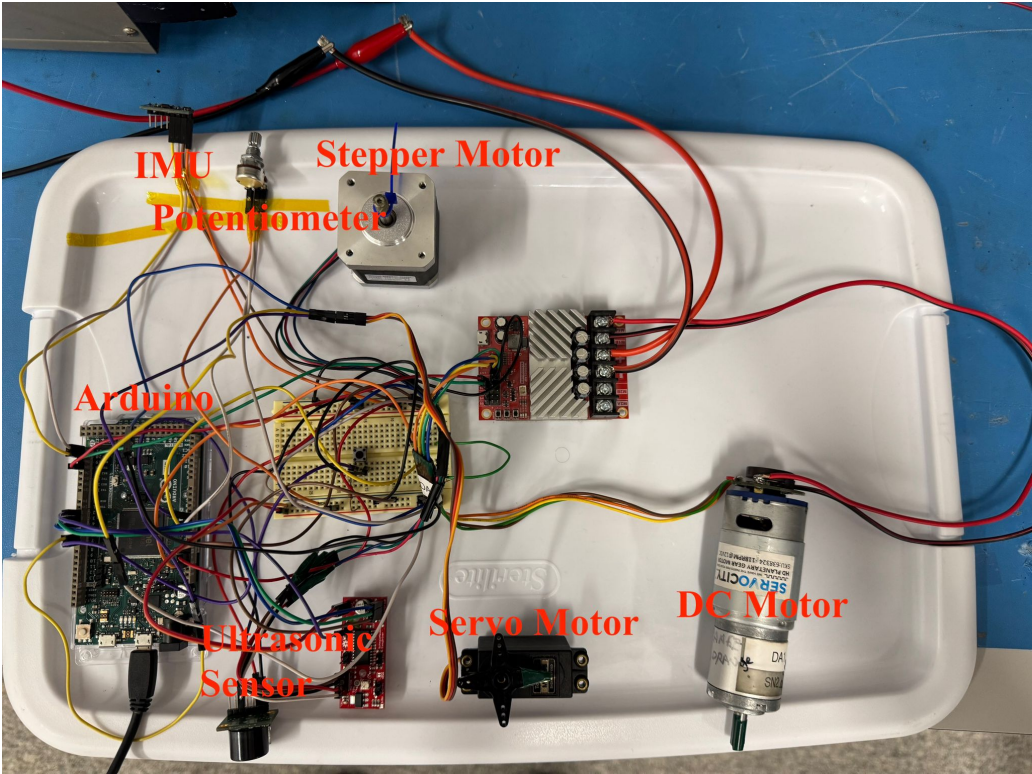


Figure 2: Sensors & Motor Control Lab Physical Setup

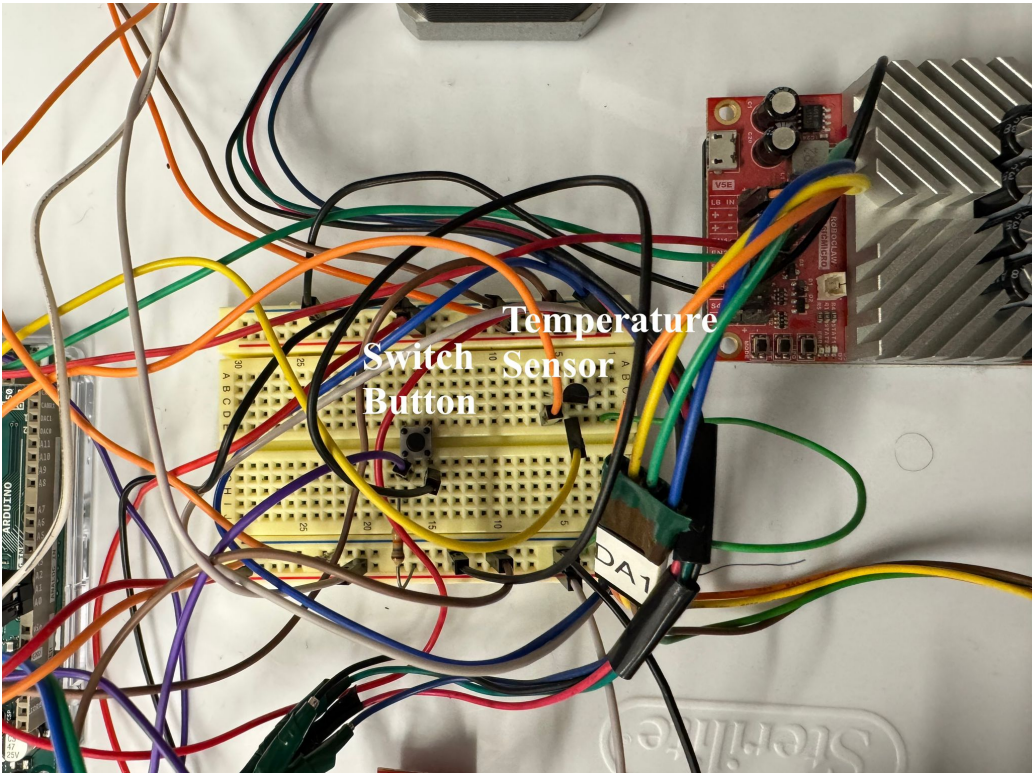


Figure 3: Close-up of Breadboard Setup

1.2 MRSD Capstone Project: Lunar ROADSTER

Software: Our team began working on the project over the winter break, and my initial focus was to set up the software stack. I started with the Jetson AGX Xavier, which will serve as the core of our system and runs the entire software stack. Since we are building on the work of the previous MRSD team, Crater Grader, the Jetson we got from them was initially flashed with Ubuntu 18.04. However, being an outdated version, we decided to upgrade the entire software, including transitioning from ROS2 Galactic to ROS2 Humble using Docker. To facilitate this, I re-flashed the Jetson with Ubuntu 20.04, configured VNC for remote access, and fully set up Docker for easier deployment. I also helped set up of the operations terminal (our main workstation) and conducted teaching sessions for my team on running and using the Jetson, and using Docker as well.

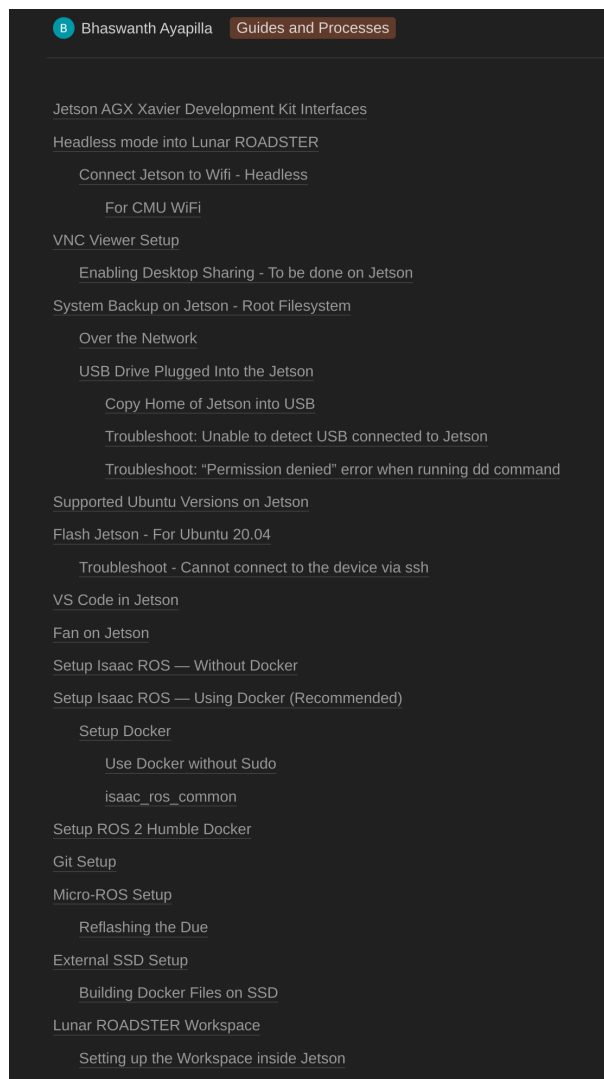


Figure 4: Documentation of Jetson AGX Xavier Setup

Sensors: My next task was setting up the motor encoders. The rover is equipped with 4 motors — two drive motors and two steer motors, both in the front and rear. All motors are controlled by the Roboclaw motor controllers, and I interfaced them with an Arduino Due. To enable communication between the motors and the Jetson AGX Xavier, I configured Micro-ROS on the Arduino. This will allow the system to send drive commands to the motors and also receive encoder data and other feedback by publishing and subscribing to relevant ROS topics. Following this, I set up the ZED 2i depth camera with

the help of my teammate William, which we will use for active mapping and validating the grading done by our rover.

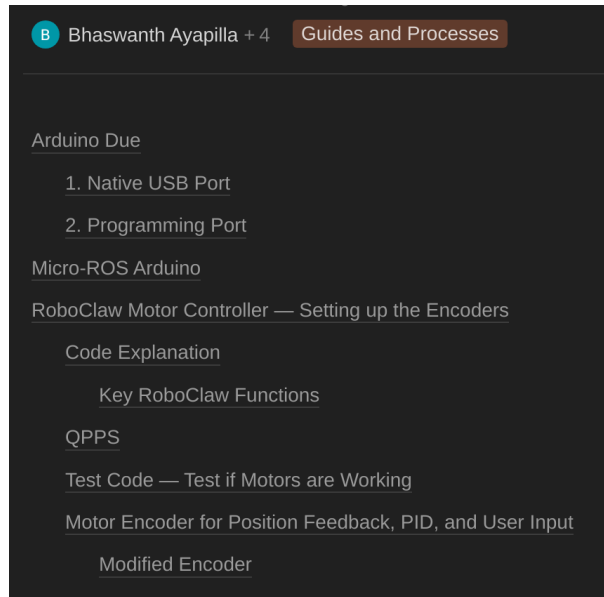


Figure 5: Documentation of Arduino Due, Micro-ROS, Encoder Setup

Teleoperation: Since we had Crater Grader’s entire software, a big initial step was to get the rover running and teleoperate it with a joystick. Me and William worked on this during the winter break and successfully got it running. Because of this, the team is now able to carry out all the necessary subsequent tests on the Moon Yard.

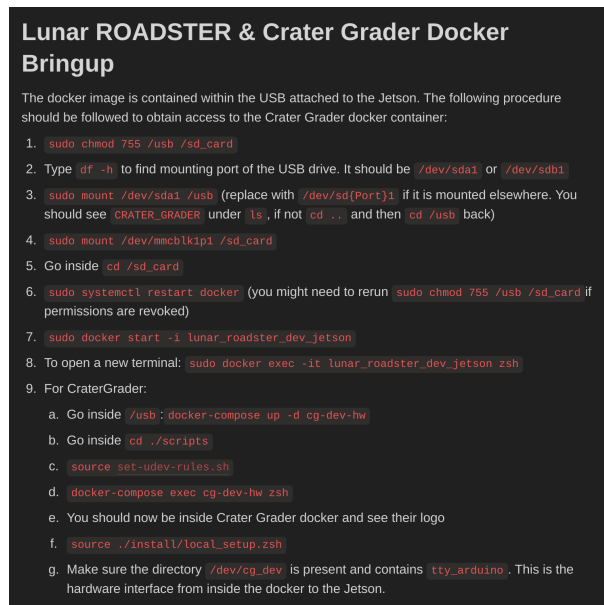


Figure 6: Documentation of Rover Bringup and Teleoperation

Equipment: In the first week of January, our team underwent a training for the FARO 3D Scanner with Wennie Tabib from FRC. This training was essential as we will use the scanner’s output to generate maps for the rover’s navigation and to evaluate the performance of the grading operation. After the training, we tested out the scanner and

successfully mapped the Moon Yard at the Planetary Robotics Lab. Over the winter break, the team also received training from Warren "Chuck" Whittaker to use the TS16 Leica-Geosystems Robotic Total Station. We will use this total station to track a prism mounted on the rover and ultimately localize it.

2 Challenges

2.1 Sensors and Motors Lab

The biggest challenge I faced in the sensors and motors lab was implementing the filter. Initially, I used a median/mode filter, which required a sorting algorithm like bubble sort. However, running the filter alongside sorting took too much time, causing delays that blocked other sensor and motor functions. Instead, I switched to a mean filter, which provided faster processing without the need for sorting.

2.2 MRSD Capstone Project: Lunar ROADSTER

Setting up Docker on the Jetson AGX Xavier was a significant challenge for me. I was fairly new to Docker and it was a pretty big learning curve for me as I had to write the dockerfiles and scripts myself. Another major challenge was teleoperating the rover using the existing software stack. I spent a considerable amount of time trying to understand the general architecture and code of Crater Grader. Additionally, I faced issues in running the ZED camera's SDK on the Jetson, despite building the relevant docker image. Ultimately, I just decided to use the camera without the SDK, and use OpenCV and other relevant libraries to get our job done.

3 Teamwork

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

3.1 Sensors and Motors Lab

- **Bhaswanth Ayapilla:** Interfacing Ultrasonic Range Finder, implementing mean filter and transfer function, implementing pushbutton for servo control, README file for the complete code.
- **Ankit Aggarwal:** Implemented the stepper motor speed and direction controller using a potentiometer.
- **Deepam Ameria:** Interfacing IMU sensor and servo motor controller.
- **Simson D'Souza:** Implemented DC motor control with encoder feedback and interfaced an IR sensor for distance measurement.
- **Boxiang (William) Fu:** GUI development and Arduino template code.

3.2 MRSD Capstone Project: Lunar ROADSTER

- **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).
- **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).
- **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.
- **Boxiang (William) Fu:** Moon Pit Crater Distribution, LAN Setup, setting up teleop (in collaboration with Bhaswanth), ZED camera setup (in collaboration with Bhaswanth)

4 Plans

4.1 Sensors and Motors Lab

Working on the sensors and motors lab provided valuable insight into sensor integration and potential challenges we might encounter. The knowledge gained will certainly benefit our capstone project. However, since this assignment is not directly related to our project, we have no further plans for it. As part of the lab, we used the RoboClaw motor driver originally installed on our rover to control the DC motor, which we will reinstall after the demonstration. Similarly, the Arduino Due will be returned to the rover setup.

4.2 MRSD Capstone Project: Lunar ROADSTER

Our team is working diligently to meet our first internal milestone on 12th February. By this deadline, we aim to complete all hardware-related tasks, including manufacturing the excavator blade, iterating on wheel designs, and performing quality assurance for the mechanical subsystem. Before the deadline, we also aim to complete the localization task and simulating the rover's navigation. Once these are completed, we will begin working on the excavator planner and the FSM planner, transitioning navigation from simulation to real-world deployment, and implementing validation using the depth camera. My current focus is on the localization of the rover. Me and William will set up the total station in the Moon Yard and use the data obtained from it and also an onboard IMU to localize the rover. Following this, I will be working on navigation and validation along with Simson. Once all team members complete their individual tasks, we will begin rigorous testing of the complete rover operation on the Moon Yard.

5 Code

Below is the Arduino code for the Sensors and Motors Lab:

```
#include <Adafruit_Sensor.h>
#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
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, 1);

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(";");
}

```

```

void handleCommand(char commandType, int value) {
    switch (commandType) {
        case 'R':
            servoMotorController(value);
            break;
        case 'S':
            stepperMotorController(value);
            break;
        case 'V':
            velDCMotorController(value);
            break;
        case 'A':
            angleDCMotorController(value);
            break;
        case 'B':
            buttonStepperMotorController(value);
            break;
        default:
            break;
    }
}

```

```

bool isNumeric(String str) {
    if (str.length() == 0) return false;

    int startIndex = 0;

    if (str[0] == '-') {
        if (str.length() == 1) return false;
        startIndex = 1;
    }

    for (unsigned int i = startIndex; i < str.length(); i++) {

```

```

        if (!isDigit(str[i])) {
            return false;
        }
    }

    return true;
}

// CONTROLLER FUNCTIONS

void servoMotorController(int control) {
    /*
    INPUT: Integer in min/max range of 0 to 180 corresponding to desired angle
    OUTPUT: Void
    */

    SERIAL_PORT.print("Servo motor controller received command: ");
    SERIAL_PORT.print(control);
    SERIAL_PORT.print(";");

    // TODO: IMPLEMENT FUNCTION BELOW
    servoAngle = control;
    servo.write(servoAngle);
}

void stepperMotorController(int control) {
    /*
    INPUT: Integer in min/max range of -180 to 180 corresponding to desired angle
    OUTPUT: Void
    */

    SERIAL_PORT.print("Stepper motor controller received command: ");
    SERIAL_PORT.print(control);
    SERIAL_PORT.print(";");

    // TODO: IMPLEMENT FUNCTION BELOW
    if (PotControlFlag == 0) {
        globalStepperValue = map(control, -180, 180, -1600, 1600);
        globalStepperAngle = control;
    }
}

void velDCMotorController(int control) {
    /*
    INPUT: Integer in min/max range of -118 to 118 corresponding to desired RPM
    OUTPUT: Void
    */

    SERIAL_PORT.print("Velocity DC motor controller received command: ");

```

```

SERIAL_PORT.print(control);
SERIAL_PORT.print(";");

// TODO: IMPLEMENT FUNCTION BELOW
// Set motor velocity (positive for forward, negative for reverse)
int dc_motor_speed = (control / 118.0) * 100.0;
int encoderSpeed = (dc_motor_speed * MAX_QPPS) / 100; // Scale input speed (user e
roboclaw.SpeedM1(ADDRESS, encoderSpeed);
}

void angleDCMotorController(int control) {
  /*
  INPUT: Integer in min/max range of -360 to 360 corresponding to desired angle
  OUTPUT: Void
  */

  // SERIAL_PORT.print("Angle DC motor controller received command: ");
  // SERIAL_PORT.print(control);
  // SERIAL_PORT.print(";");

  // TODO: IMPLEMENT FUNCTION BELOW
  uint32_t currentPos = roboclaw.ReadEncM1(ADDRESS);
  targetPosition = currentPos + (control * DEGREE_TO_TICKS);

  // Serial.print("Moving motor to position: ");
  // Serial.println(targetPosition);

  roboclaw.SpeedAccelDeccelPositionM1(ADDRESS, 10000, MAX_QPPS, 10000, targetPosition

  isMoving = true; // Set flag for movement tracking
}

void buttonStepperMotorController(int control) {
  /*
  INPUT: Boolean with 0 indicating GUI control and 1 indicating potentiometer control
  OUTPUT: Void
  */

  SERIAL_PORT.print("Button controller received command: ");
  SERIAL_PORT.print(control);
  SERIAL_PORT.print(";");

  // TODO: IMPLEMENT FUNCTION BELOW
  PotControlFlag = control;
}

// CALLBACK FUNCTIONS
int servoMotorStateCallback() {
  /*

```



```

INPUT: Void
OUTPUT: Integer in min/max range of 0 to 180 corresponding to servo motor angle
*/

// TODO: IMPLEMENT FUNCTION BELOW
return servoAngle;
}

int stepperMotorStateCallback() {
/*
INPUT: Void
OUTPUT: Integer in min/max range of -180 to 180 corresponding to stepper motor angle
*/

// TODO: IMPLEMENT FUNCTION BELOW

return globalStepperAngle;
}

int velDCMotorStateCallback() {
/*
INPUT: Void
OUTPUT: Integer in min/max range of -118 to 118 corresponding to DC motor RPM
*/

// TODO: IMPLEMENT FUNCTION BELOW

int speed = roboclaw.ReadSpeedM1(ADDRESS);
speed = speed * 118 / 6718;

return speed;
}

int angleDCMotorStateCallback() {
/*
INPUT: Void
OUTPUT: Integer in min/max range of -360 to 360 corresponding to DC motor angle
*/

// TODO: IMPLEMENT FUNCTION BELOW

int enc = roboclaw.ReadEncM1(ADDRESS);
int angle = (enc * 360) / 3416;
angle = angle % 360;

return angle;
}

int potentiometerSensorCallback() {
/*

```

```

INPUT: Void
OUTPUT: Integer corresponding to potentiometer reading
*/

// TODO: IMPLEMENT FUNCTION BELOW
PotVal = analogRead(PotentiometerPin);

if (PotControlFlag == 1) {
    globalStepperValue = map(PotVal, 0, 1022, -1600, 1600);
    globalStepperAngle = map(globalStepperValue, -1600, 1600, -180, 180);
}

return PotVal;
}

double imuSensorCallback() {
    /*
    INPUT: Void
    OUTPUT: Double corresponding to sensed IMU pitch reading
    */

    // 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 - 0.5) * 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;
    servo.write(servoAngle);
  }
}

double transferFunctionCallback(double electricalInput) {
  /*
  INPUT: Double corresponding to electrical input voltage
  OUTPUT: Double corresponding to ultrasonic reading (cm)
  */

  // TODO: IMPLEMENT FUNCTION BELOW
  double val = electricalInput * 0.498 * 2.54;

  return val;
}

void stepperCallback() {
  /*
  INPUT: Void
  OUTPUT: Void
  */
  // TODO: IMPLEMENT FUNCTION BELOW
  stepper.moveTo(globalStepperValue);
  stepper.run();
}

```