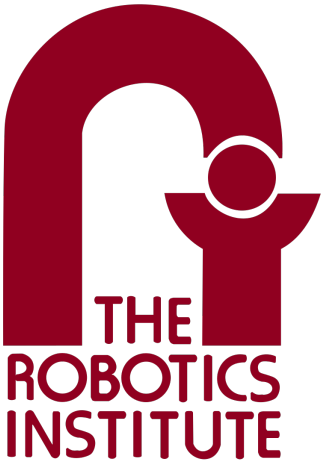
Individual Lab Report - 1



Lunar ROADSTER

Team I

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

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

February 7, 2025

Carnegie Mellon University
Robotics Institute

1 Individual Progress

1.1 Sensors and Motors Lab

I was responsible for the control of the stepper motor, interfacing the potentiometer and controlling the stepper motor using the potentiometer.

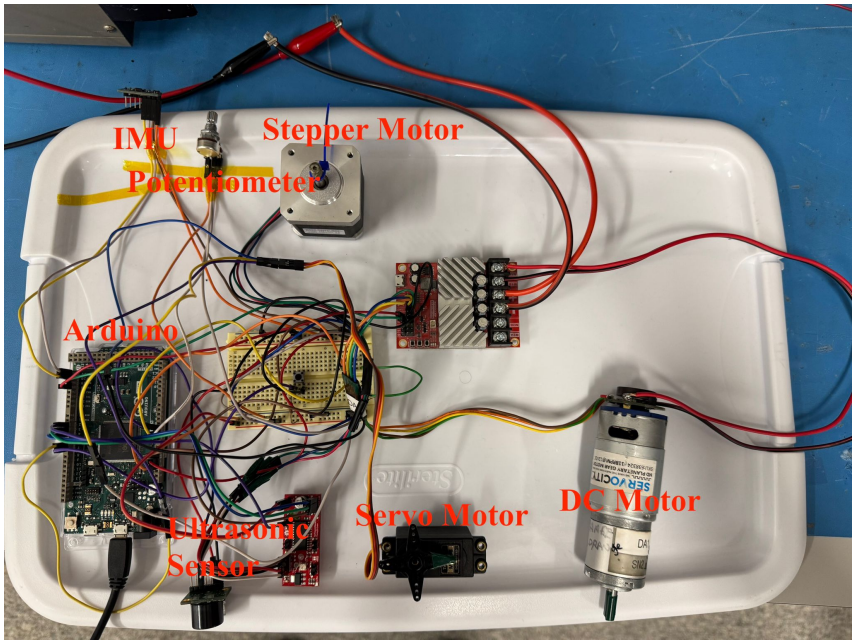


Figure 1: Hardware Setup

To setup the hardware, I used the A3967 EasyDriver Stepper Motor. It requires 4 pins to be connected to the arduino (EN, STEP, DIR, GND), 4 pins to the motor (dependent on the phases of the wiring) and 2 main power pins (PWR, GND). To confirm the phases of the stepper motor wiring, I used a multimeter. The potentiometer needs an analog pin, a 5V pin and a GND pin.

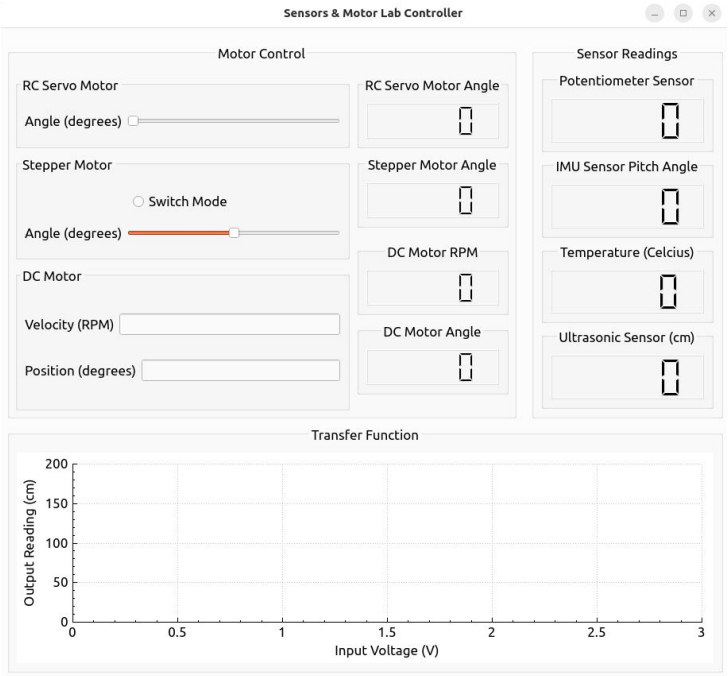


Figure 2: Graphical User Interface

I used the AccelStepper Library to implement the basic control functions of the stepper motor. As I needed to control the motor with multiple inputs, I created a global variable (globalStepperValue) to be called by the stepper controller function. This global variable can be updated by both the GUI and the potentiometer value depending on the state. Additionally, to report the angle value to the GUI, the GlobalStepperAngle variable is mapped to convert the step value into a degree value (3200 steps = 360 degrees of rotation).
The potentiometer is interfaced using the AnalogRead function. This value is reported to the GUI and is used to control the stepper motor, as described above. The potentiometer reports values between [0,1023]
The code I used for the control is shown below:

```
#include <AccelStepper.h>

#define EN_StepperDriver 2
#define Stp_StepperDriver 3
#define Dir_StepperDriver 4
#define PotPin A0

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

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

void setup() {
  // put your setup code here, to run once:
  pinMode(EN_StepperDriver, OUTPUT);
  digitalWrite(EN_StepperDriver, LOW);  // enable stepper(s)
  stepper.setMaxSpeed(2000);
  stepper.setAcceleration(1000);
  stepper.setSpeed(0);

}

void loop(){
 stepperCallback();
}

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 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;
}

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 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;
}

void stepperCallback() {
  /*
  INPUT: Void
  OUTPUT: Void
  */

  // TODO: IMPLEMENT FUNCTION BELOW
  stepper.moveTo(globalStepperValue);
  stepper.run();
}
```

1.2 Lunar ROADSTER

1.2.1 Rover Hardware Setup/Maintenance

As we inherited the CraterGrader work system, I was responsible for the maintenance of the hardware system to bring it back to working condition. This involved maintenance of the rover's drive system and drive system.

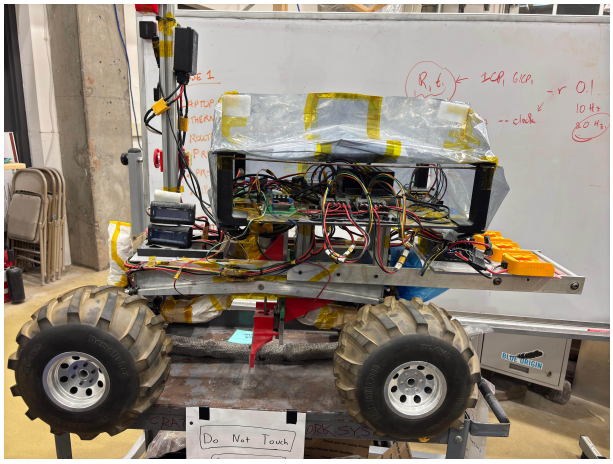


Figure 3: CraterGrader: As we inherited it

The rover's drive system consists of 2 driving motors with 2 differentials, powering the 4 wheels of the rover. These required servicing which included changing rusted bolts, oiling the gearboxes, and aligning the wheels.
Additionally, we mapped the rover's electrical power circuit in a diagram to allow us to rewire the rover with ease.

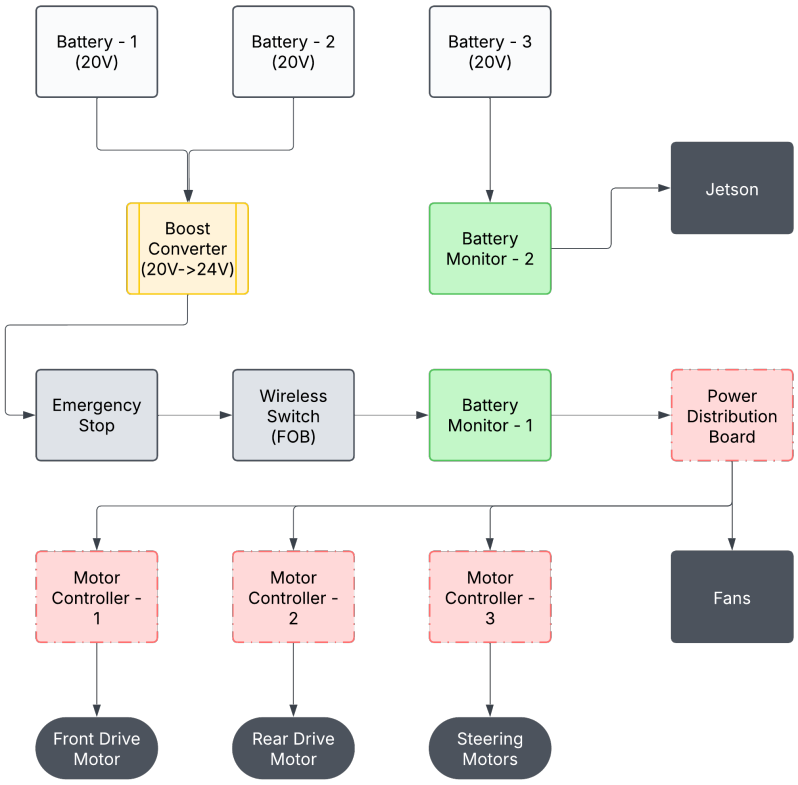


Figure 4: Power Circuit for the Rover

1.2.2 Wheel Design and Prototyping

As the rover needs to doze sand along with moving forward, we need to maximize the traction generated by the ground. Considering that Moon Yard has sandy terrain, the following requirements for designing the wheel were met:

- Rigid, Metal wheel

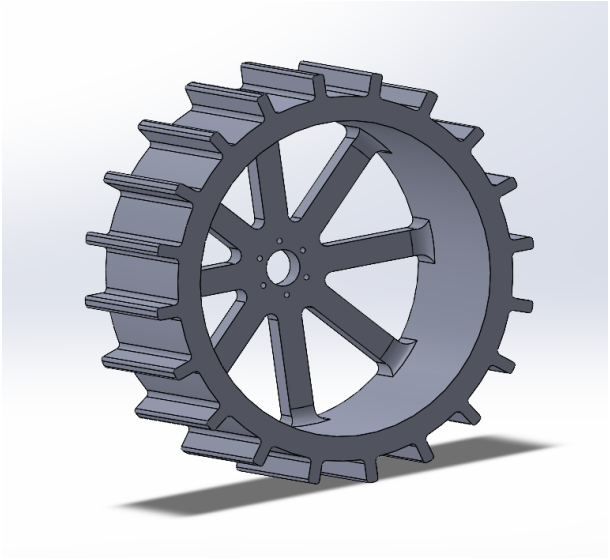


Figure 5: CAD Design of the First Wheel Prototype

- Diameter 22cm, width around 15cm (test for more traction).
- Connecting spokes with a simple wheel hub suited to the current suspension system
- Semi-floating bearing assembly
- Lifted Grousers to ensure traction in sand (Test for number and height)
- Possibly create distinct tire tracks to create features for visual odometry
- Simple Manufacturing Method

Figure 6: Printed First Prototype

1.2.3 Project Management

My primary responsibilities include conducting daily stand-up meetings, interacting with our supervisor and any other lab members, keeping the team on schedule and setting

priorities as needed. In addition, the team uses Notion and Discord as primary methods of communication and documentation. All written code (including Sensors and Motors Lab Code) is pushed to GitHub to maintain version control.

2 Challenges

2.1 Sensors and Motors Lab

A challenging part of my work was the integration of my code with the GUI. I set up a simple code structure individually for the motor control which ran in the Arduino loop. However, the function calls inside the GUI varied greatly as every sub-function was not called at the same rate due to existing code. Hence, I had to set global contexts for control variables and ensure that the functions that update the control variable were running continuously. The other callback functions for the GUI ran as required by the user's input.

2.2 Lunar ROADSTER

The wheel design has been relatively challenging due to the lack of documentation. Most of our decisions are based on Red's guidance and my previous design experiences. As a result, we will have to iteratively design the wheel with extensive testing. The hardware maintenance task was tedious as there was no clear documentation regarding the circuits and electrical functioning. We hence had to carefully dismantle it wire-by-wire and create a circuit diagram.

3 Teamwork

3.1 Sensors and Motors Lab

Bhaswanth Ayapilla: Interfacing Ultrasonic Range Finder, implementing Mean/Median 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.
Deepam Ameria: Interfacing IMU sensor and the servomotor controller
Boxiang (William) Fu: GUI development and Arduino template code.

3.2 Lunar ROADSTER

Bhaswanth Ayapilla: 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 members).
Simson D'Souza: Prototype Dozer Development (in collaboration with Deepam), preliminary tests for teleop and grading with prototype dozer (in collaboration with Deepam), Circuit Diagram (in collab with Ankit), FARO scanner setup and Moon Pit scanning (in collaboration with team members), Processed point cloud data to generate an occupancy grid map for navigation.
Deepam Ameria: Prototype Dozer Development (in collaboration with Simson), preliminary tests for teleop 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 members)
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

As our project is heavy on hardware such as sensors and motors, we all benefited from working on this lab. Additionally, we incorporated our project's motor drivers, DC motor and Arduino Due to increase relevance. However, there are no direct future plans for the circuit constructed.

4.2 Lunar ROADSTER

Our goal by the next PR is to have the hardware design finalized and have made significant progress in manufacturing. My individual goals will be:

- Designing more wheel iterations based on our tests,
- Helping Deepam with manufacturing the dozer
- Interfacing the VectorNav IMU to publish data on ROS topics
- Designing a more efficient electronics structure for the rover
- As PM, ensure the team stays on track, conduct meetings and conflict resolution

5 Sensors and Motor Control Lab Quiz

5.1 ADXL355 Datasheet

1. The sensor's typical range is given as ±3.6g. However, the minimum range is ±3g
2. The sensor's typical dynamic range is 7.2g (70m/s2) and the minimum range is 6g.
3. The purpose of the capacitor is to decouple the sensor from any noise through the power supply. It achieves this by resisting the minute changes in voltage due to noise as it offers a low-impedance path to ground for high-frequency signals.
4. $TF = 1.5V + (300mV/g * a(X, Y))$
5. The largest expected nonlinearity error is 0.3% of full scale = $0.0108g = 0.106\, m/s^2$
6. The sensor's bandwidth for the X and Y axes is 0 - 1600 Hz
7. $N = N_{density} * \sqrt{1.6 * BW} = 150 * \sqrt{1.6 * 25} = 948.683 \mu g\ rms$
8. To determine the RMS noise experimentally,
    - Set the accelerometer on a stable surface with no movement or vibrations to keep it static.
    - Power it with a stable voltage supply and connect it to a microcontroller to read the output. Ensure that all operating conditions are stable
    - Assume that the output signal is primarily noise, and the noise is Gaussian. Use the z-axis acceleration equal to the gravity vector as a reference.
    - Use the TF to calculate the acceleration of the output voltage over time.
    - Compute the mean and RMS using all the recorded acceleration values to find the noise

5.2 Signal Conditioning

5.2.1 Filtering

1. Two problems while using a moving average filter are:
    - **Sensitivity to Outliers:** As all readings are given equal weightage, an outlier will largely skew the results leading to inaccurate representations of the actual data.
    - **Introduction of Lag:** As the moving average filter needs multiple readings to compute the mean, the sensor data will lag. This can introduce problems when a real-time response is needed.
2. Two problems while using a median average filter are:
    - **Non-Linearity:** As the median filter does not use a simple mathematical operation on the input, it will cause non-linearity. This will make it more difficult to analyze and predict the readings.
    - **Computational Complexity:** Median filters require the use of sorting algorithms. This greatly increases the computational complexity relative to filters that perform simple mathematical operations. This makes it harder to implement real-time and resource-limited systems.

5.2.2 Op-amps

The given circuit is a gain and offset Op-amp circuit. The equation for V_{out} is:

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

where, $V_1 = V_{ref}, V_2 = V_{in}$

1. **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)**

   $$V_1 = V_{ref}, V_2 = V_{in}$$
   $$0 = -1.5(1 + \frac{R_f}{R_i}) - V_{ref}\frac{R_f}{R_i} \qquad (1)$$
   $$5 = 1.0(1 + \frac{R_f}{R_i}) + V_{ref}\frac{R_f}{R_i} \qquad (2)$$

   Using these equations, we can calculate the required values,

   $$\frac{R_f}{R_i} = 1, V_{ref} = -3V$$

2. **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)**

   This calibration cannot be performed using the given circuit. For the required outputs, we need a unity gain and an offset of 2.5.

   If $V_2 = V_{in}$, we would need $\frac{R_f}{R_i} = 0$, with which we cannot achieve the required offset of 2.5.

   If $V_1 = V_{in}$, we would need $\frac{R_f}{R_i} = -1$, which is not physically possible, as resistances cannot have negative values.

5.3 Control

1. A digital input for PID control can be formed using the following equation:

   $$V_{out}(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt}$$

   where,

   - V_{out} = Output Voltage of the PID Controller
   - $e(t) = V_{des} - V(t)$ = Error Signal
   - K_p = Proportional Gain
   - K_i = Integral Gain
   - K_d = Derivative Gain

2. If the system is sluggish, I would increase the **Proportional Gain Term (P)** of the controller. The proportional gain will directly increase the system rise time by increasing the influence of the error term on the control input.

3. If the system has a steady-state error, I will increase the **Integral Gain Term (I)** of the controller. The integral term accumulates past errors and drives the steady-state error of the system to zero.

4. If the system still has an overshoot, I will increase the **Derivative Gain Term (D)** of the controller. The damping term will dampen rapid changes in the system and will slow down the system when it approaches the setpoint.