INDIAN INSTITUTE OF TECHNOLOGY KANPUR

Robotics Club, Summer Project 2025

Step Explorer

Submitted by:

ANURAG RAMESHGOUDA PATIL	Roll No.: 240119
PRIYANSHU VERMA	Roll No.: 240811
ANANTHAN R	Roll No.: 240119
KARAN CHAUHAN	Roll No.: 240519
DEEVASH BAGDAL	Roll No.: 240334
GOURAV BAJWA	Roll No.: 240407
HARSH RAJ	Roll No.: 240432
VISHAKHA SHARMA	Roll No.: 241173
MANISH KAJLA	Roll No.: 240622

Project Mentors:

RIDHIMA SHARMA	Roll No.: 230854
DEEP ARYAN SINGH	Roll No.: 230346
GOLDEN KUMAR	Roll No.: 230419

Acknowledgement

We take this opportunity to express our sincere gratitude to the **Robotics Club, IIT Kanpur** for initiating and organizing the Summer Project, which enabled us to pursue the development of our project titled *Step Explorer: A Stair-Climbing Robot*. This project provided us with a unique platform to convert theoretical knowledge into practical application and work as a cohesive team towards solving real-world engineering challenges. We are immensely thankful to our mentors — **Ridhima Sharma**, **Golden Kumar**, and **Deep Aryan Singh** — for their constant encouragement, expert guidance, and technical support throughout the duration of the project. Their patience, insightful feedback, and ability to help us navigate through design and implementation hurdles were instrumental in the successful completion of our work. Their mentorship taught us not only technical skills but also how to think critically, collaborate effectively, and persevere through setbacks.

We extend our gratitude to the entire Robotics Club community for fostering an environment of peer learning. The availability of shared resources, timely help from fellow club Secretaries and Coordinators, and open discussions helped us understand and tackle complex problems in a structured and efficient manner. The welcoming atmosphere of the club encouraged us to ask questions, share ideas, and learn from each other without hesitation.

We would also like to thank the institute, Indian Institute of Technology Kanpur, for providing us with access to essential resources, workspaces, and the academic infrastructure that facilitated our work. The spirit of innovation, encouragement of student-led initiatives, and emphasis on practical learning at IIT Kanpur made this project both possible and impactful.

Last but certainly not least, we would like to express our heartfelt appreciation to each member of the *Step Explorer* team. The synergy, commitment, creativity, and late-night brainstorming sessions that we shared were the backbone of this project. Working with such a motivated and passionate team has been both an enjoyable and enriching experience, and we look forward to carrying this learning forward in future endeavors.

Contents

1 Introduction

The StepXplorer project was undertaken as part of the Robotics Club, IIT Kanpur's summer projects, with the goal of developing a stair-climbing robot capable of navigating vertical steps using an innovative mechanical design. Unlike traditional legged robots, our approach relies on a wheeled rotating mechanism that enables the robot to climb over small steps through controlled motion and stability. Staircases pose a unique challenge for mobile robots, especially when operating in environments designed primarily for humans.

Our project aimed to address this gap by designing a compact, Arduino-controlled robot that can climb stairs using precise actuation and minimal electronics. The project emphasized hands-on learning in mechanical design, control systems, and embedded programming. Through iterative design and testing, the team explored multiple climbing mechanisms,tuned motor control algorithms. This documentation outlines the motivation, design method- ology, control logic, challenges encountered, and outcomes of the SteXplorer project.

The secondary objective of our project was to develop a self-balancing platform that maintains the payload surface in a vertical orientation at all times, using servo motors for active stabilization.

2 Background Study and Training

As a prerequisite and training process, resources that explained the working of the components to be used were provided. Multiple designs of similar step-climbing robots were used as inspiration for the design we have implemented. Below are some of the Resources Shared:

- Basic Tutorials for CAD.
- Basic Tutorials for Arduino.
- Understanding Breadboard Connections.
- PID controllers.
- Running a motor.
- TB66oo stepper motor driver.

3 System Design and Implementation

3.1 Components Used

3.1.1 HC-SR04 ultrasonic sensor

Our system employs ultrasonic sensors placed at three different vertical heights to distinguish between stairs and obstacles. If each of the three sensors detects a different height level, the surface is identified as a staircase. In contrast, if the height readings are the same or nearly equal, the surface is treated as an obstacle.

Additionally, two side-mounted sensors are used for collision avoidance. If the distance measured by the right-side sensor is less than 10 cm, the robot halts its movement to the right. Similarly, if the left-side sensor detects a distance below 10 cm, leftward movement is restricted to prevent collisions.

3.1.2 MPU6050 IMU module

The MPU-6050 is a 6-axis(combines 3-axis Gyroscope, 3-axis Accelerometer) motion tracking device. Changes in motion, acceleration and rotation can be detected.

This module is used in the implementation of the secondary objective (balancing the payload).

3.1.3 NEMA 17 stepper motor

These motors have been used as the main driving motors of the bot as they have a very high torque and can be controlled very precisely.

3.1.4 Servo motor

These motors have been used in the secondary objective (balancing a payload)

3.1.5 Arduino Mega

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560. It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button.

We have used the Arduino Mega microcontroller as the main processing module of the bot. All the sensors and motor drivers are connected to an Arduino Mega.

3.1.6 DIV268N or TB6600 Stepper Motor Driver

Stepper motor drivers are required as an interface between the microcontroller and the stepper motors. It is used to control the motors according to our requirements.

3.2 Mechanical Design

We have implemented a four-wheel drive design that takes input from sensors and adjusts its path accordingly to climb stairs and avoid obstacles. We selected this mechanical design due to its compactness and ease of construction compared to more complex alternatives. The simplicity of the mechanism makes it more reliable and easier to fabricate within the given constraints.
Additionally, the dimensions have been carefully chosen to comply with the guidelines of a robotics competition in which the Robotics Club, IIT Kanpur, will be participating.

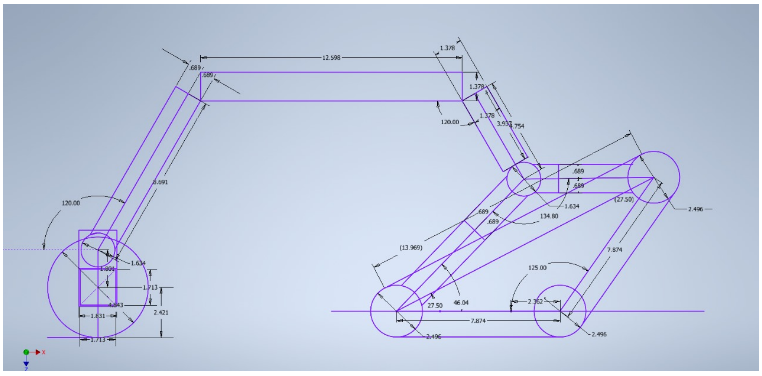

Figure 1: motor mount

Figure 2: the front hinges

3.3 Electronics

The bot circuit consists of two Arduino Mega microcontrollers, one of which processes the inputs from the ultrasonic sensors while the second one runs the motors.

A custom PCB was made which consists of 6 ultrasonic sensors arranged 2 each at different heights, which will help detect stairs and obstacles. The GND and VCC pins of all these sensors are connected together. One sensor each was also kept on the left and right side of the robot to measure the clearance on both sides. All the sensors are then connected to the Arduino Mega, which then processes the data and sends the simple binary output through digital pins to the other Arduino Mega.

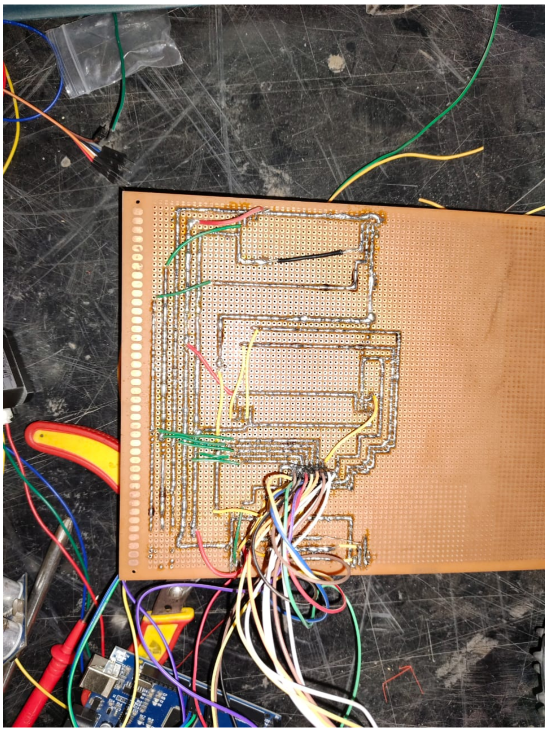

Figure 3: PCB Circuit Back

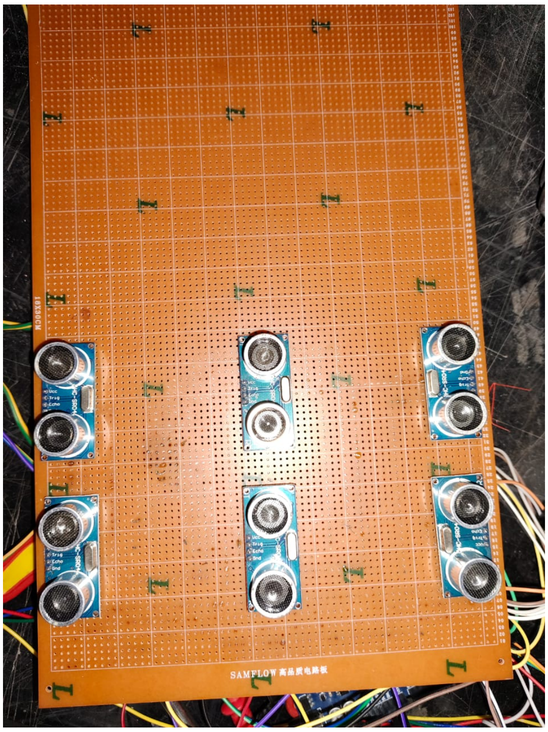

Figure 4: PCB Circuit Front

The second Arduino Mega controls the motors. All the NEMA 17 stepper motors are connected according to the motor driver datasheet to the motor drivers, and the DIR+ and PUL+ pins of each motor driver are connected to the Arduino Mega. the DIR- and PUL- pins are connected to a common ground.

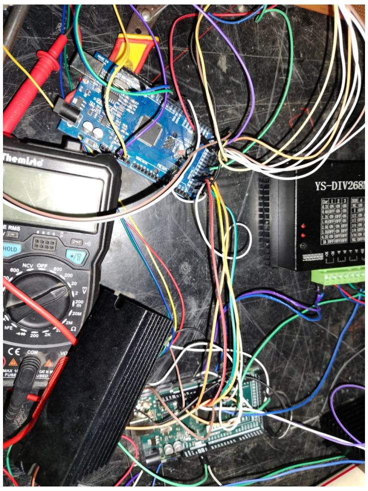

Figure 5: arduino to arduino

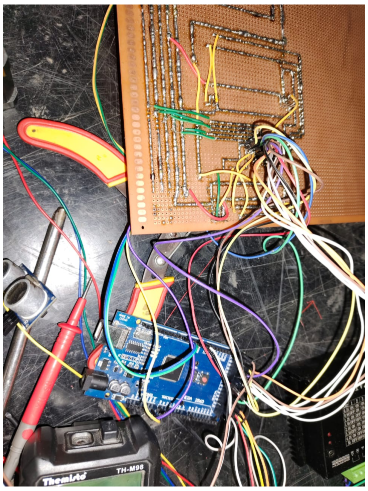

Figure 6: Sensor to arduino

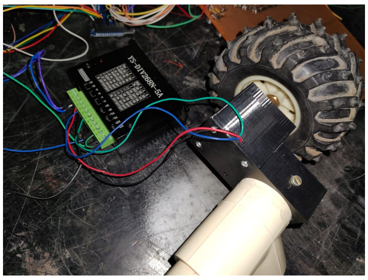

Figure 7: Driver

This Arduino is connected by jumper wires to the other Arduino Mega which gives it input on how the motors are to be run.

3.4 Balancing the Payload

The practical applicability of our robot depends heavily on the stabilization of the payload platform where objects are placed. To ensure this, we designed a balancing surface with dimensions of 20 cm $\times$ 20 cm. Stabilization is achieved using four servo motors positioned to control the tilt in multiple directions. This required precise torque calculations to ensure the system could reliably balance a payload weighing up to 5 kg. Following the analysis, support arms were 3D printed with sufficient strength and rigidity. These arms were then securely mounted using M3 screws of 15 mm and 18 mm lengths, allowing robust and stable articulation.

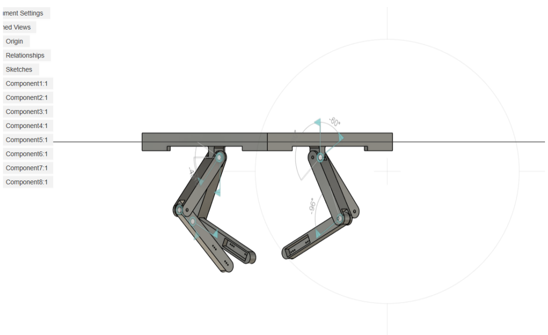

Figure 8: Balancing CAD

Figure 9: 3D printed parts with hinges

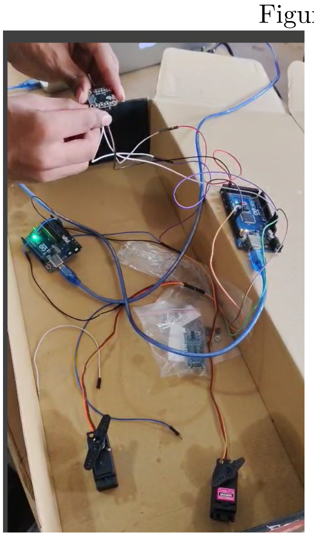

Figure 10: Working of the Circuit Demo

3.5 Control Algorithm

The control Algorithm of the bot consists of the following parts:

Code for operating the sensors

```
    const int trig1 = 25;

const int echo1 = 24;

const int trig2 = 29;

const int echo2 = 28;

const int trig3 = 31;

const int echo3 = 30;

const int trig4 = 33;

const int echo4 = 32;

const int trig5 = 35;
const int echo5 = 34;

const int trig6 = 43;
const int echo6 = 42;

const int trig7 =2 ;
const int echo7 = 3;

const int trig8 = 4;
const int echo8 = 5;


float duration1;
float duration2;
float duration3;
float duration4;
float duration5;
float duration6;
float duration7;
float duration8;

float distance1;
float distance2;
float distance3;
float distance4;
float distance5;
float distance6;
float distance7;
float distance8;

const int leftF = 47;
const int rightF = 48;
const int leftB = 49;
const int rightB = 50;
const int speed = 51;

void setup(){
 Serial.begin(9600);

 pinMode(trig1,OUTPUT);
 pinMode(echo1,INPUT);

 pinMode(trig2,OUTPUT);
 pinMode(echo2,INPUT);

 pinMode(trig3,OUTPUT);
 pinMode(echo3,INPUT);

 pinMode(trig4,OUTPUT);
 pinMode(echo4,INPUT);

```

```
 pinMode(trig5,OUTPUT);
 pinMode(echo5,INPUT);

 pinMode(trig6,OUTPUT);
 pinMode(echo6,INPUT);

 pinMode(trig7,OUTPUT);
 pinMode(echo7,INPUT);

 pinMode(trig8,OUTPUT);
 pinMode(echo8,INPUT);

  pinMode(leftF,INPUT);
  pinMode(leftB,INPUT);
  pinMode(rightF,INPUT);
  pinMode(rightB,INPUT);

 digitalWrite(speed,HIGH);
 digitalWrite(leftF,HIGH);
 digitalWrite(leftB,LOW);
 digitalWrite(rightF,HIGH);
 digitalWrite(rightB,LOW);
}

void loop(){
  digitalWrite(trig1,LOW);
  delay(10);
  digitalWrite(trig1,HIGH);
  delay(10);
  digitalWrite(trig1,LOW);
  duration1 = pulseIn(echo1,HIGH);

  digitalWrite(trig2,LOW);
  delay(10);
  digitalWrite(trig2,HIGH);
  delay(10);
  digitalWrite(trig2,LOW);
  duration2 = pulseIn(echo2,HIGH);

  digitalWrite(trig3,LOW);
  delay(10);
  digitalWrite(trig3,HIGH);
  delay(10);
  digitalWrite(trig3,LOW);
  duration3 = pulseIn(echo3,HIGH);

  digitalWrite(trig4,LOW);
  delay(10);
  digitalWrite(trig4,HIGH);
  delay(10);
  digitalWrite(trig4,LOW);
  duration4 = pulseIn(echo4,HIGH);

  digitalWrite(trig5,LOW);
  delay(10);
  digitalWrite(trig5,HIGH);
  delay(10);
  digitalWrite(trig5,LOW);
  duration5 = pulseIn(echo5,HIGH);

  digitalWrite(trig6,LOW);
  delay(10);
  digitalWrite(trig6,HIGH);
  delay(10);
  digitalWrite(trig6,LOW);
  duration6 = pulseIn(echo6,HIGH);

  digitalWrite(trig7,LOW);
  delay(10);
  digitalWrite(trig7,HIGH);
  delay(10);
  digitalWrite(trig7,LOW);
  duration7 = pulseIn(echo7,HIGH);
```

```

  digitalWrite(trig8,LOW);
  delay(10);
  digitalWrite(trig8,HIGH);
  delay(10);
  digitalWrite(trig8,LOW);
  duration8 = pulseIn(echo8,HIGH);

  distance1 = duration1*0.343*0.05;
  distance2 = duration2*0.343*0.05;
  distance3 = duration3*0.343*0.05;
  distance4 = duration4*0.343*0.05;
  distance5 = duration5*0.343*0.05;
  distance6 = duration6*0.343*0.05;
  distance7 = duration7*0.343*0.05;
  distance8 = duration8*0.343*0.05;

  Serial.print("Bottom right= " + String(distance1));//bottom
  Serial.print(" ");
  Serial.print("Bottom left= " + String(distance2));//bottom
  Serial.print(" ");
  Serial.print("Mid right=" + String(distance3));//mid
  Serial.print(" ");
  Serial.print("Mid left=" + String(distance4));//mid
  Serial.print(" ");
  Serial.print("Top right="+String(distance5));
  Serial.print(" ");
  Serial.println("Top left="+String(distance6));
  Serial.print("Left = " + String(distance7));
  Serial.print("");
  Serial.println(" Right = "+String(distance8));

  float avgDist = (distance1+distance2+distance3+distance4+distance5+distance6)/6;

   if(distance7<15){
    digitalWrite(rightF,LOW);
    digitalWrite(rightB,HIGH);
  }

  if(distance8<15){
    digitalWrite(leftF,LOW);
    digitalWrite(leftB,HIGH);
  }

  if(avgDist<20){
    digitalWrite(leftF,LOW);
    digitalWrite(leftB,HIGH);
  }else {
    digitalWrite(leftB,LOW);
    digitalWrite(rightB,LOW);
    digitalWrite(leftF,HIGH);
    digitalWrite(rightF,HIGH);
  }


  delay(500);
}

bool compare(int a, int b){
  int dif = a-b;
  if(dif<0) dif*=-1;
  if(dif<2) return 1;
  return 0;
}
```

Listing 1: Ultrasonic Sensor Array with Collision Avoidance

Code for getting input from the first Arduino and running motors according to the input received.

```
#include <AccelStepper.h>

// Motor control pins
const int FR = 5;    // Front right motor
const int FL = 3;    // Front left motor (small driver)
const int BR = 9;    // Back right motor
const int BL = 6;    // Back left motor

// Input pins
const int RFront = 8;    // Right front switch
const int RBack = 2;     // Right back switch
const int LFront = 10;   // Left front switch
const int LBack = 7;     // Left back switch
const int speedPin = 4; // Speed control

int speed;  // Current speed setting

// Initialize stepper motors
AccelStepper front_right(1, FR, 10);  // (1 = driver, step pin, direction pin)
AccelStepper front_left(1, FL, 13);
AccelStepper back_right(1, BR, 40);
AccelStepper back_left(1, BL, 28);

void setup() {
  // Set input pins
  pinMode(RFront, INPUT);
  pinMode(RBack, INPUT);
  pinMode(LFront, INPUT);
  pinMode(LBack, INPUT);
  pinMode(speedPin, INPUT);

  // Configure motor parameters
  front_right.setMaxSpeed(400);
  front_right.setAcceleration(400);
  front_left.setMaxSpeed(400);
  front_left.setAcceleration(400);
  back_right.setMaxSpeed(400);
  back_right.setAcceleration(400);
  back_left.setMaxSpeed(400);
  back_left.setAcceleration(400);

  speed = 600; // Default speed
}

void loop() {
  // Set speed based on speed pin
  if(digitalRead(speedPin) == HIGH) {
    speed = 600; // High speed
  } else {
    speed = 300; // Low speed
  }

  // Right front movement
  if(digitalRead(RFront) == HIGH) {
    front_right.moveTo(front_right.currentPosition() + 100);
    back_right.moveTo(back_right.currentPosition() + 100);
    front_right.setSpeed(speed);
    back_right.setSpeed(speed);
    front_right.runSpeed();
    back_right.runSpeed();
  }

  // Right back movement
  if(digitalRead(RBack) == HIGH) {
    front_right.moveTo(front_right.currentPosition() - 100);
    back_right.moveTo(back_right.currentPosition() - 100);
    front_right.setSpeed(speed);
    back_right.setSpeed(speed);
    front_right.runSpeed();
    back_right.runSpeed();
  }

```

```
  // Left front movement
  if(digitalRead(LFront) == HIGH) {
    front_left.moveTo(front_left.currentPosition() + 100);
    back_left.moveTo(back_left.currentPosition() + 100);
    front_left.setSpeed(speed);
    back_left.setSpeed(speed);
    front_left.runSpeed();
    back_left.runSpeed();
  }

  // Left back movement
  if(digitalRead(LBack) == HIGH) {
    front_left.moveTo(front_left.currentPosition() - 100);
    back_left.moveTo(back_left.currentPosition() - 100);
    front_left.setSpeed(speed);
    back_left.setSpeed(speed);
    front_left.runSpeed();
    back_left.runSpeed();
  }
}
```
Listing 2: Four-Wheel Stepper Motor Control

Code for running the servo based on the MPU output to get the balanced surface.

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

Adafruit_BNO055 bno = Adafruit_BNO055(55);

Servo rollServo;  // Z-axis
Servo pitchServo; // Y-axis

const int ROLL_SERVO_PIN = 9;
const int PITCH_SERVO_PIN = 10;
const int SERVO_CENTER = 90;

// Z-axis
float rollAngle = 0;
float previousRoll = 0;
int rollServoPosition = SERVO_CENTER;

// Y-axis
float pitchAngle = 0;
float previousPitch = 0;
int pitchServoPosition = SERVO_CENTER;

const float ROLL_GAIN = 3.0;
const float PITCH_GAIN = 3.0;
const float DEADBAND = 2.0;
const int MAX_SERVO_CHANGE = 10;

bool servoTestMode = true;
unsigned long lastTestTime = 0;
int testStep = 0;

void setup(void)
{
  Serial.begin(9600);
  Serial.println("2-Axis Balancing Bot");

  rollServo.attach(ROLL_SERVO_PIN);
  pitchServo.attach(PITCH_SERVO_PIN);
  rollServo.write(SERVO_CENTER);
  pitchServo.write(SERVO_CENTER);
  Serial.println("Servos initialized at center position");

  if(!bno.begin())
  {
    Serial.print("No BNO055 detected");
    while(1);
  }

  delay(1000);
  bno.setExtCrystalUse(true);
  Serial.println("BNO055 initialized");

  sensor_t sensor;
  bno.getSensor(&sensor);

  Serial.println("2-axis compensation");
  delay(1000);
}

void loop(void)
{
  sensors_event_t event;
  bno.getEvent(&event);

  rollAngle = event.orientation.z;
  pitchAngle = event.orientation.y;

  if (rollAngle > 180) {
    rollAngle = rollAngle - 360;
```

```
  }

  if (pitchAngle > 180) {
    pitchAngle = pitchAngle - 360;
  }

  bool rollActive = abs(rollAngle) > DEADBAND;
  bool pitchActive = abs(pitchAngle) > DEADBAND;

  if (rollActive) {

    int targetRollServoPos = SERVO_CENTER - (rollAngle * ROLL_GAIN);

    targetRollServoPos = constrain(targetRollServoPos, 0, 180);

    int rollServoChange = targetRollServoPos - rollServoPosition;
    if (abs(rollServoChange) > MAX_SERVO_CHANGE) {
      if (rollServoChange > 0) {
        rollServoPosition += MAX_SERVO_CHANGE;
      } else {
        rollServoPosition -= MAX_SERVO_CHANGE;
      }
    } else {
      rollServoPosition = targetRollServoPos;
    }

    rollServo.write(rollServoPosition);
  }

  if (pitchActive) {
    int targetPitchServoPos = SERVO_CENTER - (pitchAngle * PITCH_GAIN);

    targetPitchServoPos = constrain(targetPitchServoPos, 0, 180);

    int pitchServoChange = targetPitchServoPos - pitchServoPosition;
    if (abs(pitchServoChange) > MAX_SERVO_CHANGE) {
      if (pitchServoChange > 0) {
        pitchServoPosition += MAX_SERVO_CHANGE;
      } else {
        pitchServoPosition -= MAX_SERVO_CHANGE;
      }
    } else {
      pitchServoPosition = targetPitchServoPos;
    }

    pitchServo.write(pitchServoPosition);
  }

  Serial.print("X: ");
  Serial.print(event.orientation.x, 2);
  Serial.print("\tY(Pitch): ");
  Serial.print(pitchAngle, 2);
  if(pitchActive) Serial.print(" ACTIVE");
  Serial.print("\tZ(Roll): ");
  Serial.print(rollAngle, 2);
  if(rollActive) Serial.print(" ACTIVE");
  Serial.print("\tRoll Servo: ");
  Serial.print(rollServoPosition);
  Serial.print("  \tPitch Servo: ");
  Serial.print(pitchServoPosition);
  Serial.println("  ");

  previousRoll = rollAngle;
  previousPitch = pitchAngle;
  delay(50);
}
```

Listing 3: Four- Servo Motor Control

4 Problems Faced

Several challenges were encountered during the development of the robot. One of the major issues involved the connections of the ultrasonic sensors. The use of jumper wires and soldered joints often led to unstable connections, which resulted in the microcontroller receiving inaccurate or fluctuating values. Ensuring consistent and error-free signal transmission required repeated verification and reinforcement of connections.

Another significant issue was the lower-than-expected torque output from the motors. This not only affected the climbing performance but also required additional attention in synchronizing all four motors. Precise control was necessary to ensure that all motors started and stopped simultaneously in order to produce smooth and coordinated motion.

Additionally, problems arose during the 3D printing phase. Frequent power disruptions led to minor imperfections in the printed parts. As a result, post-processing steps such as sanding and trimming were required to ensure that the components fit correctly and operated smoothly.

5 Current Work

We are currently focused on calibrating the ultrasonic sensors by determining their respective offsets to improve distance accuracy and consistency. Simultaneously, we are engaged in the body-building phase of the robot, assembling the structural components and integrating the subsystems. In parallel, we are also tuning the PID parameters for the servo motors to achieve a smooth and stable balancing mechanism for the payload platform.