Range Balancing Bot

An autonomous vehicle system that maintains a fixed distance from walls or a moving object while operating an open ground or conveyor mechanism.

Project Information

- **Course**: PDE4446
- **Authors**:
  - Aman Mishra (M00983641)
  - Arman Shaikh (M01096196)

Problem Statement

How can an autonomous vehicle accurately maintain a fixed distance from a wall or boundary while on a conveyor belt or a manually manipulated boundary when placed on the floor?

Solution Overview

This project implements a closed-loop feedback system using an Arduino microcontroller to precisely regulate the vehicle's lateral position. The system employs a PID (Proportional-Integral-Derivative) controller for stable performance and accurate distance maintenance.

Components

Hardware

- **Arduino Board**: Main microcontroller executing the PID algorithm
- **HC-SR04 Ultrasonic Sensor**: Measures wall distance for PID input
- **DC Motors and Wheels**: Provides propulsion and directional control
- **L293D Motor Driver**: Controls motor direction and speed via PWM
- **NEMA Stepper Motor**: Drives the conveyor belt
  - Powered by separate power supply

    - Controlled via A4988 stepper driver
- **Capacitor**: Provides steady current protection
- **Potentiometer**: Manual conveyor belt speed control

Key Features

- Real-time distance measurement and correction
- User-adjustable conveyor belt speed
- Safety protocols including:
    - Output clamping on PID controller
    - Maximum speed limits

Tuning Approach

This involved an iterative tuning process:

- Start with Ki and Kd at zero
- Increase or decrease Kp until the robot starts to oscillate around the Setpoint.
- Reduce or increase Kp to about half the value that caused oscillations.
- Slowly increase or decrease Ki to eliminate any small, persistent steady-state error.
- Slowly increase or decrease Kd to reduce overshoot and dampen oscillations caused by Kp.

The core solution is a PID controller that dynamically modulates the vehicle's velocity to maintain a fixed Setpoint distance from a wall. An ultrasonic sensor continuously provides the real-time distance measurement, which serves as the Process Variable (PV).

The PID algorithm computes the error signal ($e=SP-PV$) and generates a corresponding Manipulated Variable by applying proportional (Kp), integral (Ki), and derivative (Kd) gains. This MV is translated into a Pulse Width Modulation (PWM) signal, which is fed to an L293D H-bridge motor driver.

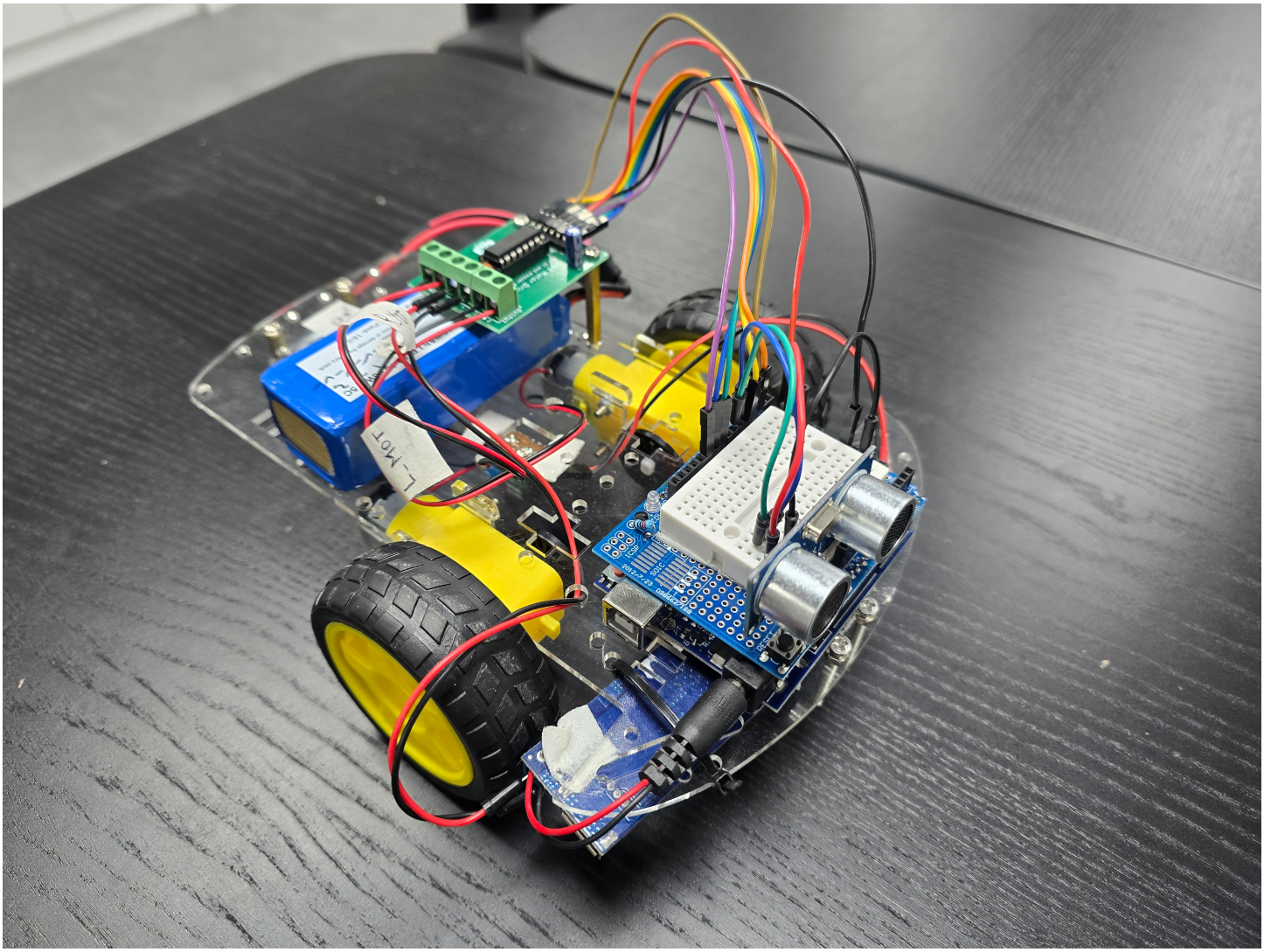
The driver, in turn, powers the DC propulsion motor, thus forming a system that actively corrects for positional drift. This control loop operates concurrently with a secondary system: a user-adjustable conveyor belt driven by a NEMA stepper motor, whose speed is governed by analog input from a potentiometer.

Critical safety protocols were embedded in the software, including output clamping on the PID controller to a maximum speed limit (255 PWM) and a minimum speed floor (125 PWM) to prevent H-bridge thermal damage from stall current when the robot's weight prevents movement at lower speeds.

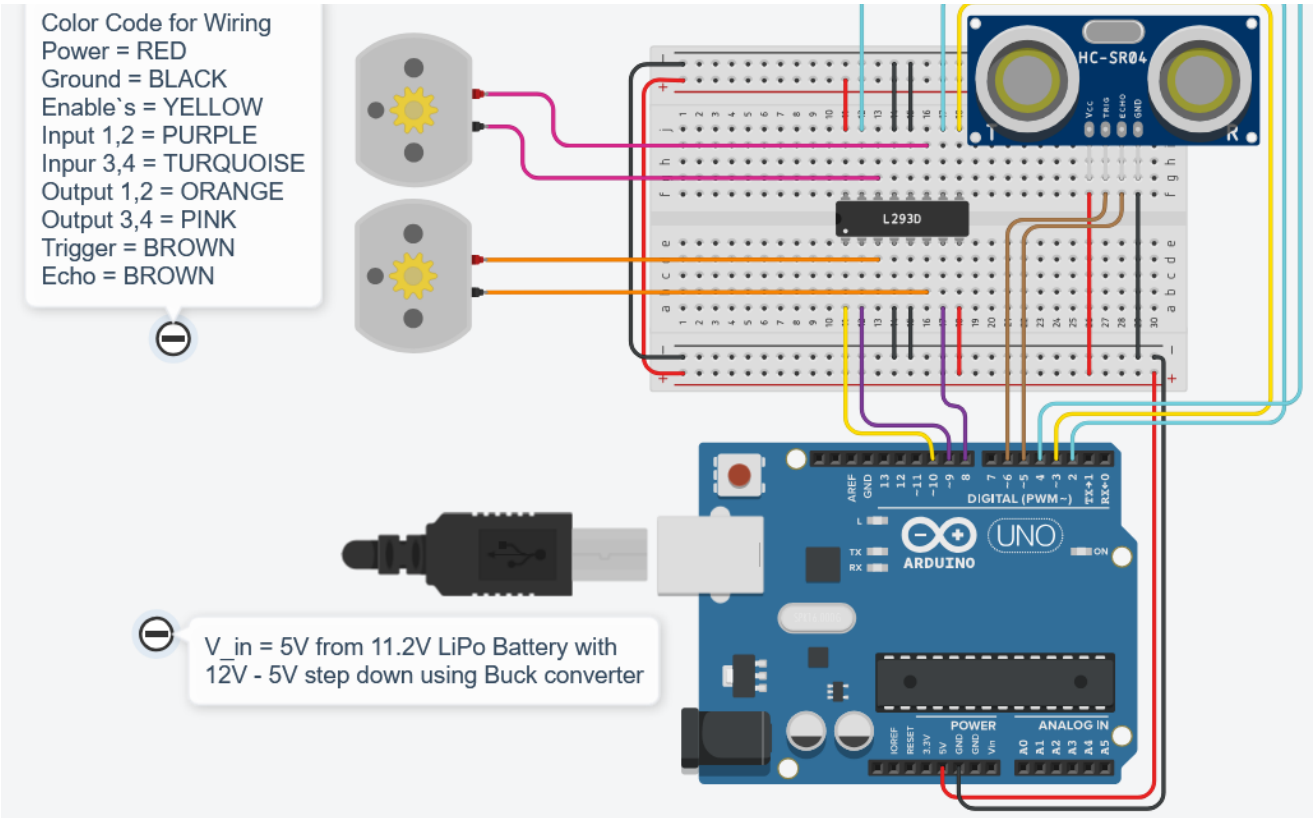
Implementation Process

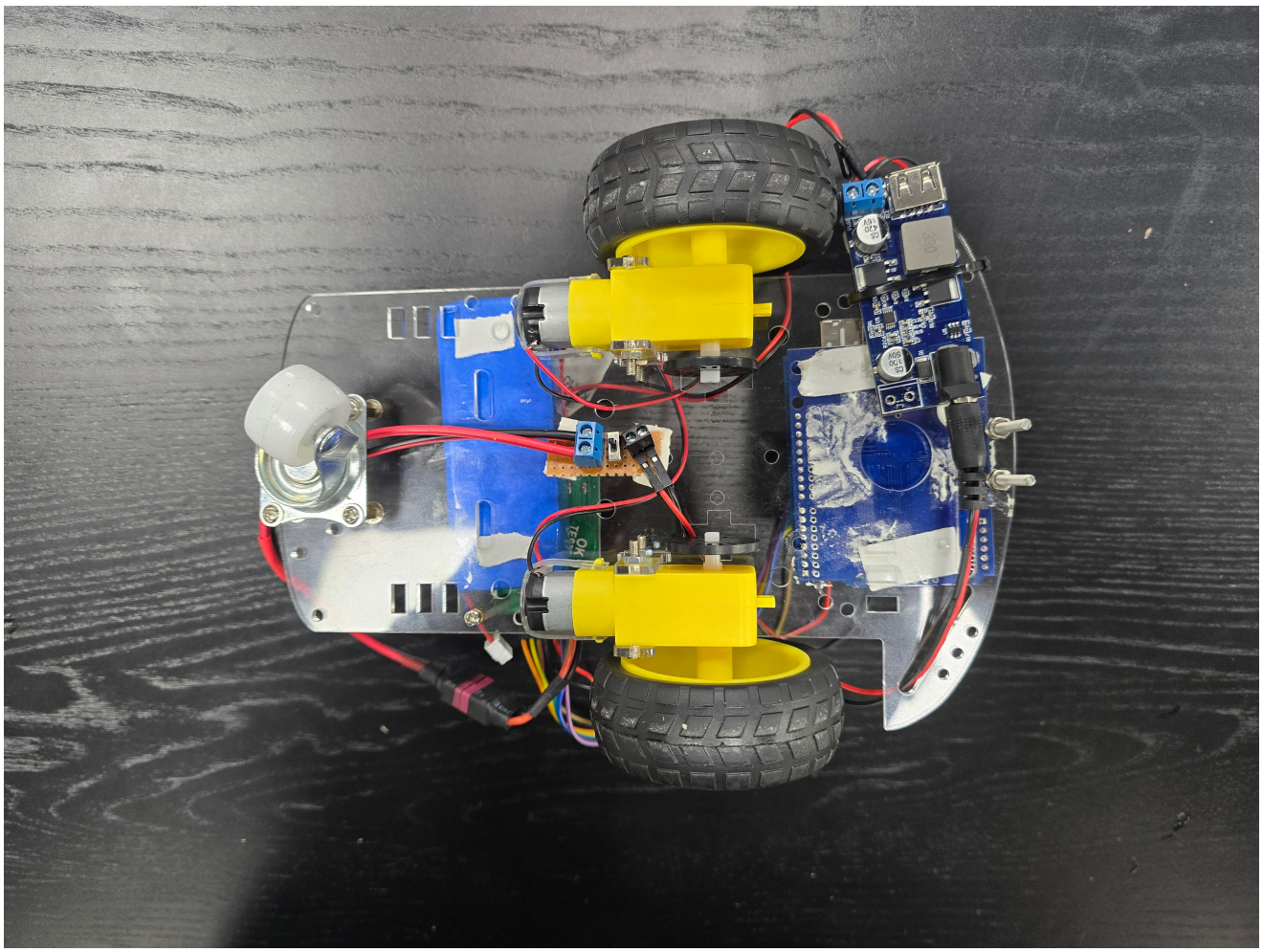
Phase 1: Robot Chassis Assembly

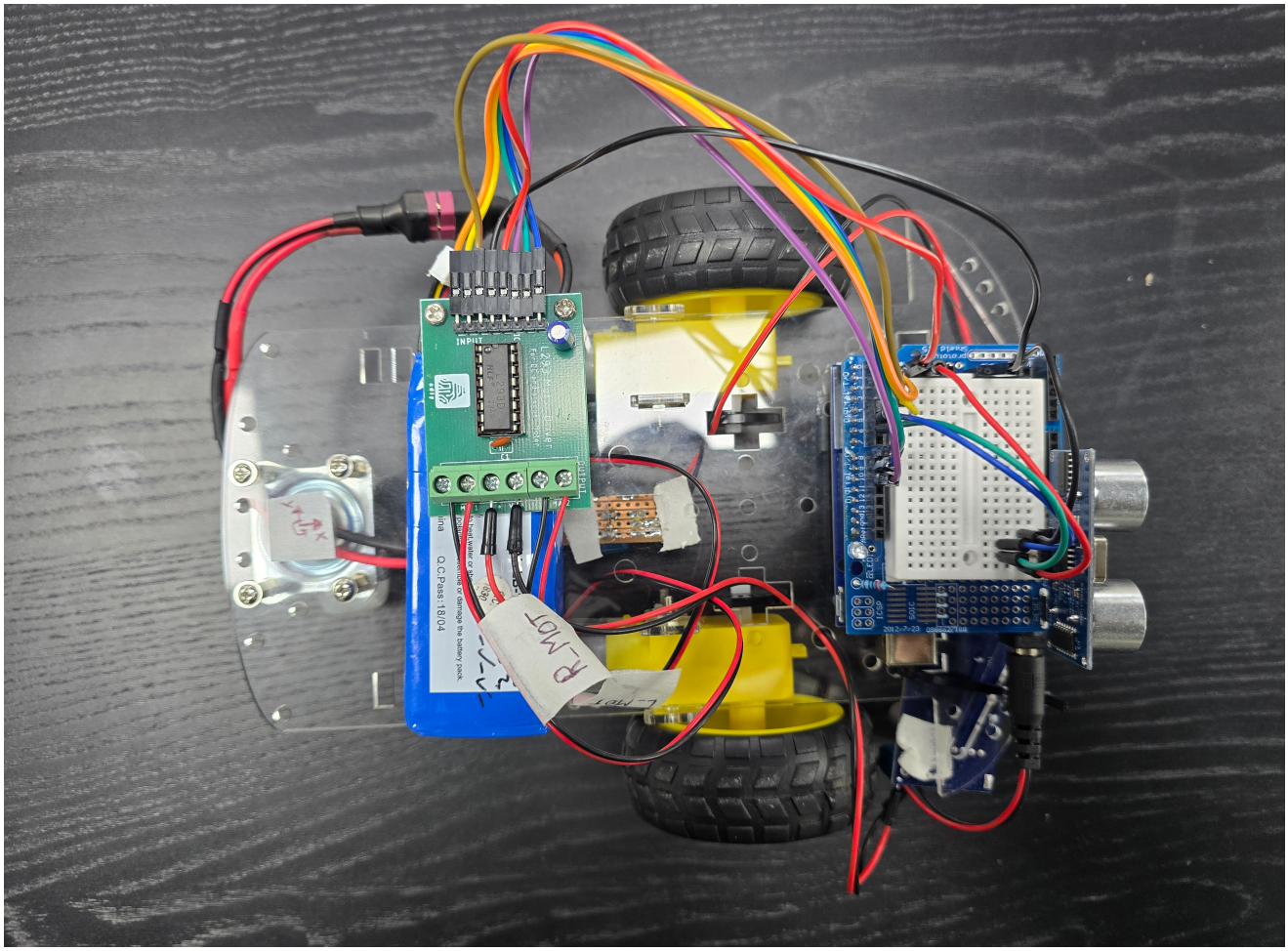
1. **Base Construction**
    - Build chassis framework
    - Mount DC motors and attach wheels
    - Install core electronics (Arduino, L298N, buck converter, battery)
    - Mount ultrasonic sensor facing tracking wall



2. **Wiring**
    - Connect battery to L298N (VCC and GND)
    - Wire L293D outputs to DC motors
    - Connect control pins to Arduino
    - Configure PWM pins for speed control


Color Code for Wiring
Power = RED
Ground = BLACK
Enable`s = YELLOW
Input 1,2 = PURPLE
Inpur 3,4 = TURQUOISE
Output 1,2 = ORANGE
Output 3,4 = PINK
Trigger = BROWN
Echo = BROWN
HC-SR04
L293D
V_in = 5V from 11.2V LiPo Battery with
12V - 5V step down using Buck converter
ARDUINO
UNO
DIGITAL (PWM~)
POWER
ANALOG IN

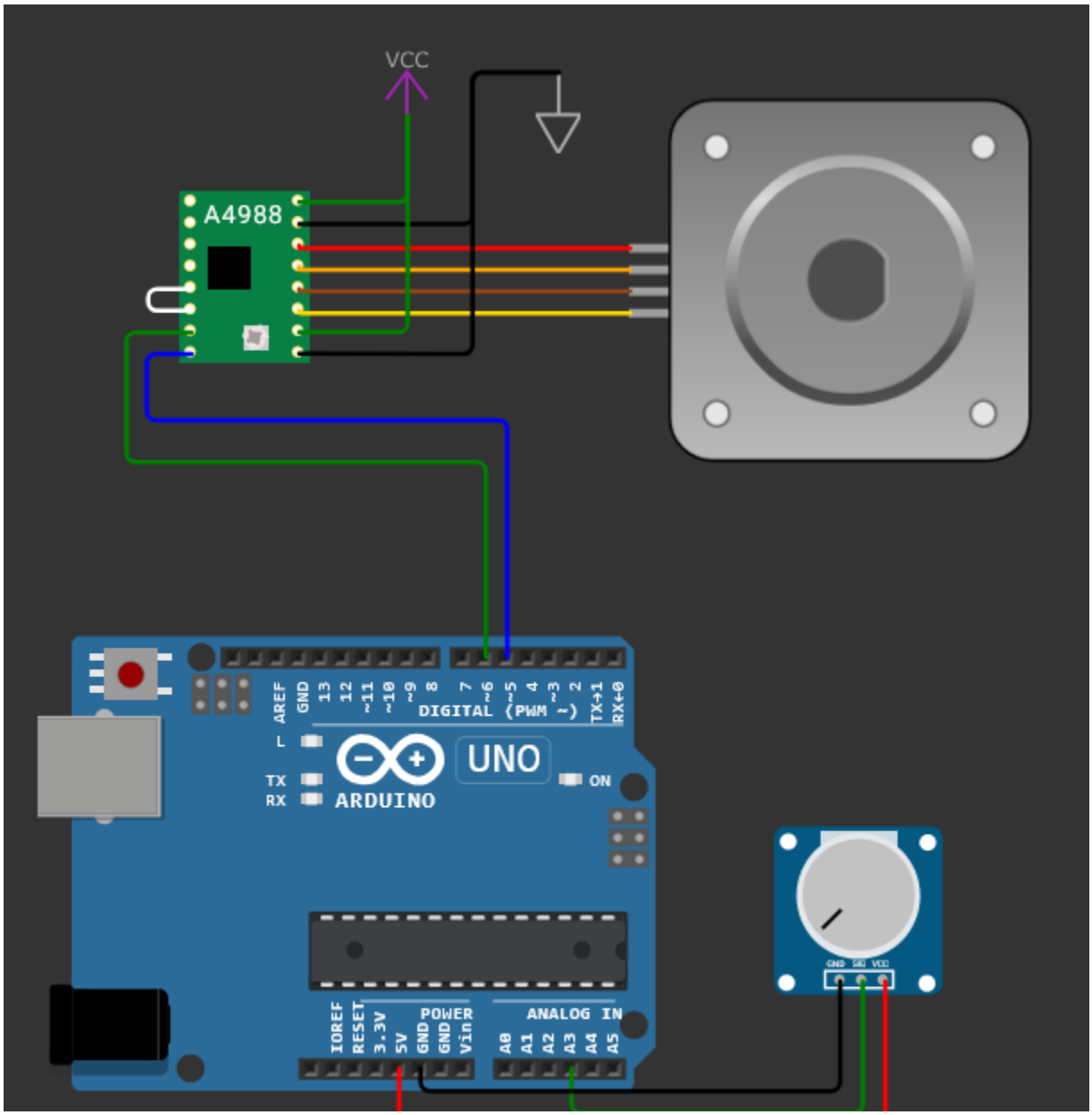


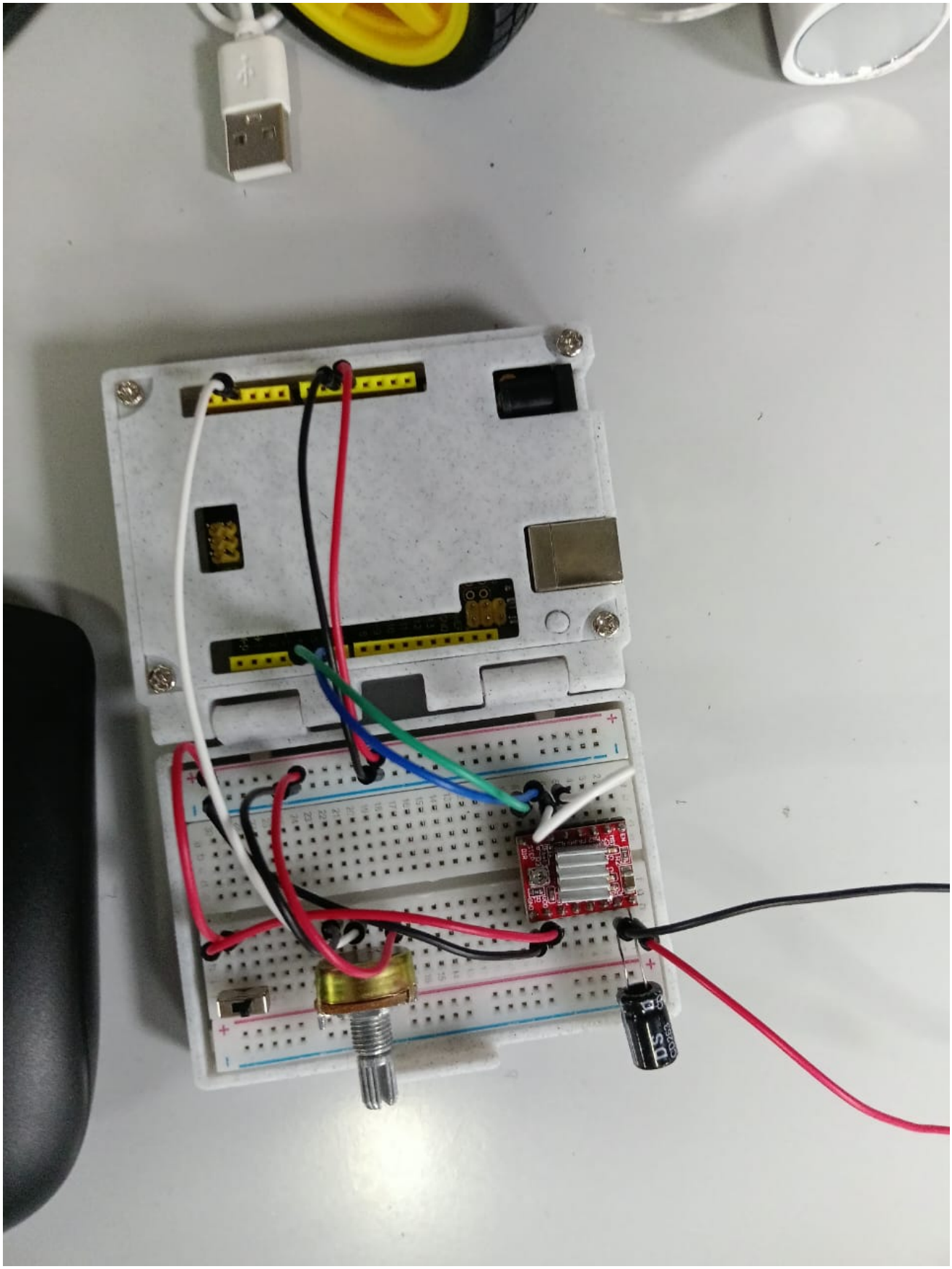


3. **Initial Testing**
    - Upload basic movement test sketch
    - Verify forward/backward motion
    - Test motor direction and connections

Phase 2: Conveyor Integration

1. **Mechanical Assembly**
    - Mount NEMA stepper motor to frame
    - Couple motor shaft to drive roller
    - Install and tension conveyor belt
    - Mount A4988 driver and potentiometer


VCC
A4988
DIGITAL (PWM ~)
UNO
ARDUINO
TX
RX
ON
POWER
ANALOG IN



2. **Electronic Setup**

```
// Pin Configuration
const int dirPin = 5;    // Stepper direction pin
const int stepPin = 6;   // Stepper step pin
const int potPin = A3;   // Potentiometer input

// Speed Parameters
const int minSpeed = 200;
const int maxSpeed = 1000;
```

3. **Control Implementation**

```
#include <AccelStepper.h>

// Initialize stepper with driver pins
AccelStepper stepper(AccelStepper::DRIVER, stepPin, dirPin);

void setup() {
  stepper.setMaxSpeed(maxSpeed);
  stepper.setSpeed(800);  // Initial speed
}

void loop() {
  // Read and map potentiometer value
  int potValue = analogRead(potPin);
  long stepSpeed = map(potValue, 0, 1023, minSpeed, maxSpeed);

  // Update and run stepper
  stepper.setSpeed(stepSpeed);
  stepper.runSpeed();
}
```

Phase 3: PID Implementation

1. **System Configuration**

```
// Pin Definitions
#define trigPin 6
#define echoPin 5
#define enA 10
#define in1 9
#define in2 8
#define enB 3
#define in3 4
#define in4 2

// PID Parameters
double Kp = 1.5;     // Proportional gain
double Ki = 0.80;    // Integral gain
double Kd = 1.2;     // Derivative gain

// Control Variables
double setPoint = 25.0;          // Target distance (cm)
double setpointVariance = 0.5;   // Acceptable deviation
const double outputLimitMin = 125;
const double outputLimitMax = 255;
```

2. **PID Algorithm Implementation**

```
double computePID() {
  double deltaTime = (double)(currentTime - previousTime) / 1000.0;

  // Proportional Term
  propError = setPoint - Input;
  double pTerm = fabs(Kp * propError);

  // Integral Term
  integralError += propError * deltaTime;
  integralError = constrain(integralError, outputLimitMin, outputLimitMax);
  double iTerm = Ki * integralError;

  // Derivative Term
  diffError = (propError - lastPropError) / deltaTime;
  double dTerm = Kd * diffError;

  // Compute and Constrain Output
  double output = pTerm + iTerm + dTerm;
  output = constrain(output, outputLimitMin, outputLimitMax);

  lastPropError = propError;
  return output;
}
```

3. **Safety Implementation**
   - Output clamping: `constrain(output, outputLimitMin, outputLimitMax)`
   - Minimum speed floor (125 PWM) to prevent H-bridge overheating due to high stall current when robot mass prevents movement at lower PWM values
   - Direction control based on output sign
   - PWM speed application to motors
4. **System Testing**
   - Basic position maintenance
   - Conveyor operation interference testing
   - Safety system verification
     - Maximum speed limits
     - Large-error emergency stop
     - System stability checks

Results

Floor Operation Performance

The robot demonstrated excellent performance in maintaining a target distance of 25 cm from walls during floor operation. Under optimal conditions (wall surface parallel to the sensor), the system achieved:

- Steady-state position within ±0.5 cm of target (24.5-25.5 cm range)
- Consistent response to moving wall surfaces
- Reliable position correction and maintenance

When encountering non-parallel wall surfaces, the system exhibited:

- Increased settling time due to ultrasonic sensor echo variations
- Eventually achieved steady state despite initial difficulties
- Maintained stability once settled

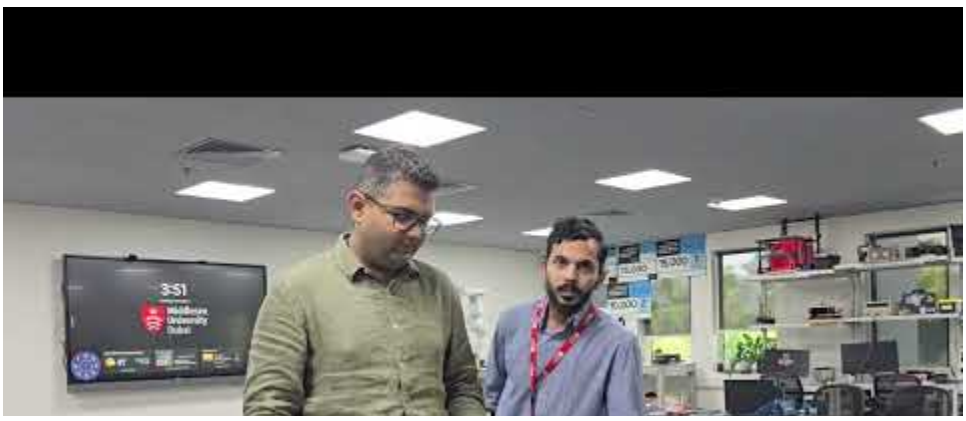
Conveyor System Limitations

The conveyor system implementation faced significant mechanical challenges:

- Excessive robot mass exceeded the conveyor belt's operational capacity
- Frequent mechanical coupling failures between stepper motor and drive shaft
- PID parameters optimized for floor operation proved incompatible with conveyor dynamics
- Project scope constraints led to prioritizing floor operation optimization

Based on these findings, development efforts were concentrated on perfecting the floor-based operation, where the system demonstrated reliable and precise distance management capabilities.

Demonstration Video: https://www.youtube.com/embed/GML_GhrLPKk



Challenges and Solutions

Vehicle Control

- **Challenge**: Free wheel causing directional instability
- **Solution**: Implemented 10% speed decrease on right wheel for stability

Conveyor System (Abandoned)

- **Challenge 1**: Belt tension and surface friction issues
  - Initial attempt: Added paper tape layer for increased roughness
  - Final solution: Increased development time so conveyor abandoned
- **Challenge 2**: Motor slippage due to worn 3D printed holders
  - Temporary fix: New holders with hot glue reinforcement
  - Long-term solution: Develop screw and nut lock to hold the stepper motors but project was abandoned as conveyor belt was out of scope.

Contribution Matrix

Task	Aman Mishra	Arman Shaikh
Hardware Assembly	[60%]	[40%]
Simulation Sketches	[20%]	[80%]
Electrical Wiring	[70%]	[30%]
Robot PID Tuning	[50%]	[50%]
Project Report	[25%]	[75%]
Demonstration Video	[50%]	[50%]

References

- VSCode Copilot was used in the documentation of the code. In addition it was used for correcting all grammatical errors and improving the layout of README.md in the project.
- AccelStepper.h example code for the conveyor belt code for continuous rotation at set speed.
- No intellectual property was made using AI.