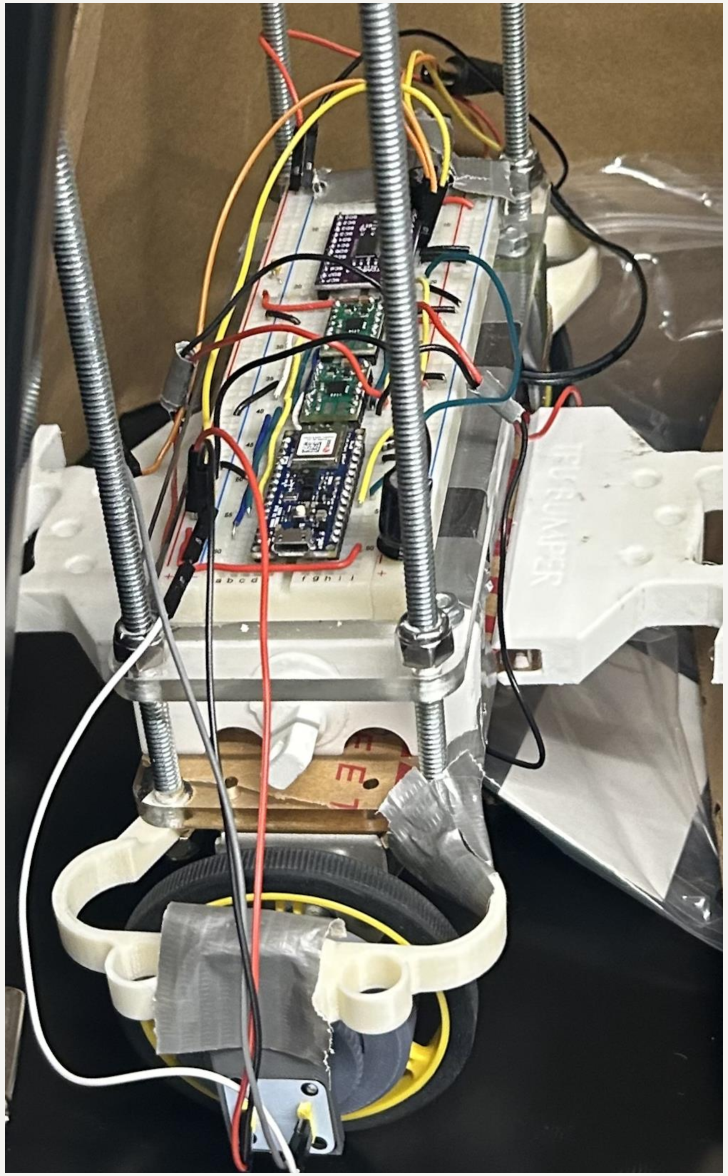
Self-Balancing Robot

-ROSEMARY CHAN

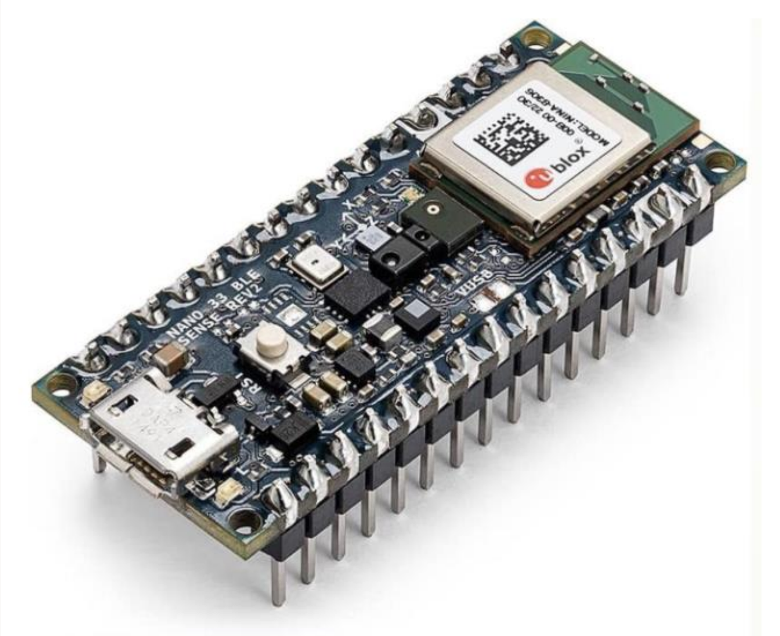
Project Overview

- Designed and built a self-balancing two-wheeled robot
- 3-person team, brainstorm together and choose best implementation
- Objectives:
  - Balance in place (±4 cm target)
  - Move forward/backward 50 cm and pause
  - Turn 45° and pause
  - Climb a 10° ramp



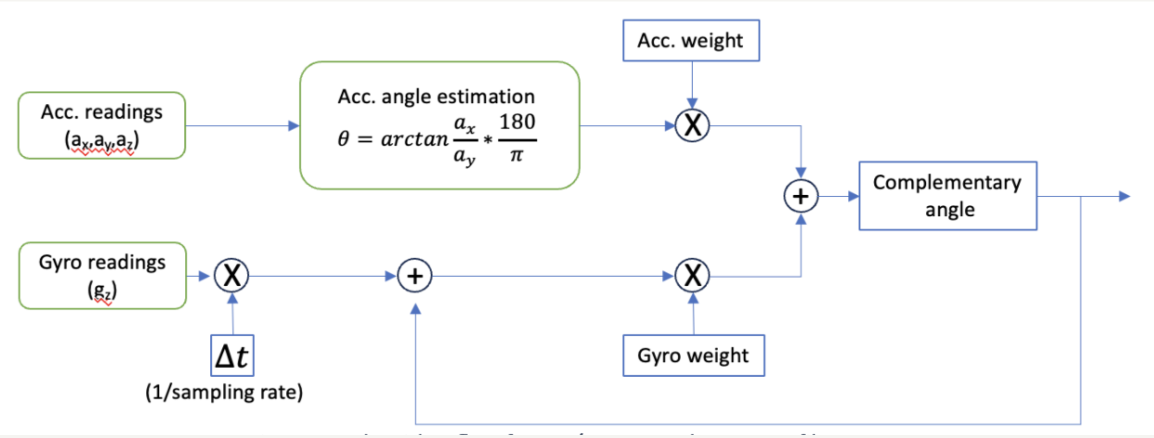
Hardware & Tools Used

- Arduino Nano 33 BLE Sense Rev2 (NRF 52840 mcu)
- 2 × AS5600 magnetic encoders (via TCA9548A I^2C mux)
- 2 × DRV8833 dual H-bridge motor drivers (paralleled)
- 3D-printed chassis + 8 × 1.5 V rechargeable batteries
- Flutter Bluetooth controller app
- Arduino IDE
- Oscilloscope, Multimeter, Power supply



Angle & Velocity Sensing

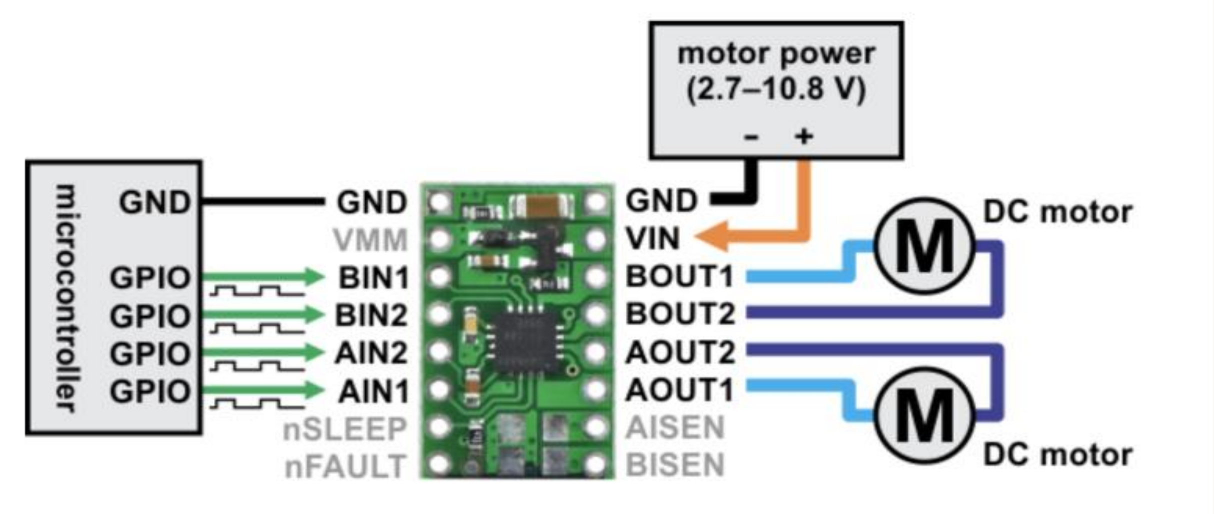
- Encoders → Angular velocity via I^2C multiplexer
- IMU (accelerometer + gyro) → Tilt angle
- Complementary filter for angle estimation:




```
if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable()) {
  // Read accelerometer
  IMU.readAcceleration(acc_x, acc_y, acc_z);
  acc_theta = atan2(acc_y, acc_z) * 180.0 / M_PI;

  // Read gyroscope
  IMU.readGyroscope(gyro_x, gyro_y, gyro_z);
  gyro_x = -gyro_x;

  // Calculate delta time
  unsigned long now = micros();
  float dt = (now - start_time) / 1000000.0;
  start_time = now;

  // Integrate gyro
  gyro_theta += gyro_x * dt;

  // Complementary filter
  theta = (1 - k) * acc_theta + k * (old_theta + gyro_x * dt);
```

PWM Motor Control

- Arduino outputs PWM to H-bridges
- Vary duty cycle → set motor speed
- Offset wheel speeds to turn




```
void forward(int num, int pwm){
  if(num == A){
    analogWrite(AIN1,255);
    analogWrite(AIN2,constrain(255 - pwm*turn_L, 0, 255));
  } else if(num == B){
    analogWrite(BIN1,255);
    analogWrite(BIN2,constrain(255-pwm*turn_R, 0, 255));
  }
}
```

Balancing PID

- Compute error = θ_measured – θ_setpoint (0°).
- PID → PWM (0–255).
- Sign of PID output → forward/backward direction; magnitude → speed.

```
// PID calculations
error = (offset_angle + desired_angle) - theta;
d_theta = (old_theta - theta) / dt;
i_theta += ki *(error + old_error) / 2.0 * dt;
i_theta = constrain(i_theta,-50,50);
pid_out_imu = (kp * error) + (i_theta) + (kd * d_theta);

error_en = desired_vel - vel;
i_theta_en += ki_en *(error_en + old_error_en) / 2.0 * dt;
pid_out_en = (kp_en * error_en) + constrain(i_theta_en,-50,50);

pid_out = k_pid * pid_out_imu + (1-k_pid)*pid_out_en;

if(pid_out < -255 or pid_out > 255) {
  pwm = 255;
}
else pwm = int(abs(pid_out));
```

Forward/Backward Motion

Version 1: Weighted PWM Blend

- Compute velocity error → PWM.
- Blend with balance-PID PWM: pwm_final = k*pwm_balance + (1–k)*pwm_velocity.
- Problem: tendency to accelerate forever to maintain tilt → eventual tip-over.

```
//Angular speed
angV0 =encoderLeft.getAngularSpeed(AS5600_MODE_RPM);
I2CMux.closeChannel(0);
I2CMux.openChannel(1);
angV1 = encoderRight.getAngularSpeed(AS5600_MODE_RPM);
I2CMux.closeChannel(1);
I2CMux.openChannel(0);
vel =(-angV0 + angV1)/2;

// PID calculations
error = (offset_angle + desired_angle) - theta;
d_theta = (old_theta - theta) / dt;
i_theta += ki *(error + old_error) / 2.0 * dt;
i_theta = constrain(i_theta,-50,50);
pid_out_imu = (kp * error) + (i_theta) + (kd * d_theta);

error_en = desired_vel - vel;
i_theta_en += ki_en *(error_en + old_error_en) / 2.0 * dt;
pid_out_en = (kp_en * error_en) + constrain(i_theta_en,-50,50);

pid_out = k_pid * pid_out_imu + (1-k_pid)*pid_out_en;

if(pid_out < -255 or pid_out > 255) {
  pwm = 255;
}
else pwm = int(abs(pid_out));
```

Version 2: Cascaded PID

- Outer PID: velocity error → desired tilt angle.
- Inner PID: balance using that tilt setpoint.
- Result: smooth, steady motion and proper pauses.

```
vel = vel * PI / 180 * 4;// in cm/sec
Serial.print("velocity: ");
Serial.println(vel);

//displacement into a capping(?) constant
error_en = desired_vel - vel;
Serial.print("velocity desired: ");
Serial.println(desired_vel);

Serial.print("velocity error: ");
Serial.println(error_en);
i_theta_en += ki_en *(error_en + old_error_en) / 2.0 * dt;
Serial.print("kp_en: ");
Serial.println(kp_en);
Serial.print("i theta_en: ");
Serial.println(i_theta_en);
pid_out_en = (kp_en * error_en) + constrain(i_theta_en,-10,10);
Serial.print("output desired angle:  ");
Serial.println(pid_out_en);
desired_angle = constrain(pid_out_en,-15,15);

// PID calculations
error = (offset_angle + desired_angle) - theta;
d_theta = (old_theta - theta) / dt;
i_theta += ki *(error + old_error) / 2.0 * dt;
i_theta = constrain(i_theta,-50,50);
pid_out = (kp * error) + (i_theta) + (kd * d_theta);
if(pid_out < -255 or pid_out > 255) {
  pwm = 255;
}
else pwm = int(abs(pid_out));
```

Turning

- Left wheel turn faster than right
- Turn right: go forwards
- Turn left: go backwards
- Required to return to original position

```
void forward(int num, int pwm){
  if(num == A){
    analogWrite(AIN1,255);
    analogWrite(AIN2,constrain(255 - pwm*turn_L, 0, 255));
  } else if(num == B){
    analogWrite(BIN1,255);
    analogWrite(BIN2,constrain(255-pwm*turn_R, 0, 255));
  }
}
```

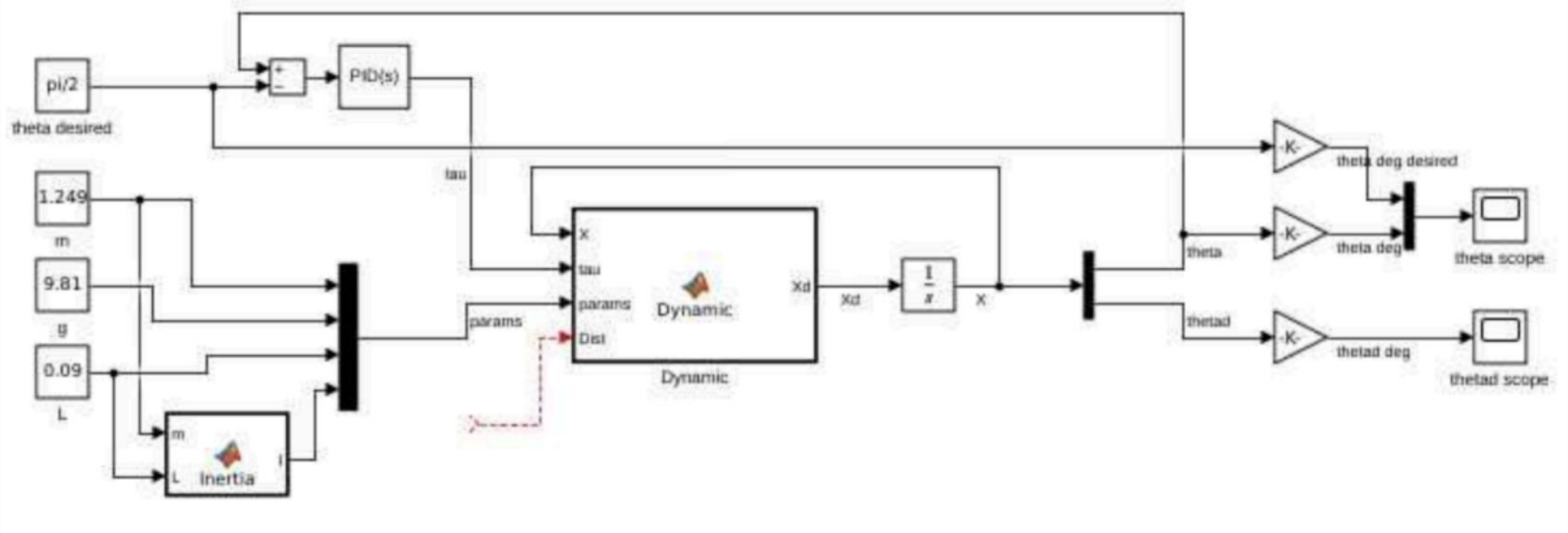
```
void forward(int num, int pwm){
  if(num == A){
    analogWrite(AIN1,255);
    if(turn_L > 0)
      analogWrite(AIN2,constrain(int((255 - pwm)*turn_L), 0, 255));
    else if(turn_L == 1)
      analogWrite(AIN2,constrain(255 - pwm, 0, 255));
    else
      analogWrite(AIN2,constrain(int((255 - pwm)/(-turn_L)), 0, 255));
  }
  else if(num == B){
    analogWrite(BIN1,255);
    analogWrite(BIN2,constrain(int((255-pwm)*turn_R), 0, 255));
  }
}
```

Testing

Sensor Validation

Wheel Characterization

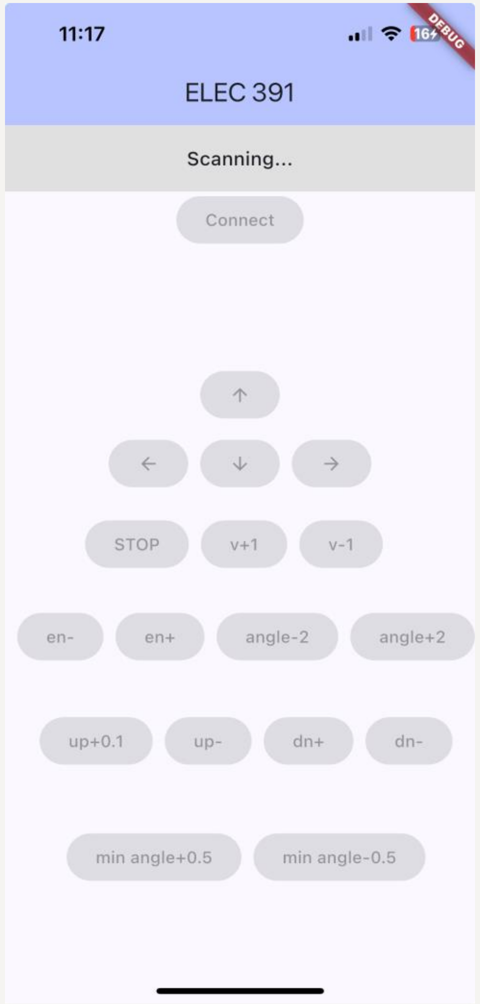
Simulink Modeling

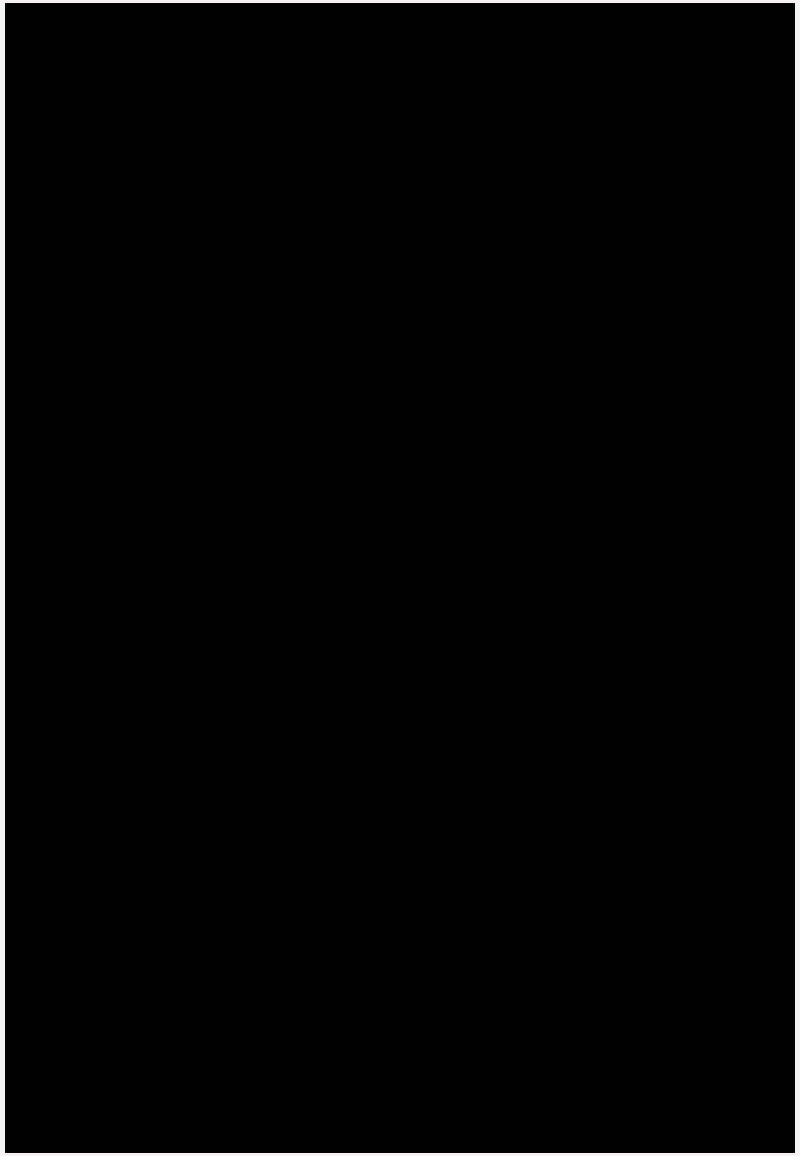


Tuning

- **Flutter app**: Sends real-time commands
- **Serial**: Adjust PID constants

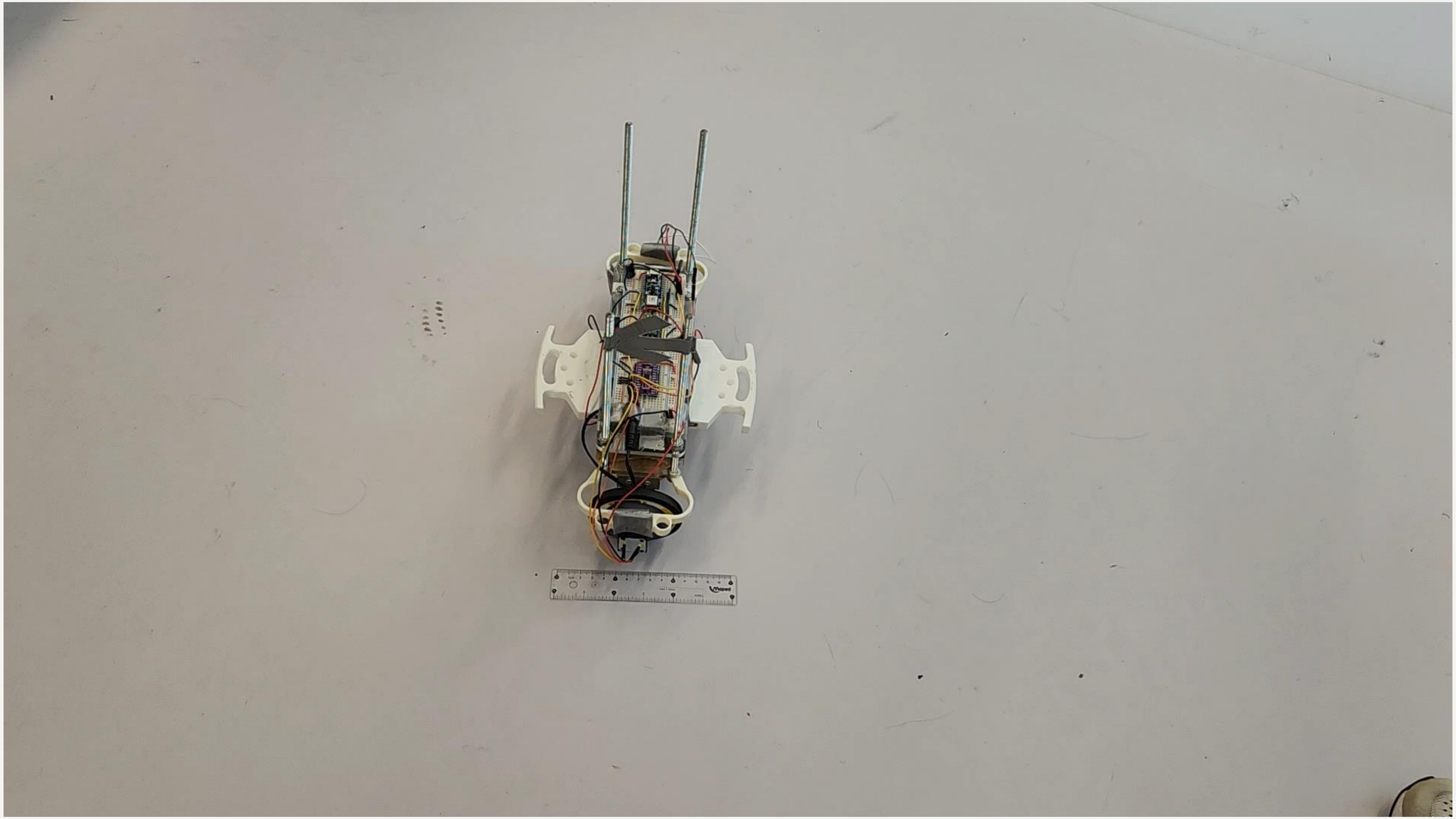
```
else if (strcmp((const char*)receivedString, "b") == 0) {
  turn_L = 1;
  turn_R = 1;
  desired_vel = -5;
  kp_en = -0.1;
}
else if (strcmp((const char*)receivedString, "s") == 0) {
  desired_vel = 0;
  turn_L = 1;
  turn_R = 1;
  kp_en = 0;
  desired_angle = 0.3;
}
else if (strcmp((const char*)receivedString, "p") == 0) {
  desired_vel +=0.1;
}
```





Final Results

- Balanced within ±2 cm (goal: ±4 cm)
- Moved 50 cm & paused correctly
- Turned 45° with minimal overshoot
- Climbed a 10° ramp



Challenges & Lessons

- Offset errors due to battery level, chassis bumps
- Wheel differences required compensation factors
- Final night: loose screw caused slipping → retuned everything at 3 AM

Future Improvements

- Organize and clean up code
- Use timer for autonomous motion

Links:

-Full project
(https://github.com/Rosemarychannan/Self_Balancing_Robot/tree/main)
-Angle test
(https://github.com/Rosemarychannan/Self_Balancing_Robot/tree/main/Starter_Assignment)
-Prototype pwm
(https://github.com/Rosemarychannan/Self_Balancing_Robot/tree/main/Prototype)
-Final code
(https://github.com/Rosemarychannan/Self_Balancing_Robot/tree/main/Final_Product)

THANK YOU