MAE/ECE 5320 Mechatronics
2025 Spring
Final Project Report

(Cover Page)

Balancing Unicycle Robot

Submitted by

Team members:
Will Jenkinson, A02211501
Nate Herbst, A02307138
Trenton Peterson, A02304960
Spencer Stuart, A02306879

Contents

(a template report structure, 8-12 pages, 12 font size, 1-inch margin)

Section 1. Project abstract

The objective of this project is to design and build a self-balancing reaction wheel unicycle that maintains stability using a reaction wheel. Unlike traditional self-balancing vehicles that use direct wheel movement for stability (such as Segways or hoverboards), our system will rely on a high-inertia reaction wheel to counteract disturbances and keep the unicycle upright.

The motivation behind this project is to explore the applications of control systems, dynamic balancing, and embedded mechatronics in a real-world scenario. This project will integrate mechanical design, electrical engineering, and control system implementation to develop a functional prototype.

Below is a conceptual diagram illustrating the project. See Figure 1.

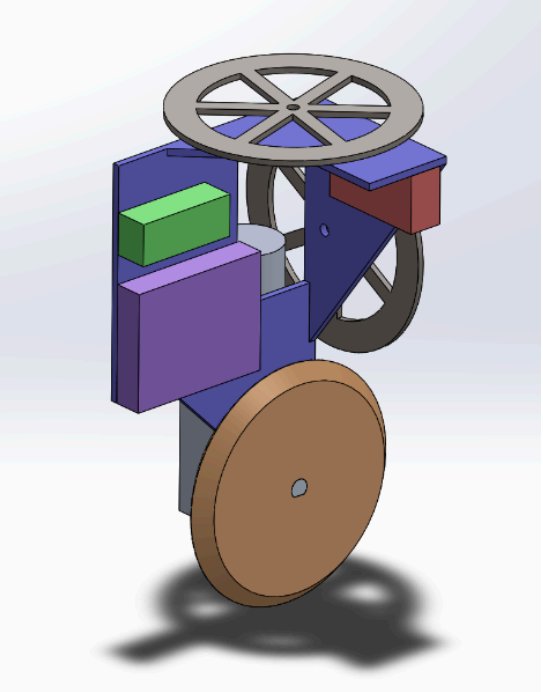


Figure 1: Concept Design

Section 2. Team members and contributions

- **Trenton:** Implementation overview**,** Design, Assembly, and Controls
- **Nate:** Embedded system programming and control algorithm development
- **Will:** Design, Fabrication, Debug
- **Spencer:** Controls, Reporting

Section 3. Project description

So far, the robot has been fully assembled and is currently in the testing phase. The use of simple 3D-printed components streamlined the manufacturing process, allowing us to quickly prototype and iterate on the design. On the software side, development is ongoing. We have created a basic wiring diagram outlining the electrical layout, and we are actively troubleshooting and refining the control logic. As testing continues, we will improve the automation and feedback processes to ensure smooth and stable operation.

A. Mechanical design

This project involves using two high-inertia “flywheel”-type disks to balance a self-righting unicycle robot. The system incorporates several key components that work together to maintain stability and orientation. The flywheels, custom-designed for this project, are 3D printed with embedded pennies to increase mass and, consequently, their rotational inertia. These two spinning disks—mounted orthogonally to the top and back of the device—generate the necessary gyroscopic forces to stabilize the unicycle. These are diagrammed using SolidWorks, as shown in the drawings below. See Figures 2 and 3.

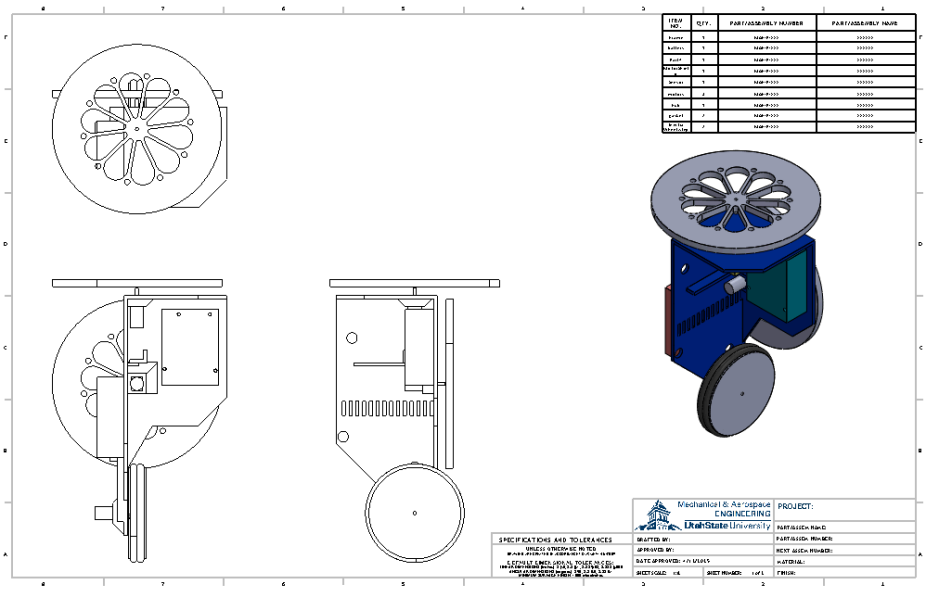


Figure 2: Drawing of Unicycle Robot

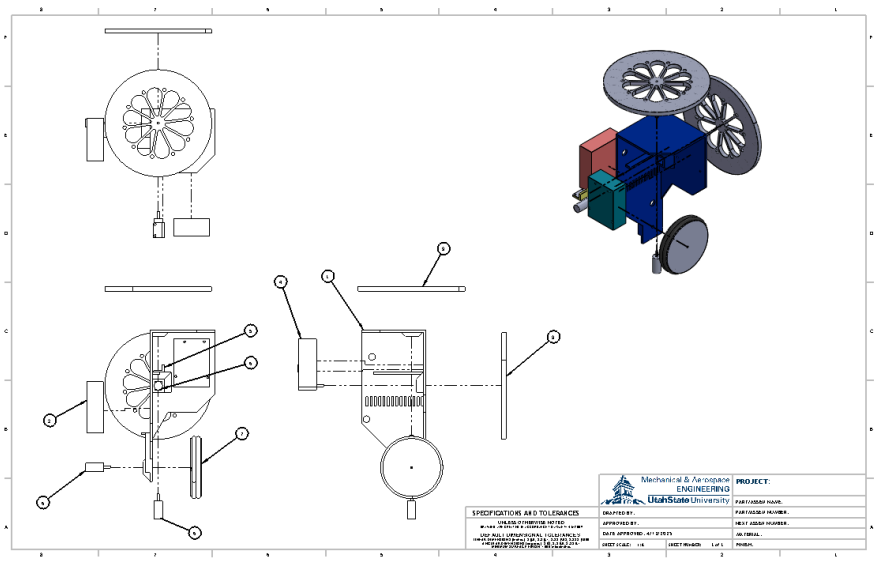


Figure 3: Exploded View Drawing of Unicycle Robot

These components are attached to a 3D printed frame as seen in Figure 4. High precision 3D printers constructed the parts to insure high quality parts and measurements.

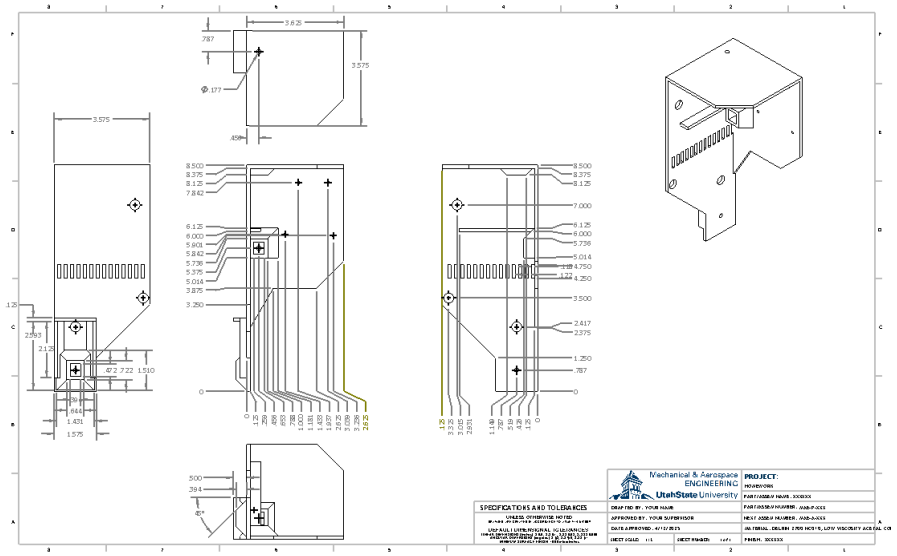
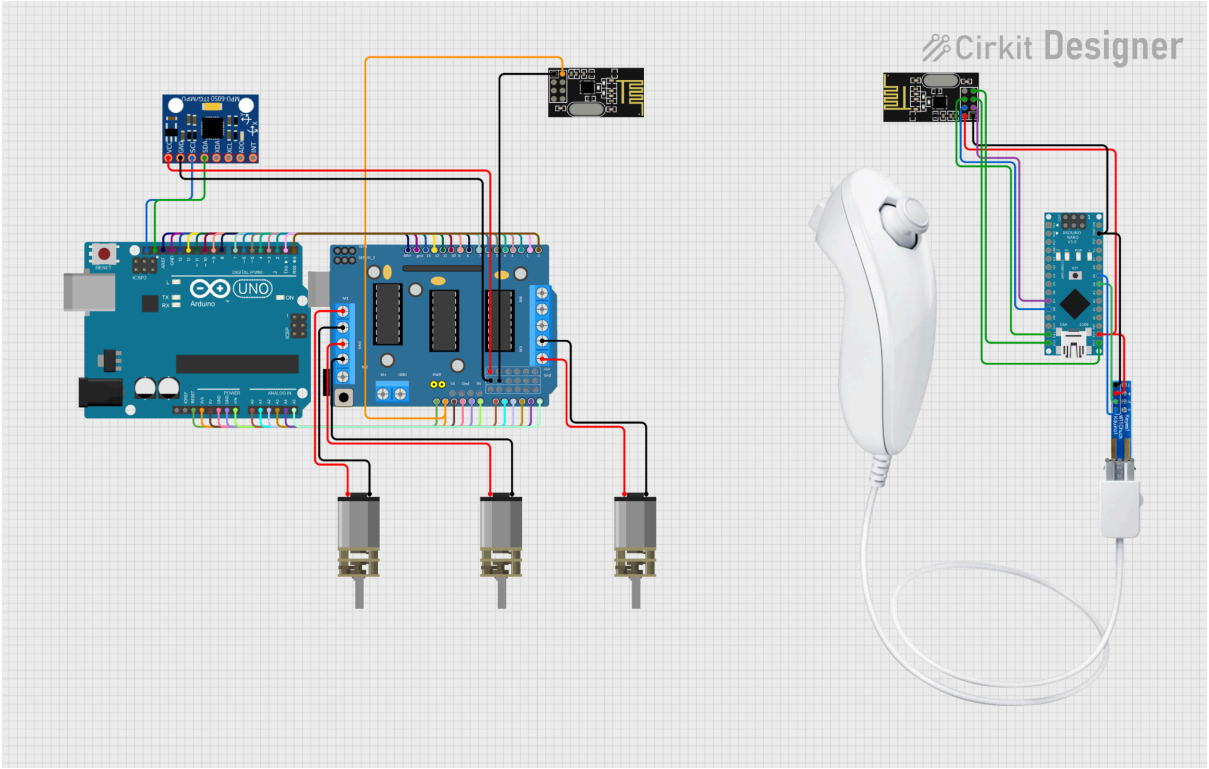


Figure 4: Robot Frame Drawing

B. Electrical design



C. Hardware selections

- Arduino Uno
- 6050 MPU
- L263D DC Motor Drive, Arduino Motor Shield
- 12V 200-RPM N20 DC Motors
- 12V 1500mAh Li-ion Battery

D. Closed-loop control system

A closed-loop control system governs the speed and direction of the flywheels, using real-time feedback to maintain balance and keep the robot upright. Looking ahead, we plan to implement a Wii remote interface via wireless communication, enabling user control to navigate the robot through a randomized course.

```
#include <Wire.h>
#include <AFMotor.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <math.h> // For atan2 and pow

#define SPEED_BUFFER 0

AF_DCMotor motorX(1); // Motor on M1 for X-axis correction
AF_DCMotor motorY(4); // Motor on M2 for Y-axis correction
```

```
Adafruit_MPU6050 mpu;

// PID Constants (tune these!)
double KpX = 2.0;
double KiX = 0;
double KdX = 0.1;

double KpY = 1.0;
double KiY = 0;
double KdY = 0;

// PID Variables
double setpointX = 0;
double setpointY = 0;

double inputX, inputY, outputX, outputY;
double errorX, errorY, lastErrorX, lastErrorY, integralX, integralY,
derivativeX, derivativeY;

unsigned long lastTime;

void setup() {
 Serial.begin(115200);
 Serial.println("MPU6050 with PID Motor Control");
 Wire.begin();

 if (!mpu.begin()) {
   Serial.println("Failed to find MPU6050 chip");
   while (1) {
     delay(10);
   }
 }
 Serial.println("MPU6050 Found!");

 mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
 Serial.println("MPU: Accelerometer range set to: +-8G");
 mpu.setGyroRange(MPU6050_RANGE_500_DEG);
 Serial.println("MPU: Gyro range set to: +- 500 deg/s");
 mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
 Serial.println("MPU: Filter bandwidth set to: 21 Hz");
 Serial.println("");

 motorX.setSpeed(0);
 motorX.run(RELEASE);
 motorY.setSpeed(0);
 motorY.run(RELEASE);

 lastTime = millis();
}

void loop() {
 sensors_event_t a, g, temp;
 mpu.getEvent(&a, &g, &temp);

 // Calculate tilt angles (in degrees)
 inputX = atan2(a.acceleration.y, sqrt(pow(a.acceleration.x, 2) +
pow(a.acceleration.z, 2))) * 180 / M_PI;
```

```
 inputY = atan2(-a.acceleration.x, sqrt(pow(a.acceleration.y, 2) +
pow(a.acceleration.z, 2))) * 180 / M_PI;

 Serial.print("Angle X: ");
 Serial.print(inputX);
 Serial.print(" degrees --- ");
 // Serial.println("");

 // Run the PID control for both axes
 controlPID(inputX, setpointX, KpX, KiX, KdX, errorX, lastErrorX,
           integralX, derivativeX, outputX, motorX, true);

 Serial.print("Angle Y: ");
 Serial.print(inputY);
 Serial.print(" degrees --- ");
 // Serial.println("");

 controlPID(inputY, setpointY, KpY, KiY, KdY, errorY, lastErrorY,
           integralY, derivativeY, outputY, motorY, false);

 lastErrorX = errorX;
 lastErrorY = errorY;

 // Limit integral term to prevent windup
 integralX = constrain(integralX, -100, 100); // Adjust limits as
needed
 integralY = constrain(integralY, -100, 100);

 delay(10); // Adjust delay as needed
}

// PID control function
void controlPID(double input, double setpoint, double Kp, double Ki,
double Kd,
               double &error, double &lastError, double &integral,
double &derivative,
               double &output, AF_DCMotor &motor, bool flipped) {

 unsigned long now = millis();
 double timeChange = (double)(now - lastTime);

 error = setpoint - input;
 integral += error * timeChange / 1000.0; // Integral over time
 derivative = (input - lastError) / (timeChange / 1000.0); // Rate of
change

 output = Kp * error + Ki * integral + Kd * derivative;

 // Limit the output to the motor's speed range (0-255)
 int motorSpeed = constrain(abs(output), 0, 225);
 Serial.print("MotorSpeed: ");
 Serial.println(motorSpeed);

 if (output > 0 + SPEED_BUFFER) {
   if (flipped) motor.run(FORWARD);
   else motor.run(BACKWARD);
   motor.setSpeed(motorSpeed);
 } else if (output < 0 - SPEED_BUFFER) {
```

```
    if (flipped) motor.run(BACKWARD);
    else motor.run(FORWARD);
    motor.setSpeed(motorSpeed);
  } else {
    motor.run(RELEASE);
    motor.setSpeed(0);
  }
  lastTime = now;
}
```

Section 4. Results

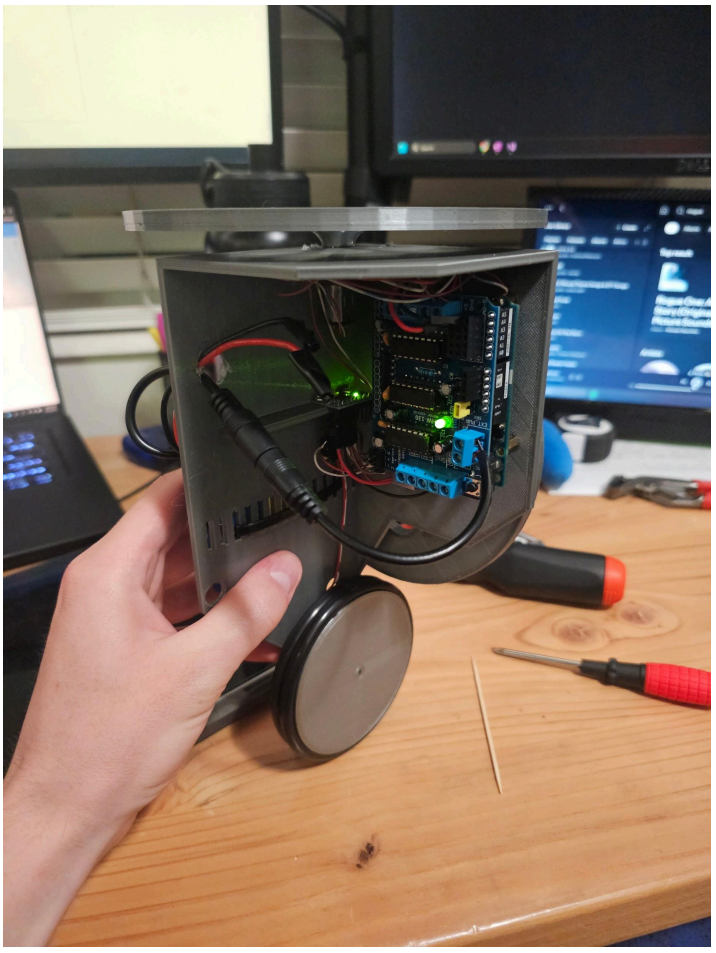


Figure 5: Current RobotDesign

Robot currently responds to different angles other than level correctly but rhe PID tuning is not intense enough for the time being.

Section 5. Conclusions