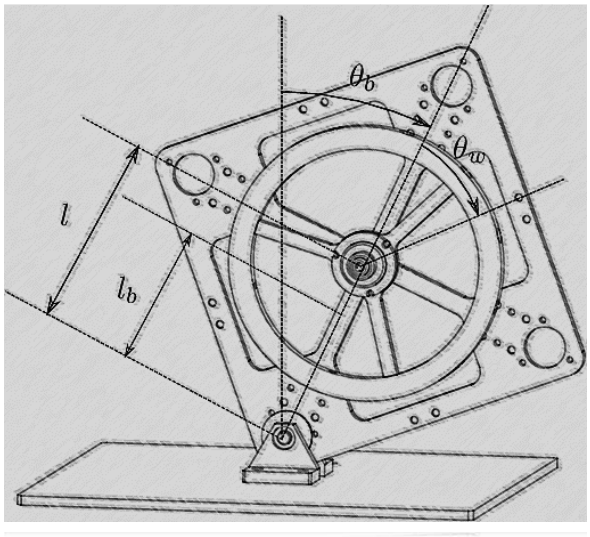
Self-Balancing: 1D Reaction Wheel

Rawan Mallah, Omar Kassar



12/5/2023

Table of Contents

I. Project Description

i. Abstract

The concept of self-balancing systems has gained significant traction in various fields, ranging from robotics to aerospace applications. Designing autonomous vehicles, spacecraft, and robotic platforms is impacted by the fundamental difficulty of achieving stability along a single axis. This project's self-balancing 1D reaction wheel system is a demonstration of how creatively contemporary electronics, control systems, and programming were used to meet this issue.

ii. Introduction

This project studies the design and execution of a self-balancing 1D reaction wheel system with a variety of parts. The aim of this project is to link precise motor control with sensor data to provide an efficient and effective solution for preserving stability along a single axis.

The primary parts are a BLDC (Brushless) motor, an MPU9250 (gyroscope) sensor, an Arduino Uno microcontroller, and a power supply. essential inertial data is supplied by the MPU9250, and the Arduino Uno interprets this data to operate the BLDC motor and stabilize the system.

iii. Hardware Components

- **Arduino Uno:** functions as the central processing unit, handling algorithms for motor control and data processing and collection.

Figure.1: Arduino Uno

- **MPU9250 Inertial Measurement Unit (IMU):** a sensor that integrates a magnetometer, gyroscope, and 3-axis accelerometer. It provides the change in angle over for the real-time state estimate of the system.

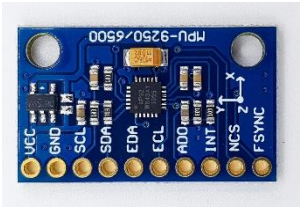


Figure.2: MPU9250

- **BLDC Motor:** The brushless DC motor serves as the response wheel and produces torque which supports the system balance.

Figure.3: BLDC Motor

- **Power Source:** Provides the BLDC motor with the electrical power it needs to operate.

Figure.4: power supply

II. Functional Specifications

i. How this project is applied in other applications

One-dimensional self-balancing systems, such as reaction wheel systems, have applications in various fields, particularly in situations where one-axis stability is essential. The following are some significant application examples of 1D self-balancing mechanisms:

- **Quadcopters and Drones:**
  Gyroscopes and accelerometers are frequently used by drones to maintain steady flight and change their orientation in response to human orders or external conditions.
- **Personal Robots and Assistants:**
  1D self-balancing systems might be used in home robotic assistants or personal robots to increase stability and agility. They might be able to move around interior spaces more skillfully and complete activities more precisely.

- **Home Automations:**

  Smart home devices, such as cameras, could benefit from 1D self-balancing mechanisms to ensure a stable view for monitoring purposes. This could be especially useful in security cameras or baby monitors.

- **Educational Tools:**

  One use for 1D self-balancing systems might be instructional robots or gadgets. Not only would this raise their level of involvement, but it would also give users practical training on stability and control concepts.

- **Satellite and Spacecraft Control:**

  Satellites and spacecraft frequently use reaction wheels to adjust their orientation in orbit. The satellite can counteract external torques and maintain a certain orientation for accurate data collecting or communication by altering the speed of the response wheel.

ii. Weight

Our project with its base, without its power supply, is: 700g.

iii. Dimensions

Base Length:

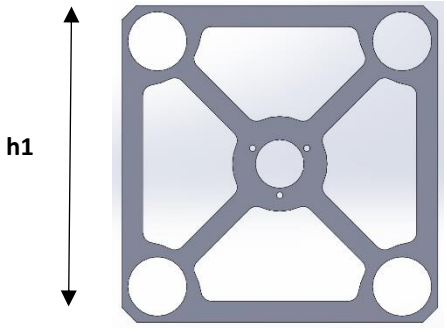


Figure.5: Square

h1=15cm

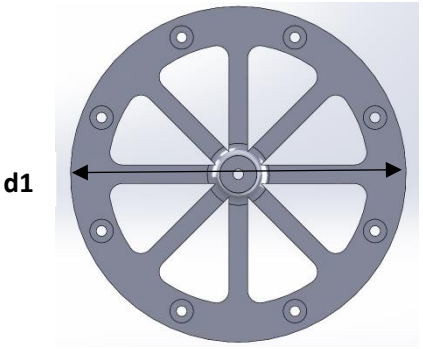


Figure.6: wheel

d1=13.5cm

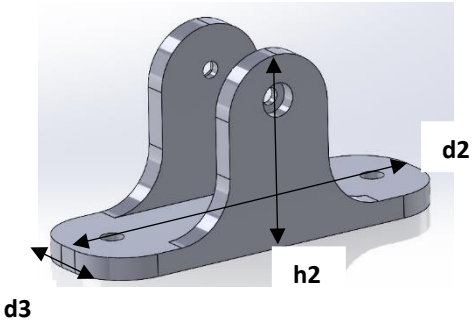


Figure.7: pin

h1=3cm

d2=7cm

d3=2.4cm

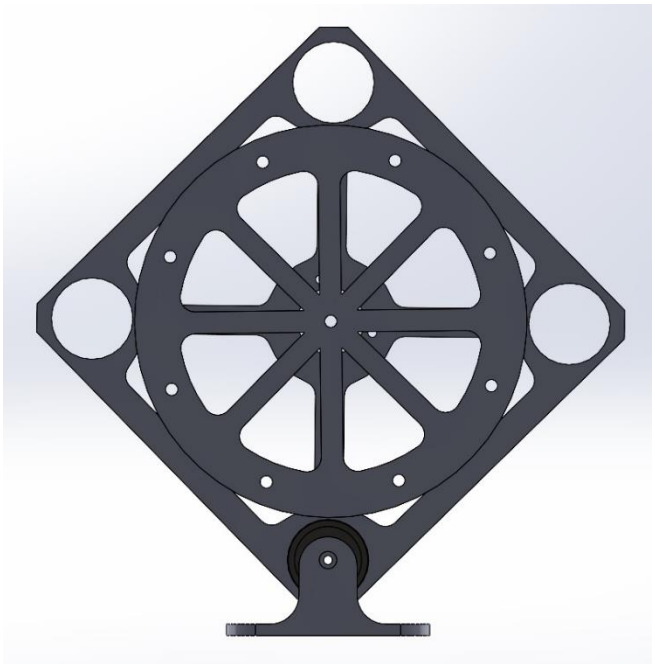


Figure.8: final robot

iv. Mass needed to lift the square:

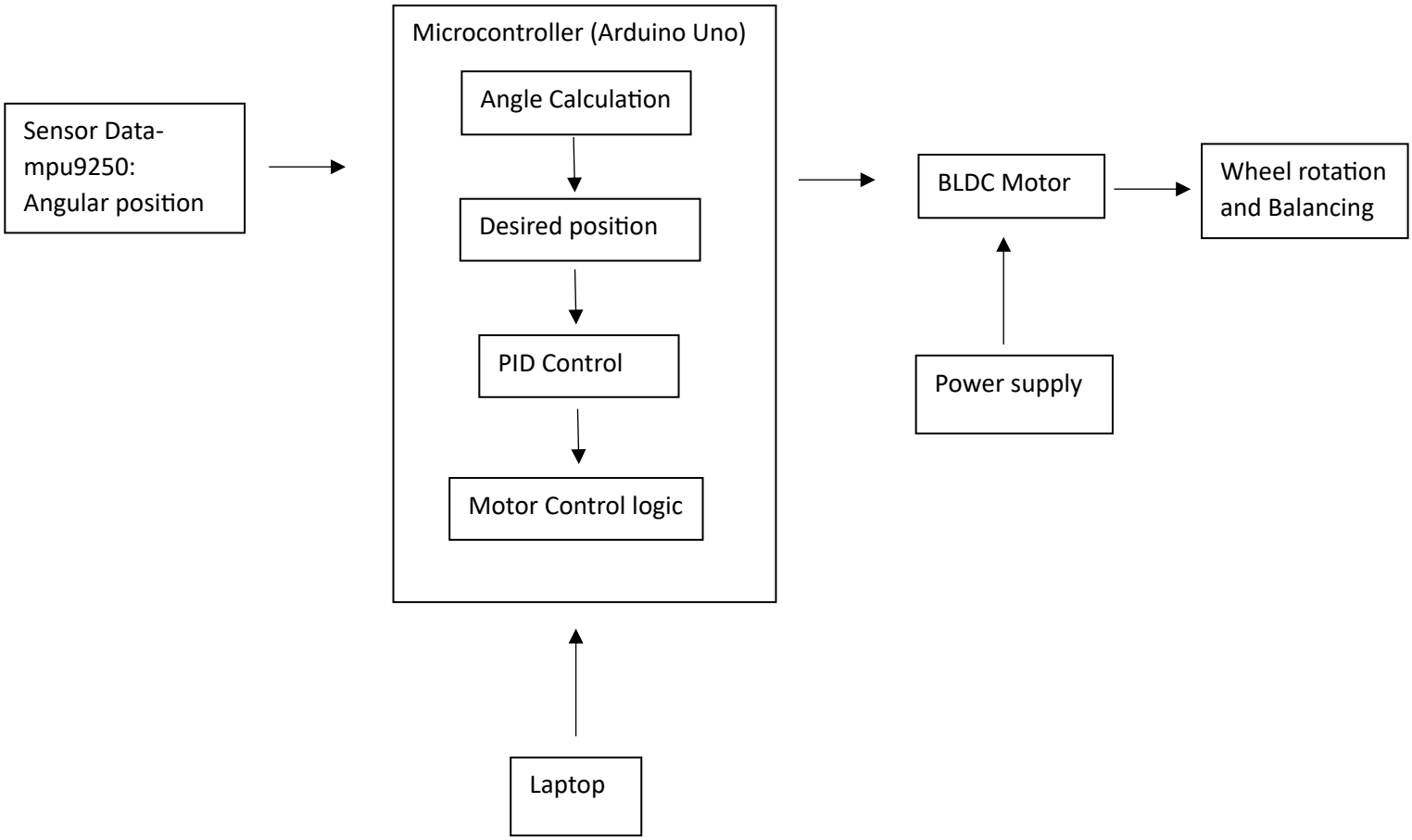
The mass added to the wheel to lift the square is equal to10g.

v. power supply of the project

- Our project gets its power from a power supply labeled 220V AC to 12V DC with 3A.
- <u>The motors' wires are connected as follows:</u>
    1. One wire to 12V from the power supply and one to the 5V on the Arduino.
    2. One GND wire to the GND on the Arduino.
    3. PWM wire to the PWM pin (9) on the Arduino.
    4. Brake wire to pin 8 on the Arduino.
    5. DIR wire for the direction of rotation is connected to pin 7 on the Arduino.

- The MPU9250 is connected as follows:
    1. Positive to the 5V on the Arduino.
    2. GND to the GND on the Arduino.
    3. SDA to pin A5 on the Arduino.
    4. SCL to pin A4 on the Arduino.

III. Block Diagram



IV. Principle Of Operation

Our 1D reaction wheel self-balancing project operates on the basis of coordinating the MPU9250, PID control algorithm, and motor to sustain balance along a single axis.

The following is a detailed explanation of the principle of operation:

i. Sensor Data Collection:

the MPU9250 is in charge of continuously determining the angular position and velocity along the selected axis (1D). It incorporates data from a 3-axis accelerometer, gyroscope, and magnetometer to provide accurate orientation information. The real-time angular position data is then added or subtracted from the pre-set setpoint, and the result is what our system interprets as feedback. When this practice is applied, the square would remain precisely balanced at its setpoint.

ii. Feedback Processing and PID Control:

The Arduino Uno, which is the central unit of our project, is responsible for receiving the data from the sensor and then interpreting it to send the right feedback for the motor to act respectively.

Based on the feedback from the MPU9250, the PID (Proportional-Integral-Derivative) control algorithm is used to determine the proper measure to take. Three primary parts are employed by the PID controller:

- **Proportional (P):** Responds to the present error (difference between intended and actual angle).
- **Integral (I):** Considers how errors from the past compound over time.
- **Derivative (D):** forecasts future mistakes based on the present rate of change.

The system operates in a closed-loop configuration, with a sensor continuously measuring the system's state and providing feedback to the PID controller. This feedback mechanism ensures the system constantly makes corrections to counteract disturbances or changes in orientation. The iterative process allows the system to adapt and maintain balance as external forces or disturbances act on it.

iii. Motor Control:

The PID controller's output indicates the necessary corrective action to keep the system in a balanced condition. The BLDC motor is then controlled by means of this data. In order to counteract the observed departure from the required angle, the motor's speed and direction are modified. To change the direction of rotation, the DIR pin is given LOW to turn clockwise and HIGH to rotate counterclockwise.

iv. Maintaining Balance:

When the system deviates from the required angle, the PID controller determines and executes the appropriate motor adjustments to return the system to a balanced state. Because of its dynamic response, the 1D reaction wheel can withstand external stresses while remaining stable.

In brief, our research integrates sensor data, PID control, and motor input to construct a closed-loop control system which alters the motor's operation autonomously to maintain the 1D response wheel balanced. As a result, the system self-stabilizes and can respond to changes in its orientation in real time.

V. Complete Schematic Diagram

We used a circuit with the same system and concept as our hardware project in our simulation. The direction and speed of rotation of the DC motor are determined by the error values computed by the Arduino following a change in the sensor's temperature. Since 25 degrees is the set point, the motor must be at rest. The motor will accelerate and rotate in a different direction in response to changes in temperature beyond 25 degrees. Thus, when the temperature exceeds 25 degrees, the motor will rotate clockwise, and when its below 25 degrees, the motor would rotate counterclockwise.

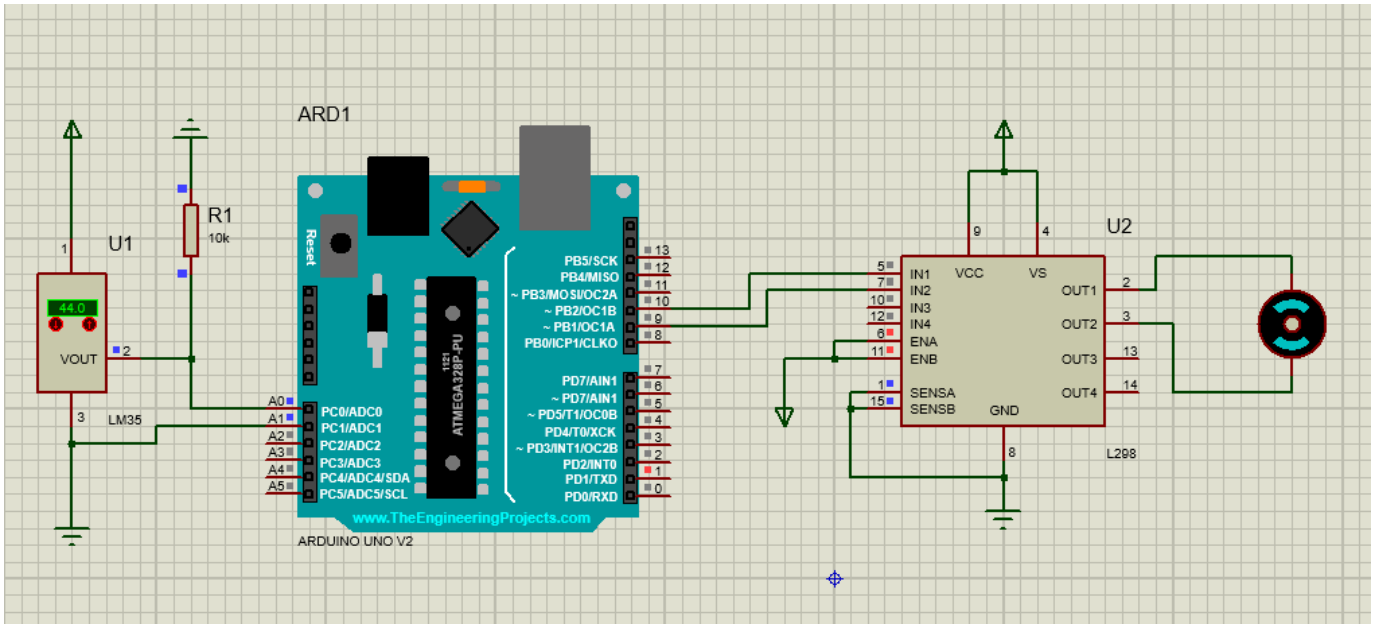


Figure.9: Schematic Simulation

VI. Bill of Quantities

The number and cost of each component used in this project are listed below:

i. One Arduino Uno board

ii. One BLDC motor

iii. One MPU9250 Sensor

iv. One Power Supply

v. Eleven Wires

vi. PlA plastic

VII. Results and Conclusion

i. Results:

The 1D reaction wheel self-balancing project delivered promising results, demonstrating the accuracy of the integrated system. The MPU9250 delivered constant accurate sensor data, allowing the PID controller to perform exact calculations based on the angular displacement from the intended balanced position. In response to the controller's output, the BLDC motor dynamically generated torque to rapidly eliminate deviations and preserve stability along the intended axis. Through considerable testing and study, the system demonstrated its ability to effectively self-balance in the face of external interruptions. The project's real-time adaptability is proven by the rapid reaction time of the closed-loop control system. The total performance surpassed goals, showcasing the great integration of motor response, PID control, and sensor feedback in achieving the project's primary goal of 1D self-balancing.

ii. Conclusion:

Through the collaboration between MPU9250, Arduino Uno, and BLDC motor, the 1D reaction wheel self-balancing project demonstrates how sensor fusion and control theory are used in real-world applications. The PID control algorithm exhibits its effectiveness in practical circumstances by offering resilient and flexible means of preserving equilibrium. The study demonstrated the broad scope of robotics and control engineering by successfully achieving the technical goal of balancing the system along a single axis. The knowledge acquired can aid in the creation of more sophisticated systems, possibly branching out into multi-axis stabilization or finding uses in automation, robotics, and other fields.

VIII. Appendix

Real Hardware Code:

```
#include "MPU9250.h"

#define DIR 7
#define BRAKE 8
#define PWM 3

double dt, last_time;
double input, output, integral, previous, error;
const double Setpoint = 3.5;

//Specify the links and initial tuning parameters
double kp = 40, ki = 0.5, kd = 2;  //38-0.5-2 or 40-0.5-2   100-5-10  130-6-12

MPU9250 mpu;

void setup() {

  TCCR2B = TCCR2B & B11111000 | B00000001;

  Serial.begin(115200);
  Wire.begin();

  pinMode(PWM, OUTPUT);
  pinMode(DIR, OUTPUT);
  pinMode(BRAKE, OUTPUT);

  input = mpu.getPitch();

  digitalWrite(BRAKE, HIGH);
  analogWrite(PWM, 0);

  delay(2000);

  if (!mpu.setup(0x68)) {
    while (1) {
      Serial.println("MPU connection failed. Please check your connection with
connection_check example.");
      delay(5000);
    }
  }
```

```
}

void loop() {

  input = mpu.getPitch();

  double now = millis();
  dt = (now - last_time) / 1000.00;
  last_time = now;

  input = mpu.getPitch();


  error = Setpoint - input;
  output = pid(error);

  if (output < 0) {
    digitalWrite(DIR, HIGH);
    output = -output;
  }
  else
    digitalWrite(DIR, LOW);


  if (output >= 255) {
    output = 255;
  }

  output = map(output, 0, 255, 255, 0);

  TCCR2B = TCCR2B & B11111000 | B00000001;
  analogWrite(PWM, output);

  if (mpu.update()) {
    static uint32_t prev_ms = millis();
    if (millis() > prev_ms + 25) {
      print();
      prev_ms = millis();
    }
  }
}


void print() {
  Serial.print("Pitch angle: ");
```

```
  Serial.print(input, 2);
  Serial.print("    ;error: ");
  Serial.print(error, 2);
  Serial.print("    ;output: ");
  Serial.println(output, 2);
}

double pid(double error) {

  double proportional = error;
  integral += error * dt;
  double derivative = (error - previous) / dt;
  previous = error;
  double output = (kp * proportional) + (ki * integral) + (kd * derivative);
  return output;
}
```