Automated Seesaw

EE333 - Control Systems

17 May 2025

Juan Ali Ruiz Guzman

Abstract:

This project covers the experimentation and implementation of control systems. Implementing a control system is meant to mitigate errors and noise within systems, and the system I chose as a base for this project was a brushless motor-controlled seesaw. The original system uses an Arduino to control the brushless motor and a potentiometer to adjust the speed of the brushless motor. This project aimed to automate the seesaw and implement a control system to help reduce errors and unwanted oscillations within the system. The idea behind this was to add an accelerometer to the system as an input in place of the potentiometer so that the system can adjust the brushless motor's speed constantly to maintain a set level height/angle. The control system that was implemented was a PID controller. The project was successful in its implementation of the accelerometer as an input, where it measures the lever arm fully down as -30 degrees from level and returns live updates in angle based on where the lever is as it moves up and down. The PID controller was successfully implemented as well by providing a quick response to any changes in angle, whether they be from external movement or slight changes in angle readings from the accelerometer. The PID controller played a major role in the dampening of the system and reduced excessive overshoot. The system achieved real-time balance within ~±1° of level. Challenges included motor power scaling, sensor noise, and power regulation.

Introduction:

The system used in this project was a single-sided balancing seesaw. This system had some challenges that included being nonlinear, unstable, open-loop, underactuated, and underdamped. Thus, the purpose and main goal were to stabilize the seesaw using feedback control with the accelerometer and to dampen the system with a control system. This kind of system and implementation of feedback control and control systems is relevant to applied control systems, embedded systems, and sensor-actuator integration.

System Description:

Physical system

The seesaw in this system consisted of a base plate, a post, a lever arm, a hinge/pivot, and a motor mount.

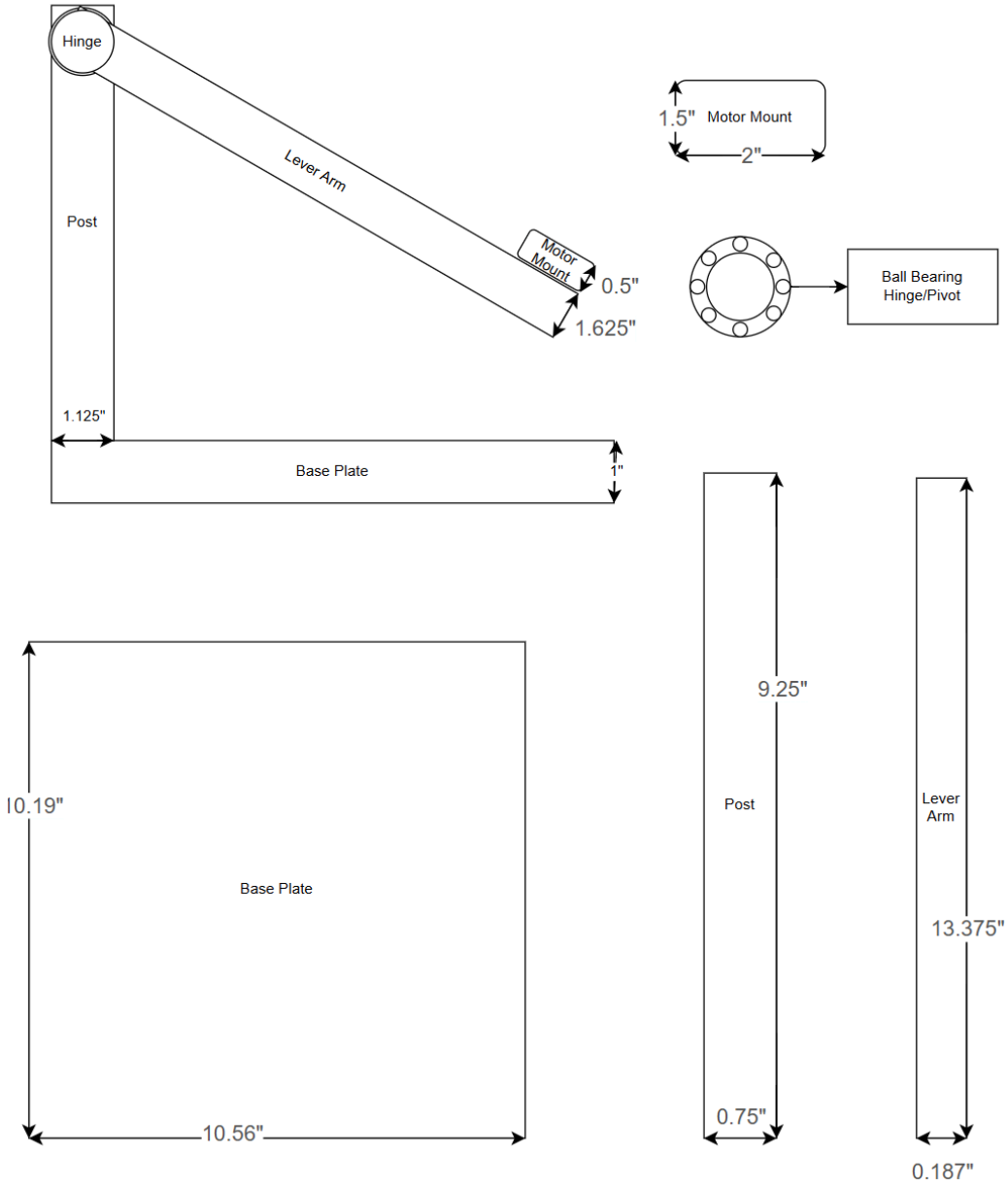

Figure 1: Seesaw Diagram and Dimensions

Sensors and Actuators

Table 1: Component Table

Component	Type	Role
GY-521 MPU-6050	Accelerometer/Gyro	Measure angle
A2212 1400KV Brushless Motor	BLDC Motor	Adjust the lift force
30A/50A ESC BEC	PWM motor driver	Convert signal to motion

ELEGOO MEGA R3 Board ATmega2560	Controller	Run PID & interface
8060 Propeller	Propeller	Works with the Motor to Lift
OVONIC 3s Lipo Battery	Battery	Supplies Power
20k Potentiometer	Tuning Resistor	For Live PID Tuning
D4 Pin Button	Button	Auto/Live PID Tuning
D5 Pin Button	Button	PID auto-tune trigger
D6 Pin Button	Button	Stop button
D7 Pin Button	Button	Shutdown button

System Modeling

System Assumptions:

- Negligible Friction
- Negligible Air Resistance
- Negligible ESC noise

Inputs of the system:

- D4 PID function button
- D5 Run PID auto tune button
- D6 stop button
- D7 system shutoff button
- GY-521 MPU-6050

Outputs of the system:

- Serial communication values of the system
- Brushless motor

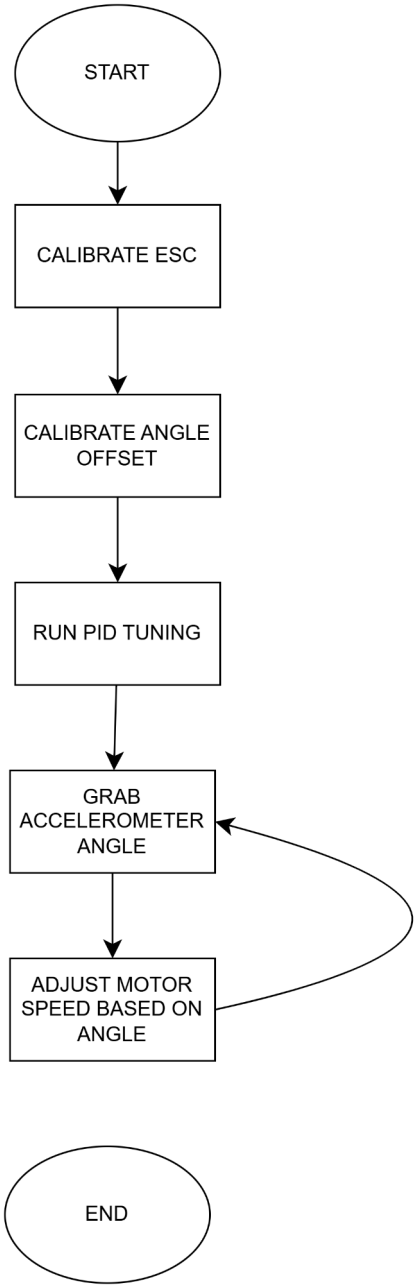

Figure 2: System Flowchart

Control Objectives:

With the original implementation of the system, which had the potentiometer control the brushless motor speed, Manual Adjustments with the Potentiometer resulted in 3 3-second rise time on average. Once the system was successfully implemented, the goal for the control system was to reduce the settling time to less than 2 seconds, maintain overshoot at less than

12%, and reduce the steady state error to below 5%. The importance of these goals was to allow for smoother adjustments to the brushless motor speed. This would also reduce unwanted oscillations and oscillation amplitudes in the system.

Control Design:

Controller Selection

Because the system is rather simple and requires rapid correction of angle errors, a classic PID controller was selected. The proportional factor reacts to the current angle error immediately. The Integral factor accounts for accumulated error over time and helps eliminate steady-state error. Lastly, the Derivative factor predicts future error based on how fast the angle is changing. This adds dampening to prevent overshoot and oscillations.

Controller Implementation

The initial idea for implementing the PID controller was live tuning, where 3 potentiometers control each of the components of the PID controller. The issue with live tuning at the early stages was that it was very difficult to tune while figuring out the right PWM ranges for the motor. This led to the implementation of the PID controller through the Arduino code. The code performs a change in motor speed and then, based on how the motor reacts, it tunes the values for the PID components and stores them.

In the `loop()`, the system continuously:

- Reads the current tilt angle from the MPU6050
- Filters the signal to reduce noise
- Computes the error between the desired level (0 degrees) and the actual angle
- Applies the PID formula to calculate a correction value
- Writes a corresponding PWM value to the ESC (range 50–60)

Controller Tuning

Auto-tuned gains were:

- Kp = 0.8
- Ki = 0.01
- Kd = 0.2

The effect of tuning was observed via real-time plots of angle vs PWM, which confirmed that proper tuning resulted in stable convergence with minimal overshoot.

Closed-Loop Performance

In practice, the system was able to stabilize within one degree of the level point after being released from any tilt. Rise time was under 3 seconds with minimal oscillations, and no overshoot beyond 1-2 degrees was observed. Settling time was around 3-4 seconds on average, depending on the disturbances forced on the system by external factors like a hand.

Hardware Implementation and Results:

Hardware Setup

The hardware system consists of the following key components:

- **Arduino Mega 2560** – Handles all sensor reading, PID computation, and PWM output.
- **MPU6050 Accelerometer** – Mounted near the center of the seesaw arm to read tilt angle.
- **A2212 1400kV Brushless Motor + Propeller** – Attached to the end of the arm to apply lifting force.
- **50A ESC** – Controls motor speed through Arduino's PWM signal.
- **11.1V 3S LiPo Battery** – Supplies adequate current and voltage for the BLDC motor.
- **Tactile Pushbuttons (D4–D7)** – Used to enable/disable motor, initiate auto-tune, stop motor, and shutdown system.

The accelerometer is wired to the analog pins A4 and A5 (SDA/SCL) while the ESC is wired to D9, which sends the PWM signal.

Results

Upon startup, the system performs ESC calibration, offset angle normalization, and performs PID autotuning. Based on the tilt error, the motor spins up, and the PID controller modulates the PWM signal in real time. The system stabilizes the seesaw after startup at 59-60 PWM and will decrease the PWM as the angle increases and increase as the tilt angle decreases.

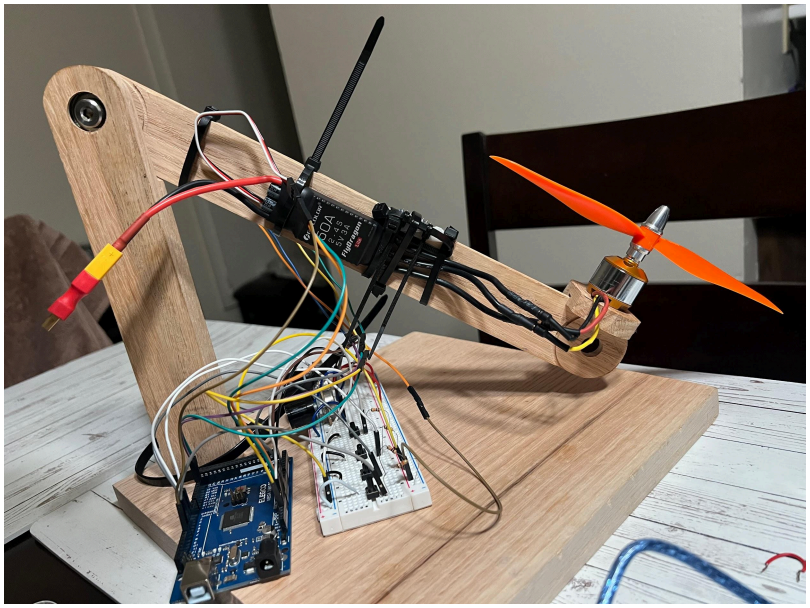

Figure 3: Physical system

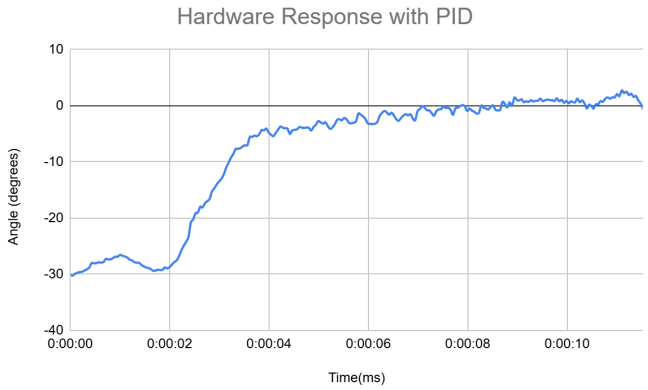

Figure 4: Plot with PID controller

Demonstration

For the demonstration. The Arm is held down to tune PID, calibrate ESC, and perform the offset angle normalization. Then the arm is released, and the system lifts the arm to level and stabilizes. In the demo, the arm is lifted above the level, and the motor decreases speed. Once the arm is released the arm dampens the fall and re-levels itself. The arm is lifted and released once more to confirm the behavior and then pushed down to -30 degrees once more to show the system is fully functional and working. A plot of the leveling from -30 degrees is shown in Figure 4. More pictures of the system can be found in Appendix B.

Conclusion and Future Work

This project successfully demonstrated the design and real-time implementation of a self-balancing seesaw system using a PID controller on an Arduino platform. The combination of an MPU6050 accelerometer sensor for feedback and a brushless motor controlled through a 50A ESC allowed for responsive and stable control over the system's angular position. The integration of auto-tuning, real-time data logging, and hardware interactivity resulted in a robust experimental platform. The system could balance from a 30° starting tilt to within ±1° of the horizontal level. Key challenges such as motor overshoot, noisy sensor data, and power instability were effectively addressed with signal filtering, PWM scaling, and careful hardware selection.

Valuable insights from embedded systems, PID controllers, and sensor-actuator integration were gained through this experiment. The design and development process highlighted the importance of incremental testing, proper parameter constraints, and hardware safety in real-world control systems.

Challenges faced during the project's development were: PWM tuning, PID tuning, and Accelerometer tuning. Because this was the first time I had worked with BL motors, calibrating the ESC was something new, and while calibration was always successful. I didn't realize that I was then writing full power values to the ESC after calibrating, which caused the motor to immediately spin at full speed. This resulted in the seesaw going to 180 degrees from level. This was successfully resolved by later writing a much smaller PWM value to the ESC after calibration during startup sequences. PID proved difficult in the early stages of the project because of a faulty battery and ESC in the system. Live tuning with potentiometers resulted in negligible changes in the system behavior. Once the faulty hardware was replaced, the live tuning then had a noticeable effect. When swapping the hardware, the automated PID code values were still set at Kp = 3, and turning the system back on caused the motor to jump to extremely high speeds, resulting in the propeller smashing into the table and splitting apart. Lastly, Accelerometer readings proved challenging to read if the arm was not held at the level during startup. This challenge was addressed by adjusting the code so that during startup, the arm is held down and the angle is set to -30 degrees, as it should be read.

Future work could expand the system to support dual-axis balancing using a second actuator and gyroscope data. Additional improvements might include:

- Implementing a complementary or Kalman filter for more accurate orientation tracking
- Adding a wireless module for remote data logging and control
- Integrating a graphical user interface for live tuning and monitoring
- Add adaptive tuning for incrementing the weight to the arm during operation
- Implementing the system with a different microcontroller, such as the PIC18F47K42

This project lays a strong foundation for further research and application in dynamic stabilization systems, autonomous robotics, and control education platforms.

References

Special Thanks to How To Mechatronics for the background information and inspiration for this project!

https://howtomechatronics.com/tutorials/arduino/arduino-brushless-motor-control-tutorial-esc-bldc/

- Wire.h Library
- MPU6050.h Library
- Servo.h Library
- ChatGPT for aide in PID Controller Code

Appendix

A System Figures:

Figure 5: ELEGOO MEGA R3 Board ATmega2560

Figure 6: A2212 1400KV Brushless Motor

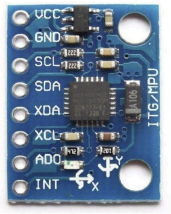

Figure 7: GY-521 MPU-6050

Figure 8: 8060 Propeller

B Hardware Results And Figures:

Figure 10: Example of too much overshoot

Figure 11: Broken Propeller from incorrect PID values

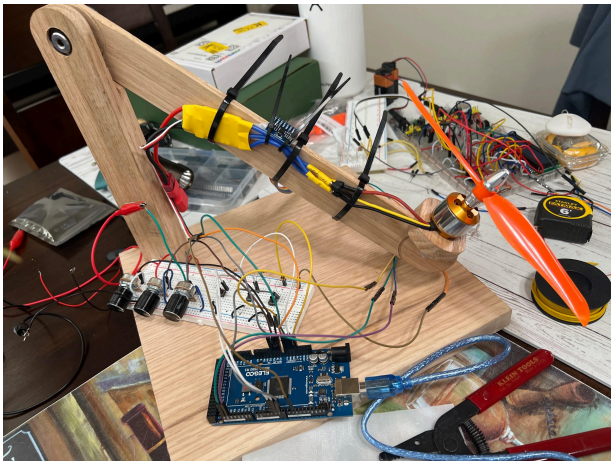

Figure 12: System halfway through the project with faulty hardware

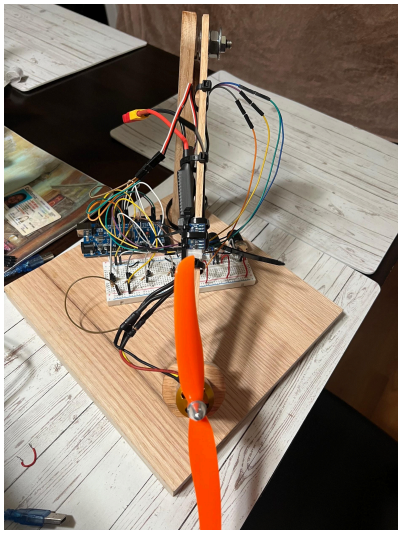

Figure 13: Top View of Final System

C Full Code:

```
#include <Wire.h>
#include <MPU6050.h>
#include <Servo.h>

MPU6050 mpu;
Servo esc;

int16_t ax, ay, az;

// PID variables
float Kp = 1.5, Ki = 0.5, Kd = 0.05;
float integral = 0;
float lastError = 0;
unsigned long lastTime;

float angleFiltered = 0;
const float alpha = 0.95;
float setpoint = 0.0;

const int escPin = 9;
const int switchPin = 4;
const int tuneSwitchPin = 5;
const int stopButtonPin = 6;
```

```
const int shutdownButtonPin = 7; // NEW shutdown button

const int ESC_MIN = 50;
const int ESC_MAX = 60;
const float ANGLE_LIMIT = 35.0;

bool motorEnabled = false;
bool autoTuned = false;
bool systemStopped = false;
bool systemShutdown = false;

float pwmFiltered = 59;
float angleOffset = 0.0;

void calibrateESC() {
  Serial.println("Calibrating ESC...");
  esc.write(60);
  delay(2000);
  esc.write(50);
  delay(2000);
  Serial.println("ESC Calibrated.");
}

void checkSwitch() {
  motorEnabled = (digitalRead(switchPin) == HIGH);

  if (digitalRead(stopButtonPin) == LOW) {
    systemStopped = true;
  }
  if (digitalRead(shutdownButtonPin) == LOW) {
    systemShutdown = true;
  }
}

void calibrateAngleOffset() {
  Serial.println("Hold lever in DOWN position. Calibrating zero offset...");
  delay(3000);

  mpu.getAcceleration(&ax, &ay, &az);
  float rawAngle = atan2(ay, az) * 180 / PI;
  angleOffset = rawAngle - (-30.0);

  Serial.print("Angle Offset set to: ");
  Serial.println(angleOffset);
```

```
}

void runAutoTune() {
  Serial.println("Auto-tuning PID...");

  esc.write(59);
  delay(1000);

  esc.write(59 + 1);
  delay(300);
  esc.write(59);

  delay(2000);

  Kp = 0.8;
  Ki = 0.01;
  Kd = 0.2;

  autoTuned = true;

  Serial.println("Auto-tuning complete.");
  Serial.print("Kp: "); Serial.println(Kp);
  Serial.print("Ki: "); Serial.println(Ki);
  Serial.print("Kd: "); Serial.println(Kd);
}

void setup() {
  Serial.begin(9600);
  Wire.begin();
  mpu.initialize();

  if (!mpu.testConnection()) {
    Serial.println("MPU6050 connection failed!");
    while (1);
  }

  pinMode(switchPin, INPUT_PULLUP);
  pinMode(tuneSwitchPin, INPUT_PULLUP);
  pinMode(stopButtonPin, INPUT_PULLUP);
  pinMode(shutdownButtonPin, INPUT_PULLUP);

  esc.attach(escPin, 1000, 2000);
  esc.write(59);
```

```
  calibrateESC();
  calibrateAngleOffset();
  runAutoTune();
  lastTime = millis();
}

void loop() {
  checkSwitch();

  if (systemShutdown) {
    esc.write(50);
    Serial.println("System Shutdown - Motor OFF");
    delay(500);
    while (true);
  }

  if (systemStopped) {
    esc.write(50);
    Serial.println("System Stopped - Motor OFF");
    delay(500);
    return;
  }

  if (!motorEnabled) {
    esc.write(59);
    delay(200);
    return;
  }

  if (!mpu.testConnection()) {
    esc.write(59);
    return;
  }

  mpu.getAcceleration(&ax, &ay, &az);
  float rawAngle = atan2(
    ay, az) * 180 / PI;
  float angle = rawAngle - angleOffset;
  angleFiltered = alpha * angleFiltered + (1 - alpha) * angle;
  angleFiltered = constrain(angleFiltered, -ANGLE_LIMIT, ANGLE_LIMIT);

  float error = setpoint - angleFiltered;
  if (abs(error) < 0.5) error = 0;
```

```
  unsigned long now = millis();
  float dt = (now - lastTime) / 1000.0;
  lastTime = now;

  integral += error * dt;
  integral = constrain(integral, -100, 100);

  float derivative = (error - lastError) / dt;
  lastError = error;

  float output = Kp * error + Ki * integral + Kd * derivative;
  output *= 0.35;
  output = constrain(output, -3, 3);

  int pwm = 59 + output;
  pwm = constrain(pwm, ESC_MIN, ESC_MAX);

  pwmFiltered = 0.9 * pwmFiltered + 0.1 * pwm;
  esc.write((int)pwmFiltered);

  Serial.print("Angle:"); Serial.print(angleFiltered);
  Serial.print(",PWM:"); Serial.print(pwmFiltered);
  Serial.print(",Kp:"); Serial.print(Kp);
  Serial.print(",Ki:"); Serial.print(Ki);
  Serial.print(",Kd:"); Serial.println(Kd);

  delay(20);
}
```

D Data Sheets:

Arduino Datasheet and Specifications:
https://docs.arduino.cc/hardware/mega-2560/#tech-specs

Brushless Motor Specifications:
https://media.digikey.com/pdf/Data%20Sheets/Seeed%20Technology/108990010_Web.pdf

Accelerometer MPU-6050 Datasheet:
https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf