Lebanese University
Faculty of Engineering III
Electrical and Electronic Department

# DRONE STABILIZER

COMMUNICATION AND MINI-PROJECT

by

**Mahdi Allaw 6488**

**Saadeddine Dakdouki 6308**

**Spring 2023-2024**

---

Supervisor:

**Dr. Abed El Latif Samhat**

**Dr. Reda El Chall**

Defended on 20 June 2024 in front of the following jury:

**Dr. Abed El Latif Samhat**

**Dr. Reda El Chall**

**Dr. Jihad El Sahili**

# DEDICATION

This project is submitted as part of the requirements for the Mini Project course in the 3rd year of Engineering (Electronics and Communication) at the Faculty of Engineering III, Lebanese University. The work was conducted between February and June 2024.

# ACKNOWLEDGMENT

First, we extend our gratitude to our supervisors, Dr. Abdullatif Samhat, Professor of Electrical Engineering and Dr. Reda El-Chall. Their patience, encouragement, and for motivating us throughout this project. We are grateful for their inspiration, ideas, and constructive feedback.

We also wish to thank Dr. Jihad El Sahili, Head of the Mechanical Department, for providing mechanical instruments and his trust in this project. Additionally, we thank Mahmoud Ghosn and Mazen El-Halabi, President and Vice President of the ASME Club, respectively, for their continuous support in this project.

Moreover, we thank The RL4Eng that aims to improve the quality of higher education in our university and make it more relevant into the today's digital transformation world through establishing Remote and Virtual Laboratories for training Engineering students.

# SUMMARY

This project is designed for educational purposes, focusing on implementing a PID Controller (Proportional, Integral, Derivative) to regulate the roll angle along a single axis of a drone. The objective is to gain a comprehensive understanding of drone stability in flight and delve deeper into the principles of PID control, particularly in relation to how gain adjustments impact system behavior.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ABBREVIATIONS

| | |
|---|---|
| IC | Integrated Circuit |
| PID | Proportional Integral Derivative |
| ESC | Electronic Speed Controller |
| MPU | Motion Processing Unit |
| KV | Velocity Constant |
| RPM | Rotation Per Minute |
| PVC | Polyvinyl Chloride |
| BLDC | Brushless DC motor |
| RC | Radio Controlled |
| LiPo | Lithium Polymer |
| LCD | Liquid Crystal Display |
| SCL | Serial Clock |
| SDA | Serial Data |
| LSB | Least Significant Bit per g |
| PWM | Pulse with Modulation |
| Accx | Acceleration in the direction of x axis |

# CHAPTER 1 INTRODUCTION

## 1.1 Control System

Control is everywhere. Automatic control is essential in any field of engineering and science. Automatic control is an important and integral part of space-vehicle systems, robotic systems, modern manufacturing systems, and any industrial operations involving control of temperature, pressure, drone stability, humidity, flow, etc. It is desirable that most engineers and scientists are familiar with theory and practice of automatic control.

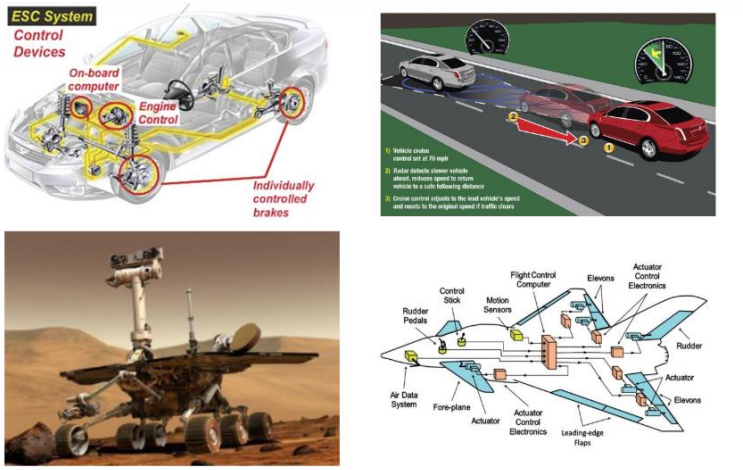


***Figure 1.1 Control system applications.***

A car's driving system can be visualized as a loop. Sensors like cameras and LiDAR feed information on the surroundings and car's condition to the controller, the car's "brain" that interprets it and sends instructions to actuators like the steering wheel and brakes to control the car, all while considering the desired path or speed.

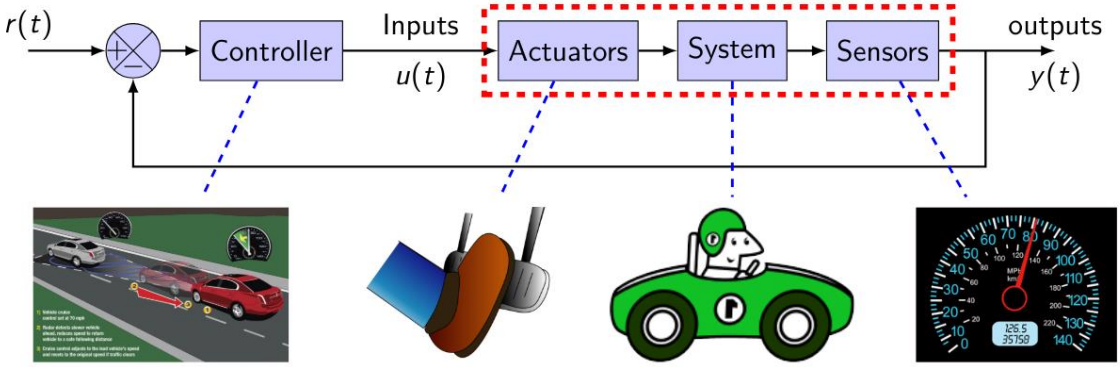


***Figure 1.2 Control system loop.***

It is interesting to note that more than half of the industrial controllers in use today are PID controllers or modified PID controllers. Because most PID controllers are adjusted on-site, many different types of tuning rules have been proposed in literatures. Using these tuning rules, delicate and fine tuning of PID controllers can be made on-site. Also, automatic tuning methods have been developed and some of the PID controllers may possess on-line automatic tuning capabilities.

The usefulness of PID controls lies in their general applicability to most control systems. In particular, when the mathematical model of the plant is not known and therefore analytical design methods cannot be used, PID controls prove to be most useful. In the field of process control systems, it is well known that the basic and modified PID control schemes have proved their usefulness in providing satisfactory control, although in many given situations they may not provide optimal control.

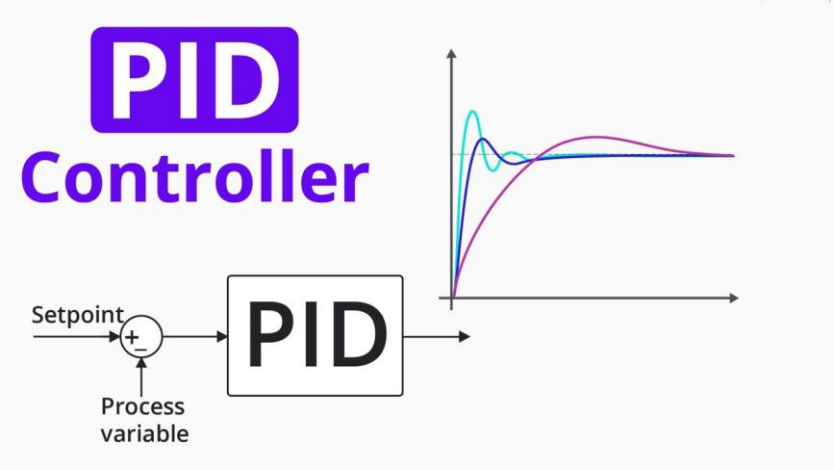


***Figure 1.3 PID Controller.***

## 1.2 Drone Stability Problem

Drone stability refers to the ability of a drone to maintain a desired flight attitude or return to a desired position after being disturbed. It is a critical aspect of drone flight dynamics, ensuring that the drone can perform tasks accurately and safely in various conditions. Stability in drones is primarily managed through control systems, which continuously adjust the drone's motors and control surfaces to maintain balance and orientation.

Static Stability: This refers to the drone's initial response to a disturbance. A statically stable drone will initially return to its original position or attitude when disturbed.

The image below illustrates the importance of stability for a drone, particularly after experiencing external forces such as water impacts from extinguishing the fire.

*Figure 1.4 Drone fire fighter.*

## 1.3 Objective for stabilization

To maintain stability, drones use control systems, often employing PID (Proportional, Integral, Derivative) controllers. A PID controller adjusts the drone's motor speeds based on the difference between the desired and actual positions or attitudes.

The roll angle of a drone, which refers to its tilt to the left or right, is crucial for maintaining stable flight. By implementing PID control for roll stability, one can grasp the fundamental concepts necessary for drone stability. Once mastered, these principles can be applied to control other angles, ensuring comprehensive stability and control.

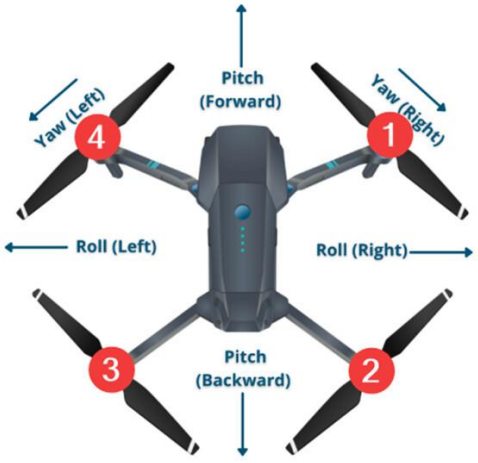


*Figure 1.5 Drone movements.*

## 1.4 Report Outline

This text is organized into 5 chapters. The outline of each chapter may be summarized as follows:

Chapter 1 Introduction to control system and drone stability.

Chapter 2 Provides an introduction to the system design, beginning with the mechanical structure and followed by the electrical structure. Additionally, the chapter covers the real-life implementation of the system architecture, including circuit design and component wiring. Detailed explanations of each design aspect and their integration will be discussed to provide a comprehensive understanding of the overall system.

Chapter 3 Provides a detailed explanation of the system functionality, focusing specifically on the motor and ESC (Electronic Speed Controller), I2C communication, the MPU 6050 sensor, the manual control box, and the PID algorithm. Each section will include practical implementation examples to illustrate how these components and concepts are integrated into the overall system. This chapter aims to give a comprehensive understanding of each element and its role in achieving the desired system performance.

Chapter 4 Outlines the implementation steps of the system software. It includes detailed descriptions of the software coding process, the testing procedures employed, and the results obtained. This chapter will provide a comprehensive guide to the software development lifecycle, from initial coding through rigorous testing, ensuring the system meets all specified requirements and functions as intended.

Chapter 5 Concludes the project and discusses future work. It summarizes the main findings and achievements, evaluates the system's performance, and suggests possible improvements and areas for further research and development.

# CHAPTER 2 SYSTEM DESIGN

## 2.1 Introduction

Understanding the comprehensive design and implementation of a technical system involves several critical aspects. Firstly, the mechanical structure focuses on the physical components and their arrangement, including frameworks, bearings, and any moving parts essential for the system's operation. Secondly, the electrical structure encompasses the various electrical components such as microcontrollers, sensors, and power supplies, along with their roles in the system's functionality. Additionally, the circuit diagram is a detailed graphical representation that illustrates the electrical connections and components within the system, facilitating easier troubleshooting and analysis.

## 2.2 System Requirements

System requirements specify the minimum hardware, network specifications necessary for a system to operate effectively. These include details such as processor speed, memory capacity and components. By defining system requirements, stakeholders can ensure that the system meets its intended purpose and can be successfully implemented within its intended environment.

### 2.2.1 Mechanical structure

The mechanical structure of the project serves as its backbone, providing stability and resilience against external forces such as vibration. By prioritizing a robust mechanical design from the outset, we ensure a solid foundation for the project's functionality.

***Figure 2.1 Mechanical structure.***

**1.a Wooden Base:** Provides a sturdy foundation for the assembly, offering stability and support for the components on it.

***Figure 2.2 Wooden base.***

**1.b Rod:** A cylindrical component used for structural support, alignment, as a pivot point within the assembly. Used to hold the outer of the bearings from any causal vibrations.

***Figure 2.3 Rod.***

**1.c Screw:** Fastener used to join two or more components together securely by creating a threaded connection.

***Figure 2.4 Screw.***

**1.d Washer:** A washer is a thin plate (usually circular) with a hole in the middle. It's used to distribute the load of a threaded fastener, such as a bolt or a screw, over a larger area. This helps prevent damage to the surface being fastened and can provide a level of insulation, dampening.

***Figure 2.5 Washer.***

1.e **Hex Nut:** A hex nut is a type of fastener with a threaded hole, usually with six sides (hence the term "hex"). It's designed to be used with a mating bolt or screw to fasten parts together. The hexagonal shape allows for easy tightening and loosening using a wrench or a socket tool.

***Figure 2.6 Hex nut.***

1.f **Open Channel:** This type of duct is used to route and protect electrical cables. It is most commonly made of PVC (polyvinyl chloride) which is a lightweight, durable, and inexpensive material. But it was used to give more flexible movements to the position of the Rods.

***Figure 2.7 Open channel.***

1.g **Bearings:** Mechanical components used to reduce friction between moving parts, facilitating smooth rotation. Has an outer connected to the vertical Rod and inner connected to the horizontal Rod that is connected to the thin plate.

***Figure 2.8 Bearing.***

**1.h 2mm Stainless Steel Plate:** Thin yet durable material commonly used for fabrication, providing strength for holding any components. Used for holding the electric components.

***Figure 2.9 Plate.***

**1.i Drone Arm:** Structural component of a drone responsible for supporting the motors and propellers and providing stability from air resistance during flight.

***Figure 2.10 Drone arm.***

### 2.2.2 Electrical structure

While the mechanical structure is essential, its value is greatly enhanced by the implementation of the electrical structure, which adds control and functionality to the system.

**2.a Arduino Microcontroller:** Acts as the brain of the system, responsible for receiving input data from the MPU-6050 sensor, processing it, and controlling the brushless motor accordingly. It executes the calculation of angles algorithm and sends appropriate signals to the brushless motor.

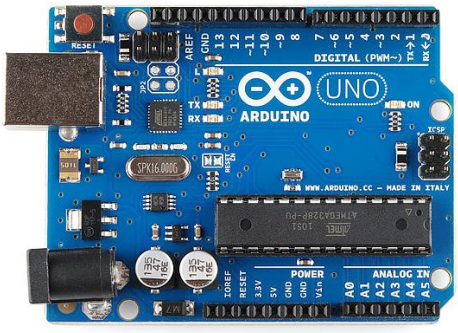


***Figure 2.11 Arduino UNO.***

**2.b MPU-6050 Sensor Module:** Combines an accelerometer and a gyroscope to detect changes in orientation and motion along three axes. It provides real-time data on the device's tilt and rotation, essential for calculating angle and adjusting the brushless motor's position.

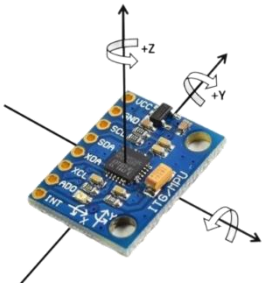


*Figure 2.12 MPU 6050*

**2.c Brushless Motor:** An electric motor that operates without the need for brushes to transfer electrical current to the rotor. Instead, it uses electronic commutation to control the motor's rotation. This design offers advantages such as higher efficiency, lower maintenance, and longer lifespan compared to a brushed motor.

*Figure 2.13 BLDC motor.*

**2.d Propellers:** Propellers are mechanical devices with blades that rotate to generate thrust, moving an object through air. Commonly used in aircraft, drones, boats, and submarines, propellers convert rotational energy into linear motion, enabling propulsion. They are designed with varying blade counts, shapes, and materials to optimize performance for the project.

*Figure 2.14 Propeller.*

**2.e Electronic Speed Controller (ESC):** A device used to control the speed and direction of a brushless motor. It interprets signals from a control source, such as an Arduino or microcontroller, and adjusts the motor's voltage and current accordingly. ESCs are commonly used in applications like drones, and electric bicycles for precise motor control.


***Figure 2.15 ESC.***

**2.f Power Supply for the ESC:** A lithium battery power supply for the Electronic Speed Controller (ESC) provides efficient and high-energy storage, crucial for powering motors in the project. Lithium batteries offer high energy density, lightweight design, and long cycle life, making them ideal for reliable and sustained ESC performance in drones.


***Figure 2.16 Lithium Battery.***

**2.g Power Supply (5V output power bank):** Supplies the necessary electrical power to the Arduino microcontroller, MPU-6050 sensor, and servo motor, ensuring proper functionality of the entire system. Normally the Arduino drive a low current.

***Figure 2.17 Power Bank.***

**2.h Switches:** Switches are electrical components used to interrupt or redirect the flow of electricity in a circuit. Switches are essential in controlling devices, appliances, and machinery by allowing users to turn them on or off or change their operating modes.

***Figure 2.18 Switch.***

**2.i Potentiometer:** A potentiometer is a three-terminal resistor with an adjustable center terminal that acts as a voltage divider. It is commonly used to control electrical devices such as volume controls on audio equipment, light dimmers, and as input for electronic circuits, allowing for precise adjustments of resistance and, consequently, voltage levels.

***Figure 2.19 Potentiometer.***

**2.j LCD:** Liquid Crystal Display, is a flat-panel display technology commonly used in televisions, computer monitors, and digital instrument panels. It operates by using liquid crystals that align when an electric current is applied, modulating light to produce images. LCDs are favored for their low power consumption, thin profile, and ability to display sharp, clear images.

***Figure 2.20 LCD.***

2.k **Breadboard:** it is a compact circuit board used for prototyping and embedded applications. It efficiently handles multiple wires on the same pin through the use of multiplexing or additional circuitry. This allows a single pin to manage multiple inputs/outputs, conserving valuable pin resources and enabling complex functionality in a compact design.

***Figure 2.21 Mini breadboard.***

2.l **Jumper Wires:** Used to establish electrical connections between the Arduino, MPU-6050 sensor, brushless motor, and breadboard, facilitating communication and power distribution within the project setup. There is 3 types female to male, female to female and male to male.

***Figure 2.22 female to male Wires.***

## 2.3 System Architecture

When wiring components to an Arduino, it's essential to establish clear connections to ensure proper functionality and avoid potential damage. Each component typically requires power, ground, and data connections. Power is usually supplied from the Arduino's 5V or 3.3V pins, while ground connections provide a reference point for electrical circuits. Data connections enable communication between the Arduino and external devices, such as sensors, actuators, or displays.

For example, connecting a sensor like the MPU6050 involves establishing connections for power (VCC), ground (GND), and data (SCL and SDA) to the corresponding pins on the Arduino.

- Connect SCL pin of MPU6050 to Arduino's I2C clock pin (usually A5).
- Connect SDA pin of MPU6050 to Arduino's I2C data pin (usually A4).
- Connect VCC pin of MPU6050 to Arduino's 3.3V output pin.
- Connect GND pin of MPU6050 to Arduino's ground (GND) pin.

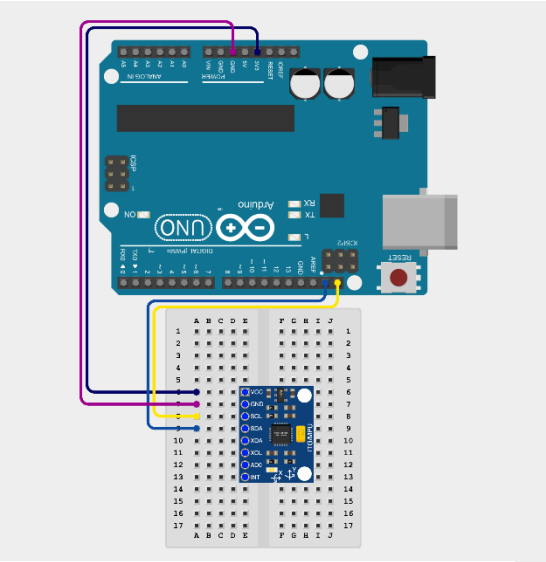

***Figure 2.23 MPU connection.***

Similarly, connecting motors with ESCs requires power connections from a suitable power source, signal connections to Arduino pins for motor control, and ground connections for completing the circuit.

- Connect the signal wire of the left ESC to pin 9 on the Arduino.
- Connect the power wires of the left ESC to the positive (+) and negative (-) terminals of the (11.1-14.8) V, 7AH battery.
- Connect the signal wire of the right ESC to pin 10 on the Arduino.
- Connect the power wires of the right ESC to the positive (+) and negative (-) terminals of the battery.
- Connect both ESC VCC and GND on the Arduino.
- Connect the three motor wires from the left motor to the corresponding three motor wires on the left ESC.

- Connect the three motor wires from the right motor to the corresponding three motor wires on the right ESC opposite to the ones on the left.

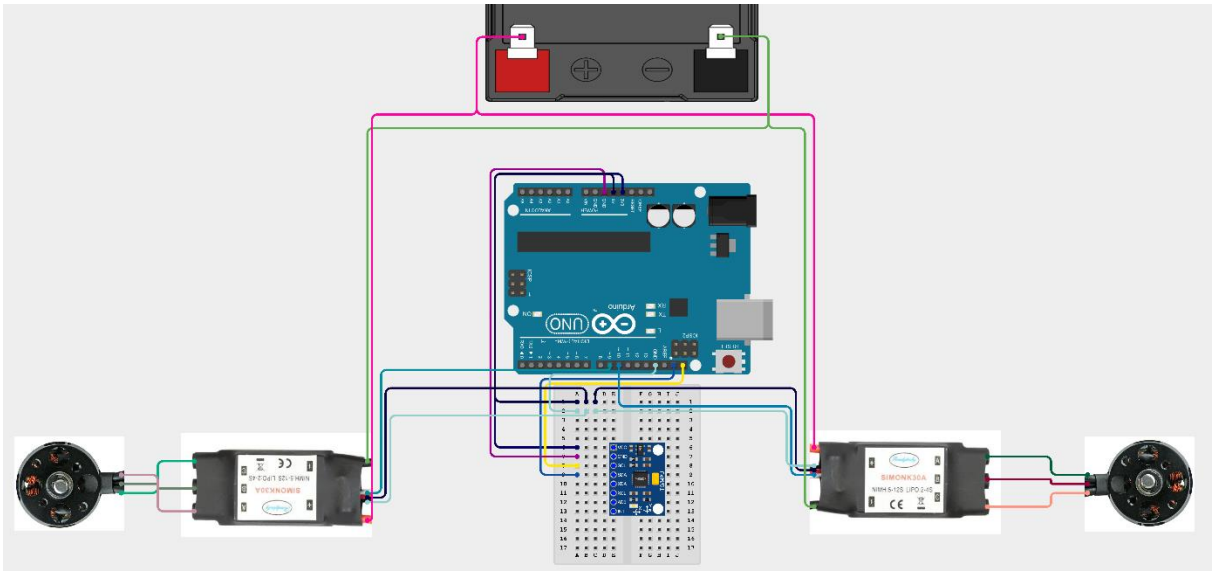

***Figure 2.24 BLDC and ESC connection.***

Connecting a display screen like the LCD involves establishing connections for power (VCC), ground (GND), and data (SCL and SDA) to the corresponding pins on the Arduino if the I2C communication protocol is used which it is.

- Connect VCC pin of the LCD to Arduino's 5V output pin.
- Connect GND pin of the LCD to any of Arduino's ground (GND) pins.
- Connect SDA pin of the LCD to Arduino's I2C data pin (usually A4).
- Connect SCL pin of the LCD to Arduino's I2C clock pin (usually A5).

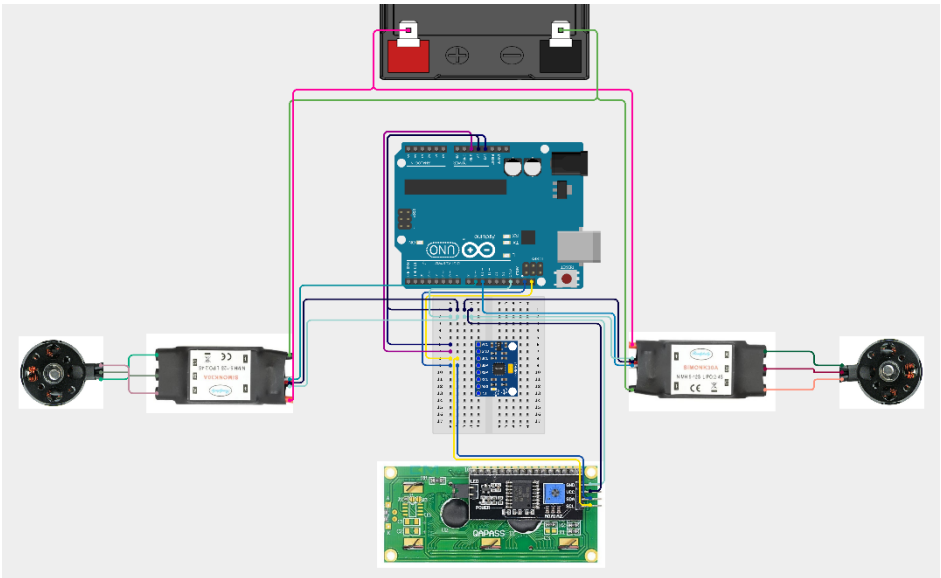

***Figure 2.25 LCD connection.***

Now to fully make the manual box we will make a connection with the 4 potentiometers and 4 switches with the Arduino.

- Connect the middle pin of each potentiometer to analog pins A0, A1, A2, and A3 on the Arduino.
- Connect the right pin of each potentiometer to the 5V output pin on the Arduino.
- Connect the left pin of each potentiometer to any of the ground (GND) pins on the Arduino.
- Connect the middle pin of each switch to digital pins 2, 3, 4, and 5 on the Arduino.
- Connect the left pin of each switch to the 5V output pin on the Arduino.
- Connect the right pin of each switch to any of the ground (GND) pins on the Arduino.

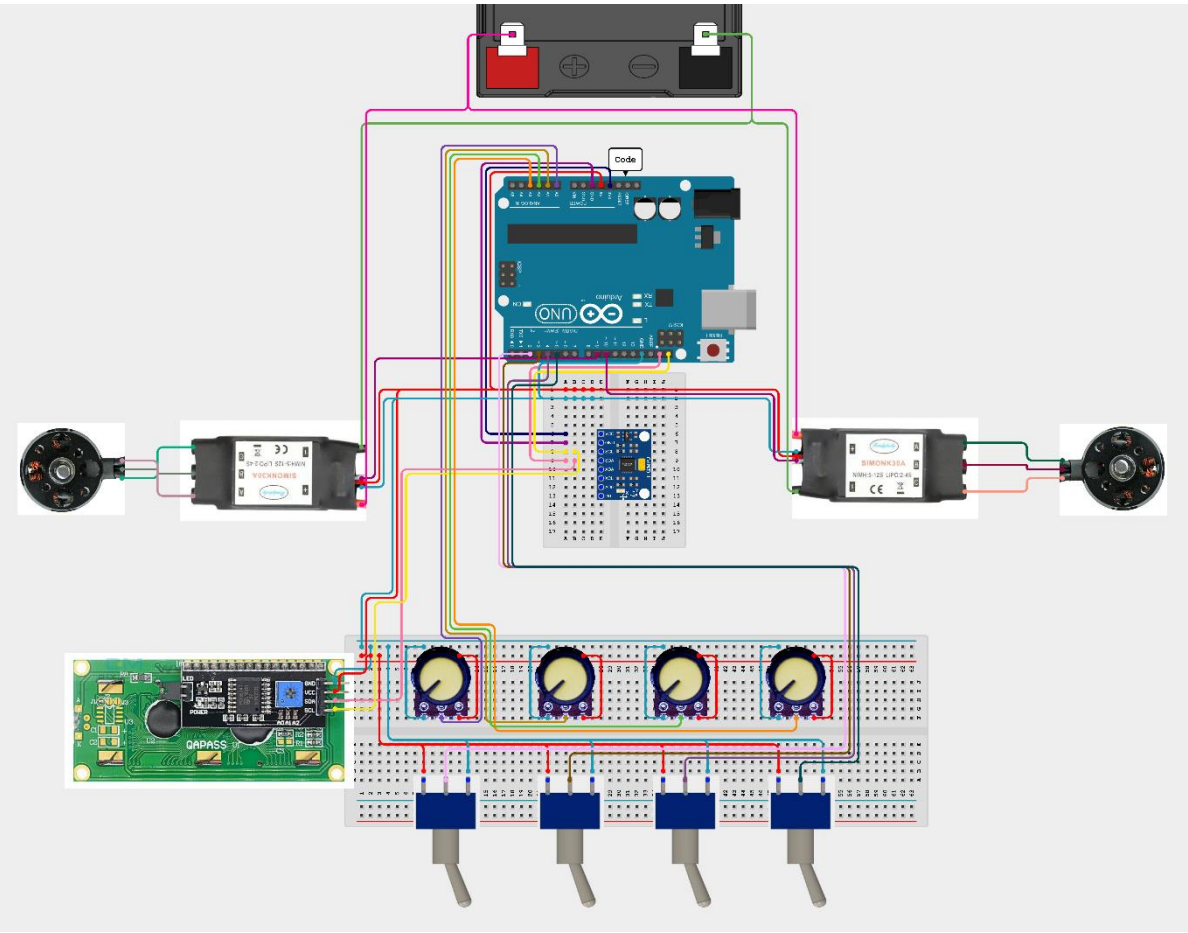


***Figure 2.26 Project circuit diagram.***

## 2.4 Conclusion

By following specific wiring instructions tailored to each component, users can seamlessly integrate various hardware elements into their Arduino projects. It's crucial to double-check connections and pin configurations to ensure safe operations.

# CHAPTER 3 SYSTEM FUNCTIONALITY

## 3.1 Introduction

In this chapter, we will delve into the intricate workings of a drone stabilizer control system, focusing on several key components and concepts that ensure its stable and efficient operation. Our discussion will cover system functionality, motors and Electronic Speed Controllers (ESCs), I2C communication, the MPU6050 sensor, the manual control box, and the implementation of PID control algorithms.

A drone stabilizer system functionality encompasses all the integrated components and their interactions to achieve stable flight. This involves the seamless coordination of hardware and software elements, which work together to control the drone's movement and maintain its stability in the air. Understanding these functionalities is crucial for developing a robust and reliable drone system.

## 3.2 Motors and ESC

### 3.2.1 Overview

A brushless DC motor or BLDC is an electric motor powered by direct current and generates its motion without any brushes like in conventional DC Motors.

Brushless motors are more popular nowadays than conventional brushed DC motors because they have better efficiency, can deliver precise torque and rotation speed control, and offer high durability and low electrical noise, thanks to the lack of brushes.

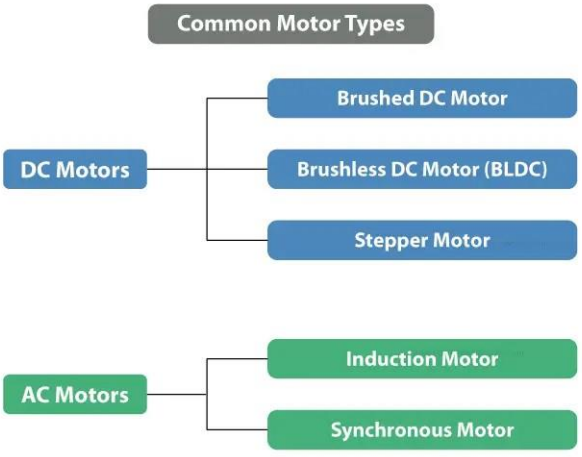


*Figure 3.1 Motor types.*

### 3.2.2 BLDC Motor

BLDC motors are used in applications where efficiency and longevity is required, like washing machines, air conditioners and other consumer electronics. Also they are used in spin hard disc drives, RC models like RC Airplane cars and so on.

A brushless motor (BLDC) consist of two main parts, a stator and a rotor. For this illustration the rotor is a permanent magnet with two poles, while the stator consists of coils arranged as shown in the picture below.

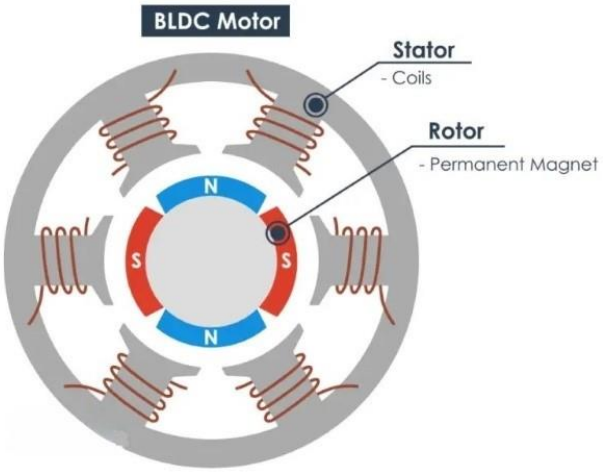


*Figure 3.2 Inside of a BLDC*

We all know that if we apply current through a coil it will generate a magnetic field and the magnetic field lines or the poles depends on the current direction.

So if we apply the appropriate current, the coil will generate a magnetic field that will attract the rotors permanent magnet. Now if we activate each coil one after another the rotor will keep rotating because of the force interaction between permanent and the electromagnet.

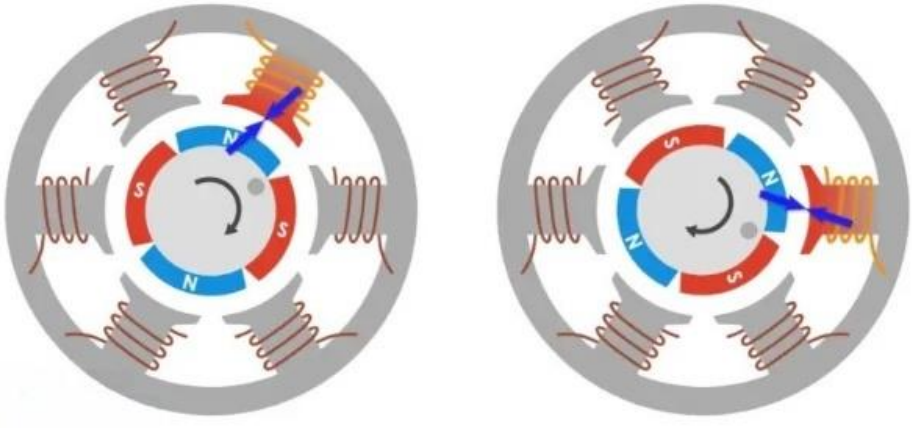

*Figure 3.3 A glimpse of turning the rotor.*

In order to increase the efficiency of the motor we can wind two opposite coils as a single coil in way that will generate opposite poles to the rotors poles, thus we will get double attraction force.

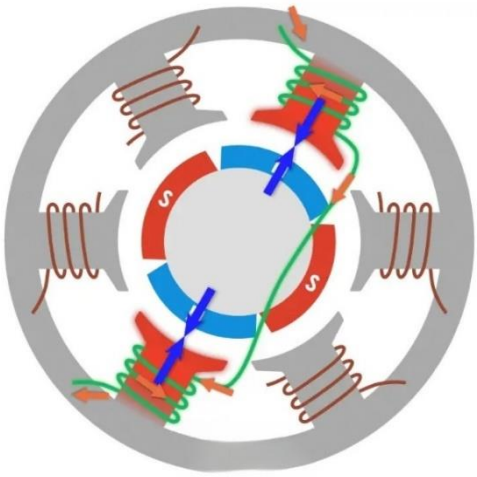


*Figure 3.4 Double attraction force in BLDC*

With this configuration we can generate the six poles on the stator with just three coils or phase. We can further increase the efficiency by energizing two coils at the same time. In that way one coil will attract and the other coil will repel the rotor. In order the rotor to make a full 360 degrees' cycle, it need six steps or intervals.

If we take a look at the current waveform we can notice that in each interval there is one phase with positive current, one phase with negative current and the third phase is turned off. This gives the idea that we can connect the free end points of each of the three phases together and so we can share the current between them or use a single current to energize the two phases at the same time.

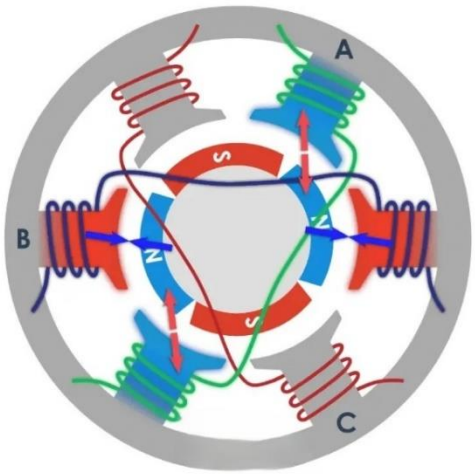


*Figure 3.5 Double attraction and repulsion force in BLDC*

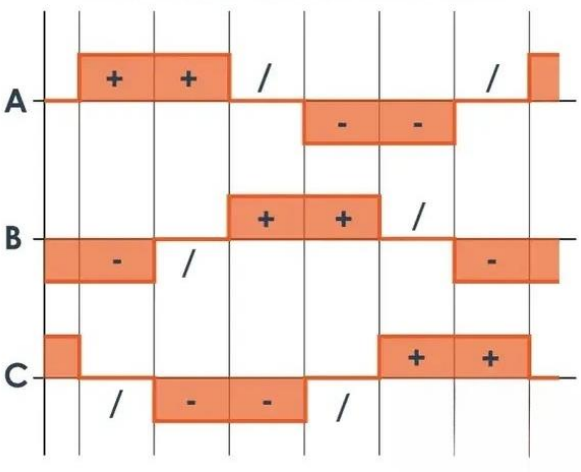


*Figure 3.6 BLDC motor current waveform*

Here's an example. If we pull up phase A High, or connect it to the positive DC voltage, with some kind of switch, for example a MOSFET, and on the other side, connect the phase B to ground, then the current will flow from VCC, through phase A, the neutral point and phase B, to ground. So, with just a single current flow we generated the four different poles which cause the rotor to move.

With this configuration we actually have a star connection of the motor phases, where the neutral point is internally connected and the other three ends of the phases come out of the motor and that's why brushless motor have three wires coming out of it.

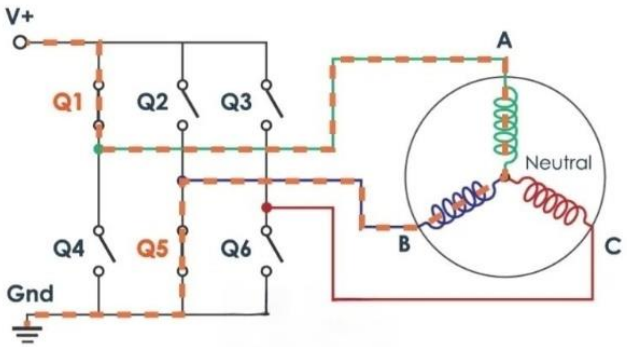


***Figure 3.7 Three phase system***

So, in order the rotor to make full cycle we just need to activate the correct two MOSFETS in each of the 6 interval and that's what ESCs are actually all about.

### 3.2.3 ESC

An ESC or an Electronic Speed Controller controls the brushless motor movement or speed by activating the appropriate MOSFETs to create the rotating magnetic field so that the motor rotates. The higher the frequency or the quicker the ESC goes through the 6 intervals, the higher the speed of the motor will be.

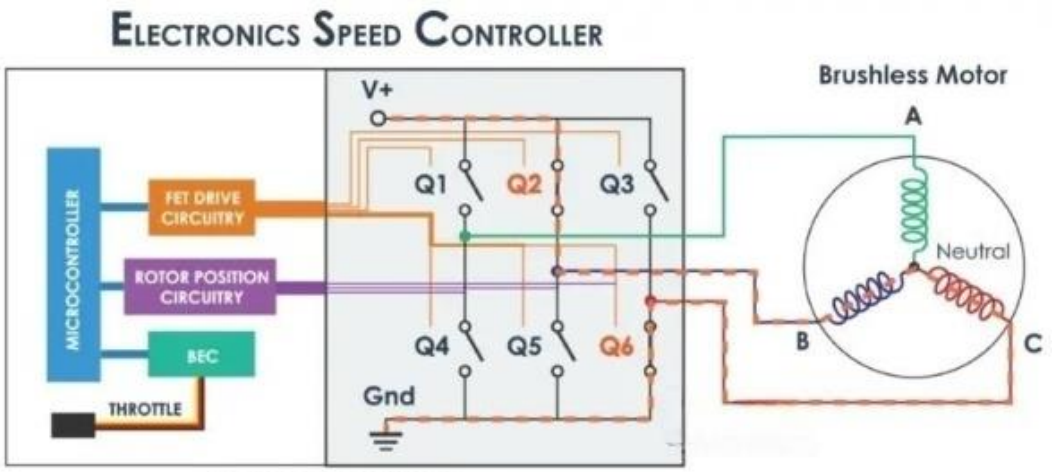


***Figure 3.8 ESC.***

So that's the basic working principle of brushless DC motors and ESCs and it's the same even if we increase the number of poles of the both the rotor and the stator. We will still have a three-phase motor, only the number of intervals will increase in order to complete a full cycle.

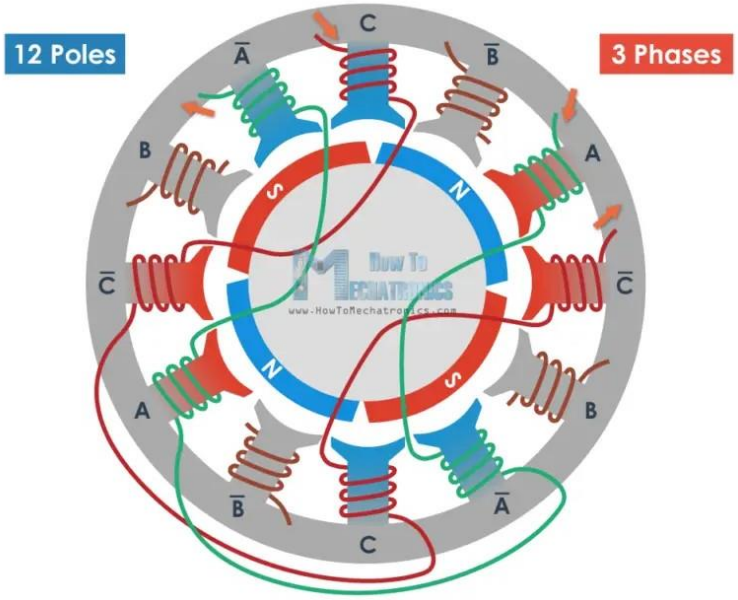


***Figure 3.9 BLDC working principle.***

Here we can also mention that BLDC motors can be in runners or out runners. An in runner brushless motor has the permanent magnets inside the electromagnets, and vice versa, an out runner motor has the permanent magnets outside the electromagnets. Again, they use the same working principle and each of them has its own strengths or weaknesses.

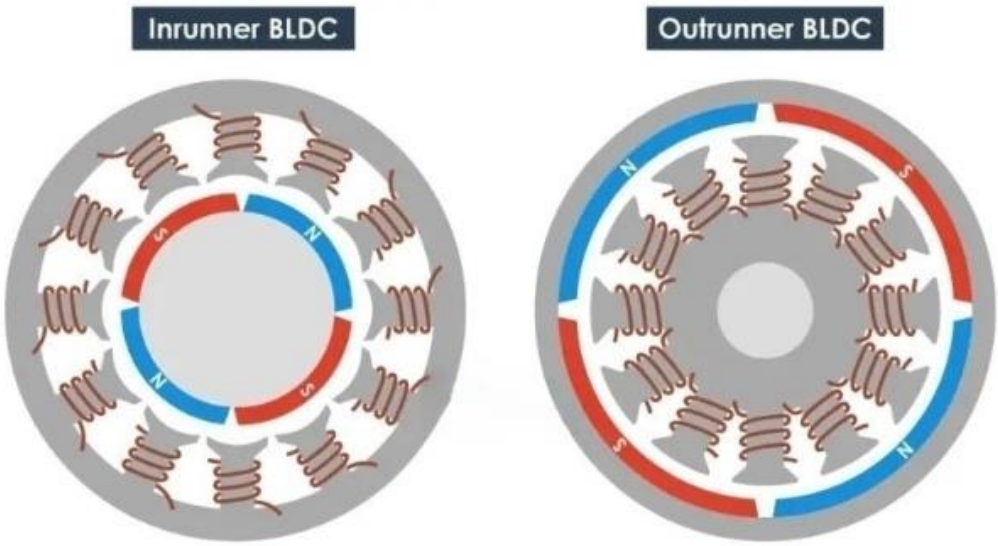


***Figure 3.10 Types of BLDC.***

### 3.2.4 Practical implementation

The "KV" of a brushless motor refers to its velocity constant, measured in RPM (rotations per minute) per volt. It indicates how many revolutions per minute the motor will turn when one volt is applied to it, with a higher KV value meaning the motor spins faster per volt. This value is crucial for matching the motor's performance to the intended application, such as determining the suitable propeller size for a drone or RC aircraft. On the other hand, "ESC 3S-4S" stands for Electronic Speed Controller compatible with LiPo battery configurations of 3S (11.1 volts) and 4S (14.8 volts).

$$RPM = KV \times Voltage$$

The 3S-4S specification indicates the ESC's compatibility with different LiPo battery configurations, ensuring proper voltage regulation and optimal motor performance within the specified voltage range. Matching the KV of the motor with the appropriate ESC voltage compatibility is crucial for efficient and safe operation in various applications, such as RC vehicles, drones, or electric-powered models. In this project we will use the 960 KV brushless motor and ESC of Maximum 40A of current that is compatible with 3S-4S LiPo battery.

***Figure 3.11 960 KV BLDC and 4S ESC.***

## 3.3 I²C Communication

### 3.3.1 Overview

The I2C communication bus is very popular and broadly used by many electronic devices because it can be easily implemented in many electronic designs which require communication between a master and multiple slave devices or even multiple master devices. The easy implementations come with the fact that only two wires are required for communication between up to almost 128 (112) devices when using 7 bits addressing and up to almost 1024 (1008) devices when using 10 bits addressing.

The two wires, or lines are called Serial Clock (or SCL) and Serial Data (or SDA). The SCL line is the clock signal which synchronize the data transfer between the devices on the I2C bus and it's generated by the master device. The other line is the SDA line which carries the data.

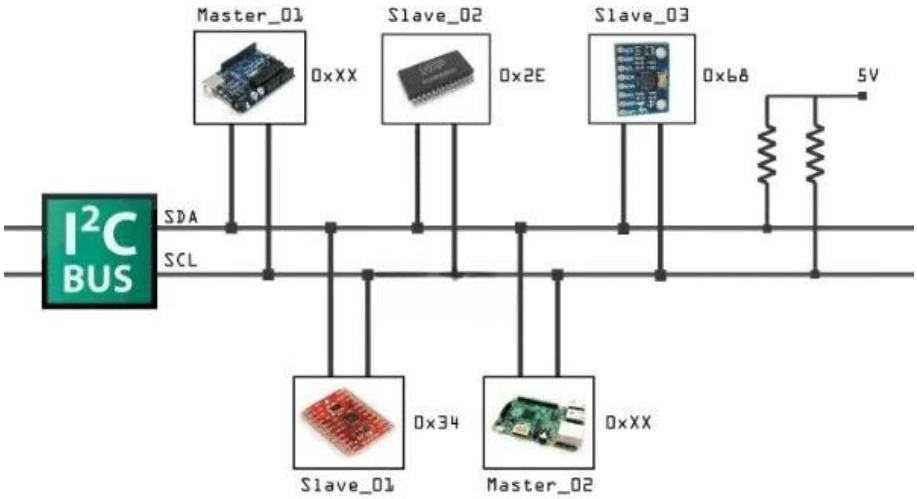


*Figure 3.12 I2C Bus.*

### 3.3.2 I2C Protocol

The data signal is transferred in sequences of 8 bits. So after a special start condition occurs comes the first 8 bits sequence which indicates the address of the slave to which the data is being sent. After each 8 bits sequence follows a bit called Acknowledge. After the first Acknowledge bit in most cases comes another addressing sequence but this time for the internal registers of the slave device. Right after the addressing sequences follows the data sequences as many until the data is completely sent and it ends with a special stop condition.

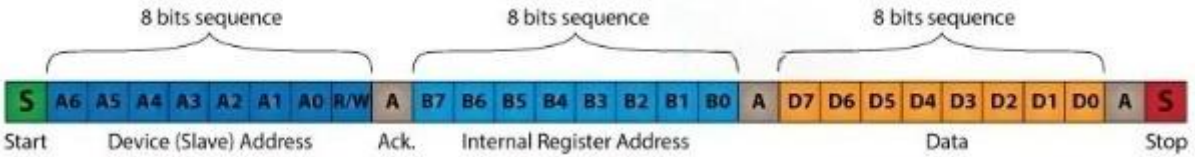


*Figure 3.13 I2C Protocol.*

### 3.3.3 Practical implementation

In this project, both the MPU 6050 and the LCD utilize I2C communication with the Arduino serving as the master and both peripherals acting as slaves. This setup allows for efficient communication using only two pins: SCL (clock) and SDA (data). By employing the I2C protocol, the Arduino can easily exchange data with the MPU 6050 to retrieve sensor readings and with the LCD to display relevant information without requiring a large number of pins, thus optimizing resource utilization and simplifying the wiring configuration.

## 3.4 MPU 6050

### 3.4.1 Overview

The MPU 6050 is a versatile and widely-used sensor module that integrates a 3-axis gyroscope, a 3-axis accelerometer, and a digital thermometer into a single chip. It is designed to provide accurate and comprehensive motion tracking and orientation data, making it ideal for applications such as drones, robotics, and wearable technology.

### 3.4.2 Accelerometer

When thinking about accelerometers it is often useful to image a box in shape of a cube with a ball inside it.

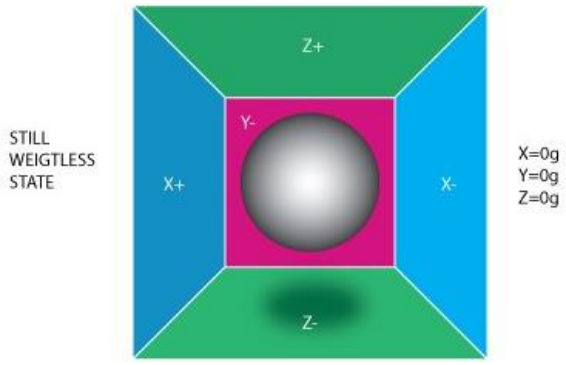


***Figure 3.14 Ball in a cube.***

If we take this box in a place with no gravitation fields or for that matter with no other fields that might affect the ball's position the ball will simply float in the middle of the box. You can imagine the box is in outer space far away from any cosmic bodies, or in weightless state. From the picture below you can see that we assign to each axis a pair of walls (we removed the wall Y+ so we can look inside the box). Imagine that each wall is pressure sensitive. If we move suddenly the box to the left (we accelerate it with acceleration 1g = 9.8m/s^2), the ball

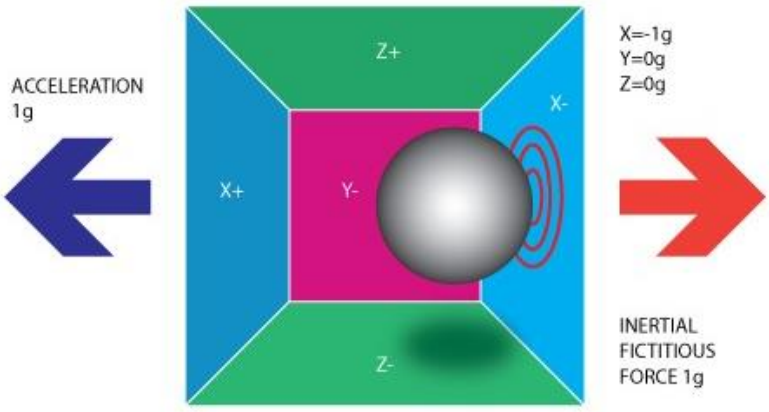


***Figure 3.15 Ball after acceleration.***

will hit the wall X-. We then measure the pressure force that the ball applies to the wall and output a value of -1g on the X axis.

Please note that the accelerometer will actually detect a force that is directed in the opposite direction from the acceleration vector. This force is often called Inertial Force or Fictitious Force. One thing you should learn from this is that an accelerometer measures acceleration indirectly through a force that is applied to one of its walls (according to the model, it might be a spring or something else in real life accelerometers). This force can be caused by the acceleration, but it is not always caused by acceleration.

If we take the model and put it on Earth the ball will fall on the Z- wall and will apply a force of 1g on the bottom wall, as shown in the picture below:

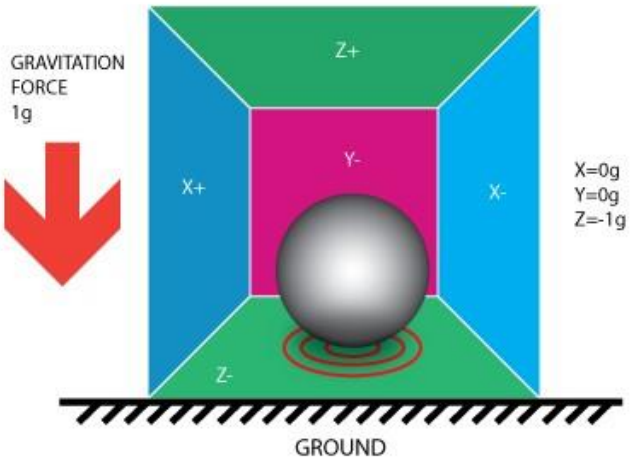


***Figure 3.16 Ball in normal conditions.***

In this case the box isn't moving but we still get a reading of -1g on the Z axis. The pressure that the ball has applied on the wall was caused by a gravitation force. In theory it could be a different type of force for example. This was said just to prove that in essence accelerometer measures force not acceleration. It just happens that acceleration causes an inertial force that is captured by the force detection mechanism of the accelerometer.

in real life It measures acceleration by measuring change in capacitance. Its micro structure looks something like this. It has a mass attached to a spring which is confined to move along one direction and fixed outer plates. So when an acceleration in the particular direction will be applied the mass will move and the capacitance between the plates and the mass will change. This change in capacitance will be measured, processed and it will correspond to a particular acceleration value.

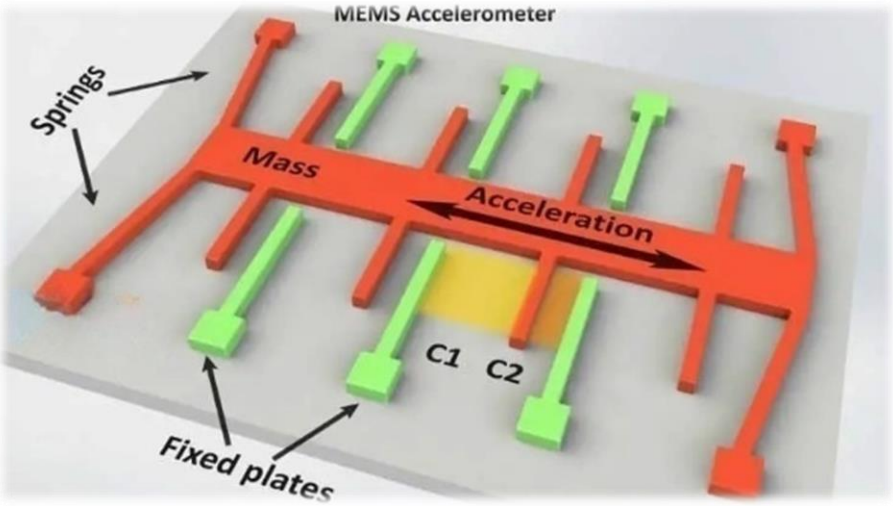


*Figure 3.17 Accelerometer in real life.*

The accelerometer measures acceleration along the X, Y, and Z axes. Under normal conditions on Earth, the accelerometer registers a gravitational acceleration of approximately 9.81 m/s², commonly denoted as 1 g. For instance, when the sensor is placed flat on a surface without movement, the Z-axis acceleration ($Acc_z$) reads 1 g, while the X and Y axes show zero acceleration. Similarly, when the sensor is positioned such that one of its axes is perpendicular to the surface, the corresponding axis will register an acceleration of 1 g due to Earth's gravity.

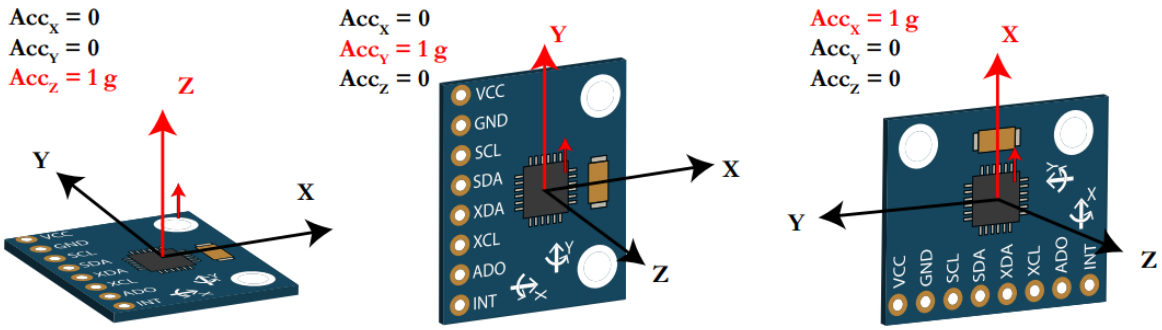


*Figure 3.18 Accelerometer in chip.*

Off course, any other direction not along one of the three main axes will result in a nonzero acceleration value for all three directions $Acc_x$, $Acc_y$ and $Acc_z$.

### 3.4.3 Gyroscope

To stabilize your quadcopter, you need measurements of its three-dimensional orientation. In this project, it is sufficient to know the rotation rates when rolling, a sensor able to record these rotation rates is called a gyroscope.

the gyroscope measures angular rate using the Coriolis Effect. When a mass is moving in a particular direction with a particular velocity and when an external angular rate will be applied a force will occur, which will cause perpendicular displacement of the mass. So similar to the accelerometer, this displacement will cause change in capacitance which will be measured, processed and it will correspond to a particular angular rate.

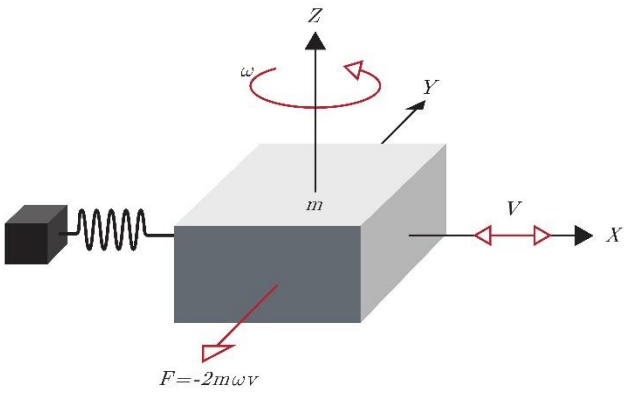


***Figure 3.19 Gyroscope working principle.***

The micro structure of the gyroscope looks something like this. A mass that is constantly moving, or oscillating, and when the external angular rate will be applied a flexible part of the mass would move and make the perpendicular displacement.

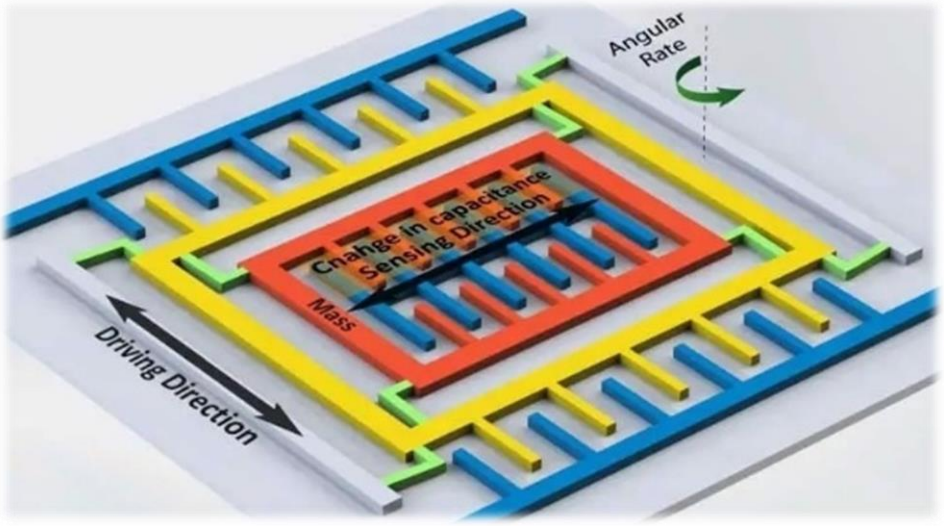


***Figure 3.20 Gyroscope in real life.***

The gyroscope that we will use is included in the MPU-6050, a low-cost off-the shelve orientation sensor. While the MPU is not a very precise sensor, its accuracy is sufficient to get great results balancing the two motors. we will use the gyroscope to measure the roll rate: this means measuring angular rates in degrees per second (°/s) not the angle measure (°).

### 3.4.4 Practical implementation

In this project, we will use the MPU 6050 sensor, which features both an accelerometer and a gyroscope, to measure the roll angle. This data will provide feedback on the tilt of the two motors. We will then apply a PID algorithm based on the calculated angle to adjust the motor positions accordingly.

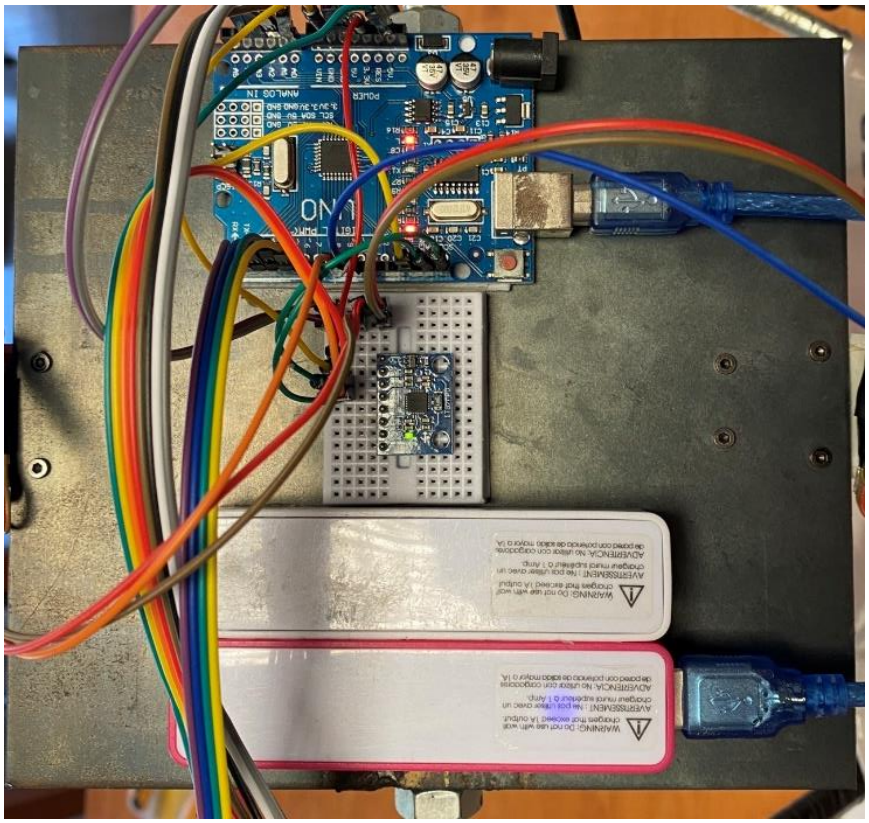

*Figure 3.21 MPU 6050 in the project.*

## 3.5 Manual control box

### 3.5.1 Potentiometer

A potentiometer is a type of variable resistor with three terminals. The first terminal is connected to a fixed voltage, the third terminal is connected to the ground, and the second terminal is the wiper, which slides across the resistive material inside the potentiometer. By turning the wiper of the potentiometer, the position of the wiper changes, thereby altering the resistance between the wiper and the two fixed terminals. This change in resistance results in a variable voltage output that corresponds to the position of the wiper. In this project, four potentiometers are used: one to read the desired angle for the motors, and the other three to set the PID constants (Proportional, Integral, Derivative). These inputs allow for manual adjustment of the target angle and the PID tuning parameters.

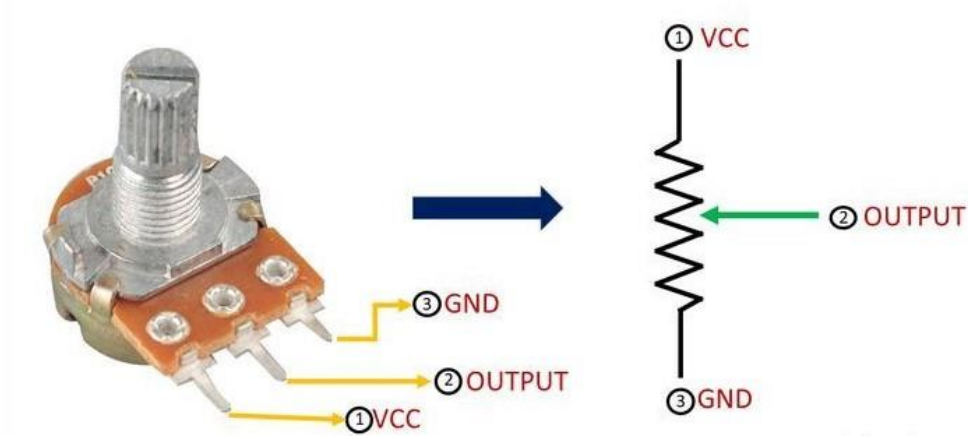


*Figure 3.22 Potentiometer working principle.*

### 3.5.2 Switch

Switches are used to control the functionality of the system. There are four switches in total. The first switch is used for manually turning the system on and off. This allows the user to enable or disable the manual control system at will. The other three switches are used to control whether the PID constants are active or not. When these switches are turned off, the corresponding PID constant (Proportional, Integral, or Derivative) is set to zero. When the switches are turned on, the PID constants are read from their respective potentiometers, allowing for dynamic adjustment of the control parameters based on the potentiometer settings.

*Figure 3.23 switch.*

### 3.5.3 LCD

An LCD (Liquid Crystal Display) works by manipulating liquid crystals to control light and create visual images. The basic structure of an LCD consists of two polarizing filters with liquid crystal solution sandwiched between them. When an electric current is applied to the liquid crystals, they align in such a way that they either block or allow light to pass through, depending on the desired display. This process enables the LCD to display text, numbers, and images.

To interface with an LCD, the microcontroller sends data and commands via a parallel or serial communication protocol. In parallel mode, several data lines are used to transfer information simultaneously, along with control lines for enabling the display and selecting whether data or command is being sent. In serial mode, fewer lines are used, and data is sent sequentially, which simplifies wiring but can be slower.

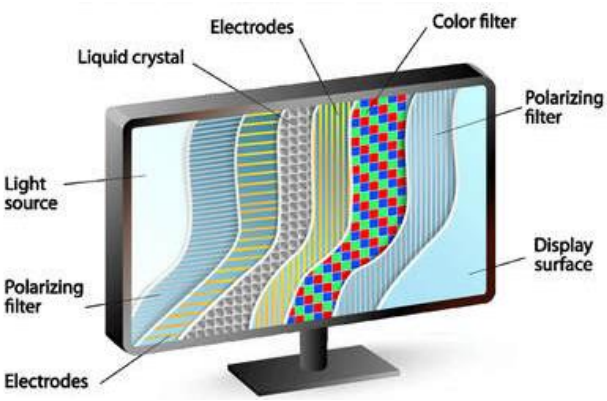


***Figure 3.24 Liquid crystal display.***

In this project, the LCD displays critical information such as the desired angle, the actual angle measured by the MPU 6050 sensor, and the values of the PID constants. The microcontroller continuously updates the LCD with real-time data, ensuring the user has up-to-date feedback on the system's performance. The desired angle and PID constants are read from the potentiometers and switches, and the actual angle is calculated based on the sensor readings. The microcontroller processes these inputs and then sends the corresponding data to the LCD for display.

These components make connection with the Arduino for manipulating the system behavior. The real-time visual feedback is essential for monitoring and adjusting the system to achieve the desired motor control.


***Figure 3.25 Manual box implementation.***

## 3.6 PID

To maintain stability, drones use control systems, often employing PID (Proportional, Integral, Derivative) controllers. A PID controller adjusts the drone's motor speeds based on the difference between the desired and actual position or angle. here is a block diagram of a PID feedback control system:

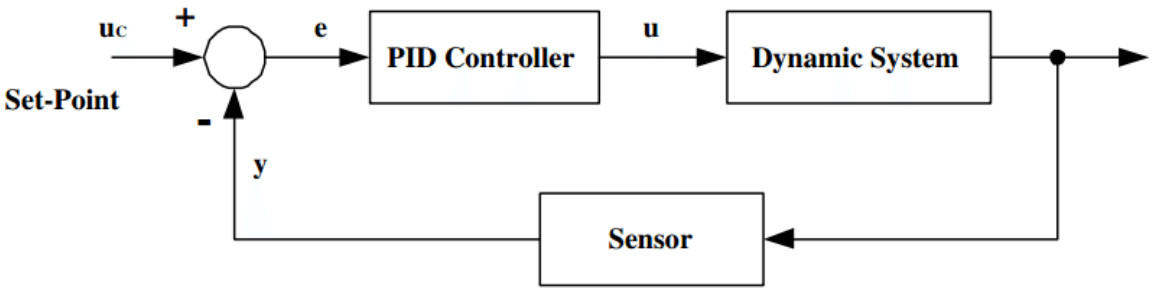


*Figure 3.26 Control system with PID loop.*

The transfer function for a PID controller is:

$$Gc(s) = K_p + \frac{K_i}{s} + K_d s = K_p\,(1 + \frac{1}{T_i s} + T_d s)$$

The popularity of PID controllers can be attributed partly to their good performance over a wide range of operating conditions and partly to their functional simplicity that allows engineers to operate them in a simple, straightforward manner.

To implement the PID controller, three parameters must be determined, P, I and D Gains There are many methods available to determine acceptable values of the PID gains. The process of determining the gains is often called PID tuning.

A common approach to tuning is to use manual PID tuning methods, whereby the PID control gains are obtained by trial-and-error, and this will be used since it has no effect on breaking the system. But you need to understand in each tune what the impact will be.

| PID Gain | Percent Overshoot | Settling Time | Steady-State Error |
|---|---|---|---|
| Increasing $K_P$ | Increases | Minimal impact | Decreases |
| Increasing $K_I$ | Increases | Increases | Zero steady-state error |
| Increasing $K_D$ | Decreases | Decreases | No impact |

*Figure 3.27 PID tuning.*

Proportional Control (P): Responds to the current error the difference between the desired and actual roll angle. A higher proportional gain results in a stronger corrective action. If we apply only the Proportional (P) algorithm, the system response will exhibit certain characteristics. Specifically, we will observe an overshoot and oscillations over time. It takes a considerable amount of time for the system to reach a steady state, and even then, a steady-state error remains present.

$$g(t) = K_p \times e(t)$$

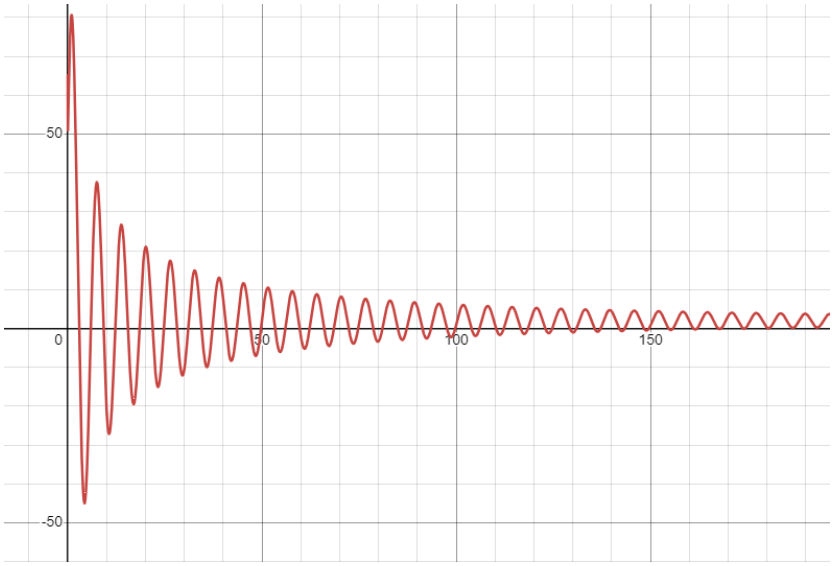


***Figure 3.28 Proportional control response.***

Derivative Control (D): Predicts future errors based on the rate of change of the error, providing a damping effect to reduce overshoot and oscillations. When we apply the Proportional-Derivative (PD) algorithm, the system's response improves compared to using only the Proportional (P) control. The Proportional term helps to reduce the immediate error, while the Derivative term predicts future errors by considering the rate of change of the error. This combination results in:

Reduced Overshoot: The Derivative term helps to dampen the response, decreasing the extent of overshoot.

Less Oscillation: By addressing the rate of change, the Derivative term helps to reduce oscillations, making the system more stable.

Faster Settling Time: The system reaches its steady state more quickly.

$$g(t) = K_p \times e(t) + K_d \times \frac{de(t)}{dt}$$

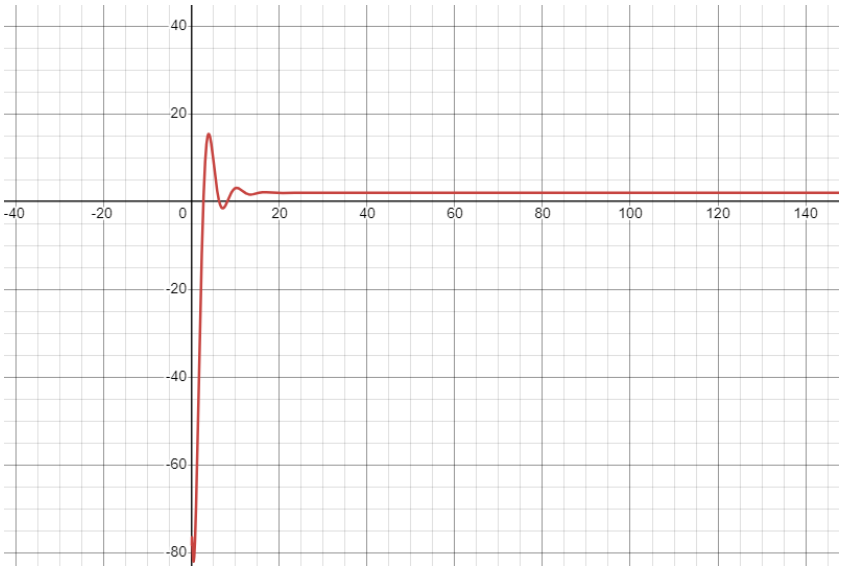


***Figure 3.29 PD control response.***

Integral Control (I): Accounts for past errors by integrating the error over time, which helps eliminate residual steady-state errors. When the error changes rapidly, Kd helps in quickly stabilizing the system by counteracting this change. When we apply the Proportional-Integral-Derivative (PID) algorithm, the system response is significantly enhanced compared to using only P or PD control. The PID control algorithm combines the strengths of the Proportional, Integral, and Derivative terms to provide a well-rounded and efficient control system. This combination results in addition:

Zero Steady-State Error: The Integral term ensures that any residual error is corrected over time, resulting in no steady-state error.

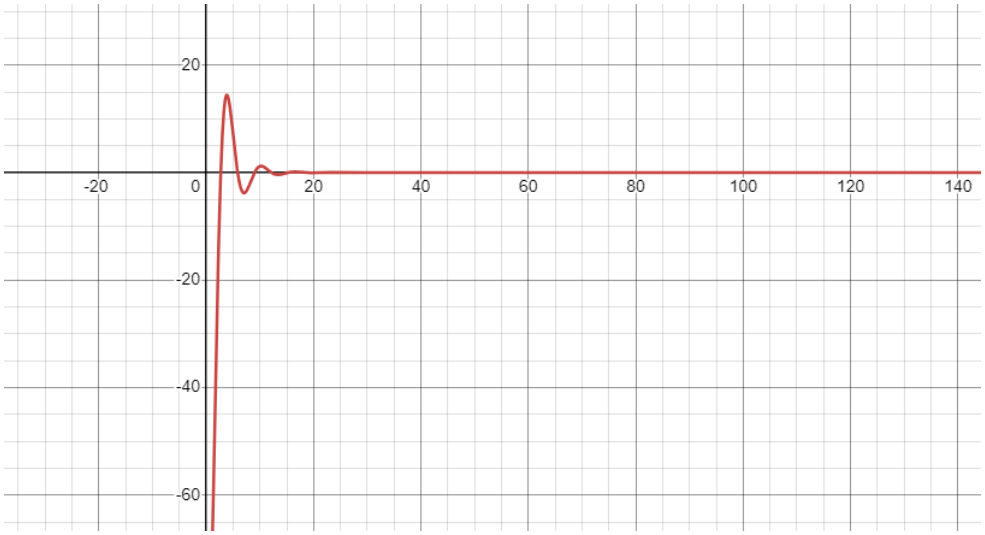


***Figure 3.30 PID control response.***

Together, these components work to optimize the control system's response to achieve desired set points while minimizing error and instability.

## 3.7 Conclusion

In this chapter, we have explored the various components and mechanisms that contribute to the stability and functionality of a drone stabilizer. Understanding the intricate details of system functionality, motors and ESCs, I2C communication, the MPU6050 sensor, the manual control box, and PID control is essential for developing and maintaining a robust drone system.

The motors and ESCs are the powerhouse of the drone, providing the necessary thrust and precise speed control required for balanced flight. I2C communication ensures efficient data transfer between the flight controller and sensors, enabling real-time adjustments. The MPU6050 sensor plays a critical role in detecting orientation and movement, supplying the flight controller with vital data to maintain stability.

The manual control box serves as the human interface, allowing operators to command the drone with precision. The implementation of PID control algorithms is crucial in minimizing errors and achieving responsive and stable flight control. Each aspect of the PID controller Proportional, Integral, and Derivative works together to correct deviations and maintain the desired flight parameters.

By integrating these components effectively, we can achieve a high level of control and stability in drone operations. This understanding lays the groundwork for further advancements in drone technology, enabling more sophisticated applications and enhanced performance in various fields.

# CHAPTER 4 IMPLEMENTATION AND RESULTS

## 4.1 Introduction

This chapter outlines the implementation process of the system. It encompasses detailed explanations of software coding, testing procedures, and the results obtained.

## 4.2 Implementation Tools

We will begin by defining the libraries and variables utilized in the system. Typically, the variable names are chosen to clearly indicate their purposes.

### 4.2.1 Necessary libraries

Wire.h: This library facilitates communication between a microcontroller and I2C (Inter-Integrated Circuit) devices. It allows for reading and sending data to and from external I2C components, such as sensors, displays, and other peripherals. The Wire library provides a straightforward way to handle the I2C protocol, enabling the microcontroller to act as either a master or a slave in the communication process. Common functions include begin(), requestFrom(), beginTransmission(), endTransmission(), and write(), which collectively manage the setup, data request, and data transmission stages of I2C communication.

Servo.h: This library provides an efficient method for controlling multiple servo motors or actuators using only one timer from the microcontroller. The Servo library abstracts the complexity of PWM (Pulse Width Modulation) signal generation, allowing users to easily set the position of servos with simple commands. It supports a wide range of servos, enabling applications in robotics, animatronics, and remote-controlled devices. Key functions include attach(), detach(), write(), writeMicroseconds(), and `read(), which allow for attaching a servo to a specific pin, setting its angle, and reading its current position.

LiquidCrystal_I2C.h: This library is designed for controlling LCD (Liquid Crystal Display) screens that use the I2C communication protocol. It provides functions similar to the LiquidCrystal library, but optimized for I2C interface, which reduces the number of pins required to connect the display to the microcontroller. LiquidCrystal_I2C simplifies the management of text displays, including initializing the display, positioning the cursor, and printing text. Important functions include begin(), clear(), home(), setCursor(), and print(), enabling easy manipulation of text and cursor positioning on the LCD.

```
#include <Wire.h>
#include <Servo.h>
#include <LiquidCrystal_I2C.h>
```

*Figure 4.1 Library used.*

These libraries are integral components of the Arduino ecosystem, facilitating the integration and control of various hardware modules and sensors in projects ranging from simple displays to complex robotics.

### 4.2.2 Necessary variables

```
int manualOverridePin = 2,
  kpOnPin = 3,
  kiOnPin = 4,
  kdOnPin = 5;

int manualOverideState = 0,
  kpOnState = 0,
  kiOnState = 0,
  kdOnState = 0;

int potPinA = A0,
  potPinP = A2,
  potPinI = A3,
  potPinD = A1,
  potValueA, potValueP, potValueI, potValueD;

const int MPU_addr = 0x68;

float AcX, AcY, AcZ, GyX, GyY, GyZ;
float AccAnglePitch, AccAngleRoll, RollAngle = 0, PitchAngle = 0;
double dt, now, PrevTime;

double error, previous = 0, pwmLeft, pwmRight, proportional,
integral = 0, derivative, PID_output = 0;

double kp=0.85,
  ki=0.0005,
  kd=0.60;
int throttle=1300;
float desired_angle = 0;
```

*Figure 4.2 Variables.*

The first four variables, manualOverridePin, kpOnPin, kiOnPin, and kdOnPin, are designated to define the pins for the four switches. These switches are used to toggle the manual box and the proportional (kp), integral (ki), and derivative (kd) controls on or off.

The next four variables indicate the states of these switches, either high or low. These variables are used to monitor whether the manual override, kp, ki, or kd controls are active or inactive.

Similarly, for the four potentiometers, we use the analog pins A0, A1, A2, and A3. The variables potPinA = A0, potPinP = A2, potPinI = A3, and potPinD = A1 define these pins. Correspondingly, potValueA, potValueP, potValueI, and potValueD store the potentiometer readings.

We define the address of the MPU-6050 sensor for I2C communication with the following constant, MPU_addr.

Next, we define the variables that will hold data from the accelerometer and gyroscope, as well as the calculated angles. Notably, we do not define a separate variable for the gyro angle since we use a complementary filter, making it unnecessary. The variables AcX, AcY, AcZ, GyX, GyY, GyZ store sensor data. The variables AccAnglePitch, AccAngleRoll, RollAngle, PitchAngle store the calculated pitch and roll angles for accelerometer and all angles.

The variable now stores the current time in milliseconds, dt holds the time elapsed since the previous iteration in seconds, and PrevTime retains the previous time value for comparison in the next loop iteration.

In brief, error represents the difference between the desired and actual angles. previous maintains the previous error value within the loop. pwmLeft and pwmRight are for controlling the left and right BLDC motors, respectively, via PWM input. The remaining variables, proportional, integral, derivative, and PID_output, are used in PID calculation to adjust motor output based on the error signal.

In brief, kp, ki, and kd store the proportional, integral, and derivative constants for the PID controller, respectively. throttle holds the base throttle value for the motors. desired_angle stores the target angle to which the system aims to stabilize.

right_prop, left_prop, lcd, These objects are then utilized in the setup and loop functions to control the servo motors and interact with the LCD display, respectively, throughout the program.

```
Servo right_prop;
Servo left_prop;
LiquidCrystal_I2C lcd(0x27, 16, 2);
```

***Figure 4.3 Object initialization.***

## 4.3 Implementation setup

In the setup() function, the initial configuration of the system is being performed. Serial.begin(9600): This line initializes serial communication with a baud rate of 9600, allowing for debugging and data transmission between the microcontroller and an external device.

pinMode(): The pinMode() function is used to set the pin modes for various input pins. The pins manualOveridePin, kpOnPin, kiOnPin, kdOnPin, potPinA, potPinP, potPinI, and potPinD are configured as input pins.

Wire.begin(): This function initializes the I2C communication protocol, enabling communication with I2C devices connected to the microcontroller.

setupMPU(): This is a custom function call that sets up the MPU-6050 sensor. It initializes the sensor, configures its settings, and prepares it for data acquisition. We will take about it later.

right_prop.attach(10) and left_prop.attach(9): These lines attach servo objects to specific pins (‘10’ and ‘9’, respectively) for controlling the right and left propellers of the system.

left_prop.writeMicroseconds(1000) and right_prop.writeMicroseconds(1000): These commands set the initial positions of the left and right propellers by writing a PWM signal of 1000 microseconds, presumably to set them to a neutral position.

delay(7000): This command introduces a delay of 7000 milliseconds (7 seconds). It's likely used to allow time for the system to stabilize or for initialization processes to complete.

lcd.init(): This function initializes the LCD display. During initialization, the LCD controller is configured with parameters such as display size, cursor options, and any custom characters if needed. It sets up the necessary communication protocol (I2C in this case) and prepares the display for further commands.

lcd.backlight(): This function turns on the backlight of the LCD display. Many LCD displays come with an adjustable backlight for better visibility in different lighting conditions. By calling this function, you activate the backlight, making the text or graphics displayed on the LCD visible.

```
void setup()
{
  Serial.begin(9600);

  pinMode(manualOverridePin, INPUT);
  pinMode(kpOnPin, INPUT);
  pinMode(kiOnPin, INPUT);
  pinMode(kdOnPin, INPUT);
  pinMode(potPinA, INPUT);
  pinMode(potPinP, INPUT);
  pinMode(potPinI, INPUT);
  pinMode(potPinD, INPUT);

  Wire.begin();
  setupMPU();

  right_prop.attach(10);
  left_prop.attach(9);

  left_prop.writeMicroseconds(1000);
  right_prop.writeMicroseconds(1000);

  delay(7000);

  lcd.init();
  lcd.backlight();
}
```

***Figure 4.4 Setup() function.***

The setupMPU() function is responsible for initializing the MPU-6050 sensor by configuring its registers with specific values. Here's a breakdown of what each part of the function does:

Wire.beginTransmission(MPU_addr): Initiates a communication session with the MPU-6050 sensor using the I2C protocol.

Wire.write(0x6B): Writes the address of the register to be configured. In this case, it's 0x6B, which corresponds to the power management register.

Wire.write(0): Writes the value 0 to the register specified in the previous step. This value typically configures the sensor to wake up and start measuring.

Wire.endTransmission(): Ends the communication session with the sensor, ensuring that the written data is transmitted and processed.

Initialize the MPU-6050 sensor configuration for Gyrometer. This can be done by choosing the AFS_ SEL setting using register 0x1B (see the MPU-6050 documentation). The options for the gyro meter correspond to bits 3 and 4. You will choose a scale range of ±500 °/s. it is correspond to 00001000 which is 0x8 in hexadecimal.

***Table 1 Gyroscope scale configuration.***

| FS_SEL | Full scale range | LSB sensitivity |
|---|---|---|
| 0 | ±250 °/s | 131 LSB°/s |
| 1 | ±500 °/s | 65.5 LSB°/s |
| 2 | ±1000 °/s | 32.8 LSB°/s |
| 3 | ±2000 °/s | 16.4 LSB°/s |

Initialize the MPU-6050 sensor configuration for accelerometer. This can be done by choosing the AFS_ SEL setting using register 0x1C (see the MPU-6050 documentation). The options for the accelerometer correspond to bits 3 and 4. You will choose a scale range of ±8 g. it is correspond to 00010000 which is 0x10 in hexadecimal.

***Table 2 Accelerometer scale configuration.***

| FS_SEL | Full scale range | LSB sensitivity |
|---|---|---|
| 0 | ±2 g | 16384 LSB/g |
| 1 | ±4 g | 8192 LSB/g |
| 2 | ±8 g | 4096 LSB/g |
| 3 | ±16 g | 2048 LSB/g |

In summary, The setupMPU() function initializes the MPU-6050 sensor by configuring its power management, gyroscope, and accelerometer registers with specific values. This process ensures that the sensor is properly initialized and ready for precise data acquisition and measurement. By setting appropriate register values, the function prepares the sensor to accurately capture acceleration and angular velocity data along its axes. This initialization step is essential for the reliable operation of the MPU-6050 sensor in various applications requiring motion sensing and orientation tracking.

```
void setupMPU()
{
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);
  Wire.write(0);
  Wire.endTransmission();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x1B);
  Wire.write(0x8);
  Wire.endTransmission();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x1C);
  Wire.write(0x10);
  Wire.endTransmission();
}
```

***Figure 4.5 SetupMPU() function.***

## 4.4 Reading Data

The ReadAccValues() function reads the accelerometer values from the MPU-6050 sensor by accessing the appropriate registers via the I2C communication protocol. The sensor provides raw data for acceleration along the x, y, and z axes, which are then converted to physical units and stored in the variables AcX, AcY, and AcZ, respectively.

Similarly, the ReadGyroValues() function reads the gyroscope values from the sensor. The raw data for angular velocity along the x, y, and z axes is retrieved and converted to physical units before being stored in the variables GyX, GyY, and GyZ, respectively.

In the context of the MPU-6050 sensor, the values 4096 and 65.5 are scaling factors used to convert the raw sensor data into meaningful physical units.

For the accelerometer the raw data output by the accelerometer is represented as a 16-bit signed integer. The value 4096 represents the full range of the sensor's output (±16g). By dividing the raw data by 4096, we scale the data to the range of ±1g (acceleration due to gravity). The offset values (-0.02 for AcX, +0.035 for AcY) represent calibration adjustments specific to the sensor setup.

For the gyroscope similarly, the raw data output by the gyroscope is represented as a 16-bit signed integer. The value 65.5 represents the sensitivity of the sensor in units of LSB (Least Significant Bit) per degree per second (°/s). By dividing the raw data by 65.5, we scale the data to obtain the angular velocity in degrees per second (°/s).

These scaling factors are necessary to interpret the raw sensor data accurately and convert it into physical units relevant to the application. Adjusting these scaling factors ensures that the sensor readings align with the expected range and resolution for the specific measurement requirements.

```
void ReadAccValues()
{
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B);
  Wire.endTransmission();
  Wire.requestFrom(MPU_addr,6);
  AcX=Wire.read()<<8|Wire.read();
  AcY=Wire.read()<<8|Wire.read();
  AcZ=Wire.read()<<8|Wire.read();

  AcX = AcX / 4096.0-0.02;
  AcY = AcY / 4096.0+0.035;
  AcZ = AcZ / 4096.0;
}

void ReadGyroValues()
{
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x43);
  Wire.endTransmission();
  Wire.requestFrom(MPU_addr,6);
  GyX=Wire.read()<<8|Wire.read();
  GyY=Wire.read()<<8|Wire.read();
  GyZ=Wire.read()<<8|Wire.read();

  GyX = GyX / 65.5;
  GyY = GyY / 65.5;
  GyZ = GyZ / 65.5;
}
```

***Figure 4.6 Reading accelerometer and gyroscope Data.***

The ReadPotValues() function reads the values from potentiometers connected to the Arduino and adjusts control parameters accordingly. Here's what each part of the function does:

potValueA = analogRead(potPinA), Reads the analog voltage value from the potentiometer connected to pin potPinA and stores it in the variable potValueA.

desired_angle = map(potValueA, 0, 1022, -80, 80), Maps the analog input from the potentiometer to a desired angle range (-80 to 80 degrees).

Reads the state of the switches connected to pins kpOnPin, kiOnPin, and kdOnPin to determine whether to adjust the proportional (kp), integral (ki), and derivative (kd) control parameters, respectively.

If the switch is in the LOW state (pressed or activated), the corresponding potentiometer value is read (potValueP, potValueI, or potValueD).

The analog input is mapped to a specific range relevant to the control parameter (kp, ki, or kd) and converted to a double precision floating-point value.

The converted value is rounded and scaled appropriately to match the desired range for the control parameter.

If the switch is in the HIGH state (not pressed or deactivated), the control parameter is set to 0.

This function dynamically adjusts the PID control parameters (kp, ki, kd).

```
void ReadPotValues()
{
  potValueA = analogRead(potPinA);
  desired_angle = map(potValueA, 0, 1022, -80, 80);

  kpOnState = digitalRead(kpOnPin);
  if (kpOnState == LOW)
  { potValueP = analogRead(potPinP);
    kp = static_cast<double>(map(potValueP, 0, 1022, 0, 170));
    kp = round(kp) / 100.0; }
  else
  { kp = 0; }

  kiOnState = digitalRead(kiOnPin);
  if (kiOnState == LOW)
  { potValueI = analogRead(potPinI);
    ki = static_cast<double>(map(potValueI, 0, 1022, 0, 100));
    ki = round(ki) / 100000.0; }
  else
  { ki = 0; }

  kdOnState = digitalRead(kdOnPin);
  if (kdOnState == LOW)
  { potValueD = analogRead(potPinD);
    kd = static_cast<double>(map(potValueD, 0, 1022, 0, 120));
    kd = round(kd) / 100.0; }
  else
  { kd = 0; }
}
```

***Figure 4.7 ReadPotValues() function.***

## 4.5 Display Data

The PrintLCD() function displays key parameters and values on an LCD screen. It prints the proportional (kp), integral (ki), and derivative (kd) constants used in the control algorithm, along with the desired angle (desired_angle) and the current roll angle (RollAngle). The function formats the information neatly, making it easy for users to monitor and understand the system's behavior in real-time.

```
void PrintLCD()
{
  lcd.setCursor(0, 0); lcd.print("kp="); lcd.setCursor(3, 0); lcd.print(kp);

  lcd.setCursor(7, 0); lcd.print("ki="); lcd.setCursor(9, 0); lcd.print(ki, 5);
  lcd.setCursor(9, 0); lcd.print("=");

  lcd.setCursor(0, 1); lcd.print("kd="); lcd.setCursor(3, 1); lcd.print(kd);

  lcd.setCursor(7, 1); lcd.print("DA"); lcd.setCursor(9, 1);
  lcd.print(desired_angle); lcd.setCursor(12, 1); lcd.print("/");
  lcd.setCursor(13, 1); lcd.print(RollAngle);
}
```

***Figure 4.8 PrintLCD() function.***

## 4.6 PID

The pid() function calculates the output of a PID (Proportional-Integral-Derivative) controller based on the current error angel. Here's a breakdown of what each part of the function does:

$$proportional = error$$

Assigns the current error value to the proportional term of the PID controller.

$$integral = \int error \times dt$$

Calculates the integral term of the PID controller. It is hard sometimes to calculate an integral in a microcontroller and it would take a lot of time, so by doing a killing approximation we can say the integral is basically the sum of areas (error x dt).

$$integral = \sum error \times dt$$

In coding it is easy to implement a summation when there's a loop.

$$integral_{i+1} = integral_i + error \times dt$$

If the error is within a specified range (-4 to 4 in this case), the integral term is updated by adding the product of the error and time step (dt) to the previous integral value. Otherwise, the integral term is reset to 0 to prevent it from saturation.

$$derivative = \frac{d(error)}{dt} = \frac{\Delta error}{\Delta t} = \frac{error - previous}{dt}$$

Calculates the derivative term of the PID controller by taking the difference between the current error and the previous error, divided by the time step (dt).

$$previous = error$$

Updates the previous error value for use in the next iteration of the PID calculation.

Returns the overall PID output, calculated as the sum of the proportional, integral, and derivative terms, each multiplied by their respective coefficients (kp, ki, and kd). This function effectively computes the PID control signal based on the current error, adjusting the system's behavior to minimize deviations from the desired setpoint.

```
double pid(double error)
{
  proportional = error;
  integral = (error > -4 && error < 4) ? (integral + error * dt) : 0;
  derivative = (error - previous) / dt;
  previous = error;
  return (kp * proportional) + (ki * integral) + (kd * derivative);
}
```

***Figure 4.9 pid() function.***

## 4.7 Main function

To measure the absolute roll and pitch angles of a Drone, we can explore two methods using data from the MPU-6050 sensor:

**Integrating Gyro Rotation Rates:** One approach is to integrate the rotation rates measured by the gyroscope to calculate the change in angles over time. This method involves summing up the rotation rate values over small time intervals (iterations) to estimate the current angle. However, this method can introduce errors, especially during yaw movements, where changes in direction can affect pitch and roll angles even when there's no rotation rate along those axes.

$$Angle_{pitch}(\mathrm{k}) = \int_0^{k\times Ts} Rate_{pitch}(t)\mathrm{d}t$$

With Ratepitch in degrees per second (°/s), Anglepitch in degrees (°), Ts the duration of one iteration and k the number of iterations. Discretization of this integral to use in your code gives the equation:

$$Angle_{pitch}(k) = Angle_{pitch}(k-1) + Rate_{pitch}(k)\cdot Ts$$

**Using Accelerometer Data:** The accelerometer measures acceleration along the X, Y, and Z axes. Through some clever mathematical equations, this accelerometer property will enable you to calculate the exact roll and pitch angles of your quadcopter. Let's assume you roll around the X axis until you reach the angle $\theta_{roll}$. To visualize this transformation, a box bounded by the X, Y and Z directions is sketched on the figure below.

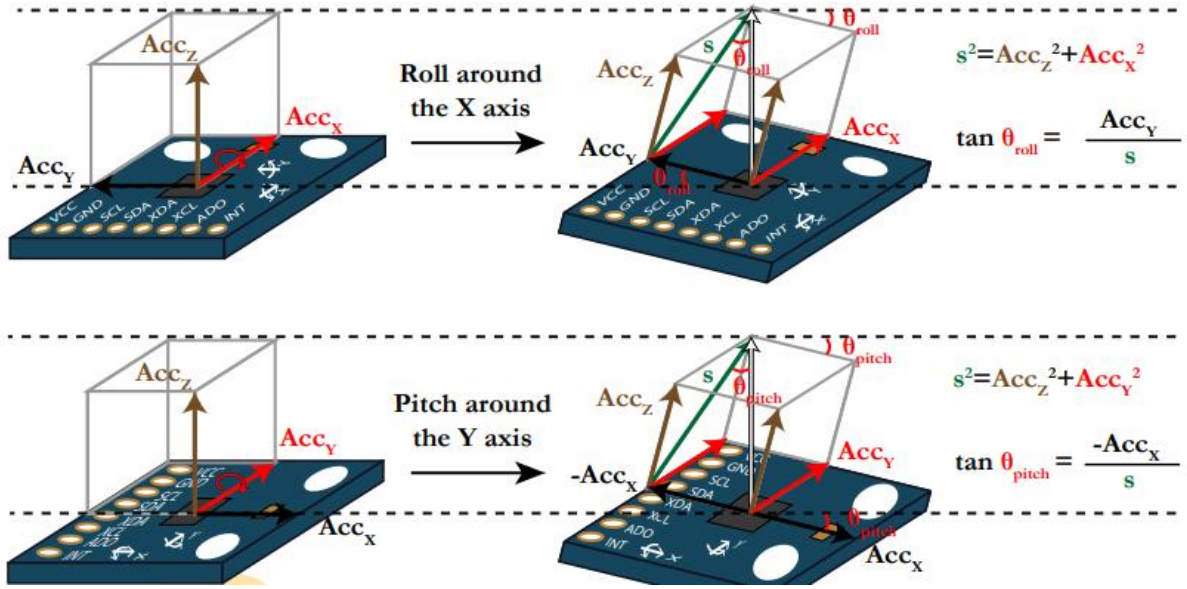


***Figure 4.10 Angle Calculation.***

From your basic trigonometry knowledge, you know that the tangent of the angle of a triangle is equal to the length of the opposite side of the triangle divided by the length of the adjacent side of the triangle. In the case of the angle $\theta_{roll}$, the opposite side is equal to $Acc_y$ while the adjacent side is equal to a certain length s.

Using the Pythagoras rule on the triangle formed by s, $Acc_x$ and $Acc_z$, you are able to derive that $s^2 = Acc_x{}^2 + Acc_z{}^2$, thus the pitch angle can be expressed by the equation:

$$\theta_{pitch} = \tan^{-1}\left(\frac{Acc_y}{\sqrt{Acc_x{}^2 + Acc_z{}^2}}\right)$$

Through similar reasoning and with the help of the next figure you can express the roll angle by:

$$\theta_{roll} = \tan^{-1}\left(\frac{-Acc_x}{\sqrt{{Acc_y}^2 + {Acc_z}^2}}\right)$$

Now this all wouldn't mean anything if you don't know how to code it so there's a full detailed coding:

The loop() function orchestrates the real-time operation of the control system. It begins by computing the elapsed time since the previous iteration, crucial for time-dependent calculations. The state of the manual override switch is then checked: when activated, it adjusts control parameters based on potentiometer readings through the ReadPotValues() function; otherwise, default parameters are used.

Next, sensor data acquisition occurs. The accelerometer and gyroscope values are read from the MPU-6050 sensor, providing essential information about the system's orientation and motion. These values are utilized in subsequent calculations to determine the system's current roll and pitch angles. Then using a complementary filter to get angle from both sensors. With 80% from accelerometer and 20% from gyroscope. Hence the accelerometer can be affected by vibration and there's not in this project and the gyroscope is affected with iteration time that take a loop which is high.

Following this, the control parameters and sensor readings are displayed on an LCD screen to provide real-time feedback to the user.

The PID control algorithm is then applied to compute the control signal necessary to minimize the error between the desired and actual roll angles. The PID output is constrained within specified limits to prevent excessive control actions.

Finally, the computed control signal is translated into motor commands. The throttle values for the left and right motors are adjusted accordingly based on the PID output, ensuring that the system responds appropriately to maintain its desired orientation. These throttle values are constrained within acceptable bounds before being sent to the motor controllers.

This iterative process continues indefinitely, enabling the control system to continuously adjust motor outputs based on sensor feedback, ultimately achieving and maintaining the desired orientation of the system.

```
void loop()
{
  now = millis();
  dt = (now - PrevTime) / 1000.0;
  PrevTime = now;

  manualOverideState = digitalRead(manualOveridePin);
  if (manualOverideState == LOW)
  { ReadPotValues(); }
  else
  { kp = 0.85;
    ki = 0.0005;
    kd = 0.60;
    desired_angle = 0; }

  ReadAccValues();
  ReadGyroValues();

  AccAnglePitch = atan(AcY / sqrt(AcX * AcX + AcZ * AcZ)) * 180 / PI;
  AccAngleRoll = -atan(AcX / sqrt(AcY * AcY + AcZ * AcZ)) * 180 / PI;

  PitchAngle = 0.2 *(PitchAngle + GyX*dt) + 0.8*AccAnglePitch;
  RollAngle = 0.2 *(RollAngle + GyY*dt) + 0.8*AccAngleRoll;

  PrintLCD();

  error = RollAngle - desired_angle;
  PID_output = pid(error);

  PID_output = constrain(PID_output, -1000, 1000);

  pwmLeft = throttle - PID_output;
  pwmRight = throttle + PID_output;

  pwmLeft = constrain(pwmLeft, 1000, 2000);
  pwmRight = constrain(pwmRight, 1000, 2000);

  left_prop.writeMicroseconds(pwmLeft);
  right_prop.writeMicroseconds(pwmRight);
}
```

***Figure 4.11 Main function.***

## 4.8 Results

Combining the mechanical and electrical structures with coding creates the magic that powers our drone stabilizer. To achieve this, we first connect the Electronic Speed Controllers (ESCs) to the power supply and wait for them to start beeping. Then, we connect the Arduino to the power bank and allow a 7-second period for the ESC calibration. After this process, we observe the stabilizer in action, bringing our creation to life.

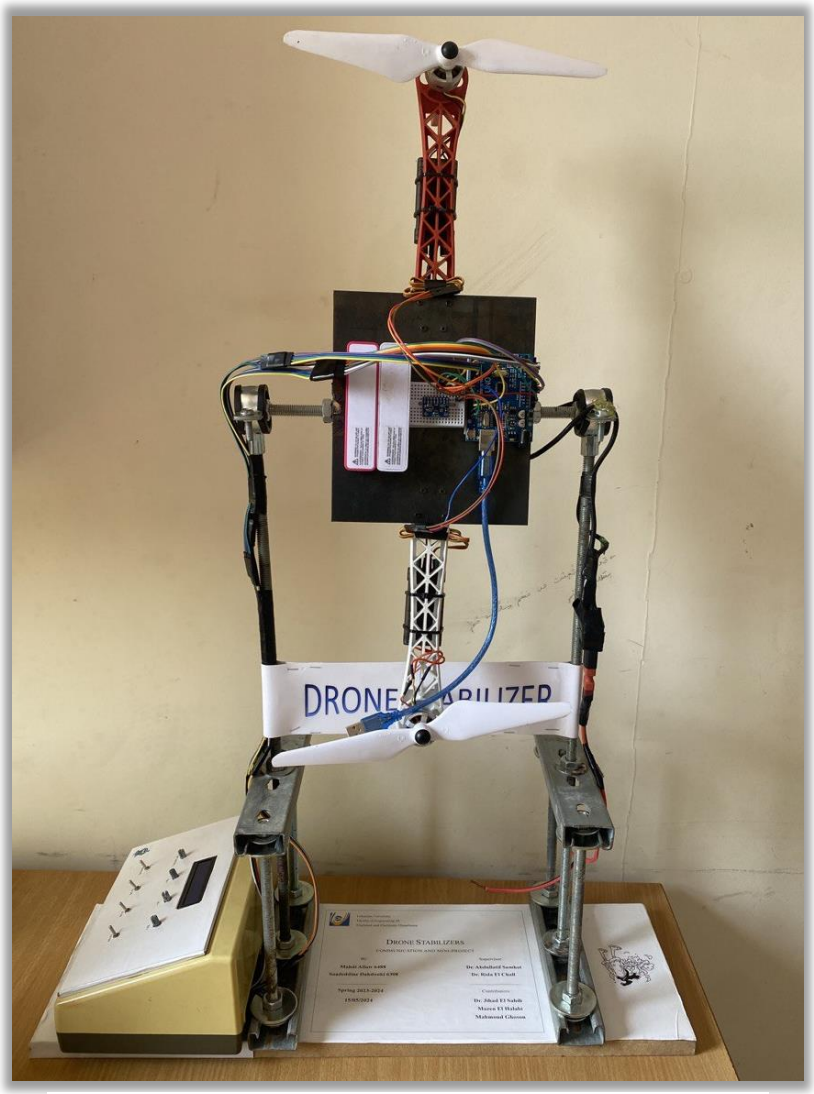

*Figure 4.12 Result.*

## 4.9 Conclusion

In this chapter, we have detailed the implementation process of our drone stabilizer system. We outlined the steps for connecting the mechanical and electrical components and described the integration with the Arduino for PID control and calibration. Through meticulous software coding and rigorous testing procedures, we successfully achieved a stable and functional drone stabilizer. The results demonstrate the effectiveness of our approach, showcasing the seamless combination of hardware and software to create a reliable and responsive system. This accomplishment underscores the importance of precise calibration and careful integration in developing advanced robotic solutions.

# CHAPTER 5 CONCLUSION

## 5.1 Conclusion

We began by establishing the fundamental principles of control systems and their critical role in maintaining stability across various engineering applications. We then highlighted the importance of stable flight for drones, emphasizing its contribution to safe and accurate operations. The concept of PID control was introduced, a prevalent method used in drones to adjust motor speeds based on the discrepancy between desired and actual positions or angles.

Following the foundational concepts, we delved into the intricate details of the system's design. This included a thorough examination of the mechanical structure, the electrical components that power the system, and the system architecture that dictates how these components interconnect. We emphasized the importance of defining hardware and software requirements to ensure the system functions as intended within its environment. Essential components like potentiometers, switches, the LCD display, motors and their corresponding Electronic Speed Controllers (ESCs), the MPU-6050 sensor, and the microcontroller were each explained in detail, along with their specific functionalities within the system.

Chapter 3 explored the system's functionality, providing insights into how user inputs from potentiometers and switches are processed by the microcontroller. We explained how the microcontroller interacts with sensors and actuators (motors) to achieve and maintain stability. The role of the LCD display in providing real-time feedback on the system's performance was also highlighted.

Chapter 4 focused on the implementation process, outlining the libraries and variables utilized in the code. We explained the functionalities of various functions responsible for tasks such as setting up the system, reading sensor data, calculating control signals, and adjusting motor outputs. The chapter also detailed the methods used to measure the drone's absolute roll and pitch angles, incorporating both accelerometer data and gyroscope rotation rates for improved accuracy. A complementary filter was introduced to further enhance the accuracy of the angle measurements. Finally, the loop function, which orchestrates the real-time operation of the system, was discussed. This function continuously acquires sensor data, calculates control signals, and adjusts motor outputs to maintain the desired orientation of the drone.

The final chapter presented the conclusion of the project. We described the process of setting up and calibrating the system, including the crucial step of ESC calibration. The chapter culminated in highlighting the successful creation of a functional and stable drone stabilizer system.

In conclusion, this project serves as a testament to the effectiveness of combining a well-designed control system with meticulous integration of hardware and software. The documented process of designing, implementing, and testing a drone stabilizer control system using a PID controller provides a valuable resource for anyone interested in developing or understanding drone stabilization technology. This project not only achieves its goal of creating a functional stabilizer but also offers valuable insights into the intricate interplay between various components that contribute to a stable and well-controlled drone.

## 5.2 Future work

You can notice in Chapter 4, we not only calculated the roll angle but also the pitch angle. did you ever ask yourself why? The answer is simple, to demonstrate how easy it is to calculate and implement these angles, paving the way for future work. Once these angles are calculated, the next step is to calculate the error between the desired and actual angles and feed this error into the PID controller.

As for the yaw movement, that's a challenge left for future work. Implementing a fully functional drone involves tuning the PID constants to ensure stability. But what about the 9 constants now! Is tuning the constants easy and enough for stabilization?

# REFERENCES

[1] farid-golnaraghi-benjamin-c-kuo-automatic-control-systems.

[2] Modern Control Systems 13th.

[3] Katsuhiko Ogata _ Modern Control Engineering 5th Edition.

[4] How to mechatronics, https://howtomechatronics.com/

[5] Gate analysis made simple, https://gunjanpatel.wordpress.com/

[6] https://electronoobs.com/

[7] carbon Aeronautics, https://www.youtube.com/@carbonaeronautics .

# APPENDICES

Appendix A: Testing brushless motor.

```
#include <Servo.h>

Servo ESC;
int throttle = 1300;

int potPin = A0, potValue;
void setup()
{
  pinMode(potPin, INPUT);
  ESC.attach(9);
  ESC.writeMicroseconds(1000);
  delay(7000);
}

void loop()
{
  potValue = analogRead(A0);
  potValue = map(potValue, 0, 1023, 0, 700);
  ESC.writeMicroseconds(throttle + potValue);
}
```

Appendix B: MPU testing and calibration.

```
#include<Wire.h>

const int MPU_addr=0x68;  // I2C address of the MPU-6050
float AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;

float AcX_Angle, AcY_Angle, AcZ_Angle, GyX_Angle, GyY_angle, GyZ_angle, TotalX_Angle,
TotalY_Angle, TotalZ_Angle;
float RateRoll, RatePitch, RateYaw;
float AngleRoll, AnglePitch;
float elapsedTime, time, timePrev;

void setup(){
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);
  Wire.write(0);     // set to zero (wakes up the MPU-6050)
  Wire.endTransmission(true);
  Serial.begin(9600);
  time = millis();
}
void loop(){
  timePrev = time;  // the previous time is stored before the actual time read
  time = millis();  // actual time read
  elapsedTime = (time - timePrev) / 1000;

  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr,14,true);  // request a total of 14 registers
  AcX=Wire.read()<<8|Wire.read();
  AcY=Wire.read()<<8|Wire.read();
  AcZ=Wire.read()<<8|Wire.read();
  Tmp=Wire.read()<<8|Wire.read();
  GyX=Wire.read()<<8|Wire.read();
  GyY=Wire.read()<<8|Wire.read();
  GyZ=Wire.read()<<8|Wire.read();

  // ±250°/s
  RateRoll = GyX / 131.0;
  RatePitch = GyY / 131.0;
  RateYaw = GyZ / 131.0;

  // ±500°/s
  // RateRoll=GyX/65.5;
  // RatePitch=GyY/65.5;
  // RateYaw=GyZ/65.5;
```

```
  // for 4g
  AcX = AcX / 16384.0-0.02;
  AcY = AcY / 16384.0+0.035;
  AcZ = AcZ / 16384.0;

  // for 2g
  // AcX=AcX/4096-0.02;
  // AcY=AcY/4096+0.05;
  // AcZ=AcZ/4096;

  AnglePitch = atan(AcY/sqrt(AcX*AcX+AcZ*AcZ))*1/(3.142/180);
  AngleRoll = -atan(AcX/sqrt(AcY*AcY+AcZ*AcZ))*1/(3.142/180);

  TotalX_Angle = 0.5 *(TotalX_Angle + RateRoll*elapsedTime) + 0.5*AnglePitch;
  TotalY_Angle = 0.5 *(TotalY_Angle + RatePitch*elapsedTime) + 0.5*AngleRoll;

  // Serial.print("Acceleration X [g]= ");
  // Serial.print(AcX);
  // Serial.print(" Acceleration Y [g]= ");
  // Serial.print(AcY);
  // Serial.print(" Acceleration Z [g]= ");
  // Serial.println(AcZ);
  Serial.print("Roll angle [°]= ");
  Serial.print((int)TotalY_Angle);
  Serial.print(" Pitch angle [°]= ");
  Serial.println((int)TotalX_Angle);
  // Serial.print(" Acc Roll angle [°]= ");
  // Serial.print(AngleRoll);
  // Serial.print(" Acc Pitch angle [°]= ");
  // Serial.println(AnglePitch);

}
```

Appendix C: Debugging.

```
void Print()
{
  Serial.print(desired_angle);
  Serial.print(",");
  Serial.println(0);

  Serial.print("Acceleration X [g]= ");
  Serial.print(AcX);
  Serial.print(" Acceleration Y [g]= ");
  Serial.print(AcY);
  Serial.print(" Acceleration Z [g]= ");
  Serial.println(AcZ);

  Serial.print(" Acc Roll angle [°]= ");
  Serial.print(AccAngleRoll);
  Serial.print(" Acc Pitch angle [°]= ");
  Serial.println(AccAnglePitch);

  Serial.print("Gyro X = ");
  Serial.print(GyX);
  Serial.print(" Gyro Y = ");
  Serial.print(GyY);
  Serial.print(" Gyro Z = ");
  Serial.println(GyZ);

  Serial.print("Roll angle [°]= ");
  Serial.print(RollAngle);
  Serial.print(", ");
  Serial.print(" Pitch angle [°]= ");
  Serial.print(PitchAngle);
  Serial.print(", ");
  Serial.print(pwmLeft);
  Serial.print(", ");
  Serial.print(1300);
  Serial.print(", ");
  Serial.println(pwmRight);

  Serial.print("DA = ");
  Serial.print(desired_angle);
  Serial.print(", kp = ");
  Serial.print(kp, 5);
  Serial.print(", ki = ");
  Serial.print(ki, 5);
  Serial.print(", kd = ");
  Serial.print(kd, 5);
  Serial.print(", potValueA = ");
  Serial.print(potValueA);
  Serial.print(", potValueP = ");
  Serial.print(potValueP);
  Serial.print(", potValueI = ");
  Serial.print(potValueI);
  Serial.print(", potValueD = ");
  Serial.println(potValueD);
}
```