CHAPTER I

INTRODUCTION

1.1 Statement of the Problem

Conventional DC motor control systems often rely on open-loop configurations that lack dynamic responsiveness and precision, especially when the desired speed varies during operation. In such systems, external disturbances like load changes, voltage fluctuations, and mechanical resistance can significantly affect motor performance, leading to inaccurate or unstable motion. These limitations are especially problematic in applications requiring real-time directional control and speed regulation, such as turret-like systems where bi-directional motor movement must be continuously adjusted by a user.

In the context of this project, the use of a joystick to define the desired motor speed introduces variability that must be accurately interpreted and tracked. Without a closed-loop feedback mechanism, the actual motor speed may deviate significantly from the user-defined setpoint, resulting in sluggish response, overshooting, or drift. Moreover, direct manual control is susceptible to inconsistencies and lacks the ability to self-correct during operation.

To address these challenges, there is a need for an integrated control system that combines real-time user input, motor feedback via an encoder sensor, and a PID controller capable of dynamically adjusting motor output to match the desired speed. This system must also support reliable direction control using a motor driver (RPWM/LPWM) and maintain stable performance across a range of speeds and directions. The use of a voltage regulator further ensures consistent power delivery, protecting the circuit from supply-related instability.

Thus, this project aims to design and implement a closed-loop DC motor speed control system with adjustable setpoint input via joystick, real-time RPM feedback using an encoder module, and optimized PID-based correction. The solution will provide a low-cost, responsive, and accurate control platform suitable for educational robotics and mechanical positioning applications, such as those mimicking turret systems or tracked vehicles.

CHAPTER II

METHODOLOGY

2.1 Conceptual Framework

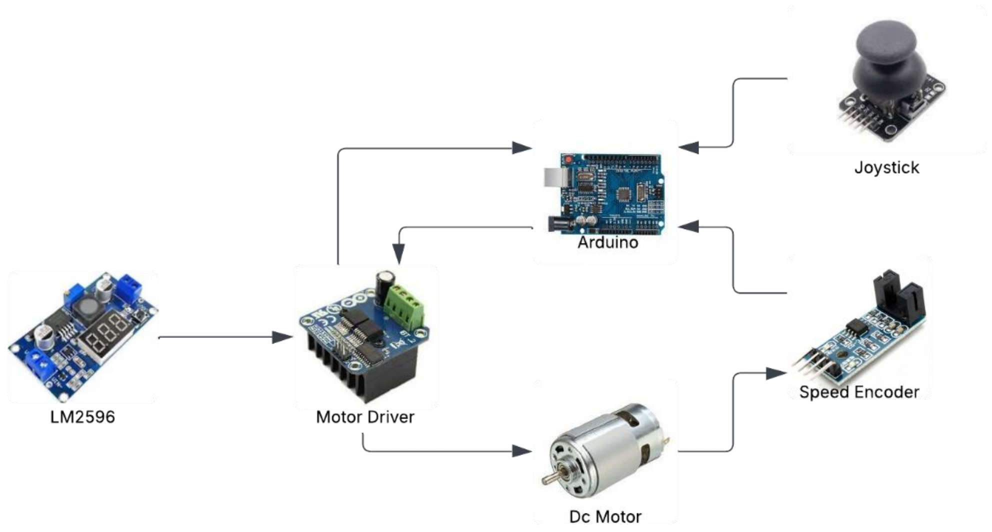
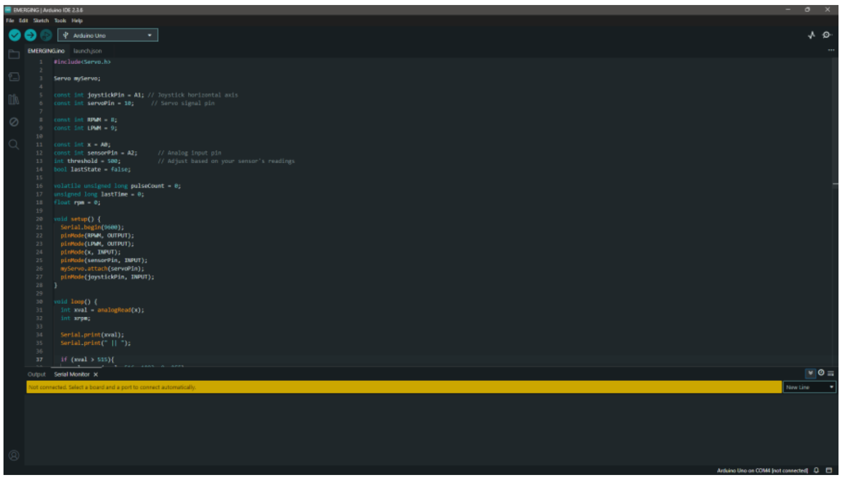


Figure 2.1: Conceptual Framework

The conceptual framework illustrates a closed-loop motor control system centered around an Arduino microcontroller. A joystick module serves as the primary user input device, sending directional and speed signals to the Arduino. The Arduino processes this input and send corresponding control signals to the BTS7960 motor driver, which then powers and drives the DC motor. To ensure appropriate voltage levels, an LM2596 voltage regulator supplies stable power to the motor driver. As the motor operates, a speed encoder sensor monitors its rotational speed and provides real-time feedback to the Arduino. This feedback allows the system to adjust motor performance dynamically, enabling precise speed and direction control based on user input and motor behavior. The integration of input (joystick), processing (Arduino), output (motor), and feedback (encoder) components forms a responsive and efficient motor control system.

2.2 Software and Hardware Used

2.2.1 Software


EMERGING | Arduino IDE 2.3.6
File Edit Sketch Tools Help
Arduino Uno
EMERGING.ino
launch.json
Output
Serial Monitor
New Line
Arduino Uno on COM4 [not connected]

```
#include<Servo.h>

Servo myServo;

const int joystickPin = A1; // Joystick horizontal axis
const int servoPin = 10;     // Servo signal pin

const int RPWM = 8;
const int LPWM = 9;

const int x = A0;
const int sensorPin = A2;       // Analog input pin
int threshold = 500;            // Adjust based on your sensor's readings
bool lastState = false;

volatile unsigned long pulseCount = 0;
unsigned long lastTime = 0;
float rpm = 0;

void setup() {
  Serial.begin(9600);
  pinMode(RPWM, OUTPUT);
  pinMode(LPWM, OUTPUT);
  pinMode(x, INPUT);
  pinMode(sensorPin, INPUT);
  myServo.attach(servoPin);
  pinMode(joystickPin, INPUT);
}

void loop() {
  int xval = analogRead(x);
  int xrpm;

  Serial.print(xval);
  Serial.print(" || ");

  if (xval > 515){
```

```
Not connected. Select a board and a port to connect automatically.
```

Figure 2.2: Arduino IDE

The primary software used in this project is the Arduino IDE. The Arduino IDE serves as the development platform for writing, compiling, and uploading code to the Arduino microcontroller. It supports C and C++ programming languages and provides a user-friendly interface for configuring input/output pins, integrating sensor data, and controlling actuators such as motors and servos. In this project, the Arduino IDE was used to implement logic for reading button or joystick input, interpreting IR sensor data to calculate RPM, and sending control signals to the DC motor and servo. Built-in libraries and serial monitoring tools were also utilized for debugging and real-time performance testing.

2.2.2 Hardware

Figure 2.3: Arduino IDE

1. Arduino Uno - serves as the central controller of the project. It processes input from the user, reads sensor data from the IR sensor, and generates output signals to control the DC motor and servo motor. It also manages data flow from the rotating IR sensor through the slip ring and performs real-time speed monitoring. The Arduino Uno is highly suitable for this project due to its reliability, ease of programming, and broad compatibility with various sensors and actuators.

   General Specifications:

   - Microcontroller: ATmega328P
   - Operating Voltage: 5V
   - Input Voltage (recommended): 7–12V
   - Digital I/O Pins: 14
   - Analog Input Pins: 6
   - Memory: 32 KB Flash, 2 KB SRAM, 4 KB EEPROM

Figure 2.4: LM 2596

2. LM2596 - is used to regulate power within the system. Since the DC motor and servo require specific operating voltages different from the Arduino, this module converts a higher voltage input (like 12V) to a stable 5V or 6V. It ensures each component receives the correct voltage, helping protect against overvoltage and enabling reliable operation throughout the control loop.

   General Specifications:

   - Input Voltage: 4V – 40V
   - Output Voltage: 1.25V – 37V
   - Max Output Current: 2A
   - Efficiency: Up to 92%
   - Adjustable via onboard potentiometer

Figure 2.5: Motor Driver

3. Motor Driver - as the interface between the Arduino and the DC motor. It enables bidirectional rotation (clockwise and counterclockwise) and allows speed control using PWM signals from the Arduino. In this project, the motor driver facilitates joystick-based direction control while supporting PID-driven speed adjustments.

   General Specifications:

   - Operating Voltage: 5V – 35V
   - Max Current: 2A per channel
   - Control: PWM speed control and direction
   - Number of Channels: 2
   - Includes onboard heat sink

Figure 2.6: DC motor

4. DC Motor - serves as the primary actuator responsible for the horizontal rotation of the turret-like platform, replicating the turning movement of a military tank. In this project, the motor is directly coupled to the rotating platform without a gear system, allowing for a faster response and simpler mechanical design. The motor's rotational speed is dynamically adjusted based on real-time feedback from the IR speed encoder, ensuring stable and accurate motion through PID control.

   General Specifications:

   - Voltage: 6V – 12V
   - Maximum Speed: 95 RPM
   - Torque: Moderate, suitable for lightweight turret rotation
   - Output Shaft: Compatible with encoder disk and rotating platform
   - Features: Direct drive, bidirectional control via motor driver
   - Output Shaft: Compatible with encoder disk and gear

Figure 2.7: Joystick

5. Joystick - provides user input to control both axes of motion. The X-axis controls the speed and direction of the DC motor (left/right), while the Y-axis can be assigned to control the servo motor for vertical movement (up/down). The variable analog output is mapped to adjust motor speed dynamically, acting as the system’s setpoint controller for the PID loop.

   General Specifications:

   - Operating Voltage: 5V
   - Output: Two analog signals (X and Y), one digital (press)
   - Analog Range: 0V – 5V
   - Used to generate variable speed setpoints and direction

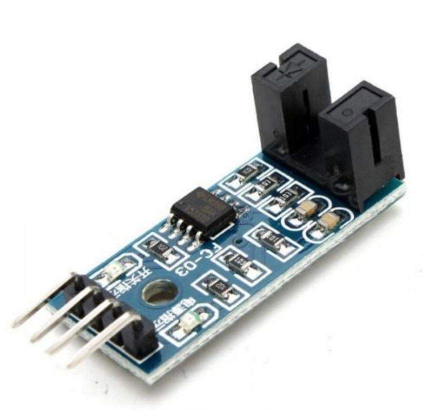


Figure 2.8: Speed Encoder Sensor

6. Speed Sensor - to measure real-time RPM of the DC motor. An encoder disk attached to the motor shaft reflects or interrupts the IR beam, generating pulses. These pulses are counted by the Arduino to determine rotational speed, which is then used in the PID controller to adjust motor output and stabilize speed.

   General Specifications:

   - Operating Voltage: 3.3V – 5V
   - Output: Digital (pulses for speed)
   - Detection: Infrared reflection or beam interruption
   - Mounted to read rotation from a gear-mounted disk
   - Enables closed-loop speed feedback

Figure 2.8: Slip ring

7. Slip Ring - is utilized in this project to provide a stable electrical connection between the stationary base (Arduino) and the rotating turret platform. This component allows continuous 360-degree rotation of the DC motor-driven turret without twisting or damaging wires. In this setup, the slip ring transmits control signals and power to the **servo motor** mounted on the rotating section, enabling vertical (up and down) aiming. The use of the slip ring ensures uninterrupted signal flow and mechanical freedom for multi-axis movement, crucial for a turret-style system.

   General Specifications:

   - Channels: 3 wires
   - Voltage Rating: 0–240V
   - Current Rating: Up to 2A per wire
   - Rotational Speed: Up to 300 RPM
   - Features: Supports continuous rotation with low noise and signal integrity

2.3 Budget

The estimated total budget for the project is approximately ₱1,584.00. A majority of the expenses were allocated to the core electronic components required for the motor control system, including the DC motor, Arduino Uno, motor driver, and slip ring. Unlike more complex systems requiring custom mechanical fabrication, this project focused on functionality and cost-efficiency, eliminating the need for outsourced services such as 3D printing. The remaining portion of the budget covered essential modules such as the joystick, speed encoder, and LM2596 voltage regulator. All components were sourced locally or online at minimal cost, making the project both affordable and practical for prototyping.

Table 2.1: Breakdown of Costs

Category	**Costs (PHP)**
Arduino Uno	155.00
LM 2596	140.00
Motor Driver	305.00
DC Motor	700.00
Joystick	49.00
Speed Sensor	35.00
Slip Ring	200.00
TOTAL	1584.00

2.4 Duties and Responsibilities

2.2.1 Hardware Management

This group of project members is responsible for overseeing the selection, integration, and testing of the hardware components used in the real-time DC motor speed control system. This includes configuring the DC motor, servo motor, rotary slip ring, Speed encoder, joystick, LM2596, and motor driver. The team ensures proper hardware interfacing with the Arduino and validates physical system behavior through real-world testing. Responsibilities also include designing the physical layout and ensuring secure connections for stable operation. This responsibility is assigned to Mr. Peter Paul Sanglitan, Mr. Limuen C. Untong, Mr. Jeremiah L. Llaneta, Mr. Benzar S. Esmail, and Ms. Bai Jannah Jheyhan G. Kumpo.

2.2.2 Coding

This project member is in charge of programming and software integration for the Arduino-based motor control system. Tasks include implementing the PID control algorithm, interfacing the Speed encoder sensor for RPM feedback, processing joystick input for setpoint adjustment, and configuring the servo motor for vertical movement. The researcher also ensures real-time responsiveness and parameter tuning within the Arduino IDE environment to achieve stable and accurate motor performance. This responsibility is assigned to Mr. Jeremiah Lu Llaneta.

2.2.3 Documentation

This project member manages the documentation phase of the project, compiling all technical data, system design explanations, and implementation details into a complete and cohesive thesis. Duties include writing project descriptions, maintaining proper formatting, and ensuring that all written outputs align with the guidelines provided by the research adviser. This responsibility is assigned to Mr. Benzar S. Esmail.

CHAPTER III

IMPLEMENTATION DEVELOPMENT

3.1 Design

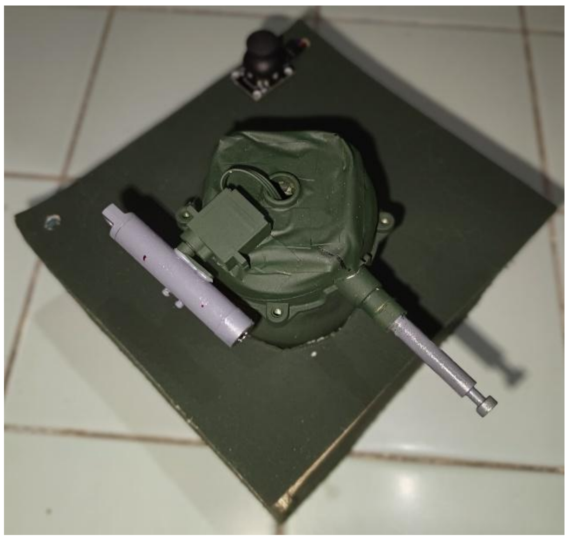


Figure 3.1: Actual Model of the platform

Figure 3.1 illustrates the completed prototype of the DC motor and servo-based turret platform. The system is designed to simulate a military tank turret with two degrees of freedom—horizontal rotation (yaw) driven by a DC motor, and vertical aiming (pitch) controlled by a servo motor. The joystick, mounted at the rear of the base platform, serves as the user input for adjusting the rotation direction and angle. The model's rotary platform houses internal components including the motor driver, IR speed encoder, and rotary slip ring for signal and power transfer. The base platform conceals the Arduino microcontroller and supporting power supply circuitry. This integrated setup allows real-time control of rotation speed and aiming angle, demonstrating precise response to joystick input through a PID-regulated feedback system.

3.2 Diagram

3.2.1 Schematic

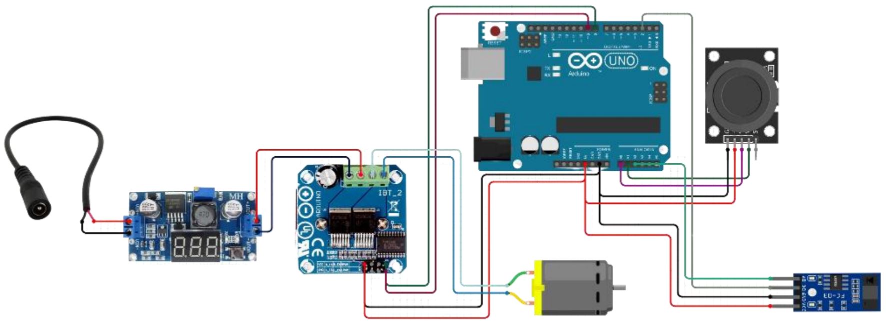


Figure 3.2: Schematic Diagram

Figure 3.2 illustrating how each hardware component is interconnected to form the complete motor control system. The joystick module serves as the primary input device, connected to the Arduino to control the direction and speed of the DC motor. The Arduino microcontroller processes this input and sends corresponding signals to the motor driver, which then powers and controls the DC motor. An IR speed encoder sensor is attached to the motor to provide real-time feedback on its rotational speed. This data is sent back to the Arduino for closed-loop speed monitoring and adjustment. The LM2596 DC-DC step-down converter supplies a stable, regulated voltage to the motor driver and other components, ensuring consistent operation of the system.

3.3 Code

```
#include<Servo.h>

Servo myServo;

double Kp = 1.0;
double Ki = 0.1;
double Kd = 0.05;

double setpoint = 0;
double inputRPM = 0;
double output = 0;
double lastError = 0;
double integral = 0;

unsigned long lastPIDTime = 0;

const int joystickPin = A1;
const int servoPin = 10;
const int RPWM = 8;
const int LPWM = 9;
const int x = A0;
const int sensorPin = A2;

int threshold = 500;
bool lastState = false;
volatile unsigned long pulseCount = 0;
unsigned long lastRPMMeasureTime = 0;
float rpm = 0;

void setup() {
  Serial.begin(9600);
  pinMode(RPWM, OUTPUT);
  pinMode(LPWM, OUTPUT);
  pinMode(x, INPUT);
  pinMode(sensorPin, INPUT);
  myServo.attach(servoPin);
  pinMode(joystickPin, INPUT);
}

void loop() {
  int xval = analogRead(x);
  setpoint = map(xval, 0, 1023, -255, 255);

  int sensorValue = analogRead(sensorPin);
```

```cpp
  bool currentState = (sensorValue > threshold);

  if (currentState && !lastState) {
    pulseCount++;
  }
  lastState = currentState;

  unsigned long currentTime = millis();
  if (currentTime - lastRPMMeasureTime >= 1000) {
    rpm = pulseCount * 60.0;
    pulseCount = 0;
    lastRPMMeasureTime = currentTime;
  }
  inputRPM = rpm;

  double error = setpoint - inputRPM;
  double deltaTime = (currentTime - lastPIDTime) / 1000.0;

  integral += error * deltaTime;
  double derivative = (error - lastError) / deltaTime;

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

  output = constrain(output, -255, 255);

  if (output > 0) {
    analogWrite(RPWM, abs(output));
    analogWrite(LPWM, 0);
  } else if (output < 0) {
    analogWrite(LPWM, abs(output));
    analogWrite(RPWM, 0);
  } else {
    analogWrite(RPWM, 0);
    analogWrite(LPWM, 0);
  }

  lastError = error;
  lastPIDTime = currentTime;

  int joystickValue = analogRead(joystickPin);
  int angle = map(joystickValue, 0, 1023, 45, 135);
  myServo.write(angle);

  Serial.print("Setpoint: "); Serial.print(setpoint);
  Serial.print(" | RPM: "); Serial.print(inputRPM);
  Serial.print(" | Output: "); Serial.println(output);

  delay(20);
}
```

Figure 3.3: Code of the System

Figure 3.3 presents the complete Arduino code implementation that governs the operation of the system. It outlines how each component functions in coordination—such as motor speed regulation, servo control, and joystick input—based on the control logic defined during the system design phase.