Laporan Praktikum IMCLab

Dosen Pengampu:

Assoc. Prof. Dr. Basuki Rahmat, S.Si, MT, ITS-AI

Mata Kuliah:

Mikrokontroler (A081)

Disusun Oleh:

Rizky Ananda Ramadhan (23081010088)

PROGRAM STUDI INFORMATIKA

FAKULTAS ILMU KOMPUTER

UNIVERSITAS PEMBANGUNAN NASIONAL "VETERAN"

JAWA TIMUR

TAHUN 2025

BAB I
PENDAHULUAN

1.1. Latar Belakang

Dalam aplikasi industri, menjaga kestabilan kecepatan motor DC merupakan aspek krusial. Motor DC sering kali mengalami fluktuasi kecepatan akibat perubahan beban atau tegangan sumber. Untuk mengatasi hal ini, diperlukan sistem kendali tertutup (closed-loop system) yang mampu menyesuaikan daya secara otomatis agar kecepatan aktual tetap sesuai dengan target yang diinginkan.

Praktikum ini menggunakan ESP32 sebagai unit pemroses utama. Sistem ini mengintegrasikan sensor encoder untuk umpan balik kecepatan, algoritma PID (Proportional-Integral-Derivative) untuk stabilitas, serta protokol MQTT untuk kendali jarak jauh. Selain itu, komunikasi serial melalui Python juga digunakan untuk pengujian lokal yang cepat.

1.2. Rumusan Masalah

1. Bagaimana cara mengimplementasikan komunikasi antara Python dan ESP32 melalui jalur serial?
2. Bagaimana mekanisme protokol MQTT digunakan untuk mengontrol fungsi On/Off motor secara remote?
3. Bagaimana algoritma PID diterapkan untuk menjaga kestabilan RPM motor DC terhadap target (setpoint) tertentu?

1.3. Tujuan Praktikum

1. Membangun konektivitas antara perangkat keras (ESP32) dengan antarmuka komputer (Python) dan cloud (MQTT).
2. Mengkalibrasi pembacaan RPM menggunakan sensor encoder melalui sistem interrupt.
3. Menganalisis kinerja kontroler PID dalam meminimalkan error kecepatan pada motor DC.

BAB II
TINJAUAN TEORI

2.1. ESP32 dan PWM (Pulse Width Modulation)

ESP32 memiliki fitur PWM hardware yang sangat baik. Kecepatan motor dikontrol dengan mengatur duty cycle pada pin PWM. Dalam praktikum ini, resolusi yang digunakan adalah 8-bit (0-255), yang menentukan tegangan rata-rata yang diberikan ke motor melalui motor driver [1]. Modul PWM pada ESP32 memungkinkan kontrol frekuensi yang presisi untuk berbagai aplikasi otomasi [3].

2.2. Protokol MQTT (Message Queuing Telemetry Transport)

MQTT adalah protokol komunikasi ringan yang berbasis publish-subscribe. Protokol ini ideal untuk IoT karena hemat daya dan bandwidth. Data kecepatan (RPM) dipublikasikan ke broker, sementara perintah kontrol diterima oleh ESP32 melalui topik langganan (subscription) [2]. MQTT sangat efisien untuk perangkat dengan sumber daya terbatas dan jaringan yang tidak stabil [4].

2.3. Kendali PID (Proportional-Integral-Derivative)

PID adalah algoritma kontrol yang paling umum digunakan dalam sistem kendali industri [5].

1) **Proportional (P):** Menghasilkan koreksi sebanding dengan error saat ini.
2) **Integral (I):** Akumulasi error masa lalu untuk menghilangkan error statis (steady-state error).
3) **Derivative (D):** Memprediksi error masa depan untuk meredam osilasi.

2.4. Komunikasi Serial (Python-Arduino)

Library pyserial pada Python memungkinkan komputer mengirim data byte ke port COM mikrokontroler. Ini berguna untuk debugging dan kontrol manual tanpa memerlukan koneksi internet [6]. Komunikasi serial tetap menjadi standar dasar untuk pertukaran data antara PC dan unit mikrokontroler.

BAB III
METODOLOGI DAN HASIL

3.1. Lingkungan dan Alat Praktikum

Hardware: ESP32, Motor DC, Encoder Sensor, Motor Driver (L298N/sejenis), PC.

Software: Arduino IDE, Python 3.11, MQTT Broker (EMQX).

Konfigurasi Pin:

- Motor: GPIO 27, 26, 12 (PWM).
- Encoder: GPIO 13 (Interrupt).
- LED: GPIO 2.

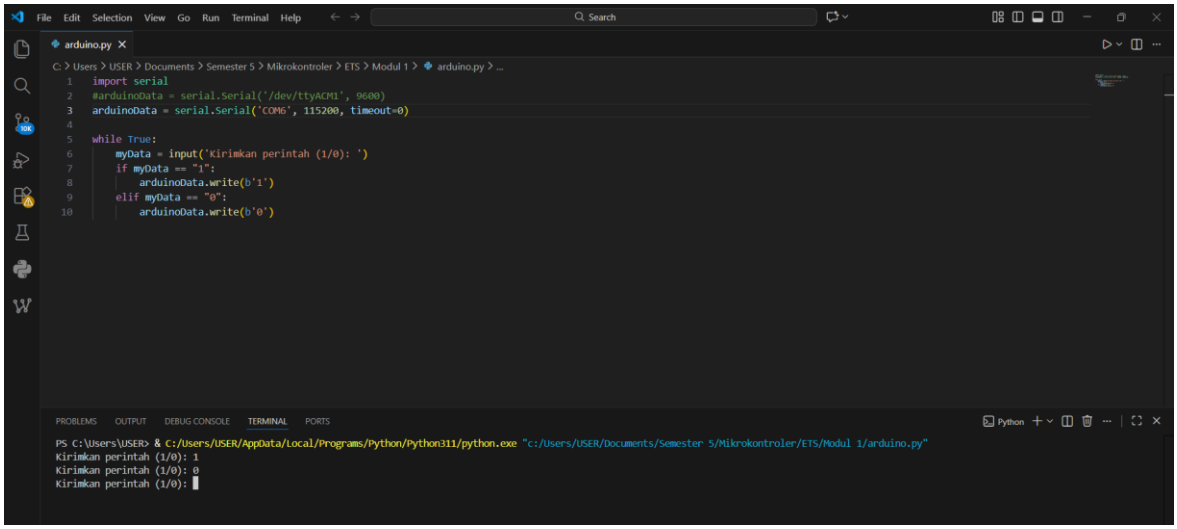
3.2. Langkah-Langkah Praktikum dan Hasil

Tahap 1: Pengujian Kontrol Serial via Python

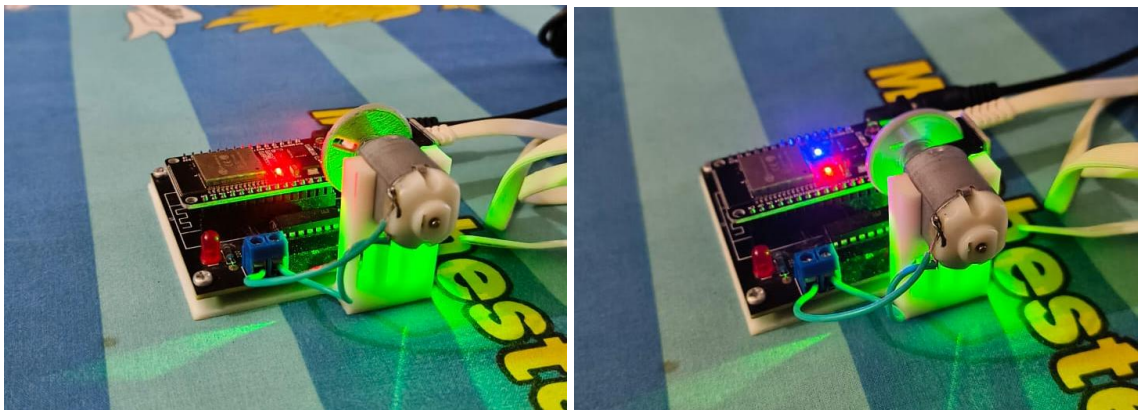
Kode Program:

```
import serial
#arduinoData = serial.Serial('/dev/ttyACM1', 9600)
arduinoData = serial.Serial('COM6', 115200, timeout=0)


while True:
    myData = input('Kirimkan perintah (1/0): ')
    if myData == "1":
        arduinoData.write(b'1')
    elif myData == "0":
        arduinoData.write(b'0')
```

Tindakan: Menjalankan skrip arduino.py lalu mengirimkan teks '1' dan '0' melalui port serial (COM6).



Hasil: ESP32 menerima sinyal 1 atau 0 melalui jalur serial dan mengaktifkan fungsi MotorOn() dengan duty cycle tetap.



Off (0) On (1)

Tahap 2: Implementasi On/Off via MQTT

Kode Program:

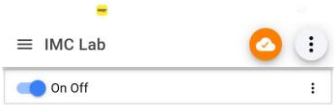
```
#include <WiFi.h>
#include <PubSubClient.h>
#include <Arduino.h>

int ledPin = 2;
char myData = 0;

int motor1Pin1 = 27;
int motor1Pin2 = 26;
int enable1Pin = 12;

// Setting PWM properties
const int freq = 30000;
const int pwmChannel = 0;
const int resolution = 8;
int dutyCycle = 200;

const byte pin_rpm = 13;
int volatile rev = 0;
//int rpm = 0;

const char* ssid = "DANERA 5G"; // Enter your WiFi name
const char* password =  "danera1234"; // Enter WiFi password

#define mqttServer "broker.emqx.io"
#define mqttPort 1883

WiFiServer server(80);
WiFiClient espClient;
PubSubClient client(espClient);

String Topic;
String Payload;
```

```
// constants
const int baud = 115200;       // serial baud rate


void setup()
{
  pinMode(ledPin, OUTPUT);
  Serial.begin(115200);
  // sets the pins
  pinMode(motor1Pin1, OUTPUT);
  pinMode(motor1Pin2, OUTPUT);
  pinMode(enable1Pin, OUTPUT);
  pinMode(pin_rpm, INPUT_PULLUP);
  //  pinMode(pin_rpm, INPUT);
  //attachInterrupt(digitalPinToInterrupt(pin_rpm), isr, RISING);

  // configure LED PWM functionalitites
  ledcSetup(pwmChannel, freq, resolution);

  // attach the channel to the GPIO to be controlled
  ledcAttachPin(enable1Pin, pwmChannel);

  // testing
  Serial.print("Testing DC Motor...");

  // Connect to WiFi network
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
```

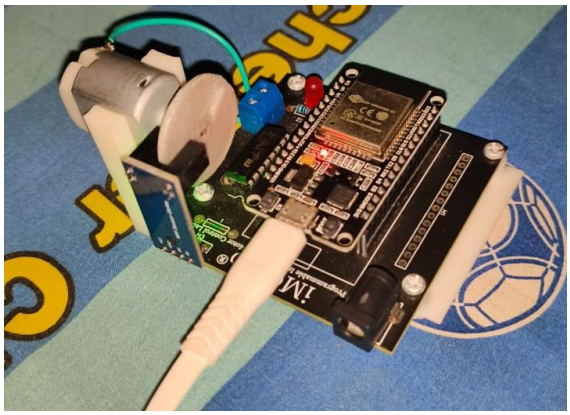
```
  // Connect to Server IoT (CloudMQTT)
  client.setServer(mqttServer, mqttPort);
  client.setCallback(receivedCallback);

  while (!client.connected()) {
    Serial.println("Connecting to CLoud IoT ...");

//    if (client.connect("ESP32Client", mqttUser, mqttPassword )) {
     if (client.connect("iMCLab On/Off")) {

      Serial.println("connected");
      Serial.print("Message received: ");

    } else {
      Serial.print("failed with state ");
      Serial.print(client.state());
      delay(2000);
    }
    client.subscribe("ImcRizky");
  }
}

void MotorOn()
{
  // Move DC motor forward with increasing speed
  dutyCycle = 205;
  digitalWrite(motor1Pin1, HIGH);
  digitalWrite(motor1Pin2, LOW);
  ledcWrite(pwmChannel, dutyCycle);
}

void MotorOff()
{
  dutyCycle = 0;
  digitalWrite(motor1Pin1, HIGH);
  digitalWrite(motor1Pin2, LOW);
  ledcWrite(pwmChannel, dutyCycle);
```

```
}

void receivedCallback(char* topic, byte* payload, unsigned int length) {

/* we got '1' -> MotorOn() */
  if ((char)payload[0] == '1') {
        MotorOn();
        Serial.println("Motor On");
  }

/* we got '0' -> Motoroff */
  if ((char)payload[0] == '0') {
        MotorOff();
        Serial.println("Motor Off");
  }
}

void loop()
{

client.loop();

myData = int(Serial.read());

if (myData == '1'){
  Serial.println("LED is on !!!");
  digitalWrite(ledPin, HIGH);
  MotorOn();
}
else if (myData == '0'){
  Serial.println("LED is off !!!");
  digitalWrite(ledPin, LOW);
  MotorOff();
}

}
```

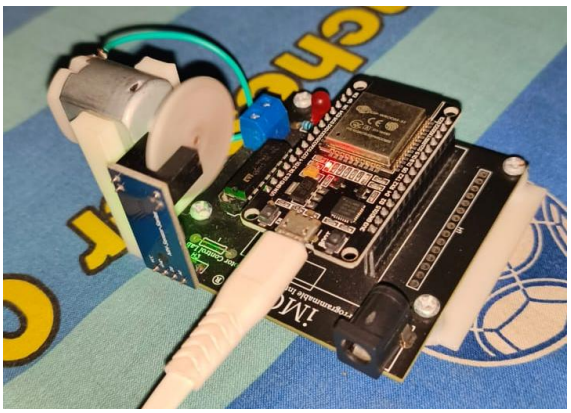
Tindakan: Menggunakan broker broker.emqx.io untuk mengirim pesan ke topik ImcRizky.



Hasil: Motor dapat dikontrol secara On Off melaui remote menggunakan aplikasi Iot MQTT Panel. Hal ini membuktikan bahwa pustaka PubSubClient berhasil mengelola koneksi WiFi dan MQTT secara simultan tanpa mengganggu putaran motor.



Off (0)



On (1)

Tahap 3: Perancangan Algoritma PID

Kode Program:

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <Arduino.h>

// --- KONFIGURASI WIFI & MQTT ---
const char* ssid = "DANERA 5G";
const char* password = "danera1234";
#define mqttServer "broker.emqx.io"
#define mqttPort 1883
#define mqttClientName "ESP32_PID_Rizky_UAS"

// --- PIN DEFINITIONS ---
const int ledPin = 2;
const int motor1Pin1 = 27;
const int motor1Pin2 = 26;
const int enable1Pin = 12; // PWM Pin
const int pin_rpm = 13;    // Encoder Input

// --- KONFIGURASI MOTOR & ENCODER ---
// PENTING: Ubah ini sesuai jumlah lubang piringan encoder anda!
const float PULSES_PER_REV = 20.0;

// PWM Properties
const int freq = 30000;
const int pwmChannel = 0;
const int resolution = 8; // 0-255

// --- VARIABEL PID ---
double Kp = 1.5;  // Default Kp
double Ki = 0.5;  // Default Ki
double Kd = 0.05; // Default Kd

double target_rpm = 0; // Target kecepatan
double current_rpm = 0;
```

```
double error = 0, last_error = 0, integral = 0, derivative = 0;
double output_pwm = 0;

// Variabel Timing & Encoder
volatile long pulse_count = 0;
unsigned long last_time = 0;
unsigned long sample_time = 100; // Hitung PID setiap 100ms
bool motor_active = false;

// Objek WiFi & MQTT
WiFiClient espClient;
PubSubClient client(espClient);

// Interrupt Service Routine (Agar pembacaan sensor cepat)
void IRAM_ATTR isr() {
  pulse_count++;
}

void setup() {
  Serial.begin(115200);

  pinMode(ledPin, OUTPUT);
  pinMode(motor1Pin1, OUTPUT);
  pinMode(motor1Pin2, OUTPUT);
  pinMode(enable1Pin, OUTPUT);
  pinMode(pin_rpm, INPUT_PULLUP);

  // Setup Interrupt
  attachInterrupt(digitalPinToInterrupt(pin_rpm), isr, RISING);

  // Setup PWM Motor
  ledcSetup(pwmChannel, freq, resolution);
  ledcAttachPin(enable1Pin, pwmChannel);

  // Koneksi WiFi
  setupWifi();
```

```
  // Koneksi MQTT
  client.setServer(mqttServer, mqttPort);
  client.setCallback(receivedCallback);
}

void setupWifi() {
  delay(10);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected");
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect(mqttClientName)) {
      Serial.println("connected");
      // Subscribe ke topik-topik kontrol
      client.subscribe("ImcRizky/command");   // '1' Start, '0' Stop
      client.subscribe("ImcRizky/setpoint");  // Input Target RPM
      client.subscribe("ImcRizky/kp");        // Input nilai Kp
      client.subscribe("ImcRizky/ki");        // Input nilai Ki
      client.subscribe("ImcRizky/kd");        // Input nilai Kd
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      delay(2000);
    }
  }
}
```

```
void receivedCallback(char* topic, byte* payload, unsigned int length) {
  // Konversi payload ke String supaya mudah dibaca
  String message = "";
  for (int i = 0; i < length; i++) {
    message += (char)payload[i];
  }

  Serial.print("Message ["); Serial.print(topic); Serial.print("] "); Serial.println(message);

  // --- LOGIKA PARSING PESAN ---
  if (String(topic) == "ImcRizky/command") {
    if (message == "1") {
      motor_active = true;
      Serial.println("SYSTEM START");
    } else if (message == "0") {
      motor_active = false;
      target_rpm = 0;
      stopMotor();
      integral = 0; // Reset memori PID
      Serial.println("SYSTEM STOP");
    }
  }
  else if (String(topic) == "ImcRizky/setpoint") {
    target_rpm = message.toFloat();
  }
  // --- LIVE TUNING PID ---
  else if (String(topic) == "ImcRizky/kp") {
    Kp = message.toFloat();
  }
  else if (String(topic) == "ImcRizky/ki") {
    Ki = message.toFloat();
  }
  else if (String(topic) == "ImcRizky/kd") {
    Kd = message.toFloat();
  }
}
```

```
void driveMotor(int pwmVal) {
  // Batasi PWM 0-255 (karena resolusi 8 bit)
  if (pwmVal > 255) pwmVal = 255;
  if (pwmVal < 0) pwmVal = 0;

  digitalWrite(motor1Pin1, HIGH);
  digitalWrite(motor1Pin2, LOW);
  ledcWrite(pwmChannel, pwmVal);
}

void stopMotor() {
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, LOW);
  ledcWrite(pwmChannel, 0);
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  // --- LOOP PID SETIAP X MILIDETIK ---
  unsigned long now = millis();
  if (now - last_time >= sample_time) {

    // 1. HITUNG RPM (Aman dari gangguan interrupt)
    noInterrupts(); // Stop interrupt sebentar
    long pulses = pulse_count;
    pulse_count = 0;
    interrupts();   // Nyalakan lagi

    // Rumus: (Pulsa / PPR) * (60000ms / delta_waktu)
    current_rpm = (pulses / PULSES_PER_REV) * (60000.0 / (now - last_time));
```

```
// 2. LOGIKA PID
if (motor_active) {
  error = target_rpm - current_rpm;

  // Integral (Penjumlahan error seiring waktu)
  integral += error * (sample_time / 1000.0);

  // Batasi Integral (Anti-Windup) supaya tidak 'menggila'
  if(integral > 1000) integral = 1000;
  if(integral < -1000) integral = -1000;

  // Derivative (Kecepatan perubahan error)
  derivative = (error - last_error) / (sample_time / 1000.0);

  // Rumus Utama PID
  double pid_out = (Kp * error) + (Ki * integral) + (Kd * derivative);

  // Simpan last error
  last_error = error;

  // Eksekusi ke Motor
  driveMotor((int)pid_out);
  output_pwm = pid_out;

  digitalWrite(ledPin, HIGH); // LED indikator nyala
} else {
  stopMotor();
  digitalWrite(ledPin, LOW);
}

// 3. KIRIM DATA KE HP (Untuk Grafik)
// Kirim hanya jika ada perubahan signifikan atau setiap loop
client.publish("ImcRizky/graph/rpm", String(current_rpm).c_str());
client.publish("ImcRizky/graph/target", String(target_rpm).c_str());

// Debugging di Serial Monitor Arduino
```

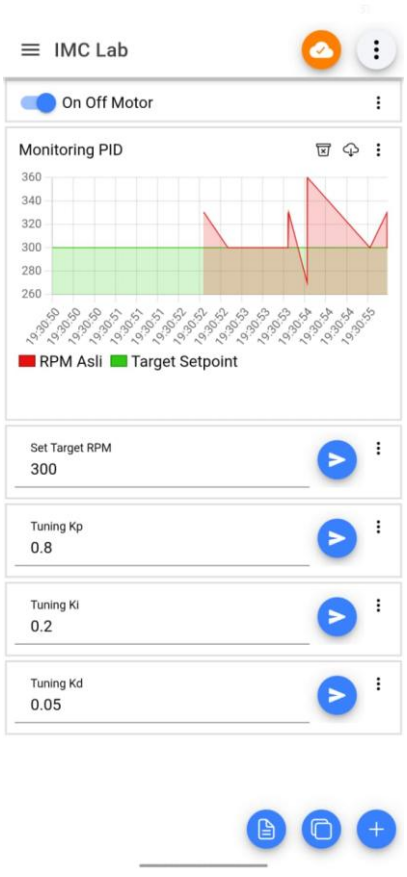
```
    Serial.print("Tgt:"); Serial.print(target_rpm);
    Serial.print(" RPM:"); Serial.print(current_rpm);
    Serial.print(" PWM:"); Serial.print(output_pwm);
    Serial.print(" Kp:"); Serial.print(Kp);
    Serial.print(" Ki:"); Serial.println(Ki);

    last_time = now;
  }
}
```

Tindakan: Memasukkan rumus PID ke dalam loop utama dengan sampling time 100ms. Nilai awal yang ditetapkan adalah Kp=1.5, Ki=0.5, Kd=0.05.

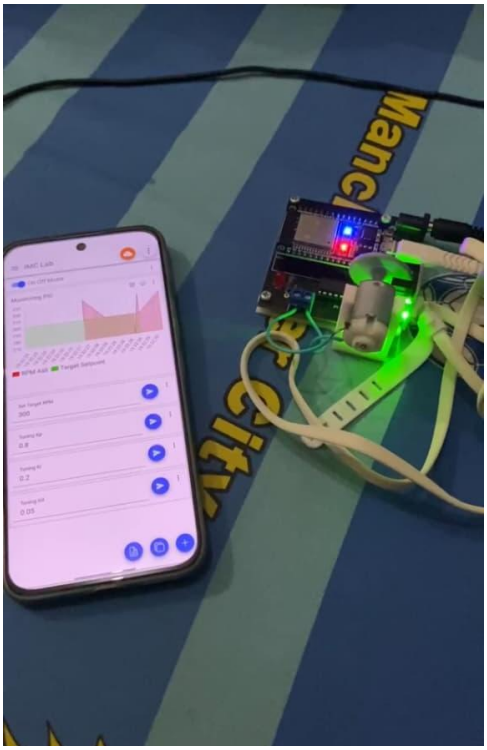
Hasil: Sistem mampu menghitung error antara target_rpm dan current_rpm, kemudian menyesuaikan output PWM untuk mengejar target tersebut.

Tahap 4: Live Tuning dan Monitoring Grafik

Tindakan: Mengirim nilai Target RPM, Kp, Ki, dan Kd baru melalui Aplikasi IoT MQTT Panel.



Hasil: Kecepatan motor berubah secara dinamis sesuai setpoint. Data RPM aktual dipublikasikan ke topik ImcRizky/graph/rpm sehingga dapat dipantau dalam bentuk grafik real-time melalui dashboard Aplikasi IoT MQTT Panel.



BAB IV
PENUTUP

4.1 Kesimpulan

Berdasarkan hasil praktikum yang telah dilaksanakan, dapat disimpulkan bahwa sistem kendali motor DC berbasis ESP32 telah berhasil mengintegrasikan tiga jalur kendali utama secara harmonis, yaitu kontrol lokal melalui jalur serial Python, kontrol jarak jauh menggunakan protokol MQTT, dan kontrol otomatis presisi dengan algoritma PID. Penerapan algoritma PID terbukti memberikan keunggulan signifikan dibandingkan metode kontrol On/Off konvensional karena kemampuannya dalam menjaga kestabilan kecepatan aktual terhadap target (*setpoint*) secara dinamis, bahkan saat menghadapi gangguan beban fisik yang mendadak. Selain itu, penggunaan sistem *interrupt* pada sensor encoder sangat krusial untuk menghasilkan pembacaan RPM yang akurat, sementara transisi ke sistem kendali cerdas ini tidak hanya meningkatkan akurasi operasional tetapi juga mampu mengoptimalkan efisiensi daya serta memperpanjang usia pakai komponen motor DC.

4.2 Saran

Disarankan untuk menambahkan sistem pengamanan pada dashboard MQTT (seperti penggunaan password atau sertifikat SSL) dan menggunakan motor driver yang mendukung kapasitas arus lebih tinggi untuk pengujian beban berat di masa mendatang guna menjaga reliabilitas hardware.

DAFTAR PUSTAKA

[1] Espressif Systems. (2024). *ESP32 Technical Reference Manual*.

[2] MQTT.org. *MQTT Protocol Specification*.

[3] Rahmat, B. (2022). "Implementation of PWM for DC Motor Control in ESP32-based Systems". *Journal of Embedded Systems and Robotics*.

[4] Hunkeler, S., Truong, H. L., & Stanford-Clark, A. (2008). "MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks". *Third International Conference on Communication Systems Software and Middleware*.

[5] Ogata, K. (2010). *Modern Control Engineering* (5th ed.). Pearson.

[6] Liechti, C. (2020). *pySerial Documentation*.

[7] EMQX Broker Documentation. (2024). *MQTT Essentials for IoT Developers*.