Nama : Rizky Surya Pratama

NIM : 224308045

Kelas : TKA-6B

KONTROL PID

**PROGRAM TRANSFER FUCTION :**

% Parameter Motor

R = 0.4;

L = 2.7;

J = 0.0004;

B = 0.0022;

Ke = 0.015;

Kt = 0.05;

% Numerator dan Denominator

num = Ke;

den = [L\*J (L\*B + R\*J) (R\*B + Ke\*Kt)];

% Transfer Function (Open-loop)

Gs = tf(num, den);

Nilai transfer fuction :

A number and a line

AI-generated content may be incorrect.

Langkah awal sebelum menentukan nilai PID, yang harus dilakukan menentukan nilai transfer fuction dari sistem, dengan menentukan nilai dari parameter motor yang digunakan. Parameter yang digunakan meliputi resistansi (R = 0.4 Ω), induktansi (L = 2.7 H), momen inersia (J = 0.0004 kg·m²), koefisien gesekan viskus (B = 0.0022 N·m·s), konstanta tegangan balik (Ke = 0.015 V·s/rad), dan konstanta torsi (Kt = 0.05 N·m/A). Selanjutnya, memasukan parameter ke rumus fungsi matematika transfer fuction. Fungsi transfer fuction menggambarkan hubungan antara tegangan masukan terhadap kecepatan sudut motor sebagai keluaran.

**PARAMETER KONTROL PID**

% PID parameters

kp = 53.7208;

Ti = 0.73245;

Td = 0.17711;

ki = kp / Ti;

kd = kp \* Td;

% Buat controller PID

PID = pid(kp, ki, kd);

% Closed-loop transfer function (Negative feedback, unity)

T = feedback(PID \* Gs, 1);

% Step response

step(T)

title('Step Response with PID Controller')

grid on

% Hitung karakteristik step response

info = stepinfo(T)

% Tampilkan hasil karakteristik

fprintf('\nKarakteristik Step Response:\n');

fprintf('Rise Time : %.4f s\n', info.RiseTime);

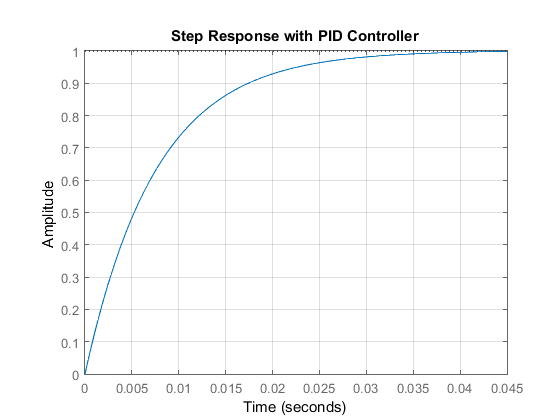
fprintf('Peak Time : %.4f s\n', info.PeakTime);

fprintf('Overshoot : %.2f %%\n', info.Overshoot);

fprintf('Settling Time : %.4f s\n', info.SettlingTime);

Setelah fungsi transfer fuction diperoleh, langkah selanjutnya adalah penentuan parameter PID yang terdiri dari nilai proporsional (**Kp**), integral (**Ti**), dan derivatif (**Td**). Nilai-nilai yang digunakan adalah Kp = 53.7208, Ti = 0.73245, dan Td = 0.17711. Dari parameter tersebut, nilai **Ki** dan **Kd** dihitung berdasarkan hubungan berikut. Nilai Ki diperoleh dari rumus Ki=Kp/Ti, di mana Ti adalah konstanta waktu integral yang mengatur kecepatan aksi integral dalam menghilangkan kesalahan steady-state. Selanjutnya nilai Kd dihitung dengan rumus Kd=Kp×Td, di mana Td adalah konstanta waktu derivatif yang berfungsi memprediksi perubahan error sehingga dapat mengurangi overshoot dan mempercepat respons sistem. Sehingga diperoleh Ki ≈ 73.3402 dan Kd ≈ 9.5142. Setelah itu, kontroler PID dibentuk dengan menggunakan fungsi pid(kp, ki, kd). Dari hasil simulasi step respon yang ditampilkan, diketahui bahwa sistem memiliki performa sangat baik dengan rise time 0.0166 detik, settling time 0.0294 detik, dan overshoot hanya sebesar 0.10%, menandakan bahwa sistem mampu merespons dengan cepat, akurat dan stabil.

**Hasil Run :**



|  |  |
| --- | --- |
| info =  RiseTime: 0.0166  SettlingTime: 0.0294  SettlingMin: 0.9002  SettlingMax: 1.0010  Overshoot: 0.1002  Undershoot: 0  Peak: 1.0010  PeakTime: 0.0554 | Karakteristik Step Response:  Rise Time : 0.0166 s  Peak Time : 0.0554 s  Overshoot : 0.10 %  Settling Time : 0.0294 s |

**SIMULINK :**

A diagram of a circuit

AI-generated content may be incorrect.

**HASIL SIMULINK :**

A screenshot of a graph

AI-generated content may be incorrect.

Kemudian, untuk Model Simulink menggambarkan alur kerja sistem kendali PID untuk mengatur kecepatan motor secara tertutup (closed-loop). Sistem ini menggunakan blok *constant* sebagai input referensi sebesar 40 RPM, yang kemudian dibandingkan dengan keluaran sistem melalui blok pengurang untuk menghasilkan nilai error. Error ini selanjutnya diproses melalui kontroler PID yang dibangun secara manual menggunakan kombinasi penguat proporsional, integral, dan derivatif. Jalur proporsional langsung mengalikan error dengan konstanta Kp melalui blok gain, sementara jalur integral memproses error melalui Integrator (1/s) lalu dikalikan dengan Ki, dan jalur derivatif menghitung perubahan error menggunakan blok Derivative (du/dt) kemudian dikalikan dengan Kd. Ketiga sinyal hasil pengolahan PID dijumlahkan untuk membentuk sinyal kendali yang masuk ke blok transfer fuction. Output dari fungsi transfer fuction berupa kecepatan yang dikonversi dari satuan radian per detik menjadi RPM, dan dikembalikan ke jalur umpan balik agar sistem bekerja secara tertutup. Seluruh proses ini divisualisasikan melalui blok Scope untuk mengamati respons sistem terhadap perubahan input. Hasil simulasi yang ditampilkan pada Scope menunjukkan bahwa sistem merespons input referensi 40 RPM dengan sangat baik. Respons sistem diawali dari nol, kemudian meningkat cepat menuju nilai referensi. Sistem juga menunjukkan waktu naik dan waktu tunak yang sangat singkat, serta tidak mengalami kesalahan pada kondisi tunak (steady-state error). Hal ini menandakan bahwa parameter PID yang digunakan telah disetel dengan tepat dan menghasilkan performa sistem yang akurat dan stabil.

**PROGRAM ARDUINO IDE :**

#include <WiFi.h>

#include <PubSubClient.h>

// ===========================

// KONFIGURASI WIFI & MQTT

// ===========================

const char\* ssid = "TKA 6Barokah";

const char\* password = "65432100";

const char\* mqtt\_server = "broker.hivemq.com";

const int mqtt\_port = 1883;

WiFiClient espClient;

PubSubClient client(espClient);

// ===========================

// PIN KONFIGURASI

// ===========================

const int pinMotorPWM = 5;

const int pinREn = 21;

const int pinLEn = 22;

const int encoderPinA = 34;

const int encoderPinB = 35;

// ===========================

// VARIABEL PID & MOTOR

// ===========================

kp = 53.7208;

Ti = 0.73245;

Td = 0.17711;

ki = kp / Ti;

kd = kp \* Td;

float error = 0, lastError = 0, integral = 0;

float targetSpeedKmh = 0;

float actualSpeedKmh = 0;

int pidPWM = 0;

int throttleLevel = 0;

String motorDirection = "forward"; // Default arah

// ===========================

// ENCODER

// ===========================

volatile long encoderCount = 0;

unsigned long lastSpeedCheck = 0;

void IRAM\_ATTR encoderISR() {

encoderCount++;

}

// ===========================

// WIFI DAN MQTT SETUP

// ===========================

void setup\_wifi() {

Serial.print("Connecting to WiFi ");

Serial.print(ssid);

WiFi.mode(WIFI\_STA);

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println();

Serial.println("WiFi connected, IP: " + WiFi.localIP().toString());

}

void reconnect() {

while (!client.connected()) {

Serial.print("Connecting to MQTT...");

String clientId = "ESP32Client-";

clientId += String(random(0xffff), HEX);

if (client.connect(clientId.c\_str())) { // Connect tanpa user/password

Serial.println("connected!");

client.subscribe("motor/throttle");

client.subscribe("motor/direction");

} else {

Serial.print("failed, rc=");

Serial.print(client.state());

Serial.println(" try again in 5 seconds");

delay(5000);

}

}

}

// ===========================

// CALLBACK MQTT

// ===========================

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

String msg;

for (unsigned int i = 0; i < length; i++) {

msg += (char)payload[i];

}

Serial.print("[MQTT] Topic: ");

Serial.print(topic);

Serial.print(" | Message: ");

Serial.println(msg);

if (String(topic) == "motor/throttle") {

int throttle = msg.toInt();

throttleLevel = constrain(throttle, 0, 3);

switch (throttleLevel) {

case 1: targetSpeedKmh = 5; break;

case 2: targetSpeedKmh = 10; break;

case 3: targetSpeedKmh = 15; break;

}

Serial.println("[MQTT] Throttle level set: " + String(throttleLevel));

}

if (String(topic) == "motor/direction") {

msg.toLowerCase();

if (msg == "forward" || msg == "reverse" || msg == "neutral") {

motorDirection = msg;

Serial.println("[MQTT] Direction set: " + motorDirection);

}

}

}

// ===========================

// SETUP

// ===========================

void setup() {

Serial.begin(115200);

pinMode(pinREn, OUTPUT);

pinMode(pinLEn, OUTPUT);

ledcSetup(0, 5000, 8); // 5kHz, 8-bit PWM

ledcAttachPin(pinMotorPWM, 0);

pinMode(encoderPinA, INPUT\_PULLUP);

pinMode(encoderPinB, INPUT\_PULLUP);

attachInterrupt(digitalPinToInterrupt(encoderPinA), encoderISR, RISING);

setup\_wifi();

client.setServer(mqtt\_server, mqtt\_port);

client.setCallback(callback);

}

// ===========================

// MENGHITUNG KECEPATAN

// ===========================

void updateSpeed() {

noInterrupts();

long count = encoderCount;

encoderCount = 0;

interrupts();

float rpm = (count / 20.0) \* 60.0; // 20 pulses per rotation

float wheelCircumference = 3.14 \* 0.1; // Diameter roda 10 cm

actualSpeedKmh = (rpm \* wheelCircumference \* 60.0) / 1000.0;

Serial.println("[ESP32] Actual Speed: " + String(actualSpeedKmh, 2) + " km/h");

client.publish("motor/actual\_speed", String(actualSpeedKmh, 2).c\_str());

}

// ===========================

// KONTROL ARAH

// ===========================

void controlDirection() {

if (motorDirection == "forward") {

digitalWrite(pinREn, HIGH);

digitalWrite(pinLEn, LOW);

} else if (motorDirection == "reverse") {

digitalWrite(pinREn, LOW);

digitalWrite(pinLEn, HIGH);

} else { // neutral

digitalWrite(pinREn, LOW);

digitalWrite(pinLEn, LOW);

}

}

// ===========================

// PID CONTROL

// ===========================

void PIDControl() {

if (motorDirection == "neutral" || targetSpeedKmh == 0) {

ledcWrite(0, 0);

pidPWM = 0;

integral = 0;

lastError = 0;

return;

}

error = targetSpeedKmh - actualSpeedKmh;

integral += error;

float derivative = error - lastError;

lastError = error;

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

pidPWM = constrain((int)output, 0, 255);

ledcWrite(0, pidPWM);

Serial.println("[ESP32] PID PWM: " + String(pidPWM));

client.publish("motor/pid\_output", String(pidPWM).c\_str());

}

// ===========================

// LOOP UTAMA

// ===========================

void loop() {

if (!client.connected()) {

reconnect();

}

client.loop();

controlDirection();

unsigned long now = millis();

if (now - lastSpeedCheck >= 1000) {

lastSpeedCheck = now;

updateSpeed();

PIDControl();

}

}

Selanjutnya dibuat program untuk realisasi dari sistem yang mengatur kecepatan dan arah motor dengan menggunakan ESP32, yang terhubung melalui WiFi dan mengandalkan komunikasi MQTT. Sistem ini mengimplementasikan pustaka WiFi.h untuk menghubungkan jaringan dan PubSubClient.h untuk berkomunikasi dengan broker HiveMQ, yang berjalan pada port 1883 tanpa memerlukan autentikasi. Pin-pinnya termasuk pin PWM yang bertanggung jawab untuk mengatur kecepatan motor, serta pin untuk kontrol arah motor kanan dan kiri. Encoder incremental yang terpasang pada pin 34 dan 35 berfungsi untuk menghitung jumlah putaran motor agar dapat memberikan kecepatan sebenarnya.  
  
Pengaturan kecepatan motor menggunakan metode kontrol PID. Berbagai parameter kontrol seperti konstanta proporsional (Kp), integral (Ki), dan derivatif (Kd) ditentukan berdasarkan nilai Kp, waktu integral (Ti), dan waktu derivatif (Td). Nilai-nilai ini dipakai dalam penghitungan selisih kecepatan (error), integral error, dan perubahan error untuk menghasilkan sinyal PWM yang sesuai. Namun, dalam kode ini terdapat kesalahan karena tipe data untuk variabel PID seperti kp, ki, dan kd tidak didefinisikan secara jelas sebagai float.  
  
Sistem ini juga mengandalkan sistem WiFi yang diatur dalam fungsi setup\_wifi(), di mana ESP32 akan terhubung ke jaringan lokal dengan SSID dan kata sandi yang sudah ditentukan. Setelah terhubung, ESP32 akan berusaha untuk menyambungkan diri ke broker MQTT dan berlangganan pada dua topik utama: motor/throttle untuk menerima instruksi perubahan tingkat kecepatan dan motor/direction untuk mendapatkan instruksi perubahan arah motor (maju, mundur, atau netral). Fungsi callback() bertanggung jawab untuk memproses pesan yang diterima dari broker dan mengonversi nilai throttle menjadi kecepatan target dalam km/jam, serta menyimpan arah motor berdasarkan informasi yang diterima.  
  
Interupsi juga diterapkan dalam kode ini menggunakan fungsi encoderISR() untuk menghitung pulsa dari encoder setiap kali ada perubahan sinyal, yang memungkinkan sistem untuk menghitung kecepatan motor secara real-time. Namun, kode ini masih belum lengkap karena belum mencakup fungsi setup() dan loop() yang seharusnya menangani inisialisasi pin, aktivasi interupsi, dan pengaturan motor secara terus-menerus menggunakan logika PID. Agar dapat berfungsi secara menyeluruh, program perlu melengkapi bagian loop() yang mengelola kontrol PID dan mengendalikan sinyal PWM berdasarkan kesalahan kecepatan yang dihitung, serta mempublikasikan data sensor jika diperlukan.