Nama: Moh. Farhan Baihaqi

NIM: 224308037

Kelas TKA 6-B

Kontrol PID

**Program transfer fuction :**

Sebelum menentukan nilai PID harus menentukan nilai transfer fuction terlebih dahulu, dengan menentukan nilai dari parameter motor yang digunakan. Nilai parameter motor yang dimasukkan adalah resistansi armature(R),induktansi armature(L), momen inersia rotor(J), koefisien redaman viskos(B), konstanta gaya(Ke), konstanta torsi(Kt). Selanjutnya, dengan memasukkan rumus matematika 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.

**Parameter kontrol PID**

Dalam proses tuning PID di MATLAB, Langkah awal dimulai dengan memodelkan sistem yang akan dikendalikan. Selanjtnya dapat membuka aplikasi PID Tuner melalui menu Apps di MATLAB, perintah pidTuner di command window, atau langsung dari blok PID Controller di Simulink dengan menekan tombol "Tune". PID Tuner kemudian secara otomatis menghitung nilai optimal Kp, Ki, dan Kd berdasarkan nilai transfer fuction. 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. Setelah parameter diperoleh sudah sesuai nilai Kp, Ki, dan Kd dimasukkan ke dalam program.

% 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');

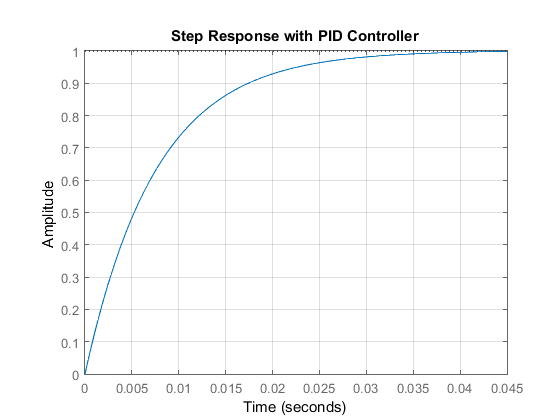
fprintf('Rise Time : %.4f s\n', info.RiseTime);

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

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

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

**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 :**

Membuat diagram blok control PID dimulai dengan pemberian nilai setpoint ke dalam Simulink. Kontrol PID akan menghitung nilai Kp,Ki,Kd. Kemudian muncul grafik PID dengan membuka scope.

A diagram of a circuit

AI-generated content may be incorrect.

**Hasil Simulink :**

A screenshot of a graph

AI-generated content may be incorrect.

**Program ARDUINO IDE :**

Program PID dalam program ini digunakan sebagai pengatur kecepatan motor agar sesuai dengan target yang diinginkan. PID (Proportional-Integral-Derivative) mengontrol sinyal PWM yang mengatur daya ke motor berdasarkan selisih antara kecepatan target (targetSpeedKmh) dan kecepatan aktual (actualSpeedKmh) yang diukur dari sensor encoder. Komponen proporsional (Kp) memberikan respons langsung terhadap error saat ini, komponen integral (Ki) mengakumulasi error dari waktu ke waktu untuk menghilangkan offset, dan komponen derivatif (Kd) memperhitungkan perubahan error untuk mengurangi overshoot. Jika motor dalam mode netral atau target kecepatan nol, output PWM dimatikan. Dengan pendekatan ini, motor dapat mempertahankan kecepatan yang stabil dan sesuai dengan perintah throttle yang diterima melalui MQTT.

#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();

}

}