Tugas Final Project Mikrokontroller

Mata Kuliah: Mikrokontroller A081

Dosen Pengampu:

Prof. Dr. Basuki Rahmat, S.Si, MT,

Disusun oleh :

Nathanael Kristian Sujarwo (23081010271)

Program Studi Informatika

Fakultas Ilmu Komputer

Universitas Pembangunan Nasional UPN "Veteran" Jawa Timur

2025

BAB I
Pendahuluan

1.1 Latar Belakang

Perkembangan teknologi mikrokontroler saat ini telah mengalami pergeseran paradigma dari sistem yang bersifat lokal menuju sistem yang terkoneksi secara global melalui Internet of Things (IoT). Konsep IoT memungkinkan perangkat embedded untuk tidak hanya mengeksekusi fungsi kontrol secara mandiri, tetapi juga dapat menerima perintah dari jarak jauh dan mengirimkan data monitoring ke server atau client melalui protokol komunikasi yang efisien seperti MQTT. Dalam konteks pembelajaran mikrokontroler tingkat lanjut, integrasi antara hardware control (PWM, digital I/O) dengan software communication (WiFi, MQTT) menjadi kompetensi penting yang perlu dikuasai mahasiswa teknik elektro dan komputer.

Pada project ini, ESP32 dipilih sebagai platform mikrokontroler karena memiliki kemampuan WiFi built-in dan dukungan library yang matang untuk komunikasi MQTT, sehingga cocok untuk implementasi sistem IoT sederhana hingga menengah. Motor DC digunakan sebagai aktuator yang akan dikontrol menggunakan driver H-bridge, di mana pengaturan arah motor dilakukan melalui pin IN1 dan IN2, sedangkan pengaturan kecepatan dilakukan melalui sinyal PWM pada pin ENA dengan resolusi 8-bit (nilai 0–255). Broker MQTT publik broker.hivemq.com pada port 1883 dipilih sebagai media komunikasi untuk menghubungkan ESP32 dengan program client Python, menggunakan tiga topik utama: imclabled untuk kontrol LED indikator, imclabmotorsetpoint untuk pengiriman setpoint motor, dan imclabmotorspeed untuk feedback kecepatan motor.

Selain aspek kontrol, project ini juga menekankan pentingnya data acquisition dan analisis perilaku sistem berdasarkan data eksperimen. Dengan menjalankan skenario step test (perubahan setpoint bertahap) dan merekam respons sistem dalam bentuk CSV yang berisi kolom time, setpoint, speed, dan error, mahasiswa dapat mempelajari bagaimana sistem merespons perubahan input dan mengidentifikasi karakteristik dinamika sistem. Data yang terkumpul kemudian dimanfaatkan untuk membangun model prediksi menggunakan LSTM (Long Short-Term Memory), sebuah arsitektur neural network yang dirancang khusus untuk menangani data sequential/time series, sehingga project ini menggabungkan tiga domain sekaligus: embedded systems, IoT communication, dan machine learning.

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, rumusan masalah pada proyek ini adalah:

1. Bagaimana merancang dan mengimplementasikan firmware ESP32 yang dapat terkoneksi ke jaringan WiFi dan broker MQTT, serta mampu melakukan subscribe pada topik perintah untuk menerima setpoint motor dan kontrol LED?

2. Bagaimana mengimplementasikan logika kontrol motor DC menggunakan driver H-bridge dengan pengaturan arah forward/stop dan pengaturan kecepatan melalui PWM berdasarkan nilai setpoint 0–255?
3. Bagaimana merancang mekanisme monitoring dengan cara melakukan publish feedback kecepatan motor secara periodik ke topik MQTT agar dapat dimonitor dan dicatat oleh program client?
4. Bagaimana melakukan eksperimen step test menggunakan program Python MQTT client, merekam data respons sistem, dan menyimpannya dalam format CSV yang siap dianalisis?
5. Bagaimana membangun dan melatih model LSTM untuk memprediksi kecepatan motor berdasarkan sequence setpoint, serta mengevaluasi performa model menggunakan metrik yang sesuai?

1.3 Tujuan

Tujuan dari proyek ini meliputi:

1. Mengimplementasikan firmware ESP32 yang mampu terkoneksi WiFi, terkoneksi broker MQTT, melakukan subscribe pada topik imclabled dan imclabmotorsetpoint, serta memproses pesan yang masuk melalui callback function.
2. Mengimplementasikan kontrol motor DC menggunakan driver H-bridge dengan logika forward (IN1 HIGH, IN2 LOW) ketika setpoint > 0 dan stop (kedua pin LOW) ketika setpoint = 0, serta pengaturan PWM sesuai nilai setpoint.
3. Mengimplementasikan mekanisme publish feedback ke topik imclabmotorspeed secara berkala (setiap ~500 ms) untuk keperluan monitoring dan logging data eksperimen.
4. Membuat program Python MQTT client yang dapat mengirim setpoint bertahap (step test), menerima feedback dari topik speed, lalu menyimpan data time/setpoint/speed/error ke file motorresponsedata.csv.
5. Membangun arsitektur LSTM dengan sequence timesteps = 10, melatih model menggunakan dataset hasil eksperimen, dan mengevaluasi performa menggunakan metrik MAE, RMSE, dan R^2.

1.4 Manfaat

Manfaat dari project ini adalah memberikan pengalaman praktik yang komprehensif dalam mengintegrasikan tiga domain teknologi: embedded systems (ESP32 + motor control), IoT communication (WiFi + MQTT publish/subscribe), dan data science (data logging + LSTM modeling). Dari sisi hardware, mahasiswa belajar mengatur pin digital untuk kontrol arah motor, membuat sinyal PWM untuk kontrol kecepatan, serta memahami bagaimana driver H-bridge bekerja sebagai interface antara mikrokontroler dan motor yang membutuhkan arus tinggi. Dari sisi software, mahasiswa memahami bagaimana protokol MQTT bekerja dengan pola publish/subscribe, bagaimana menangani event-driven programming melalui callback, serta bagaimana melakukan reconnect otomatis ketika koneksi terputus.

BAB II

Landasan Teori

2.1 Mikrokontroler ESP32

ESP32 adalah mikrokontroler 32-bit dual-core yang dikembangkan oleh Espressif Systems dan banyak digunakan dalam aplikasi IoT karena memiliki WiFi dan Bluetooth built-in, sehingga tidak memerlukan modul komunikasi eksternal tambahan. Pada project ini, ESP32 berperan sebagai node kontrol yang menerima perintah setpoint motor dari broker MQTT, mengeksekusi kontrol ke aktuator melalui pin GPIO, dan mengirimkan feedback monitoring secara periodik ke topik yang telah ditentukan. ESP32 mendukung pengaturan PWM melalui peripheral LEDC (LED Control) yang dapat dikonfigurasi dengan frekuensi dan resolusi tertentu, pada kode project digunakan frekuensi 5000 Hz dan resolusi 8-bit sehingga nilai PWM dapat diatur dari 0 hingga 255.

2.2 Protokol MQTT dan Arsitektur Publish/Subscribe

MQTT (Message Queuing Telemetry Transport) adalah protokol komunikasi yang dirancang untuk komunikasi machine-to-machine (M2M) dan IoT, menggunakan arsitektur publish/subscribe yang memisahkan pengirim (publisher) dan penerima (subscriber) melalui perantara broker. Dalam arsitektur ini, publisher tidak perlu mengetahui siapa subscriber-nya, cukup mempublikasikan data ke topik tertentu, dan semua subscriber yang sudah subscribe ke topik tersebut akan menerima data secara otomatis.

Pada project, ESP32 bertindak sebagai subscriber untuk topik imclabled dan imclabmotorsetpoint (menerima perintah), sekaligus bertindak sebagai publisher untuk topik imclabmotorspeed (mengirim feedback), sedangkan program Python bertindak sebagai publisher untuk topik perintah dan subscriber untuk topik feedback. Broker yang digunakan adalah broker.hivemq.com port 1883, broker publik gratis yang sering digunakan untuk keperluan testing dan pembelajaran.

2.3 Kontrol Motor DC dengan Driver H-Bridge dan PWM

Motor DC adalah aktuator yang umum digunakan dalam robotika dan sistem kontrol karena mudah dikontrol dan memiliki respons yang cepat. Namun motor DC membutuhkan arus yang relatif besar (beberapa ratus mA hingga beberapa ampere) sehingga tidak dapat dikontrol langsung oleh pin GPIO mikrokontroler yang umumnya hanya mampu menyuplai puluhan mA. Oleh karena itu digunakan driver motor seperti H-bridge (misalnya L298N, L293D, atau driver sejenis) yang berfungsi sebagai interface antara mikrokontroler dan motor, dimana mikrokontroler mengirim sinyal logika rendah (low current) dan driver mengubahnya menjadi sinyal daya tinggi (high current) untuk menggerakkan motor.

Pada project ini, driver H-bridge dikontrol menggunakan tiga pin: IN1 dan IN2 untuk mengatur arah putaran motor (forward, reverse, atau brake), dan ENA untuk mengatur kecepatan

motor melalui sinyal PWM. Logika kontrol yang diimplementasikan adalah forward/stop: ketika setpoint lebih besar dari 0, pin IN1 diberi HIGH dan IN2 diberi LOW sehingga motor berputar forward, lalu pin ENA diberi sinyal PWM sesuai nilai setpoint (0–255) untuk mengatur kecepatan, sedangkan ketika setpoint sama dengan 0, kedua pin IN1 dan IN2 diberi LOW dan PWM diset 0 sehingga motor berhenti. Teknik pengaturan kecepatan menggunakan PWM bekerja dengan cara mengubah duty cycle (rasio waktu HIGH terhadap periode sinyal), semakin besar duty cycle maka daya rata-rata yang diterima motor semakin besar sehingga kecepatan motor meningkat.

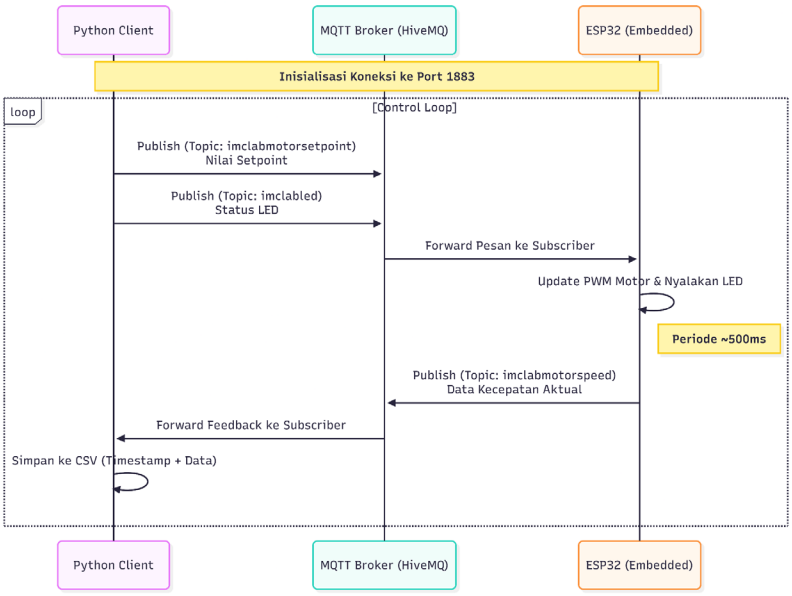
2.4 LSTM (Long Short-Term Memory) untuk Prediksi Time Series

LSTM adalah arsitektur neural network yang merupakan varian dari Recurrent Neural Network (RNN), dirancang khusus untuk mengatasi masalah vanishing gradient dan mampu mempelajari dependensi jangka panjang pada data sequential. Berbeda dengan feedforward neural network yang memproses setiap input secara independen, LSTM memiliki mekanisme memori internal yang dapat menyimpan informasi dari langkah waktu sebelumnya, sehingga cocok untuk data time series, natural language processing, dan aplikasi sequence-to-sequence lainnya. Pada project ini, LSTM digunakan untuk memprediksi nilai speed motor berdasarkan urutan (sequence) nilai setpoint yang diberikan, dengan harapan model dapat mempelajari hubungan dinamis antara perintah setpoint dan respons kecepatan motor.

Proses pelatihan LSTM melibatkan beberapa tahapan: normalisasi data menggunakan MinMaxScaler agar nilai berada di rentang 0–1 (mempercepat konvergensi training), pembentukan sequence dengan window size tertentu (pada project digunakan TIMESTEPS = 10 sehingga model melihat 10 langkah setpoint terakhir untuk memprediksi speed saat ini), pembagian dataset menjadi train dan test set, lalu training menggunakan optimizer Adam dan loss function Mean Squared Error (MSE). Setelah training, performa model dievaluasi menggunakan metrik MAE (Mean Absolute Error), RMSE (Root Mean Squared Error), dan R^2 (coefficient of determination), serta model dan scaler disimpan dalam format .h5 dan .pkl agar dapat digunakan kembali tanpa perlu melatih ulang.

BAB III
Analisis Kebutuhan Sistem

3.1 Kebutuhan Fungsional



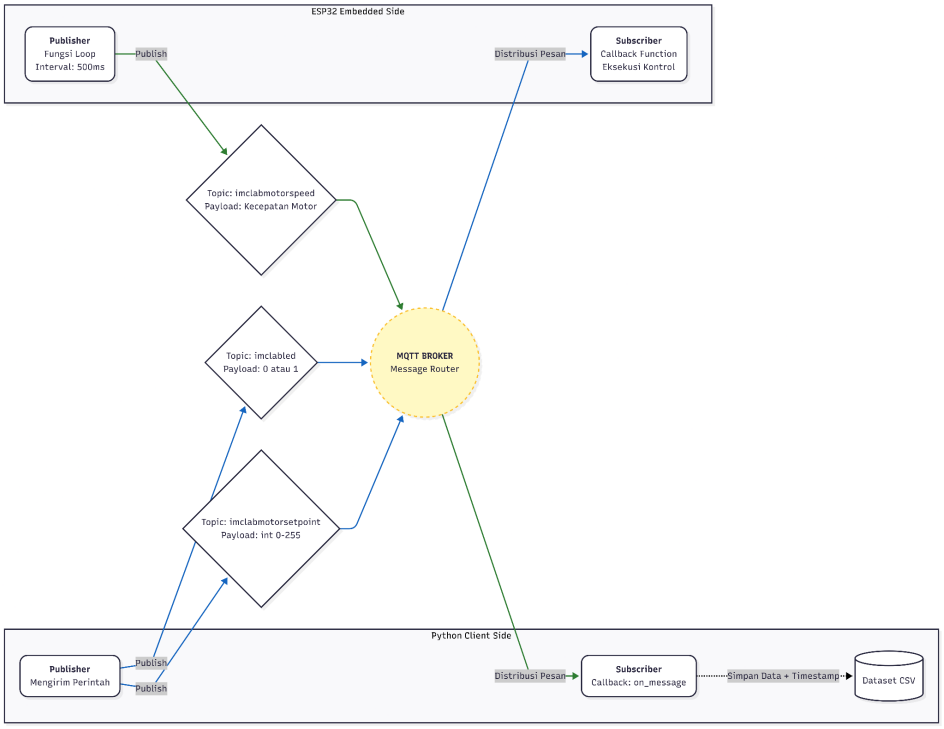
Gambar 3.1 Sequence Diagram Arsitektur Komunikasi Sistem

Berdasarkan arsitektur komunikasi sistem pada Gambar 3.1, kebutuhan fungsional dimulai dari fase inisialisasi dimana Python Client, MQTT Broker (HiveMQ), dan ESP32 harus terhubung ke port 1883 untuk membentuk jalur komunikasi publish/subscribe. Setelah koneksi terbentuk, sistem memasuki control loop yang melibatkan komunikasi dua arah.

Python Client harus mampu melakukan publish ke topik imclabmotorsetpoint (nilai setpoint 0–255) dan topik imclabled (status LED 0/1), lalu broker menerima dan mem-forward pesan ke ESP32 yang telah subscribe topik tersebut. Di sisi ESP32, firmware harus memproses payload melalui callback function dan mengeksekusi kontrol: mengatur PWM motor sesuai setpoint pada pin IN1, IN2, dan ENA, serta menyalakan/mematikan LED pada pin 2.

ESP32 kemudian melakukan publish feedback ke topik imclabmotorspeed secara periodik setiap ~500 ms yang berisi data kecepatan motor, dan broker mem-forward feedback tersebut ke Python Client yang telah subscribe. Python Client menerima feedback melalui callback on_message(), mencatat data bersama timestamp dan setpoint aktif (membentuk record time, setpoint, speed, error), lalu menyimpannya ke file CSV setelah eksperimen selesai untuk keperluan analisis dan pemodelan LSTM. Sistem ini memungkinkan kontrol dan monitoring real-time dimana pengiriman perintah dan feedback berjalan paralel secara asynchronous melalui mekanisme MQTT, sehingga data dapat terekam kontinyu tanpa blocking communication selama eksperimen berlangsung.

3.2 Kebutuhan Non-Fungsional



Gambar 3.2 Diagram Topologi MQTT Publish/Subscribe

Dari sisi non-fungsional, interval publish feedback pada ESP32 yang ditetapkan 500ms seperti ditunjukkan pada Gambar 3.2 harus dijaga konsistensinya agar data yang terekam di sisi Python memiliki sampling rate yang stabil dan tidak terjadi gap data yang berlebihan selama eksperimen. Hal ini penting karena ketidakstabilan interval publish akan menyebabkan dataset yang tidak merata dan dapat memengaruhi kualitas training model LSTM nantinya.

Callback function pada kedua sisi (ESP32 dan Python) harus dapat memproses pesan secara cepat dan tidak blocking agar message queue di broker tidak menumpuk dan komunikasi tetap responsif. Pada ESP32, callback harus dapat mengeksekusi kontrol motor (pengaturan PWM dan pin arah) dalam waktu yang singkat, sedangkan pada Python, callback on_message() harus dapat menyimpan data ke list tanpa delay yang signifikan.

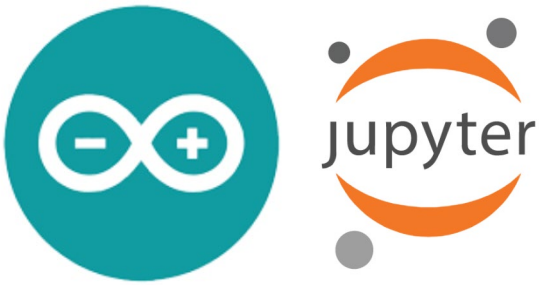
Koneksi MQTT harus reliable dengan mekanisme reconnect otomatis ketika koneksi terputus, terutama pada sisi ESP32 yang menjalankan eksperimen dalam durasi tertentu (misalnya 25 detik step test). Broker publik yang digunakan (broker.hivemq.com) harus dapat menangani message throughput yang dibutuhkan sistem, mengingat ESP32 publish setiap 500ms dan Python publish sesuai skenario step test. Dataset CSV yang dihasilkan harus memiliki format yang konsisten dengan kolom time, setpoint, speed, dan error agar dapat langsung dimuat ke notebook untuk preprocessing dan training LSTM tanpa memerlukan cleaning data yang kompleks.

3.3 Kebutuhan Perangkat Keras

Gambar 3.3 Kit IMCLAB ESP32

Kebutuhan perangkat keras meliputi ESP32 sebagai mikrokontroler yang menjalankan publisher dan subscriber MQTT, motor DC sebagai aktuator, driver H-bridge untuk interface antara ESP32 dan motor dengan pin mapping IN1 pada GPIO 27, IN2 pada GPIO 26, dan ENA pada GPIO 12, serta LED indikator pada GPIO 2. Diperlukan juga power supply yang sesuai untuk motor, breadboard dan kabel jumper untuk prototyping, serta komputer/laptop yang menjalankan Python Client dan terhubung ke jaringan WiFi yang sama dengan ESP32.

3.4 Kebutuhan Perangkat Lunak



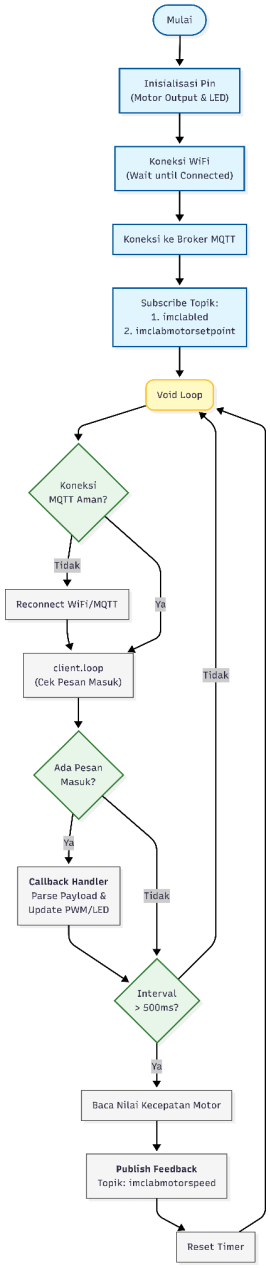
Gambar 3.4 Perangkat Lunak

Dari sisi perangkat lunak, dibutuhkan Arduino IDE atau PlatformIO sebagai lingkungan pengembangan untuk menulis, mengompilasi, dan mengupload firmware ke ESP32, dengan library yang diperlukan antara lain WiFi.h untuk koneksi WiFi dan PubSubClient.h untuk implementasi MQTT client. Dari sisi komputer/laptop, dibutuhkan Python (versi 3.x) dengan library paho-mqtt untuk komunikasi MQTT, pandas untuk manipulasi data dan penyimpanan CSV, serta library time untuk keperluan delay dan timestamp.

Untuk tahap pemodelan LSTM, dibutuhkan stack machine learning yang meliputi numpy dan pandas untuk manipulasi data, matplotlib dan seaborn untuk visualisasi, scikit-learn untuk preprocessing (MinMaxScaler) dan metrik evaluasi (MAE, RMSE, R^2), serta tensorflow/keras untuk membangun arsitektur LSTM, melakukan training, dan menyimpan model. Notebook Jupyter atau Google Colab dapat digunakan sebagai environment untuk eksplorasi data, training model, dan visualisasi hasil, karena memudahkan iterasi dan debugging dibandingkan dengan script Python biasa.

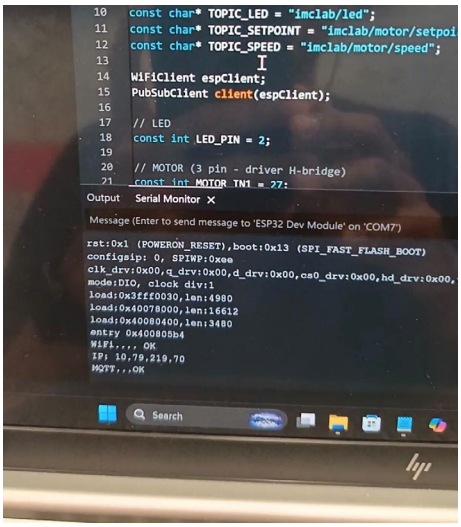
BAB IV
Implementasi dan Pengujian Sistem

4.1 Implementasi Firmware ESP32



Gambar 4.1 Flowchart Firmware ESP32

Implementasi firmware ESP32 mengikuti alur pada Gambar 4.1 yang dimulai dengan inisialisasi pin motor (IN1=GPIO27, IN2=GPIO26, ENA=GPIO12) dan LED (GPIO2) sebagai OUTPUT dengan kondisi awal mati. Pin ENA dikonfigurasi sebagai PWM dengan frekuensi 5000 Hz dan resolusi 8-bit untuk kontrol kecepatan motor 0-255.



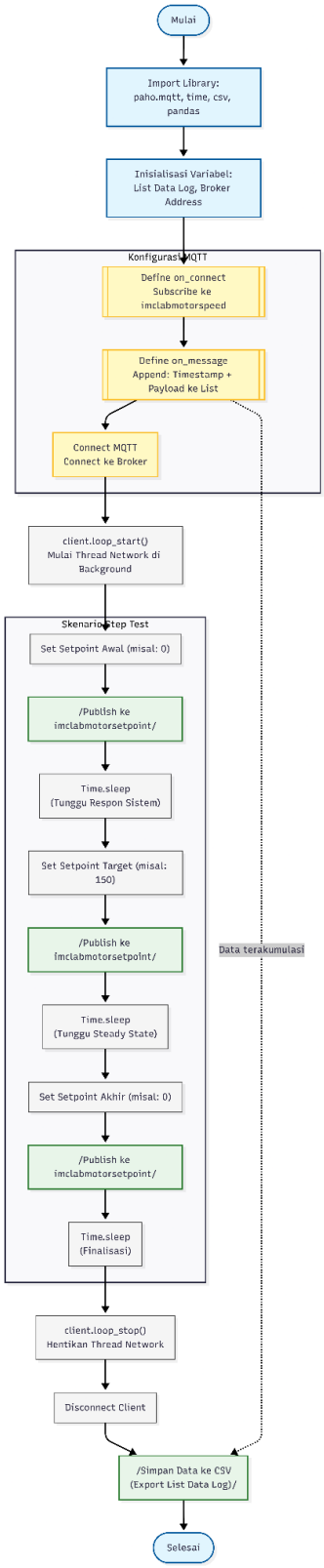
Gambar 4.2 Screenshot Serial Monitor dan kode

Firmware kemudian melakukan koneksi WiFi yang ditunggu hingga status connected, seperti terlihat pada Gambar 4.2 yang menampilkan output "WiFi...., OK" dan IP address "10.79.210.70" pada Serial Monitor. Setelah WiFi terhubung, firmware melakukan koneksi ke broker MQTT (broker.hivemq.com:1883) dan subscribe topik imclabled serta imclabmotorsetpoint, dengan konfirmasi "MQTT...OK" pada Serial Monitor. Definisi topik ini terlihat pada screenshot kode baris 10-12 di Gambar 4.2.

Pada void loop, firmware melakukan pengecekan koneksi MQTT secara kontinyu sesuai flowchart Gambar 4.1: jika tidak aman, lakukan reconnect, kemudian jalankan client.loop() untuk cek pesan masuk. Ketika ada pesan, callback handler melakukan parse payload dan update PWM/LED: untuk topik imclabled mengatur status LED, sedangkan untuk imclabmotorsetpoint mengatur arah motor (forward jika >0, stop jika 0) dan PWM sesuai nilai setpoint.

Bagian publisher diimplementasikan dengan pengecekan interval: jika sudah lewat 500ms, firmware membaca kecepatan motor (motorSetpoint), publish ke topik imclabmotorspeed, lalu reset timer sebelum kembali ke awal loop.

4.2 Implementasi Program Python MQTT Client



Gambar 4.3 Flowchart Program Python

Implementasi program Python mengikuti alur pada Gambar 4.3 yang dimulai dengan import library (paho.mqtt, time, pandas), inisialisasi list data log kosong, dan definisi broker address serta topik MQTT. Pada tahap konfigurasi MQTT (kotak kuning di flowchart), callback on_connect didefinisikan untuk subscribe topik imclabmotorspeed, sedangkan callback on_message mengappend timestamp dan payload ke list setiap kali feedback diterima dengan mencatat waktu, setpoint aktif, speed, dan error.

Program kemudian melakukan connect ke broker yang berhasil ditunjukkan pada Gambar 4.4 dengan output "Connected rc=0", lalu memanggil client.loop_start() untuk menjalankan thread network di background agar penerimaan pesan berjalan paralel dengan pengiriman perintah. Skenario step test dijalankan sesuai flowchart Gambar 4.3 dan terlihat pada output Gambar 4.4: dimulai dengan setpoint 0 ("→Setpoint: 0" dan "Speed: 0"), kemudian setpoint 100 ("→Setpoint: 100" dan "Speed: 100"), berlanjut ke setpoint 180 seperti yang terlihat pada bagian bawah screenshot.

Setelah step test selesai, program memanggil client.loop_stop() dan disconnect client, lalu menyimpan data ke CSV seperti yang ditampilkan Gambar 4.5 dengan output "✓ Saved 42 samples to motor_response_data.csv". Preview tabel menunjukkan kolom time, setpoint, speed, error, dengan pola transisi pada baris 5 (setpoint 100, speed 0, error 100) dan steady state pada baris 6-9 (setpoint dan speed 100, error 0).

4.3 Pengujian Eksperimen Step Test dan Analisis Data

	time	setpoint	speed	error
1	1766131216.9473054	0	0	0
2	1766131217.4582636	0	0	0
3	1766131217.948798	0	0	0
4	1766131218.4480028	100	0	100
5	1766131218.9562511	100	100	0
6	1766131219.5301108	100	100	0
7	1766131219.974191	100	100	0
8	1766131220.4986565	100	100	0
9	1766131220.9915752	100	100	0
10	1766131221.4529157	100	100	0
11	1766131221.9848619	100	100	0
12	1766131222.475294	180	100	80
13	1766131222.966121	180	180	0
14	1766131223.4671788	180	180	0
15	1766131223.9621527	180	180	0
16	1766131224.461203	180	180	0
17	1766131225.0292344	180	180	0
18	1766131225.4987676	180	180	0
19	1766131225.981603	180	180	0
20	1766131226.466741	255	180	75
21	1766131226.9600954	255	255	0
22	1766131227.5018785	255	255	0
23	1766131227.9700754	255	255	0
24	1766131228.5211005	255	255	0

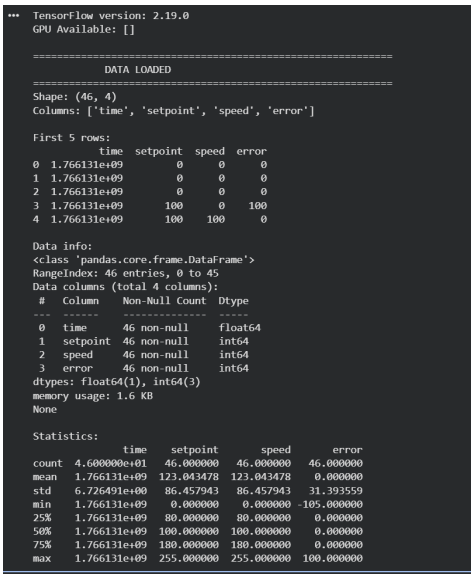
Gambar 4.4 Tabel motor_response_data

Pengujian eksperimen step test dilakukan dengan menjalankan program Python yang mengubah setpoint secara bertahap (0 →100 →180 →255 →150 →80 →0) dengan jeda waktu pada setiap step agar data feedback dapat terekam dengan baik. Hasil eksperimen berhasil disimpan dengan konfirmasi yang memuat 42 sampel data dengan 4 kolom: time (timestamp Unix), setpoint (nilai perintah), speed (nilai feedback), dan error (selisih setpoint - speed).

Tabel 4.4 menunjukkan preview data yang menampilkan pola transisi antar step dengan jelas: baris 1-3 menunjukkan kondisi awal dengan setpoint dan speed 0 (error 0), baris 4 menunjukkan transisi pertama dimana setpoint berubah ke 100 tetapi speed masih 0 (error 100), dan baris 5-11 menunjukkan steady-state dengan setpoint dan speed sama-sama 100 (error 0). Transisi kedua terlihat pada baris 12 dimana setpoint berubah ke 180 tetapi speed masih 100 (error 80), kemudian baris 13-19 mencapai steady-state dengan setpoint dan speed 180 (error 0). Pola serupa terlihat pada transisi ketiga di baris 20 dimana setpoint berubah ke 255 tetapi speed masih 180 (error 75), lalu baris 21-24 mencapai steady-state dengan setpoint dan speed 255 (error 0).

Pola transisi ini menunjukkan delay komunikasi MQTT antara saat Python mengirim perintah setpoint baru dan saat ESP32 memproses serta mempublish feedback yang telah diupdate. Perlu dipahami bahwa feedback "speed" berasal dari variabel motorSetpoint di firmware ESP32, bukan sensor kecepatan sebenarnya, sehingga data lebih merepresentasikan sinkronisasi komunikasi publish/subscribe daripada dinamika fisik motor. Data CSV ini kemudian dimuat ke notebook untuk preprocessing (normalisasi, sequence formation) dan training model LSTM.

4.5 Implementasi dan Evaluasi Model LSTM

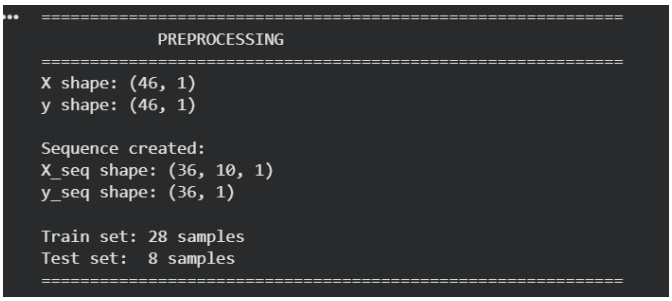


```
TensorFlow version: 2.19.0
GPU Available: []

============================================================
                    DATA LOADED
============================================================
Shape: (46, 4)
Columns: ['time', 'setpoint', 'speed', 'error']

First 5 rows:
           time  setpoint  speed  error
0  1.766131e+09         0      0      0
1  1.766131e+09         0      0      0
2  1.766131e+09         0      0      0
3  1.766131e+09       100      0    100
4  1.766131e+09       100    100      0

Data info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 46 entries, 0 to 45
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   time      46 non-null     float64
 1   setpoint  46 non-null     int64
 2   speed     46 non-null     int64
 3   error     46 non-null     int64
dtypes: float64(1), int64(3)
memory usage: 1.6 KB
None

Statistics:
               time    setpoint       speed       error
count  4.600000e+01   46.000000   46.000000   46.000000
mean   1.766131e+09  123.043478  123.043478    0.000000
std    6.726491e+00   86.457943   86.457943   31.393559
min    1.766131e+09    0.000000    0.000000 -105.000000
25%    1.766131e+09   80.000000   80.000000    0.000000
50%    1.766131e+09  100.000000  100.000000    0.000000
75%    1.766131e+09  180.000000  180.000000    0.000000
max    1.766131e+09  255.000000  255.000000  100.000000
```

Gambar 4.5 Data Loaded

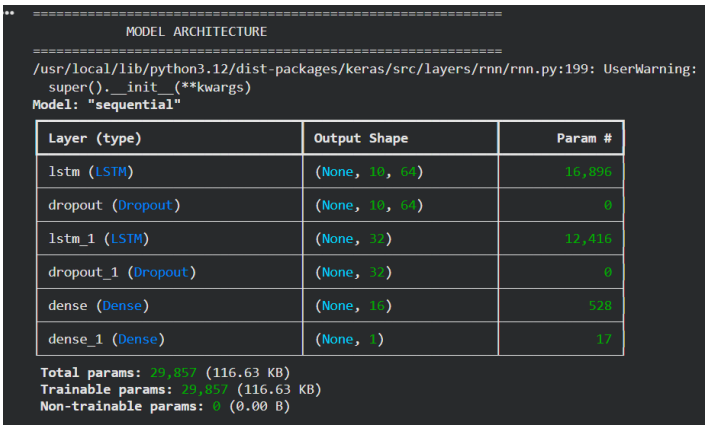
Implementasi model LSTM dimulai dengan tahap preprocessing data seperti yang ditampilkan pada output pertama yang menunjukkan dataset awal memiliki shape (46, 4) dengan

kolom time, setpoint, speed, dan error. Statistik deskriptif menunjukkan mean setpoint dan speed sama-sama 123.043478, mean error 0.0, dan standar deviasi error 31.393559, yang mengindikasikan bahwa secara rata-rata feedback speed sama dengan setpoint yang dikirim tetapi terdapat variasi terutama saat transisi.



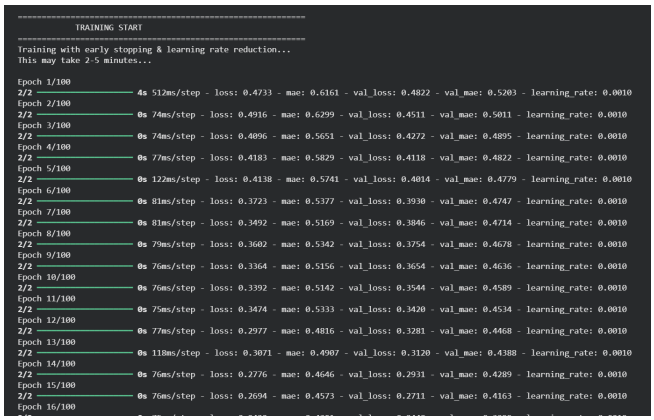
Gambar 4.6 Tahapan Preprocessing

Data kemudian diproses dengan memilih kolom setpoint sebagai input (X) dan speed sebagai target (y), menghasilkan X shape (46, 1) dan y shape (46, 1). Setelah normalisasi menggunakan MinMaxScaler, data dibentuk menjadi sequence dengan timesteps = 10, menghasilkan X_seq shape (36, 10, 1) yang berarti 36 sampel dengan 10 langkah waktu per sampel dan 1 fitur, serta y_seq shape (36, 1). Dataset sequence kemudian dibagi menjadi train set (28 samples) dan test set (8 samples) dengan rasio 80:20.



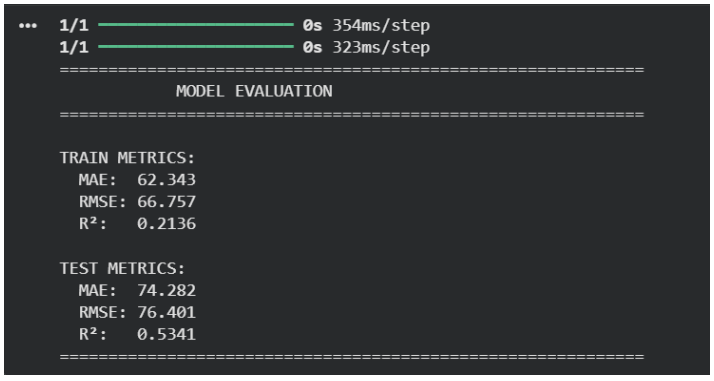
Gambar 4.7 Model Architecture

Arsitektur model yang dibangun ditampilkan pada tabel "MODEL ARCHITECTURE" yang terdiri dari: LSTM layer pertama dengan 64 unit menghasilkan output shape (None, 10, 64) dengan 16,896 parameters, Dropout layer pertama, LSTM layer kedua dengan 32 unit menghasilkan output shape (None, 32) dengan 12,416 parameters, Dropout layer kedua, Dense layer dengan 16 unit menghasilkan 528 parameters, dan Dense output layer dengan 1 unit menghasilkan 17 parameters. Total parameters model adalah 29,857 (116.63 KB) dengan semua parameters trainable dan tidak ada non-trainable parameters.



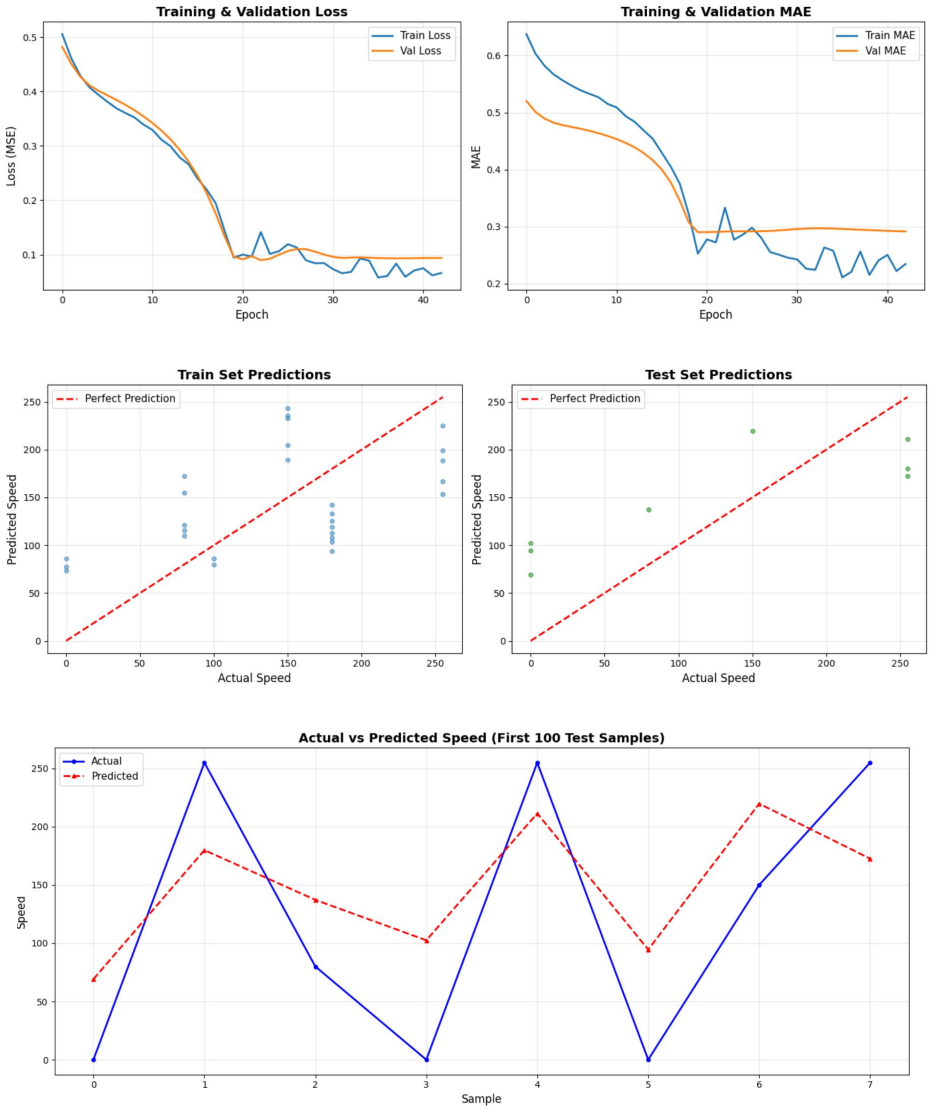
Gambar 4.8 Training Data

Proses training ditampilkan pada output "TRAINING START" yang menunjukkan model dilatih dengan early stopping dan learning rate reduction selama maksimal 100 epoch. Pada epoch pertama, training loss adalah 0.4733 dengan mae 0.6161, dan validation loss 0.4822 dengan val_mae 0.5203. Training menunjukkan pola penurunan loss yang konsisten, dimana pada epoch 16 training loss turun menjadi 0.2694 dengan mae 0.4573, dan validation loss 0.2711 dengan val_mae 0.4163. Grafik "Training & Validation Loss" menunjukkan kedua kurva (Train Loss dan Val Loss) turun secara konsisten dari epoch 0 hingga sekitar epoch 20 dimana validation loss mencapai stabil di sekitar 0.1, sedangkan grafik "Training & Validation MAE" menunjukkan pola serupa dengan MAE turun dari sekitar 0.65 menjadi stabil di sekitar 0.3.



Gambar 4.9 Model Evaluasi

Evaluasi model dilakukan setelah training selesai dengan hasil "MODEL EVALUATION" menunjukkan Train Metrics: MAE = 62.343, RMSE = 66.757, R^2 = 0.2136, dan Test Metrics: MAE = 74.282, RMSE = 76.401, R^2 = 0.5341. Nilai R^2 test sebesar 0.5341 menunjukkan bahwa model mampu menjelaskan 53.41% variansi data test, yang merupakan performa moderat mengingat ukuran dataset yang kecil (hanya 8 sampel test). Nilai MAE test sebesar 74.282 dalam skala asli (0-255) menunjukkan bahwa rata-rata error prediksi sekitar 74 unit dari nilai aktual.



Gambar 5. Hasil LSTM

Visualisasi hasil prediksi ditampilkan pada grafik "Train Set Predictions" dan "Test Set Predictions" yang menunjukkan scatter plot antara Actual Speed (sumbu X) dan Predicted Speed (sumbu Y) dengan garis merah putus-putus sebagai Perfect Prediction. Pada train set, sebagian besar titik tersebar di sekitar garis diagonal dengan beberapa outlier, sedangkan pada test set (ditampilkan dengan warna hijau) juga menunjukkan pola serupa dengan sebaran yang lebih lebar mengindikasikan adanya error prediksi. Grafik time series "Actual vs Predicted Speed (First 100 Test Samples)" menampilkan perbandingan antara Actual (garis biru) dan Predicted (garis merah putus-putus) yang menunjukkan bahwa model dapat mengikuti pola perubahan speed meskipun terdapat lag dan smoothing pada prediksi dibandingkan dengan nilai aktual yang lebih tajam perubahannya.

BAB V
Kesimpulan dan Saran

5.1 Kesimpulan

Sistem kontrol motor berbasis IoT menggunakan protokol MQTT telah berhasil diimplementasikan dengan arsitektur publish/subscribe yang melibatkan ESP32 sebagai embedded device, HiveMQ sebagai broker, dan Python Client sebagai controller. Komunikasi dua arah antara Python Client dan ESP32 berjalan dengan baik, dimana Python Client dapat mengirimkan perintah setpoint motor dan kontrol LED ke ESP32 melalui topik imclabmotorsetpoint dan imclabled, sedangkan ESP32 dapat mengirimkan feedback kecepatan motor secara periodik setiap 500ms melalui topik imclabmotorspeed.

Eksperimen step test berhasil mengumpulkan 46 sampel data dengan pola perubahan setpoint $0 \rightarrow 100 \rightarrow 180 \rightarrow 255 \rightarrow 150 \rightarrow 80 \rightarrow 0$, yang menunjukkan bahwa sistem dapat merekam data secara kontinyu dengan delay komunikasi MQTT yang teridentifikasi pada momen transisi antar step. Data yang terekam memiliki format yang konsisten dengan kolom time, setpoint, speed, dan error, serta menunjukkan pola steady-state yang jelas setelah setiap transisi setpoint.

Model LSTM yang dibangun dengan arsitektur 2 layer LSTM (64 dan 32 unit) dengan dropout regularization berhasil dilatih menggunakan 28 sampel training data dan dievaluasi pada 8 sampel test data. Hasil evaluasi menunjukkan performa moderat dengan Test $R^2 = 0.5341$, Test MAE = 74.282, dan Test RMSE = 76.401, yang mengindikasikan bahwa model mampu menjelaskan 53.41% variansi data test dan memiliki rata-rata error prediksi sekitar 74 unit dalam skala 0-255. Visualisasi prediksi menunjukkan bahwa model dapat mengikuti pola perubahan speed meskipun terdapat lag dan smoothing pada prediksi dibandingkan nilai aktual.

Perlu dicatat bahwa pada implementasi saat ini, feedback "speed" yang dipublish ESP32 berasal dari variabel motorSetpoint di firmware, bukan dari pembacaan sensor kecepatan motor yang sesungguhnya, sehingga data lebih merepresentasikan sinkronisasi komunikasi publish/subscribe daripada dinamika fisik motor seperti inersia, overshoot, atau settling time.

5.2 Saran

Untuk pengembangan lebih lanjut, disarankan menambahkan sensor encoder atau Hall effect pada motor DC agar dapat membaca kecepatan motor yang sesungguhnya, sehingga dataset dapat merepresentasikan dinamika fisik motor seperti inersia dan overshoot secara lebih akurat. Ukuran dataset yang saat ini hanya 46 sampel perlu diperbesar dengan melakukan eksperimen berulang atau menambah variasi pola setpoint (random step, sinusoidal, ramp) agar model dapat menggeneralisasi lebih baik.

Eksplorasi hyperparameter tuning seperti variasi jumlah layer LSTM, unit per layer, dropout rate, learning rate, dan timesteps dapat dilakukan untuk meningkatkan performa model yang saat ini mencapai Test $R^2 = 0.5341$. Sistem juga dapat dikembangkan dengan menambahkan fitur closed-loop control dimana prediksi LSTM digunakan untuk mengatur setpoint motor secara adaptif, sehingga sistem dapat melakukan self-tuning untuk mencapai target kecepatan dengan lebih akurat.

Untuk meningkatkan reliability sistem IoT, implementasi QoS level 1 atau 2 pada MQTT dan buffer data pada ESP32 disarankan agar pesan tidak hilang saat koneksi tidak stabil, yang penting untuk aplikasi industri.