PID Control Of The Angular Position Of A DC Motor

Team 4

Team Members

Pratyanshu Pandey
Vedansh Mittal
Bhaskar Joshi
Utkarsh Upadhyay
Prince Singh Tomar

Index

Abstract

The position control of motors is crucial in an application for precise control systems like moving robotic arms to pick stuff in the auto assembly line. Robots arms are now being used in very precise works like performing operations. Therefore the control of the position of motors will become more important in the future. Generally in a DC Motor, speed control can be achieved by varying the terminal voltage but position control of the shaft cannot be achieved efficiently. Hence we will use PID to control the angular position of the DC motor. The design document contains components and conceptual ideas of the project while the IoT document contains details about the hardware specification and code.

Developer Document

Introduction

Problem Statement

The aim of the project is to reach the target angle using PID logic from any given initial position. We also wish to give the user more room for experimenting with the PID constant along with the option of providing the target angle. Along with these the project also includes security, a dashboard to show the collected data, statistics based on the previous results of PID tweaks.

Purpose

The use of DC motors are present in a lot of fields like defense, industry, robotics, medical, etc. Control of the position of the angular motor is crucial for the working of many machines in these fields like controlling an arm of a robot. These require the motor to rotate at a particular angle, Hence Control of the motor position is required. We also created a dashboard for the users' check do experimentation with PID constants on their defined target angle

Overview

The system includes :

1. Final product:

   A microcontroller running PID algorithm controlling motor shaft whose target angular position can be provided remotely via the dashboard and supporting statistics and analytics on the sensor readings viewable from dashboard over the internet.

2. Platforms:

   OM2M platform to communicate from the microcontroller. Database to store permanent data. Cloud platform to host the dashboard.

3. Dashboard:

   A dashboard to provide the target angular position and PID constants to conduct a trial and observe the results on a graph and a live stream.

Design Document

a. System Requirements

The following points specify the system requirements:

- The system must be able to measure the angular position of the motor shaft. The sensor readings would be in terms of incremental counts which need to be converted into angles in degrees after some calibration.
- The system must be able to send period control signals to the motor to control its angular position based on the PID logic.
- The system must allow the user to send the target angular position of the motor remotely.
- The system must allow the user to tweak the PID parameters remotely for a particular trial.
- The system must graphically represent the current state of the motor to the remote user.
- The system must provide a live stream of the motor angular position to the users.
- The system must generate and display statistics of past trials.
- The system must allow new users to create accounts.
- The system must allow existing users to log in using defined credentials.
- The data in the system must be secure.
- The system must detect data corruption.
- The system must be robust to power and network failure and system crashes.

b. System Specifications

The following are the specifications of each component of the system:

- ESP32 Microcontroller
  The microcontroller runs the main PID algorithm and reads the angular position of the motor using a sensor and generates control logic commands periodically. It has an inbuilt WiFi client which connects to the internet for sending readings and receiving commands.

- LN298N Dual H Bridge DC Motor Driver Controller
  The motor driver is responsible for converting command signals from ESP32 for controlling the motor angular position to appropriate voltage using a power source from an external battery.
- DC Metal (with Encoder) Gear Motor GA25-370
  The motor has a 25mm gearbox length and supports:
  - Rotation speed: 35 RPM
  - Reduction ratio: 171
  - Load Speed: 27 RPM
  - Max. Efficiency: 4 Kg-cm
  - Stall Torque: 9 Kg-cm

  The Motor has an inbuilt quadrature encoder added to the micro metal gear motors (with extended back shaft), these motors use a magnetic disc and hall effect sensors to provide 11 counts per revolution of the motor shaft. The sensors operate from 3.3 V to 5 V and provide digital outputs that can be connected directly to a microcontroller or other digital circuit.
- Visual measurement setup
  The motor shaft is attached to a custom-made protractor with a pointer which allows the users to visually verify the angular position of the shaft.

c. Stakeholders

The stakeholders of the system are:

- System Owner: IIITH
- System Advocate: Dr. Harikumar Kandath and Sudhansh Yelishetty
- System Developers: Pratyanshu Pandey, Vedansh Mittal, Bhaskar Joshi, Utkarsh Upadhyay, Prince Singh Tomar
- Users: Anyone who has an account on the dashboard.

d. System Components

- Microcontroller
  The microcontroller used is ESP32 which runs the PID algorithm and controls the motor and sends readings to the cloud.

- Motor Driver
  Converts signals from ESP32 to appropriate voltages using external battery power sources.

- DC Motor
  The motor whose angular position is to be controlled.

- Rotary Encoder
  Sends incremental counts as two voltage signals which are decoded to compute the angular position of the motor shaft.

- OM2M server

It serves as an intermediary between the dashboard and ESP32 exchanging messages and storing them.

- MongoDB Atlas (Database)
  The cloud server stores user credentials and statistics information and provides and updates the data as requested by the dashboard.

- Dashboard:
  It consists of two subcomponents:
  - Backend server: verifies all the requests made by the frontend and fetches the data from OM2M and Database and updates it.
  - Frontend: Allows users to interact with the IoT system and get live feedback.

Design Details:

Conceptual Flow of Development

The project required systematically working on several components in multiple phases. Here we give the conceptual flow of how the project was developed.

1. Choosing the Hardware

1. Microcontroller:
   ESP32 was picked over ESP8266 and Arduino boards. ESP32 has inbuilt WiFi capabilities, a faster and multicore processor, high program storage space, and comes at a decent price for these features. These capabilities make it a perfect choice for the task.

2. Sensor:
   The sensors used for measuring the angular position of a rotating shaft of a DC Motor are called rotary encoders. These are of 2 kinds - relative and absolute encoders. We picked a quadrature rotary encoder (relative encoder - measures relative angular position) because of its cheaper cost and easy availability. Quadrature encoders are also available as pre-attached on the shaft of DC motors with a unified pinout.

3. DC motor with Encoder :
   The geared DC Motor comes with the encoder attached to the rear shaft of the motor. If the resolution of the encoder is X on the rear shaft then it is Gear Ratio * X on the main shaft. On the contrary, if the RPM of the rear shaft in R then the RPM for the main shaft is R / Gear Ratio. These factors counteract each other. We need a system with good resolution and speed on the main

shaft. The GA25-370 (12V 130RPM) Geared DC Motor with Gear Ratio of 46 and Encoder resolution of 11 on the rear shaft was a good fit.

4. Motor driver:
   Given the DC Motor and Microcontroller, L298N DC Motor Driver was picked as it supports motors up to 36V, seamlessly integrates with ESP32, is easily available, and is rather cheap.

2. Hardware Integration and PID Logic

1. This phase included designing the circuit and interfacing the hardware with ESP32. The sensor was then calibrated to match the output in the real world.
2. Initially, a simple version of PID Logic was implemented and experiments were performed to fine-tune the three constants in the PID control.
3. The PID algorithm was then further optimized using selective use of the integral term to improve accuracy and convergence times.

3. OneM2M integration

1. In this phase, we integrated the ESP32 subsystem with the OneM2M server hosted on IIITH servers. The ESP32 was configured such that it reads the target angle from OneM2M and sends back sensor readings periodically.
2. This created an issue. Since posting data to OneM2M was taking a lot of time, the time between successive iterations of the PID algorithm became high and the convergence time increased dramatically.
3. To solve this ESP32 was reprogrammed to use core 0 for all the networking tasks and core 1 for PID logic.

4. Dashboard Frontend and Backend

1. The backend was set up as a medium of communication between OM2M and the frontend.
2. The front end was created to give users live feedback of the experiments, showing the plot of target angle and current position and live twitch stream of DC motor.

5. Security and Authorization

1. Communication between ESP32 and the backend through OneM2M was encrypted using a custom end-to-end symmetric encryption algorithm.
2. Hashing techniques were employed for the verification of data.
3. This meant that all data stored in One M2M servers were now in an encrypted and hashed format.
4. Communication between the frontend and backend is encrypted through HTTPS.

5. User login/registration along with JWT was added for authorization. A MongoDB database provider Mongo Atlas was used for storing credentials.

6. Statistics and Optimisations

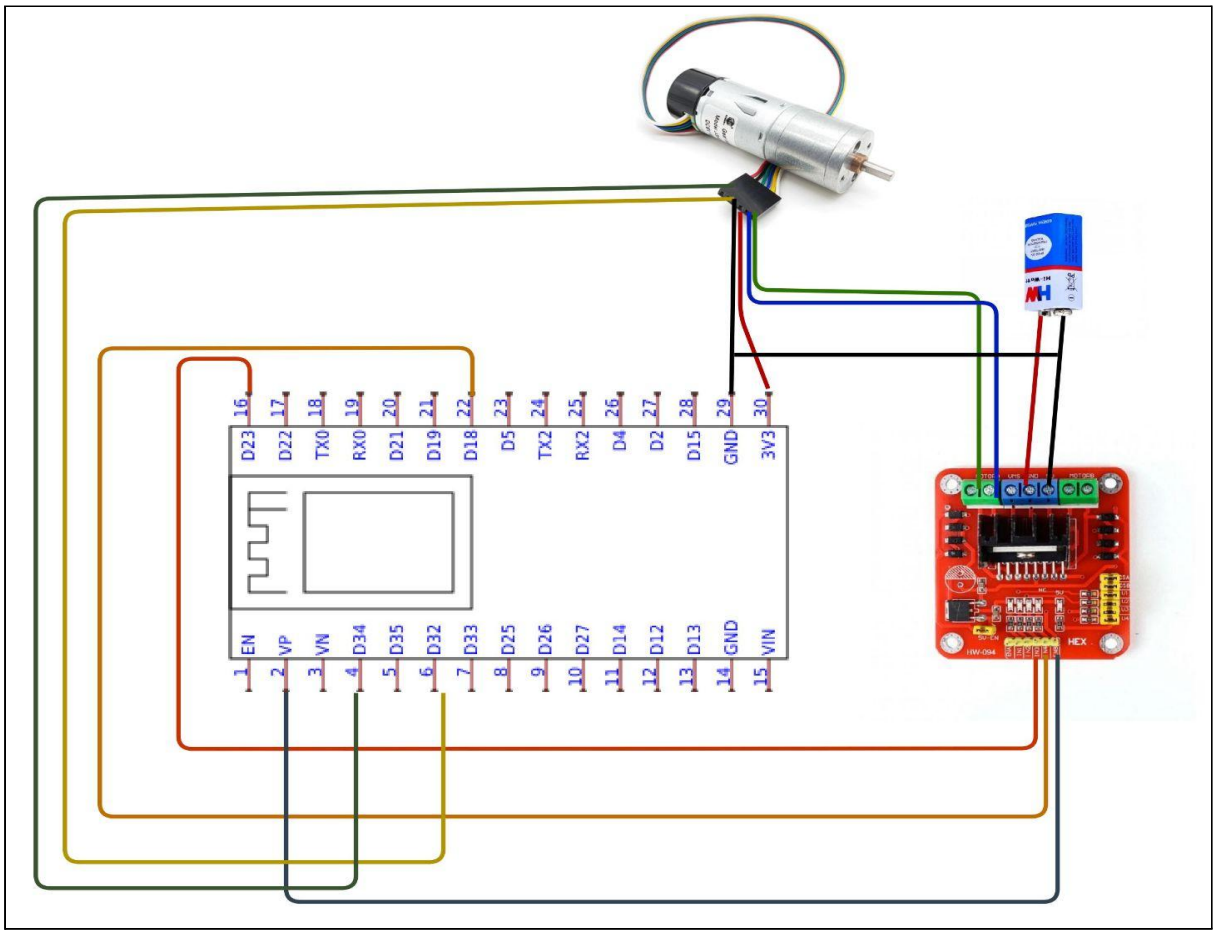
1. Additional statistics like the total number of runs and the average convergence time were also added to the system.
2. The option of filling PID constants along with the target angle was added to the system.
3. The PID algorithm was further optimized with uniform sampling.
4. The custom end-to-end symmetric encryption algorithm was further improved.

Subsystems Description

ESP32 Subsystem:

The ESP32 subsystem consists of the main circuit along with all the software that goes with it. This subsystem is responsible for getting target angle and PID constants from OneM2M servers, running the PID algorithm with these values, and then periodically sending sensor data back to OneM2M.

Circuit Diagram:



Circuit Diagram

How it Works?:

The ESP32 microcontroller has 2 processing cores and FreeRTOS allows us to multiprogram these cores. Thus, on startup, the subsystem is set up by initializing Serial, WiFi, and pin modes. Then the 2 cores are initialized with their respective tasks.

Core 0:

1. This core is responsible for all the networking tasks.
2. It begins by checking if ESP32 is connected to WiFi. If not it retries connecting to WiFi.
3. Then it sends a GET request to the OneM2M server to get all available data containing target angles and PID constants.
4. The core also maintains a list of identifiers of the recently finished trials. The data received from OneM2M is matched against this list and the new entries in the data are identified.
5. One by one, for each entry, we verify the hash, decrypt the data, parse the values and send it to core 1 using an IPC message queue.
6. The core then waits for getting sensor data through another IPC message queue. Each set of sensor data is formatted appropriately, encrypted, hashed, and sent

to the OneM2M server. This goes on till we receive an end token denoting the end of the current trial.

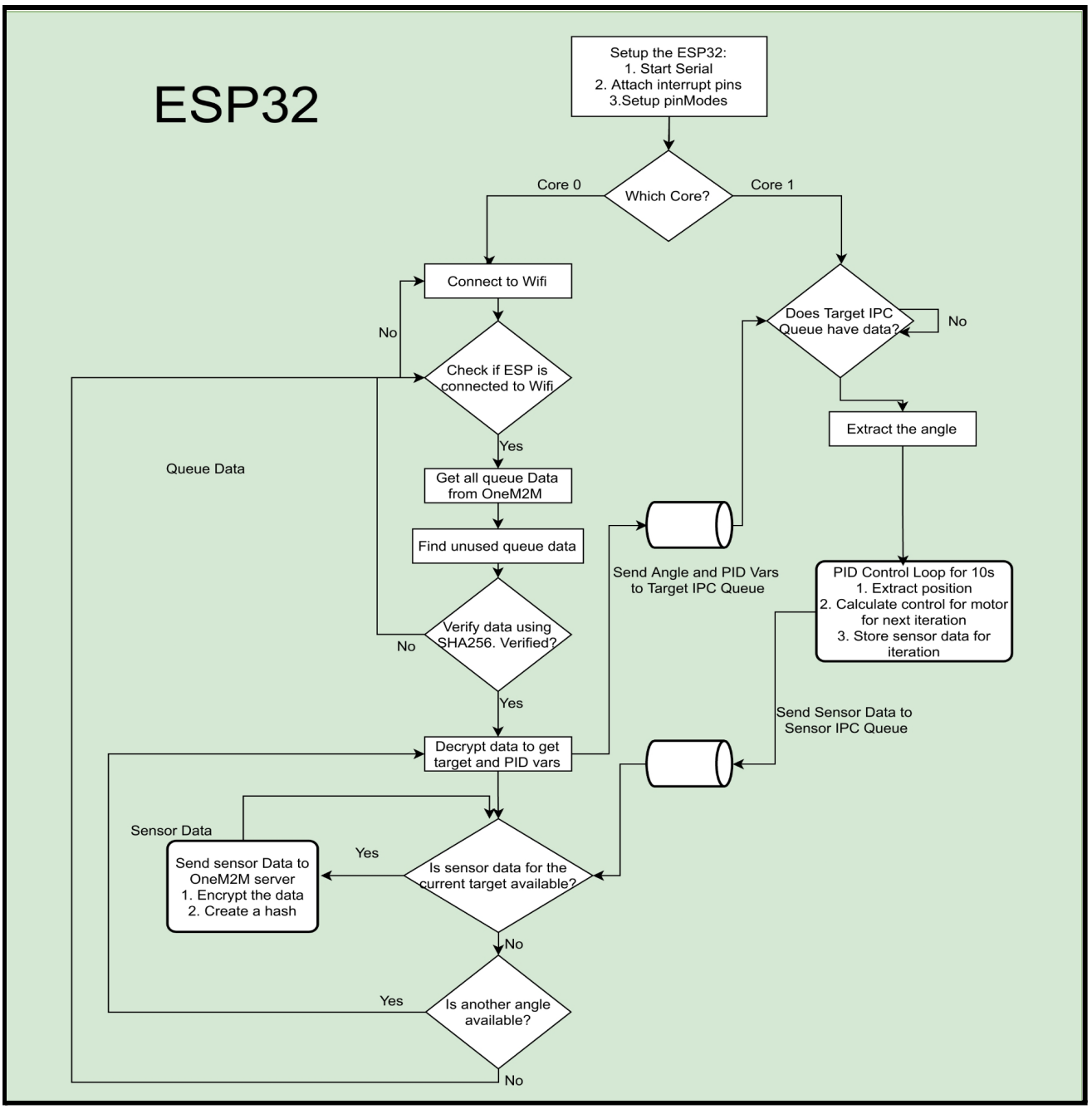
7. On receiving the end token, the last set of data is sent to the OneM2M server and we move to the next entry.

Core 1:

1. The core on setup waits on the IPC message queue for target data for the next trial. On receiving target data for a trial, the PID algorithm starts.
2. For each iteration of the algorithm:
    a. The current angular position is read from the sensor and stored in a structure along with the timestamp.
    b. The PID terms are calculated according to the formula.

$$u(t) = K_\mathrm{p} e(t) + K_\mathrm{i} \int_0^t e(\tau)\,\mathrm{d}\tau + K_\mathrm{d} \frac{\mathrm{d}e(t)}{\mathrm{d}t},$$

    c. The control signal is generated according to the formula but is capped at 255. This control signal is provided to the motor using PWM.
3. The sampling rate for reading sensor data, which is also the time between 2 iterations of the PID loop, is uniform and equals 100 ms.
4. An optimization in the PID algorithm is that the integration term is only used once a certain level of convergence is reached. This improves convergence times and reduces final error.
5. Periodically after 3 iterations, the sensor data stored is sent to core 0 by an IPC message queue.
6. The PID algorithm runs for 10 seconds per trial and at the end, an end token is sent to core 0.



The flow of Code for ESP32 Subsystem

How data is extracted from the sensor?:

1. The quadrature rotary encoder works through interrupts and a counting variable that is initialized with 0. It has 2 lines A and B.
2. An interrupt is attached to line A. If the signal on line A is rising it implies a change in angular position and line B is sampled:
    a. If on sampling B > 0 it implies angular position has increased by 1 and the counting variable is incremented.

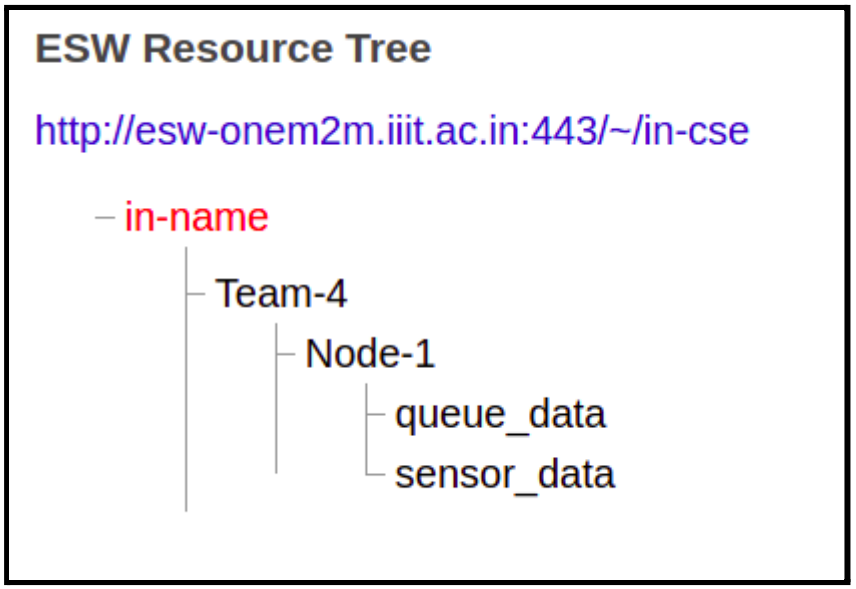
    b. If on sampling B <= 0 it implies angular position has decreased by 1 and the counting variable is decremented.
3. The counting variable is global and can be sampled by the PID function. But the counting variable does not directly correspond to 1 degree.
4. After experimentation and calibration, it is found that 1 count = 0.64056939501 degrees.

OneM2M Subsystem:

The OneM2M subsystem is the OM2M server hosted on IIIT servers. The subsystem acts as an intermediary between the backend and the ESP32.

Resource tree structure?:

The resource tree of the Subsystem is:



- **queue_data:**
  This container has size 5 and contains values for the target angles and PID constants.

- **sensor_data:**
  This container has size 115 and contains data from the sensor for the current and some previous trials.

Data Structure:

Each container instance has an attribute “**rn**” or resource name which is unique and can be used as an identifier. Both the above-mentioned containers store data in the format of

“SHA256 hash of encrypted data“ + “|” + “encrypted string”.
Thus data in the OM2M server is always encrypted. The encrypted string is different for the 2 containers:

- **queue_data:**

Here the encrypted string is the encrypted version of Target angle, K_p, K_i, K_d concatenated into a single string separated by a colon(:);

- **sensor_data:**

Here the string starts with the resource name of the trial to which this sensor data belongs, followed by a colon and the target angle.
The sensor data consists of pairs of timestamps and current angular positions. The string for each pair is formed as “timestamp” + “:” + “angular position”.
Each of these pairs of values is concatenated to the string using a comma as a separator.
The final string thus is
<resource_name>:<target_angle>,<timestamp1>:<position1>,<timestamp2>:<position2>
.....

The end token consists of an additional pair of 10000000:-10000000.

Dashboard

Backend

The backend server has been developed using the ExpressJS library in NodeJS. It is responsible for the following operations:

- Verifying new user registrations requests frontend against if the user already exists, and creating new users in the database (MongoDB Atlas).
- Verifying login requests from the front end and providing tokens (JWT) for the session.
- Providing statistics data from the database.
- Updating the statistics in the database upon completion of a trial.
- Providing data related to the current trial of the experiment for authorized users.
- Forward data provided by the user (target angle and PID constants) after authorization to ESP32 (via OM2M).

Frontend

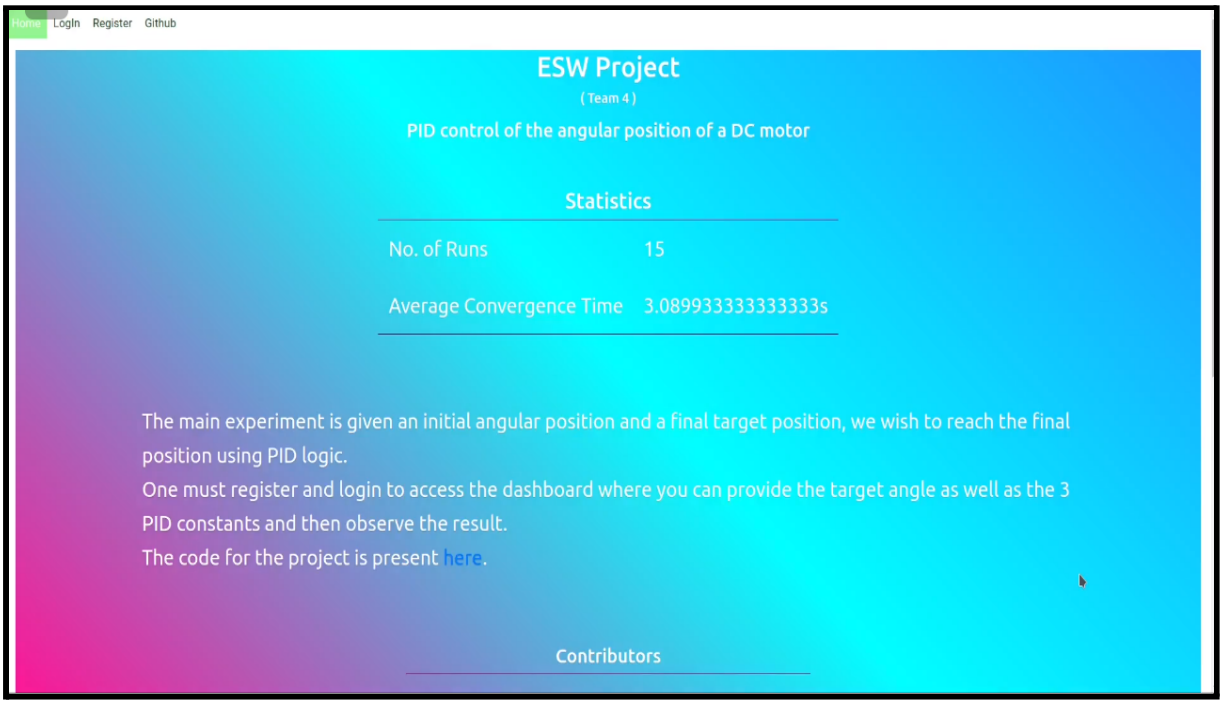
- Description :

  The frontend is developed in React JS. It contains the following functions:

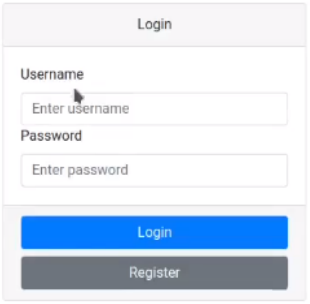
  - User Login and Registration. Gives access to Dashboard after the user is verified from the backend.

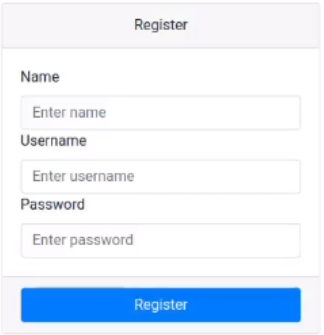
- Sending User details to backend upon registration of new users.
- Dashboard to show the angle reading received from the backend server which in turn gets data from the OM2M platform.
- Form to tweak PID constants and target angle, and receive feedback.
- Live stream of DC motor using twitch.
- Displaying statistics obtained from past experiments.

- UI Design :

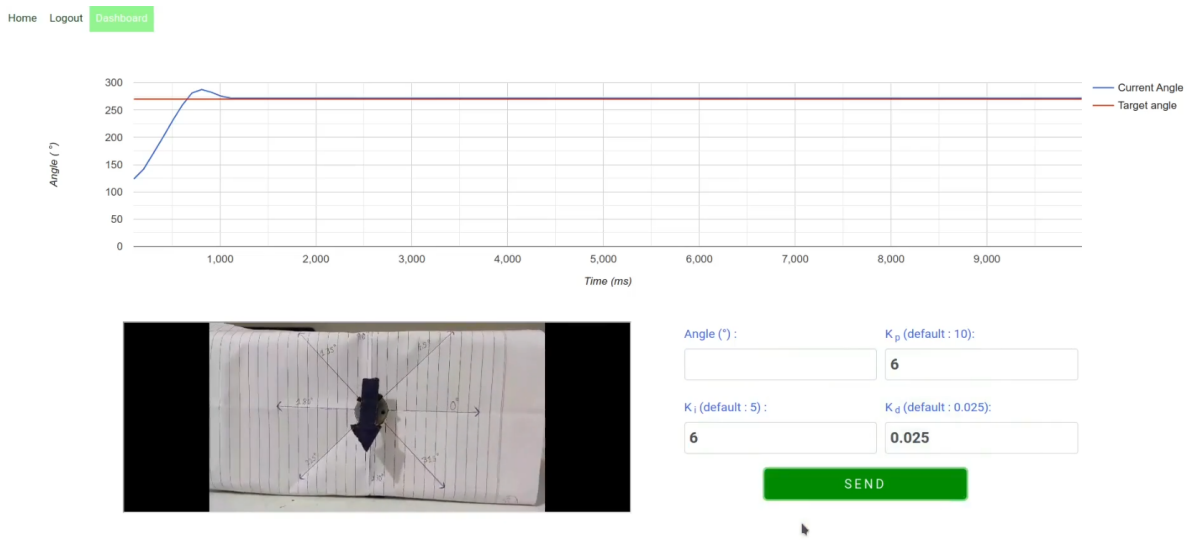


About Page



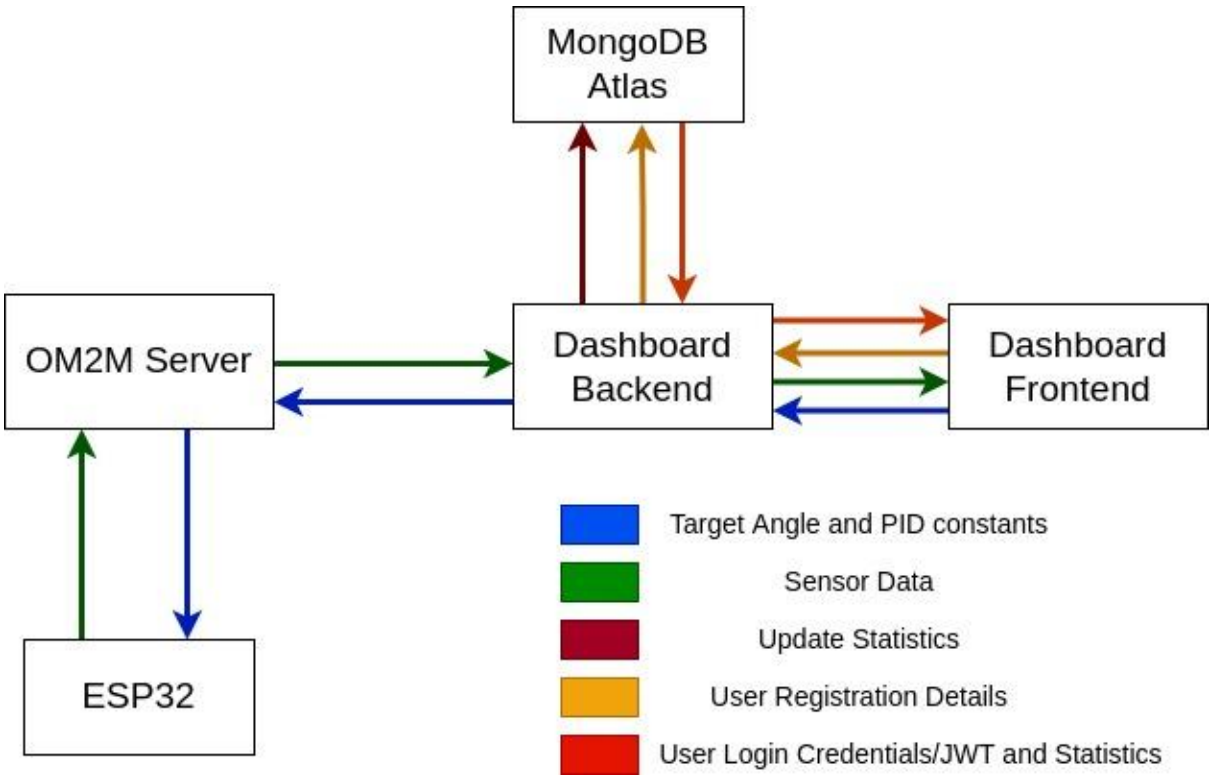


Login Form / Register Form



Dashboard

Subsystems Interactions



The pictorial representation of Subsystem Interactions

Security Algorithms:

Verification/Hashing:

The algorithm used for hashing is SHA256.

Encryption/Decryption:

For encrypting data to be stored in, we use a custom end-to-end symmetric encryption algorithm. The secret for the algorithm consists of 3 parts:

1. The key string: a string
2. Byte Substitution Array: An array of derangement of numbers from 0 to 255
3. Derangement Array: An array of derangement of numbers from 0 to key length - 1.

The number of rounds is given by: key_length % 4 + 2.

For encryption:

1. First, the input string is padded with "#" till its length is a factor of key_length.
2. For each round of encryption:
    a. Each character in the text is substituted with the corresponding ASCII character in the byte substitution array.
    b. Now the text is divided into blocks of size key_length. For each block, a string rearrangement is performed using the derangement array. The ith element of the array is the index to which the ith element in the byte substituted text goes.
    c. Now each block of the text is XOR'd with the key string.
3. The final text is then further encoded to base64.

For decryption:

1. First, the reverse byte substitution array and the reverse derangement array are calculated.
2. The text is first decoded from base64 notation.
3. For each round of decryption:
    a. Now the text is divided into blocks of size key_length and each block is XOR'd with the key string.
    b. For each block, the string rearrangement is performed using the reverse derangement array.
    c. Each character in the text is substituted with the corresponding ASCII character in the reverse byte substitution array.
4. After all the rounds the padded characters are trimmed from the end of the string.

Communication

The various subsystems defined interaction between each other as defined below. The communications over the networks are completely encrypted and robust as specified.

- Between ESP32 and OM2M

  The communication between ESP32 and OM2M server happens via the HTTPS requests as defined in oneM2M standards. The request needs to have **user_id** and **key** in the header for authorization.
  The data messages are again encrypted to secure the data in case the data from OM2M gets compromised. The encrypted content is then hashed to allow data corruption detection.
  There are 2 communication channels between the components:
  - ESP32 to OM2M: The sensor data generated by the PID loop is formatted in the appropriate data structure, encrypted, hashed, and sent to the **sensor_data** container of the OM2M server.
  - OM2M to ESP32: All the data for trial in **queue_data** container is fetched. The hash of each entry is verified, the data is decrypted, the values are parsed and forwarded to the PID function.

- Between backend and OM2M

  The communication between backend and OM2M server happens via the HTTPS requests as defined in oneM2M standards. The request requires **user_id** and **key** to get done. The data messages are again encrypted using a symmetric key encryption algorithm to secure the data in case the data from OM2M gets compromised. The encrypted content is then hashed using the SHA256 algorithm to allow data corruption detection.
  .The following are the two communication channels between the two components:
  - Backend to OM2M: The data received from the frontend to initiate a trial at the ESP32 is parsed by the backend, and sent to the **queue_data** container of the OM2M. The data is encrypted and hashed as defined.
  - OM2M to Backend: Upon request from the frontend, the sensor readings stored in the **sensor_data** container of the OM2M server are fetched. The data is checked against encryption and hashing as defined.

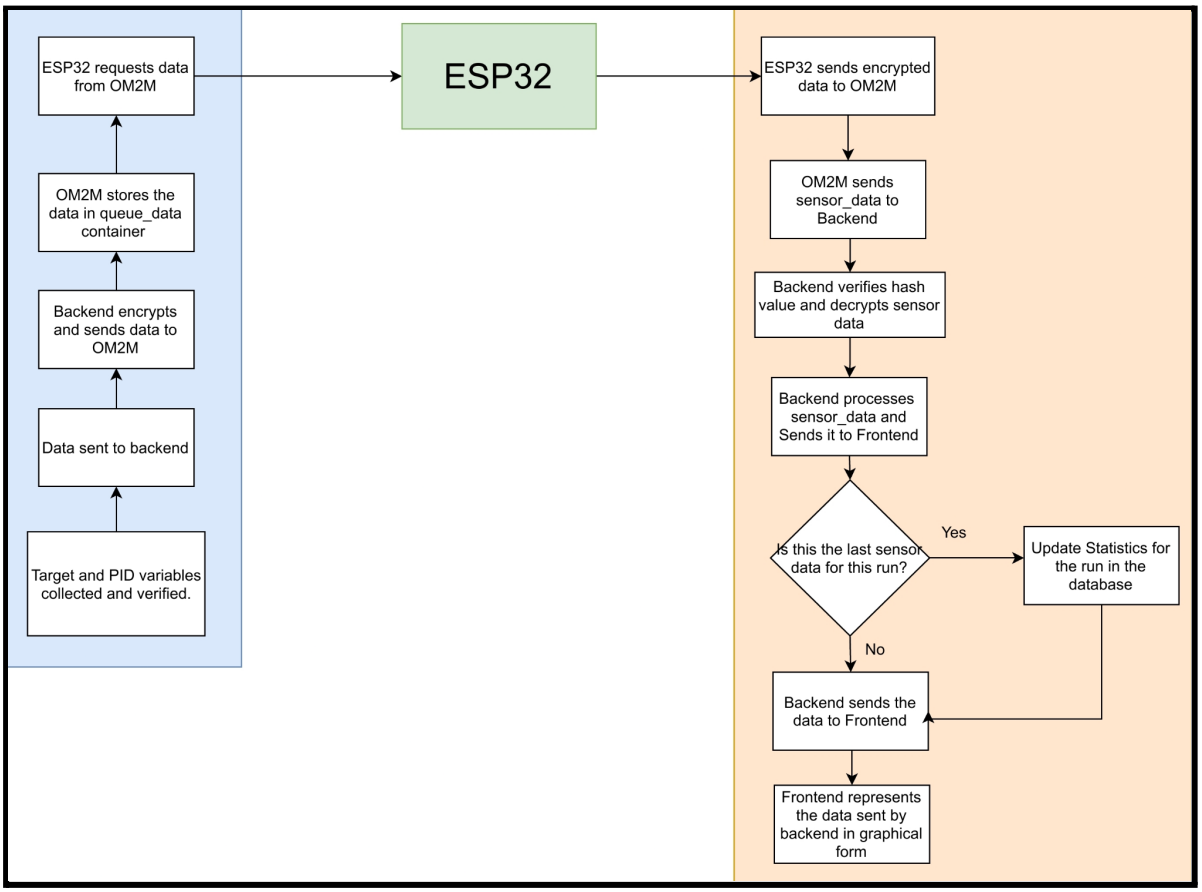
- Between backend and frontend

  Backend and frontend communicate via HTTPS requests. The communication between frontend and backend uses XMLHttpRequests of Axios library. The verification of the user is done by sending the token to the backend and receiving user-related data. The Frontend sends requests to Backend after every 1000 milliseconds demanding the plot and statistics data, which backend, in turn, gets from OM2M and Mongo Atlas.
  - Backend to Frontend: Request user verification by sending JSON web token. Receives request after every 1000 milliseconds for plotting and statistical data. Receives details of the user upon new registration for storing on cloud. The User password is encrypted using the bcrypt library before saving it on the cloud.

- Frontend to Backend: Sends values of target angle, PID constants to the backend which are encrypted and hashed before sending them to OM2M

Data Flow

The following flow diagrams show the flow of data in the system.



Queue and Sensor Data Flow

Operational Requirements

System needs

- ESP32 needs a stable power supply via a power bank or similar source.
- ESP32 needs a stable WiFi connection preferably with less ping.
- The Motor Driver needs external 9V batteries to drive the motor.
- As an initial setup, we need to calibrate the rotary encoder to convert incremental counts to angels in degree.
- Browser access to interact with the dashboard.

Analytical System

We provide analytics of two types, live statistics regarding the current trial of the experiment and the statistics over the past conducted trials.

- For live trial: We provide a live graph to be displayed on the dashboard which features a plot of the current angular position and the target angular position against the time. This information is displayed at the time user initiates a trial from the dashboard.
- For overall statistics: We store the total count of the number of trials done till now and the average convergence time, where convergence time is defined as time spent from the start of the experiment till the current angular position stays under 4 degrees of error from the target angle. This information is displayed on the home page of the website.

Code Links

- ESP32 code: https://github.com/pratyanshupandey/ESW-Project
- Dashboard code: https://github.com/VedanshM/esw_dashboard

IoT Project Document

Design Documents

Refer to the above chapter on Design Document.

a. Hardware Specifications

i. Microcontroller: Espressif ESP32-WROOM-32D board

ESP32-WROOM-32 (ESP-WROOM-32) is a powerful, generic Wi-Fi+BT+BLE MCU module that targets a wide variety of applications. We use the ESP32 Wi-Fi Module to communicate with the OM2M server for the exchange of data.
We use the following pins in the microcontroller:

- PIN D2: for providing Pulse Width Modulation (PWM) for the DC motor.

- PIN D32, D34: for reading the DC motor rotary encoder with interrupts.
- PIN D18, D23: for selecting the direction of the motor.
- PIN GND: for connecting DC motor to ground.
- PIN 3V3: for connecting DC motor to VCC.

ii. Motor: 25mm DC Metal Encoder Gearmotors - GA25-370

This is a DC Motor with Quadrature Encoder. With quadrature encoders added to the micro metal gear motors (with extended back shaft), these motors use a magnetic disc and hall effect sensors to provide 11 counts per revolution of the motor shaft. The sensors operate from 3.3 V to 5 V and provide digital outputs that can be connected directly to a microcontroller or other digital circuit.
It's specifications are:

- Voltage range: 3.3V - 5V
- Stall Current: 1.8A (6V), 1.3A (12V), 0.7A (24V)
- No-Load Current: <0.10A (6V); <0.06A (12V), <0.04A (24V)
- At Load Current: 0.45A (6v & 12V), 0.25A (24V)
- Angular speed: 130 RPM
- Reduction ratio/ Gear Ratio: 46
- Load speed: 100 RPM
- Max. efficiency (torque): 1 kg-cm
- Stall torque: 3.6 kg-cm
- Gear box length: 21 mm

iii. Motor Driver: L298N Dual H Bridge DC/Stepper Motor Driver Controller Module

The L298N Dual H Bridge DC/Stepper Motor Driver Controller Module is for driving two robot motors. It uses the popular L298N Dual H-Bridge Motor Driver chip and is powerful enough to drive motors from 5-35 Volts at up to 2 Amps per channel. The flexible digital input controls allow each motor to be fully independent with complete control over speed, direction, and braking action, as well as the 2 phase stepper motor.
It has the following specifications:

- Driver IC: L298N
- Input Supply Voltage (VDC): 5 ~ 35
- Supply Current (A): 2 (peak per channel)
- Logic Voltage (V): 4.5 to 5.5
- Logic Current (mA): 0 to 36
- Control signal input voltage (V): 4.5 to 5.5
- Maximum Power (W): 20
- Operating Temperature (°C): -25 to 130
- Length (mm): 55
- Width (mm): 60
- Height (mm): 30
- Weight (gm): 35

iv. Battery: 9V Original HW High-Quality Battery

It has a Universal 9V battery size and connecting points, which are required for the DC motor. It also provides a constant voltage (of 9V) until it lasts.
Specifications:

- Dimensions (mm) LxWxH: 4.5 x 2.2 x 1.5
- Weight (gm): 178
- Discharge Time: 270hm, 9hours
- Nominal Voltage (V): 9
- Jacket Material: Metal
- Discharge Terminal Type: Press Stud Terminal
- Size (L x W) mm: PP3

v. Battery snap connector: 9V 10cm Battery Connector

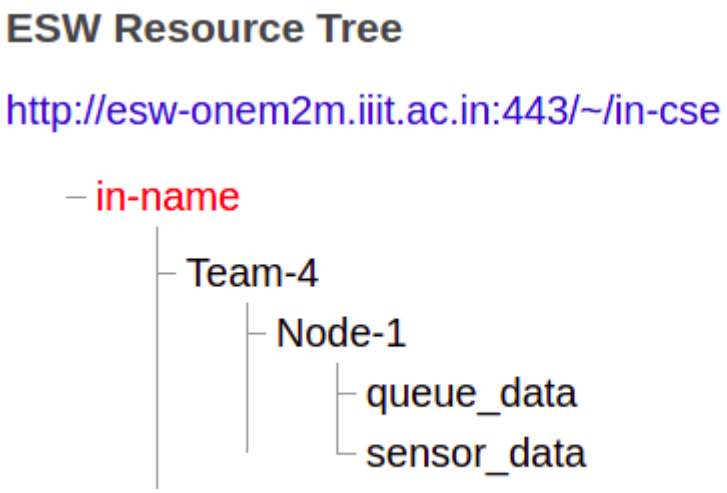
This is a 9V Battery Connector with a 10 cm cable. It is used to connect a 9V battery to the required application. This snap power cable to DC 9V clip male line battery adapter is mostly used to power Arduino boards as well as many development boards.
Specifications:

- Connector for: 9V Battery
- Length (cm): 14
- Weight (gm): 1.5 gm

b. Communication

i. OneM2M

OneM2M is a global initiative to develop IoT standards to enable interoperable, secure, and simple-to-deploy services for the IoT ecosystem.
We use the oneM2M standards to communicate between the IIIT server setup at OM2M platform and the ESP32 or the backend server.
Our oneM2M Resource Tree is as follows:



Below is the function used to post sensor data:

```
struct SensorData
{
    int timestamp[100];
    float pos[100];
    int count;
    int target;
    bool last;
    String rn;
};

void sendSensorData(struct SensorData sensor_data_st)
{
    if(sensor_data_st.count <= 0 && sensor_data_st.last == false)
      return;

    // Format sensor data
    String sensor_data = sensor_data_st.rn + ":" +
                         (String)sensor_data_st.target;
    for(int i = 0; i < sensor_data_st.count; i++)
        sensor_data += "," + (String)sensor_data_st.timestamp[i] +
":" +
                       (String)sensor_data_st.pos[i];
    if (sensor_data_st.last == true)
      sensor_data += ",10000000:-10000000";


    http.begin(post_sensordata_url.c_str());
    http.addHeader("X-M2M-Origin", USERNAME + ":" + PASSWORD);
    http.addHeader("Content-Type", "application/json;ty=4");
```

```
    String enc = base64::encode(encrypt(sensor_data, key));
    String hashed = calc_sha256(enc) + "|" + enc;

    String ciRepresentation =
        "{\"m2m:cin\": {"
        "\"con\":\"" + hashed + "\""
        "}}";

    int httpResponseCode = http.POST(ciRepresentation);
    http.end();
}
```

ii. Wi-Fi

The ESP32 board has a built-in Wi-Fi module, and we use the 'WiFi.h' library in ESP32 to facilitate a connection between the IIIT server at OM2M platform and the ESP32 board.
The WiFi module in ESP32 board has the following features:

- 802.11 b/g/n/d/e/i/k/r (802.11n (2.4 GHz) up to 150 Mbps)
- A-MPDU and A-MSDU aggregation and 0.4 guard interval support
- WMM • TX/RX A-MPDU, RX A-MSDU Immediate Block ACK
- Defragmentation
- Automatic Beacon monitoring (hardware TSF)
- 4 × virtual Wi-Fi interfaces
- Simultaneous support for Infrastructure Station, SoftAP, and Promiscuous modes Note that when ESP32 is in Station mode, performing a scan, the SoftAP channel will be changed.
- Antenna diversity

iii. GPIO pins

To get the sensor data from the motor regarding the current angle we use interrupts as explained here.

```
int position = 0;
void setup() {
    pinMode(ENCA,INPUT);
    pinMode(ENCB,INPUT);
    attachInterrupt(digitalPinToInterrupt(ENCA),readEncoder,RISING);
```

```
}

void readEncoder(){
    int b = digitalRead(ENCB);
    if(b > 0){
        position++;
    }
    else{
        position--;
    }
}
```

To send the control signal to DC Motor we use PWM capabilities of GPIO pins. All general-purpose input output pins can be used to generate PWM except digital input pins from GPIO pins 34-39. ESP32 provides 16 PWM channels.

```
void setup(){
    analogWriteResolution(PWM, 8);
    pinMode(IN1,OUTPUT);
    pinMode(IN2,OUTPUT);
}

// Set motor to run at pwmVal in the given direction
void setMotor(int dir, int pwmVal, int pwm, int in1, int in2){
    if(dir == 1){
        digitalWrite(in1,HIGH);
        digitalWrite(in2,LOW);
    }
    else if(dir == -1){
        digitalWrite(in1,LOW);
        digitalWrite(in2,HIGH);
    }
    else{
        digitalWrite(in1,LOW);
        digitalWrite(in2,LOW);
    }
    analogWrite(pwm, pwmVal);
}
```

c. Software Specifications

We have used the following libraries in our code:

i. mbedtls/base64.h

Used for decoding base64 strings to normal ascii strings.

ii. mbedtls/md.h

Used to implement the SHA256 hashing algorithm.

iii. analogWrite.h

Provides an analogWrite polyfill for ESP32 using the LEDC functions. Used for PWM signal to DC Motor.

iv. Arduino_JSON.h

Used to process JSON received from OM2M server.

v. time.h

Provides timekeeping functionality for Arduino. It contains Date and Time functions, with provisions to synchronize to external time sources like GPS and NTP (Internet).

vi. WiFi.h

Enables network connection (local and Internet) using the ESP32 built-in WiFi.

vii. HTTPClient.h

Used to easily make HTTP GET, POST, and PUT requests to a web server.

viii. base64.h

It is used to Encode an ASCII string to base64 format.

d. Data Handling Model

The user first enters the target angle data, and optionally the PID constants, in the dashboard of the website. This data is then sent to the ‘queue_data’ container in the OneM2M server, and the ESP32 can retrieve this data from the server as soon as it’s connected to the server. Once the ESP32 receives this data, it processes it and sends the information to the motor in order to change the angle to the current target angle.
The ESP32 then sends this data to the ‘sensor_data’ container in the OneM2M server. The backend can then retrieve this information and send it to the dashboard, where the angle data is displayed in graphical form.

e. Integration Framework

We have the following components in our project:

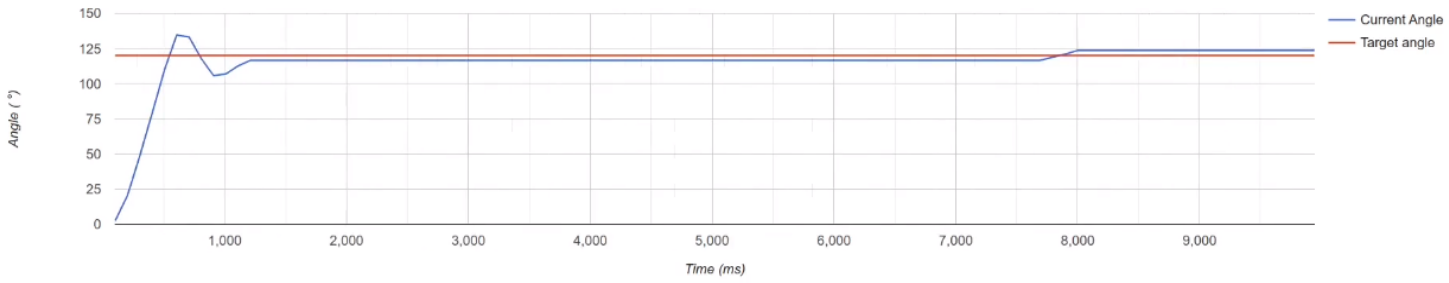
- **Frontend**
- **Backend**
- **OneM2M**
- **MongoDB database**
- **ESP32**

The communication between different entities is done through HTTPS requests.
The agreed-upon data structures used for integration are defined here.
The communication between entities is explained here.

f. Data Visualisation and Analysis Framework

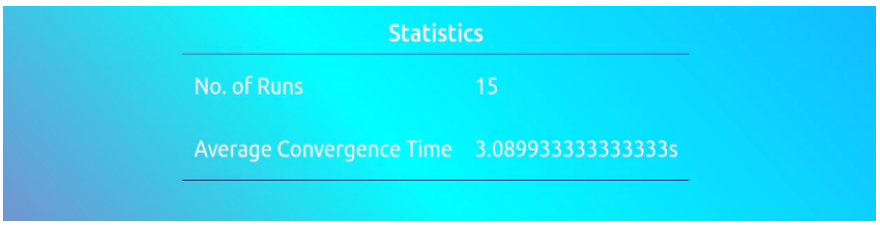
i. react-google-charts

React Google Charts offers a declarative API to easily render charts.
We use it to construct a live statistical plot of the currently ongoing trial on our website's dashboard.



ii. MongoDB database

We use this database to store the convergence time (the time it takes for the motor to get within a specific tolerant range of the target angle) of each trial.
The total number of runs along with the average convergence time is displayed on the Home page of the website.



END