PID Position Control of a DC Motor

Team - 25 (SALT)

Project Report

Team Members

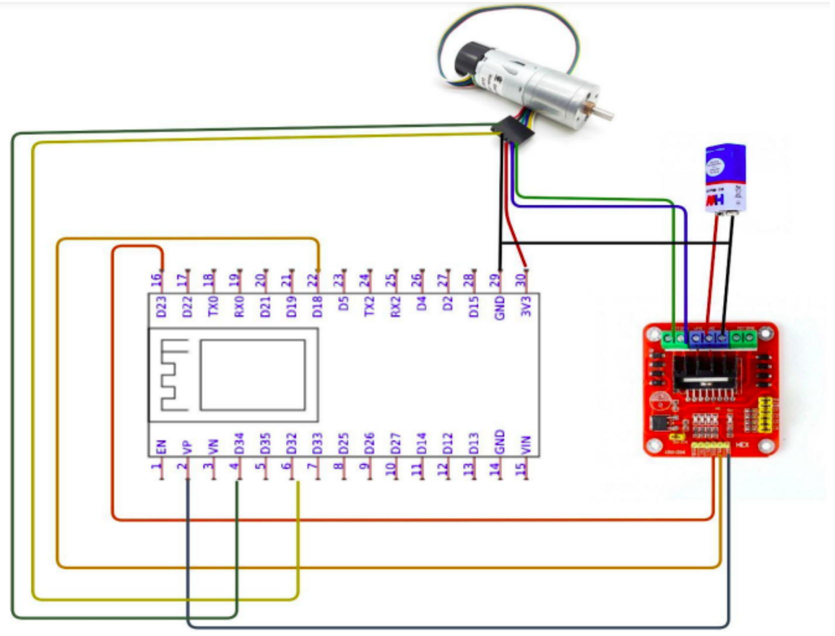
Name	**Roll no.**
Lavisha Bhambri	2020101088
Sneha Raghava Raju	2020101125
Abhijith A	2020101030
Tisha Dubey	2020101101

1. Motivation

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 experiment is to create a system consisting of a DC motor whose angular position is controlled by an ESP32 board. The system also has an encoder attached to the DC motor that can be used to find the angular position of the main shaft of the DC motor. The ESP32 sends the sensor data to the cloud. A web application visualizes the passed sensor data, the current sensor output and the live feed of the motor. It also contains an option to give commands to rotate the motor by a certain angle and change PID constants. The user can give the command and observe in real time how the result is achieved.

2. Circuit diagram & specs of the hardware



Hardware Specifications

1. Microcontroller:

   Espressif ESP32-WROOM-32D board. It is a powerful, generic Wi-Fi and BT module that targets a wide variety of applications. We use the ESP32 Wi-Fi Module to communicate with the OM2M and thingspeak server for the exchange of data. We use the following pins in the microcontroller:

   a. PIN D5: for providing Pulse Width Modulation (PWM) for the DC motor.
   b. PIN D32, D34: for reading the DC motor rotary encoder with interrupts.
   c. PIN D19, D18: for selecting the direction of the motor.
   d. PIN GND: for connecting DC motor to ground.
   e. PIN 3V3: for connecting DC motor to VCC.

2. Motor:

DC motor with encoder: spg30e-60k. This is a DC Motor with a magnetic quadrature type rotary encoder. The quadrature encoders provide two pulses which are out of phase, for sensing the direction of the shaft rotation. The hall effect sensors of the encoder operate from 3.3 V to 5 V and provide digital outputs that can be connected directly to a microcontroller or other digital circuit.

Specifications:

a. Voltage: 12V
b. No-Load Current: 70 mA
c. Output Power: 1.1 Watt
d. Rated Speed: 75 RPM
e. Rated Torque: 3 kg-cm
f. Stall Torque: 10 kg-cm

3. <u>Motor Driver:</u>

L298N Dual H Bridge 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.

Specifications:

a. Driver IC: L298N
b. Input Supply Voltage (VDC): 5 ~ 35
c. Supply Current (A): 2 (peak per channel)
d. Logic Voltage (V): 4.5 to 5.5
e. Logic Current (mA): 0 to 36
f. Control signal input voltage (V): 4.5 to 5.5
g. Maximum Power (W): 20
h. Operating Temperature (°C): -25 to 130

4. SMPS Power Adaptor: We are using an SMPS power adaptor to power the circuit.

   Specifications:

   a. Input: 110V to 240V AC 50-60 Hz
   b. Output: 12V - 2 amp

3. Methodology

Our system includes:

1. Hardware: A microcontroller running PID algorithm controlling a motor shaft. Its target angular position can be provided remotely via the dashboard and supporting statistics and analytics on the sensor readings viewable from the dashboard over the internet.
2. Platforms: Thingspeak is used to communicate from the microcontroller. Database is used to store user and slot 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 of the experiment.

To develop the system within the given timelines, we used the **Agile Methodology.** The Agile methodology is a way to manage a project by breaking it up into several phases.

We divided the project into 6 sprints:-

1. Sprint 1 - Choosing the hardware
2. Sprint 2 - Hardware Integration and PID Logic
3. Sprint 3 - Thingspeak integration
4. Sprint 4 - Dashboard Frontend and Backend
5. Sprint 5 - Security and Authorization

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
    a. 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.
    b. 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.
    c. 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 SPG30E-60K 12V Geared DC Motor with Gear Ratio of 60 and speed of 75 was a good fit.
    d. 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
    a. 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.
    b. Initially, a simple version of PID Logic was implemented and experiments were performed to fine-tune the three constants in the PID control.

    c. The PID algorithm was then further optimized using selective use of the integral term to improve accuracy and convergence times.
3. <u>Dashboard Frontend and Backend</u>
    a. The dashboard is implemented using MernStack and the deployment is done online using MongoDB Atlas.
    b. Backend - The backend for the dashboard stores the user details, and the slot booking details.
    c. Frontend - 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. It allows users to interact with the IoT system and get live feedback. It also uploads the angle and constant values to Thingspeak.
4. Thingspeak Data Upload
    a. The angle and PID constant data is uploaded to Thingspeak using a HTTP Post request.
    b. In the ESP32 code, the angle and PID constant values are retrieved from Thingspeak and used to control the circuit.
5. Security and Authorization
    a. Communication between ESP32 and Thingspeak is done.
    b. Communication between the frontend and backend is encrypted through Passport.js library and JWT token.
    c. User login/registration along with JWT was added for authorization. A MongoDB database provider Mongo Atlas was used for storing credentials.
6. Statistics and graphs
    a. Graphical analysis is done between the input angle and target angle vs time.
    b. The option of filling PID constants along with the target angle was added to the system.

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 Thingspeak servers, running the PID algorithm with these values, and then periodically sending sensor data back to Thingspeak.

Dashboard Subsystem: This is responsible for taking and storing user credentials, checking and assigning available time slots. It also sends the user entered values (angle, and PID constants) to Thingspeak.

4. Deployment campaign

Heroku is a container-based cloud Platform as a Service (PaaS). Heroku is used to deploy, manage, and scale modern apps. We used heroku to make our system elegant, flexible, and easy to use.

1. Heroku git URL - https://git.heroku.com/esw-team-25.git
2. Deployed Application URL - https://esw-team-25.herokuapp.com/home

5. Data validation

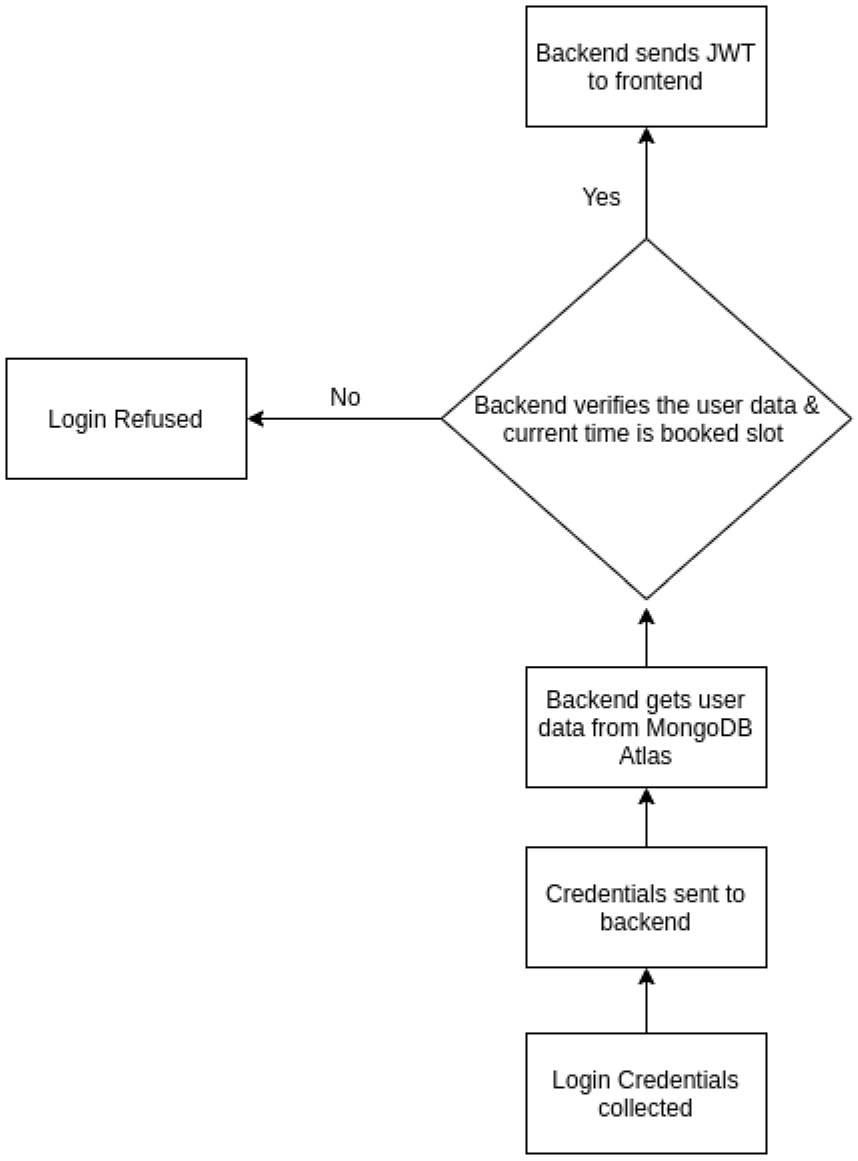
We have used graphical analysis for this. For different input values of kp, ki, kd, angle, we are plotting these graphs along with the graph of output angle by dc motor(current_angle).

The average convergence time was less than 1 second approximately.

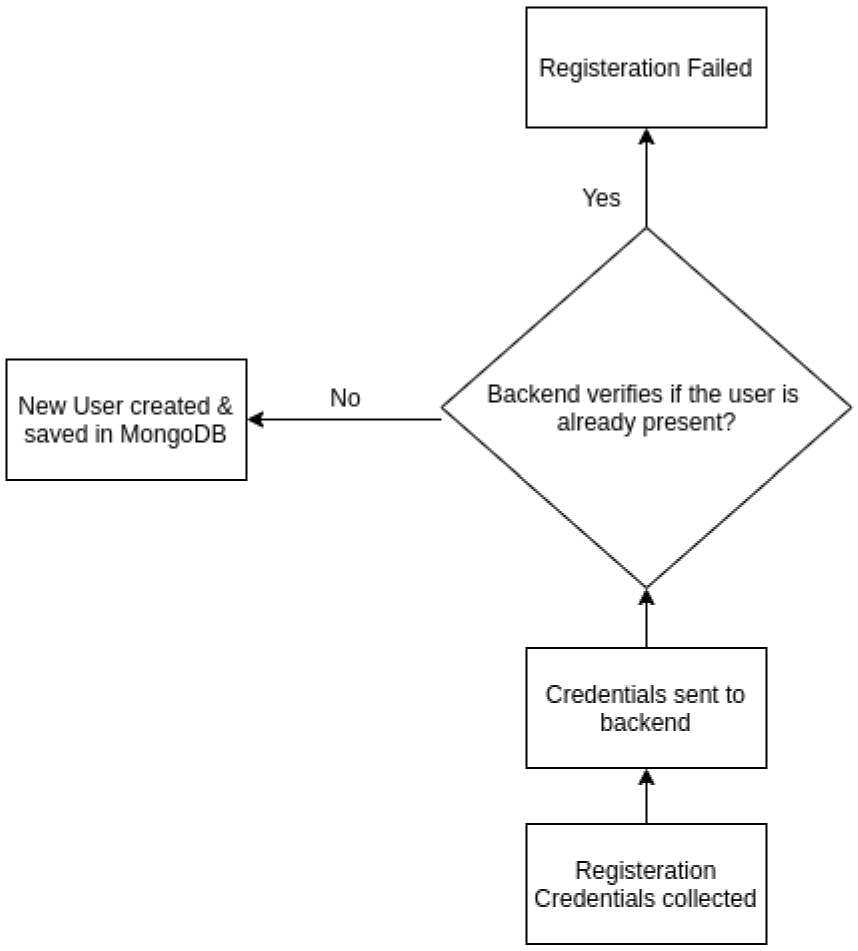
The final error was in the range ±2 degrees i.e. ±0.55%.

6.Flow diagram

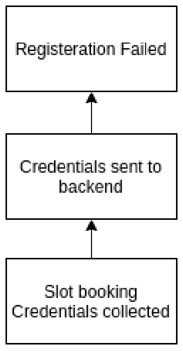
User Login


Backend sends JWT to frontend
Yes
Login Refused
No
Backend verifies the user data & current time is booked slot
Backend gets user data from MongoDB Atlas
Credentials sent to backend
Login Credentials collected

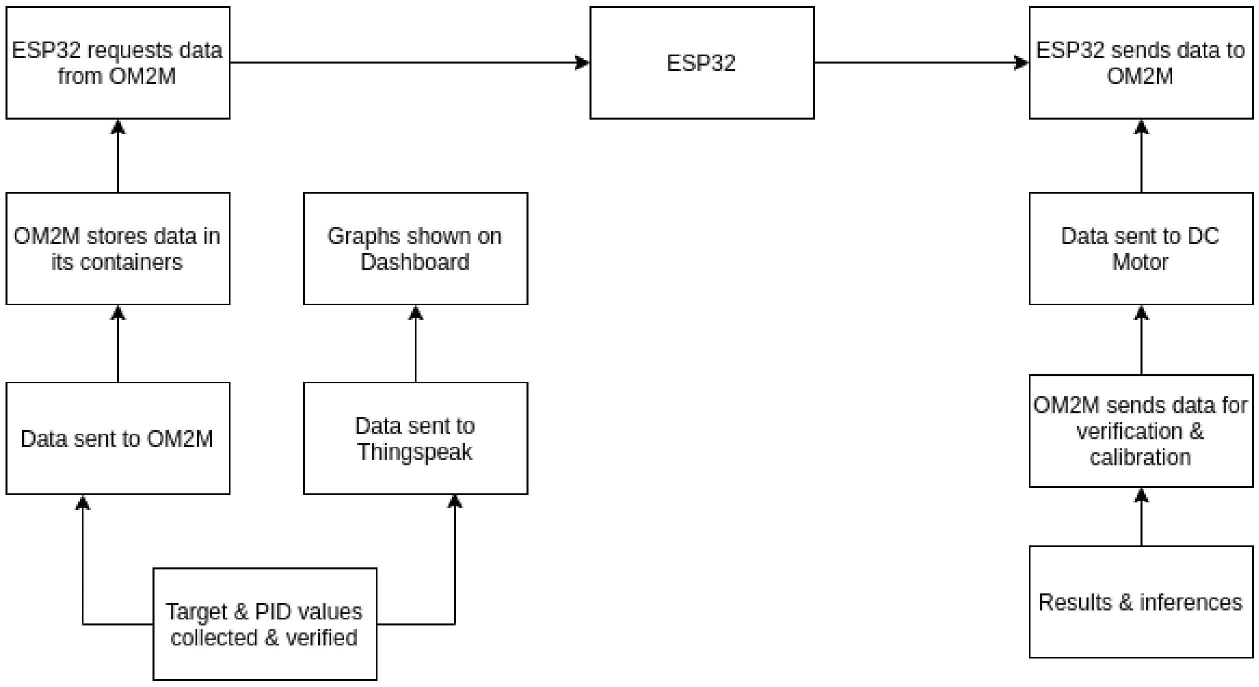
Sign Up page


Registeration Failed
Yes
Backend verifies if the user is already present?
No
New User created & saved in MongoDB
Credentials sent to backend
Registeration Credentials collected

Slot booking page


Registeration Failed
Credentials sent to backend
Slot booking Credentials collected

Input values



ESP32 requests data from OM2M
ESP32
ESP32 sends data to OM2M
OM2M stores data in its containers
Graphs shown on Dashboard
Data sent to DC Motor
Data sent to OM2M
Data sent to Thingspeak
OM2M sends data for verification & calibration
Target & PID values collected & verified
Results & inferences

7. Communication to ThingSpeak and oM2M

Thingspeak serves as an intermediary between the dashboard and ESP32. The target angle values are also pushed to the OM2M server hosted on IIIT servers.

Between ESP32 and ThingSpeak

The communication between ESP32 and Thingspeak happens via HTTP requests.

There are 2 communication channels between the components: -

1. Thingspeak to ESP32: The user data is read from Thingspeak and passed to the PID control function in the ESP32 code.
2. ESP32 to Thingspeak: The target value entered by the user is compared to the last achieved motor angle and the error value (target - current position) is pushed to ThingSpeak.
3. ESP32 to oM2M: The target value given by the user is sent to oM2M.

8.Development of dashboard

It consists of two subcomponents: -

1. Backend server: Verifies all the requests made by the frontend and fetches the data from Database and updates it.
2. Frontend: Allows users to interact with the IoT system and get live feedback.

Backend

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

1. While registering, users will have to book the slot for performing the experiment.
2. Verifying new user registrations requests frontend against if the user already exists, user is entering in the correct slot which was booked earlier and creating new users in the database (MongoDB Atlas).
3. Verifying login requests from the front end and providing tokens (JWT) for the session.
4. Providing data related to the current trial of the experiment for authorized users.
5. Forward data provided by the user (target angle and PID constants) to ESP32 (via Thingspeak).

Frontend

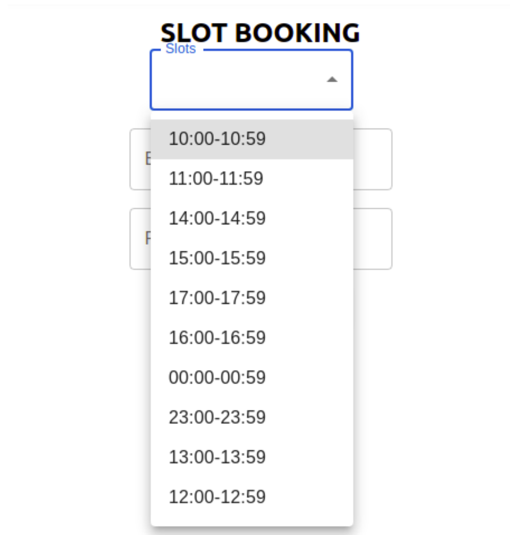
Description : The frontend is developed in React JS. It contains the following functions: -

1. User Login and Registration. Gives access to Dashboard after the user is verified from the backend.
2. Sending User details to backend upon registration of new users. - Dashboard to show the angle reading received from the ESP32 through Thingspeak.
3. Form to change the PID constants and target angle, and receive feedback.

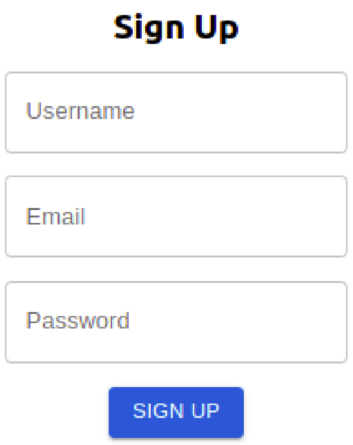
4. Live stream of DC motor using Twitch.
5. Displaying graphs and other statistics obtained from past experiments.

UI Design

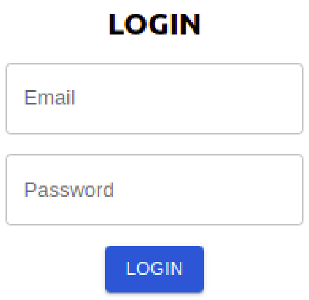
Slot booking page: -



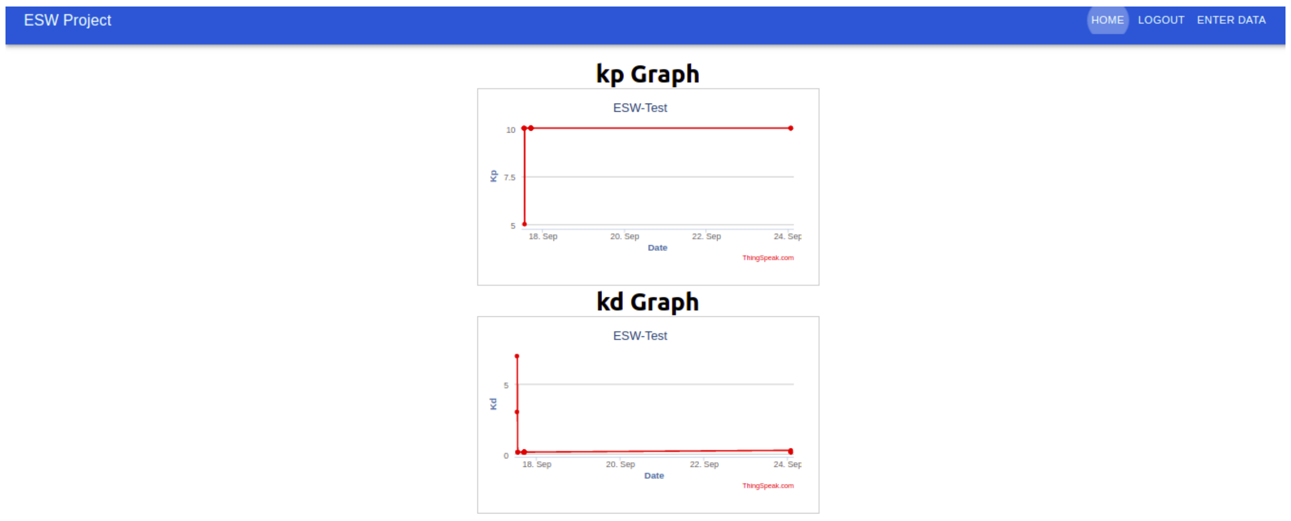
Sign up page: -



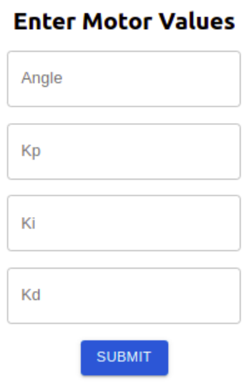
Login Page: -

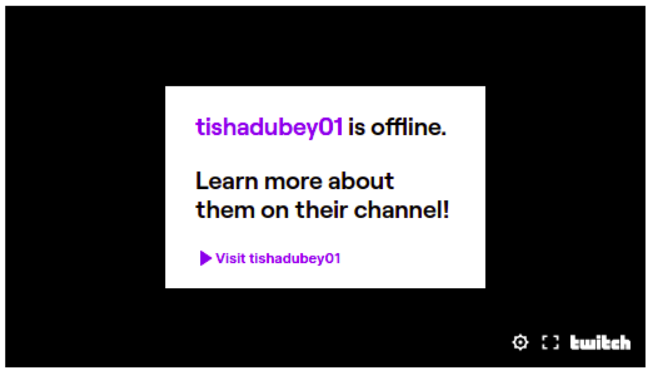


Graphical analysis: -



User Input & live broadcasting of project(shown in demo): -




tishadubey01 is offline.

Learn more about them on their channel!

9.Use of IoT security tools

We used IOT security tools while setting the communication in our project using the below tools :-

1. <u>OneM2M</u>

   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.

2. <u>Wi-Fi</u>

   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:

   1. 802.11 b/g/n/d/e/i/k/r (802.11n (2.4 GHz) up to 150 Mbps)
   2. A-MPDU and A-MSDU aggregation and 0.4 guard interval support
   3. WMM • TX/RX A-MPDU, RX A-MSDU Immediate Block ACK
   4. Defragmentation
   5. Automatic Beacon monitoring (hardware TSF)
   6. 4 × virtual Wi-Fi interfaces
   7. 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.
   8. Antenna diversity

3. <u>GPIO pins</u>

To get the sensor data from the motor regarding the current angle we use interrupts

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.

4. Software Specifications

We have used the following libraries in our code:

a. analogWrite.h - Provides an analogWrite polyfill for ESP32 using the LEDC functions. Used for PWM signal to DC Motor.
b. Arduino_JSON.h - Used to process JSON received from the OM2M server.
c. 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).
d. WiFi.h - Enables network connection (local and Internet) using the ESP32 built-in WiFi.
e. HTTPClient.h - Used to easily make HTTP GET, POST, and PUT requests to a web server.
f. base64.h - It is used to Encode an ASCII string to base64 format.

5. Data Handling Model

The user first enters the target angle data, and the PID constants, in the dashboard of the website. This data is then sent to Thingspeak, and the ESP32 can retrieve this data from it over Wi-Fi. 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 captures and sends the current angle value of the motor to ThingSpeak at 1 millisecond intervals. It also calculates the error (target angle - current angle) and uploads this value to Thingspeak, so that it can be displayed on the website. The values of angle data, kp, kd, ki, and error are displayed in graphical form on the dashboard.

6. Integration Framework

We have the following components in our project:

1. Frontend

2. Backend
3. Thingspeak
4. MongoDB database
5. 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 already explained above.

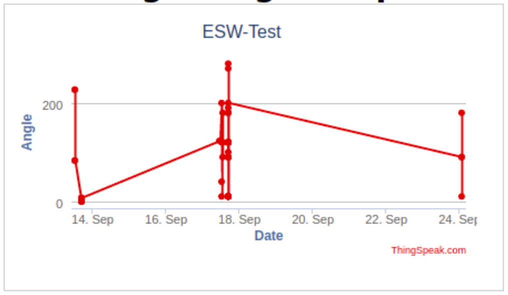
10. Data visualization and analysis

Framework

1. Thingspeak graphs

We use the thingspeak graphs to construct a live statistical plot of the currently ongoing trial on our website's dashboard. (one of the graphs is shown below)

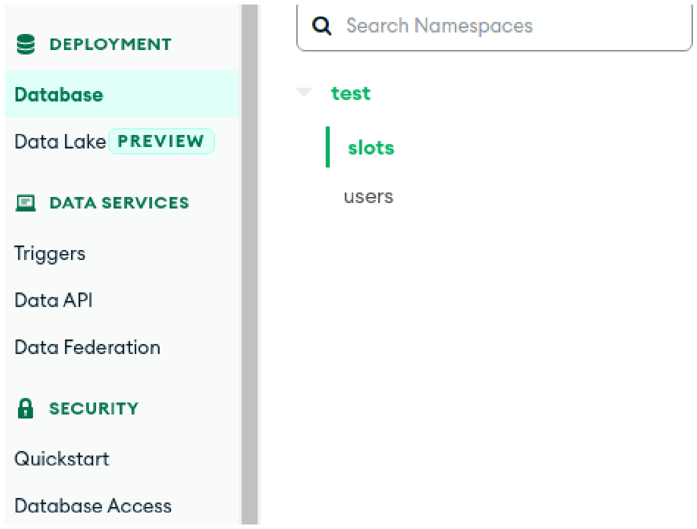
Target Angle Graph



2. MongoDB database

We use this database to store the available slots (with their start and end time) & the users who booked the particular slot.

We also stored the authenticated user details (name, password & the time filled by the user for slot booking). This time is used to verify if the user enters the correct slot.



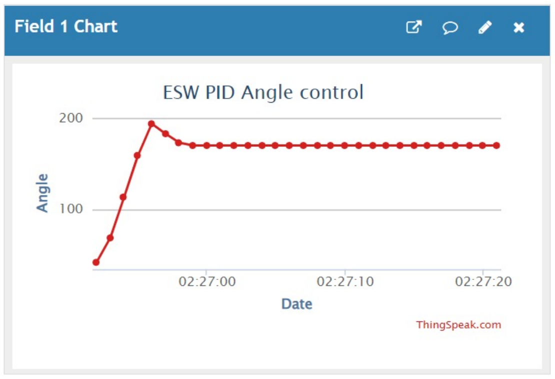
11. Final results and conclusion

The average convergence time was less than 1 second approximately.

The final error was in the range ±2 degrees i.e. ±0.55%.

The PID algorithm works by comparing the current angle to the target angle and generating the error and using the PID values to correct it. The PID constants dictate how much each part of the PID sum is represented. The Proportional constant is the most important one and the Derivative and Integral constants act as further fine tuning.

Arriving at the best PID constants was done through trial and error as shown below:

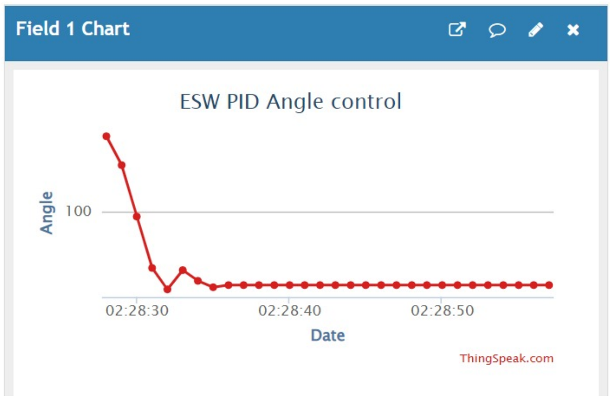


CASE - 1

Target: 180 degrees, Final Angle: 170 degrees

so Error: 10 degrees

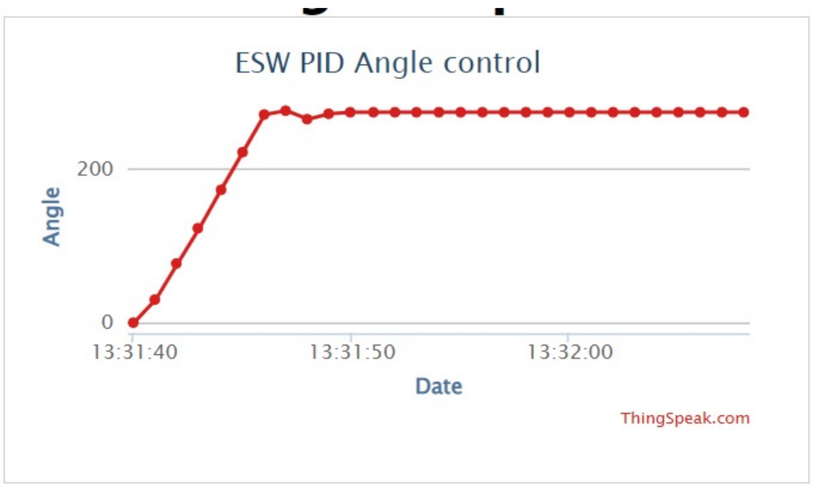
With the constants as -> kp: 10, ki: 0.4, kd: 0 , there is some overshooting of the angle.



CASE - 2

Target: 30, Reached Angle: 30, kp:20, ki:0.5, kd:0

In this case there is some damping in the angle values.



CASE - 3

Target: 270, Reached Angle: 274, Error: 4

For kp:20, ki: 0.4, kd: 0.125

For these constants the overshooting and damping are minimized and the error is small, so they are taken as the ideal constant values.

12. Final code base

1. Application Link - https://esw-team-25.herokuapp.com/home
2. GitHub Link - https://github.com/Tishadubey01/esw-pid-controller
3. Presentation Link - https://pitch.com/public/e8052033-4d3b-4aa8-a154-1bc2b894efcd