TEAM 5

PID control of a DC Motor

Hasvitha Varma, Isha Gupta, Pranjal Jain, Sharadha Iyer, Stella Sravanthi

---

Abstract:

IOT has created opportunities for people to connect devices and control them remotely, and has significantly improved performance efficiency, benefited us economically, and in general improved quality of life. This project has served to be introductory to the basics of how an IOT platform works. In this document, we begin with the developer document followed by the user document. The former is concerned with the design and technical facets of the project. In design, we highlight all the important components and concepts behind the implementation, and focus on the hardware specifications and code in the technical document.

Developer Document

1. Introduction

The role of this document is to illustrate the process from start to finish, with the documentation divided into design and technical aspects of the project

1.1 Problem Statement

We were required to control the speed of a DC motor from a remote dashboard using feedback control with arduino-controlled input voltage and an RPM sensor. This would be achieved through a PID mechanism

1.2 Purpose of the Project

This project was instrumental in helping us understand the algorithm behind a PID mechanism, and also helped gain practical knowledge on how to use different components to build a simple IOT system, and challenges that will be faced in such projects

1.3 Overview of th System

We used an ESP32 along with a Hall Sensor to control the input to the DC motor to achieve the expected speed. A dashboard operated remotely can give encrypted values for expected speed, as well as constants kP, kI, and kD for the PID mechanism to the ESP32 through a OneM2M server, and this data is decrypted and processed to achieve the desired speed. Through the process, motor RPM data is encrypted and sent to the dashboard through the server repeatedly, enabling us to plot a speed versus time graph to visualize how the RPM is attained.

2. Design Document

This document covers information pertaining to the design requirements of the entire system

2.1 System Requirements

The system must be able to detect the speed of a running DC motor, and be able to send and receive data safely through the internet so that it can be controlled through the dashboard. It must also be capable of supplying the necessary voltage to the motor for it to rotate.

2.2 System Specifications

ESP32 microcontroller is used to integrate with the L298N bridge, Hall sensor and DC motor. The microcontroller is also used to send data to the OneM2M server dynamically.

The L298 bridge controls the speed of the DC motor by giving voltage.

The Hall Sensor helps measure the speed at which the motor is rotating.

The motor and microcontroller both have an uninterrupted power supply through a rechargeable battery and laptop.

2.3 Stakeholders

The system was developed by the team members with the proper guidance of Prof. Hari Kumar and TA Sudhansh. Anyone with access to the dashboard can control the speed of the DC motor while one of our teammates has the hardware components.

2.4 Main Components

2.4.1 Microcontroller:

A single ESP32 microcontroller is used to integrate with sensors and send data to the OneM2M server through the internet.

2.4.2 OneM2M Server:

The data is collected dynamically and uploaded to the server from the microcontroller as well as the dashboard, making the motor remotely operable

2.4.3 L298N Bridge:

The H-Bridge motor driver helps control the speed of the DC motor by giving voltage.

2.4.4 Hall Sensor:

The sensor helps measure the speed at which the motor is rotating.

2.4.5 DC Motor:

This is the motor whose speed is controlled by the dashboard user

2.5 Conceptual Flow of Development:

2.5.1 Choosing sensor:

We chose A3144 hall sensor because the sensor is less difficult to position correctly than any electronic sensor and is smaller and lighter than other speed sensors. This sensor is least sensitive to vibrations and temperature fluctuations.

2.5.2 Choosing Microcontroller:

We chose ESP32 microcontroller because it was already available with us and it has all the features required for the project.

2.5.3 Choosing motor driver:

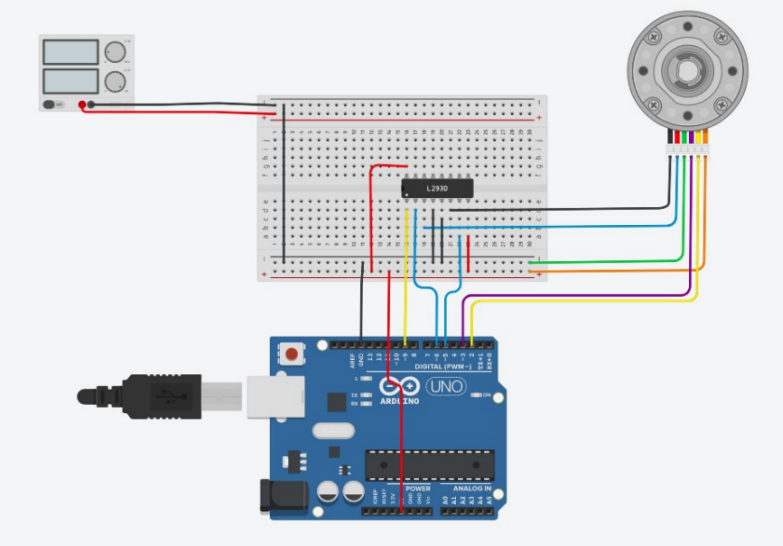
We chose the L298N H- bridge because it is very efficient in controlling the speed of the DC motor and can handle 2 motors at a time.

2.5.4 Choosing Onem2m:

We chose Onem2m because it makes the project hardware adaptable, making it more scalable for satisfying future requirements.

2.5.5 Sensor integration:

- Firstly, the code to verify all the hardware components was dumped and all the parts were verified. Then a PID control algorithm was devised on Tinkercad to verify if the algorithm was working before shifting it to hardware.



Circuit made on Tinkercad

- Later, the physical connections for the components were made.

- The hall sensor measures the RPM values of the DC motor. The L298N bridge inputs the voltage to the DC motor using the PID algorithm. In the PID algorithm the PWM gets updated based on the error of the speed and it adjusts until the desired speed is achieved.
- The code was dumped to the ESP32 Node MCU and minute changes were made to make it suitable for the hardware components.

Physical connections in the final circuit

2.6 Conceptual Flow of Node

- The initial step is to take input from the dashboard. The data is then encrypted and sent to the Onem2m server.

- After taking values from the dashboard, we need to connect to the OneM2M server. We do a POST request to send the data from the dashboard and the data is then stored in the Onem2m server.

- When data is requested by ESP32, we do a GET request and the recent data from the Onem2m server is sent to ESP32. The data received is then used to control the speed of the DC motor using the PID algorithm.

- After every measurement of the RPM values, the data is sent to the Onem2m server which is then taken by the dashboard to plot speed vs time graph.

- The dashboard is made using python script in Streamlit.

2.7 Entity Interaction

2.7.1 Microcontroller as node and Onem2m server and dashboard:

Using the createCI function the data from node is sent to onem2m server and later through a python code dashboard fetches the data from the onem2m server using the function get_data.(referred in later sections)

2.7.2 Microcontroller and Sensors:

The hall sensor interacts with the DC motor to capture the motion. The microcontroller interacts with the hall sensor to receive the detected motor speed.

2.8 Operational Requirements

2.8.1 System Needs:

The system uses oneM2M server to collect and send data. Hence, the oneM2M server and the dashboard needs to be running all the time. The hardware system requires power supply,, for this we need battery storage with backups. This power is also needed to run the motor and controller.To detect the speed of the hall sensor and motor fan should be located at close distances. Internet access to the node is necessary for data transmission.

2.8.2 UI design:

The project has Dashboard built on streamlit for user interface. The dashboard sends desired speed and parameters. The values to be sent are selected with sliders. It receives the data from onem2m containers and processes them to plot the speed vs time graph.

2.8.3 Analytical System:

The data of nodes is collected through oneM2M server. This analytics are displayed on the Dashboard and updated dynamically to observe the trend of rpm.

3. IOT Project Components

3.1 Hardware Specification

3.1.1 Microcontroller:

- A single ESP32 microcontroller is used to integrate with sensors and send data to the OneM2M server through the internet.
- The NodeMCU also uses the ESP32 WiFi module to read and transmit to the OneM2M server.

3.1.2 Sensors:

- The hall sensor works on the principle of Hall Effect, which states that whenever a magnetic field is applied in a perpendicular direction to the flow of the current, then a potential difference is induced.
- The Hall sensor is used to measure hall count which can be used to calculate RPM of a DC motor.

3.2 Communications

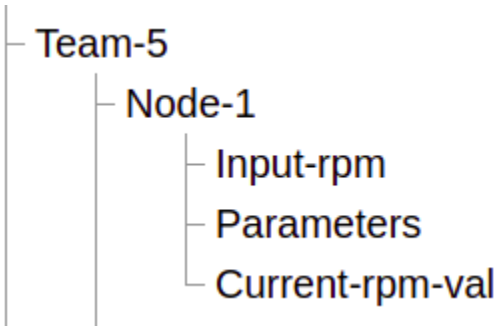
3.2.1 OneM2M:

- We have used oneM2M communication protocol which operates between server and embedded system. OneM2M server is set up by IIIT which is used for storing the data and is accessed using

POST and GET requests.
Parameters:

```
server = "https://esw-onem2m.iiit.ac.in"
cse = "/~/in-cse/in-name/"
ae = "Team-5"
```

Resource tree used on OneM2M server is as follows:



3.2.2 Server Communication:

- Server communication is used from both dashboard and esp32.
- From dashboard code, the POST request to server is used for storing input speed and parameters value and GET request is used for fetching current rpm values from server.
- From esp32 code, GET request is used for fetching input/desired speed values and input parameters values for which PID algorithms need to be performed and POST request is used for saving the current rpm values which changes along with the PID algorithm run. These are used to plot the graph on the dashboard.

POST and GET request from dashboard:

For creating the instance on server (POST Request)

```python
def create_desc_cin(uri_desc_cnt, node_description, desc_cin_labels="", data_format="json"):
    """
        Method description:
        Creates a descriptor content instance(desc_CIN) in the OneM2M framework/tree
        under the specified DESCRIPTOR CON

        This holds the detailed description for an specific AE

        Parameters:
        uri_desc_cnt : [str] URI for the parent DESCRIPTOR CON
        data_format : [str] payload format
    """

    headers = {
        'X-M2M-Origin': 'admin:admin',      # team's credentials were used
        'Content-type': 'application/{};ty=4'.format(data_format)}

    body = {
        "m2m:cin": {
            "cnf":"application/json",
            "con":node_description,
            "lbl":desc_cin_labels,
}
}


    try:
        response = requests.post(uri_desc_cnt, json=body, headers=headers)
    except TypeError:
        response = requests.post(uri_desc_cnt, data=json.dumps(body), headers=headers)
    print('Return code : {}'.format(response.status_code))
    print('Return Content : {}'.format(response.text))
```

For fetching the data from server (GET request)

```python
def get_data(uri, data_format="json"):
    """
        Method description:
        Deletes/Unregisters an application entity(AE) from the OneM2M framework/tree
        under the specified CSE

        Parameters:
        uri_cse : [str] URI of parent CSE
        ae_name : [str] name of the AE
        fmt_ex : [str] payload format
    """
    headers = {
        'X-M2M-Origin': '3S2qIgRYgA:XyRZnyl7cp',
        'Content-type': 'application/{}'.format(data_format)}

    response = requests.get(uri, headers=headers)
    print('Return code : {}'.format(response.status_code))
    print('Return Content : {}'.format(response.text))
    _resp = json.loads(response.text)
    # return response.status_code, _resp["m2m:cin"]["con"] ## To get latest or oldest content instance
    return response.status_code, _resp["m2m:cnt"]#["con"] ## to get whole data of container (all content instances)
```

POST and GET request from esp32 code:

```
String doGET(String url) {

  // Connect to the CSE address
client.setCACert(test_root_ca);

  Serial.println("\nStarting connection to server...");

  if (!client.connect(server, 443))
  {
    Serial.println("Connection failed!");
    return "error";
  }
  else {
    Serial.println("Connected to server!");
    String request = String() + "GET " + url + " HTTP/1.1\r\n" +
                     "Host: " + server + "\r\n" +
                     "X-M2M-Origin: " + CSE_M2M_ORIGIN + "\r\n" +
                     "Content-Type: application/json" + "\r\n" +
                     "Connection: close\r\n\n";
    // Make a HTTP request:
    client.println(request);

    while (client.connected()) {
      String line = client.readStringUntil('\n');
      if (line == "\r") {
        Serial.println("headers received");
        break;
      }
    }
    json_out = "";
    while (client.available()) {
       json_out = client.readString();
      Serial.println(json_out);
          json_out=json_out.substring(json_out.indexOf('{'),json_out.indexOf('}')+1);
    }
  client.stop();
  delay(5000);  //POST Data at every 20 seconds
  Serial.println();
  Serial.println("closing connection...");
  Serial.println(json_out);
  Serial.println("End of GET req");

  deserializeJson(doc, json_out);
  String vol=doc["m2m:cin"]["con"];
  Serial.println("in function vol: ");
  Serial.println(vol);
  return vol;
  }
}


String createCI(String link, String ciContent) {
  String ciRepresentation =
    "{\"m2m:cin\": {"
    "\"con\":\"" + ciContent + "\""
    "}}";
  return doPOST(link, TY_CI, ciRepresentation);
}
```

```
String doPOST(String url, int ty, String rep) {
 client.setCACert(test_root_ca);

  Serial.println("\nStarting connection to server...");

  //POST Data

  const char* origin   = "3S2qIgRYgA:XyRZnyl7cp";
  if (!client.connect(server, 443))
    Serial.println("Connection failed!");
  else {
    Serial.println("Connected to server!");
    String request = String()+ "POST " + url + " HTTP/1.1\r\n" +
                "Host: " + server + "\r\n" +
                "X-M2M-Origin:" +  CSE_M2M_ORIGIN + "\r\n" +
                "Content-Type:application/json;ty="+ ty +"\r\n" +
                "Content-Length: "+ rep.length()+"\r\n" +
                "Connection: close\r\n\n" +
                rep;
    // Make a HTTP request:
    client.println(request);

    while (client.connected()) {
      String line = client.readStringUntil('\n');
      if (line == "\r") {
        Serial.println("headers received");
        break;
      }
    }
    while (client.available()) {
      char c = client.read();
      Serial.write(c);
    }

    client.stop();
  }
    delay(5000);
}
```

3.3 Software Specifications

3.3.1 Streamlit Framework:

- Streamlit framework is used for visualizing the data/graph and for making the frontend/Dashboard. The frontend for dashboard is coded on the top of streamlit framework. User provides input values and visualizes the fetched data in the graph based on input values and PID algorithm.

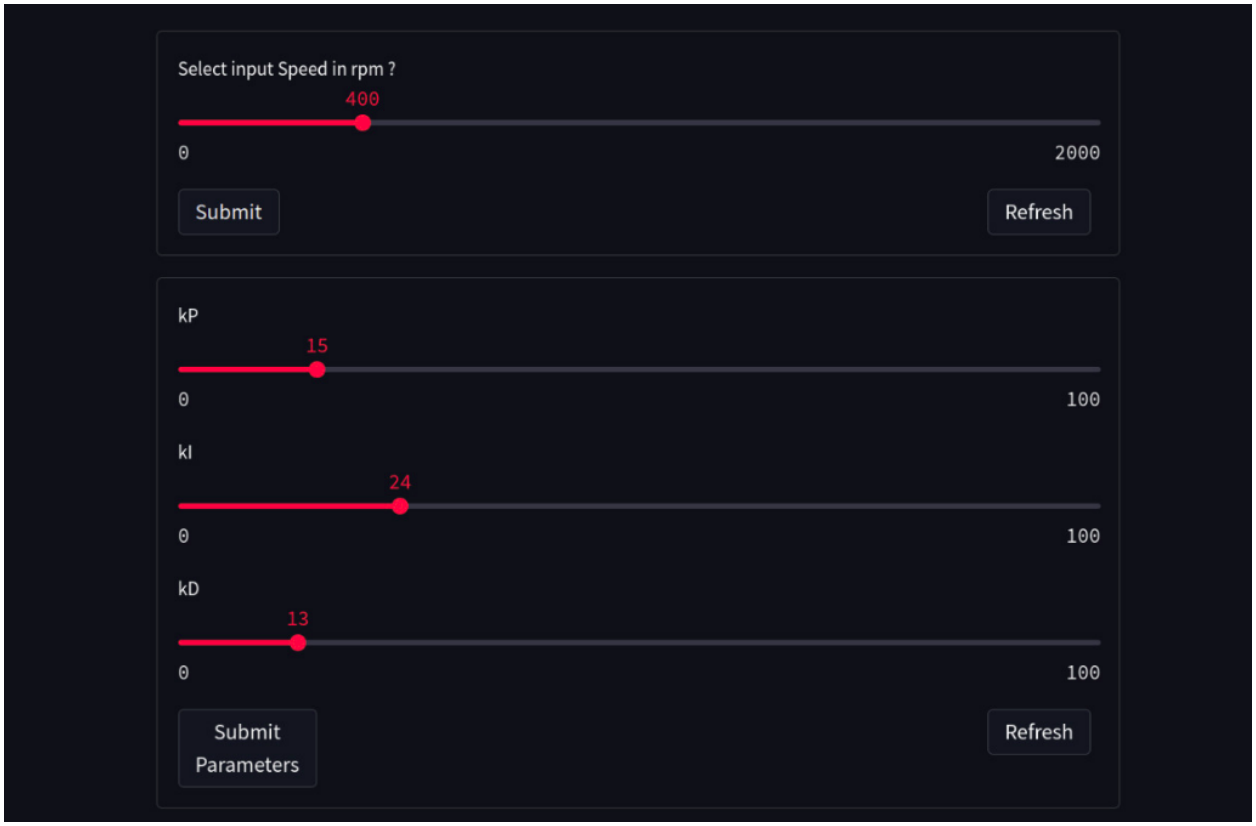
- From the streamlit API used in python code, we can write scripts and visualize. The backend APIs are called from streamlit frontend python file.

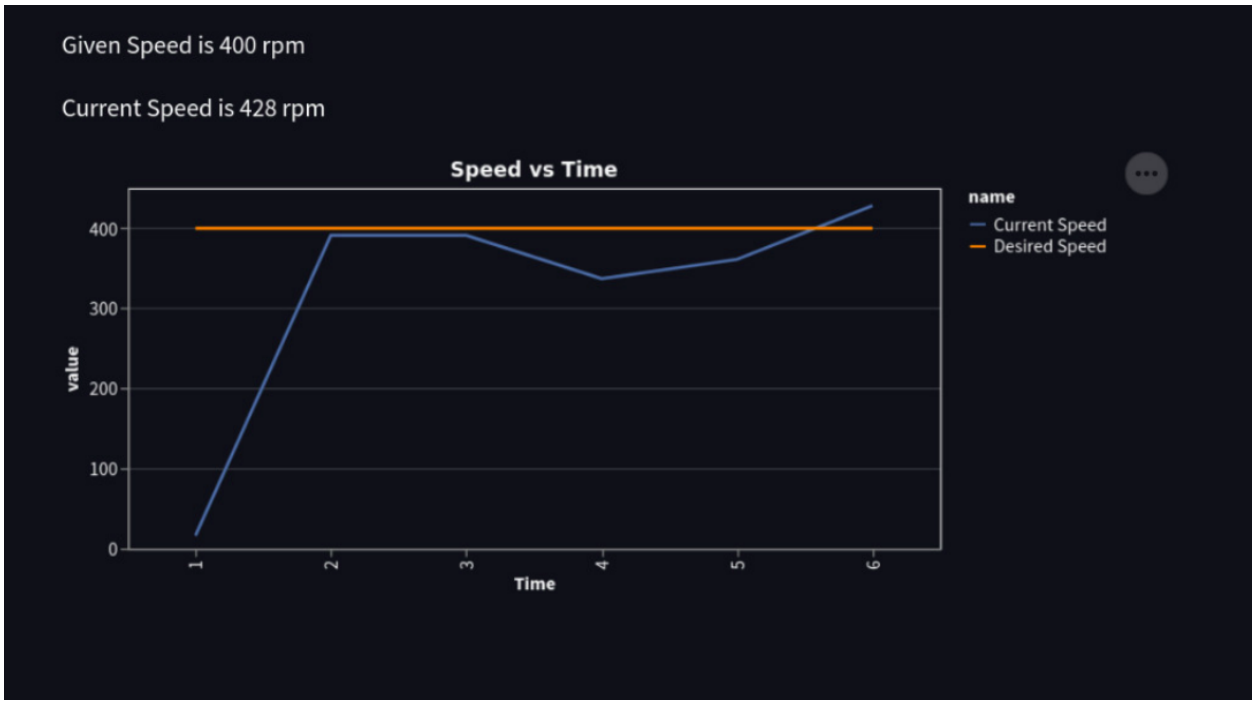
```
if submit_btn:
    st.write(f'Speed is {spd} rpm')

    spd = do_encryption(spd)
    print("xor speed",spd)
    create_data_cin(server+cse+ae+"/"+"Node-1/Input-rpm",spd,lbl_Node1)
    _,val=get_data(server+cse+ae+"/Node-1/"+"Input-rpm?rcn=4")
```

```
if submit_btn or submit_param or refresh1 or refresh2:
    given_speed = global_spd
    _,motor_speed=get_data(server+cse+ae+"/Node-1/"+"Current-rpm-val?rcn=4")
    st.write(f'Given Speed is {given_speed} rpm')
    X_ax = []
    Y_ax = []
    Y_ax1 = []
    for i in motor_speed["m2m:cin"]:
        sped,timee=i['con'].split(',')
        X_ax.append(do_decryption(timee))
        Y_ax.append(do_decryption(sped))
        print("motor speed ",do_encryption(int(sped)))
        Y_ax1.append(given_speed)
    st.write(f'Current Speed is {Y_ax[len(Y_ax)-1]} rpm')
```

3.3.2 Dashboard:

- Dashboard is created in python where frontend is created using streamlit python framework and for backend.
  We can download the created graph from the dashboard to our system.





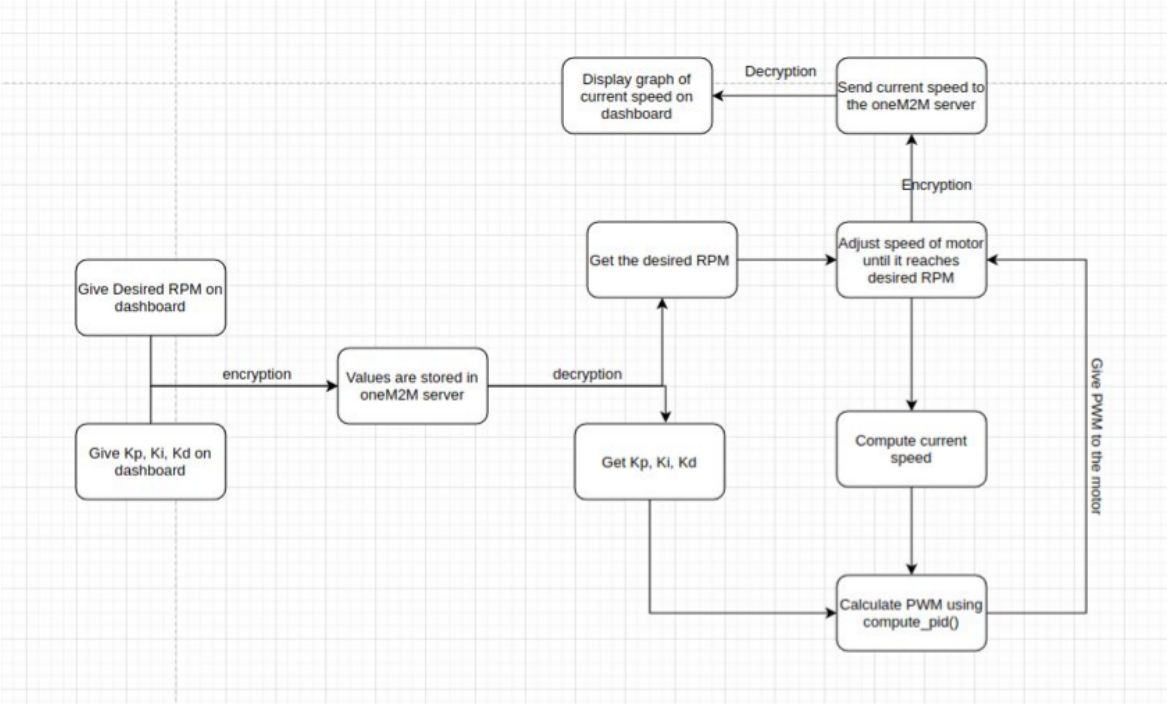
3.3.3 Libraries used :

- <WiFiClientSecure.h> Library - Used for establishing WiFi client and connecting to WiFi.

- <analogWrite.h> Library - this function is used to dump analog values(PWM values here) to the DC motor.
- pandas module - for analyzing data
- altair module - for plotting the graph in streamlit

3.4 Data Handling Model

All the data is stored on IIIT oneM2M server. Speed values are sensed using the hall sensor and error is reduced using the PID algorithm. Current Speed values for the desired speed are fetched from oneM2M server to dashboard and the graph is plotted using these values. This is the way the data is handled and visualized.



3.5 Integration Framework

The communication software components are OneM2M server and python code to analyze the data present. The data is fetched and posted on server from python script and then visualized using python libraries - pandas and altair.

3.6 Data Visualization and Analysis Framework

3.6.1 pandas :

pandas library has been used for data manipulation and analysis fetches from oneM2M server.

3.6.2 altair :

Altair is a statistical visualization library in Python. It is a quick and efficient way to visualize datasets and hence is used to plot the graph for current speed values varying over time.

3.6.3 streamlit :

Streamlit is a framework in the Python language. It helps us create web apps for data visualization and hence has been used for creating the dashboard for our IOT application.