Embedded Systems Workshop Project
PID Control of DC Motor

Team 35
Aditya Harikrish - 2020111009
Lokesh Venkatachalam - 2020111017
Nitin Rajasekar - 2020101117
Saurav Chhatani - 2020101113

Contents

1 Motivation

Feedback controls loops are widely used in several industries in order to precisely control actuators to produce the desired output. A seemingly easy problem at first glance, it shows its challenges with a deeper dive. PID controllers find use in almost all major automated processes due to low maintenance cost and high reliability. They are generally the go-to choice for applications which require constant maintenance of some fixed system parameter. Some places where PID controllers find use are to maintain flow of liquids, regulate the temperature, monitor radiation levels, and maintain pressure in pressurised chambers.

2 Block diagram/circuit diagram of the hardware

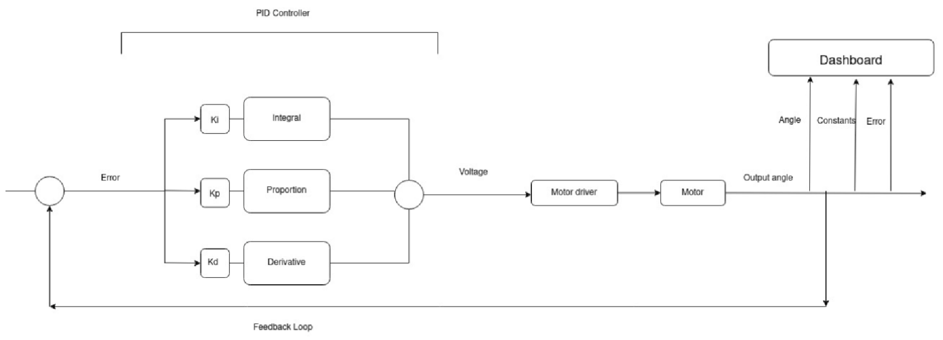


Figure 1: Workflow of the main circuitry

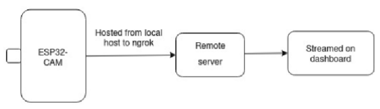


Figure 2: Workflow of the camera circuitry

3 Methodology

A proportional-integral-derivative (PID) controller is a control loop mechanism used to ensure that the output of an actuator reaches a desired value by iteratively updating the input given to the actuator. In our case, the sensor attached to the motor measures the angle by which it has turned and the actuator is the motor itself. The setpoint and program variable are the angles turned by the motor and the input given to the motor is power.

The desired value is called the setpoint (SP) and the current output of the actuator is called the program variable (PV). The error term, $e(t)$ is defined as the difference between SP and PV at the ith timestep. The input to be given to the actuator at the ith timestep is calculated as

$$u(t) = K_p e(t) + K_i \int_0^t e(x)dx + K_d \frac{de(t)}{dt}. \tag{1}$$

3.1 P Controller

$$u(t) = K_p e(t) \tag{2}$$

The proportional component is directly proportional to the error and is large when PV is far from SP. Without the integral and derivative components, $u(t)$ would become 0 when $e(t) = 0$. In the case of infinitesimally small resolutions of both, the protractor and the encoder, this alone would suffice for controlling the angle of the motor.

However, consider the example of a drone that starts from the ground level and needs to be elevated 50m. Suppose the input to the drone is the angular velocity of the fans of the drone. At the height of 50 m, $e(t) = 0 \Rightarrow K_p e(t) = 0$ and it is similarly small when $e(t)$ is small. The drone would thus never reach the setpoint of 50 m and instead hover at some height $h < 50$ m where $K_p e(t)$ balances the weight of the drone. This is called the steady state error.

3.2 PI Controller

$$u(t) = K_p e(t) + K_i \int_0^t e(x)dx \tag{3}$$

The integral component is added to bring the steady state error to 0 by accounting for the past errors. At the steady state, $e(t)$ is nonzero and thus integrating it also produces a nonzero term.

However, this causes oscillations around the setpoint due to the constant overshooting and undershooting of the setpoint. This is called the integral windup.

3.3 PID Controller

While the PI controller does reach the destination, the path it takes to do so need not be optimal due to the oscillations. The derivative component in 1 is added to predict the future respond to how fast the drone is closing in on the setpoint.

While just the P controller would suffice in the case of getting a motor to rotate by some angle (since there is no force to cause a steady state error), we have implemented a PID controller as a proof of concept for the use of PID controllers in more complex systems.

4 Deployment campaign

The oneM2M server is hosted remotely, to allow for free access to its data from the microcontroller, without having the connect both of them to the same network.

5 Data validation

The final program variables measured using the attached protractor when various setpoint angles are inputted through the dashboard are shown in fig. 3. The absolute error is measured to be at most 2°. This is due to the fact that the optical encoder attached to the motor measures a full circle in 420 steps, which implies that its resolution is $360/420° \approx 0.86°$. However, the protractor's resolution is 1°. Due to the mismatch, the error is at most 2°.

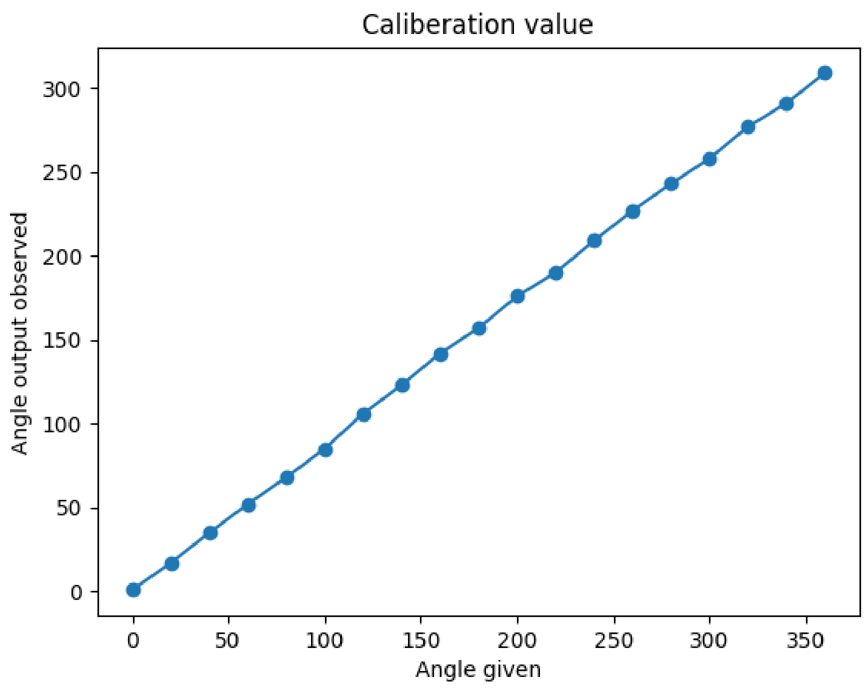


Figure 3: Home page when logged out

6 Flow diagram/chart of the entire data flow pipeline

The user's credentials are stored in the MongoDB Atlas server. After the user logs in, data from the ThingSpeak channel is fetched and displayed on the dashboard. The user can then input the setpoint angle and the ESP32 attached to the DC motor will make the latter rotate by that angle. Then, it pushes the updated data of the setpoint, program variable, error, K_p, K_i, and K_d to ThingSpeak in a single push. The dashboard then fetches the data from ThingSpeak and displays it on the dashboard.

The other data flow loop is for the camera feed, which the ESP32-CAM sends to an ngrok server, which is then sent to the dashboard.

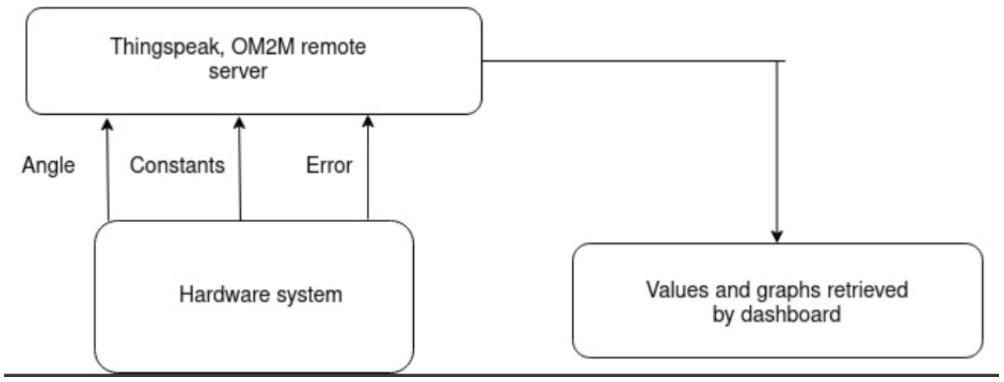

Thingspeak, OM2M remote server
Angle
Constants
Error
Hardware system
Values and graphs retrieved by dashboard

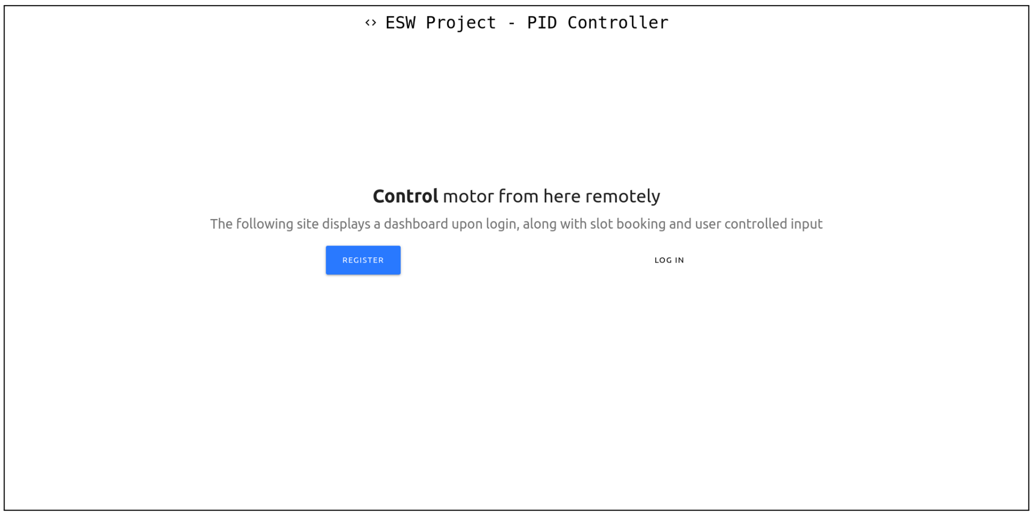
Figure 4: Home page when logged out

7 Communication to ThingSpeak and oneM2M

The microcontroller (ESP32-CAM) reads the angle turned by the motor and sends it to field1 of the ThingSpeak channel. The ThingSpeak channel is then used to send the data to the oneM2M server. The oneM2M server is then used to send the data to the dashboard.

8 Development of dashboard

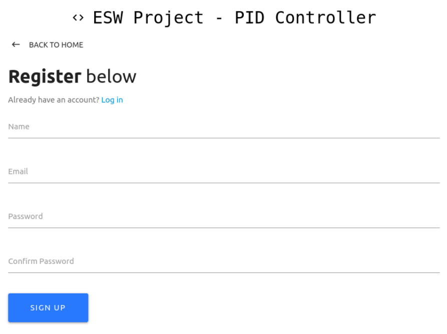
The dashboard has been developed using the MERN stack (MongoDB, ExpressJS, ReactJS, and NodeJS). The frontend is responsive, and interacts with the database in ThingSpeak. The user database is hosted on MongoDB Atlas. The dashboard is hosted remotely. The backend is robust, portable, and secure.


‹› ESW Project - PID Controller

Control motor from here remotely

The following site displays a dashboard upon login, along with slot booking and user controlled input

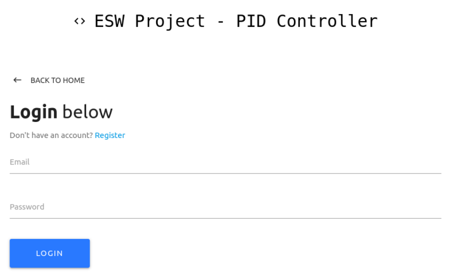
Figure 5: Home page when logged out


‹› ESW Project - PID Controller

Register below

Already have an account? Log in

Figure 6: Registration Page



‹› ESW Project - PID Controller

BACK TO HOME

Login below

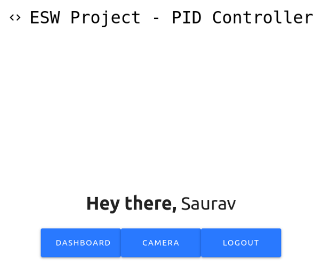
Don't have an account? Register

Email

Password

LOGIN

Figure 7: Login page

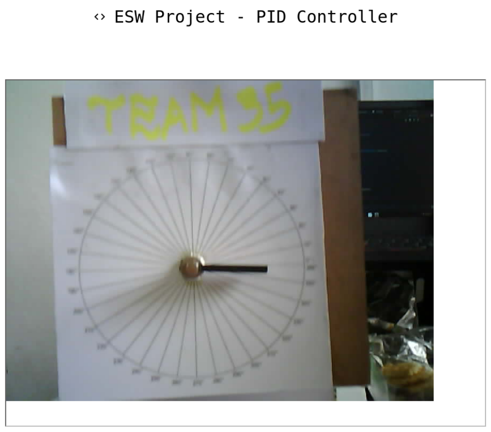


‹› ESW Project - PID Controller

Hey there, Saurav

DASHBOARD CAMERA LOGOUT

Figure 8: Login page



‹› ESW Project - PID Controller

Figure 9: Live camera feed in the dashboard

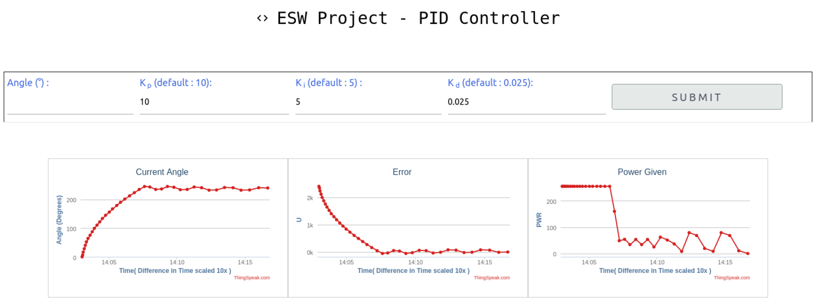
9 Use of IoT security tools in your project

The microcontroller is connected to the internet via a WiFi network. The WiFi network is secured using WPA2-PSK. The microcontroller and the dashboard are connected to the ThingSpeak channel using private read and write API keys.

10 Data visualization and analysis

Through a large number of trial-and-error experiments, we arrived at a ratio for the combination of values for K_p, K_i, and K_d that minimises the error and lets the motor reach the setpoint in the fastest possible time. We landed upon the ratios $K_p/K_d = 100$, while $K_p/K_d = 2$.

This experimental analysis is reflected in the dashboard, where the default values are in this ratio (see fig. 10).



‹› ESW Project - PID Controller

Angle (°) :	K p (default : 10):	K i (default : 5) :	K d (default : 0.025):
	10	5	0.025

Figure 10: Data visualisation in the dashboard

11 Final results and conclusion

Users can remotely register, log in, and get the motor to turn by a desired angle during their booked slots. The dashboard also allows users to view the live camera feed from the microcontroller, to change the PID controller parameters, and view live the data visualisation of the motor's angle.

12 Final code base

Please click here to go to the GitHub repository.