IOT-BASED SPEED CONTROL OF DC MOTOR USING PID CONTROLLERS AND BLYNK APP VIA ESP32

SUBMITTED BY:

MERUGA KISHAN BALAJI - 521201

KAKULAPATI CHANDRA PRAVEEN - 521145

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY ANDHRA PRADESH-534102

ANDHRA PRADESH, INDIA.

SUBMITTED TO:

DR. RAVI KUMAR JATOTH (PROFESSOR)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL-506004

TELANGANA, INDIA.

ABSTRACT

This project aims to design and implement PID controllers for the speed control of a DC shunt motor using the Blynk app on a mobile device and an ESP32 board. The objective is to develop a remote-control system that enables users to adjust the motor speed conveniently from their mobile phones.

The project begins with a comprehensive literature review, exploring existing knowledge on PID controllers, DC motor speed control, and the use of the Blynk app for remote control applications. The system architecture is designed to facilitate the interaction between the mobile device, Blynk app, ESP32 board, and the DC shunt motor. The PID controllers are designed based on proportional, integral, and derivative terms, and carefully tuned to achieve stability and optimal performance.

The implementation process includes programming the ESP32 board using the Arduino IDE, establishing communication protocols, and integrating virtual pins on the Blynk app to control the motor speed remotely. A comprehensive experimental setup is created to test the system, including connections between the ESP32 board and the DC shunt motor.

The obtained results are analyzed, considering performance metrics such as speed response, accuracy, and stability of the motor control system. The findings demonstrate the successful implementation of PID controllers for speed control, validating the feasibility of using the Blynk app and ESP32 board in an IoT-based motor control application. The project combines principles of control theory, IoT, and embedded systems to achieve an efficient and user-friendly motor control solution.

Table of Contents

1. INTRODUCTION

In today's fast-paced world, automation and remote-control systems have become an integral part of our daily lives, enhancing convenience and efficiency across various domains. The project titled “IoT-Based Speed Control of DC Shunt Motor using PID Controllers and Blynk App via ESP32 Board” brings together the power of Internet of Things (IoT) technology and advanced control algorithms to revolutionize the way we control DC shunt motors.

The application of “IoT-Based Speed Control of DC Shunt Motor using PID Controllers and Blynk App via ESP32 Board” holds significant relevance and benefits in industries such as rocks-crushing machines and situations where workers need to maintain a safe distance from the equipment.

By integrating IoT-based motor control using PID controllers and the Blynk app, operators can conveniently adjust the speed of the DC shunt motor remotely. These include the following advantages such as Safety of the worker, Efficiency. In these applications, where the workers should maintain a safe distance from the equipment, this project offers enhanced safety, efficiency, and real-time control, showcasing the significance of IoT-based solutions.

1.1 PROBLEM STATEMENT

The control of DC shunt motors in various industrial applications, particularly in rocks-crushing machines, and situations where workers need to maintain a safe distance from the equipment, poses challenges in terms of efficiency, safety, and convenience. Traditional control methods often lack the flexibility and real-time adjustments required to optimize the motor speed and ensure operator safety. There is a need for an innovative solution that combines IoT technology, PID controllers, and a user-friendly interface to enable remote speed control of DC shunt motors, enhancing safety and efficiency in these critical applications.

1.2 MOTIVATION

The increasing demand for automation and remote-control systems in today's fast-paced world has prompted a search for innovative solutions to enhance efficiency and safety in industrial processes. By utilizing Internet of Things (IoT) technology and advanced control algorithms such as PID controllers, we can address the challenges posed by controlling DC shunt motors in rocks-crushing machines and scenarios where workers must maintain a safe distance from the equipment. The motivation behind this project is to revolutionize the traditional control methods, offering a cutting-edge solution that allows operators to remotely control motor speeds using their smartphones, ensuring real-time adjustments, improved safety, and increased efficiency.

1.3 SCOPE OF THIS PROJECT

The scope of the project "IoT-Based Speed Control of DC Shunt Motor using PID Controllers and Blynk App via ESP32 Board" encompasses the design and implementation of an intelligent motor control system for DC shunt motors in rocks-crushing machines and industries with remote control needs. The project aims to leverage IoT technology to enable seamless communication between the

Blynk app on a mobile device and the ESP32 board, which serves as the interface for motor control.

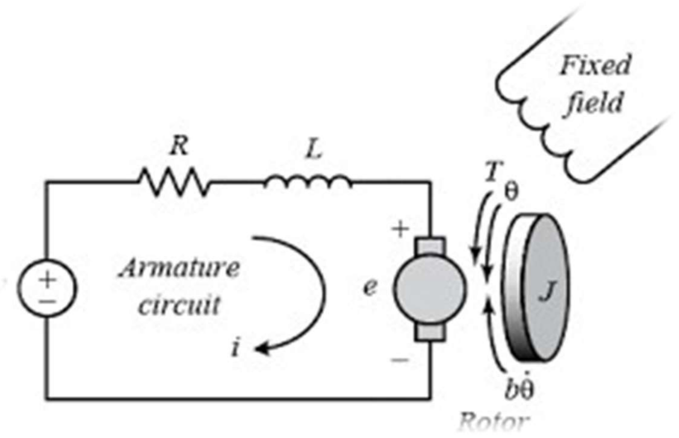
The focus of the project will be on developing and fine-tuning PID controllers to regulate the speed of the DC shunt motor accurately. Additionally, the project will involve the integration of the Blynk app, providing operators with a user-friendly interface to remotely adjust motor speeds in real-time. The system will be designed to provide feedback and performance data, allowing operators to monitor and optimize motor speed according to the application requirements.

To ensure the practicality and applicability of the solution, the project will conduct extensive experimental tests on a DC shunt motor in simulated rocks-crushing machine scenarios and environments where safe distance control is necessary. The results will be analyzed to validate the effectiveness of the IoT-based motor control system in improving safety, efficiency, and real-time control.

The project will serve as a proof-of-concept, demonstrating the potential of IoT-based solutions and PID controllers in modern engineering applications. It is anticipated that the successful implementation of this project will open avenues for further research and applications in various industries that require distance and safety considerations for motor control.

2. MATHEMATICAL MODEL OF DC MOTOR

DC shunt motors are very important sources of control systems. Mathematical models of dc shunt motors are in linear state conditions. The motor armature is designed as a circuit with armature resistance R_a connected to each other within armature inductance L_a, and there is a voltage source indicating the back emf to the motor (electromotive force) in the armature when the rotor rotates circularly with field rotation.



$I_a = $ Armature Current

$E_b(t) = $ Back emf of motor

$R_a = $ Armature Resistance

$T_L(t) = $ Load Torque

$T_m(t) = $ Motor Torque

$\theta_m(t) = $ Displacement

$L_a = $ Armature Inductance

$K_t = $ Constant Torque

$E_a(t) = $ Voltage applied

$K_b = $ Constant back emf

$\varphi = $ Magnetic flux

$\omega_m(t) = $ Angular velocity of motor

$J_m = $ Rotor Inertia

$B_m = $ Friction coefficient viscous

$$T_m(t) = K_m(t)\varphi I_a(t)$$

$$T_m(t) = K_t I_a(t)$$

Where K_i is the constant torque of motor in N-m/A.

Starting with applied input voltage $E_a(t)$ the reason and cause of the equations for the rotating dc shunt motor circuit condition.

$$\frac{dI_a(t)}{dt} = \frac{1}{L_a}E_a(t) - \frac{R_a}{L_a}I_a(t) - \frac{1}{L_a}E_b(t)$$

$$T_m(t) = K_m I_a(t)$$

$$E_b(t) = K_b\frac{d\theta_m(t)}{dt} = K_b\omega_m(t)$$

$$\frac{d^2\theta_m(t)}{dt} = \frac{1}{J_m}T_m(t) - \frac{1}{J_m}T_l(t) - \frac{B_m}{J_m}\frac{d\theta_m(t)}{dt}$$

Where $T_l(t)$ is noticed as a frictional torque for a load applied which is as coulomb friction.

$$\frac{\theta_m(S)}{E_a(S)} = \frac{K_t}{L_aJ_mS^3 + (R_aJ_m + B_mL_a)S^2 + (K_bK_t + R_aB_m)S}$$

Here, we are taking speed as output. So, we are changing displacement of motor (θ_m) into speed (ω_m).

$$\omega_m(t) = \frac{d\theta_m(t)}{dt}$$

$$\frac{\omega_m(S)}{E_a(S)} = \frac{K_t}{L_a J_m S^2 + (R_a J_m + B_m L_a)S + (K_b K_t + R_a B_m)}$$

As we are giving PWM signal here, the input should be percentage duty cycle of the PWM signal, let it be ‘x’.

If V_t = Load voltage, then

$$E_a = V_t \frac{x}{255}$$

Then the required transfer function is

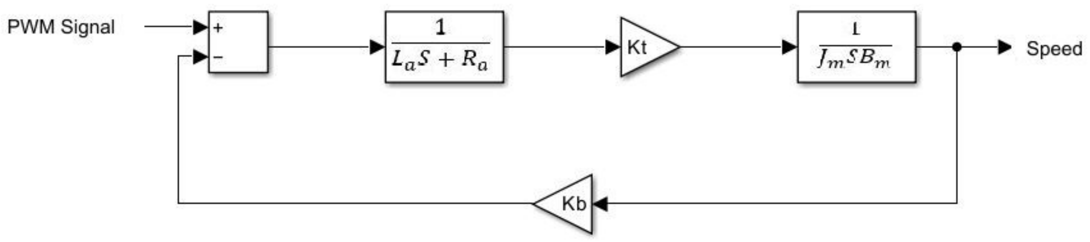
$$\frac{\omega_m(S)}{x(S)} = \frac{V_t}{255}\left(\frac{K_t}{L_a J_m S^2 + (R_a J_m + B_m L_a)S + (K_b K_t + R_a B_m)}\right)$$

Where,

$$R_a = 1.86\,\Omega$$
$$L_a = 1.1295 * 10^{-4}\,H$$
$$J_m = 8.241 * 10^{-4}\ Kg.m^2$$
$$B_m = 3.82 * 10^{-3}\ m.\frac{s}{rad}$$
$$K_t = 0.085711\,N.\frac{m}{A}$$
$$K_b = 0.085711\,V.s$$
$$V_t = 230\,V$$

$$\frac{\omega_m(S)}{x(S)} = \left(\frac{773.053}{(9.304 * 10^{-4})S^2 + (15.332)S + 144.512}\right)$$

Block diagram:



3. SPEED CONTROL OF DC MOTOR

3.1 INTRODUCTION

The dc motor converts the mechanical power into dc electrical power. One of the most important features of the dc motor is that their speed can easily be control according to the requirement by using simple methods. Such type of control is impossible in an AC motor. The concept of the speed regulation is different from the speed control. In speed regulation, the speed of the motor changes naturally whereas in dc motor the speed of the motor changes manually by the operator or by some automatic control device.
The speed is dependent upon the supply voltage V, the armature circuit resistance Ra and the field flux ϕ, which is produced by the field current. The speed of the DC Motor is given by the relation shown below.

$$N = \frac{V - I_a R_a}{k\varphi}$$

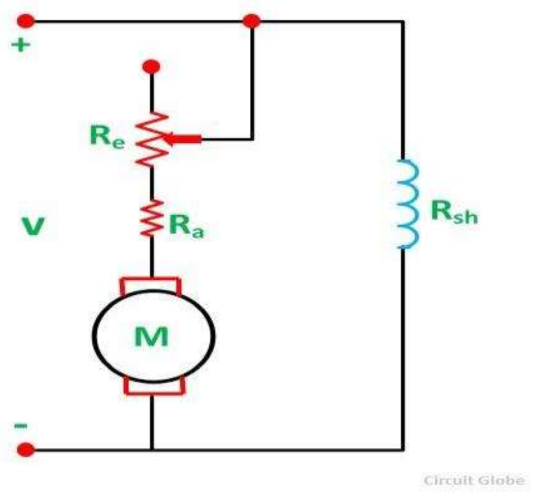
3.2 SPEED CONTROL METHODS

For controlling the speed of DC Motor, the variation in voltage, armature resistance and field flux is taken into consideration. There are three general methods of speed control of a DC Motor. They are as follows,

1. Variation of resistance in the armature circuit (Armature Resistance Control or Rheostatic Control).
2. Variation in field flux (Field Flux Control).
3. Variation in applied voltage (Armature Voltage Control).

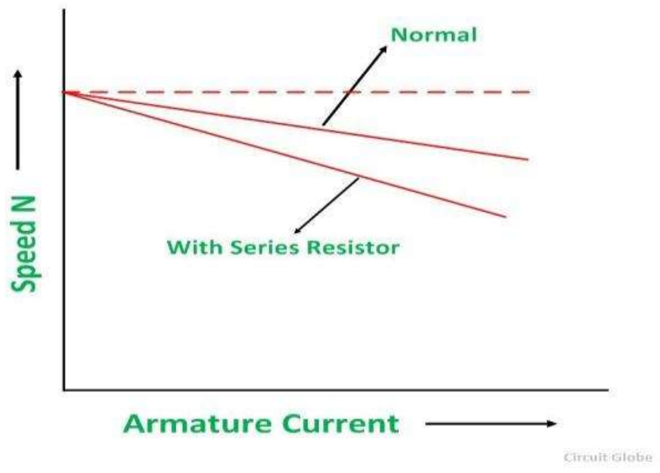
3.2.1 Armature Resistance or Rheostatic Control of Shunt Motor

In this method, a variable resistor R_e is put in the armature circuit. The variation in the variable resistance does not affect the flux as the field is directly connected to the supply mains.



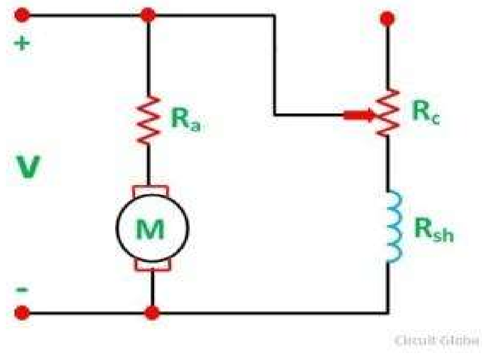
Connection diagram of a shunt motor of the armature resistance control method.

The Speed Current characteristics of Shunt Motor is,



3.2.2 Field Flux Control of Shunt Motor

Flux is produced by the field current. Thus, the speed control by this method is achieved by control of the field current. In a Shunt Motor, the variable resistor RC is connected in series with the shunt field windings as shown in the figure below. This resistor RC is known as a **Shunt Field Regulator.**

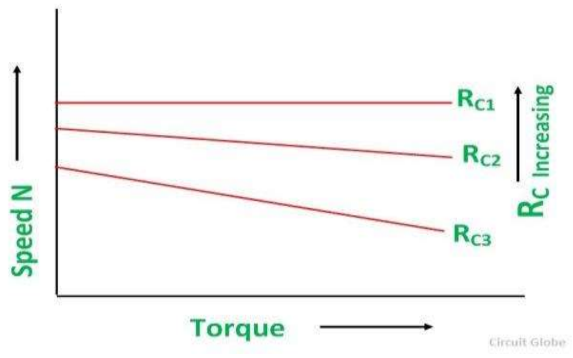


Shunt Field Regulator

The shunt field current is given by the equation shown below.

$$I_{sh} = \frac{V}{R_{sh} + R_C}$$

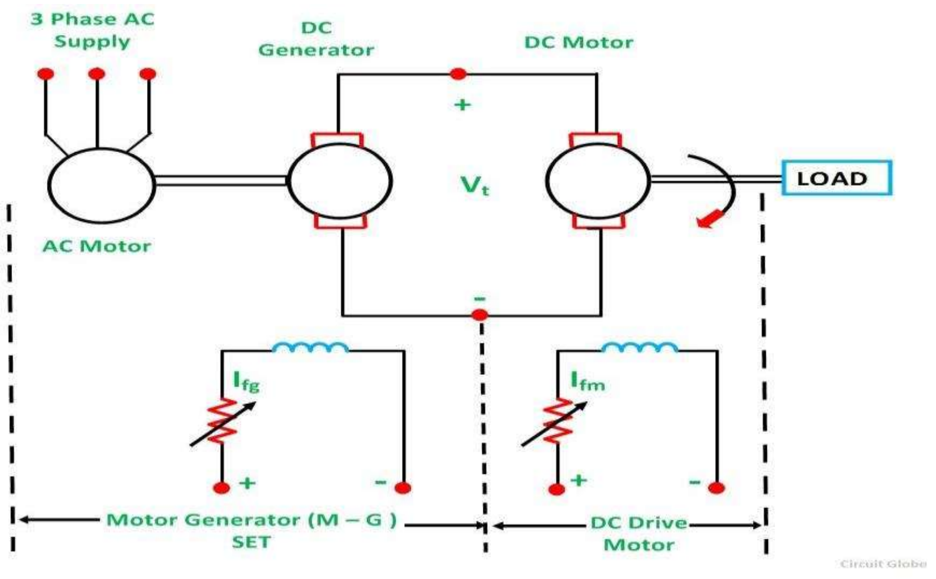
The connection of RC in the field reduces the field current, and hence the flux is also reduced. This reduction in flux increases the speed, and thus, the motor runs at a speed higher than the normal speed. Therefore, this method is used to give motor speed above normal or to correct the fall of speed because of the load.



Speed-Torque curve for a shunt motor

3.2.3 Armature Voltage Control of Shunt Motor

In the armature voltage control method, the speed control is achieved by varying the applied voltage in the armature winding of the motor. This speed control method is also known as Ward Leonard Method, which is discussed in detail under the topic Ward Leonard Method or Armature Voltage Control.

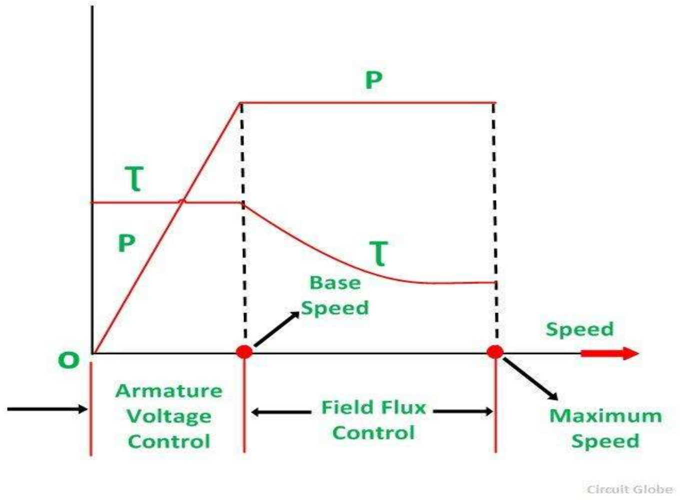


In the above system, M is the main DC motor whose speed is to be controlled, and G is a separately excited DC generator. The generator G is driven by a 3-phase driving motor which may be an induction motor or a synchronous motor. The combination of the AC driving motor and the DC generator is called the Motor-Generator (M-G) set.

The voltage of the generator is changed by changing the generator field current. This voltage when directly applied to the armature of the main DC motor, the speed of the motor M changes. The motor field current I_{fm} is kept constant so that the flux (ϕ_m) remains constant. While the speed of the motor is controlled, the motor armature current (I_a) is kept equal to its rated value.

The generated field current (I_{fg}) is varied such that the armature voltage V_t changes from zero to its rated value. The speed will change from zero to the base speed. Since the speed control is carried out with the rated current I_a and with the constant motor field flux, a constant torque is directly proportional to the armature current, and field flux up to the rated speed is obtained. The product of torque and speed is known as power, and it is proportional to speed. Thus, with the increase in power, speed increases automatically.

The Torque and Power Characteristic is shown in the figure below.



Hence, with the armature voltage control method, constant torque, and variable power drive are obtained from speed below the base speed. The Field flux control method is used when the speed is above the base speed. In this mode of operation, the armature current is maintained constant at its rated value, and the generator voltage V_t is kept constant. The motor field current is decreased and as a result, the motor field flux also decreases. This means that the field is weakened to obtain a higher speed. Since (V_tI_a) and (EI_a) remain constant, the electromagnetic torque is directly proportional to the field flux (ϕ_m) and the armature current I_a. Thus, if the field flux of the motor is decreased the torque decreases. Therefore, the torque decreases, as the speed increases.

Thus, in the field control mode, constant power and variable torque are obtained for speeds above the base speed. When speed control over a wide range is required, a combination of armature voltage control and field flux control is used. This combination permits the ratio of maximum to minimum speed available speeds to be 20 to 40. For closed-loop control, this range can be extended up to 200. The driving motor can be an induction or synchronous motor. An induction motor operates at a lagging power factor. The synchronous motor may be operated at a leading power factor by over-excitation of its field. Leading reactive power is generated by an over-excited synchronous motor. It compensates for the lagging reactive power taken by other inductive loads. Thus, the power factor is improved.

4. PID CONTROLLER

4.1 INTRODUCTION

PID (Proportional-Integral-Derivative) controllers are widely used in control systems to regulate and maintain desired performance characteristics. A PID controller operates by continuously measuring the error between a desired setpoint and the actual output of a system and adjusting the control signal accordingly.

The proportional term provides a control response proportional to the error, the integral term eliminates steady-state errors by integrating the accumulated error over time, and the derivative term accounts for the rate of change of the error, enabling anticipatory control actions.

In this project, PID controllers will be designed and implemented for the speed control of a DC shunt motor. The aim is to leverage the advantages of PID control to achieve accurate and stable speed regulation, allowing for efficient and precise control of the motor's rotational speed.

4.2 TYPES OF CONTROLLERS

Controller configuration and type must be chosen so that it will satisfy all the design specifications with the proper selection of its elements.

There are three types of controllers;

1. Proportional Controller
2. Integral Controller
3. Derivative Controller

We use the combination of these modes to control our system such that the process variable is equal to the setpoint (or as close as we can get it). These three types of controllers can be combined into new controllers:

1. Proportional and integral controllers (PI Controller)
2. Proportional and derivative controllers (PD Controller)
3. Proportional integral derivative controller (PID Controller)

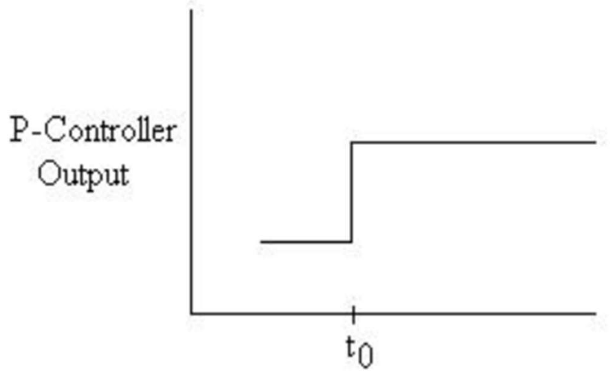
4.2.1 Proportional Controller (P)

Proportional control is a form of feedback control. It is the simplest form of continuous control that can be used in a closed-looped system. P-only control minimizes the fluctuation in the process variable, but it does not always bring the system to the desired set point. It provides a faster response than most other controllers, initially allowing the P-only controller to respond a few seconds faster. Although the P-only controller does offer the advantage of faster response time, it produces deviation from the set point. This deviation is known as the offset, and it is usually not desired in a process.

The control action for a P controller is proportional to the error,

$$G_c(s) = K_p E(s)$$

Where K_p is the proportional gain constant.



P-controller output for a step input.

4.2.2 Proportional-Integral Controller (P-I)

P-I controller produces an output which is combination of outputs of Proportional and Integral controllers. P-I controller is a form of feedback control. It provides a faster response time than I-only controller due to the addition of the proportional action. P-I control stops the system from fluctuating, and it is also able to return the system to its set point. The aim to design P-I controller is to decrease the steady state error without effecting the stability of the system.

Transfer function of the P-I Controller is,

$$G_c(s) = K_p + \frac{K_i}{s}$$

The time domain equation is given by,

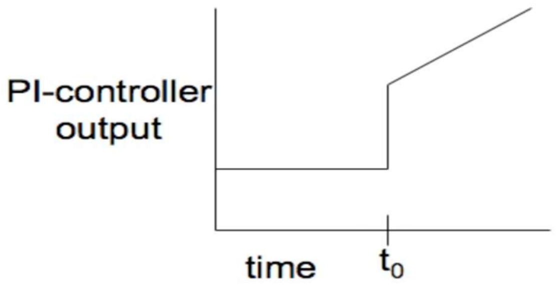
$$u(t) = K_p e(t) + K_i \int e(t)\, dt$$

By taking Laplace Transform both sides,

$$U(s) = K_p\, E(s) + K_i\, \frac{E(s)}{s}$$

Effects of P-I Controller:

- Enhanced damping ratio and reduction of maximum overshoot.
- Increase rise time and decreases BW.
- Improves gain margin, phase margin.
- Attenuates high-frequency noise.



P-I Controller output for step input.

4.2.3 Proportional-Derivative Controller (P-D)

The stability of the system will be increased with the help of P-D Controller as it has the capability of predicting the future error of the response of the process. In order to avoid the sudden disturbances, the derivative part attenuates this sudden noise signal from the error signal to maintain the system response at desired value. In this controller, the purpose of the D-only controller is to predict the error in order to increase stability of the closed loop system.

A series P-D Controller can be represented using the transfer function,

$$G_c(s) = K_p + K_d\, s$$

The time domain equation is given by,

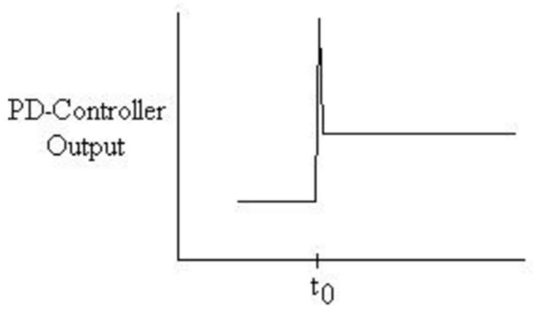
$$u(t) = K_p e(t) + K_d \frac{de(t)}{dt}$$

By taking Laplace Transform both sides,

$$U(s) = K_p E(s) + K_d s E(s)$$

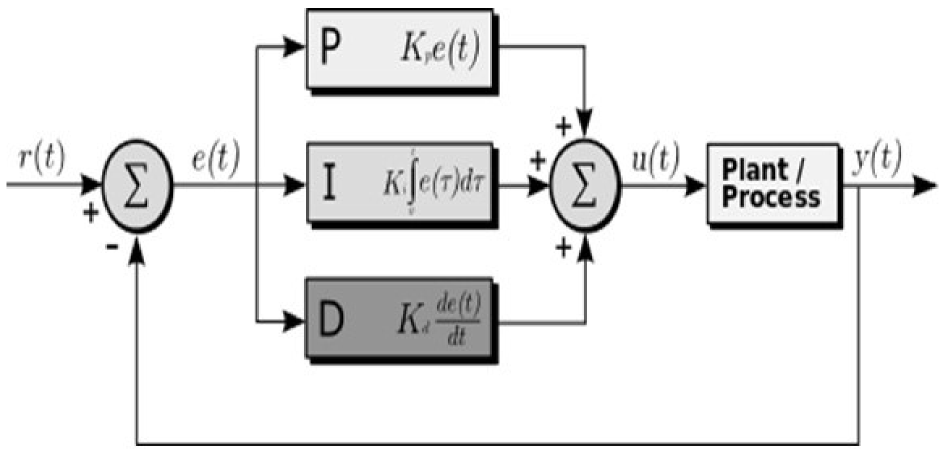
Effects of P-D Controller:

- Improves damping and reduces maximum overshoot.
- Reduce rise time and settling time but increases BW.
- Possibly prominent noise at higher frequencies since differentiation of a signal amplifies noise and thus this term in the controller is highly sensitive to noise in the error term.



P-D controller output for step input.

4.2.4 Proportional-Integral-Derivative Controller (PID)



The above figure shows the structure of the PID controller. It consists of a PID block which gives its output to the process block. Process/plant consists of final control devices like actuators, control valves, and other control devices to control various processes of industry/plant.

A feedback signal from the process plant is compared with a set point or reference signal u(t) and the corresponding error signal e(t) is fed to the PID algorithm. According to the proportional, integral, and derivative control calculations in the algorithm, the controller produces a combined response or controlled output which is applied to plant control devices.

The transfer function of a PID Controller is,

$$G_c(s) = K_p + \frac{K_i}{s} + K_d s$$

The time domain is given by,

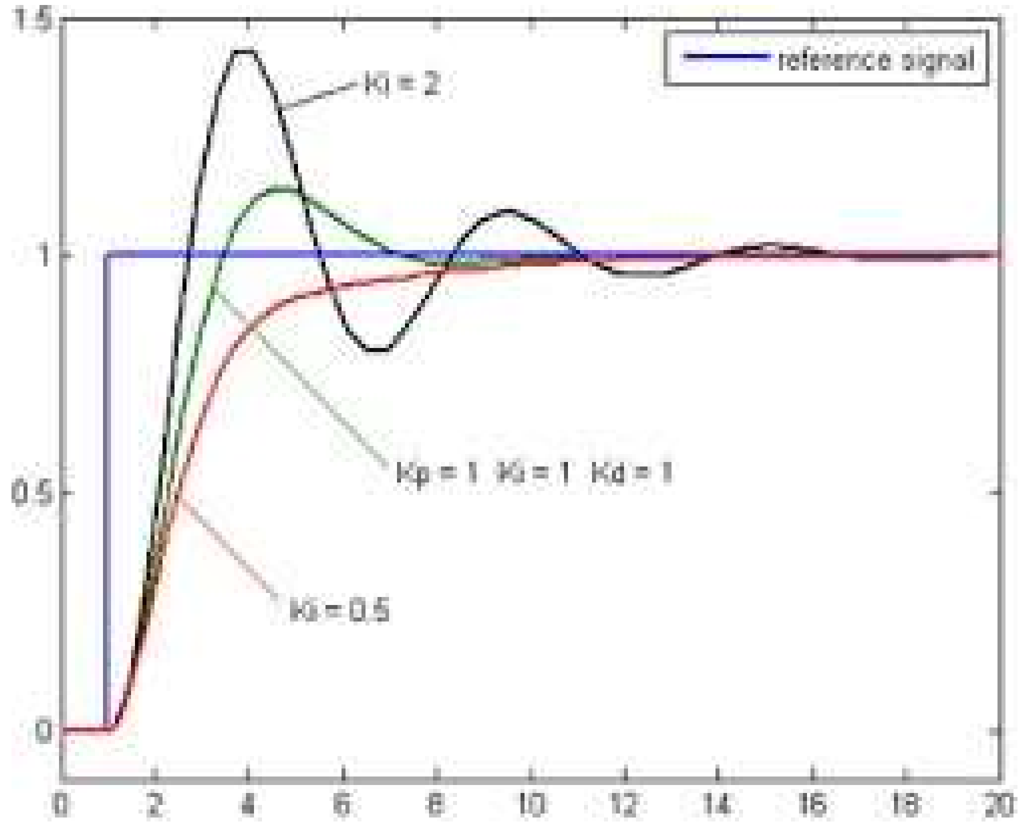
$$u(t) = K_p e(t) + K_i \int e(t)\, dt + K_d \frac{de(t)}{dt}$$

By taking Laplace Transform on both sides,

$$U(s) = K_p E(s) + \frac{K_i}{s} E(s) + K_d s E(s)$$

$$U(s) = E(s)\left(K_p s + K_i + K_d s^2\right)$$

$$\frac{U(s)}{E(s)} = \left[\frac{K_p s + K_i + K_d s^2}{s}\right]$$



PID Response

5. BLYNK IoT APP

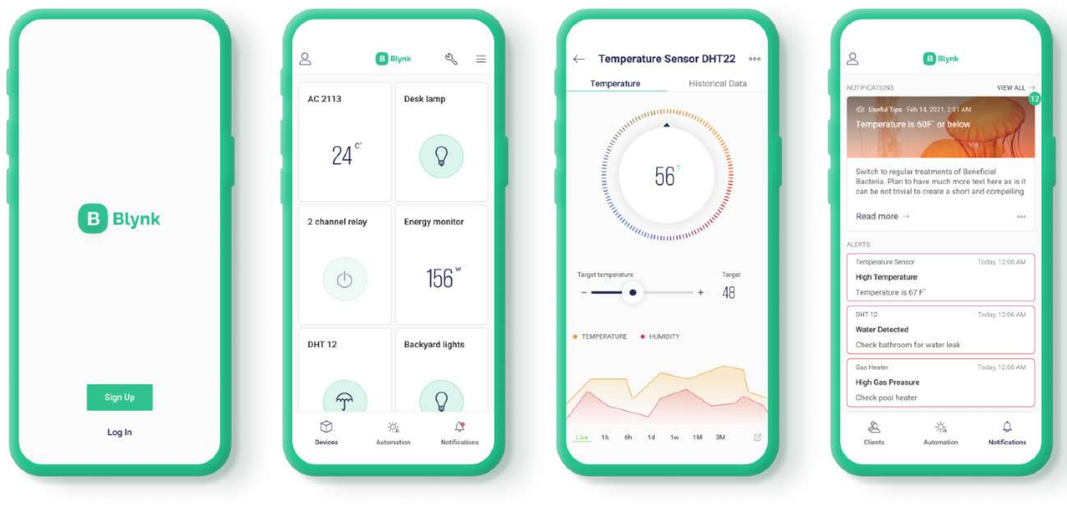
5.1 INTRODUCTION

Blynk is an Internet of Things (IoT) platform that provides a mobile app for controlling and monitoring connected devices remotely. It offers a simple and user-friendly interface, making it easy for developers, hobbyists, and professionals to build IoT applications without extensive programming knowledge. Blynk is designed to connect hardware devices, such as ESP32, Arduino, Raspberry Pi, ESP8266, and others, to the Blynk Cloud. The mobile app acts as a control panel, allowing users to interact with their connected devices.

Blynk provides various features to develop IoT applications, including:

- Widget Library: Blynk offers a wide range of pre-built widgets that can be dragged and dropped into the app's interface. These widgets include buttons, sliders, graphs, displays, and more, enabling users to create custom control interfaces for their devices.
- Data Visualization: The app allows you to visualize real-time data from your connected devices in the form of graphs, gauges, and charts. You can monitor sensor readings, track metrics, and analyze trends.
- Notifications and Alerts: Blynk supports push notifications, email alerts, and SMS notifications, allowing you to receive instant updates or trigger actions based on certain events or conditions.
- Cloud Connectivity: Blynk Cloud serves as a bridge between the mobile app and the connected devices. It provides a secure connection and handles data synchronization, ensuring seamless communication between the app and your IoT devices.
- Third-Party Integration: Blynk integrates with various popular platforms and services like IFTTT (If This, Then That), Zapier, and more. This enables you to connect your IoT projects with other web services and automate actions across different applications.
- Energy Efficiency: Blynk optimizes energy consumption by using a proprietary protocol called Blynk Protocol 1.0. It reduces data transmission and power usage, making it suitable for battery-powered IoT devices.

5.2 BLYNK PLATFORM



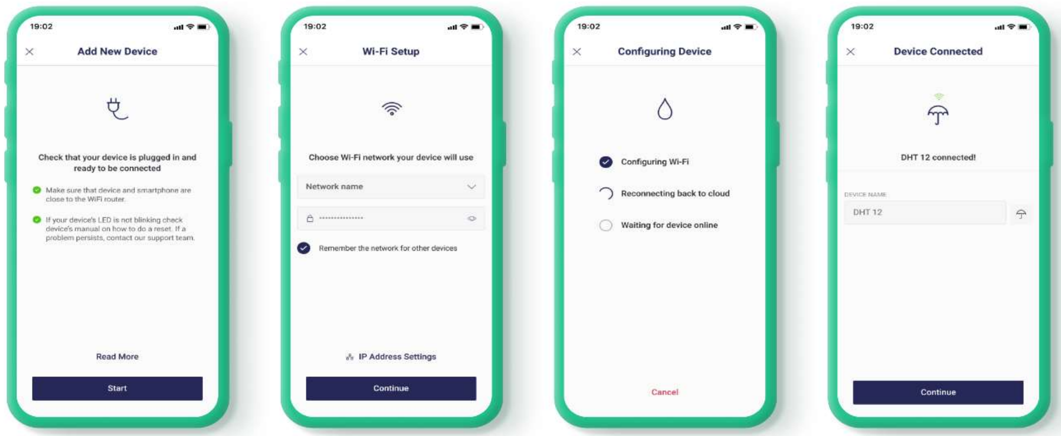
Blynk is a versatile native iOS and Android mobile application that serves these major functions:

1. Remote monitoring and control of connected devices that work with Blynk platform.
2. Configuration of mobile UI during prototyping and production stages.
3. Automation of connected device operations.

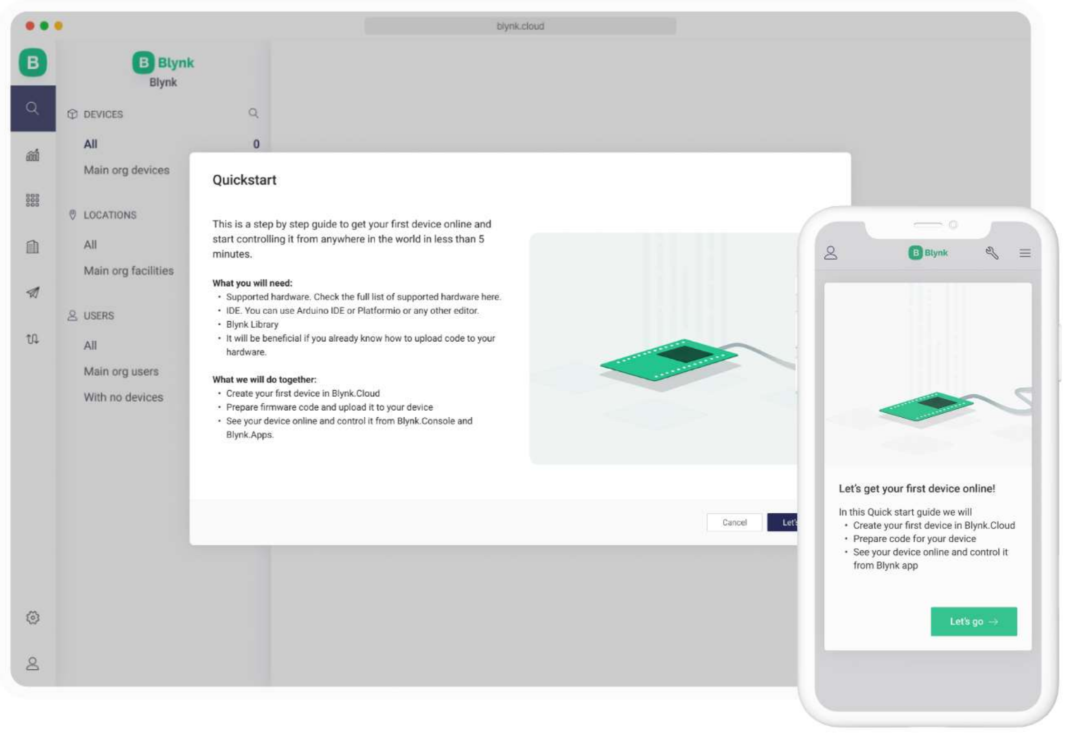
Blynk also offers a white-label solution as part of the Business Plan, allowing you to customize the app with your company logo, app icon, theme, colors, and publish it on App Store and Google Play under your company's name. These customized apps will work seamlessly with your devices.

Blynk Library is a user-friendly and portable C++ library, that comes pre-configured to work with hundreds of development boards. It implements a streaming connection protocol, allowing for low-latency and bi-directional communication.

Blynk Cloud is a server infrastructure acting as the heart of Blynk IoT platform binding all the components together.


19:02
Add New Device
Check that your device is plugged in and ready to be connected
Make sure that device and smartphone are close to the WiFi router.
If your device's LED is not blinking check device's manual on how to do a reset. If a problem persists, contact our support team.
Read More
Start
19:02
Wi-Fi Setup
Choose Wi-Fi network your device will use
Network name
Remember the network for other devices
IP Address Settings
Continue
19:02
Configuring Device
Configuring Wi-Fi
Reconnecting back to cloud
Waiting for device online
Cancel
19:02
Device Connected
DHT 12 connected!
DEVICE NAME
DHT 12
Continue

5.3 QUICKSTART



Quickstart

This is a step by step guide to get your first device online and start controlling it from anywhere in the world in less than 5 minutes.

What you will need:
- Supported hardware. Check the full list of supported hardware here.
- IDE. You can use Arduino IDE or Platformio or any other editor.
- Blynk Library
- It will be beneficial if you already know how to upload code to your hardware.

What we will do together:
- Create your first device in Blynk.Cloud
- Prepare firmware code and upload it to your device
- See your device online and control it from Blynk.Console and Blynk.Apps.

Let's get your first device online!

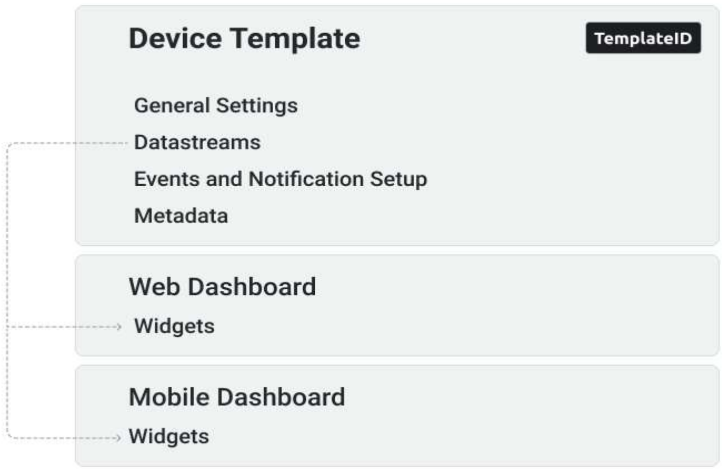
In this Quick start guide we will
- Create your first device in Blynk.Cloud
- Prepare code for your device
- See your device online and control it from Blynk app

To get started**:**

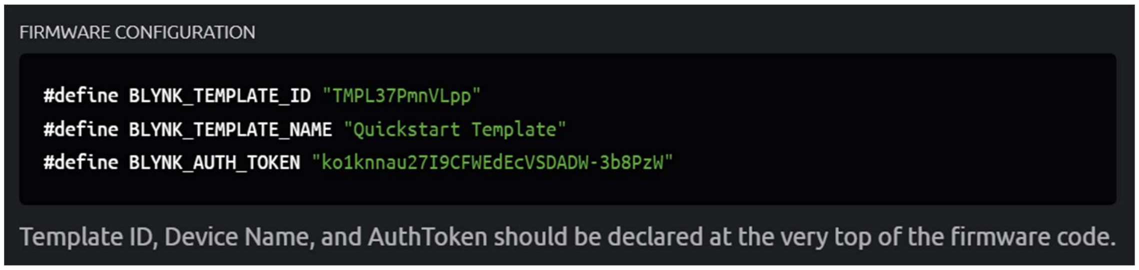
1. Create a Blynk account using Blynk.Console or Blynk.App for iOS or Android. Switch to Developer Mode in Blynk.Console or Blynk.App
2. Have a supported hardware (ESP32, Arduino, Raspberry Pi, etc). The list of supported devices is here.
3. Be familiar with the basics of electric circuitry and know how to at least blink an LED using Arduino boards, know how to install new Arduino libraries, etc.

5.3.1 Device Template

In Blynk, we use Device Templates to make it easy to work with multiple devices that perform the same functions. On a high level, Device Template is a set of configurations. Once you have created a template, you can create devices from this template and they will inherit all of the same configurations.



5.3.2 Firmware Configuration


FIRMWARE CONFIGURATION

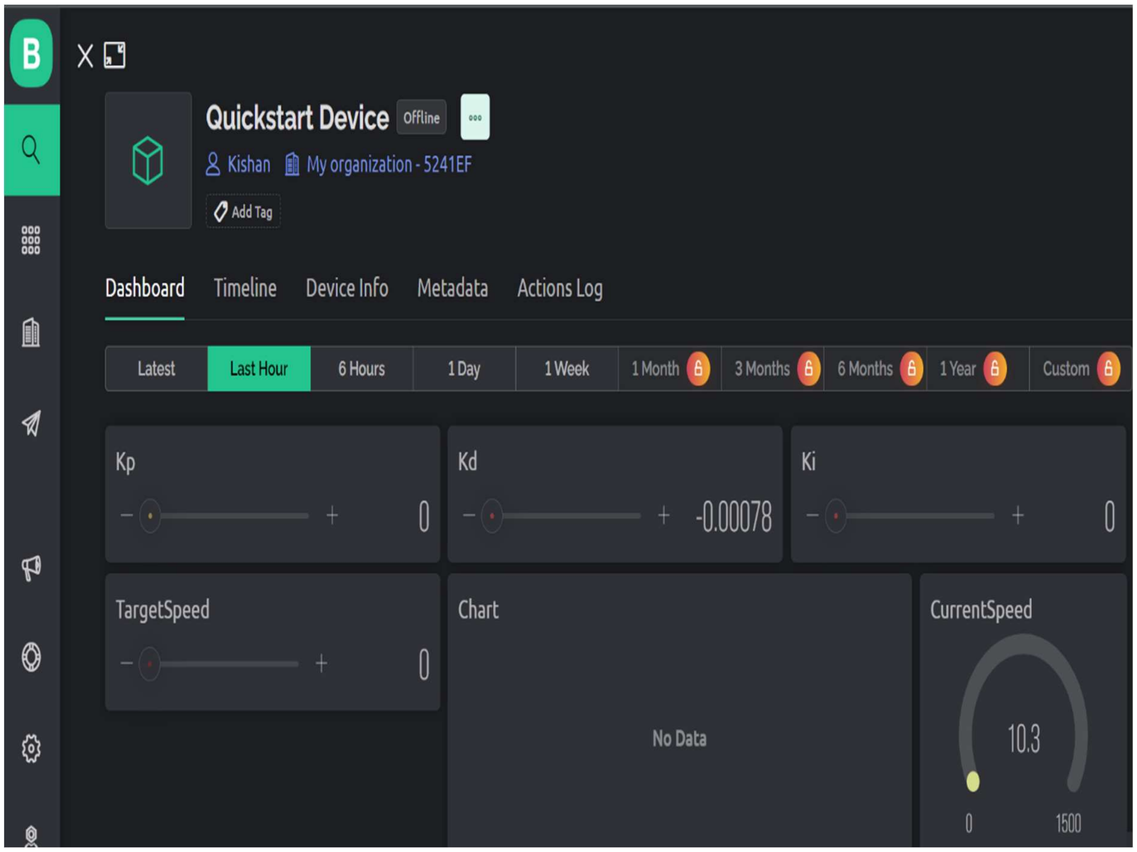
```
#define BLYNK_TEMPLATE_ID "TMPL37PmnVLpp"
#define BLYNK_TEMPLATE_NAME "Quickstart Template"
#define BLYNK_AUTH_TOKEN "ko1knnau27I9CFWEdEcVSDADW-3b8PzW"
```

Template ID, Device Name, and AuthToken should be declared at the very top of the firmware code.

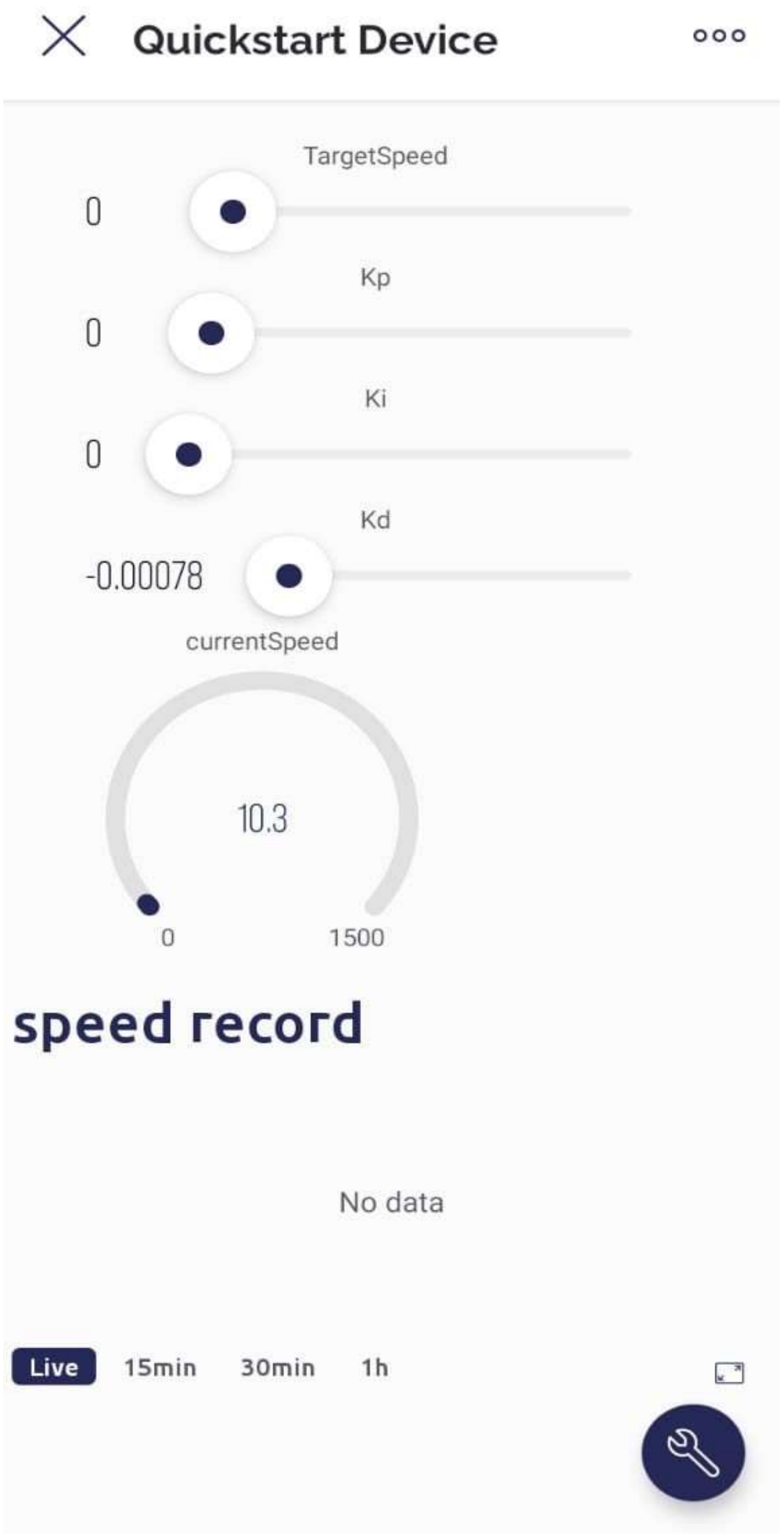
5.3.3 Web and Mobile Dashboards

The main purpose of Blynk is to make it easy to control and monitor your devices from web and mobile apps.

Dashboards are made from building blocks we call Widgets. These UI layouts are part of the Device Template. When you update the layout in the template, UI will be updated for every device.



Web Dashboard



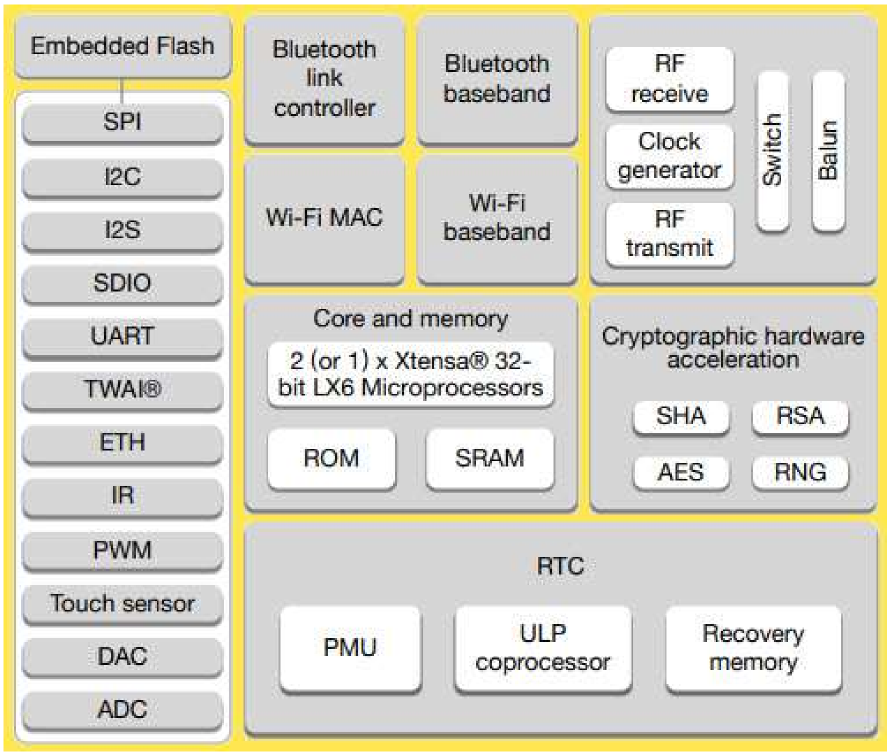
Mobile Dashboard

6. Implementation of PID Controller on ESP32

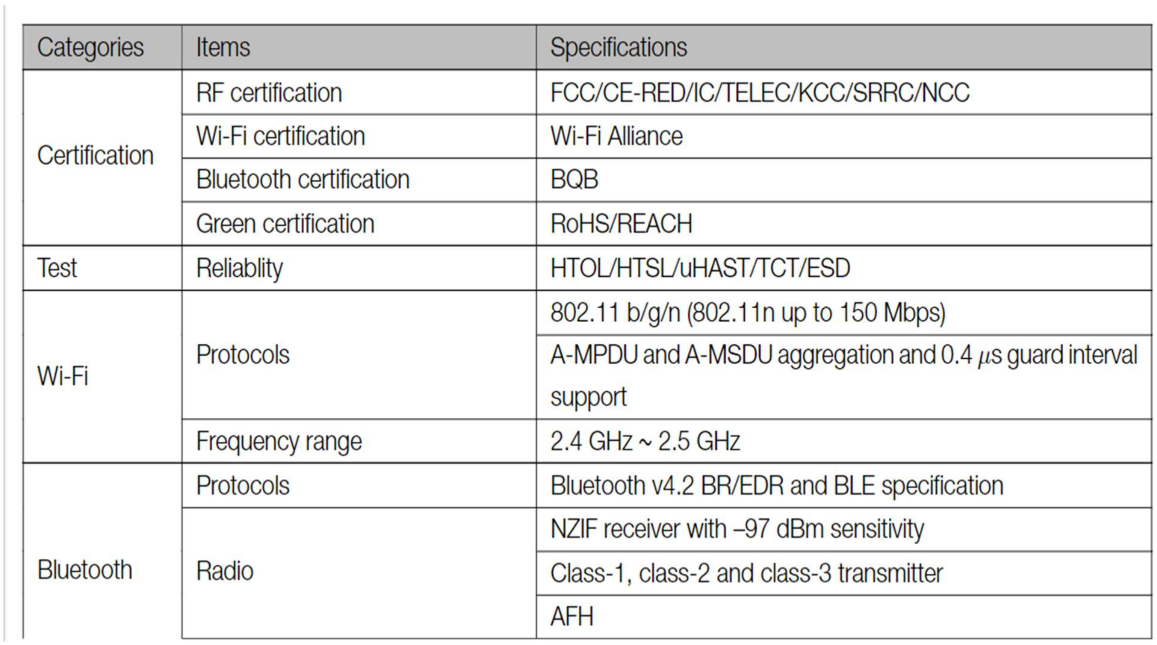
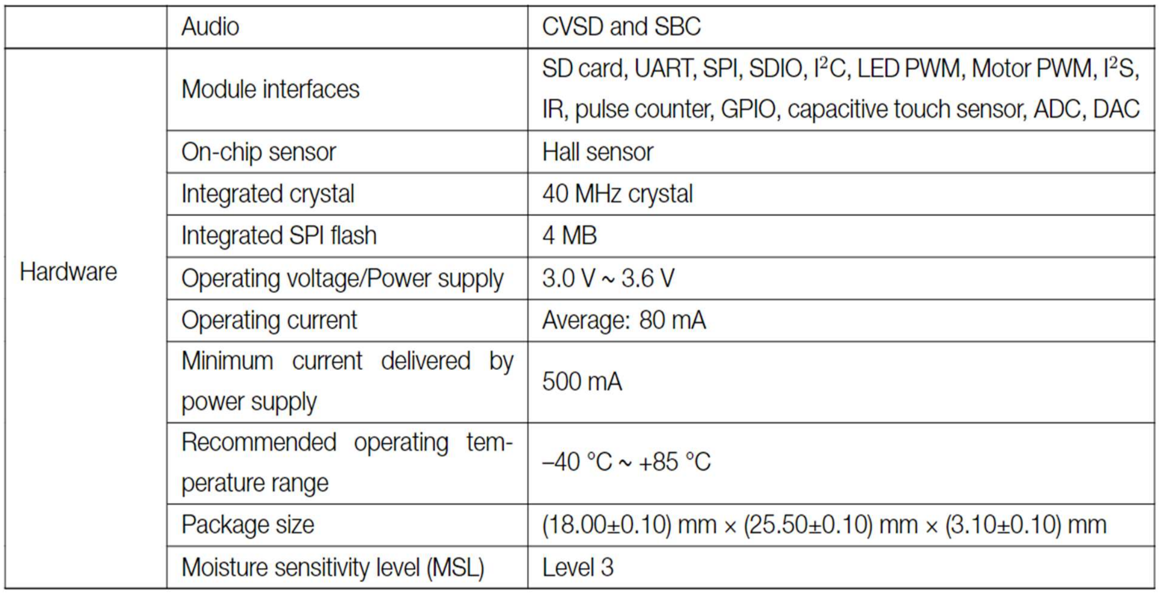
6.1 Arduino ESP-WROOM-32 DEVKIT V1

ESP32 is a low-cost System on Chip (SoC) Microcontroller from Espressif Systems, the developers of the famous ESP8266 SoC. It is a successor to ESP8266 SoC and comes in both single-core and dual-core variations of the Tensilica's 32-bit Xtensa LX6 Microprocessor with integrated Wi-Fi and Bluetooth.

The good thing about ESP32, like ESP8266 is its integrated RF components like Power Amplifier, Low-Noise Receive Amplifier, Antenna Switch, Filters and RF Balun. This makes designing hardware around ESP32 very easy as you require very few external components.

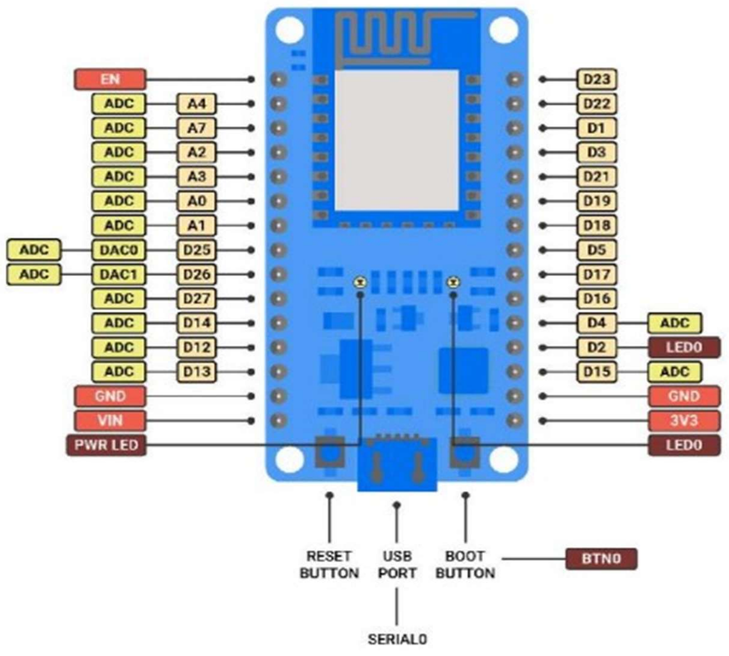


6.1.1 Specifications of ESP32

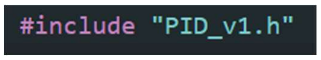
Categories	Items	Specifications
Certification	RF certification	FCC/CE-RED/IC/TELEC/KCC/SRRC/NCC
	Wi-Fi certification	Wi-Fi Alliance
	Bluetooth certification	BQB
	Green certification	RoHS/REACH
Test	Reliablity	HTOL/HTSL/uHAST/TCT/ESD
Wi-Fi	Protocols	802.11 b/g/n (802.11n up to 150 Mbps)
		A-MPDU and A-MSDU aggregation and 0.4 μs guard interval support
	Frequency range	2.4 GHz ~ 2.5 GHz
Bluetooth	Protocols	Bluetooth v4.2 BR/EDR and BLE specification
	Radio	NZIF receiver with –97 dBm sensitivity
		Class-1, class-2 and class-3 transmitter
		AFH
	Audio	CVSD and SBC
Hardware	Module interfaces	SD card, UART, SPI, SDIO, I^2C, LED PWM, Motor PWM, I^2S, IR, pulse counter, GPIO, capacitive touch sensor, ADC, DAC
	On-chip sensor	Hall sensor
	Integrated crystal	40 MHz crystal
	Integrated SPI flash	4 MB
	Operating voltage/Power supply	3.0 V ~ 3.6 V
	Operating current	Average: 80 mA
	Minimum current delivered by power supply	500 mA
	Recommended operating temperature range	–40 °C ~ +85 °C
	Package size	(18.00±0.10) mm × (25.50±0.10) mm × (3.10±0.10) mm
	Moisture sensitivity level (MSL)	Level 3

6.1.2 PINOUT



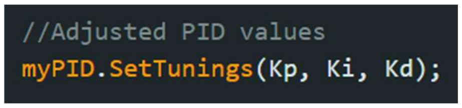
6.2 Controller Implementation on ESP32

PID instance in Arduino program is a "if" which runs continuously to update the gain parameters, sampling time, set point, process variable, manipulated variable, integral error, error, last error in the header file which is shown below.



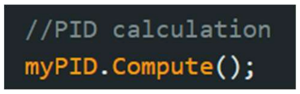
```
#include "PID_v1.h"
```

For the computation of process output, PID gain variables are needed to multiply with sample time. That is the tuning of PID gain variables is done by a command as shown below.



```
//Adjusted PID values
myPID.SetTunings(Kp, Ki, Kd);
```

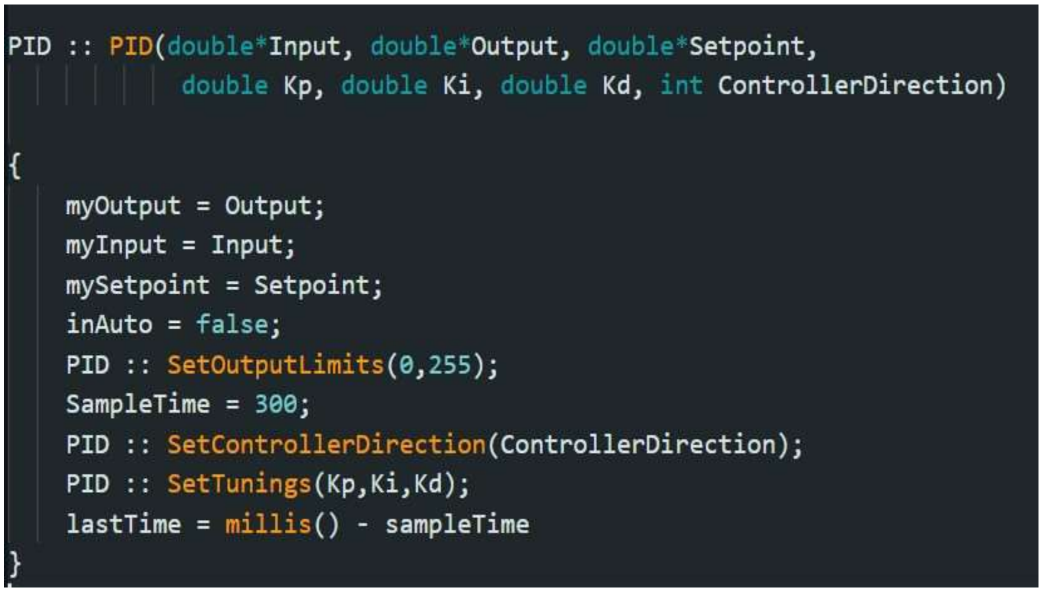
The error is the difference of setpoint and process variable is calculated in the "PID compute" section in the header file. Also, the calculation of Process variable based on all the PID gain parameters, integral error, error, sampling time is done with the command shown below.



```
//PID calculation
myPID.Compute();
```

6.2.1 Constructor

The parameters specified here are those for which we can't set up reliable defaults, so we need to have the user set them.

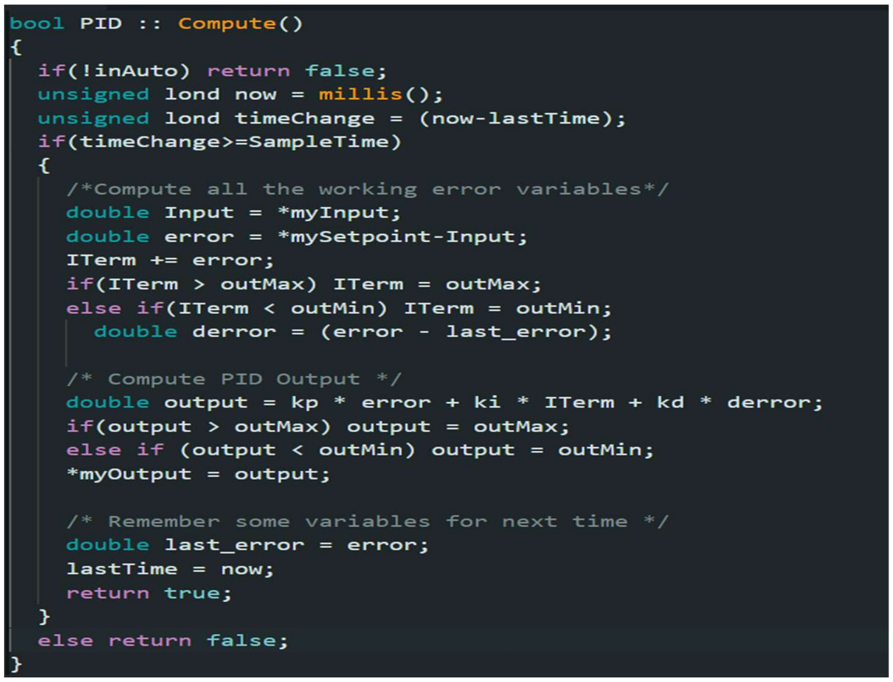


```
PID :: PID(double*Input, double*Output, double*Setpoint,
            double Kp, double Ki, double Kd, int ControllerDirection)

{
    myOutput = Output;
    myInput = Input;
    mySetpoint = Setpoint;
    inAuto = false;
    PID :: SetOutputLimits(0,255);
    SampleTime = 300;
    PID :: SetControllerDirection(ControllerDirection);
    PID :: SetTunings(Kp,Ki,Kd);
    lastTime = millis() - sampleTime
}
```

Compute( )

This function should be called every time "void loop( )" executes.

This function will decide for itself whether a new pid Output needs to be computed. Returns True when the output is computed, False when noting has been done.

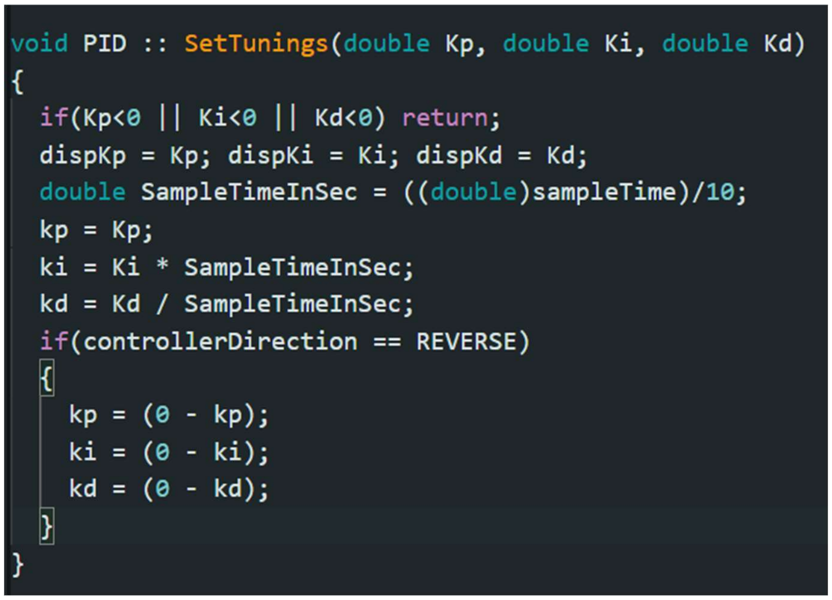


```
bool PID :: Compute()
{
  if(!inAuto) return false;
  unsigned lond now = millis();
  unsigned lond timeChange = (now-lastTime);
  if(timeChange>=SampleTime)
  {
    /*Compute all the working error variables*/
    double Input = *myInput;
    double error = *mySetpoint-Input;
    ITerm += error;
    if(ITerm > outMax) ITerm = outMax;
    else if(ITerm < outMin) ITerm = outMin;
      double derror = (error - last_error);

    /* Compute PID Output */
    double output = kp * error + ki * ITerm + kd * derror;
    if(output > outMax) output = outMax;
    else if (output < outMin) output = outMin;
    *myOutput = output;

    /* Remember some variables for next time */
    double last_error = error;
    lastTime = now;
    return true;
  }
  else return false;
}
```

SetTunings( )

This function allows the controller’s dynamic performance to be adjusted.

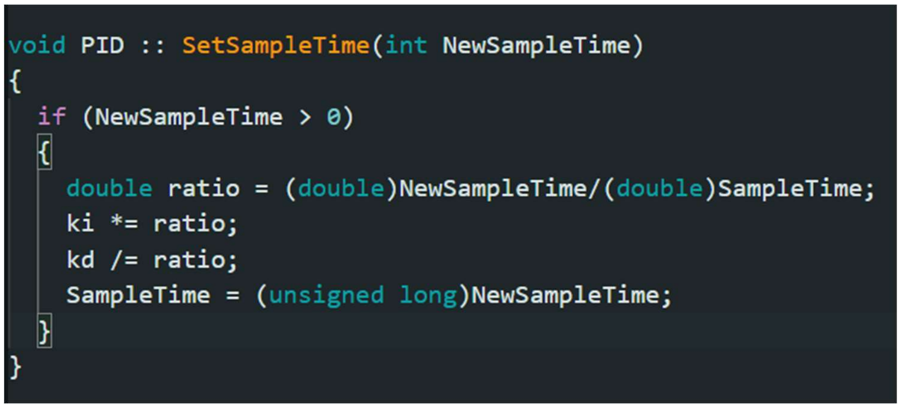
It’s called automatically from the constructor, but tunings can also be adjusted during normal operation.



```
void PID :: SetTunings(double Kp, double Ki, double Kd)
{
  if(Kp<0 || Ki<0 || Kd<0) return;
  dispKp = Kp; dispKi = Ki; dispKd = Kd;
  double SampleTimeInSec = ((double)sampleTime)/10;
  kp = Kp;
  ki = Ki * SampleTimeInSec;
  kd = Kd / SampleTimeInSec;
  if(controllerDirection == REVERSE)
  {
    kp = (0 - kp);
    ki = (0 - ki);
    kd = (0 - kd);
  }
}
```

SetSampleTime( )

Sets the period, in Milliseconds, at which the calculation is performed.



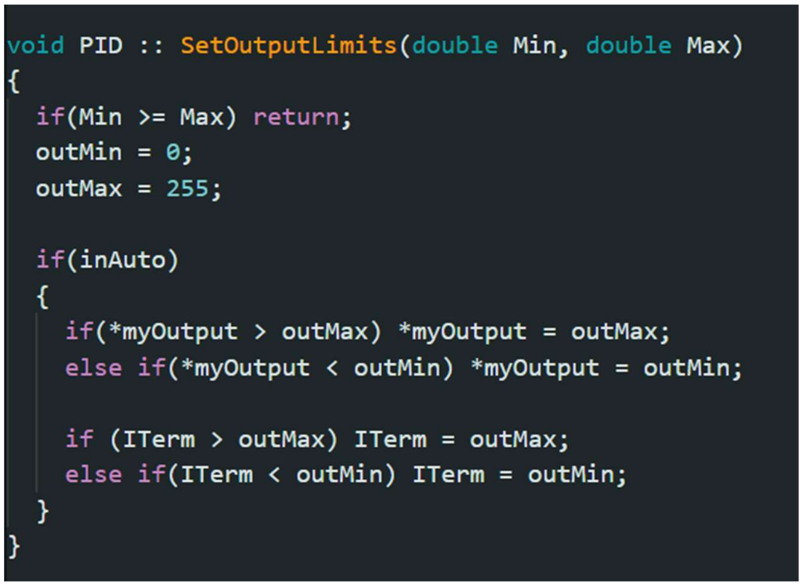
```
void PID :: SetSampleTime(int NewSampleTime)
{
  if (NewSampleTime > 0)
  {
    double ratio = (double)NewSampleTime/(double)SampleTime;
    ki *= ratio;
    kd /= ratio;
    SampleTime = (unsigned long)NewSampleTime;
  }
}
```

SetOutputLimits( )

This function will be used far more often than SetInputLimits.

While the input to the controller will generally be in the 0-1023 range (which is the default already), the output will be a little different.

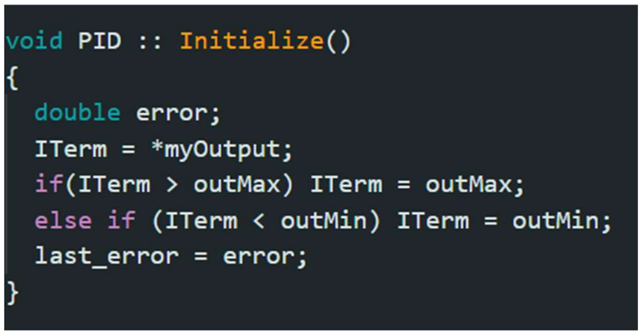
Maybe they'll be doing a time window and will need 0-8000 or something. Or maybe they'll want to clamp it from 0-125. At any rate, that can all be done here.



```
void PID :: SetOutputLimits(double Min, double Max)
{
  if(Min >= Max) return;
  outMin = 0;
  outMax = 255;

  if(inAuto)
  {
    if(*myOutput > outMax) *myOutput = outMax;
    else if(*myOutput < outMin) *myOutput = outMin;

    if (ITerm > outMax) ITerm = outMax;
    else if(ITerm < outMin) ITerm = outMin;
  }
}
```

Initialize( )

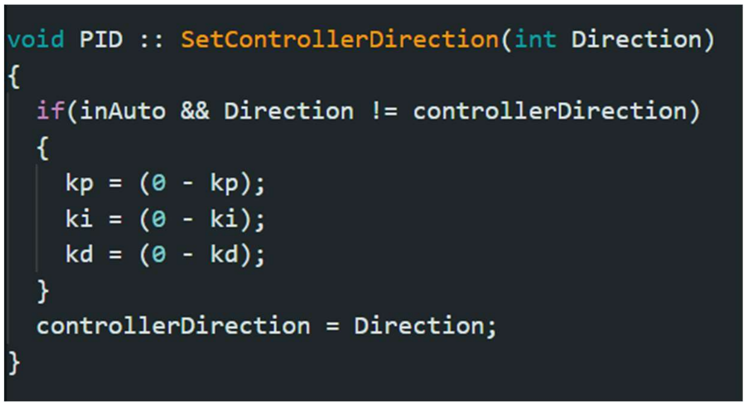
Does all the things that need to happen to ensure a bumpless transfer from manual to automatic mode.



```
void PID :: Initialize()
{
  double error;
  ITerm = *myOutput;
  if(ITerm > outMax) ITerm = outMax;
  else if (ITerm < outMin) ITerm = outMin;
  last_error = error;
}
```

SetControllerDirection( )

The PID will either be connected to a DIRECT acting process (+Output leads to +Input) or a REVERSE acting process (+Output leads to -Input). We need to know which one, because otherwise we may increase the output when we should be decreasing. This is called from the constructor.



```
void PID :: SetControllerDirection(int Direction)
{
  if(inAuto && Direction != controllerDirection)
  {
    kp = (0 - kp);
    ki = (0 - ki);
    kd = (0 - kd);
  }
  controllerDirection = Direction;
}
```

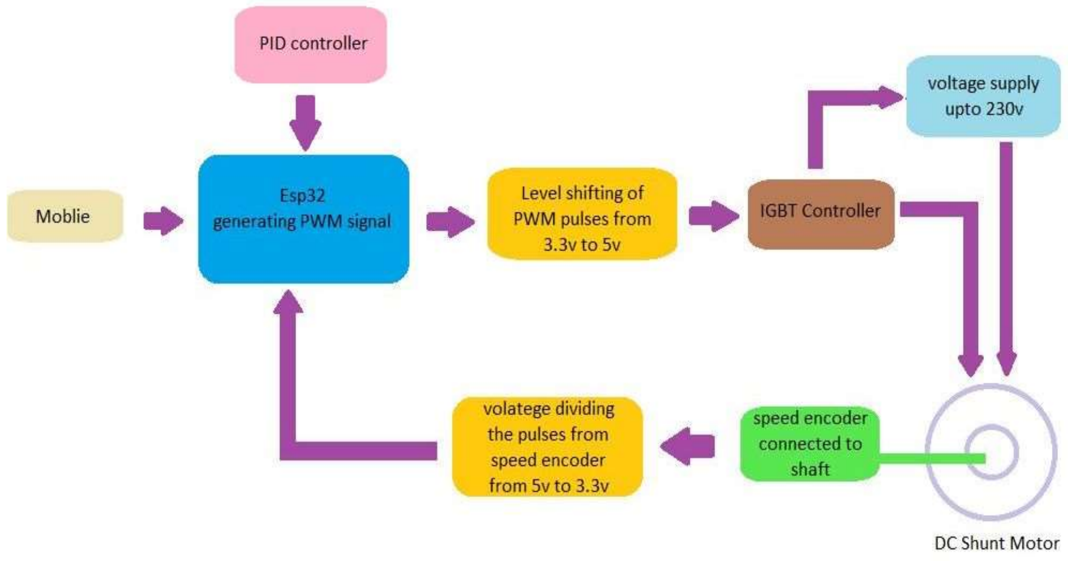
7. EXPERIMENTAL SETUP

7.1 Experimental Setup

Here the ESP32 acts as a controller. It consists of microcontroller in which the program is written and debugged into it to control the speed of the DC Shunt Motor.

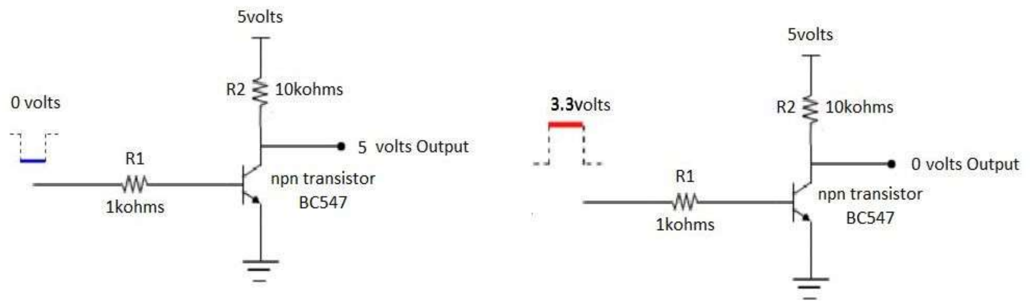
The setup consists of

- DC Shunt Motor
- ESP32 WROOM DEVKIT V1
- Speed Encoder E51S
- IGBT based DC Motor Driver
- N-P-N Transistor (BC547)
- Resistors (1K ohm, 10K ohm)
- Jumper wires
- 230V variable DC voltage supply


PID controller
Moblie
Esp32 generating PWM signal
Level shifting of PWM pulses from 3.3v to 5v
IGBT Controller
voltage supply upto 230v
volatege dividing the pulses from speed encoder from 5v to 3.3v
speed encoder connected to shaft
DC Shunt Motor

When the setup is switched ON, and the Target speed is given through mobile using BLYNK app to DC motor, error is produced. The error is given to the PID controllers, then the PID output is generated which is in the range (0-255). This PID output is used as percentage duty cycle of the PWM signal by ESP32 and generate PWM signal corresponding to PID output. This process will be continued in a loop so that the motor will rotate at desired speed.

7.2 Level Shifting



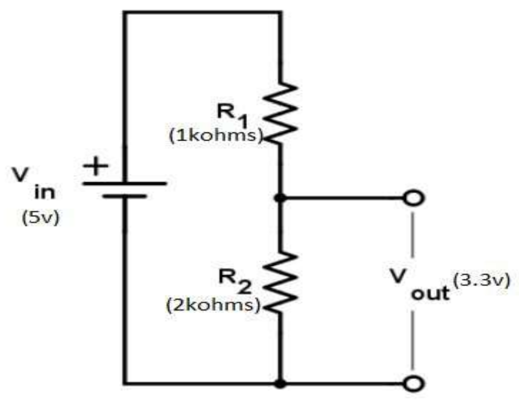
The IGBT based DC motor driver required 5V amplitude PWM signal. But ESP32 is able to generate only 3.3V amplitude PWM signal. So, we are doing level shifting of 3.3V PWM signal to 5V PWM signal.

For this Level shifting, we have used BC547 (N-P-N transistor).

7.3 Voltage Divider

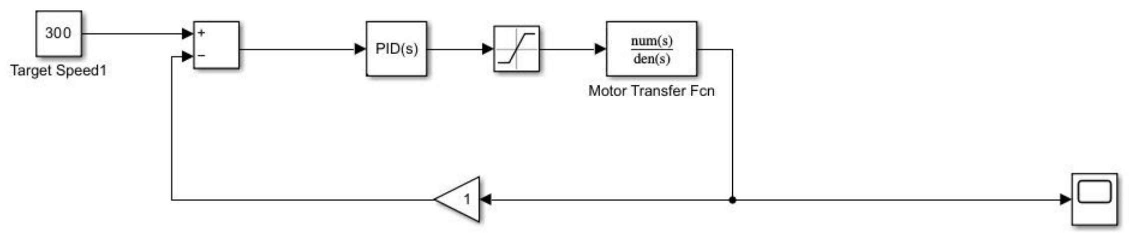
The Speed Encoder that we have used here is Autonics Speed Encoder E51S.

This Speed Encoder generates pulses of 5V amplitude, but ESP32 can withstand only a maximum of 3.3V. So, we are converting the 5V pulses that are coming out of speed encoder to 3.3V using voltage divider.



8. SIMULATION AND RESULTS

8.1 Finding Transfer function by theoretical calculations



We have determined the motor's transfer function theoretically as shown in the above block diagram.

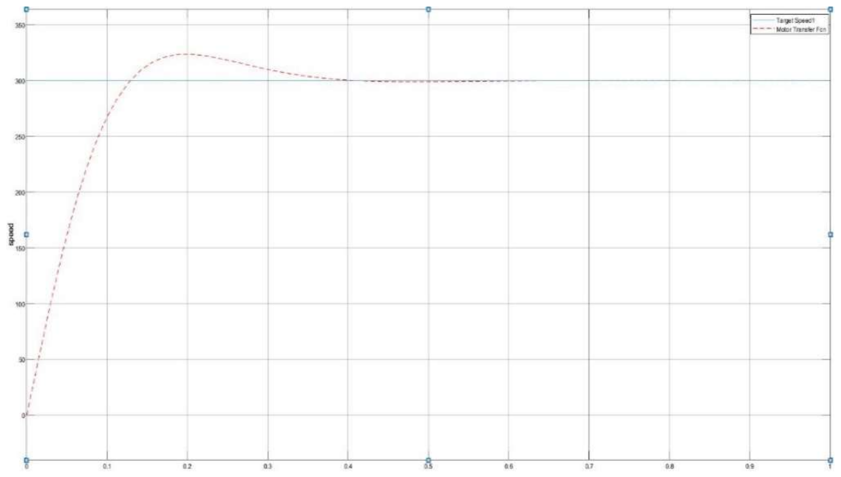
The Motor Transfer function is,

$$\frac{\omega_m(S)}{x(S)} = \left(\frac{773.053}{(9.304 * \ 10^{-4})S^2 + (15.332)S + 144.512}\right)$$

After tuning the model, the values of K_p, K_i and K_d are

$$K_p = 0.247120474909358$$
$$K_i = 4.68006121376728$$
$$K_d = -0.00086812374892241$$

Speed Result:



RiseTime: 0.0931 TransientTime: 0.3298

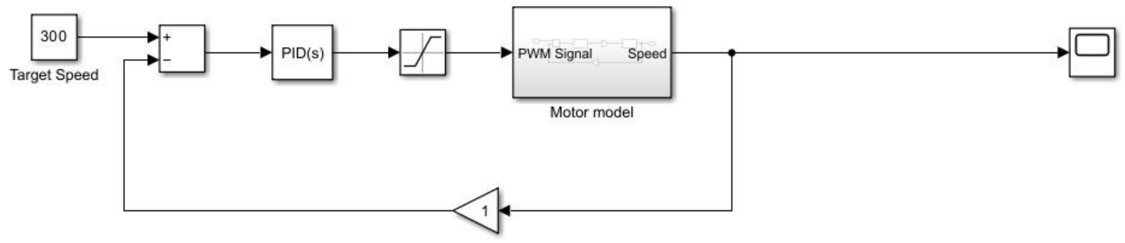
SettlingTime: 0.3298 SettlingMin: 270.6190

SettlingMax: 323.7059 Overshoot: 7.9020

Undershoot: 0 Peak: 323.7059

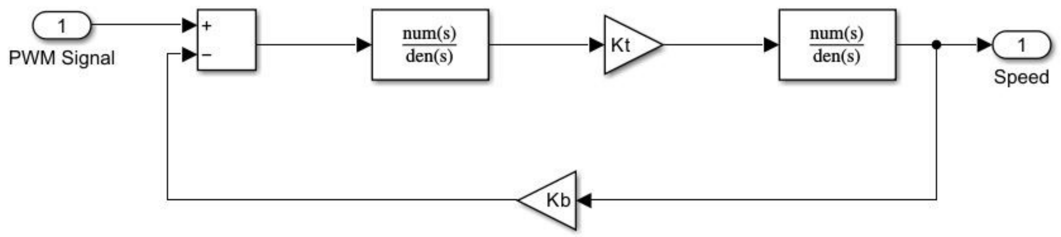
PeakTime: 0.2048

8.2 Motor Model



We have directly given the motor model to the setup.

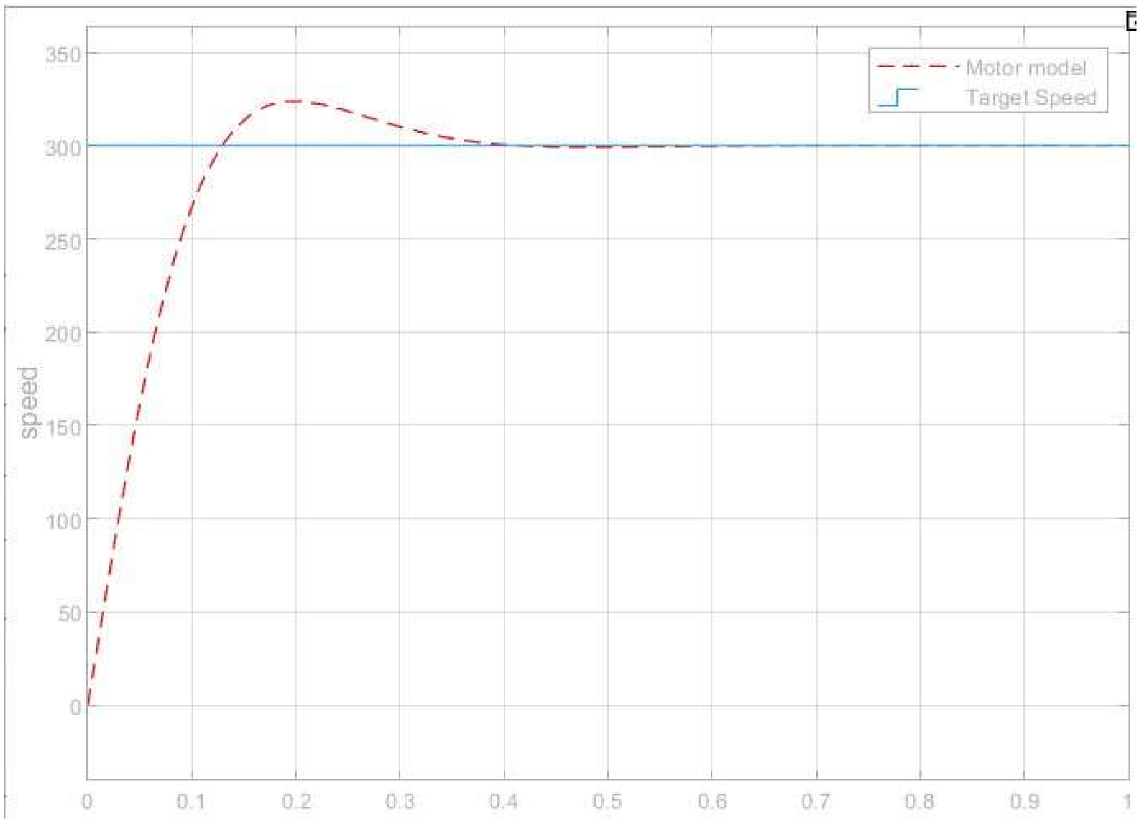
The motor model is,



After tuning, the values of K_p, K_i and K_d are,

$$K_p = 0.222860796990413$$
$$K_i = 4.22036935859341$$
$$K_d = -0.000783447577193014$$

Speed Result:



RiseTime: 0.0931	TransientTime: 0.3299
SettlingTime: 0.3299	SettlingMin: 270.5987
SettlingMax: 323.7042	Overshoot: 7.9014
Undershoot: 0	Peak: 323.7042
PeakTime: 0.2048	

CONCLUSION

Design and Implementation of PID Controllers for Speed Control of DC Shunt Motor" successfully achieved the objective of developing a remote-control system for regulating the speed of a DC shunt motor using the Blynk app on a mobile device and an ESP32 board.

Through the integration of PID controllers, the project demonstrated precise and efficient speed control of the DC shunt motor. The proportional, integral, and derivative control actions of the PID controllers provided a balanced approach, ensuring stability, accuracy, and robust performance.

The utilization of the Blynk app facilitated convenient remote-control capabilities, allowing users to adjust the motor speed easily from their smartphones. The ESP32 board served as the interface between the Blynk app and the DC shunt motor, enabling seamless communication and control.

The successful implementation of the project highlights the feasibility and practicality of using PID controllers, IoT-based solutions, and mobile app integration for motor control applications. The developed system offers enhanced automation, flexibility, and user-friendliness in regulating motor speed.

Overall, the project's outcomes demonstrate the potential of combining PID controllers, Blynk app integration, and ESP32 board for efficient and user-friendly speed control of DC shunt motors, opening doors for innovative applications in the field of motor control and automation.