ABSTRACT

BLDC motors are popular for many industrial applications due to their reliability, high efficiency, high starting torque, and reduced electrical noise. The brushless DC motors have an unlimited number of applications nowadays, many of which require the motor speeds to be controlled. The PID controllers is the answer for the mentioned problem. This project implemented a PID algorithm to regulate a brushless DC motor speed using a programmed microcontroller unit.

In this project, Arduino Uno development board was used as a controller unit and a Kelly Controller expansion board served as a motor driver and power unit. The selected BLDC motor was the COMPAGE BLDC E-rikshaw motor coming with the hall-effect sensors and encoder outputs. This project was built based on C++ Project using Arduino IDE. It implemented an algorithm serving the purpose of driving the BLDC motor and PID algorithm to control the motor speed.

The results obtained shows that PID controller eliminated the need of manual control but slightly decreased the system stability. The after effect of PID controller also decreased with an increasing target speed value due to the maximum angular acceleration of the motor.

INDEX

List of Figures

List of Table

CHAPTER 1

INTRODUCTION

Brushless direct current (BLDC) motors are used in wide-spread applications like electric vehicles (EV), commercial utilization, industrial compressors, and particularly in aerospace applications. This is mainly due to their high efficiency, low noise, robust construction, no brushes, less maintenance, and good speed-torque performance. These features attracted many researchers towards this domain. In past few decades, BLDC motor is widely used in many real world applications, but the challenge in this is controlling of the motor. [11]- [14]. The BLDC motor can be controlled by various approaches and are classified as Proportional Integral Derivative (PID), Fuzzy logic control (FLC), Artificial Neural Network (ANN) Control, Adaptive Neuro and Fuzzy Inference System (ANFIS), Sliding Mode Control (SMC), Extended Kalman Filter (EKF), Model Reference Adaptive Control (MRAC), and Model Predictive Control (MPC) [15]-[17].

The existing BLDC control architecture can be classified as follows open-loop, closed-loop and cascade control. Open-loop control (OLC) is perhaps the simplest strategy having a reference generator, control module, power module, an inverter and hall-effect sensors to control a BLDC motor. In OLC, the BLDC motor follows the reference speed which is provide by set-point generator. In OLC potentiometer could be the set-point generator, which control the motor speed. However, in certain applications the set-point generators could also be hall-effect sensors (i.e., electric bikes). The control module performs three tasks:

1. Generate pulse-width modulation (PWM) to control the motor speed
2. Provide electronic commutation based on current position information
3. Generate gating inputs to the inverter.

The inverter circuit consists of MOSFET or IGBT switches and driving circuit ICs or components.

The BLDC motor OLC is simple compared with closed-loop control (CLC), and used in constant load applications mostly and to detect initial rotor position in sensor less methods. However, the reliable starting is limited by the torque in conventional open-loop speed control. In addition, stall or locked rotor condition should be prevented by taking appropriate action. CLC in BLDC motor is used to track the user defined speed. In addition the motor speed may get affected by external disturbances such as speed barkers, obstacles, heavy wind, it may increase torque and current variations as well. The controller must respond immediately to minimize the error and variations, by changing “on time” or “off time” in the Pulse width modulation (PWM) pulses.

In today’s world, brushless direct current (BLDC) motors can be found in many applications ranging from big propulsion systems in an electric aircraft to a small compact disk (CD) drive. For some cases, it is critical for the motor speed to be controlled which leads to the need of motor controllers. The proportional-integral-derivative controllers (PID controllers) apply a control mechanism using proportional, integral and derivative terms to calculate optimal outputs from the differences between the measured values and the target value [3]. The PID concept is the most used control algorithm in industrial control systems because of its superior precision and accuracy. The PID controller is a solution to the mentioned problem. This project aimed to implement the PID controller algorithm on an Arduino to regulate a BLDC motor speed.

LITERARY REVIEW

In this section of the thesis, the theoretical ideas behind the project are explained. There are two fundamental theory parts that are the DC motor and the PID controller.

1.1 DC Motor

A DC motor generally transform electrical energy to mechanical energy in form of rotation [1]. The stationary part called stator and the moving part called rotor of a DC motor consist of permanent magnets or electromagnets (made by running the current through the solenoids) which generate magnetic fields. The attractive force between two magnetic fields made the motor rotate as illustrated in Figure 1.1

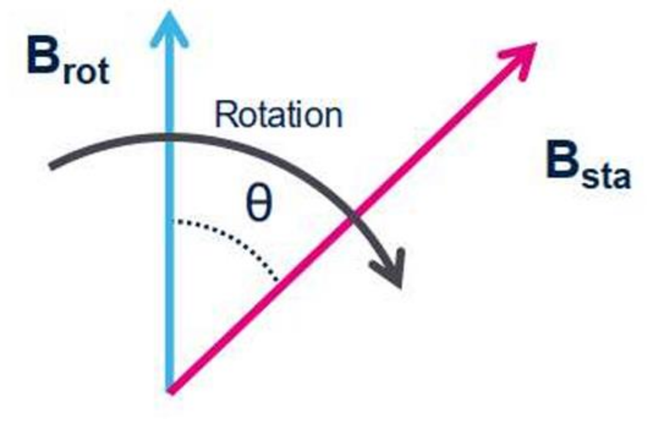


Figure 1.1 Working principle behind the rotation of DC motor [4]

The formula for the torque applied on the rotor is given below.

$$\tau \propto B \cdot B \cdot \sin(\theta)$$

Where

- τ is the torque
- B is the magnetic fields generated by the rotor
- B is the magnetic fields generated by the stator
- θ is load angle (the angle between two vectors $\vec{B}$ and $\vec{B}$)

Deduced from the formula, the maximum output torque is archived with the load angle θ being 90°. Now common two type of DC motor are 1. Brushed and 2. Brushless.

1.1.1 Brushed DC Motor

In a brushed DC motor, the rotor is composed of pairs of solenoids and the stator consists of permanent magnets. The stator electromagnetic field remains fixed while the rotor electromagnetic field is constantly changing during the rotation. The load angle is kept close to 90° by constantly charging the next pair of solenoids when they come to a specific position (as in Figure 1.2).

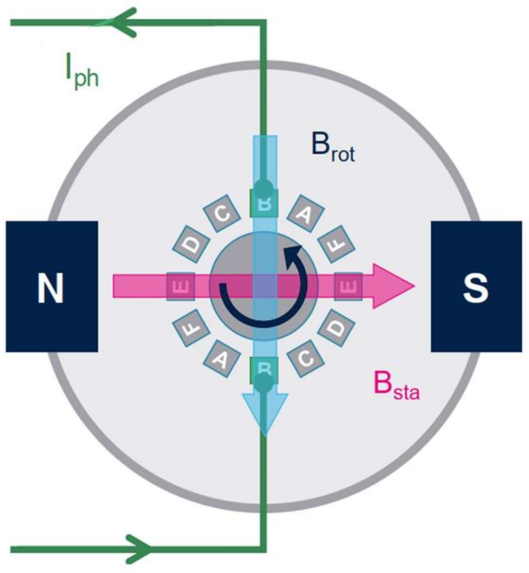


Figure 1.2. Working principle of brushed DC motor

The connectors between the circuit and solenoids are called brushes, thus the name of brushed DC motor. Changing the current the direction will make the motor to rotate in the opposite direction.

1.1.2 Brushless DC Motor

A brushless DC motor have rotor containing a permanent magnet and its three solenoids as a stator connected in a star arrangement and positioned 120° from each other. The rotor electromagnetic field changes with the rotation so that in order to keep the load angle as close to 90° as possible, the stator electromagnetic field is constantly changed according to the position of the rotor. Figure 1.3 below show the structure and working principle of a brushless DC motor.

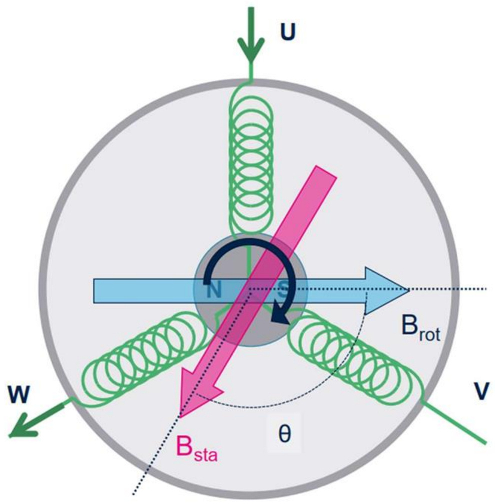


Figure 1.3. Physical structure and working principle of brushless DC motor [6]

Despite the fact that brushed DC automobiles have a low initial value and easy manipulate of motor pace, the excessive renovation of the brushes makes the brushless counterpart advanced in terms of sturdiness and reliability. However, running a brushless DC automobiles is more complicated. Brushless motors may be built in several extraordinary physical configurations: inside the traditional (inrunner) configuration, the permanent magnets are a part of the rotor. 3 stator windings surround the rotor. Inside the 'outrunner' (outside-rotor) configuration, the radial-courting among the coils and magnets is reversed; the stator coils form the centre (center) of the motor, at the same time as the everlasting magnets spin inside an overhanging rotor which surrounds the core. [2]

1.1.3 Six-step Algorithm

The six-step algorithm is a driving operation of three-phase BLDC motors. In six step, there are six current directions running through two of the three phases, which created six discrete directions of magnetic field for the stator (illustrated in Figure 1.4). To acquire the position of the rotor, the hall-effect sensors are used for sensored BLDC motors and BEMF feedbacks are used for the sensorless counterparts. Eventually, the correct step is applied accordingly to the known rotor position and the intended rotation direction.

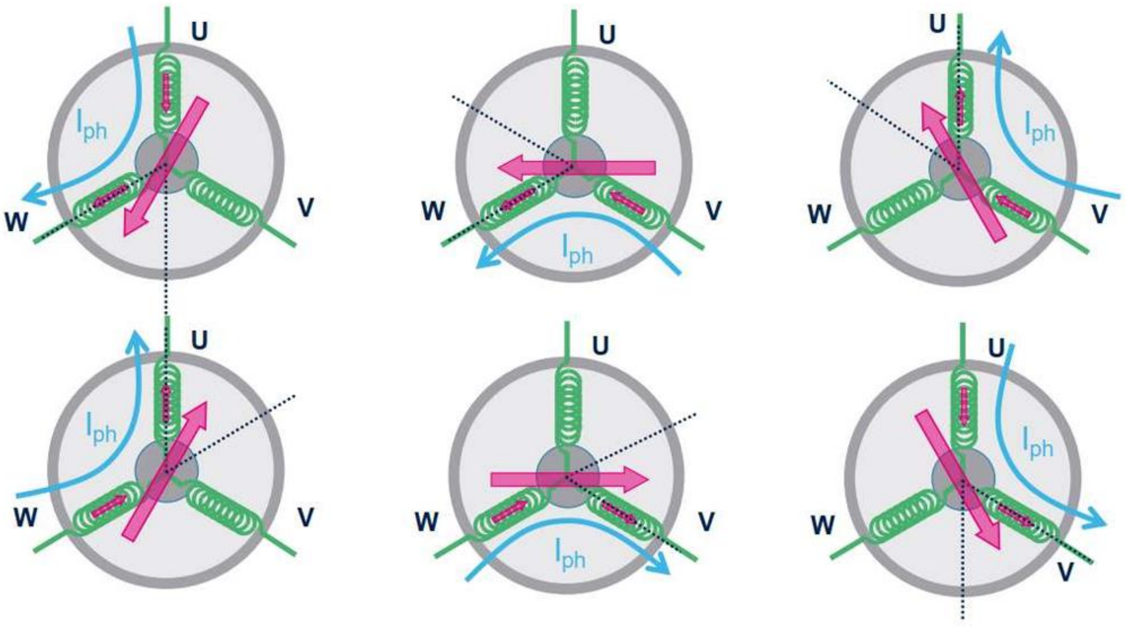


Figure 1.4. Six combinations in six-step algorithm [7]

1.2 PID Controller

A proportional-integral-derivative controller (also known as a PID controller or three-term controller) is a closed loop control mechanism that is widely used in industrial control systems. It uses the three control terms of proportional, integral and derivative to calculate the output from the error value, which is the difference between the designated target value called a setpoint (SP) and the measured feedback value called a process variable (PV).[3]

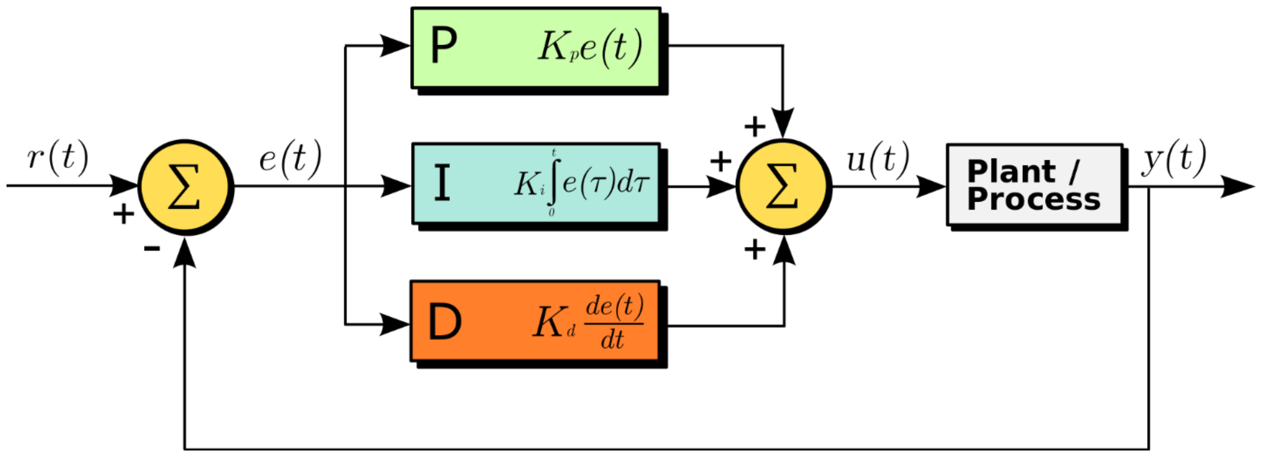


Figure 1.5. The working principle of PID controller [3]

The mathematical formula of overall control function of PID controller is

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau)d\tau + K_d \cdot \frac{de(t)}{dt}$$

Where

- t is running time
- τ is integral time
- $r(t)$ is setpoint
- $e(t)$ is error
- $u(t)$ is control variable
- $y(t)$ is process variable
- K is proportional gain
- K is integral gain
- K is derivative gain

From the block diagram in Figure 1.5, the error value $e(t)$ as a difference between setpoint $r(t)$ and process variable $y(t)$ is continuously measured then the output control variable $u(t)$ is calculated based on proportional, integral and derivative terms to minimize the error over time. Even though the PID controller has three control terms, some applications may not need all of them to have appropriate control. In those cases, the selective use of the control terms can be achieved by setting the coefficients (K, K or K) of unnecessary terms to 0 so that they have no impact on the output. [3]

1.2.1 Proportional Term

The proportional term produces an output value that is proportional to the error value. The P term takes the formula

$$P=K\cdot e(t)$$

By changing the coefficient K (also known as proportional gain), the response of output based on P term can be adjusted. If the proportional gain is too large, the system overreacts with the error value and the process variable oscillates around the setpoint, which increases the settling time and decreases the stability. Meanwhile, with small proportional gain, the system becomes less responsive, and the rise time is increased. [3]

Figure 1.6 below shows the graph of process variable responses with different values of K. The proportional term alone cannot correct the system of steady-state error, which is the difference between the desired final output and the actual one. [3]

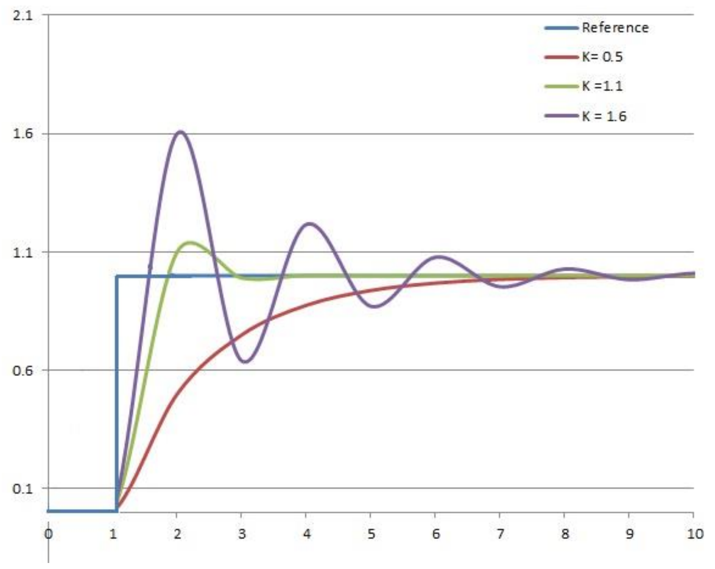


Figure 1.6. (a) System responses with different value of K [3]

1.2.2 Integral Term

The integral term considers both the error plus duration of the error when calculating an output [3]. In fact, the mathematical formula for I term comes with the integral, which is the sum of instantaneous errors over time.

$$I = Ki \, . \int_0^t \mathrm{e}(\tau)\mathrm{d}\tau$$

Increasing the K gain to, thus increasing the impact of I term, will help to eliminate the steady-state error but in exchange it increases the overshoot, settling time and decreasing system stability [3]. The graph in Figure 1.6 (b) illustrates how I term affects the system respond.

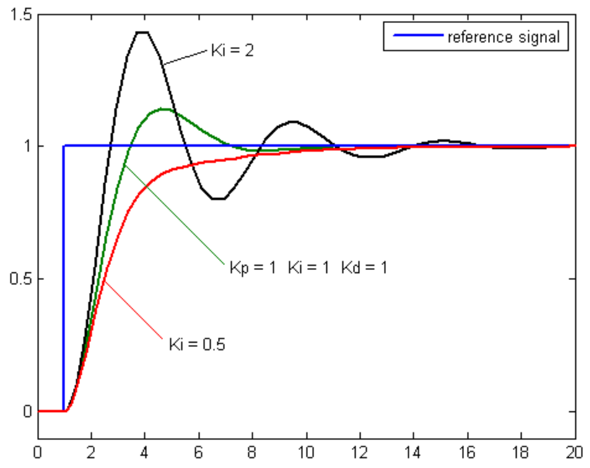


Figure 1.6. (b) System responses with different value of K [3]

1.2.3 Derivative Term

The derivative term produces an output base on the derivative of the errors, in other words, it takes into calculation the error change rate over time [3]. The mathematical formula for the derivative term is

$$D{=}K_{\mathrm{d}} \cdot \frac{d\mathrm{e}(\mathrm{t})}{dt}$$

By predicting the error changes in the system, the derivative term largely decreases the settling time while improving the stability with small K value [3]. The graph in Figure 1.6 (c) shows the system response with different values of K_{d}

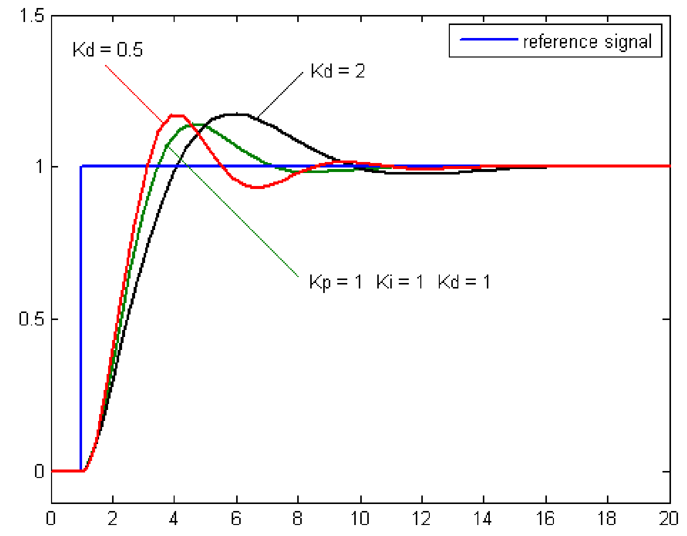


Figure 1.6(c) System responses with different value of K [3]

1.2.4 Tuning

Different control systems have different control parameter values that lead to their desired control responses and the work of finding those optimal values is called tuning. The system requirements may include the rise time, settling time, overshoot allowance but the basic one is stability [3]. There are various tuning methods with different levels of sophistication but in this project, manual tuning method was used. Manual tuning requires understanding the impact on the system of control terms and adjusting them one by one (firstly K_p, then K_i and finally K_d) until the system response meets the requirements.

CHAPTER 2

METHODOLOGY

2.1 Resources:

The resource consumed, apart from working hours and materials, can be divided into two parts: hardware and software.

2.1.1 Hardware:

Arduino UNO:

Arduino Uno is a microcontroller board based on the ATmega328P shown in below fig. It has 14 digital input/output pins (of which 6 may be used as PWM outputs), 6 analog inputs, a sixteen MHz quartz crystal, a USB connection, an electricity jack, an ICSP header and a reset button. It consists of the whole thing needed to guide the microcontroller; surely join it to a computer with a USB cable or strength it with an AC-to-DC adapter or battery to get started [8]. On this undertaking, the Arduino microcontroller may be very well-desirable to force the DAC signal for varying motor controller output which then changes the rate of BLDC motor. This improves the output reaction for the BLDC motor speed control system.

Figure 2.1. Arduino Development Board

DAC (MCP 4725):

The MCP4725 is a low-power, high accuracy, single channel, 12-bit buffered voltage output Digital-to Analog Convertor (DAC) with non-volatile memory (EEPROM). The DAC input and configuration records may be programmed to the non-volatile memory (EEPROM) by using the consumer the usage of I2C interface command. The MCP4725 has an external A0 cope with bit selection pin. This A0 pin may be tied to VDD or VSS of the consumer's software board. The MCP4725 has a -twine I2C™ like-minded serial interface for popular (one hundred kHz), fast (four hundred kHz), or excessive pace (3.4 MHz) mode. The MCP4725 is an ideal DAC device wherein design simplicity and small footprint is favored, and for applications requiring the DAC device settings to be saved for the duration of strength-off time. The device is to be had in a small 6-pin SOT-23 package [9]. DAC takes virtual fee in the form of depend from Arduino and converts it into corresponding analog price and feed it to motor controller to differ the rate of motor.

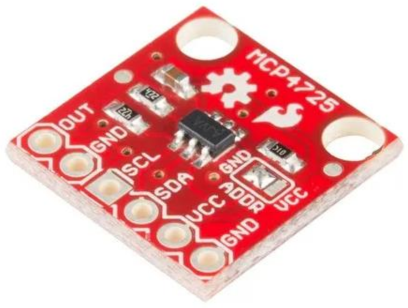


Figure 2.2. MCP 4725

Encoder:

Encoder is used to get the speed feedback from the motor. It is an incremental-type encoder that gives pulses proportional to the speed of the motor. It outputs 1024 pulses per revolution. It requires an external power supply of 10 to 30 V DC.

Figure 2.3. Encoder

Kelly Controller:

Kelly's programmable motor controllers provide efficient, smooth and quiet controls for electric motorcycles, golf carts and go-carts, as well as industrial motor control. The KLS-H motor controller must be based on the hall sensors type. Compared to the traditional trapezoidal waveform control technology, this technique is based on sinusoidal wave driving technology to reduce the operation noise and 1/3 switching loss, which well meets the noise reduction and efficiency requirements in the application of a DC brushless motor. It uses high power MOSFET's, SVPWM and FOC to achieve efficiencies of up to 99% in most cases. A powerful microprocessor brings comprehensive and precise control to the controllers [10].

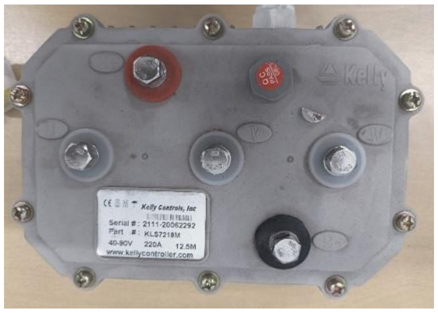


Figure 2.4. Kelly Controller

BLDC Motor:

BLDC motors are widely used for low power high speed drives and servo application. It is becoming quite popular in low range electrical vehicles. The BLDC motor used in this project is a

0.5 KW motor with rated speed of 3000 RPM, its rate voltage is 48 V with 1.4 N-m Cont. Torque.

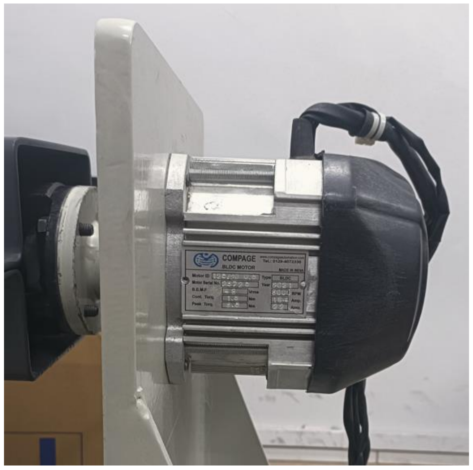


Figure 2.5. BLDC Motor

Dynamometer:

A dynamometer is a device used for measuring and analyzing the power, torque, and speed characteristics of an engine, motor, or any rotating machinery. Dynamometers play a crucial role in performance testing, research, development, and calibration of engines and motors. They provide valuable data on power and torque characteristics, allowing engineers to optimize the design, improve efficiency, and ensure the reliability of the tested machines.

In this project, dynamometer is used to load the BLDC motor electrically. Motor was loaded up to 0.4 N-m load torque by dynamometer during testing.

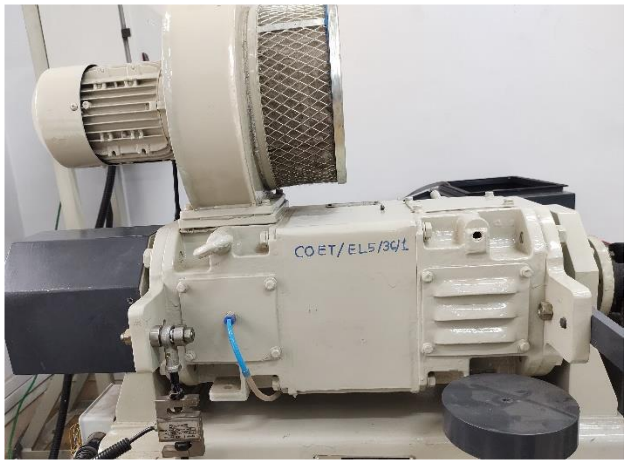


Figure 2.6. Dynamometer

Data Logger:

A data logger is a device or system that captures, stores, and organizes data for subsequent analysis and evaluation. The data logger used in this project is a vital component that facilitates the collection and recording of important data during the closed-loop speed control of the BLDC motor.

In the context of the project, the data logger is employed to record various parameters and variables related to the motor's performance and the closed-loop control system. Data Logger was used to adjust the loading value for motor using dynamometer.

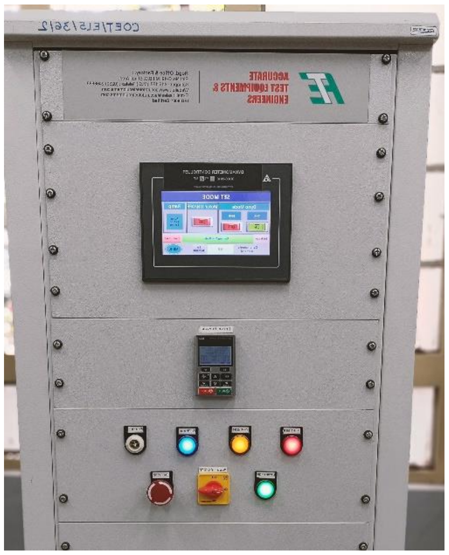


Figure 2.7. Data Logger

2.1.2 Software

Arduino IDE 2.1.0 changed into used all through the coding a part of this venture. The Arduino incorporated development surroundings - or Arduino software program (IDE) - carries a text editor for writing code, a message area, a text console, a toolbar with buttons for not unusual capabilities and a series of menus. It connects to the Arduino hardware to add programs and communicate with them.

2.1.3 System Building

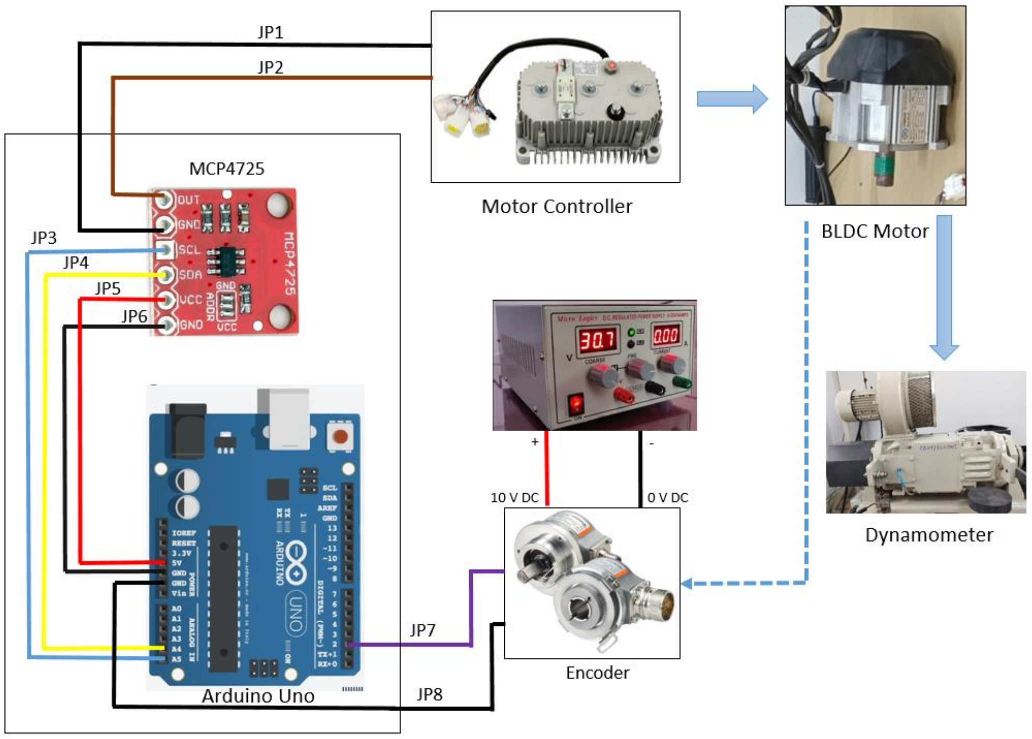


Figure 2.8 Schematic diagram of Experimental Setup

Jumper	Permitted Configuration
JP1	Ground connection (DAC and Motor throttle)
JP2	Output of DAC to Variable of Motor throttle
JP3	SCL Connection (DAC and Arduino UNO)
JP4	SDA Connection (DAC and Arduino UNO)
JP5	VCC (DAC) to 5 V (Arduino UNO)
JP6	Ground connection (DAC and Arduino UNO)
JP7	Encoder signal to Pin 2 (Arduino UNO)
JP8	Ground connection (Arduino UNO and Encoder)

Table 1. Jumper Setting

2.2 Open loop speed control

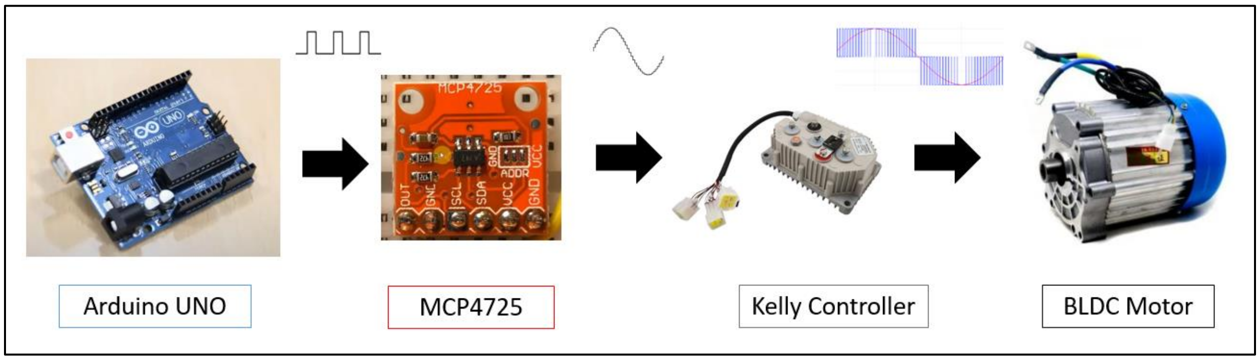


Figure 2.9 Open Loop Speed Control of BLDC Motor

The above block diagram shows the open loop speed control of the BLDC motor. In this setup, we have created an Arduino program that feeds digital voltage from Arduino to Digital to Analog (DAC) converter such that the output of DAC varies its output from 0 to 5V (Analog Voltage) which is required at the input of Kelly controller. Then Kelly controller adjusts the firing pulses of inverter circuit switches to vary the duty cycle which changes the voltage to the BLDC motor and changes its speed.

Below fig. shows a flowchart for interfacing of Arduino and Digital to Analog Converter (DAC) to obtain analog voltage at the output of DAC in the range of 0 to 3.5 V with increments of 0.5 V after a delay of 10 sec between each step. After reaching 3.5 V the output of DAC starts decreasing from 3.5 V to 0 V in steps of 0.5 V with a delay of 10 sec between each step. This program simulates DAC output for 1 complete drive cycle of the BLDC motor.

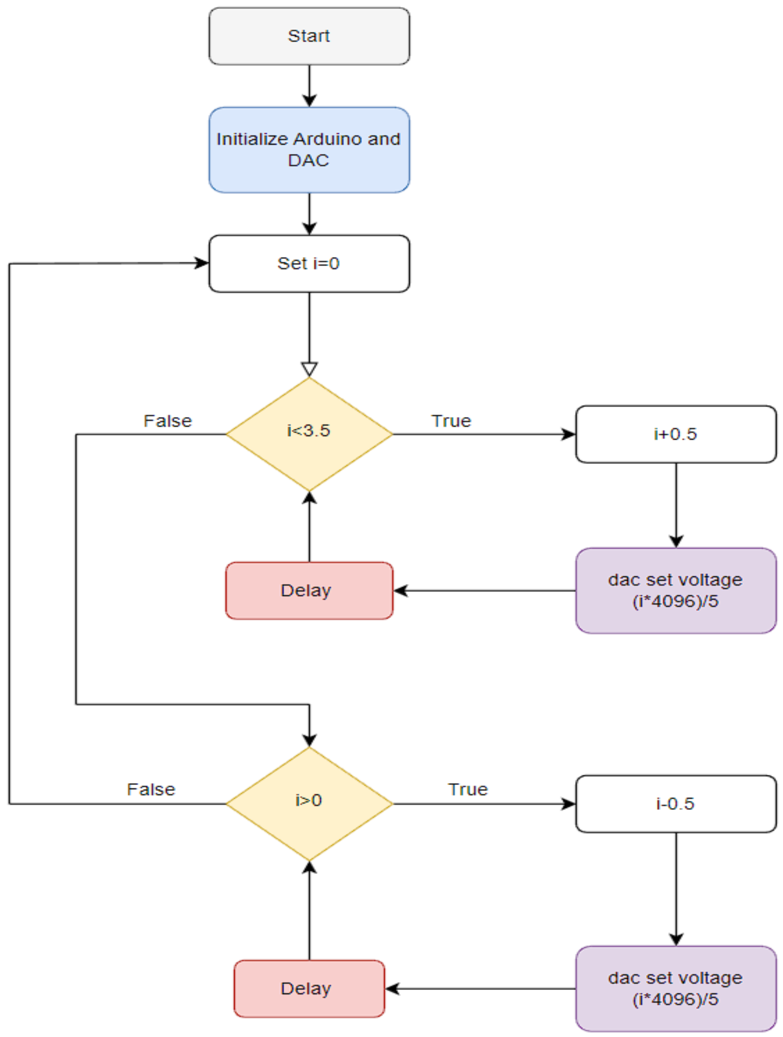

Start
Initialize Arduino and DAC
Set i=0
False
i<3.5
True
i+0.5
Delay
dac set voltage (i*4096)/5
False
i>0
True
i-0.5
Delay
dac set voltage (i*4096)/5

Figure 2.10. Algorithm for obtaining DAC Output

2.3 Closed loop Speed Control

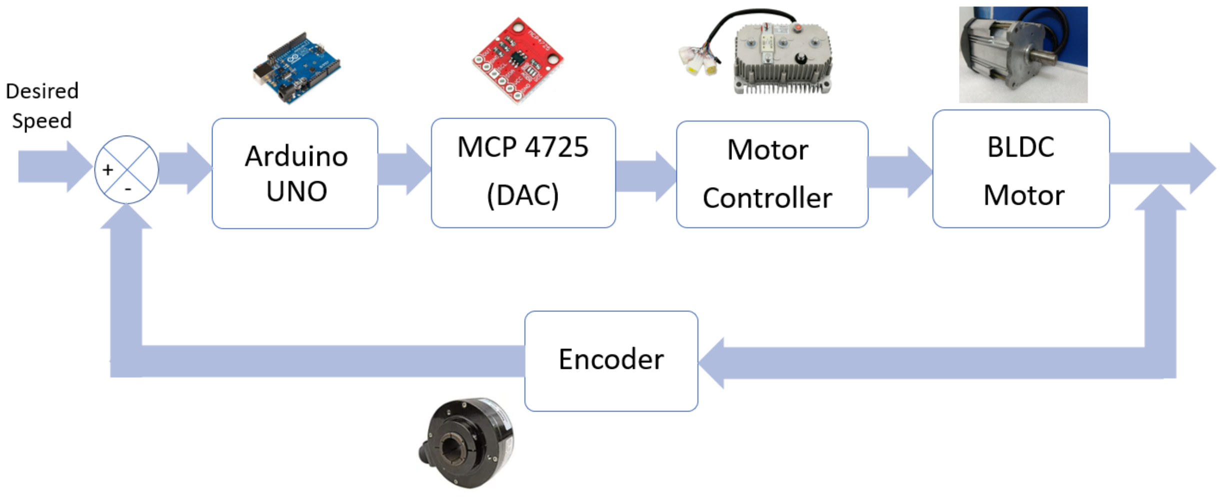


Figure 2.11 Closed Loop Speed Control of BLDC Motor

Above fig. suggests the closed-loop speed manipulate of the Brushless DC motor which is used to run the motor on the preferred pace. The rate of an electric motor may be controlled by means of converting the variable input voltage for the throttle role to a Kelly controller. The Kelly controller is a sort of motor controller this is liable for adjusting the angular velocity and torque of the motor. To trade the motor speed, a variable input voltage is despatched to the Kelly controller. This voltage is generated by a virtual-to-analog converter (DAC), which converts a digital sign acquired from an Arduino UNO board into an analog voltage. The DAC is liable for producing an analog voltage inside the range of zero-5V, that's the variety of voltages that the motor controller can receive.

The digital sign acquired by the DAC corresponds to an error signal, and the right count number is sent to the DAC IC. As soon as the DAC generates the ideal analog voltage, it's far despatched to the motor controller, which adjusts the motor's speed. An encoder is used to offer velocity feedback within the form of pulses that are then despatched to an Arduino board. The Arduino compares the favoured pace value with the remarks price from the encoder to generate a blunders sign. This mistake signal corresponds to the distinction among the real pace of the motor and the preferred velocity. The error sign is then fed to the proportional-critical-spinoff (PID) common sense in the Arduino board.

The PID good judgment generates a control sign that is a count despatched to the digital-to-analog converter (DAC). The DAC modifications the voltage to the motor in such a manner that the motor velocity achieves the desired pace. This remarks manipulate device allows the motor to run at various load situations, because the output speed is adjusted primarily based at the remarks of the real pace. Through the use of the PID common sense, the Arduino can continuously adjust the manage sign to the DAC, ensuring that the motor continues a constant speed even when there are adjustments inside the load.

2.3.1 Speed Measurement

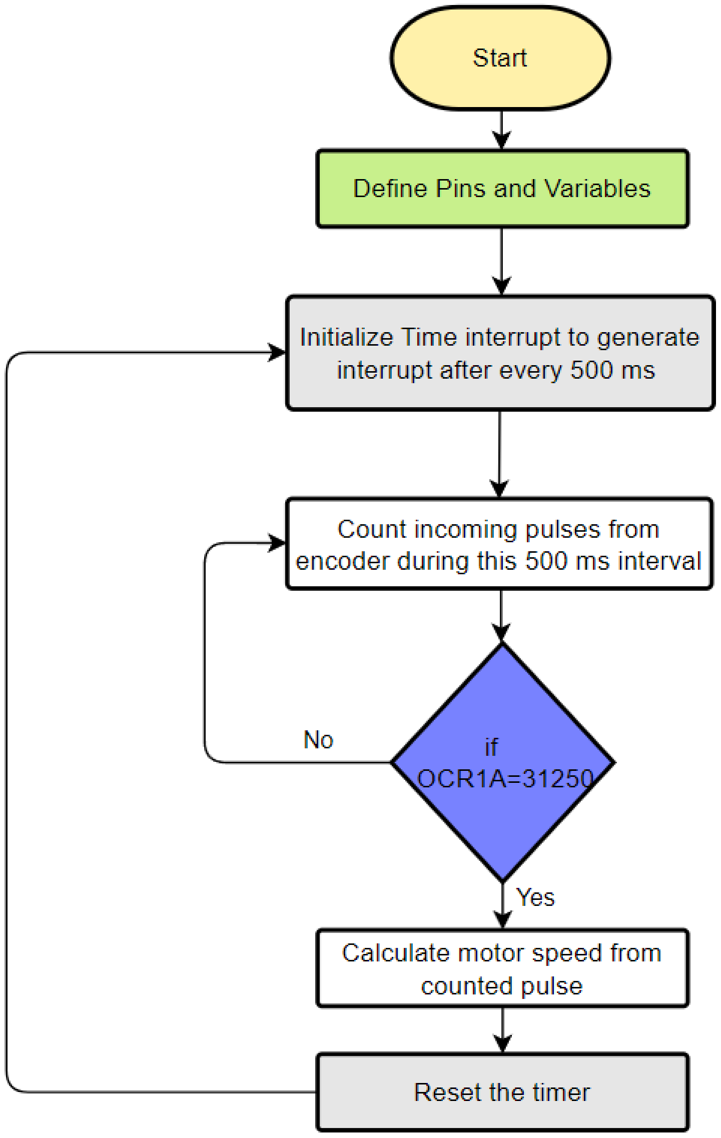

Start
Define Pins and Variables
Initialize Time interrupt to generate interrupt after every 500 ms
Count incoming pulses from encoder during this 500 ms interval
No
if OCR1A=31250
Yes
Calculate motor speed from counted pulse
Reset the timer

Figure 2.12 Flowchart for Speed measurement using Encoder Pulses

1. Initialize the necessary variables, including pulseCount and duration, to hold the pulse count and timing information.

2. Configure the interrupt pin on the Arduino and set up the timer interrupt to trigger every 500ms. This ensures that the interrupt service routine (ISR) will be executed at regular intervals.

3. The program waits for the interrupt to occur. The interrupt can be triggered by a pulse from the encoder or by the timer reaching the specified interval.

4. Pulse Detected: If a pulse is detected from the encoder, the interrupt service routine (ISR) is executed, which increments the pulseCount variable by one. This keeps track of the number of pulses received within the 500ms interval.

5. Timer Interrupt: If the interrupt is triggered by the timer reaching the specified interval (500ms), the ISR associated with the timer interrupt is executed.

6. Calculate Speed: In the ISR, the duration is calculated based on the pulseCount and the known time interval (500ms). The pulseCount is multiplied by 0.5 to convert it into seconds.

7. Speed Conversion: Using the calculated duration, the speed in RPM (Rotations per Minute) is computed. The pulseCount divided by 2 gives the number of rotations, and dividing it by the duration in minutes gives the rotations per minute.

8. Reset Variables: After calculating the speed, the pulseCount is reset to zero, preparing it for the next interval.

9. Output/Processing: You can utilize the calculated speed value as needed. This might involve displaying it on an LCD, sending it over a communication protocol, or performing any other required operations.

10. Loop: The flow returns to waiting for the next interrupt, allowing the process to repeat indefinitely.

11. End: The flowchart ends here.

By following this flowchart, you can continuously measure the speed using an encoder and Arduino UNO at regular intervals of 500ms, making use of interrupt-based programming to accurately capture and process the encoder pulses.

2.3.2 PID Controller

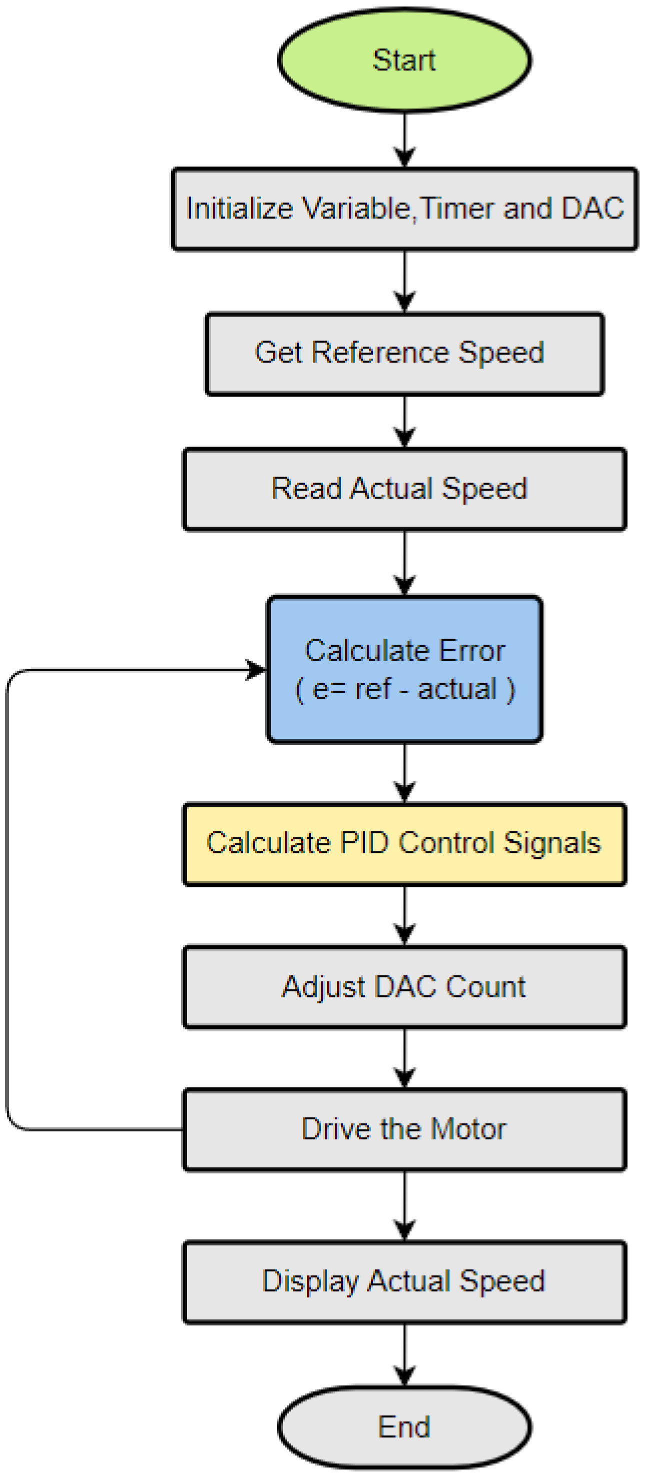


Figure2.13 Flowchart for implementation of PID Control for closed-loop control of BLDC motor

1. Initialize the necessary variables, including dutycycle, to hold the count to be sent to DAC.

2. Set the desired setpoint for the speed.

3. Initialize the PID controller variables: Kp (proportional gain), Ki (integral gain), and Kd (derivative gain). These values are determined through tuning.

4. Initialize variables for the PID controller: speed error (Nerror), integral term (sum), and derivative term (diff).

5. Enter a control loop that runs for fixed time interval of 500 msec

6. Read the current speed measurement from the encoder during this timer interval.

7. Calculate the error as the difference between the setpoint and the measured speed: error (Nerror) = setpoint (Nref) – measurement (Nactual).

8. Calculate the proportional term: proportional = Kp * error.

9. Calculate the integral term: integral = integral + (Ki * error).

10. Calculate the derivative term: derivative = Kd * (error - err_prev).

11. Compute the PID output: output = proportional + integral + derivative.

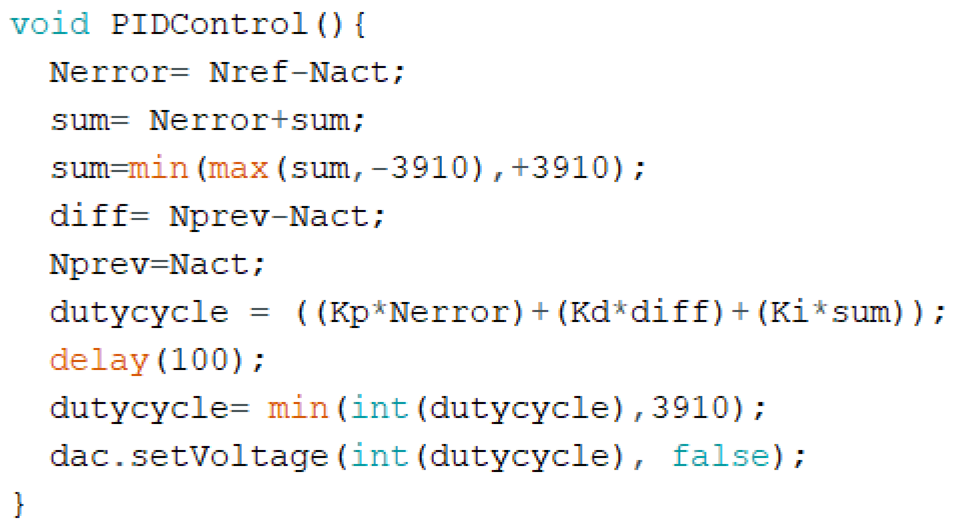
12. Apply the output to adjust the count sent of DAC.

13. Update the previous error: err_prev = error.

14. Update the previous measurement: prev_measurement = measurement.

15. Repeat the control loop.

Figure 2.14 Pseudocode for PID algorithm



```
void PIDControl(){
  Nerror= Nref-Nact;
  sum= Nerror+sum;
  sum=min(max(sum,-3910),+3910);
  diff= Nprev-Nact;
  Nprev=Nact;
  dutycycle = ((Kp*Nerror)+(Kd*diff)+(Ki*sum));
  delay(100);
  dutycycle= min(int(dutycycle),3910);
  dac.setVoltage(int(dutycycle), false);
}
```

CHAPTER 3

RESULTS

3.1. Open loop Speed Control:

For controlling the speed of motor, Throttle input motor need to be controlled. Since, DAC IC is connected to the input of Throttle signal it can provide the different magnitude of voltage which will eventually drive the motor.

In given table, it shows Voltage – Speed characteristics of BLDC motor. In this method, using programming in Arduino Uno, voltage supplied by the DAC are varied. According to change in voltage, corresponding speed change is observed.

Sr.No	DAC output voltage (in V)	Speed of Motor (in RPM)
1.	2.09	802
2.	2.42	943
3.	2.67	1031
4.	2.86	1107
5.	3.20	1224
6.	3.55	1335
7.	3.90	1447
8.	4.26	1564
9.	4.50	1623
10.	4.84	1722

Table 2. DAC Voltage Vs Speed of Motor

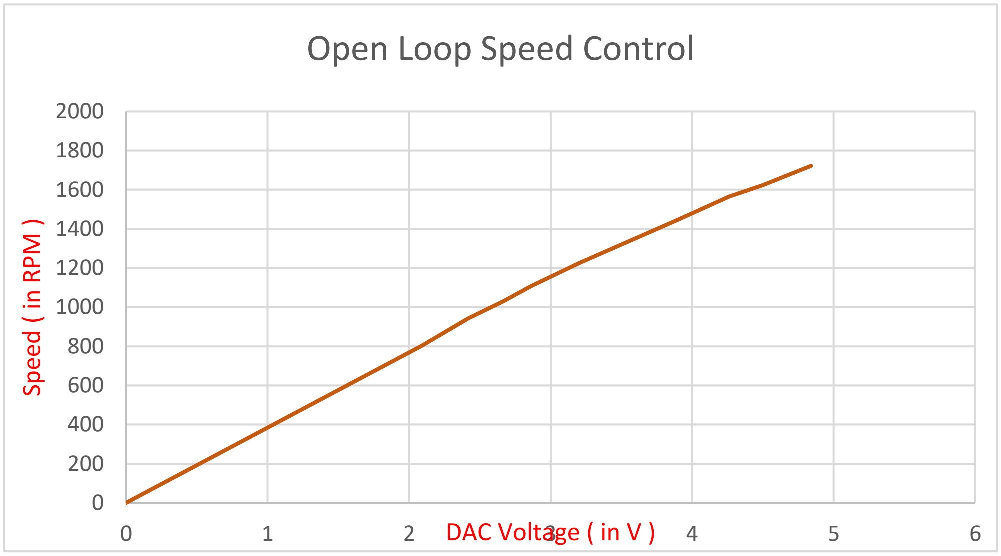


Figure 3.1 DAC Voltage Vs Speed Characteristics

3.2 Closed loop speed control:

3.2.1. Without PID Controller:

A. **Reference Speed= 1200 RPM, No Load.**

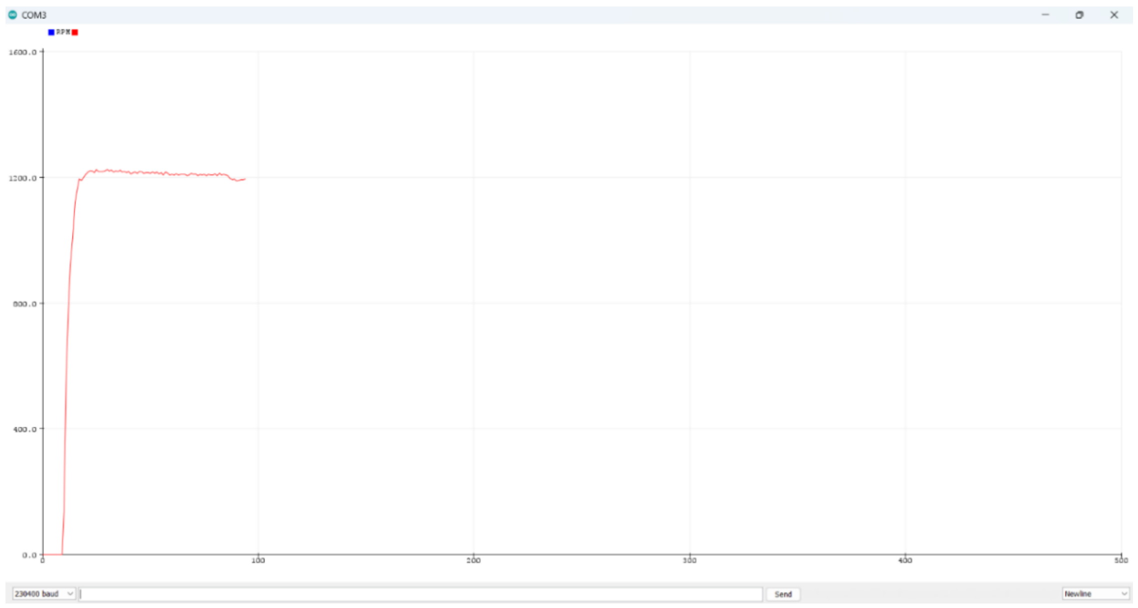


Figure 3.24 (a)

B. **Reference Speed= 1200 RPM, 0.3 N-m Load.**

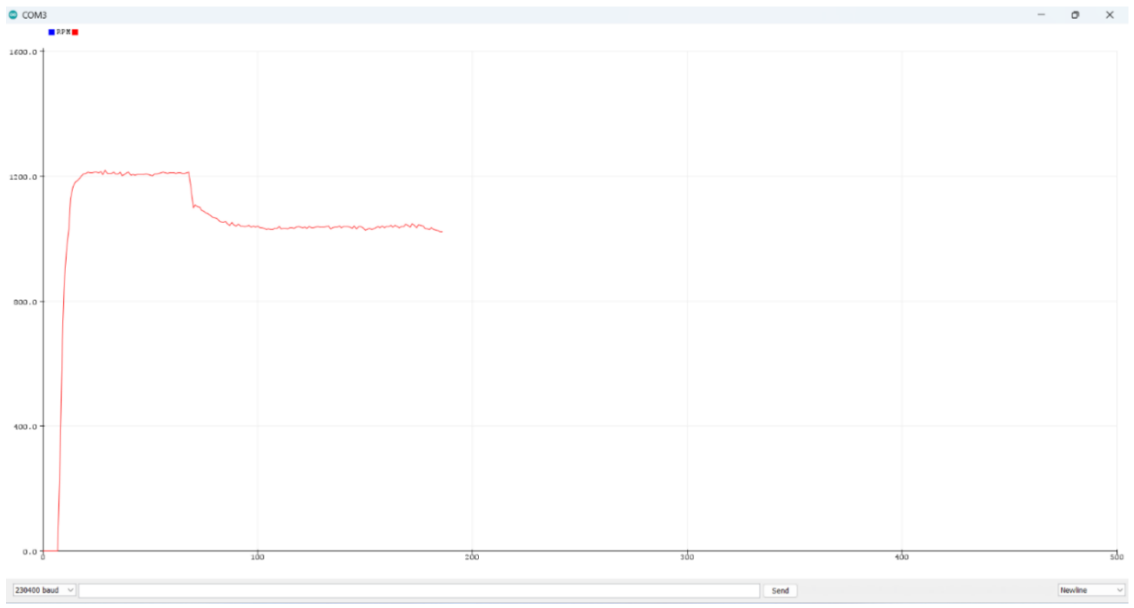


Figure 3.23 (b)

Output Response of Speed without PID Controller

Speed response of the BLDC motor at reference speed of 1200 RPM is shown in fig 3.24 (a). It show the typical constant speed curve of motor. If load is not applied on motor, it will continue to rotate at set speed.

For the set speed of 1200 RPM, if the load is applied using dynamometer, speed drops as shown in fig 3.24 (b) . For a given load speed is dropped from 1200 RPM to around 1100 RPM. This dropped speed will never come to set speed again until the load is removed. As there is absence of speed control algorithm, dropped speed will never get back to the set speed during loaded conditions. This shows the need and importance of speed control algorithm for speed control.

3.2.2 The Impact of PID controller

In these measurements, test runs were performed to tune the parameters of the PID controller. The control terms proportional and integral were selected, and derivative Term was disabled as in most real-life control systems. The PI controller was manually tuned by the following steps:

- Setting a small K_P value (preferably maximum PWM value/maximum motor speed) and K_I equal 0.
- Adding small K_I (preferably K_P /PID Sample rate) so that the system can correct the steady-state error.
- Manually increasing K_I to the maximum value that did not cause an overshoot.
- Adding small K_D so that to improve the settling time of system
- Increasing K_P so that the rise time was improved.

- Repeat the two previous steps until the rise time and settling time improvement was minor.

Figures 3.2 (a), (b), (c) below show the system responses with different value of K during tuning. Speed of motor changes when the analog output of DAC module changes and was captured on serial monitor. After tuning the PID controller, two target speed values were selected to test how effective it was at different speed ranges. The serial monitor pictures (from Figure 3.3 (a) to Fig 3.3 (c)) with the measurements displayed in the order to show the system rise time then settling time, with the PID.

A. Ref Speed = 1200 RPM, No Load.

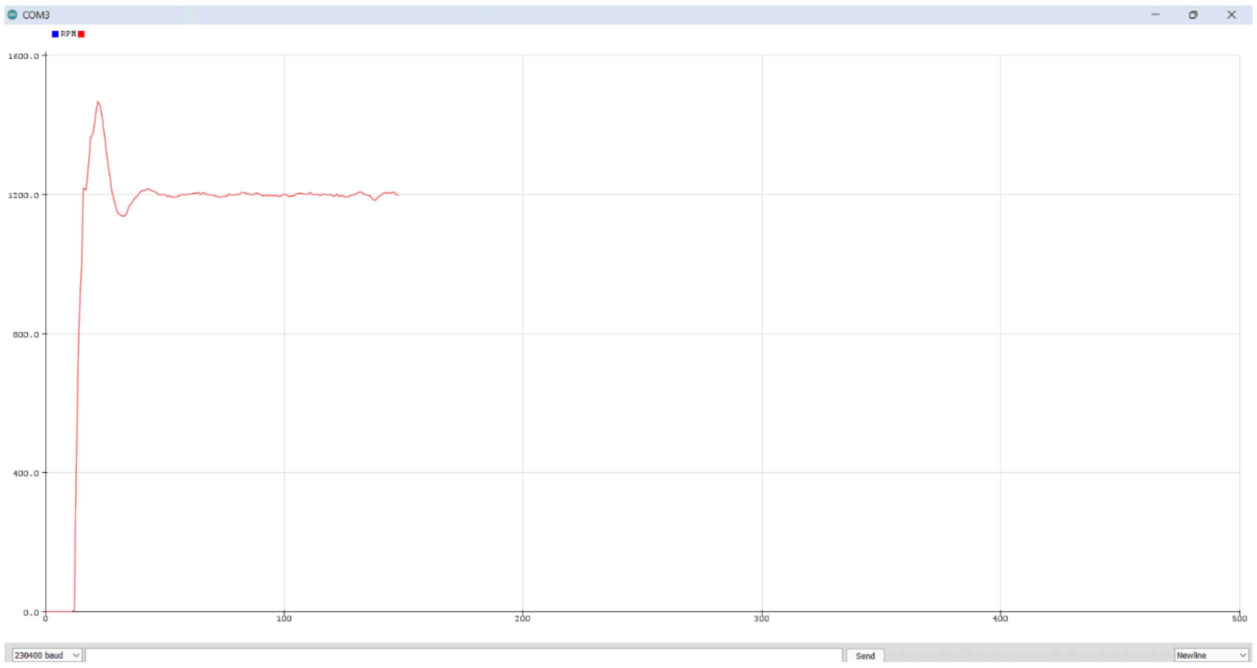


Figure 3.3 (a)

B. Ref Speed= 1200 RPM, 0.3 N-m Load.

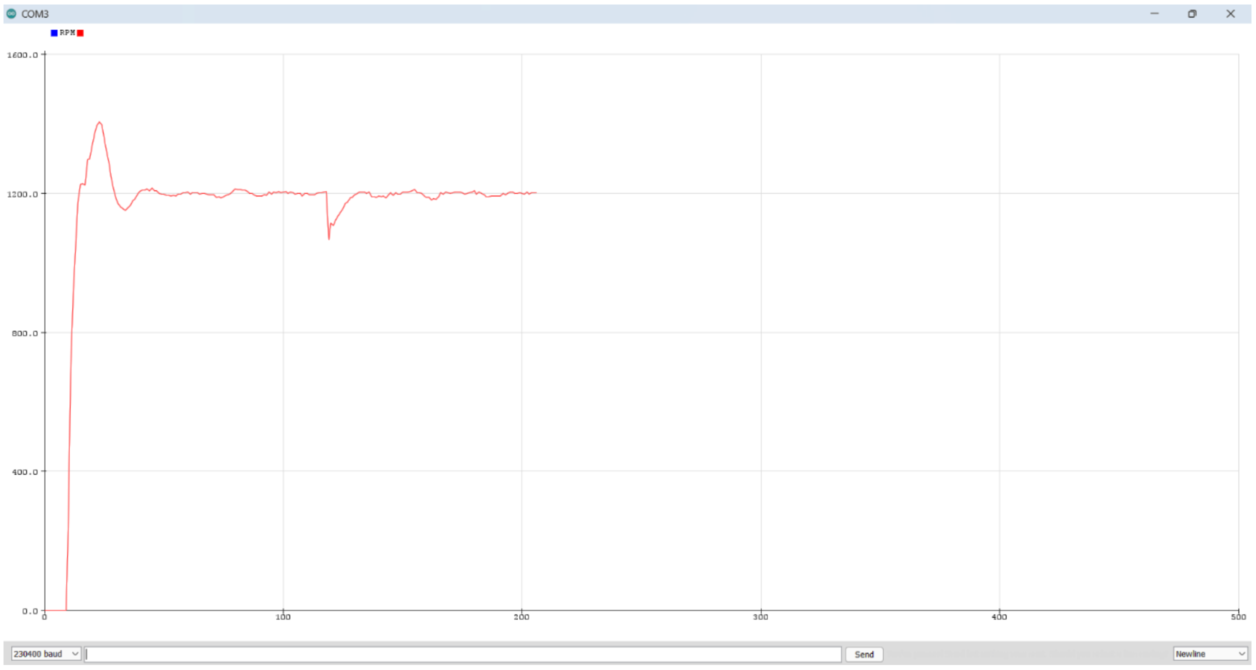


Figure 3.3 (b)

C. Ref Speed= 1100 RPM, 0.3 N-m Load.

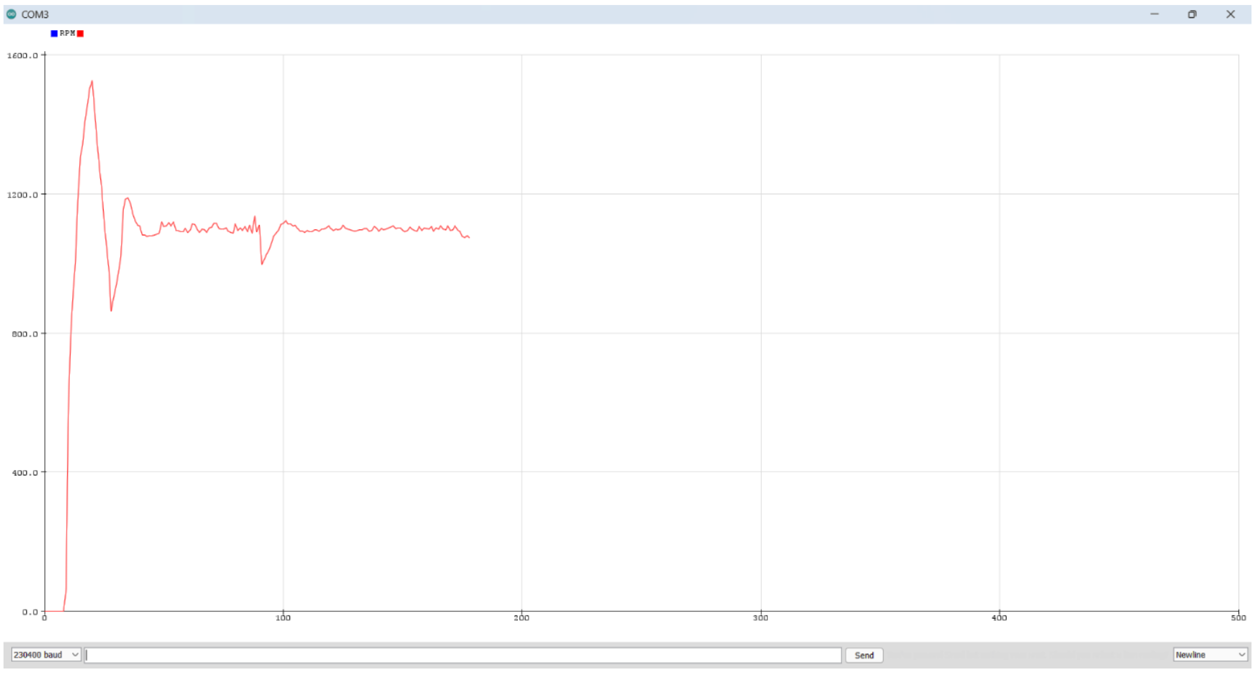


Figure 3.3 (c)

Output Response of Speed with PID Controller

Due to the implementation of PID motor settled at constant speed even after motor was loaded. For the case of set speed at 1200 RPM, the parameter values for the PID controller were K_P = 0.30, K_I = 0.8 and K_D = 0.2. The use of the PID control algorithm significantly reduced the rise time and settling time of the system after the loading of motor.

CHAPTER 4

CONCLUSION

The project implementing closed-loop speed control of a Brushless DC (BLDC) motor has been successfully accomplished. The closed-loop control system provides precise regulation and maintains the desired speed of the motor.

The speed control system employed a feedback loop that continuously measured the actual motor speed and compared it to the desired speed set point. This feedback information was used to adjust the DAC output voltage which then ensures that the motor operated at the desired speed regardless of external factors or load variations.

Key components of the closed-loop control system included Arduino UNO, motor controller, encoder to measure motor speed, PID controller, and a DAC to regulate motor throttle voltage. These components worked together to create a robust and reliable control system.

The project also involved careful calibration and tuning of the control parameters to achieve optimal performance. The PID controller was fine-tuned to provide the desired response characteristics, such as quick transient response, minimal overshoot, and steady-state accuracy.

The successful implementation of closed-loop speed control of the BLDC motor offers several advantages. It enables precise speed regulation, making it suitable for applications that require accurate speed control, such as robotics, automation systems, electric vehicles, and industrial machinery.

Overall, the project's successful implementation of closed-loop speed control for the BLDC motor demonstrates the feasibility and effectiveness of using closed-loop control techniques to achieve precise and efficient motor speed regulation. It opens up possibilities for a wide range of applications where accurate speed control is crucial, contributing to advancements in various industries.

CHAPTER 5

FUTURE SCOPE

5.1.1. Hardware-in-the-Loop (HIL):

As we delve into the future scope of our project, an essential aspect that holds immense potential and promises to work further is the integration of Hardware-in-the-Loop (HIL) system. For this purpose we need to establish a connection between the 3D experience software and the hardware setup.

Preliminary work was done using 3D experience software to establish connection between Arduino UNO board and 3D experience software where we executed the task of LED Blinking.

- **LED Blinking using Arduino UNO by HIL Technique**

For creating a digital twin of LED blinking in 3D Experience software, we have to use logical blocks in RFLP.

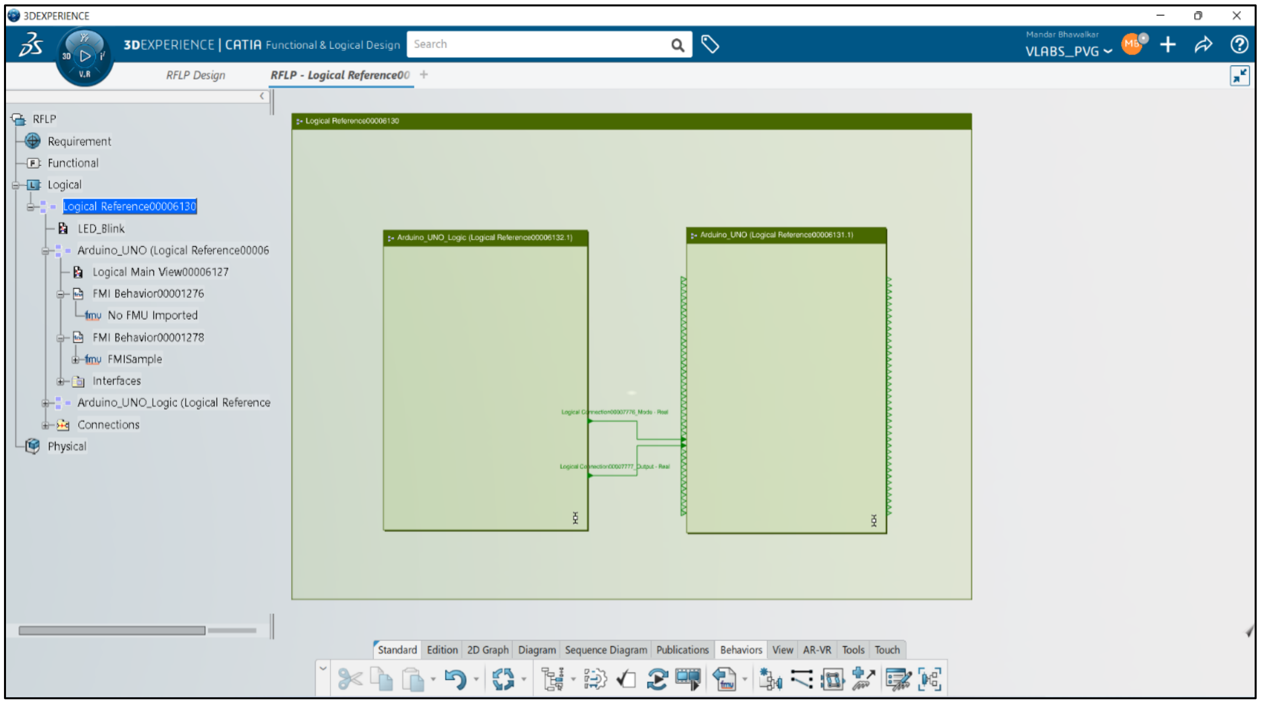

3DEXPERIENCE
3DEXPERIENCE | CATIA Functional & Logical Design
Search
Mandar Bhawalkar
VLABS_PVG
RFLP Design
RFLP - Logical Reference00
RFLP
Requirement
Functional
Logical
Logical Reference00006130
LED_Blink
Arduino_UNO (Logical Reference00006
Logical Main View00006127
FMI Behavior00001276
No FMU Imported
FMI Behavior00001278
FMISample
Interfaces
Arduino_UNO_Logic (Logical Reference
Connections
Physical
Logical Reference00006130
Arduino_UNO_Logic (Logical Reference00006132.1)
Arduino_UNO (Logical Reference00006131.1)
Standard
Edition
2D Graph
Diagram
Sequence Diagram
Publications
Behaviors
View
AR-VR
Tools
Touch

Figure 5.1 Implementation of LED Blinking by Hardware in Loop using 3D Experience Software

1. Firstly we need to add two logical blocks. One for the Arduino UNO and one for logic of the LED blinking logic.

2. We have to give behavior to each block. The Dymola behavior given to the block in which we have created the logic for Led blinking and FMI behavior for another block to replicating Arduino UNO.

3. The Arduino FMI has 82 connector pins. Each pin has two sub-pins, Mode and Value respectively. Each pin can be used in two ways, Input and Output. For each port pin, the user has to select Mode and Value.

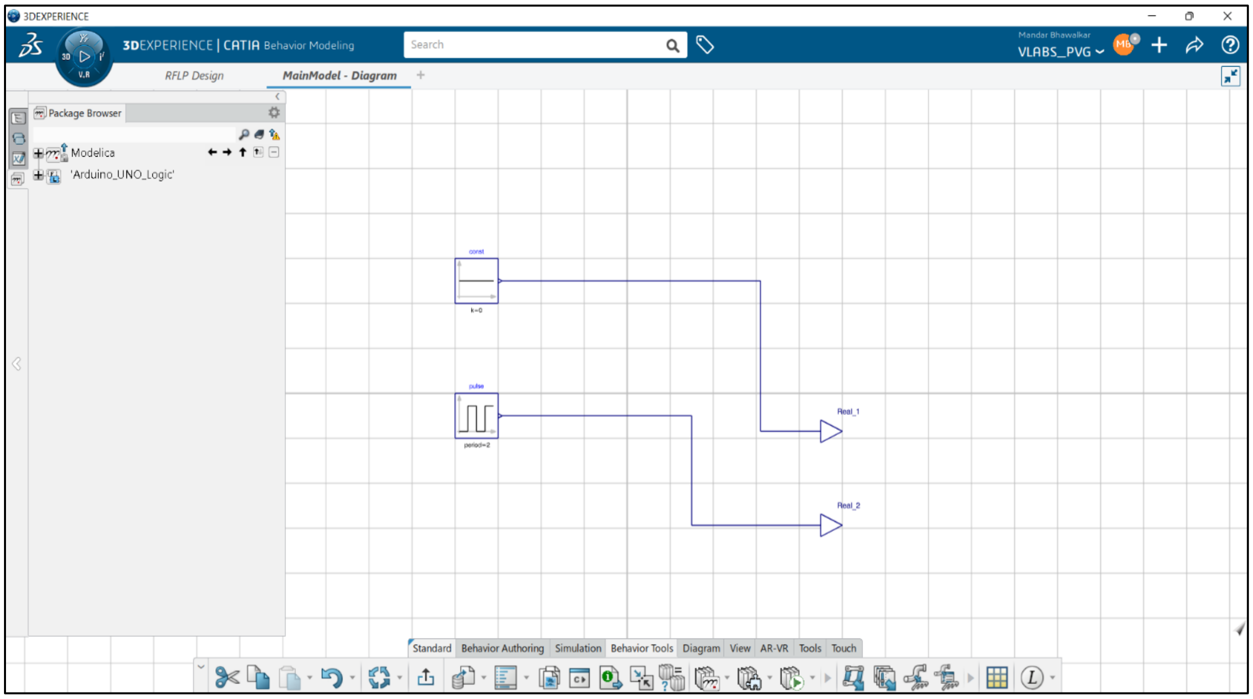


Figure 5.2 Logic for LED Blinking in Dymola

4. Dymola Behavior given to Logic Blocks will enable us to write a Modelica program for LED blinking.

5. For logic of LED blinking we have connected both sub pins of pin 13 of Arduino to the command block. Mode pin is connected to constant block which is used to configure the pin as output. Other sub pin is connected to pulse block in which have set the value of it to maximum i.e. 255 and width is kept to 50% as we have to continuously blink it.

In this way instead of directly uploading the sketch on to the Arduino board here we are giving the same logic through Modelica blocks.

5.1.2 Evaluation of Motor Performance by Applying Drive Cycle:

Drive cycles are applied to motors during testing to simulate real-world operating conditions and evaluate the motor's performance under different scenarios. Drive cycles allow for comprehensive performance evaluation of the motor. By subjecting the motor to different speed profiles, acceleration patterns, and load variations, we can assess how well the motor responds to varying operating conditions. This evaluation includes analyzing parameters such as speed control accuracy, torque output, power consumption, efficiency, and overall system stability. Applying drive cycles during testing allows for the validation and fine-tuning of PID control algorithm which we are using to regulate motor speed.