Department of Electronic and Computer Engineering

Electronic Design Project 3B (EDPB301)

Assessment Number 5

Final Report

Project Title: Revolution Counter

By

Group:		
Student Surname	**Initials**	**Student Number**
Pillay	**V**	**22008844**

Plagiarism Declaration

1. I know and understand that plagiarism is using another person's work and pretending it is one's own, which is wrong.

2. This report is my own work.

3. I have appropriately referenced the work of other people I have used.

4. I have not allowed, and will not allow anyone to copy our work with the intention of passing it off as their own work.

Pillay V	22008844	
Surname and Initials	Student Number	Signature

2 December 2022

Date

TABLE OF CONTENTS

Abstract

The aim and objective of the revolution counter project was to allow the user to control the speed and direction of two 12V DC motors and to display the difference in speed between the motors. The aim was to design a low-cost and highly efficient system which had the capability to sense revolutions made each motor. The design and construction of a GUI in MATLAB allows for the interactive aspect of the project to take place. The low-cost and high efficiency rate allows for the system to have an extended range of applications within industry in terms of measuring speed of the relevant industrial processes.

The statement of work includes a high number of engineering principles applied to design a revolution counter system which has a stable working operation. The statement of work includes the GUI construction within the MATLAB software. The construction and programming of relevant buttons to carry out the desired actions wanted to control the motors. The soldering and mounting of the motors had to be done to prevent it from moving around while it is in working operation. The statement of work also includes the PCB design of the motor driver circuit using the relevant software, the soldering of components and the correct wiring configuration to allow for speed and direction control of each motor. The relevant programming had to be done to allow for the functionality of the project to be a success in terms of interfacing the LM393 IR Count sensor, to allow for data computation to take place so the required data can be displayed. The construction of the project housing had to be done to mount the dc motors which will prevent them from moving around when power is applied. The statement of work done also includes the use of multiple testing procedures to acknowledge and improve the system's overall performance.

The results and conclusions reached includes the relative sensor measurements attained from the motion of the motors and the level of accuracy it has to offer. The overall stability and reliability levels would allow for the system to be applied within many industrial activities. It is conclusive the system is of an efficient design because of simplicity, accuracy, and cost-effectiveness.

List of Tables

Constants and Abbreviations

CHAPTER 1: Introduction

1.1 Overview

The project is based on a revolution counter. There project guidelines include the measurement of speed on two 12V DC (Direct Current) motors. It is noted that one motor should spin or rotate slower than the other and the difference in speed between the two should be displayed. This system would make use of a GUI (Graphical User Interface) in the MATLAB software which will be interfaced with an Arduino microcontroller to control and monitor the speed of the motors.

The project is made up of two main parts which are the GUI and the external circuitry. The GUI to be constructed in MATLAB would contain soft keys to start/stop and vary the direction and speed of each motor as well as display the speed of each motor and the difference in speed between them. An overview of the GUI in MATLAB is that it allows for the interaction with the system as well as receive results and compute it relevantly to be displayed on a plot. The external circuitry would be the Arduino microcontroller for computation of the relevant data to be displayed. The Arduino microcontroller would be connected to the motors via use of a motor driver circuit and to their relevant sensors. The variable speed control of the DC motors can be achieved via use of PWM (Pulse Width Modulation). This would require a driver circuit for the motors. An advantage to this would be the available IC (Integrated Circuit) packages with many channels to drive the two DC motors. The count sensor would produce a pulse train for the microcontroller to count.

The overview concludes that the system is a closed-loop system as the GUI interfaced with the microcontroller would vary the speed of the motors and the sensor would read and feedback data it receives. The system has a high level of efficiency because it is almost a fully digital system however the use of PWM is the only analogue electronic part of the system. The data received is in raw digital form as a high level (1) would be counted as one revolution and this is advantageous because it can be processed at high speeds and reduces the need of an ADC (Analogue-to-Digital Converter). The industrial applications are extensive for this system design as it is used to calculate the RPM (Revolutions Per Minute) of turbines, industrial engines, and machinery such as embroidery machines to count stitches, to count revolutions on the drivetrain connecting to the wheels of a car or truck and compute the average speed to display to the driver and many more industrial processes in terms of rotational motion. Other leisure applications include electronic fishing reels and treadmills.

1.2 Limitations of project

The revolution counter project limitations are shown in the list below:

1. Max speed count of system: There was a limit in terms of the system to be able to read high speeds of the motors in terms of the sensors capabilities of transmission.

2. High speed data processing: The data processing rate must be of a high level to match the system which operates at a high speed thus a powerful processor would be required or a different form of data transmission.

3. Data communication: The data that must be transmitted and received is done in real time. The option of implementing a different way of transporting data in real time would improve the systems overall performance in terms of a simplex communication connection (unidirectional) over a duplex communication connection (bidirectional).

4. Time: There was a limit towards the allocated time of the project deadline as there was minor delays in terms of resources.

5. Code: The limitations of the project in terms of the code was that it could not be complex because the system operates at high speeds and in real-time which can disrupt the program operation.

6. Cost: There was a limit to the cost of the revolution counter project because of demonstration purposes, however this had allowed for the innovation of cost-effective methods to be implemented within the project.

1.3 Benefits of project

The benefits of the revolution counter project are shown in the list below:

1. Cost: The overall cost of the project is low which allows for a demand in industry for its application. This benefit has been achieved via use of cost-effective techniques which was implemented in the design and construction of the project.

2. Digital system: The benefit of the system to be majority digital is that it eases the data acquisition process and the data computation process because there is no conversion of the data to or from analogue form which would increase the overall project efficiency.

3. Efficiency: The benefit of efficiency in terms of the project includes that it operates at a low voltage which would allow for low-power consumption because of implement a driver IC which can operate two DC motors simultaneously.

4. GUI: The GUI application within the MATLAB Development environment allows for accurate simulations of control systems in terms of the control centre. The benefit of the GUI acting as a control centre eases the interactive aspect of the system so that is user friendly.

5. Project Autonomous applications: The benefit of the revolution counter project includes the extended applications of the system to be implemented within many new fields of technology as it is majorly a digital system which enhances the digital age in terms of automation as the system can control the speed of processes independently and automatically via the aid of AI(Artificial Intelligence), (such as self-driving vehicles).

1.4 Broad plan of project

The broad plan of the project is an essential part of planning the project in terms of allocating time and effort towards each task in an efficient way to overall complete the project and to ensure that functional. The schedule of steps that make up the project include creating the GUI using the MATLAB software, assembling the necessary components, construction of external circuitry, programming the Arduino microcontroller and finally to interface the GUI with the Arduino microcontroller.
The table 1.4.1 below illustrates the main parts of the project.

Project	
GUI	
02-Sep	10-Sep
Component Assembly	
29-Aug	02-Sep
Construction	
02-Sep	30-Sep
Programming	
12-Sep	29-Oct
Interface GUI and Arduino	
05-Nov	22-Nov
Troubleshooting	
22-Nov	27-Nov

1.5 Preliminary Data

The preliminary data attained includes creating a MATLAB Graphical User Interface to control and display the speed of the DC motors.

This part of the project is the foundation because of the essential role it plays in the overall project. The relevant research had to be done in terms of interfacing the GUI with the Arduino microcontroller. The relevant research had to be done on the DC motors in terms of its working principle and the use of PWM (Pulse Width Modulation) to vary speed. The duty cycle percentage is proportional to the speed of the motor which is shown in Annexure A.

The use of PWM would control the speed of the motor which is dependent on the modulator voltage and duty cycle (k).

$$Duty\ Cycle\ (k) = \frac{Ton}{Ton+Toff} \times 100 \qquad [1.5.1]$$

The DC motor is advantageous because it is inexpensive and easy to control [2]. The use of an H-Bridge circuit shown in the Annexure A is used to change the direction of rotation of the motors.

$$Revolutions\ per\ minute\ (RPM) = 60\ \times Revolutions\ per\ second(RPS) \qquad [1.5.2]$$

The motor driver IC (L293D) is a dual full bridge driver which can drive two DC motors. This is a suitable and efficient component to embed within the project to drive the two motors. The sensor to be chosen would be the LM393 Photoelectric sensor. This sensor has two arms and can detect when an object is between them as the optical light axis is blocked. This sensor is chosen as it would allow for revolution counting to take place.

A disc with a slit would be attached to each of the motor's propellant shaft. The slit would allow for the optical light to pass every time the motor completes a full 360-degree rotation. This switching of the optical beam Infrared (IR) sensor allows the Arduino microcontroller to acknowledge the number of revolutions each motor makes. It is noted that each motor would need its own Photoelectric sensor to allow for the relevant to take place for each motor. This working principle is shown in Annexure A.

CHAPTER 2: Design

2.1 Detailed Specifications

The detailed specification of the project includes the programming flowchart, system's block diagram, the system's control system diagram, the system's circuit diagram and the list of relevant components. The GUI within MATLAB is the control centre of the project which would be needed to be constructed with the relevant soft keys and to be able to compute and display the data receive from the Arduino microcontroller (SEE ANNEXURE B). An addition of clockwise and anticlockwise soft keys would allow the change of direction of the motors. [3]

The GUI specification is shown in the table 2.1.1 below.

GUI soft keys	**GUI text window**
Start soft key (push-button)	Speed of each motor
Stop soft key (push-button)	Revolution counts of each motor
Speed soft key (slider-button)	Difference in speed between the motors

The two communication topologies are simplex (unidirectional) and duplex (bidirectional). A simplex communication connection allows for data to be transmitted in one direction only which is from a sender to a receiver. A duplex communication connection allows for data to be transmitted in both directions simultaneously which is from the sender to the receiver and from the receiver to the sender.

The duplex communication topology would make use of one Arduino Mega microcontroller which will be able to drive the two 12V DC motors via use of the LN293D motor driver IC as well as count the revolutions of each motor to compute and display the difference of speed between the two motors. The communication is thus bidirectional because the microcontroller is writing to the motors as well as reading from the sensors at the same time. The constraints associated with this bidirectional data communication is that it happens at a high speed and in real time, so it will require a powerful processor which can accommodate for that speed and size of data transmission. This would also lead to the MATLAB software to stop running or crash due to the high bidirectional data transmission speeds.

The simplex communication topology would make use of two Arduino microcontrollers in which both the MATLAB and Arduino IDE software would have to be integrated to carry out the system requirements. It is noted that one microcontroller would drive the two 12V motors and the other

microcontroller would read the pulses generated from the IR sensor. The communication is thus unidirectional because one microcontroller is transmitting data and one is receiving data . This indicates that one Arduino microcontroller will write to the motors and the other Arduino microcontroller will read from the sensors. The constraints associated with unidirectional data transmission includes an increase in the overall project cost. Refer to Annexure B which illustrates simplex and duplex communication topologies.

2.2 Programming Flowcharts

The programming flowchart is shown in Annexure C.

The programming flowchart illustrated in Annexure C depicts the relevant steps that will result in a functional program which will count the revolutions of each motor. The overall program does not make use of any timer libraries and only raw code has been implemented to improve code performance. The use of multiple live variables does take a toll on the overall performance as these variables are constantly being updated within real time. The rpm of each motor should be measured as well as the difference in rpm of the motors. The constraint of calculating these values would require an adequate amount of processing power as the duplex bidirectional method would allow the Arduino microcontroller to write and simultaneously read the pulse train received by the spinning of the motors. The necessary mathematics to return the rpm of each motor would be to calculate the duration of a single pulse generated. It is important for the microcontroller to digital read the input digital ports connected to the sensors to calculate pulse duration, the use of a current time variable and a previous time variable would be used to calculate the duration of one pulse as the microcontroller would be able to recognize (Negative Falling Edge ‘1’ to ‘0’) the state change of the sensor. The calculation is as follows:

$$RPM = \frac{1}{duration\ of\ pulse(\text{sec})} \times 60\ (\text{sec})$$

if duration of pulse = 0.2sec ; RPM =60/0.2.

RPM = 300

For millis() = *10^3 (60 000) and for micros() = *10^6 (60 000 000)

Pulse duration = Current time – previous time……(state change ‘1’ to ‘0’)

2.3 Block Diagram

The bidirectional and unidirectional system's block diagram is shown in Annexure D.

2.4 Control System Diagram

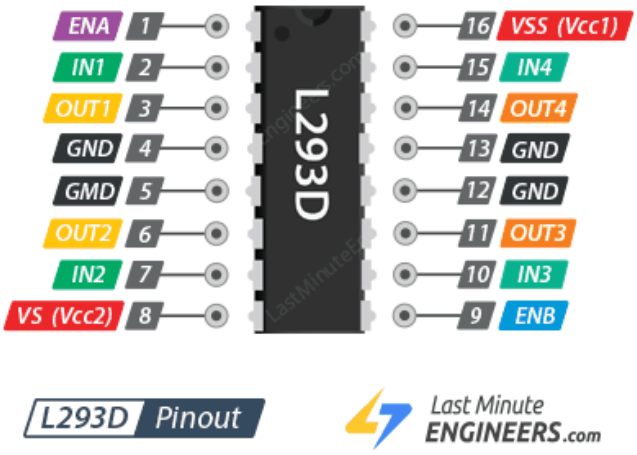
The system's control system diagram is shown in Annexure E.

2.5 Component Specifications

1. Arduino Mega microcontroller (SEE ANNEXURE G)

Operating Voltage(Logic Level)	5V
Input Voltage	7 to 12V
Digital I/O pins	54
Analog Input pins	12
PWM Output channels	15
Flash Memory	256KB (8KB bootloader)
SRAM/EEPROM	8KB / 4KB

2. One Dual H-Bridge motor driver ICs (SEE ANNEXURE F)

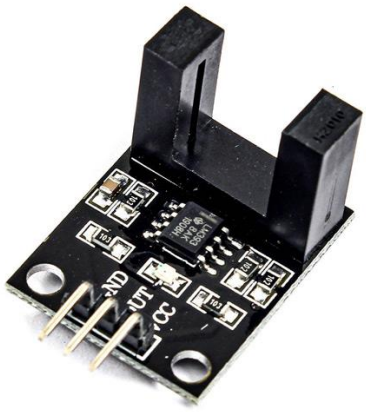


Motor Output Voltage	4.5V – 36V
Logic Input Voltage	5V
Output Current per channel	600mA
Peak Output Current per channel	1.2A

3. Two DC Motors (SEE ANNEXURE G)

Type of motor	DC Brush Motor
Operating Voltage	12V
Operating Current	800Ma
Speed	3000 RPM

4. Two Photoelectric IR(Infrared) sensors (SEE ANNEXURE F)



Optical coupling sensor grove width	5mm
Working Voltage	3.3V to 5V
Output form	Digital switch output (1 or 0)
PCB size	3.2cm x 1.4cm

The relevant components are required to carry out and fulfil the project for it to function successfully.

1. **Arduino Mega microcontroller (SEE ANNEXURE B)**

   The Arduino Mega microcontroller is used receive the signals from the sensors as well as supply the two PWM signals to drive the motors. This will be interfaced with the GUI created on MATLAB. The microcontroller is of an important role within the project as it is based in the centre of the project to relay data to MATLAB for computation. The microcontroller would have two digital port pins occupied to allow it to count the high/low logic levels (bits) which would represent the revolutions of each motor via the lm393 IR sensors This microcontroller unit will be supplied by make ELECTRONICS.

2. **One Dual H-Bridge motor driver ICs (SEE ANNEXURE A)**

   The L293D Dual H-Bridge motor driver IC is used to drive the two 12V DC motors. The IC makes use of PWM and an H-Bridge circuit to control the speed and direction of the motors. The IC can supply a current of 600mA through a channel and it operates at a voltage between 4.5V and 36V. This IC will be connected to the Arduino microcontroller to receive the two PWM signals required to drive each motor and implement the H-Bridge circuit to allow the user to vary the direction of rotation of the motors from anti-clockwise to clockwise or vice versa.

3. **Two 12V DC Motors (SEE ANNEXURE B)**

   The two 12V DC motors are used as the medium to test the systems capability to count the revolutions of each motor and to derive the speed of each motor to calculate the difference in speed between the motors using the Arduino microcontroller to receive this data and for the GUI developed in MATLAB to compute the data to display the speed, number of revolutions and the difference in speed between the motors.

4. **Two LM393 Photoelectric IR(Infrared) sensors (SEE ANNEXURE A)**

   The two LM393 Photoelectric IR sensors are used to count the revolutions of each motor. The sensor will be placed stationary and secured alongside the motor shaft which will have a disc with a 5mm slit. The slit would allow for the IR beam between the arms of the sensor to pass through causing the transistor within it to switch on resulting in a change of the logic level which the Arduino would recognize as one revolution through its digital port pins.

2.6 Circuit Diagrams (See Annexure H and Annexure I)

2.7 Motor Driver PCB Design

The software used to design the PCB was the NI Multisim and Ultiboard software. The Motor Driver PCB design consists of a single-sided PCB in which all components are through-hole. The PCB has been designed with one sixteen pin IC DIP socket holder and eight HDR male connectors on either side of the LN293D motor driver IC. The red lines depict the copper tracks which are under the board. (SEE ANNEXURE J)

2.8 Cost and Efficiency of project

Project Cost

No.	Components	Unit Price	Qty	Price	Supplier
1	Arduino Mega2560	R283.00	1	R283.00	make Electronics
2	12V DC Motor	R70.49	2	R140.98	Mantech
3	LN293D Dual H-Bridge IC	R47.50	2	R95.00	Mantech
4	LM393 Photoelectric IR count sensor	R35.70	2	R71.40	Mantech
5	Jumper leads (male to male)	R28.44	1	R28.44	Mantech
Total:	R618.82				

Efficiency

The project design is simple, yet it aims to be a stable system. The project programming which is the software side of the project will be complex with the creation of a GUI in MALAB and to interface it with the Arduino microcontroller. The revolution counter project had to make use of a digital system as the sensor chosen is of similar operation to fibre optic communication.

The sensor operation and its application to effectively measure the revolutions of the DC motor at high speeds would have make it difficult for the microcontroller to read every bit correctly as well as the overheating of the L293D IC. The essential note with respect to the two topologies, the bidirectional method would have a slower working operation and a low level of stability due to the strain put onto one microcontroller. The unidirectional method would offer more accuracy and stability at the price of two controllers. The project design is mainly derived from the BEng Tech knowledge, and it is noted that a vast amount of knowledge was applied to design an efficient system.

2.9 BEng Tech Knowledge

The BEng Tech knowledge attained from previous levels of study is listed below in terms of the role it plays within the revolution counter project:

1. Electronic Circuit Design 2A and 2B (ECDS201 and ECDS301): The knowledge of motors and the many electronic sensors used in industry. The skills of soldering and microcontroller usage was also attained.

2. Computer and IT (CPUT101): The introduction and familiarisation within the MATLAB development environment.

3. Fundamentals of Power engineering (FUPE201): The knowledge attained on PWMs applications and motor control.

4. Digital Electronics 1A and 1B (DGEA101 and DGEB101): The knowledge on Binary and Boolean values. The working principle of embedded transistor logic in packages and how memories and shift registers work as well as how to read IC’s.

5. Analogue Electronics 1A and 1B (ANLA101 and ANLB101): Circuit design techniques, voltage comparators, op-amps, and the knowledge of IC driven circuits.

6. Computer Programming 2A and Data Analytics and computation 2B (CPTP201 and CPTP301): Data computation techniques, Data handling using relevant software.

7. Fundamentals of Control Systems 2A and Control Systems 3A (FCNS201 and CYSA301): Control system techniques and using relative functions to improve the system stability via use of PID controllers and their tuning as well as design.

8. Engineering Mathematics 1A/1B/2A/2B (EMTA101, EMTB101, EMTA201 and EMTB201): The application of the complex mathematics used in programming and in the overall circuit design in terms of PWM calculations and sensor count calculations to relate with speed.

9. Engineering Physics 1A and 1B (EPHA101 and EPHB101) : The knowledge on light and the study of photons and their applications within the field of electronics in relevant sensors and fibre optic communication network systems.

10. Technical Literacy 1B (TELC101): Report format and the use of various software and extended knowledge on Microsoft word and the necessary skills to produce a technical report.

11. Electronic Design Project 3A (EDPA301): The familiarisation on how the process of project orientation is carried out and the attributes needed to be applied in this module.

12. Electrical Principles 1A and 1B(ELEP101 and ELEP201): The knowledge on the fundamentals of electricity and components necessary for electronic design techniques and the necessary calculations and laws of electricity.

13. Fundamentals of Signals and Systems (FCMC201): The analysation of various signals which are applied in the PWM signal and duty cycle percentage. The knowledge on creating plots with signals to display relevant data of the system.

CHAPTER 3: Construction

3.1 Project wiring configuration

LN293D Motor Driver IC

Component Pin	Arduino Pin
Pin 1 (Enable Motor 1)	~D9 (PWM)
Pin 2 (Input 1)	D8
Pin 3 (Output 1)	Motor 1 (-)
Pin 4 (GND)	GND
Pin 5 (GND)	GND
Pin 6 (Output 2)	Motor 1 (+)
Pin 7 (Input 2)	D7
Pin 8 (VS Vcc 2 (+12V))	+12V
Pin 9 (Enable Motor 2)	~D3 (PWM)
Pin 10 (Input 3)	D5
Pin 11 (Output 3)	Motor 2 (-)
Pin 12 (GND)	GND
Pin 13 (GND)	GND
Pin 14 (Output 4)	Motor 2 (+)
Pin 15 (Input 4)	D4
Pin 16 (VSS Vcc 1(+5V)	+5V

LM393 IR Count sensor 1

Component Pin	Arduino Pin
+5V	+5V
GND	GND
OUT	D12

LM393 IR Count sensor 2

Component Pin	Arduino Pin
+5V	+5V
GND	GND
OUT	D13

3.2 PCB Construction (SEE ANNEXURE K)

3.3 GUI construction (SEE ANNEXURE L)

3.4 Motor and Sensor mounting (SEE ANNEXURE M)

3.5 Project Housing Construction (SEE ANNEXURE N)

3.6 Code Construction (SEE ANNEXURE O)

Bidirectional code (MATLAB only/ one microcontroller) : SEE ANNEXURE Q

Unidirectional code (MATLAB and Arduino/ two microcontrollers): SEE ANNEXURE R

CHAPTER 4: Testing

Motor test

Duty Cycle (k)	Average Voltage (V)	Average Speed (RPM)
0%	The test includes recording the average voltage value in correlation with the relevant duty cycle percentage.	The test includes recording the average speed of the motors in correlation with the relevant duty cycle percentage and average voltage.
25%		
50%		
75%		
100%		

LN293D test

PWM	Motor control test procedure
Pin 1 (Enable Motor 1)	The process to test for the variation of motor 1 speed
Pin 9 (Enable Motor 2)	The process to test for the variation of motor 2 speed
H-Bridge	
Pin 2 and Pin 7 (IN1 AND IN2)	The process to test for the variation of motor 1 direction
Pin 10 and Pin 15 (IN1 AND IN2)	The process to test for the variation of motor 2 direction

GUI test

Button	Test Procedure
Clockwise Button (Motor 1)	The GUI test is implemented to test the functions of the buttons if they carry out their respective tasks in terms of controlling the speed and direction of Motor 1 and Motor 2
Anticlockwise Button(Motor 1)	
Stop Button (Motor 1)	
Slider button (Motor 1)	
Clockwise Button (Motor 2)	
Anticlockwise Button(Motor 2)	
Stop Button (Motor 2)	
Slider button (Motor 2)	

LM393 IR Sensor test

IR Sensors	Test Procedure
IR Sensor 1: Output pin connected to digital port D12	To test if the sensor generates a pulse train so the Arduino microcontroller can count the High Logic Level States (+5V) received from the rotation of motor 1.
IR Sensor 2: Output pin connected to digital port D13	To test if the sensor generates a pulse train so the Arduino microcontroller can count the High Logic Level States (+5V) received from the rotation of motor 2.

PCB test

Test Type	Output
Soldering	The test on the soldering of the LN293D Motor Driver IC as well as the relevant connector pins in terms of stable soldering and conductivity to reduce power loss.
Open or Short Circuits	The test in terms of using a multi-meter (continuity function) to identify an open or short circuit within the Motor Driver PCB design.
PCB functionality	The test in terms of PCB functionality refers to the efficiency of the PCB within the project in terms of improving wiring configuration thus easing the troubleshooting process.

System Operation test

The system operation test includes testing the results displayed from the system. This includes the rpm of motor 1, the rpm of motor 2 and the difference in rpm between motor 1 and 2. A plot of the difference in rpm will be returned when the system is in operation. The system operation test is to visualize the signals received from the IR sensors to ensure that they are in working operation and to allow MATLAB and the Arduino IDE to compute and plot the data in real time. This would allow for functionality of the overall project in terms of accuracy and stability to be of a moderate level.

RPM Motor 1	display
RPM Motor 2	display
RPM difference	Display and plot

CHAPTER 5: Results

Motor test results

Duty Cycle (k)	Average Voltage (V)	Average Speed (RPM)
0%	0V	0
25%	3V	750
50%	6V	1500
75%	9V	2250
100%	12V	3000

LN293D test results

PWM	Motor control	Result
Pin 1 (Enable Motor 1)	The process to test for the variation of motor 1 speed	Allowed for the user to control the level of speed at which motor 1 rotates in terms of varying the duty cycle of the PWM signal.
Pin 9 (Enable Motor 2)	The process to test for the variation of motor 2 speed	Allowed for the user to control the level of speed at which motor 2 rotates in terms of varying the duty cycle of the PWM signal.
H-Bridge		
Pin 2 and Pin 7 (IN1 AND IN2)	The process to test for the variation of motor 1 direction	Allowed for the user to control the direction of Motor 1
Pin 10 and Pin 15 (IN1 AND IN2)	The process to test for the variation of motor 2 direction	Allowed for the user to control the direction of Motor 2

GUI test results

Button	Output Result
Clockwise Button (Motor 1)	Motor 1 rotates in a clockwise direction.
Anticlockwise Button(Motor 1)	Motor 1 rotates in an anticlockwise direction.
Stop Button (Motor 1)	Motor 1 stops rotating/spinning.
Slider button (Motor 1)	Motor 1 experiences a proportional change in speed in terms of slider level.
Clockwise Button (Motor 2)	Motor 2 rotates in a clockwise direction.
Anticlockwise Button(Motor 2)	Motor 2 rotates in an anticlockwise direction.
Stop Button (Motor 2)	Motor 2 stops rotating/spinning.
Slider button (Motor 2)	Motor 2 experiences a proportional change in speed in terms of slider level.

LM393 IR Sensor test results

IR Sensors	Function	Output
IR Sensor 1: Output pin connected to digital port D12	digitalRead(D12)	IF Rotary Disc Blocks IR Beam: Low Logic level(+0V) IF Rotary Disc Slit allows IR Beam: High Logic Level(+5V)
IR Sensor 2: Output pin connected to digital port D13	digitalRead(D13)	IF Rotary Disc Blocks IR Beam: Low Logic level(+0V) IF Rotary Disc Slit allows IR Beam: High Logic Level(+5V)

System Operation Results (SEE ANNEXURE P)

5.2 Problems and solutions

The expected problems to be encountered within the project with their relevant solutions is shown the table 5.2.1 below.

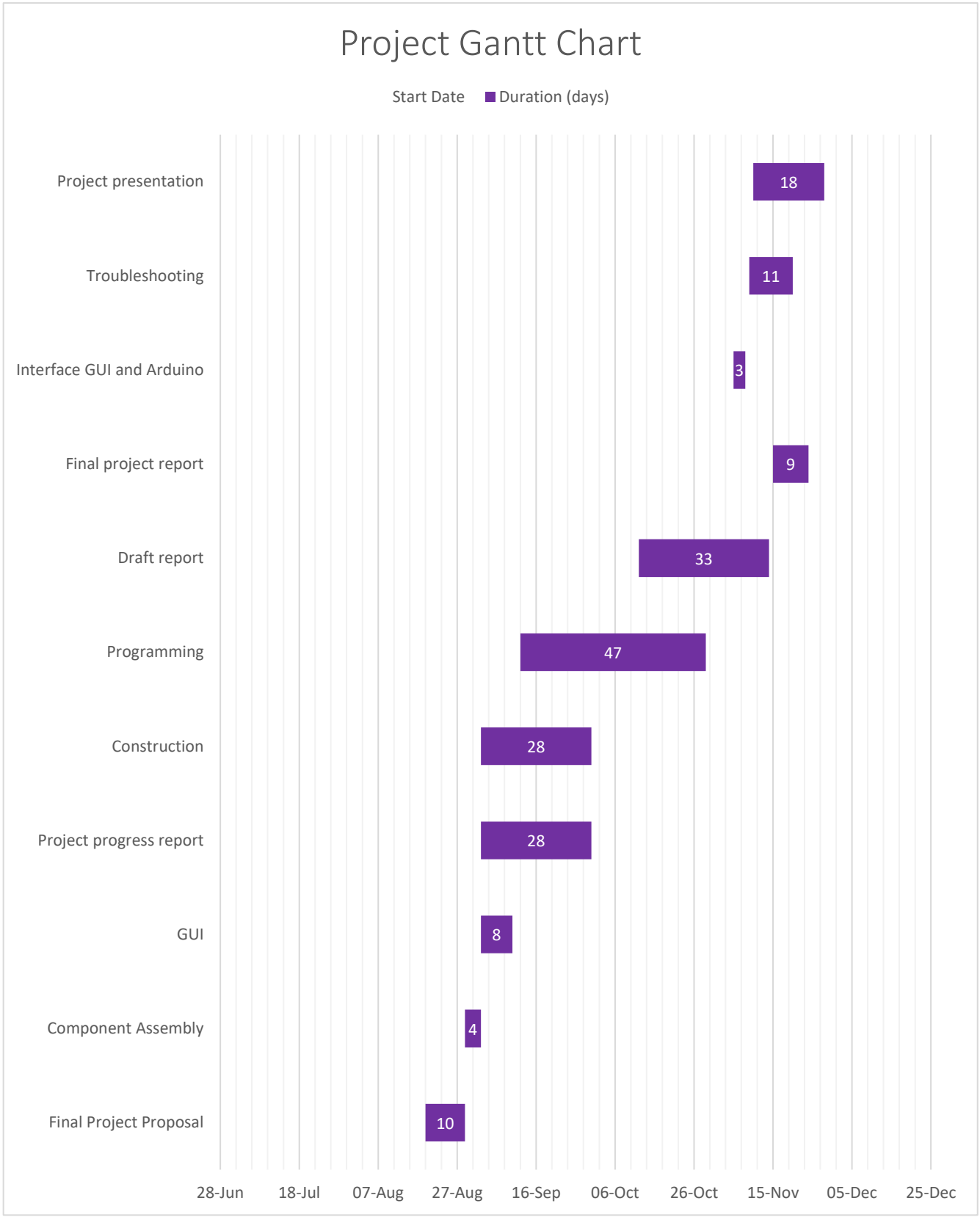
Problems	**Solutions**
The overall code of the project is a crucial part towards the system's functionality. The problem expected can be lagging while running since the use of live variables and this in turn would cause inaccurate measurements.	The solution to this problem would be to make use of functions which will ease the flow of the program. Another solution to this problem is to keep the code concise so that it does not take unnecessary storage space as well as RAM (Random Access Memory).
The LM393 Photoelectric IR sensors may have a problem with the detection of the notch on the rotating disc which will cause inaccurate measurements during the system's process. The sensor works off an IR beam between the two arms and if dirt or dust blocks the path of the beam it will result in the counting operation to stop working.	The solution for the notch would be to test various sizes which allow the beam to pass effectively through it to sense a high logic state bit ['0'] or ['1'] for the microcontroller to acknowledge the rotation accurately. The solution to prevent dust or dirt is to enclose the sensor module and allow enough air flow to prevent the settlement of dust.
The problem with the 12V DC motor can be the fast breakage of the brush within in it because of its inexpensive price and cheap manufacturer efforts.	The solution is to acquire a high quality 12V DC motor which will be able to suit this application in terms of being PWM driven. It is also suited to keep the operating voltage of the motor a good safety percentage away from its maximum.
The L293D IC operates at a voltage of 4.5V to 36V and provides 600mA of current per channel and a peak output current of 1.2A. Since this high operating power usage is dependent on the motors the IC can relatively overheat fast which can cause it to malfunction.	The solution to this problem is to implement a suitable motor with the optimum power specifications which would prevent the IC from operating at high current causing it to overheat. A heat sink would also be a solution to this problem as it would absorb the heat dissipated by the IC.
The problem associated with the Arduino UNO microcontroller is that it may not have enough storage space as well as RAM to carry out the tasks of the system in terms of reading live data from the photoelectric IR sensor because it must keep up with reading the speed of both motors.	The solution includes the usage of an Arduino Mega2560 because of its greater storage space (256KB) as well as more powerful RAM (Random Access Memory (SRAM: 8KB EEPROM: 4KB) which would meet the demands of the system comfortably. However, for efficiency within the cost of the project an Arduino UNO will be tested to see if it can meet the needs of the system.

5.3 Time Management

The time management of the project is an essential part of planning the project in terms of allocating time towards each task in an efficient way to overall complete the project and to ensure that it is functional. The schedule of steps that make up the project include creating the GUI using the MATLAB software, assembling the necessary components, construction of external circuitry, programming the Arduino microcontroller and finally to interface the GUI with the Arduino microcontroller.

The time management of the assessments are just as essential and must simultaneously be progressive alongside the project. A total of six assessments makes up the overall project and a considerable amount of time should be allocated to each assessment proportionally to its weighting. The table 5.3.1 below illustrates the two main parts of the project and their subtopics which will make up the complete project.

Project	**Assessment**
GUI 02-Sep | 10-Sep	Final Project Proposal 19-Aug 29-Aug
Component Assembly 29-Aug | 02-Sep	Project progress report 02-Sep 30-Sep
Construction 02-Sep | 30-Sep	Draft report 12-Oct 14-Nov
Programming 12-Sep | 29-Oct	Final Project Report 15-Nov 24-Nov
Interface GUI and Arduino 05-Nov | 22-Nov	Project presentation 10-Nov 28-Nov
Troubleshooting 22-Nov | 27-Nov	


Project Gantt Chart
Start Date
Duration (days)
Project presentation
Troubleshooting
Interface GUI and Arduino
Final project report
Draft report
Programming
Construction
Project progress report
GUI
Component Assembly
Final Project Proposal
18
11
3
9
33
47
28
28
8
4
10
28-Jun
18-Jul
07-Aug
27-Aug
16-Sep
06-Oct
26-Oct
15-Nov
05-Dec
25-Dec

The time management of the project must be efficiently carried out to prevent any errors from occurring during the construction process. The project must be well-organised to allow for its functionality. The organisation of time duration is essential in terms of ensuring the project progress is acceptable. The time management of tasks that are to be done to contribute to the final project should be separated efficiently and effectively. The organisation of the required materials should be well-kept and applied in the most efficient and cost-effective way. The acquisition of materials would be acquired from MANTECH Electronics PTY Ltd and make ELECTRONICS.

The time management of the above factors is crucial towards constructing a successfully functioning project that would meet the given deadlines. It is important to manage the time given efficiently as the excess time would be needed to troubleshoot any problem areas or in turn offer room for improvements within the project.

Project Improvements

The addition of an LCD (Liquid Crystal Display) to display the RPM of each motor as well as the difference between the motors. The LCD interface is illustrated below.

The analysis, modelling, optimisation, design, implementation, and investigation of the unidirectional system was to offer more accuracy and stability because the route of communication is in one direction via use of the two microcontrollers as one will control the motors via the GUI (Writer) and the other microcontroller (Reader) will acquire and display the data received (rpm1, rpm2) from the IR sensors and to finally compute the data to calculate and display the difference in RPM.

The analysis, modelling, optimisation, design, implementation, and investigation of the Bidirectional system was to offer more power to the user as the system uses a single microcontroller which would run all the system operations. A broadly defined and enhanced GUI was created to meet all the requirements of the project and to allow for the control of the motors as well as displaying the RPM values of motor 1 and motor 2. The difference in RPM can be displayed on a plot by setting the rpm and count as a vector. [i rpm]

CHAPTER 6: Conclusion

The key-points of the project includes a low-cost system using an Arduino microcontroller and the MATLAB software. The overall high level of efficiency of the system is derived from the design of the revolution counter project. The LN293D motor driver IC plays a vital role in terms of allowing the user to control the speed of each motor as well as the direction of rotation of each motor. The overall revolution counter project is a simple design but has a high level of stability. There was a number of engineering principles which was applied to allow for the correct functionality and construction of the project in terms of PCB construction, code construction, housing construction and relevant wiring configuration.

The decisions made within the project includes the selection of components with the necessary capabilities to allow the project to count the revolutions of each motor as well as to compute the difference in speed between the motors. The decision of using the LM393 IR Count sensor within the revolution counter project is because it is cheap and accurate in terms of receiving and sending the relevant pulse train signal. The decision of using the LN293D Motor Driver IC was because of the cost-effectiveness of the components in terms of being able to drive the two motors simultaneously.

The outcomes and observations of the project includes a digital speed measurement system which can reach and read high levels of speed for computation in real-time. The observation of the GUI allows for user-friendly control of the system. The outcome of the project allows for the system to be applied in many real-world applications which makes use of a digital system which allows for the data to be processed and displayed at high speeds. The outcome of the unidirectional system with the use of two microcontrollers produced more accurate results than the bidirectional system. It had also proven to be a more stable system as the processing power was adequate to run system applications.

The recommendation of the project includes improving the overall accuracy rate of the system and reducing the overall error rate of the system. This can be achieved using a more accurate speed sensor such the encoder sensors. The accuracy of the data received can be improved by the ability of an efficient low-pass filter. The addition of PID tuning would benefit the system by increasing its accuracy as well as reducing the error rate of the system.

REFERENCES

[1] MANTECH ELECTRONICS PTY LTD, "Mantech," MANTECH, [Online]. Available: https://www.mantech.co.za/default.aspx [Accessed 19 September 2022].

This reference was used because it is the main supplier of the electronic parts needed to construct the revolution counter project.

[2] Maker.io Staff, "digikey.in," maker.io, 29 3 2018. [Online]. Available: https://www.digikey.in/en/maker/blogs/2018/which-arduino-is-best-for-your-project [Accessed 27 September 2022].

This reference was used to aid with the decision of choosing the appropriate microcontroller for application within the project to ensure that its functionality is a success.

[3] make ELECTRONICS, "make.net.za," make ELECTRONICS, 2022. [Online]. Available: https://make.net.za/product/arduino-mega2560-ch340-clone/?utm_source=Google%20Shopping&utm_campaign=General%20Product%20Feed&utm_medium=cpc&utm_term=5923&gclid=CjwKCAjwx7GYBhB7EiwA0d8oeyzmEjREXthEp7RSIQTXfRTBWaiXKF8f-RhhwsRB1w28l4nnvSakKRoCT7oQAvD_BwE [Accessed 27 September 2022].

This reference was used as this is the supplier of the Arduino Mega microcontroller as Mantech supplies the same microcontroller for two times the price. It is also seen there are cheaper sensors so it is a well recommended electronic parts supplier for an efficient price.

[4] M. Damirchi, "electropeak.com," ElectroPeak.Inc, 2019. [Online]. Available: https://electropeak.com/learn/interfacing-lm393-infrared-speed-sensor-with-arduino/ [Accessed 2 October 2022].

This reference was used to gather information on the LM393 IR count sensor in terms of its specifications, working principle and to interface it with an Arduino to compute its data.

[5] Last Minute Engineers, "lastminuteengineers.com," Last Minute Engineers, 2022. [Online]. Available: https://lastminuteengineers.com/l293d-dc-motor-arduino-tutorial/ [Accessed 4 October 2022].

This reference is used as a major source of information within the project because of its similarity towards the idea of the revolution counter project in terms of driving two DC motors.

[6] P. Khatri, "DC Motor Control Using MATLAB and Arduino," circuitdigest, 3 October 2018. [Online]. Available: https://circuitdigest.com/microcontroller-projects/matlab-dc-motor-control-using-arduino [Accessed 12 October 2022].

This reference was used to gather knowledge on MATLAB in terms of the GUI that is suitable for the application of the DC motors control of speed and direction.

[7] ambhatt, "project hub," arduino, 16 February 2020. [Online]. Available: https://create.arduino.cc/projecthub/ambhatt/dc-motor-speed-control-and-measurement-afff68 [Accessed 2 November 2022].

This reference was used to gather information on interfacing the motors with the Arduino to allow for control over its speed and direction as well as to relay the data to MATLAB.

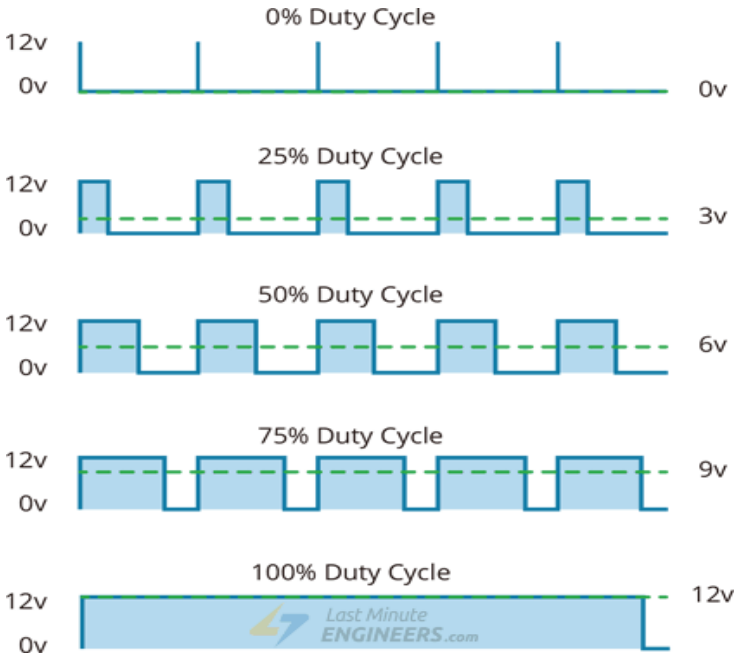
[8] "Using MATLAB and Arduino for Motor Control," MathWorks, [Online]. Available: https://www.mathworks.com/videos/using-matlab-and-arduino-for-motor-control-100737.html [Accessed 11 November 2022].

This reference was mainly used to attain knowledge of implementing the MATLAB GUI to control the two 12V DC motors.

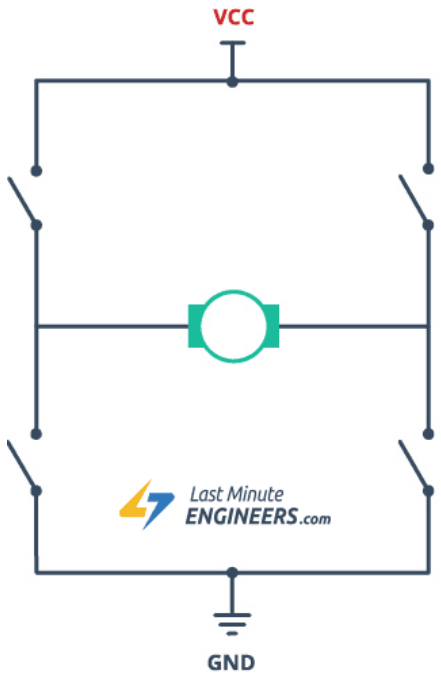
[9] "Help Center," MathWorks, [Online]. Available: https://www.mathworks.com/help/physmod/sps/ug/example-modeling-a-dc-motor.html [Accessed 6 August 2022].

This reference refers to the help centre within the MATLAB development environment which will aid users on creating a GUI and how to interface it with various other things in terms of electronic circuitry as well as its simulation.

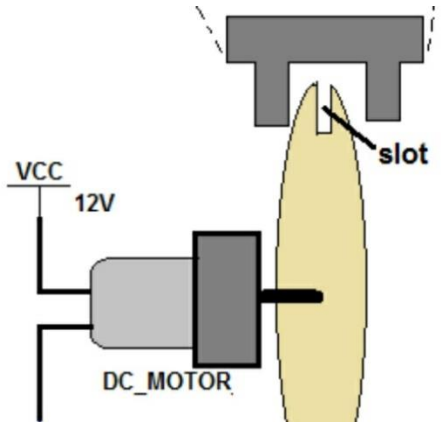
ANNEXURE A (Preliminary data)



PWM working principle in terms of varying duty cycle (k) to allow for the control of the motor speed.

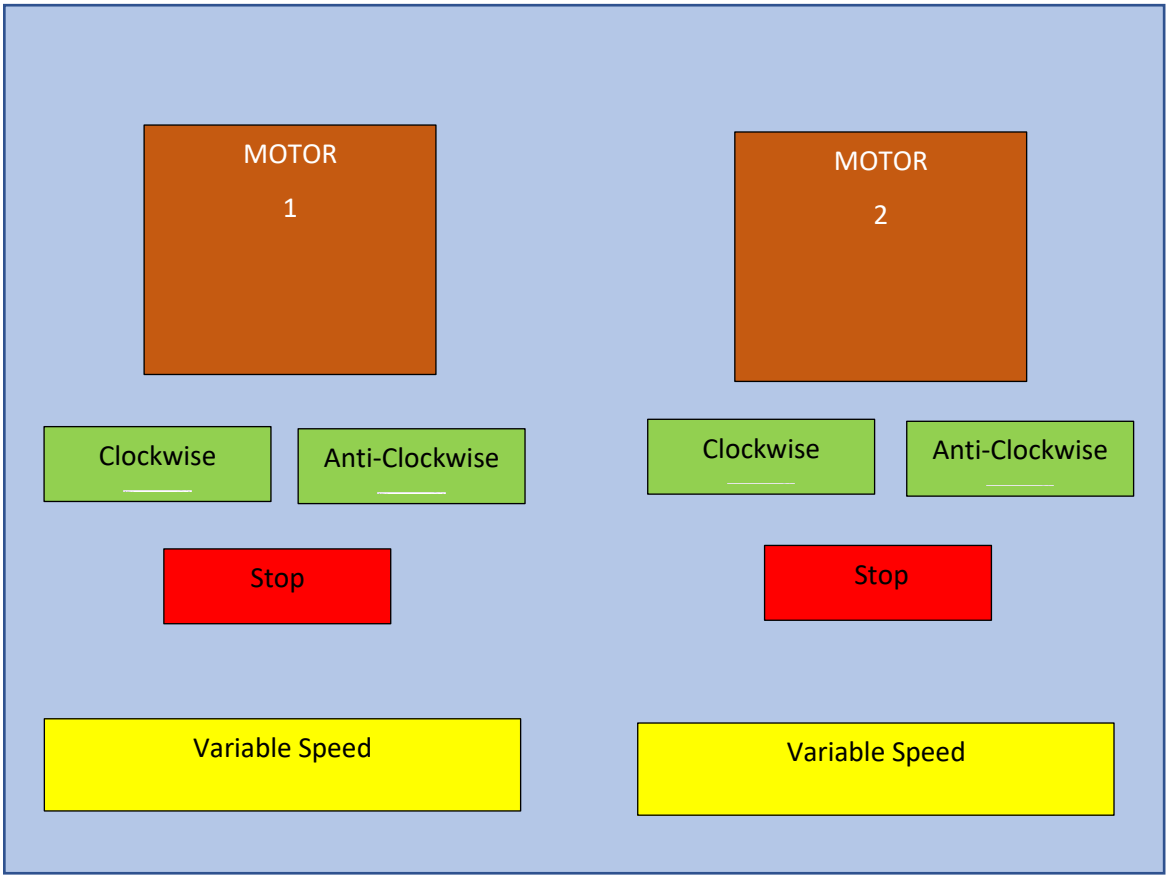


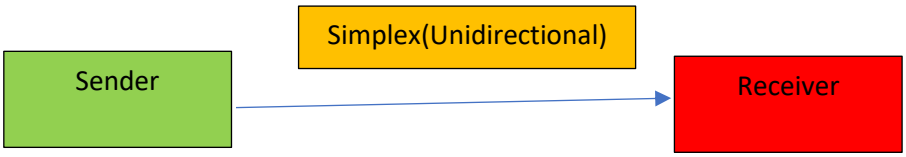
The H-Bridge circuit working principle is illustrated on the left in terms of using switches to change polarity thus allowing for the control of the motors .direction.

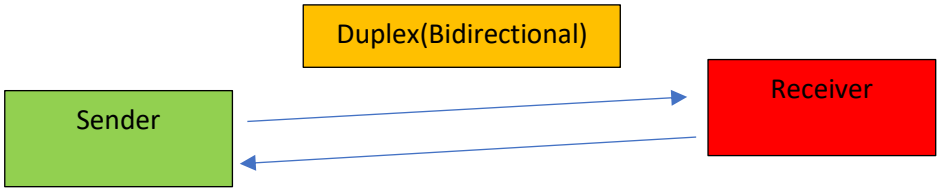


The system speed measurement design is illustrated on the left which depicts how the sensor would measure the speed of the motor.

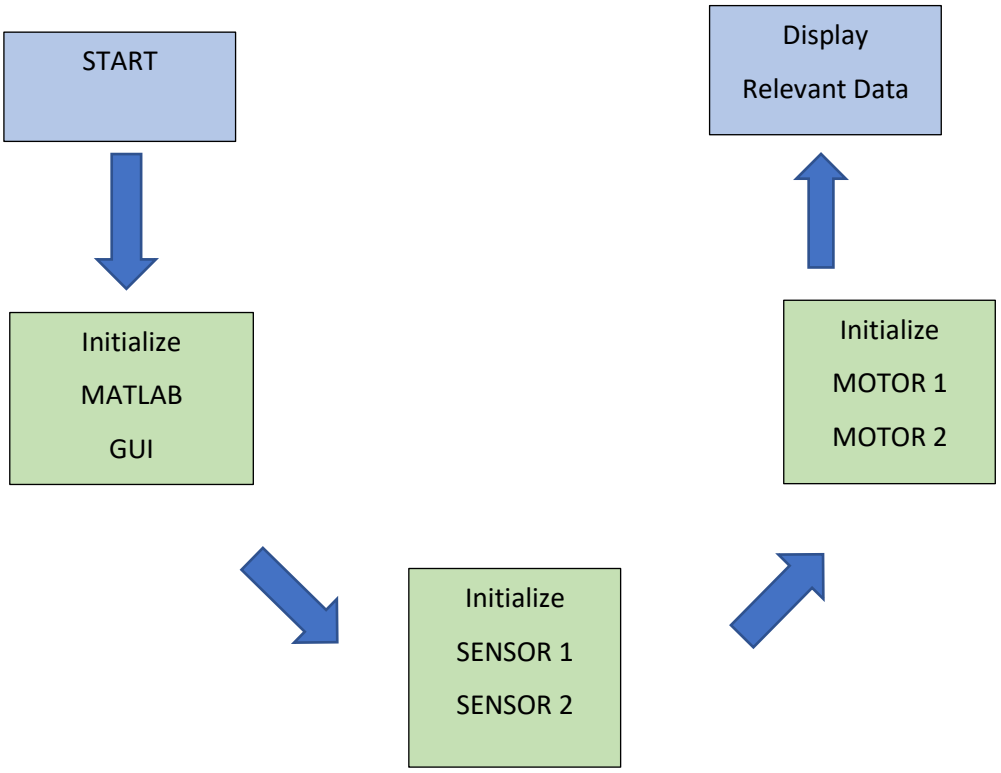
ANNEXURE B (Design)





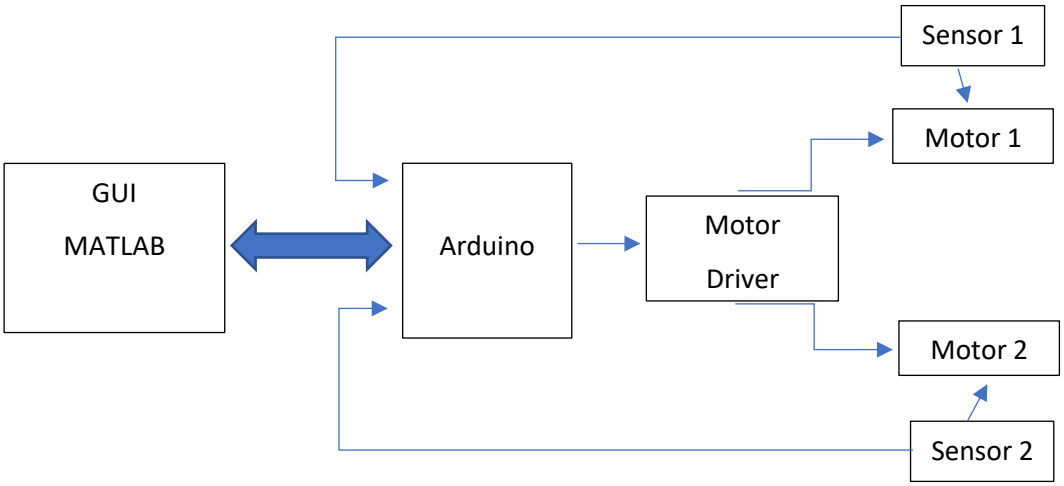


ANNEXURE C (Programming flowchart)

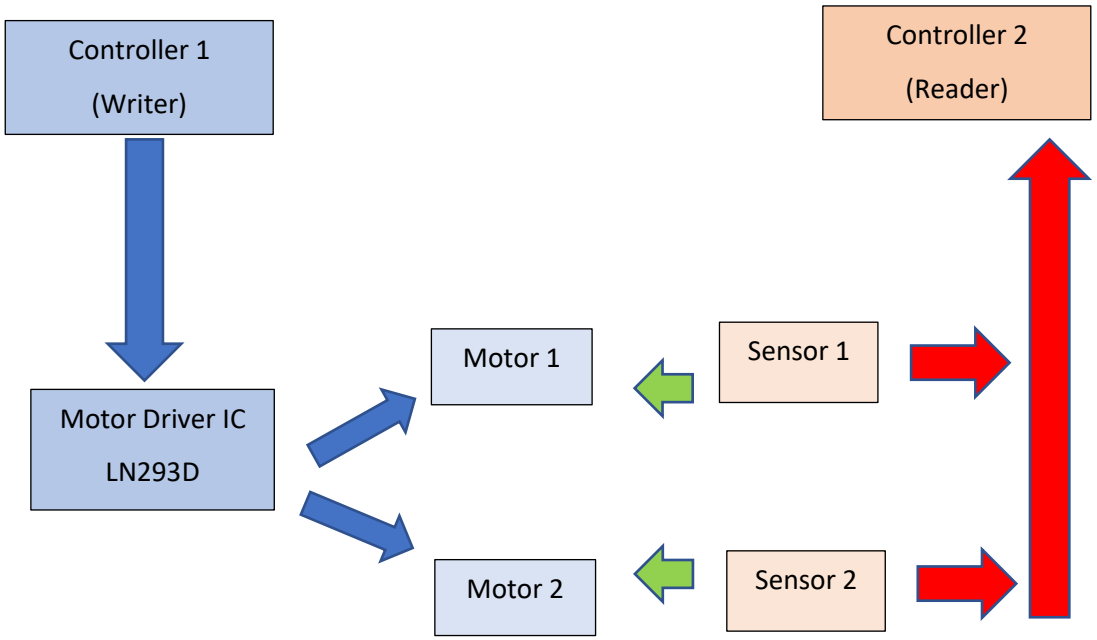


ANNEXURE D (Block diagrams)

Bidirectional System Block diagram

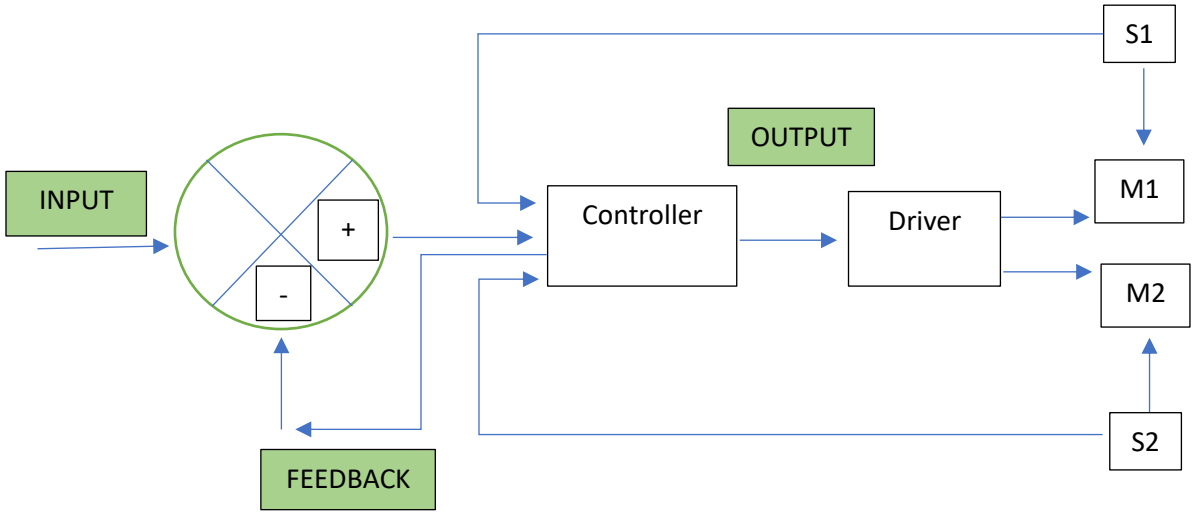


Unidirectional System Block diagram



ANNEXURE E (Control system diagram)

Control system diagram



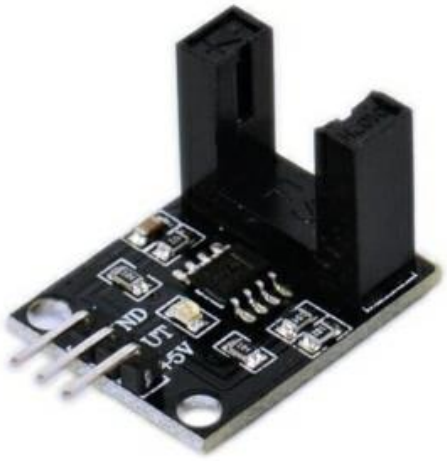
ANNEXURE F (Component Specifications)

Motor Driver IC (L293D)

Product Information	
Stock Code	35K1335 - KZN
Part Number	**L293DNE**
Description	IC DIP QUAD HALF H-BRIDGE
Manufacturer	TEXAS
Sold In	Each
Minimum Order Qty	1.00
Alternate Part Number	
Memo	PCB MOUNT QUAD HALF H-BRIDGE / DUAL H-BRIDGE DRIVER FOR STEPPER / DC BRUSH MOTORS, 36V, 0.6A PER CHANNEL, DIP PACKAGE

Photoelectric IR Count sensor (LM393)

Product Information	
Stock Code	15C5575 - CPT
Part Number	**KS0009**
Description	PHOTOELECTRIC IR GAP SENSOR MODULE
Manufacturer	KEYESTUDIO
Sold In	Each
Minimum Order Qty	1.00
Alternate Part Number	
Memo	INFRARED GAP / POSITIONING / COUNTING SENSOR MODULE, USING LM393 IC, 32MMx11MMx20MM.



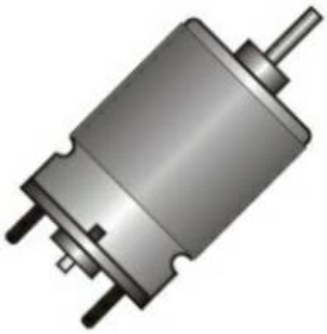
Annexure G (Component Specifications)

Arduino Mega2560

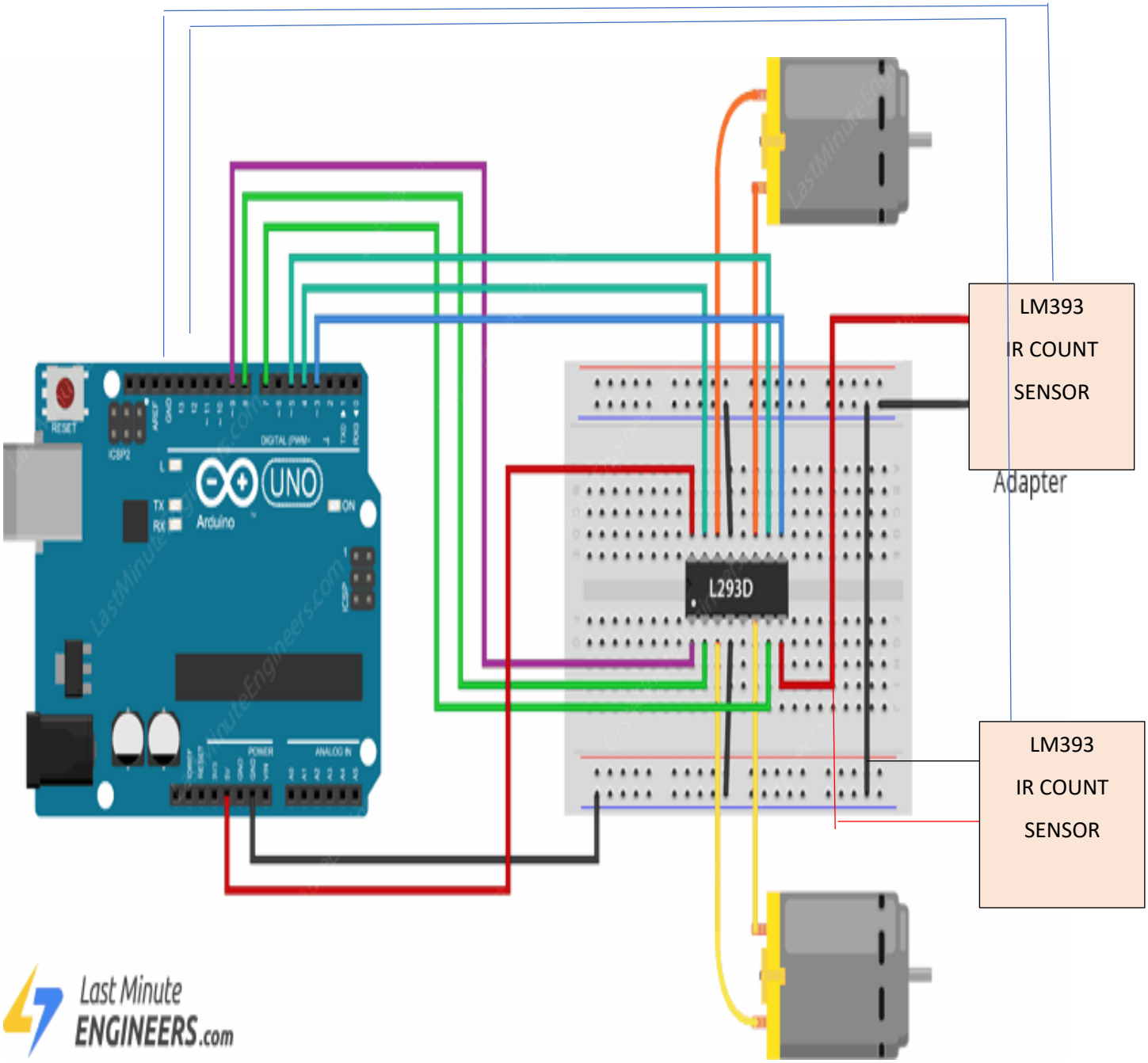
Product Information	
Stock Code	15M8291
Part Number	**MB0072**
Description	ARDUINO MEGA2560 R3 IMPROVED W/CABLE
Manufacturer	KEYESTUDIO
Sold In	Each
Minimum Order Qty	1.00
Alternate Part Number	
Memo	Arduino MEGA2560 R3 improved development board with USB cable.

12V DC Brush Motor

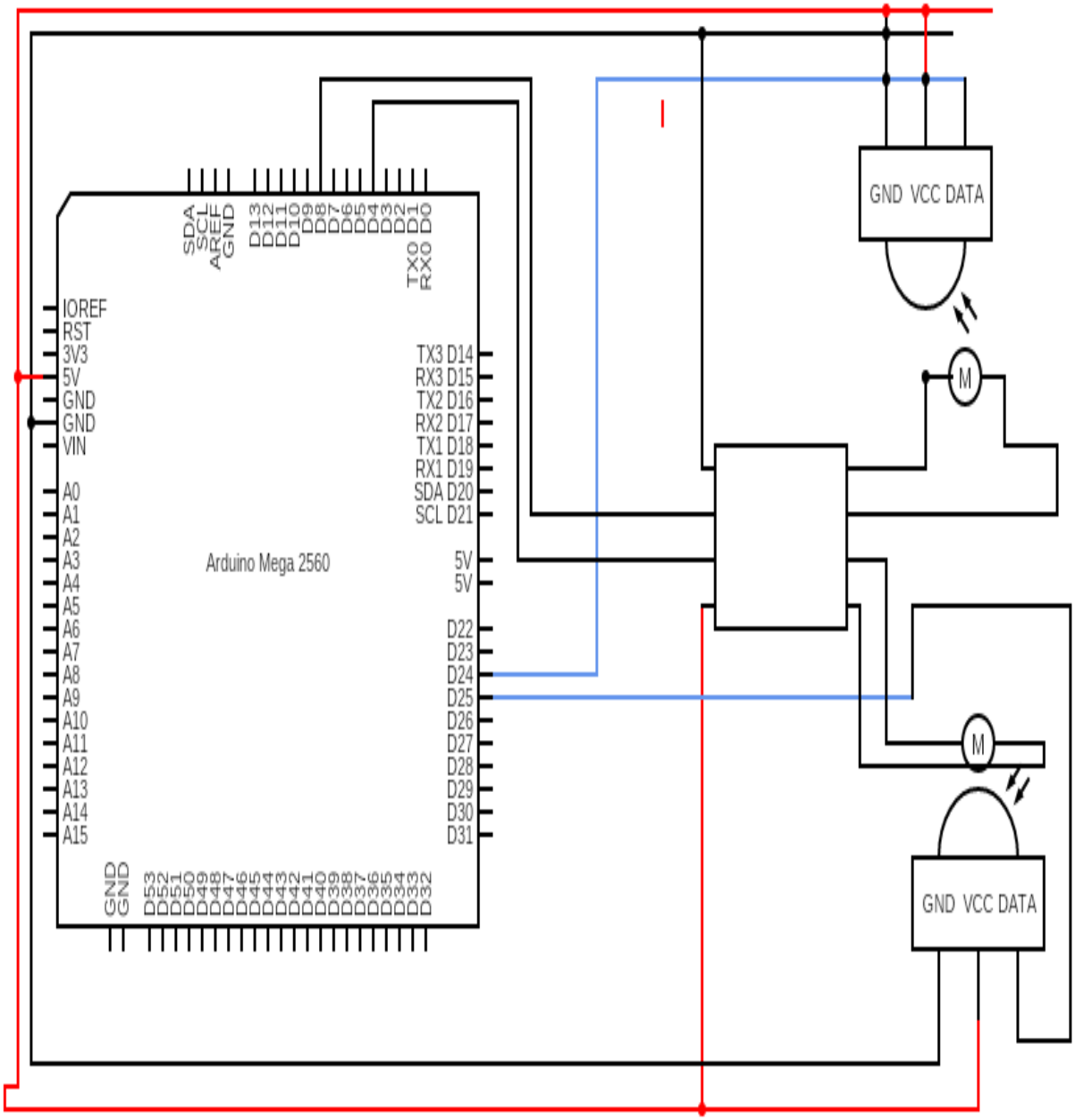
Product Information	
Stock Code	72M1755
Part Number	**SRC-545SAP-2490-70**
Description	DC BRUSH MOTOR 36x50 12VDC 0A8 3K RPM
Manufacturer	SAN XING DA
Sold In	Each
Minimum Order Qty	1.00
Alternate Part Number	
Memo	DC BRUSH MOTOR (DCBM), DC CARBON BRUSHED, 12V FLC=0.83A, FLS=2880 RPM Dm=36 Lm=50 Ls=70 FS=3.175mm, WEIGHT-165g



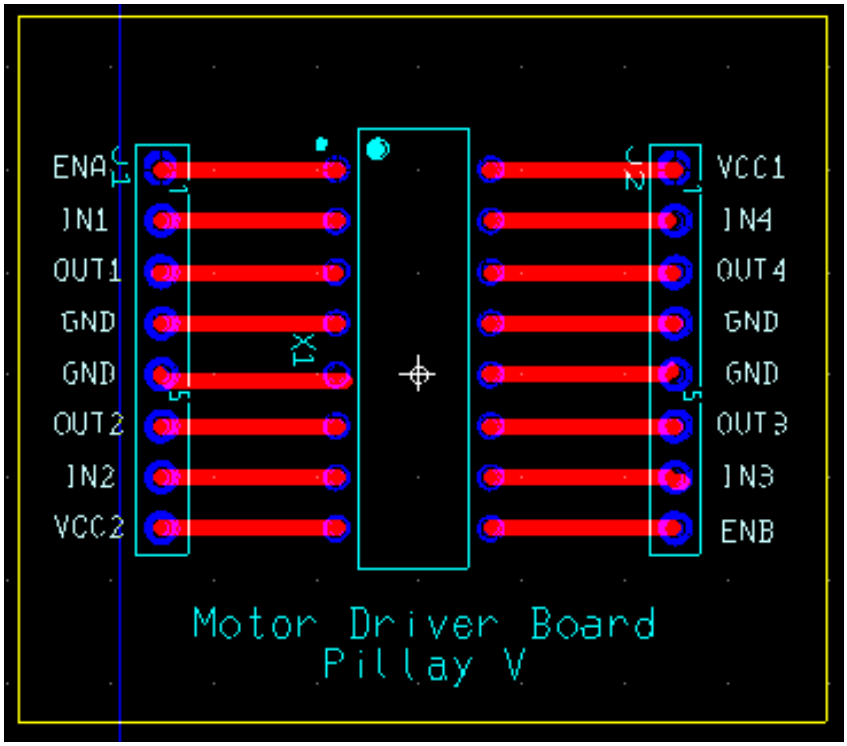
ANNEXURE H (Wiring diagram)

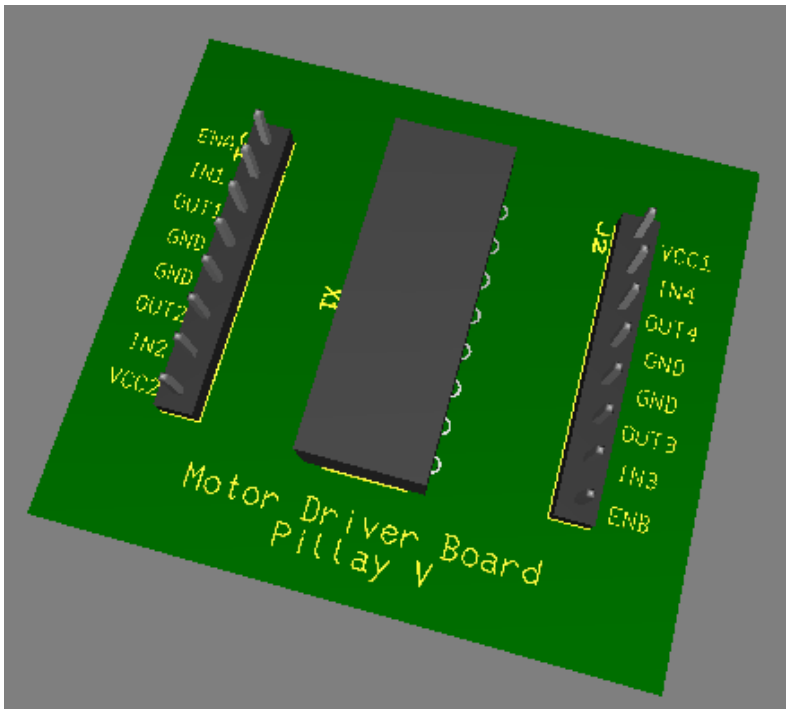


ANNEXURE I (Circuit diagram)

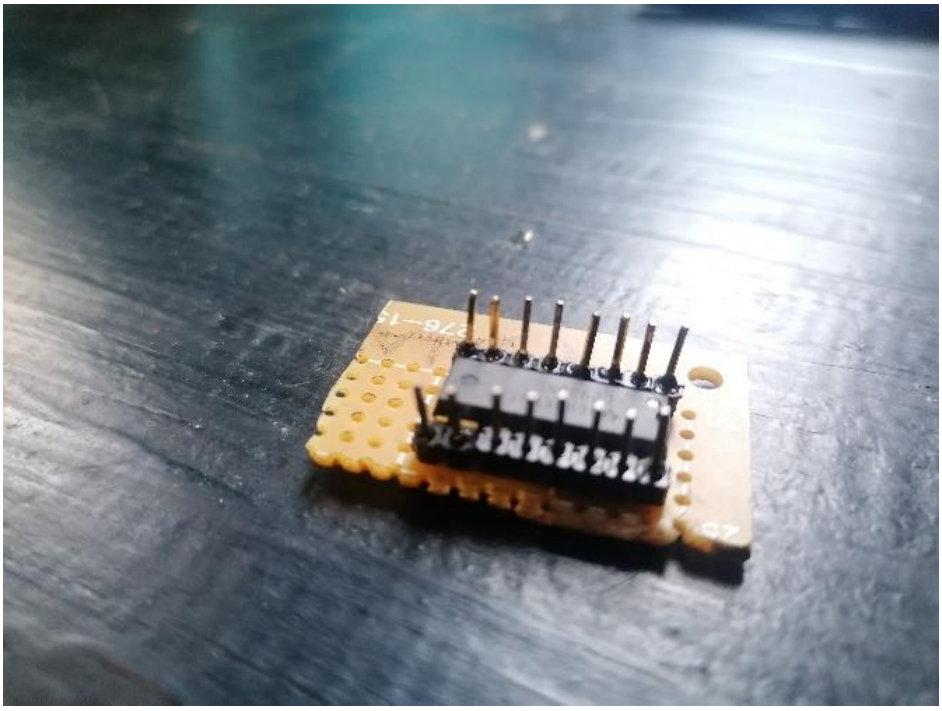


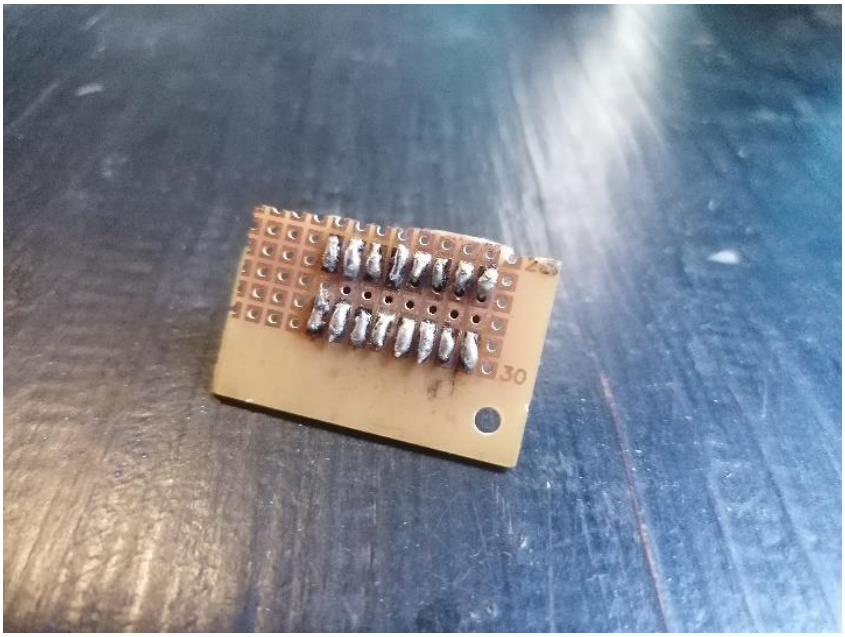
ANNEXURE J (PCB Design)





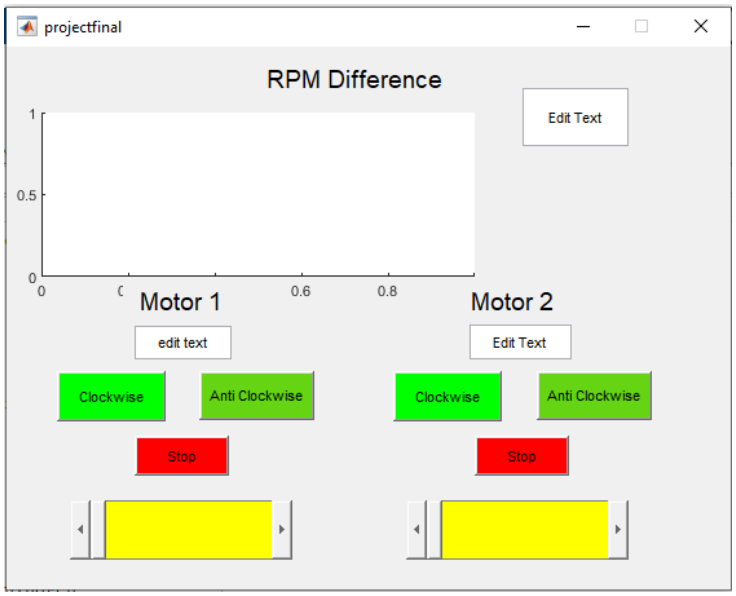
ANNEXURE K (PCB construction)



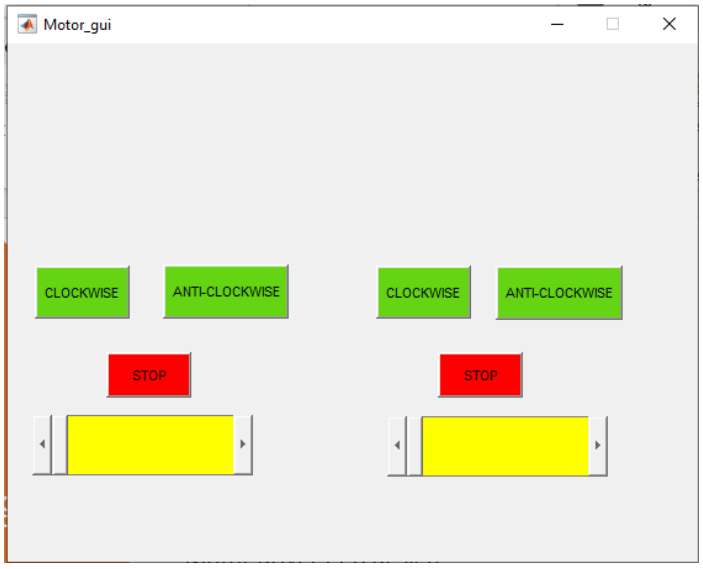


ANNEXURE L (GUI design for both system topologies)

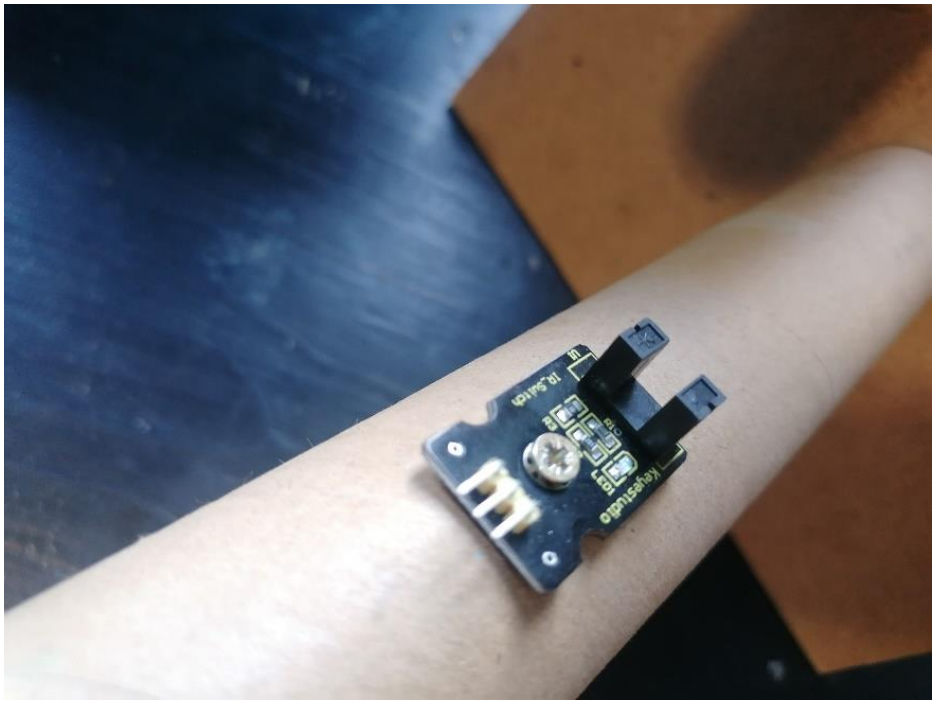
Bidirectional (Motor control and RPM sensor feedback)



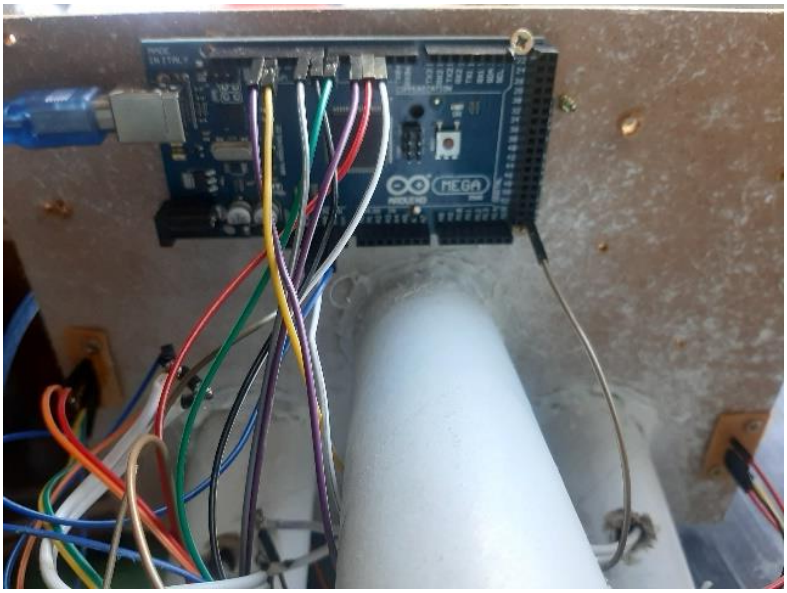
Unidirectional (Motor control only)

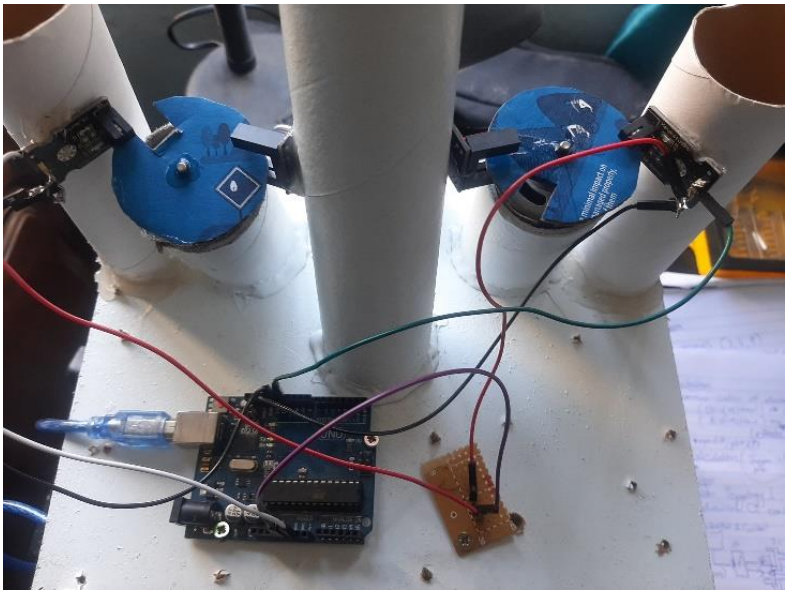


ANNEXURE M (Motor and sensor mounting)

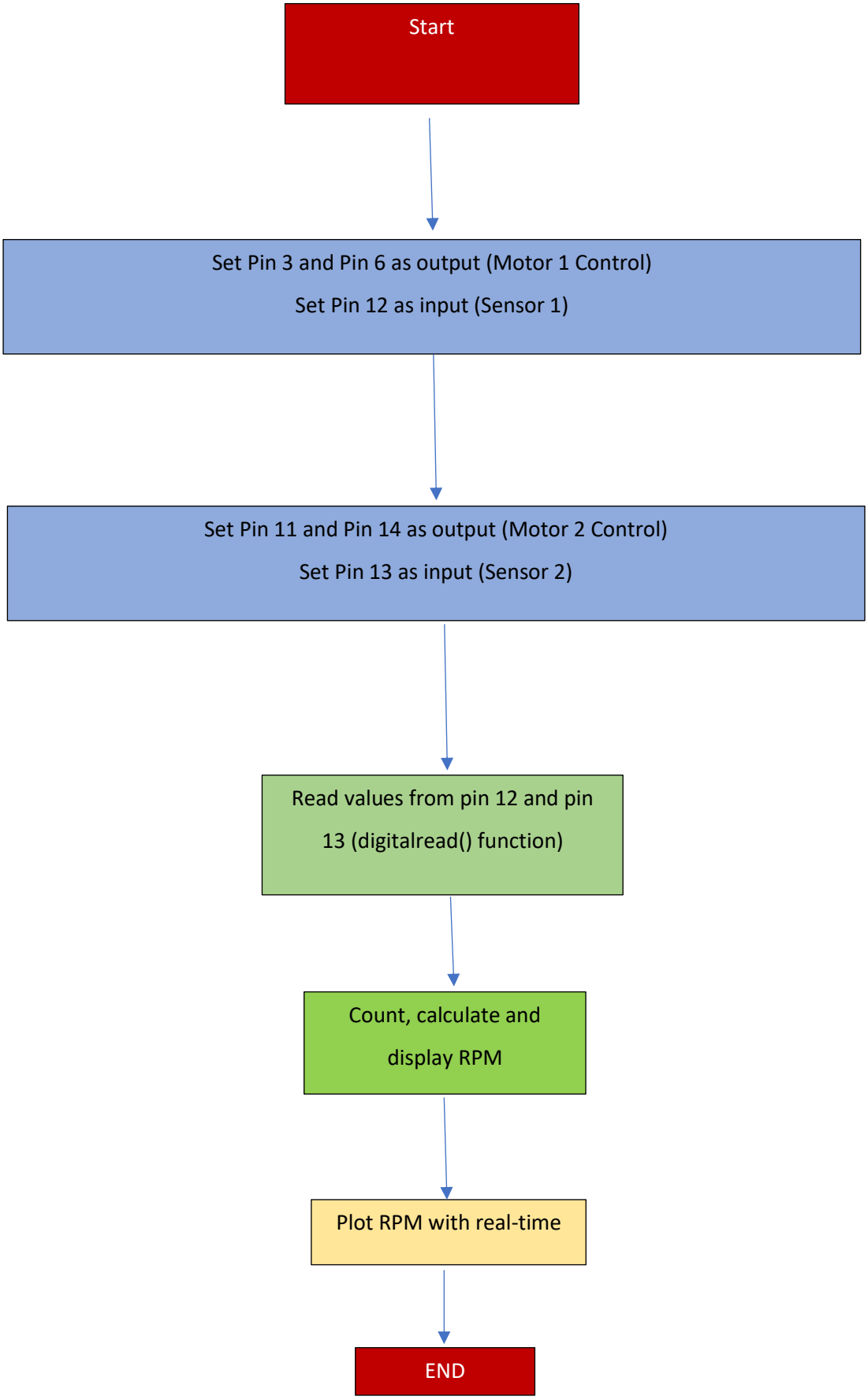


ANNEXURE N (Final system construction)



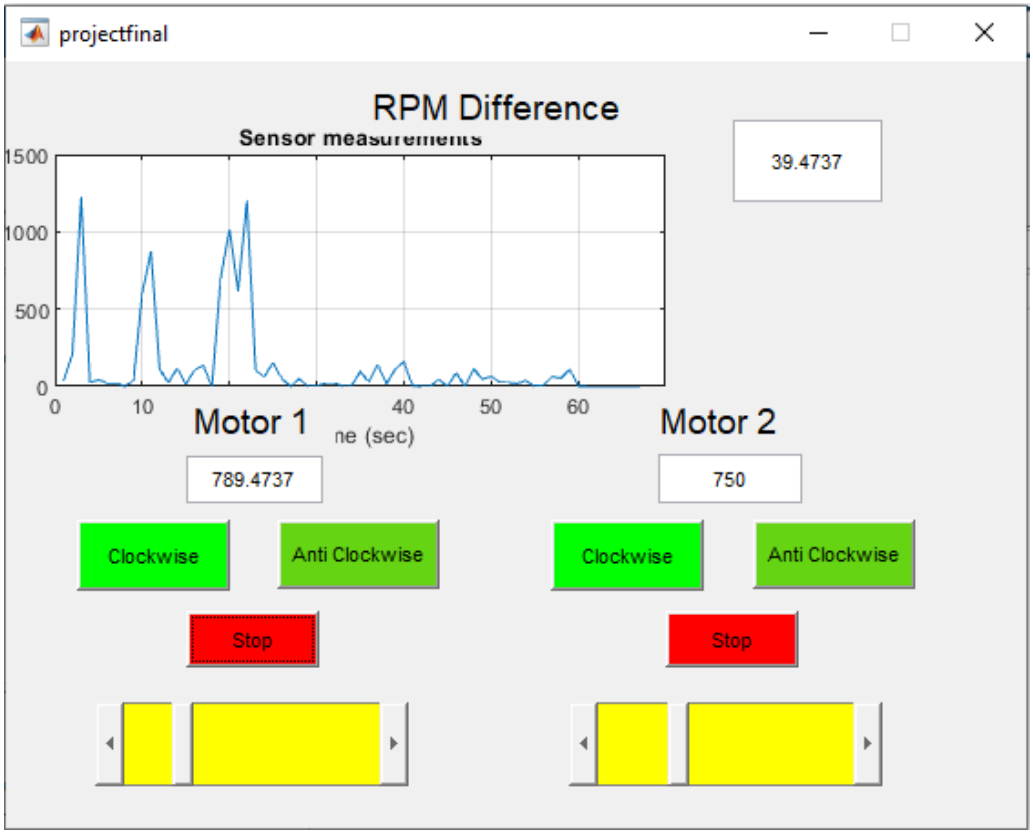


ANNEXURE O (Code construction)

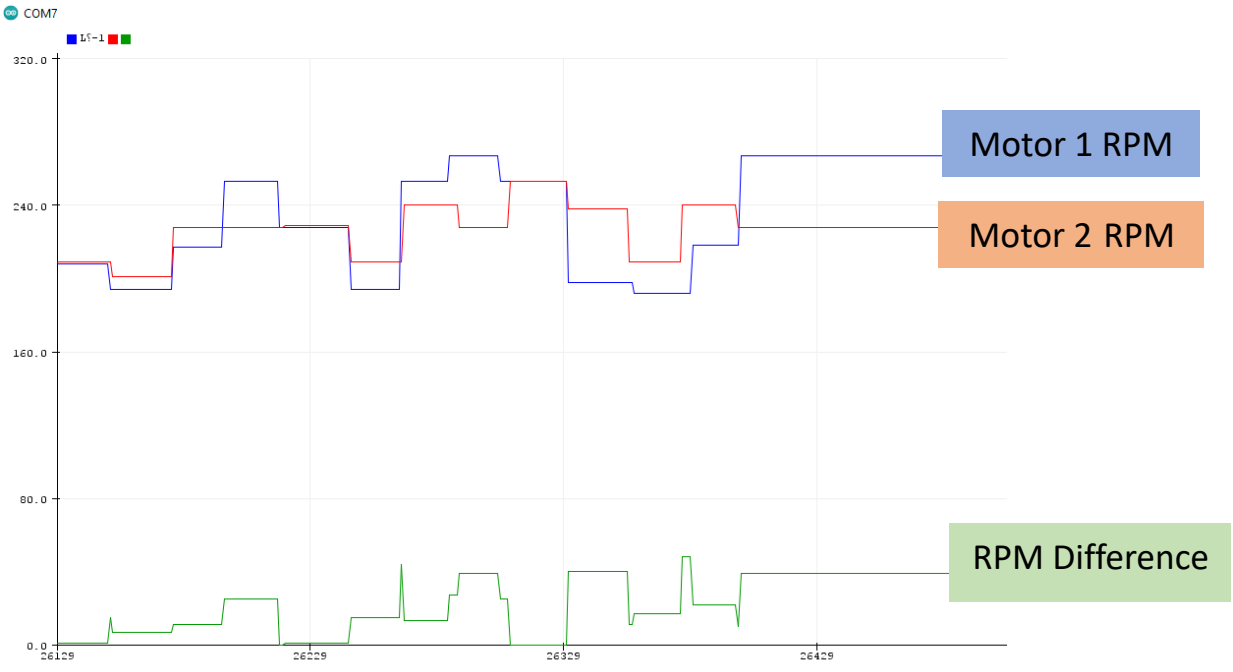


ANNEXURE P (Results obtained from both system topologies)

Bidirectional (RPM diff between the motors in real-time)



Unidirectional (Reader controller displaying the real time acquired data)



Annexure Q (Bidirectional Code) (MATLAB only) (one microcontroller)

```
function varargout = projectfinal(varargin)
% PROJECTFINAL MATLAB code for projectfinal.fig
%      PROJECTFINAL, by itself, creates a new PROJECTFINAL or raises the
existing
%      singleton*.
%
%      H = PROJECTFINAL returns the handle to a new PROJECTFINAL or the
handle to
%      the existing singleton*.
%
%      PROJECTFINAL('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in PROJECTFINAL.M with the given input
arguments.
%
%      PROJECTFINAL('Property','Value',...) creates a new PROJECTFINAL or
raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before projectfinal_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to projectfinal_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help projectfinal

% Last Modified by GUIDE v2.5 30-Nov-2022 23:29:15

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @projectfinal_OpeningFcn, ...
                   'gui_OutputFcn',  @projectfinal_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before projectfinal is made visible.
function projectfinal_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% varargin   command line arguments to projectfinal (see VARARGIN)

% Choose default command line output for projectfinal
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes projectfinal wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = projectfinal_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
clear all;
global a;
a = arduino;

% --- Executes on button press in clockwise1.
function clockwise1_Callback(hObject, eventdata, handles)
% hObject    handle to clockwise1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global a;
global rpm1;
global rpm2;
global rpmdiff;
i1 = 0;
i2 = 0;
i3 = 0;
previous_time1 = 0;
previous_time2 = 0;
keeplooping = true;
writeDigitalPin(a, 'D5', 1);
writeDigitalPin(a, 'D4', 0);
while keeplooping
    rpmdiff = [rpmdiff i3];
    i3 = rpmdiff;
 plot(rpmdiff);grid;
 title ('Sensor measurements')
 xlabel ('Time (sec)')
 ylabel ('RPM Difference')
 val1 = readDigitalPin (a,'D12');
 rpm1 = [rpm1 i1];
 i1 = rpm1;
 if(val1==1)
 t1 = clock;
 current_time1 = t1 (6);
 pulse_time1 = current_time1 - previous_time1;
 rpm1 = (10/pulse_time1)*60;
 disp('rpm1 =');
 disp(rpm1);
 previous_time1 = current_time1;
 else
     rpm1=0;
```

```
 end
 set (handles.rpm2,'string',num2str(rpm1));
  val2 = readDigitalPin (a,'D13');
 rpm2 = [rpm2 i2];
 i2 = rpm2;
 if(val2==1)
 t2 = clock;
 current_time2 = t2 (6);
 pulse_time2 = current_time2 - previous_time2;
 rpm2 = (10/pulse_time2)*60;
 disp('rpm2 =');
 disp(rpm2);
 previous_time2 = current_time2;
 else
 rpm2 = 0;
 end
  set (handles.rpm1,'string',num2str(rpm2));
 if (rpm1<rpm2)
     rpmdiff = rpm2 - rpm1;
 else
     rpmdiff = rpm1 - rpm2;
 end
 set (handles.rpmdiff,'string',num2str(rpmdiff));
 end
% --- Executes on button press in anticlock1.
function anticlock1_Callback(hObject, eventdata, handles)
% hObject    handle to anticlock1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global a;
global rpm1;
global rpm2;
global rpmdiff;
i1 = 0;
i2 = 0;
i3 = 0;
previous_time1 = 0;
previous_time2 = 0;
keeplooping = true;
writeDigitalPin(a, 'D5', 0);
writeDigitalPin(a, 'D4', 1);
while keeplooping
    rpmdiff = [rpmdiff i3];
    i3 = rpmdiff;
 plot(rpmdiff);grid;
 title ('Sensor measurements')
 xlabel ('Time (sec)')
 ylabel ('RPM Difference')
 val1 = readDigitalPin (a,'D12');
 rpm1 = [rpm1 i1];
 i1 = rpm1;
 if(val1==1)
 t1 = clock;
 current_time1 = t1 (6);
 pulse_time1 = current_time1 - previous_time1;
 rpm1 = (10/pulse_time1)*60;
 disp('rpm1 =');
 disp(rpm1);
 previous_time1 = current_time1;
 else
     rpm1=0;
 end
 set (handles.rpm2,'string',num2str(rpm1));
```

```
 val2 = readDigitalPin (a,'D13');
rpm2 = [rpm2 i2];
i2 = rpm2;
if(val2==1)
t2 = clock;
current_time2 = t2 (6);
pulse_time2 = current_time2 - previous_time2;
rpm2 = (10/pulse_time2)*60;
disp('rpm2 =');
disp(rpm2);
previous_time2 = current_time2;
else
rpm2 = 0;
end
set (handles.rpm1,'string',num2str(rpm2));
if (rpm1<rpm2)
    rpmdiff = rpm2 - rpm1;
else
    rpmdiff = rpm1 - rpm2;
end
set (handles.rpmdiff,'string',num2str(rpmdiff));
end

% --- Executes on button press in stop1.
function stop1_Callback(hObject, eventdata, handles)
% hObject    handle to stop1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global a;
writeDigitalPin(a, 'D5', 0);
writeDigitalPin(a, 'D4', 0);

% --- Executes on button press in clock2.
function clock2_Callback(hObject, eventdata, handles)
% hObject    handle to clock2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global a;
global rpm1;
global rpm2;
global rpmdiff;
i1 = 0;
i2 = 0;
i3 = 0;
previous_time1 = 0;
previous_time2 = 0;
keeplooping = true;
writeDigitalPin(a, 'D8', 0);
writeDigitalPin(a, 'D7', 1);
while keeplooping
    rpmdiff = [rpmdiff i3];
    i3 = rpmdiff;
 plot(rpmdiff);grid;
 title ('Sensor measurements')
 xlabel ('Time (sec)')
 ylabel ('RPM Difference')
 val1 = readDigitalPin (a,'D13');
 rpm1 = [rpm1 i1];
 i1 = rpm1;
 if(val1==1)
 t1 = clock;
 current_time1 = t1 (6);
 pulse_time1 = current_time1 - previous_time1;
```

```
 rpm1 = (10/pulse_time1)*60;
 disp('rpm1 =');
 disp(rpm1);
 previous_time1 = current_time1;
 else
     rpm1=0;
 end
 set (handles.rpm2,'string',num2str(rpm1));
  val2 = readDigitalPin (a,'D13');
 rpm2 = [rpm2 i2];
 i2 = rpm2;
 if(val2==1)
 t2 = clock;
 current_time2 = t2 (6);
 pulse_time2 = current_time2 - previous_time2;
 rpm2 = (10/pulse_time2)*60;
 disp('rpm2 =');
 disp(rpm2);
 previous_time2 = current_time2;
 else
 rpm2 = 0;
 end
  set (handles.rpm1,'string',num2str(rpm2));
 if (rpm1<rpm2)
     rpmdiff = rpm2 - rpm1;
 else
     rpmdiff = rpm1 - rpm2;
 end
 set (handles.rpmdiff,'string',num2str(rpmdiff));
 end
% --- Executes on button press in anticlock2.
function anticlock2_Callback(hObject, eventdata, handles)
% hObject    handle to anticlock2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global a;
global rpm1;
global rpm2;
global rpmdiff;
i1 = 0;
i2 = 0;
i3 = 0;
previous_time1 = 0;
previous_time2 = 0;
keeplooping = true;
writeDigitalPin(a, 'D8', 1);
writeDigitalPin(a, 'D7', 0);
while keeplooping
    rpmdiff = [rpmdiff i3];
    i3 = rpmdiff;
 plot(rpmdiff);grid;
 title ('Sensor measurements')
 xlabel ('Time (sec)')
 ylabel ('RPM Difference')
 val1 = readDigitalPin (a,'D12');
 rpm1 = [rpm1 i1];
 i1 = rpm1;
 if(val1==1)
 t1 = clock;
 current_time1 = t1 (6);
 pulse_time1 = current_time1 - previous_time1;
 rpm1 = (10/pulse_time1)*60;
 disp('rpm1 =');
```

```
 disp(rpm1);
 previous_time1 = current_time1;
 else
     rpm1=0;

 end
 set (handles.rpm2,'string',num2str(rpm1));
  val2 = readDigitalPin (a,'D13');
 rpm2 = [rpm2 i2];
 i2 = rpm2;
 if(val2==1)
 t2 = clock;
 current_time2 = t2 (6);
 pulse_time2 = current_time2 - previous_time2;
 rpm2 = (10/pulse_time2)*60;
 disp('rpm2 =');
 disp(rpm2);
 previous_time2 = current_time2;
 else
 rpm2 = 0;
 end
  set (handles.rpm1,'string',num2str(rpm2));
 if (rpm1<rpm2)
     rpmdiff = rpm2 - rpm1;
 else
     rpmdiff = rpm1 - rpm2;
 end
 set (handles.rpmdiff,'string',num2str(rpmdiff));
 end
% --- Executes on button press in stop2.
function stop2_Callback(hObject, eventdata, handles)
% hObject    handle to stop2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global a;
writeDigitalPin(a, 'D8', 0);
writeDigitalPin(a, 'D7', 0);

% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
slider
global a;
global rpm1;
global rpm2;
global rpmdiff;
i1 = 0;
i2 = 0;
i3 = 0;
previous_time1 = 0;
previous_time2 = 0;
keeplooping = true;
sliderm1 = get(hObject,'Value');
slider1 = sliderm1 * 20;
set(handles.slider1, 'String', num2str(slider1));
writePWMVoltage(a, 'D9', sliderm1);
guidata(hObject,handles);
while keeplooping
```

```
    rpmdiff = [rpmdiff i3];
    i3 = rpmdiff;
 plot(rpmdiff);grid;
 title ('Sensor measurements')
 xlabel ('Time (sec)')
 ylabel ('RPM Difference')
 val1 = readDigitalPin (a,'D12');
 rpm1 = [rpm1 i1];
 i1 = rpm1;
 if(val1==1)
 t1 = clock;
 current_time1 = t1 (6);
 pulse_time1 = current_time1 - previous_time1;
 rpm1 = (10/pulse_time1)*60;
 disp('rpm1 =');
 disp(rpm1);
 previous_time1 = current_time1;
 else
     rpm1=0;
 end
 set (handles.rpm2,'string',num2str(rpm1));
  val2 = readDigitalPin (a,'D13');
 rpm2 = [rpm2 i2];
 i2 = rpm2;
 if(val2==1)
 t2 = clock;
 current_time2 = t2 (6);
 pulse_time2 = current_time2 - previous_time2;
 rpm2 = (10/pulse_time2)*60;
 disp('rpm2 =');
 disp(rpm2);
 previous_time2 = current_time2;
 else
 rpm2 = 0;
 end
  set (handles.rpm1,'string',num2str(rpm2));
 if (rpm1<rpm2)
     rpmdiff = rpm2 - rpm1;
 else
     rpmdiff = rpm1 - rpm2;
 end
 set (handles.rpmdiff,'string',num2str(rpmdiff));
 end
% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



% --- Executes on slider movement.
function slider2_Callback(hObject, eventdata, handles)
% hObject    handle to slider2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
```

```
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
slider
global a;
global rpm1;
global rpm2;
global rpmdiff;
i1 = 0;
i2 = 0;
i3 = 0;
previous_time1 = 0;
previous_time2 = 0;
keeplooping = true;
sliderm2 = get(hObject,'Value');
slider2 = sliderm2 * 20;
set(handles.slider2, 'String', num2str(slider2));
writePWMVoltage(a, 'D3', sliderm2);
guidata(hObject,handles);
while keeplooping
    rpmdiff = [rpmdiff i3];
    i3 = rpmdiff;
 plot(rpmdiff);grid;
 title ('Sensor measurements')
 xlabel ('Time (sec)')
 ylabel ('RPM Difference')
 val1 = readDigitalPin (a,'D12');
 rpm1 = [rpm1 i1];
 i1 = rpm1;
 if(val1==1)
 t1 = clock;
 current_time1 = t1 (6);
 pulse_time1 = current_time1 - previous_time1;
 rpm1 = (10/pulse_time1)*60;
 disp('rpm1 =');
 disp(rpm1);
 previous_time1 = current_time1;
 else
     rpm1=0;
 end
 set (handles.rpm2,'string',num2str(rpm1));
  val2 = readDigitalPin (a,'D13');
 rpm2 = [rpm2 i2];
 i2 = rpm2;
 if(val2==1)
 t2 = clock;
 current_time2 = t2 (6);
 pulse_time2 = current_time2 - previous_time2;
 rpm2 = (10/pulse_time2)*60;
 disp('rpm2 =');
 disp(rpm2);
 previous_time2 = current_time2;
 else
 rpm2 = 0;
 end
 set (handles.rpm1,'string',num2str(rpm2));
 if (rpm1<rpm2)
     rpmdiff = rpm2 - rpm1;
 else
     rpmdiff = rpm1 - rpm2;
 end
 set (handles.rpmdiff,'string',num2str(rpmdiff));
 end
% --- Executes during object creation, after setting all properties.
function slider2_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to slider2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function rpm1_Callback(hObject, eventdata, handles)
% hObject    handle to rpm1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of rpm1 as text
%        str2double(get(hObject,'String')) returns contents of rpm1 as a
double


% --- Executes during object creation, after setting all properties.
function rpm1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to rpm1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function rpm2_Callback(hObject, eventdata, handles)
% hObject    handle to rpm2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of rpm2 as text
%        str2double(get(hObject,'String')) returns contents of rpm2 as a
double


% --- Executes during object creation, after setting all properties.
function rpm2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to rpm2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function rpmdiff_Callback(hObject, eventdata, handles)
% hObject    handle to rpmdiff (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of rpmdiff as text
%        str2double(get(hObject,'String')) returns contents of rpmdiff as a
double


% --- Executes during object creation, after setting all properties.
function rpmdiff_CreateFcn(hObject, eventdata, handles)
% hObject    handle to rpmdiff (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Annexure D (Unidirectional Code) (MATLAB and Arduino) (Two Arduinos)

```
function varargout = Motor_gui(varargin)
% MOTOR_GUI MATLAB code for Motor_gui.fig
%      MOTOR_GUI, by itself, creates a new MOTOR_GUI or raises the existing
%      singleton*.
%
%      H = MOTOR_GUI returns the handle to a new MOTOR_GUI or the handle to
%      the existing singleton*.
%
%      MOTOR_GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in MOTOR_GUI.M with the given input
arguments.
%
%      MOTOR_GUI('Property','Value',...) creates a new MOTOR_GUI or raises
the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before Motor_gui_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Motor_gui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Motor_gui

% Last Modified by GUIDE v2.5 18-Sep-2022 14:55:13

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Motor_gui_OpeningFcn, ...
                   'gui_OutputFcn',  @Motor_gui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Motor_gui is made visible.

function Motor_gui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Motor_gui (see VARARGIN)
```

```
% Choose default command line output for Motor_gui
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Motor_gui wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Motor_gui_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
clear all;
global a;
a = arduino;

% --- Executes on button press in clockwise1.
function clockwise1_Callback(hObject, eventdata, handles)
global a;
writeDigitalPin(a, 'D5', 1);
writeDigitalPin(a, 'D4', 0);

% hObject    handle to clockwise1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in anticlock1.
function anticlock1_Callback(hObject, eventdata, handles)
global a;
writeDigitalPin(a, 'D5', 0);
writeDigitalPin(a, 'D4', 1);

% hObject    handle to anticlock1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in clockwise2.
function clockwise2_Callback(hObject, eventdata, handles)
global a;
writeDigitalPin(a, 'D8', 0);
writeDigitalPin(a, 'D7', 1);

% hObject    handle to clockwise2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in anticlock2.
function anticlock2_Callback(hObject, eventdata, handles)
global a;
```

```
writeDigitalPin(a, 'D8', 1);
writeDigitalPin(a, 'D7', 0);

% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in stop1.
function stop1_Callback(hObject, eventdata, handles)
global a;
writeDigitalPin(a, 'D5', 0);
writeDigitalPin(a, 'D4', 0);
% hObject    handle to stop1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in stop2.
function stop2_Callback(hObject, eventdata, handles)
global a;
writeDigitalPin(a, 'D8', 0);
writeDigitalPin(a, 'D7', 0);
% hObject    handle to stop2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
global a;
sliderm1 = get(hObject,'Value');
slider1 = sliderm1 * 20;
set(handles.slider1, 'String', num2str(slider1));
writePWMVoltage(a, 'D9', sliderm1);
guidata(hObject,handles);

% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
slider


% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function slider2_Callback(hObject, eventdata, handles)
global a;
sliderm2 = get(hObject,'Value');
```

```
slider2 = sliderm2 * 20;
set(handles.slider2, 'String', num2str(slider2));
writePWMVoltage(a, 'D3', sliderm2);
guidata(hObject,handles);

% hObject    handle to slider2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
slider


% --- Executes during object creation, after setting all properties.
function slider2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

ARDUINO CODE Microcontroller 2 (Reader)

```
#include <LiquidCrystal_I2C.h>


LiquidCrystal_I2C lcd(0x27,16,2);

int IRsensor1 = 13;

int IRsensor2 = 12;

unsigned long duration = 0;

unsigned long duration2 = 0;

int rpm1 = 0;

int rpm2 = 0;

int rpm_diff = 0;

float rpm_a = 0;

float rpm_b = 0;

int counter = 0;

int counter2 = 0;

int present = 0;

int previous = 0;

unsigned long elapsed = 0;

unsigned long elapsed_prev = 0;
```

```
int present2 = 0;
int previous2 = 0;
unsigned long elapsed2 = 0;
unsigned long elapsed_prev2 = 0;
void setup()
 {
  Serial.begin(9600); // Start serial communication

  pinMode(IRsensor1,INPUT);
  pinMode(IRsensor2,INPUT);
    lcd.init();
    lcd.backlight();
    lcd.setCursor(0, 0);
    lcd.print("M1:");
    lcd.setCursor(9, 0);
    lcd.print("M2:");
    lcd.setCursor(0, 1);
    lcd.print("RPM diff:");
 }
 void loop()
 {

//////////////////////one rotation and pulse time//////////////////
     if (digitalRead(IRsensor1) == 1 && previous == 0)
      {
       previous = 1;
       duration = elapsed - elapsed_prev;
       elapsed_prev  = micros();
       }
     if (digitalRead(IRsensor1) == 1 && previous == 1)
      {
       previous = 1;
      }
```

```
  if (digitalRead(IRsensor1) == 0 && previous == 1)
   {
    previous = 0;
   }
  if (digitalRead(IRsensor1) == 0 && previous == 0)
   {
    previous = 0;
    elapsed = micros();
   }
 if (digitalRead(IRsensor2) == 1 && previous2 == 0)
   {
    previous2 = 1;
    duration2 = elapsed2 - elapsed_prev2;
    elapsed_prev2  = micros();
   }
  if (digitalRead(IRsensor2) == 1 && previous2 == 1)
   {
    previous2 = 1;
   }
  if (digitalRead(IRsensor2) == 0 && previous2 == 1)
   {
    previous2 = 0;
   }
  if (digitalRead(IRsensor2) == 0 && previous2 == 0)
   {
    previous2 = 0;
    elapsed2 = micros();
   }
////////////////////////////////////////////////////////////////


  rpm1 = 60000000/duration;
```

```
  rpm2 = 60000000/duration2;


   if ( (rpm_a-2) < rpm1  &&  rpm1 < (rpm_a+2))
{
 rpm_a = rpm1;
 counter = counter +1;
}
  if ( (rpm_b-2) < rpm2  &&  rpm2 < (rpm_b+2))
{
 rpm_b = rpm2;
 counter2 = counter2 +1;
}
if (rpm1 > rpm2)
{
 rpm_diff = rpm1 - rpm2;
}
 else
{
 rpm_diff = rpm2 - rpm1;
}

   lcd.setCursor(4, 0);
   lcd.print(rpm1);
   lcd.setCursor(13, 0);
   lcd.print(rpm2);
   lcd.setCursor(10, 1);
   lcd.print(rpm_diff);

Serial.print(rpm1);Serial.print(" ");
Serial.print(rpm2);Serial.print(" ");
Serial.println(rpm_diff);
 }
```

Graduate Attribute 3: Engineering Design (GA3)

1. Purpose

The engineering ability is applied to design the Revolution Counter system to allow for its functionality to be a success as well as for the system to operate in the most efficient way possible without compromising accuracy. The applicable standards of design are attained from the discipline of Electronic Engineering and by using the various skills, knowledge and attributes associated with it. The relevant codes of practice and legislation are implemented within the procedure of the project in terms of the design and overall engineering of a digital revolution counter system which can be of more accuracy than a traditional analogue system as it does not empower the fourth industrial revolution in terms of embracing the power of these single board computers.

2. Main Idea

The requirements of the project included designing and construction a revolution counter system that can count the revolutions made by two 12V DC motors to determine their speed as well the difference in speed between them. This system had been designed using electronic engineering knowledge to implement various sensors and controllers to allow for this system to function. The use of MATLAB allows for a GUI to be created which will allow for the study of the speed, difference in speed and the number of revolutions made by each motor. The GUI also allows the user to interact with the system in terms of motor control and to observe the results from different levels of speed. The safety procedures to take into consideration with regards to the OSH Act includes the use of safety workwear as well as the safety procedures taken provided when working with electricity and various tools.

3. Organization and Charter

The ability to manage and organize the design process is a difficult task to allow for the systems operation to be a success. The relevant constraints would require a considerable amount of focus so that it can be resolved. The problem solving is a vital skill needed by the engineer to troubleshoot the systems multifunctionality which will allow the working operation to be a success. The various constraints of the project include the capability of the LM393 to sense at a high rate of speed as well as for the microcontroller to read and compute the data just as fast. The constraint of time must be well managed for the Revolution Counter project to be a success as well as for the relevant deadlines to be met.

4. Design Tools

The reason for using MATLAB includes the ability that it can create a GUI as well as interface it to a microcontroller to perform various operations and read data which can be displayed on a text window for observing the behaviour of a system. The MATLAB software contains its user-friendly development environment as well its help centre which allows the user to seek guidance if there seems to be a confusion or constraint that the user may have or experience within the software. The proposed reason for using MATLAB includes that it is a very powerful software tool which can compute data at the speed we need it to especially in a simulation which would need to compute live variables quickly and display their values as well as their changes instantaneously.

5. Techno-Economic Analysis and Judgement

The alternatives in industry include traditional analogue systems are outdated which were used in the old cars and any form of old technology. These systems were also not fully accurate and lacked the capabilities digital systems have. However, the old analogue systems were designed and constructed strong to last a long time to promote its overall reliability. The use of digital systems empowers the fourth industrial revolution in terms of automation and improving the processes that keep humans alive. Digital systems are also easier and cheaper to maintain as well as troubleshoot which can remove the constraint of dead time within essential industrial processes.

6. Mechanics: Pragmatic Approach

There is no social impact with regards to the revolution counter system but there are benefits which include inspiring more people to innovate ways of improving the efficiency of the system or even developing a whole new system. The economics behind the system is relevantly even because of the advantages the system has in terms of its cost and maintenance. The revolution counter system does not endanger any legal health matter because the system emits no pollution that may harm its surrounding environment The system also has the advantage of having a low power consumption. In terms of the safety associated with the system is low as it works at a harmless voltage however it is important and a safety procedure to always work with electricity when it is not flowing/live,

7. Genre and Conventions

The ability to apply design logic is essential within the field of electronic engineering because the field is embedded with logic for it to exist. The relevant logical decision-making skills is needed to design the programming flowchart as well as the programming of the Arduino microcontroller which will carry out the tasks outlined within the programming flowchart. The Arduino microcontroller would have to make logical decisions at a fast rate. The relevant information is communicated within the report from the systems design to its final construction. The expected problems and solutions are derived from the expected shortfalls within the design process. The use of simple logic to create a basic operational system which can be interactive via the GUI in the MATLAB software is the objective with the aim of keeping things simple to reduce complex problems as well an unstable system.

Graduate Attribute 9: Independent Learning (GA3)

1. Independent learning

The independent learning attribute had made up most of the project in terms of its design and working principle. The study of DC motor control using various driver circuits to control speed and direction which have many applications in industry. The LM393 IR sensor is implemented as a count sensor and can also be applied as a position marker for use in timing machines such as engine and valve timing, X , Y and Z motors in 3D printing machines, CNC machines and printers. The system allows for the measurement and computation of revolutions on any rotating medium. The complex programming with embedded mathematics to return the revolutions of the medium in real time as well as the difference in revolutions between the mediums. This programming complexity had required a large amount of processing power. The use of unidirectional and bidirectional systems is useful in systems where the outcome would be more advantageous. The improvements can be enhanced within the system by adding a PID controller to control a setpoint by minimizing variations of speed to display as well as a low-pass filter to connect and smoothen the peaks within the pulse train signal.

2. Competence

The competence within the project was to produce a working revolution counter system which did not have to measure speed but to use that real time speed value to perform calculations which was the difference in speed between the two rotating mediums. The competence in terms of engineering skills had to be applied to allow for its working operation to be a success. The competence to further improve the system by implementing different methods of communication to improve real time data transmission as well as to demonstrate and differentiate between the two types of communication methods and which worked out well for this high-speed application. The competent skills in further improving the systems level of human interaction by addition of a broadly defined GUI and display. The engineering ethic relating to competence includes the constant effort within problem areas of the project that enabled its operation.

3. Well-developed learning skills

The well-developed learning skills includes the creation of a GUI within the MATLAB environment for simulation and demonstration purposes, the use of power electronics to drive the two 12V DC motors and the applied H-bridge circuit as well as PWM implementation. The use of LM393 and other sensors allows for the microcontroller to have abilities to sense different things that can be structured to suit any modern-day application. The learning skill of complex mathematics to be embedded in the program as well as the use of different languages in different programming environments.