An Innovative PID Controller in Conjunction with DC Electric Motor for Control of Hybrid Electric Vehicle

M Majid Hussain[1], Zulfiqar A Memon[2], M Akmal Chaudhary[2], M Siddique[3]
[1]*Faculty of Computing, Engineering and Science, University of South Wales, Treforest, Cardiff, United Kingdom*
[2]*Department of Electrical and Computer Engineering, Ajman University, Ajman, United Arab Emirates*
[3]*Department of Electrical Engineering, NFC IET, Multan, Pakistan*

Abstract **-** *This paper introduces a hybrid vehicle concept, development and implementation using both an electric motor and a petrol engine to increase efficiency and reduce carbon footprint. Initially, a prototype of a hybrid electric vehicle (HEV) is designed and the output values are measured, before a control system is developed and implemented to control the speed of the DC motor using an innovative microcontroller as the vehicle's electronic control unit (ECU), along with a proportional integral derivative (PID) controller using speed as an input. The prototype made integrated voltage, current, speed and torque sensors for feedback consequential in a closed loop control system, which successfully resulted in matching the speed input of a user-controlled pedal sensor. A user interface was developed to demonstrate the driver of the vehicle about significant variables such as the revolutions per minute (RPM) of the motor, the speed of the vehicle together with the current being drawn, and the voltage applied to the motor with overall power. A digital interface with pulse width modulation (PWM) capabilities was used to transmit a preset DC voltage to the speed controller for the output of a variable voltage from the Arduino. The results show that innovative PID controller algorithm can enhance the execution of electric motor speed at various set-points of Kp, Ki, Kd, to attain a reliable and stable speed. User interface delivers considerably better communication between the Arduino and PID controller for constraints of maximum and stable speed and operational safety.*

Keywords - *PID Controller, DC Motor, User interface, Speed control, Carbon emission.*

I. INTRODUCTION

In recent years, it has been seen that the non-renewable energy sources have been diminishing day by day, due to this the cost of power sources i.e. fuel for running the vehicles has been increasing enormously. The hybrid vehicle operating system is a combination of two or more power sources for internal operation. Typically, a combination of an internal combustion engine (ICE) and an electric motor (EM) is used in the hybrid electric vehicles (HEVs). A HEV configuration is multifaceted with many different components and each component demands specific modelling and design which is a difficult task, because one component's parameters can devastate the power level of the others. The effects of insufficient power may make the vehicle unnecessarily costly or inefficient [1, 2]. In emission-free and environment friendly urban areas, HEVs are a key factor in traffic development and, therefore, reductionof carbon footprint. The main requirement for traction motors used in HEVs is to produce torques of propulsion over a wide range of speeds. Two main electric motor types, the permanent magnet motor (PMM), and the induction motor (IM), are widely used for HEVs.

DC motors are efficient, presenting high reliability and easy maintenance. The DC motors have higher starting torque, quick starting, stopping, reversing, and variable speeds with input voltage. The DC motors are easier and cheaper to control than AC motors and the easiness to control results in smooth acceleration and efficient battery usage [3-5]. Although brushed DC motor suffers from the brush maintenance, and yet the foregoing mentioned advantages make the DC motor to be a perfect candidate for use in electric vehicles.

Electric motor and its control technology is one of the main components of hybrid electric vehicle, and largely affects vehicle’s power performance, fuel economy, and emission. Problems related to the energy crisis and substantial carbon emissions have become extremely severe, and the ideal solution appears to be the electric vehicles (EVs). The alternative choice is the HEV, as batteries have too low energy density and issues with charging and discharging. One core component of the HEV is an electric motor and different types of DC electric motor were used in HEVs, including the brushless DC motor, which has the advantages of light weight and high efficiency [6, 7]. High power density, high starting torque, high efficiency, robustness, good reliability and wide range of speed control are important aspects of electric motors used in HEVs [8-10]. Use of electric motors has become very important in everyday life especially in conveyors, air conditioning, etc. Technically, for its drive, the HEV comprises of four main parts: the control system and motor, the driving system, as the energy storage and the body, of which the motor and the drive system decide the overall characteristics of the EVs. For starters - fast start and stop, variable speed, large load and high power - main performance requirements must be considered when designing EVs. The speed control of DC motors already employs various methods, one of which is proportional integral derivative (PID) [11-12]. PID is simple in design and has advantages in every framework. For application in a HEV, PID controller is mentioned in the literature [13].

Set point tracking and disturbance rejection are the key goals of both classical and modern control system engineering, where feedback as well as feedforward can be used to achieve the set point tracking design goals and disruption rejection. However, using feedback control only to achieve these goals may in some circumstances lead to an interaction between the demands of the design, for example the multi-objective design problem mentioned in [14]. Commonly, energy-efficient control drives for electric motors face perturbations of load torque due to unmodeled dynamics of the attached equipment. Hence the efficient and reliable control of electric motors using a minimum number of sensors is a demanding, important and appropriate research subject and its solutions admit a broad range of practical applications in engineering products and development of systems. On the other hand, recent ground-breaking developments dealing with active disruptive rejection control (ADRC) were introduced. In the ADRC method the controller estimates and compensates for unknown conditions and disturbances in real time.

The main aim of this paper is to design and develop an innovative PID controller, hybrid vehicle prototype, then simulate it, and analyze the effect of incorporated voltage, current, speed and torque of DC electric motor. In addition, a new prototype to control DC motor speed using a PID controller of HEV is developed. The anticipated HEV prototype is assembled with a simple design procedure and low-cost available equipment. The design of the PID controller is subsequently introduced to tune the PID controller’s three parameters, Kp, Ki and Kd, based on proving the functionality of the proposed algorithm from a set of simulations. The foremost function of the PID controller is to keep a constant speed of the vehicle regardless of the torque demand and disturbances in the system. For better communication between different components of HEVs, a Bluetooth based user interface is developed to show the driver of the vehicle key variables such as the revolutions per minute (RPM) of the electric motor, the speed of the vehicle in miles per hour (MPH) for the current being drawn along with the voltage being applied to the motor in relation to overall power.

II. DESIGN AND DEVELOPMENT

The design of the electrical/electronic prototype began with a block diagram of the system which is developed in stages and initially tested as each stage of the development progressed. The software flow chart is then designed and followed throughout the design and implementation of the system.

From this the development of the prototype was done in stages following the order below:

- Testing of the DC motor and speed controller
- Setting up the hardware and ensuring functionality
- Developing the throttle input
- Addition of the microcontroller and PWM control
- Design and implementation of the speed feedback device
- Introduction of the PID control system
- Developing voltage and current monitoring
- User interface for the system

A. *Block diagram of overall system*

Figure 1 shows a simplified block diagram of the first prototype of the electrical/electronic system of the hybrid electric vehicle design and implementation. So far, a flow chart has been developed for the software programme. This can be seen in Fig. 2, whereas Fig.3 is a graphical concept test diagram. The system starts by testing the initial parameters of the device, such as motor voltage, battery voltage, and motor current and speed. If all of these parameters are within range, the liquid crystal display (LCD) is initialized and modified to ensure the motor is safe to run. Afterwards the ignition switch is tested along with the direction switches. The PID control is applied when the system is ready to operate using the pedal sensor as the input (set point) and the pulse width modulation (PWM) signal as the output using the shaft's speed as the feedback. Once the PID function has been applied, the software code goes back, reading the current and voltages before the LCD screen updates. Upon updating the LCD screen, the hardware switches are integrated, and the PID control is again implemented. It performed well to prove the idea-but the programme code is further streamlined for when a separate microcontroller operates the LCD screen successfully. Also, Figure 5 displays a schematic of the current motor prototype operated by electric and electronic PID.

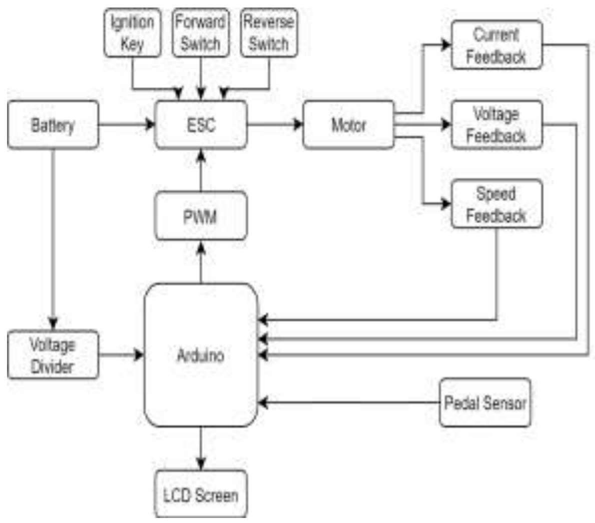


Fig 1: Block diagram of electrical/electronic system of HEV

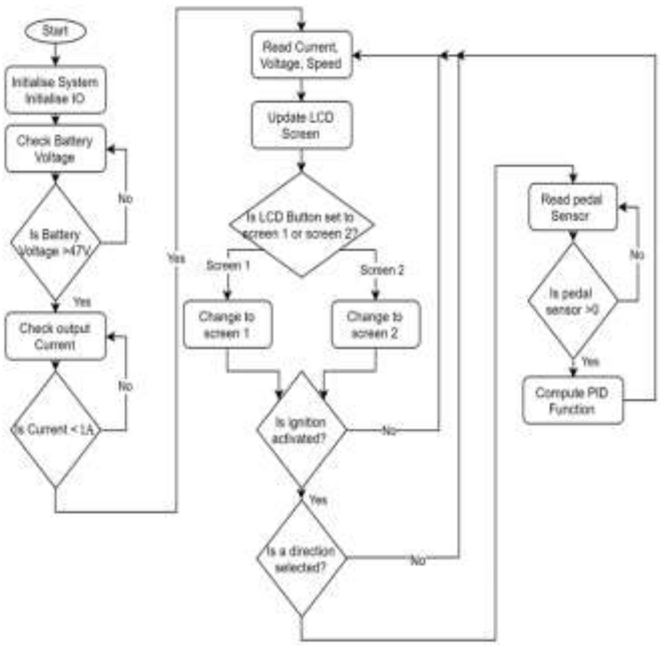


Fig. 2: Flow chart for prototype design concept

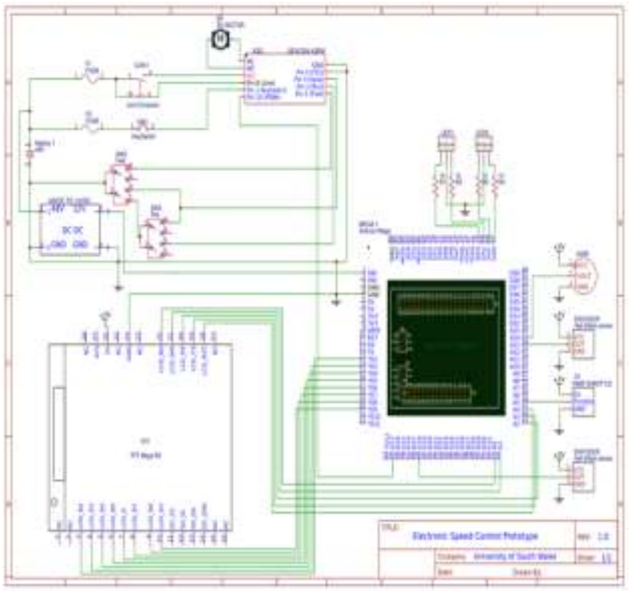


Fig. 3: Schematics of current electrical and electronic PID controlled motor prototype

B. *Initial Testing of DC Motor*

The electric motor obtained was manufactured by Lynch Motors in Honiton, UK and LEM 170, which was a 48 V, 5.54 kW motor, rated at 3264 RPM and 16.2 N-m of torque. The motor was tested to determine if it is working with a bench power supply. The motor no-load current is found to be around 6 A, and it operates at a varying supply voltage of up to 30 VDC. In the service manual, if the motor had not been used in more than two years, it was suggested to let the motor run for half an hour with less than 30% maximum current. Hence it was left running for half an hour during the engine test. The obtained DC electric motor was running at variable speeds and was working as planned. Figure 4 demonstrates the DC motor experimental set-up and test. It is found that the current drawn was between 6-7 Amps, after the motor accelerated to the final speed allowed by the voltage. The voltage is set at different levels during the testing, and the speed of the shaft is determined using a digital tachometer by adding a reflective strip to the shaft.

The test results can be seen in Fig. 5, showing a graph of the motor vs. the voltage applied to the motor. The speed rose proportionally to the voltage applied, which is expected from this DC motor, and the final speed found to be 3320 RPM, faster than the specified in the datasheet. It is because there was no load present on the machine.

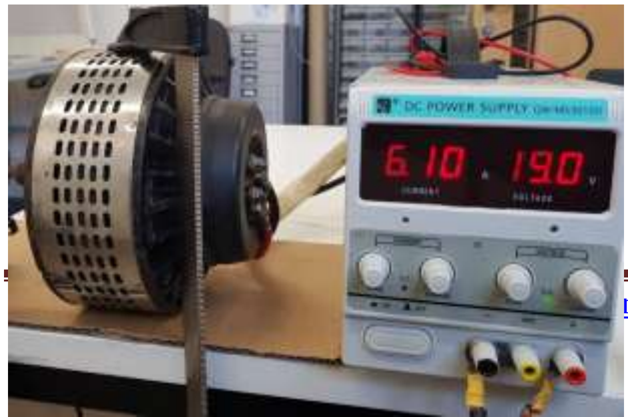


Fig. 4: Testing of the DC motor using a bench power supply

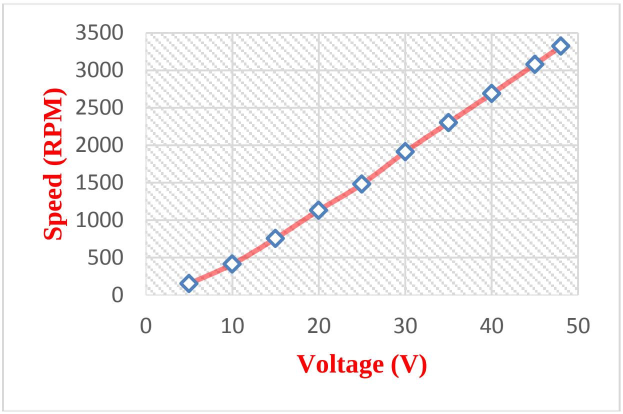


Fig. 5: Applied voltage vs RPM of the shaft

The main issue with this motor was the size, the rated current is 140 A and can peak to 400 A according to the datasheet. If batteries can be obtained that can handle the peak current and rated current, then this motor can be utilized for the vehicle. This motor would be ideal due to the high torque and traction capabilities and would give the vehicle high performance and acceleration. The speed of the motor will need to be limited with a suitable gear ratio and if possible, the controller will be set with a suitable current controlling setting. The motor was found to be fully operational and the speed of the motor was found to be 67.34 RPM per volt which roughly matches what is stated in the datasheet at 68 RPM per volt applied (68 RPM/V).

C. Testing of Brushed DC Controller

A 400 brushed DC controller with a high-power safety with digital and analogue input/output (I/O) was developed, evaluated and tested for motor control in various applications. This controller can be operated by a potential power supply from a divider, hall sensor, or 0-5 V variable. Figure 6 displays the controller. It was designed as an input for initial testing with the power supply and a prototype throttle. This is to ensure that the motor and controller were operating together before any new hardware and software wasinstalled. The devices were wired up according to the standards and setup with simple ON/OFF, forward and reverse switches. All other safety features like seat detection, brake lights, engine braking etc., were short circuited to the negative terminal to deactivate them for initial bench testing. The controller was set up with the power supply and a motorbike hand throttle as an input for initial testing. The setup can be seen in Fig.7, and a simplified schematic of the connections made is shown in Fig.8.The test of the controller is carried out by using double pole double throw (DPDT) switching method. The forward and reverse switches wired in a way that only one signal can ever be applied at any one time by using double pole double throw (DPDT) switches to ensure the fast speed (FS1) signal was applied at the same time. On second attempt the controller gave a steady green light indicating the device is 'healthy' upon a forward or reverse signal being applied to the motor and the motor spun in the required direction according to the input given by the throttle sensor. The controller and the motor worked and provided a solid base to build the rest of the prototype. The voltage going into pin 10 for the throttle signal was measured to know what limits are needed to control the full speed range of the motor. The signal applied was found between 0 – 4 V, where 0 V resulted no rotation of the motor and 4 V had the motor rotating at full speed. The speed of the shaft is also measured to find the full speed of the motor in both forward and reverse directions. The speed of motor in reverse is limited to 60% of the full speed in default configuration. The top speed of the system was found to be 3100 RPM in forward and 1780 RPM in reverse with 48 V being used as the supply voltage. This was measured with the digital tachometer. From this, a microcontroller was used to control the throttle signal and all the digital and analogue I/O of the controller. Due to this, extra monitoring and sensing was added to increase the safety of the system and develop a feedback control loop for the DC motor using PID control.

Fig. 6: 4PQM controller

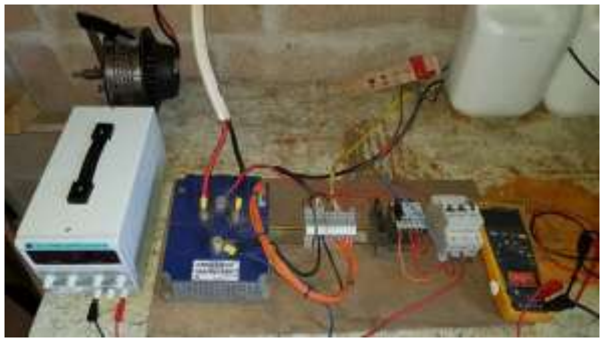


Fig. 7: 4QPM controller and LEM 170 DC Motor testing

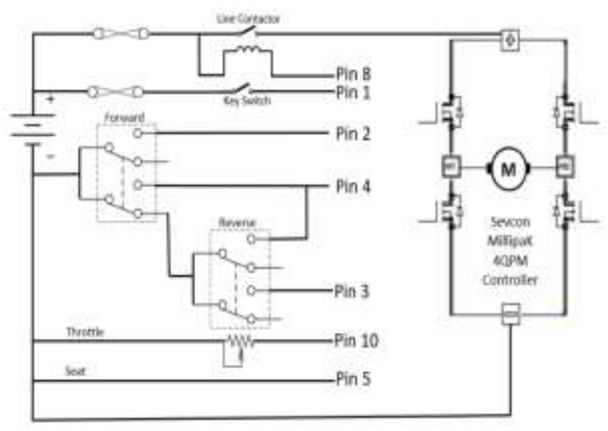


Fig. 8: Simplified schematics of wiring

D. Throttle Development for Electric Motor

The first step in implementing the brushless DC motor controller's microcontroller input was to procure a foot pedal and test a microcontroller to read its output. The selected microcontroller was an Arduino Nano, and the chosen foot pedal was a typical pedal used on electric motorcycles. The device runs on a 5 V input and produces a 1 V to 4.2 V output signal, depending on how much the pedal is pressed. To help identify I/O pins and communication protocols, a basic Arduino Nano was used. The foot pedal was connected to a 5V supply, and the output was connected to a multimeter to ensure the output voltage was within the range of the analogue reading capabilities of the Arduino. The Arduino can read up to 5 VDC and it is found that the reading ranged from 0.9 V to 4.3 V while checking the foot pedal which is slightly different from the definition of the device. This commodity does not have a datasheet, so it was manually checked. The output was found to be within the range of analogue reading capabilities of the Arduino. However, using one of the digital inputs, the foot pedal was connected to the Arduino and an analogue read function was used to read the pedal sensor voltage output. In the analogue ports the Arduino uses a 10-bit analogue to a digital converter (ADC). Using equation 1, the full analogue input set (5V) can be transformed to 1024 divisions, otherwise known as ticks or divisions:

$$2^n = \text{Maximum deviation} \qquad (1)$$

where n = the number of bits of the ADC

From this, the number of ticks can be determined and inserted into Arduino software for the minimum and maximum outputs of the pedal sensor. If 5 V is equal to 1024 ticks, then 184 ticks are represented by 0.9 V and 881 ticks are equal to 4.3 V. For reading the pedal sensor, a simple code was inserted into the Arduino, written in the structured text to read the analogue input which is a programming language based on C++. This is expressed in code listing 1.

Code listing 1: Arduino code for initial reading throttle value

```
int Throttle = A0;                  \\ Defines throttle analogue pin
intPedalSensor; \\ Variable to store throttle reading
void setup (){
Serial.begin(9600);                 \\ Setup initialising serial monitor}
void loop (){    \\ Loop program
PedalSensor=analogRead(throttle);              \\ Reads input & assigns to
variable
Serial.println(PedalSensor)              \\ Prints the value to serial Monitor
```

This software reads the analogue input continuously and displays the value on the Arduino's IDE serial display. Once the programme has been updated and checked, the pedal sensor range has been found to vary from 185 ticks to 882 ticks similar to the previously measured values. The calculated values were found to be slightly unstable and fluctuated by ±10 ticks on the serial display. In order to increase the efficiency of the analogue read function, the program's "analogue read" line has been replaced with a function that takes an average of 10 readings instead. The code for the latest read sensor feature is listed in code listing 2.

Code listing 2: Updated code for Arduino reading the throttle input

```
voidReadSensor(){
int readings [10];// Readings from the input
intreadIndex=0;                 // Current reading
int total =0;                   // the running total
  total = total - readings[readIndex];          // Read from the sensor:
  readings[readIndex]=analogRead(throttle); // New value to an index
  total = total + readings[readIndex];// Adds to the total
readIndex=readIndex+1;     // next position in the array
if(readIndex>=10){
readIndex=0;                    // Resets array after }
```

This feature generates an array and will read the analogue input 10 times before updating and restarting the array. This takes an average of readings before updating the list and assigns it to the new variable Pedal Sensor. This code produced more consistent results and it was found that the number of ticks fluctuated between ±2 rather than ±10, a fluctuation of ±2 ticks results in an average fluctuation of 0.01V which is an appropriate pedal sensor error. The next move was to map the tension reading of the pedal sensor to a value of 0-4V to replicate the input of the hand throttle on the DC motor controller.

E. Microcontroller Design for Input Control

A digital interface with PWM capabilities must be used to output a variable voltage from the Arduino to provide a preset DC voltage to the speed controller. While testing the speed controller it was found that inputs from 0 to 4 V are needed to vary the motor from no speed to full speed. The Arduino has the capacity to output 5 V from any of its digital outputs and PWM can be used to adjust this voltage within the range of 0 to 5 V. The Arduino's digital to analog converter (DAC) is an 8-bit DAC resulting in 256 equation divisions. When using the PWM output, the Arduino has a range of 0 to 255 ticks as it starts from 0 and not 1. It means that writing 255 to the digital output will provide 5 V while writing 0 will provide 0 V. Thus, in order to provide a maximum output of 4 V, the digital output must be reduced to 204 ticks which will produce a maximum voltage of 4 V. This means that the input of the pedal sensor, which was found to be 185 to 882 ticks, must be mapped to 0 to 204 ticks while writing to the PWM output. To do this, a map was used to simplify the involving mathematics and save memory for CPU processing. Virtual output was allocated as the input of the throttle electric velocity control (ESC) and the pedal sensor was mapped using this pin to provide a 0 to 4V output. In this paper the velocity output is measured by a tachometer and fed back via an ADC. It is important to note that the tachometer converts the velocity to a voltage; that voltage is then used as input in the negative terminal. The positive terminal is connected to a reference voltage. For this the latest loop code can be found in code listing 3.

Code listing 3: Arduino code for mapping throttle reading to the PWM pin

```
void loop (){
ReadSensor ();                  \\ Reads average pedal sensor reading
Serial.println(PedalSensor);                    \\ Prints Pedal Sensor
Reading
ThrottleOut=map (PedalSensor,180,880,0,200);\\ Maps Pedal value
if(ThrottleOut>0){
analogWrite (3,ThrottleOut);\\ Writes PWM value to digital output }}
```

This was plugged into the Arduino and the PWM output was connected with a multimeter, the voltage was calculated while the pedal was pressed to ensure that the voltage output was within range for the ESC. As the pedal was pressed it was noticed that the Arduino's output was within the range of 0 to 4V and was working as expected. The Arduino's PWM output was then wired to pin 10 and the ESC's negative connected to replace the hand throttle for checking the Arduino as an input throttle. The Arduino was powered up using the PC, and the power supply was used to control the DC motor equipment. The machine was tested both in forward and in reverse when calculating Arduino's throttle input. The system ran all right and ran at full speed in both forward and reverse and repeated the previous test's use of the hand throttle. The speed was again found to be 3100 RPM in forward and 1780 RPM in reverse using the optical tachometer using 48 V as the supply voltage. The Arduino circuitry added for the speed controller can be found in Fig.9 in a simple schematic. The next step is to establish a feedback on speeds so that the PID controller has a reference point.

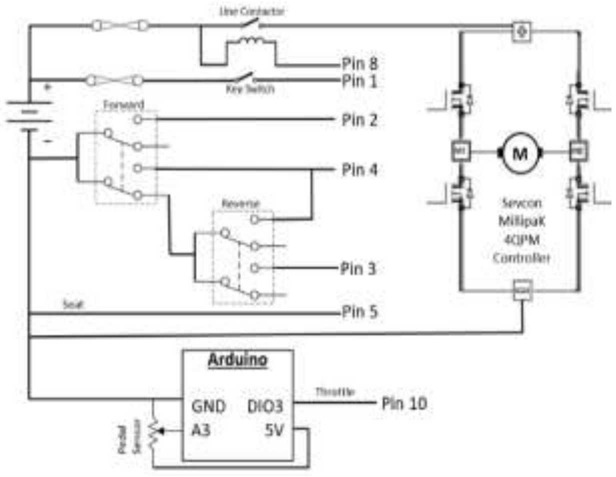


Fig. 9: Simplified schematics of system with foot pedal and Arduino

F. Development of Speed Encoder System with Hall Sensor

A speed feedback system is required in order to develop a PID controller for the DC motor. For this a hall sensor and magnets were used in order to develop an incremental encoder. The magnets were placed onto a circular disk that was attached to the motor shaft; 8 magnets were used at equal distance to provide 8 pulses per revolution. The disk was attached to the motor shaft and the hall sensor module attached to a stationary mast within sensing distance of the encoder disc. The hall sensor module was powered up with 5 V and the output was connected to a multimeter to ensure that the system was working correctly. The shaft was spun manually and when a magnet was in sensing distance of the hall sensor the indicating LED lit up and the multimeter read 4.9 VDC. This indicated that the encoder sensor was working correctly and gave a HIGH output whenever one of the magnets was sensed. Figure 10 shows the implementation of the encoder disc and hall sensor on the motor when it detected one of the magnets.

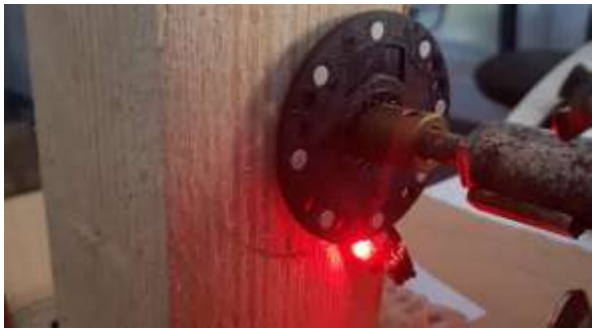


Fig. 10: Encoder system while magnet was being detected

When the physical side of the encoder was confirmed to be working, the hall sensor was connected to the Arduino. The hall sensor was powered off the 5V rail from the Arduino and the output was connected to digital input 2, which has a hardware interrupt feature incorporated into it, which can be used to detect the amount of rising edges given by the hall sensor in a given amount of time. Once the encoder was connected, a series of functions were developed to detect the number of pulses within a given time frame to deduce the RPM of the shaft. The code relating to the encoder feedback can be seen in code listing 4. This starts a timer and counts the amount of pulses given by the hall sensor until the timer is stopped. The amount of pulses is multiplied by 240 to calculate the theoretical amount of pulses in a minute at that speed and then divided by 8 as that is the number of pulses in one revolution. This code was added to the previous coding with initializing the new variables in the setup function. The system was then run using the Arduino as a throttle input while monitoring the speed by a digital tachometer and monitoring the speed given by the Arduino from the encoder. To perform comparison for accuracy, the speed was held constant with the pedal sensor while the readings were taken off both the tachometer and the Arduino. Figure 11 shows the results and it is clearly seen that the RPM measured from the encoder by the Arduino and by the tachometer were found to be within a reasonable range of each other with a maximum error of 3% at lower speeds.

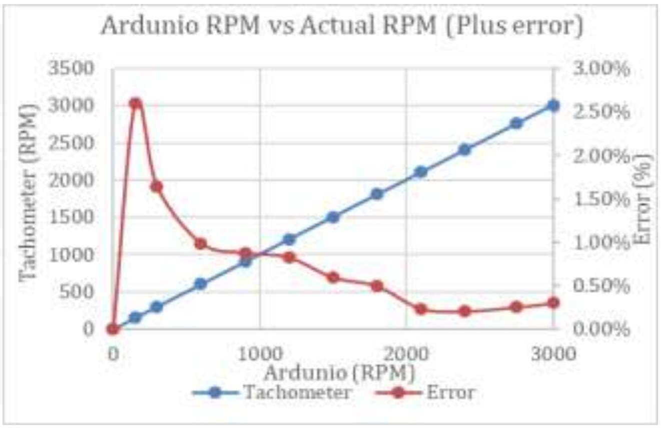


Fig. 11: The Arduino RPM vs actual rpm

The error decreased as the speed of the motor was increased until there was an error of less the 0.5%. This is within an acceptable range to start developing a PID control using speed as the feedback signal. To output a variable voltage from the Arduino a digital output is used with PWM capabilities in order to provide a variable DC voltage to the speed controller. When the speed controller was tested it was found that an input of 0-4V is required to vary the motor from no speed to full speed.

Code listing 4: Arduino code for deducting the speed from the encoder

```
#include <TimerOne.h>
voiddocount()                    // counts from the speed sensor
{
counter++;}                      // increase +1 the counter value
voidtimerIsr(){
 Timer1.detachInterrupt ();                 // Stops the timer
 RPM =(counter*30);// RPM= counter x 240 (for minute) &divide by 8
(pulses)
Serial.print(RPM);                          // Prints RPM to serial
monitor
 counter=0;                                 // reset counter to zero
 Timer1.attachInterrupt (timerIsr);}//enable the timer
voidReadEnc(){
 Timer1.initialize (250000);                // set timer for 0.25sec
attachInterrupt(0,docount, RISING);// increase counter when pin goes
High
 Timer1.attachInterrupt (timerIsr);         // enable the timer}
```

G. Development of PID Control for DC Motor

The PID control built in this system used the pedal sensor as the set point and the input taken from the velocity sensor while the output is the PWM level sent to the ESC. Instead of using the shaft's RPM as the velocity sensor data, miles per hour (MPH) was used to control the device by the vehicle's final velocity after considering the gear ratio. To do this, the shaft speed in RPM has to be converted to MPH by taking into consideration the driving wheel diameter and gear ratio to calculate the vehicle speed. A gear ratio of 1:5 was used to create the PID power, and 18-inch diameter wheels, as these are the intended attributes used in the mechanical construction. When the vehicle was assembled with real gearing ratio and wheel diameter, this was fine-tuned in software. Using the above series and measured circumference, the circumference of an 18-inch diameter wheel was determined to be 1,4363 meters. For the motor, the following formula was proven to turn the motor shaft's maximum RPM into MPH.

$$MPH_{max} = 3264\ RPM \div 5\ x\ 60 min x\ 1.4363 metre \div 1609.3 metre$$

$$MPH_{max} = 34.96\ MPH\ \therefore\ 1\ MPH = \frac{RPM}{93.36}$$

From this, it was assumed that the vehicle should run at 1 MPH at 93.36 RPM. In the programme, this equation can be used to measure the speed of the vehicle at any given time. This latest reading of MPH has been used as feedback to the PID system. Instead of being connected directly to the Arduino PWM output, the pedal sensor was mapped to a limit of 30 MPH and this new measured value was used as the controller's set point. It ensures that a relative ideal MPH can be measured and used as the set point, since the pedal is pressed from nothing to absolutely press. The PWM output of the Arduino has now been set as the output of the PID controller. It changed the control system from an open loop system to a closed loop system which could attempt to reach the speed set by the pedal sensor by adjusting the PWM output accordingly. With the proportional constant (Kp) set to 1 and the integral and derivative constants (Ki and Kd) set to 0 ready for the Ziegler Nicholas tuning method to be used, this was implemented in Arduino coding. The code listed in code listing 5 was uploaded into the Arduino and tested with only the Kp set to 1. To test the system the pedal sensor was removed through software and the set point was set to 15MPH to have a stable set point. The Kp value was increased in increments until the motor speed was oscillating just below the set point. The final Kp value that was found to be optimal with a constant of 12. Using the Ziegler Nicholas method, the Ku value was taken to be 12 and the Tu value was calculated in software to be 1.5 seconds between oscillations. These values were entered into the PID controller and the system was re-run with a set point of 15 MPH.

Code listing 5: Initial PID coding used in the Arduino

```
#include <PID_v1.h>                          \\ Includes the PID library
double Input, Output, Setpoint;              \\ Variables for PID
elements
doubleKp=1, Ki=0,Kd=0;                       \\ Variables forKp, Ki &Kd
PID PID1(&Input,&Output,&Setpoint,Kp, Ki,Kd, DIRECT);   \\PID
configuration
voidRunPID(){                                \\ Function to run the PID
loop
  MPH = RPM /93.36;            \\ Converts the RPM into MPH
  Input = MPH;                               \\ Sets MPH as PID input
  Setpoint =map (Tavg,175,870,0,30);         \\ Maps throttle to 0-
30MPH,
Sets PID setpoint
  PID1.SetOutputLimits(0,200);\\ Sets the PWM limits of the PID output
  PID1.SetMode(AUTOMATIC);\\ PID configuration
  PID1.Compute();                            \\ Computes the PID
Function
analogWrite (ESCPin, Output);                \\ Attached PID output to
the PWM pin for ESC}
```

Figure 12 shows the plot of the set point and actual motor speed for the various PID constants. In Fig. 13 to Fig. 15 the red signal represents the set point and the blue signal represents the speed in MPH that is read from the speed sensor, these figures also show the various plots with trials of various PID constants at various set points. The PID controller was found to be within reasonable tolerance and the actual speed was within ±1 MPH of the setpoint. Next the stationary setpoint was removed from the program and the pedal sensor was used as a variable input and the actual speed and set points were plotted again while the pedal sensor was varied. The system was found to respond quickly and reach the set point at every time. The PID constants can be tuned further to improve the response time and system stability, however this shall be done at the end of the development in order to consider the delay time that was added due to extra software and hardware developments. Also, no load was placed on the motor during this tuning process which may alter the final PID constants required when implemented into a vehicle.

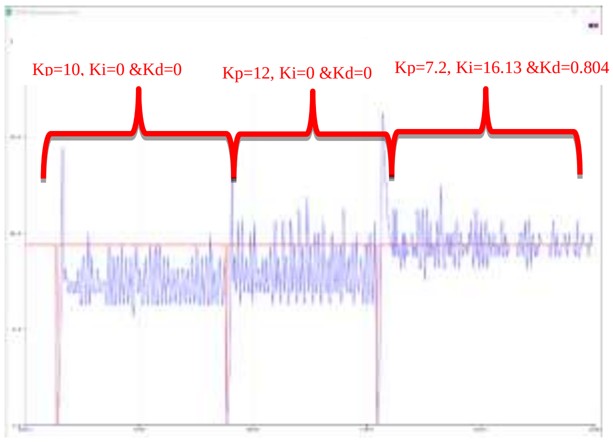


Fig. 12: Actual speed vs set-point from Arduino IDE (Kp=7.3, Ki=16.13 &Kd=0.804)

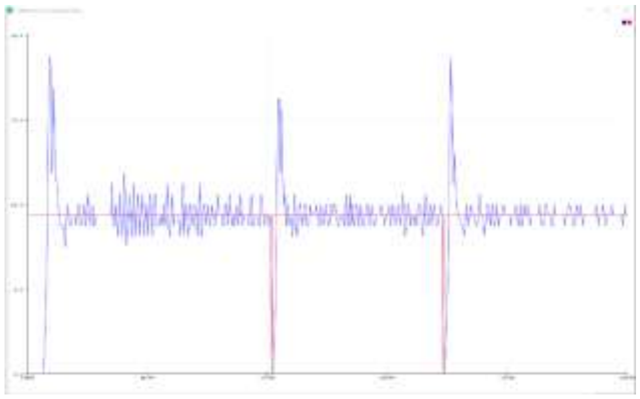


Fig. 13: Plot of actual speed vs set-point from Arduino IDE (Kp=6 Ki=10, Kd=0.6)

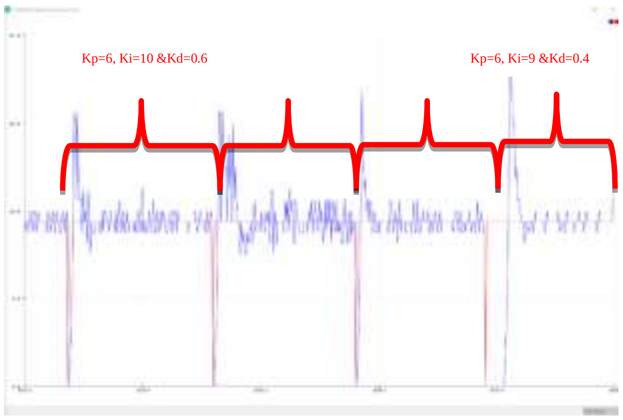


Fig. 14: Plot of actual speed vs set-point from Arduino IDE (Kp= 6, Ki=9, Kd=0.4)

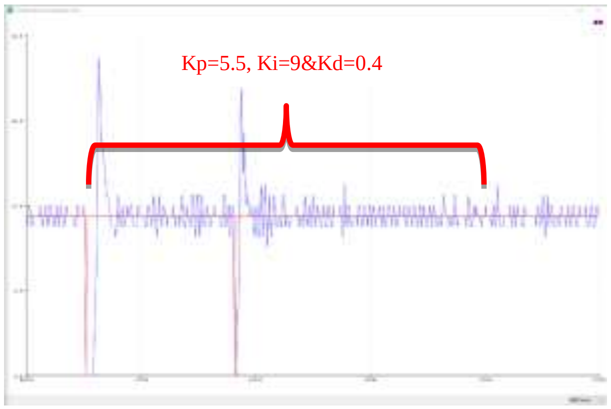
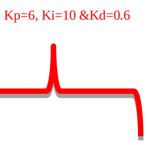


Fig. 15: Plot of actual speed vs set-point from Arduino IDE (Kp=5.5, Ki=9 &Kd= 0.4)

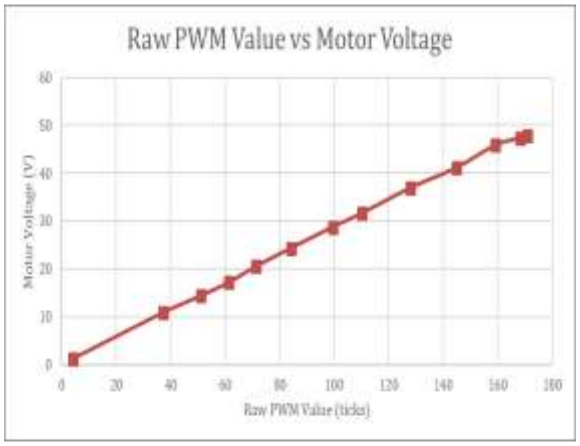


H. Voltage and Current Measurement

The voltage and current needed to be monitored in order to know the power going into the motor at any given time. The voltage sensor needed to monitor from the motor which could result in a positive or negative voltage depending on weather the motor is running in forward or reverse. The Arduino can only monitor a positive voltage therefore the negative voltage cannot be monitored in the traditional way. There are two ways in which this can be achieved easily, either:

- Develop a voltage divider for the motor where the output is tied to 2.5 V so that the positive voltage ranges between 2.5 V and 5 V and the negative voltage ranges between 0 V and 2.5V.
- Use of blocking diodes in order to block the negative voltage and convert it into a positive voltage almost like a rectifier system.

In this research work, the motor voltage and PWM signal used to run the electric speed controller to determine the voltage and RPM of electric motor. It was found that the level of PWM being applied to the



ESC is directly proportional to the voltage being applied to the motor at no load conditions. Table 1 shows the results of this experimental work and fig. 16 shows a graph relating the PWM ADC value from the microcontroller to the motor voltage.

TABLE 1
Experimental results (PWM vs motor voltage testing)

PWM Voltage	Arduino PWM	Motor Voltage	PWM to Motor Voltage Ratio
0.08	4	1.17	3.487179487
0.73	37	10.8	3.447222222
1.2	61	17.1	3.578947368
1.4	71	20.5	3.482926829
1.65	84	24.3	3.462962963
1.95	99	28.7	3.465156794
2.16	110	31.6	3.486075949
2.51	128	36.9	3.469105691
2.84	145	41.1	3.524087591
3.12	159	45.8	3.474235808
3.3	168	47.3	3.558139535
3.35	171	47.8	3.574267782

3.5	179	47.8	3.734309623

It was found that the motor voltage was about 3.5 times less compared to the raw Arduino PWM (ticks) value being applied. Also, the maximum voltage applied to the motor (48.7 V) was applied when the raw PWM value reached 171 ticks. This was programmed into the Arduino by mapping the output PWM to the motor voltage by using the ratio of 3.5. This will be upgraded to a fully functional sensor in the final prototype before implementing the system into a vehicle. The current sensor was incorporated in order to measure the current flowing into the motor when running normally and the current flowing into the battery in regenerator mode. The current sensor used was a LEM CKSR 50NP which had a supply voltage of 5Vdc and an output of voltage between 0.325 V and 4.625 V with 2.5 V as the 0-reference point. The theoretical sensitivity of the current sensor was 12.5mV per amp. The voltage supply of the current sensor was connected to the Arduino's 5V rail and the output was connected to one of the Arduino's analogue reference pins. With 2.5 V being the 0 amps reference point that means that the full ADC value of 1024 relating to 5 V would be divided by 2 to give the raw value of 512 ticks. The sensitivity of 12.5 mV/A will refer to 2.56 ticks per amp, which can be programmed into the software to be able to deduct the current flowing to or from the motor.

Fig. 16: PWM raw value vs the motor voltage

Once the current sensor was connected to the Arduino, the program was increased to incorporate the reading of the analogue pin and calculating the current from the raw ADC value (ticks). The program code used to calculate the current can be seen in code listing 6.This was uploaded on a separate Arduino and used on a charging and discharging test for a battery module in The Centre for Automotive and Power Systems Engineering (CAPSE) building in the University of South Wales. The current was varied from 0 A to 120 A in steps of 5 A every 10 seconds, the current was measured by the battery test equipment and compared with the current on the Arduino.

The results for the positive current test can be seen in Figure 17(a). The results for the negative current test can be seen in Figure 17(b). The MCV – EV/HEV battery cell tester used to test the current can perform life cycle tests and automotive battery testing up to 300 A, more can be put in series for up to 2700 A battery testing. The MCV allows for voltage, temperature, current, capacity and pressure monitoring alongside constant current and constant resistance discharging.

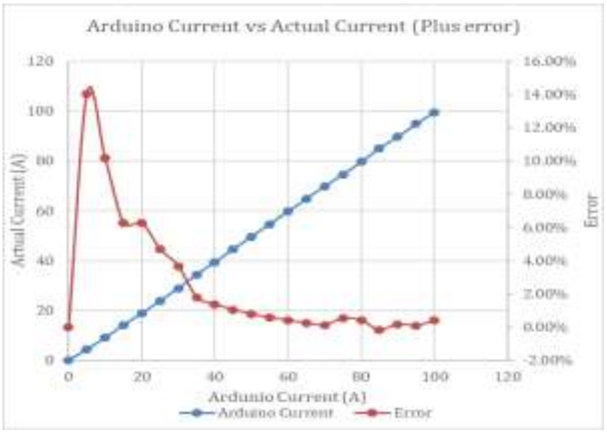


Fig. 17a: Current from the sensor vs actual current

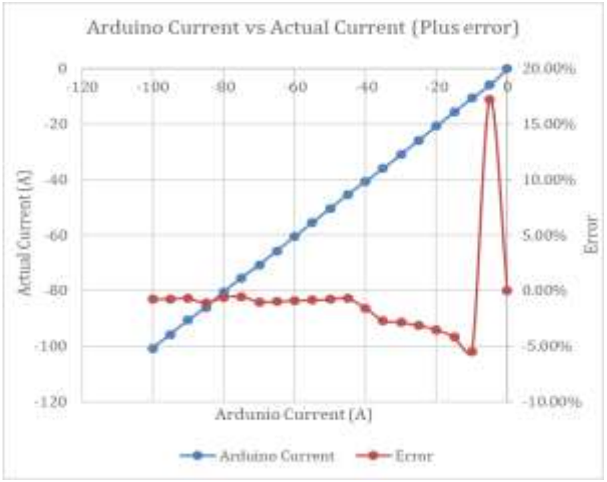


Fig. 17b: Current from the sensor vs actual current

Code listing 6Arduino function for current reading

```
voidReadCurrent(){
int readings[10];                         // Readings from the
analogue input
intreadIndex=0;                                      // the index
of the current reading
int total =0;                                        // the
running total
  total = total - readings[readIndex];      // Read from the sensor:
  readings[readIndex]=analogRead(A11);      // Log new value
  total = total + readings[readIndex];      // Add reading to total
readIndex=readIndex+1;      // Increment array position
if(readIndex>=10){
readIndex=0; }                            // Resets array after 10
readings
Amps_ADC= total /10;        // Takes the average of the readings
 Amps =(Amps_ADC-512)/2.56              // Converts ADC value into
amps
}
```

I. Development of User Interface

A user interface has been developed to inform the driver about the main variables of the motor, such as the motor's RPM, the vehicle speed in MPH and the current being drawn and the voltage applied to the motor alongside the total power. An Arduino 3.5-inch TFT touchscreen, a comprehensive colour screen intended to develop a vehicle user interface, has been used for this purpose. The screen was compatible with Arduino Nano, Uno, and Mega and came with a CD to follow with lots of examples and instructions in order to learn how to use it and how to run it. This allowed to build a user interface separately on an Arduino Nano and the screen can be seen in fig. 18.

Following the pinout diagrams seen in Figure 14, the device was linked to the Arduino Nano. The examples on the CD were transferred to the Arduino, and each was studied separately in order to understand how to modify the code for this research needs. This subsequently allowed to develop a user interface that displayed two different screens, which could be combined with a momentary button. On a monitor, the first screen showed the vehicle speed in MPH, and the RPM on a vertical bar graph. The second screen showed the motor power and current drawn, as well as the voltage of the battery. These are represented in fig.19.

Fig. 18: Arduino 3.5" TFT touch screen used for the user interface (Back and Front)

The developed user interface to inform the driver about the main variables of the engine, such as the motor's RPM, the vehicle speed in MPH, the current being drawn, and the voltage applied to the motor along with the total power is shown in fig. 20.

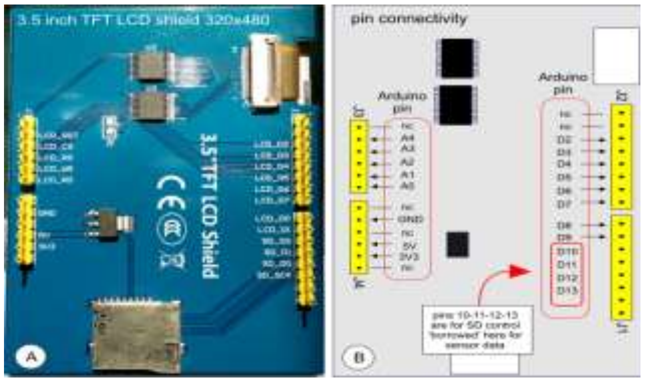

3.5 inch TFT LCD shield 320x480
3.5"TFT LCD Shield
pin connectivity
Arduino pin
Arduino pin
pins 10-11-12-13 are for SD control 'borrowed' here for sensor data
A
B

Fig. 19: Arduino screen pinout for Arduino Mega.

Fig. 20: User interface developed for speed, battery voltage and current

J. Bluetooth System for User Interface Communication

A Bluetooth system was developed as shown in

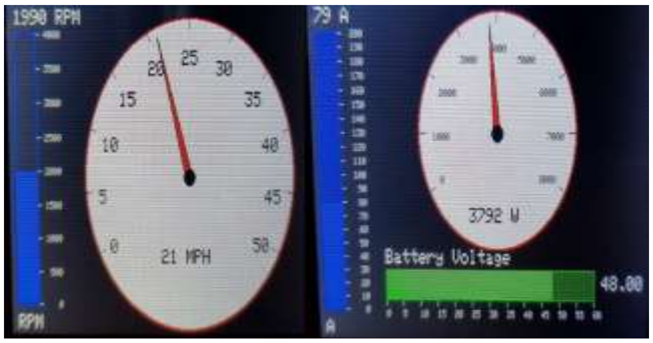


figure 21 to send data between the user interface and the main microcontroller controlling the ESC. Fig. 21 shows a simple diagram of the Bluetooth system developed using two Bluetooth modules (HC05 and HC06).

K. Design of Series Hybrid Electric Vehicle

A design was developed for two simple types of series hybrid powertrains to be incorporated into the vehicles system. The first design incorporated a battery source and an AC generator in parallel. The AC generator would be selected to be able to run the vehicle at slower cruising speeds with limited current capabilities in case the batteries were completely depleted. In normal operation the motor would be run off the power from the battery and the generator would be started automatically when the batteries reach a certain level of state of charge (SOC). The batteries would be charged off the generator or supply the motor with the excess current required via the batteries. Figure 22 shows a block diagram of the first series hybrid system being considered.

The second designed series hybrid system incorporates two sets of 48 V batteries where only one will ever be used at one time. In this system one battery pack will be used to run the vehicle while the other is on standby. If the battery on standby requires charging, then the generator will start and charge the battery through the rectifier and charging circuit. This system is preferable to the first system by being theoretically better than the first system as there will always be a charge battery pack. Figure 23 shows the block diagram for this system and again the contactor resembles a suitable power device to switch in the two batteries or the charging circuit. In both systems the microcontroller will decide when to switch in the generators, batteries and charging circuits. Also, in both systems the generator can be used to charge the vehicle's batteries while the vehicle is not being used.

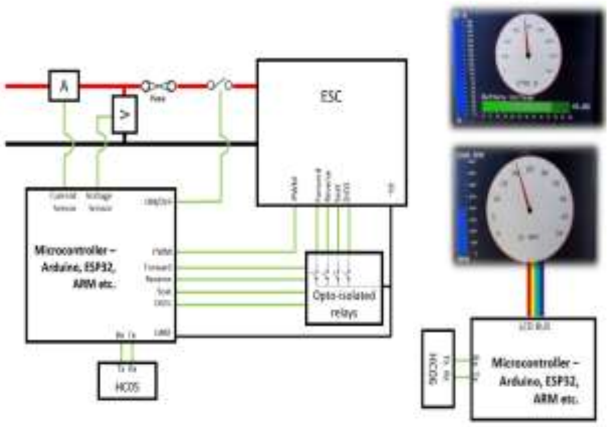

A
ESC
Current Sensor
Voltage Sensor
Microcontroller – Arduino, ESP32, ARM etc.
Opto-isolated relays
HC05
Microcontroller – Arduino, ESP32, ARM etc.
HC05

Fig. 21: Schematics of Bluetooth connected LCD screen & opto-isolated relays for digital IO

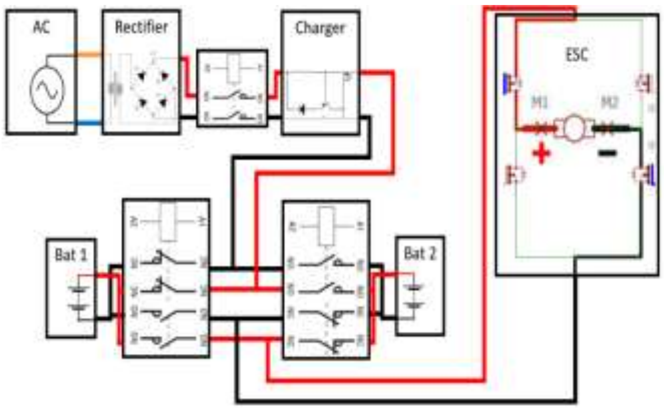


Fig. 22: First design of the series hybrid system

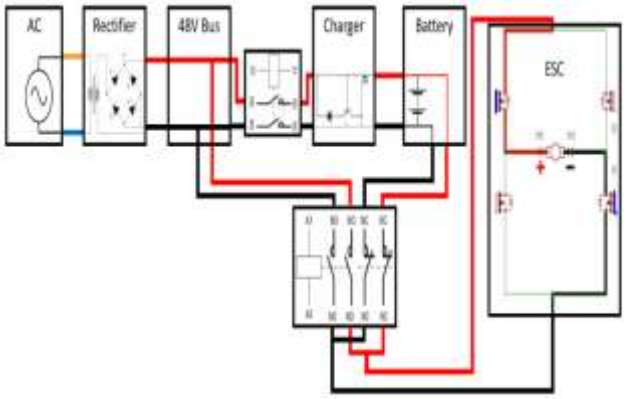
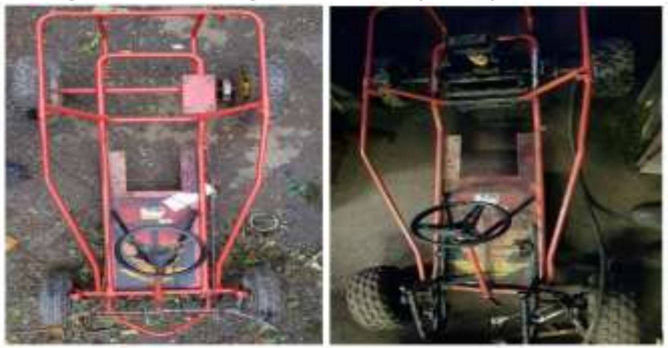


Fig. 23: Second design of the series hybrid system



L. *Frame and Mechanical System Development*

An off-road go-kart frame is used. To date, the frame has been slightly modified to include a live rear axle, and a suspension back and front. This frame is large enough to support one person and weighs 50 kg before any on-frame construction. Figure 24 displays the work done so far on the go-kart frame, before and after images.

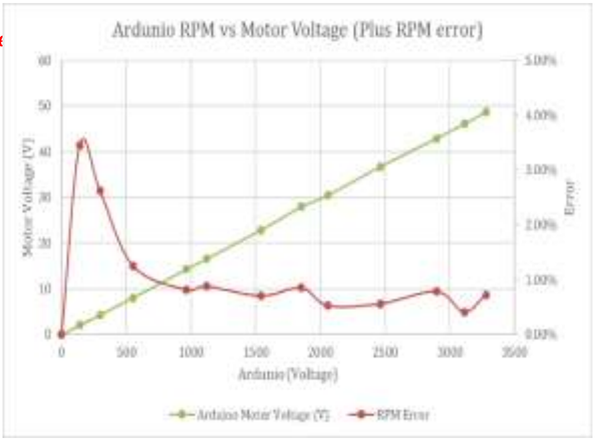
Fig. 24: Prototype before and after modification

III. TESTING AND RESULTS OF THE DESIGNED PROTOTYPE

A full installation of prototype rig is shown in fig. 25. The first stage of development of the rig was to take the engine and mount it onto a sturdy right-angle frame, which was attached to a base plate, and the base plate was attached to the rest of the circuitry. The hall sensor and the magnetic disc were connected to the motor shaft, ensuring the hall sensor and magnets were within a reasonable distance, and the hall sensor was facing the south side of the magnets. With an aluminium block used as a temporary heatsink, the speed controller was added to the system. With four holes drilled in and tapped to M6, the aluminium block was cut to size to allow the electronic speed controller to tighten it down. A coating of heatsink paste was added to the heatsink before this was finished to facilitate the thermal flow of any excess heat. Instead, as stated in the manual, the ESC was tightened to the aluminium bleak on the rig with a torque of 6 nm. The ESC's digital I/O was attached next to the ESC, and all cables were made as short as possible. The path switches were mounted, and the ignition switch was changed to a key switch for the final vehicle to be integrated. The machine integrated a DC to DC converter to provide 12 V from the battery system to the Arduino. The Arduino Mega was mounted on the base of the rig and all connections were made for the ESC, foot pedal, hall sensor, and current sensors. The screen was mounted on a wooden base and then mounted onto the rig. This was wired to the Arduino and all the wiring was checked to ensure that everything was properly connected, and there was no chance of damaging any of the first power-up equipment. Four lead acid batteries were placed on the trolley's bottom shelf and attached to the ESC. There was an opto-electronic operated relay circuit that could handle the voltage and current of a generator or petrol engine's electrical start circuit depending on the design. This allowed full power control to the IC engine's electrical start circuitry while being electrically isolated from the microcontroller in any case of high potential difference. This was tested with a small DC motor, replicating the starter motor–a high digital output from the microcontroller is required to activate the relays. They worked extremely well and will be used to track any digital signals needed to control the ESC, such as forward, reverse, start, and reset.

The testing of separate parts of the prototype was conducted as they were being developed and implemented into the system. The system was developed in stages in order to make sure that everything was working before adding the next stage of the design. Once the prototype was in the finishing stages of completion, a general overview test was completed on each design stage to ensure everything was working as intended. The test procedure included the following steps:

- Powering up the system and ensuring that no voltage was being applied to the motor at start up and the controller powered up into a healthy state
- Switching direction between forward and reverse while engaging the key-switch and ensure no voltage was applied to motor when foot pedal was not engaged
- Apply foot pedal in forward and measure PWM and motor voltage to ensure UI is within ±2V of actual voltage
- Measure current with current clamp and ensure within ±1A of UI value
- Ensure speed is constant and steady when foot pedal kept in a certain position
- Measure RPM and make sure within ±50 RPM of UI value
- Power motor to full speed and ensure does not pass 30MPH
  - Disengage direction switch and ensure ESC regens into battery
  - Ensure ESC not engaged when both forward and reverse selected
  - Repeat steps 3 to 10 in reverse
  - Ensure the ESC stays in a healthy state throughout the testing and after direction switches disengaged
  - While ESC disengaged ensure foot pedal does not run motor



- Apply both forward and reverse switches and ensure the ESC goes into a neutral state and foot pedal does not engage the motor 30MPH

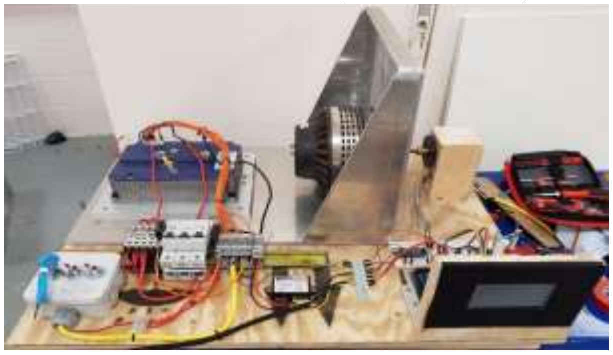


Fig. 25: Full installation of the first prototype rig

The test procedure was completed three times to ensure that the systems was operating in a safe manner, the system was repeatable, and everything worked as intended. The PID constants were altered slightly through the testing procedure as to optimize it further. The system was found to be operating as expected and the ESC powered up in a healthy state and stayed in a healthy state until the key switch was disengaged. The forward and reverse switches operated as intended and the motor was disengaged while no direction switches were selected or when both switches were selected. When the direction switches were disengaged the motor successfully regenerated power into the battery and was found to apply a maximum of 8 Amps back into the battery in no load conditions and the motor was decelerating from full speed. When the motor was set to full speed using the foot pedal it was found to overshoot for a very short duration to 31 MPH before settling at 30 MPH for the full duration while the foot pedal was fully pressed. This was acceptable for the prototype system being presented, however, the PID constants will need to be re-tuned when the system is incorporated into a small vehicle and the motor is actually loaded under various conditions. The motor RPM deduced from the encoder system was found to be within 24 RPM of the measured RPM from the non-contact tachometer. The RPM was also found to be directly proportional to the voltage applied to the motor in no load conditions. Fig. 26 shows the results for the first conducted, the results for the Arduino RPM and tachometer show a similar error to the initial testing of the encoder system in the development stage. The maximum error between the Arduino RPM and the actual RPM was found to be 3.45% at 140 RPM and went down to below 1% after 550 RPM. The speed constant was calculated by using the Arduino RPM and voltage applied to the motor. The average speed constant was found to be 67.38 RPM per volt, the datasheet of the motor claims that the motor has a speed constant of 68 RPM per volt. The speed constant from the tests was found to be very similar to the speed constant stated in the datasheet. Figure 26 shows a graph comparing the applied motor voltage and the RPM measured from the encoder using the Arduino and the overall RPM error found.

Fig. 26: Arduino RPM vs motor voltage including RPM error

The PWM from the Arduino was measured throughout the testing and compared with the motor voltage. It was found that an average of 14.46 volts was applied to the motor for every 1 volt of PWM from the Arduino for the linear part of the results. Again, the relationship was found to be directly proportional until the Arduino was supplying 3.35 volts to the ESC where no more voltage was applied to the motor if the PWM duty cycle was increased. The motor voltage was measured and compared to the calculated value used in the user interface.

The voltage measurement for the Arduino needs to be updated to an actual monitoring system. However, in the meantime the calculated motor voltage was found to be within 1 percent of the measured value. The current was measured with a clamp meter when the motor speed had settled and stopped accelerating/ decelerating. The average current drawn at no load conditions was found to be within 6-7 Amps when measured with a current clamp throughout the testing procedure. During acceleration the current was found to go up to 21 amps when the pedal is fully pressed from when the motor is at rest. The Arduino derived current was found to be higher than the measured current with an error ranging between 11% and 18% at the lower end of the current reading. This is suspected to be due to the current range being on the low end of the measuring range of current sensors. The motor was found to be oscillating slightly when the system reached a steady state which indicates that the PID constants need re-tuning for the extra software coding added into the Arduino due to the addition of the LCD screen. When the communication process between two microcontrollers has been fully developed then the efficiency of the PID programming code will increase. The first prototype of the electrical system to be developed into a hybrid system was found to be operational and working well. The PID control of the motor was found to be satisfactory for now but will need re-tuning with further development of the prototype.

IV. CONCLUSIONS

In this research, an innovative PID controller, control of DC electric motor and Bluetooth system for user interface communication have been successfully designed, analysed, simulated, built and tested on a test rig. The worked carried out reflects the DC electric motor configuration and experimental testing to control the velocity for HEVs. The simulation results show that the PID control algorithm can improve the performance of DC electric motor speed at different set-points, Kp, Ki, Kd, in order to achieve reliable and stable speed. Also, the design and implementation of innovative controller shows that how PID based controller, controlling the voltage and current of DC electric motor to improves the performance of hybrid electric vehicle. In addition, the current and voltage optimization techniques and control algorithms have also been included to improve the efficiency of DC electric motor. It has also been established from results that without appropriate PID controller the output voltage cannot be controlled, and it will give a instability in output voltage in term of error signal, however appropriately designed PID controller hybrid electric vehicle can be operated with improved torque and power. It also shows clearly that the user interface provides better connectivity between the Arduino parameters and the PID controller for optimal and stable speed and operational protection. The results of the implementation and testing showed that the proposed prototype and controller is efficient and accurate for HEVs. All results have clearly shown that by using such technologies as presented in this paper will besupportive to automotive industry towards hybrid electric vehicles.

V. REFERENCES

[1] F. Badin, J. Scordia, and R. Trigui, *"Hybrid electric vehicles energy consumption decrease according to drive train architecture, energy management and vehicle use,"* IET Hybrid Veh. Conf. pp. 213-223, 2006.

[2] M. Hussain, A. R. Mustafa, M. A. Chaudhary, and A. Razaq, *"Design and implementation of hybrid vehicle using control of DC electric motor,"* 54th International Universities Power Engineering Conference (UPEC). *2019.*

[3] G.Pipeleers, B. Demeulenaere, and J. Swevers,*"Optimal linear controller design for periodic inputs,"* International Journal of Control. pp. 1044 – 1053, 2010.

[4] B. C. Francisco, F. C. Antonio, V. G. Antonio, and R. C. J. Cesar, *"Output feedback control for robust tracking of position trajectories for DC electric motors,"* Power Syst. Res. pp.*183–189, 2014.*

[5] F. A. Sulaiman, Y. P. Panos, and A. G. Ulsoy, *"Combined robust design and robust control of an electric DC motor,"* IEEE/ASME Transactions mechatronics. pp. 574-582, 2011.

[6] C. C. Chan, and K. T. Chau, *"An overview of power electronics in electric vehicles,"*IEEE Trans. on Industrial Electronics. vol. 44, pp. 3-13, 1997.

[7] Z. Q. Zhu, and D. Howe, *"Electrical machines and drives for electric, hybrid, and fuel cell vehicles*," IEEE Proceedings, vol. 95, pp. 746 – 765, 2007.

[8] B. Z. Guo, an D. L. Zaho, *"On convergence of the nonlinear active disturbance rejection control for MIMO systems,"*SIAM Journal on Control and Optimization. pp. 1727-1757, 2013.

[9] P. Pillay, an Dr. Krishnan, *"Modeling, simulation, and analysis of permanent-magnet motor drives, Part I: The permanent-magnet synchronous motor drive,"* IEEE Trans. on Industry Applications, vol. 25, pp. 265-273,1989.

[10] P. Pillay, and R. Krishnan, *"Modeling simulation, and analysis of permanent-magnet motor drives, Part 11: The*

brushless DC motor drive," IEEE Trans. on Industry Applications, vol. 25, pp. 274-279, 1989.

[11] R. Shanmugasundram, M. Zakariah, and N. Yadainah, *"Implementation and performance analysis of digital controllers for brushless DC motor drives,"*IEEE/ASME Trans. on Mechatronics, vol. 19, pp. 213-224, 2014.

[12] G. Brahim, N. Abdelfatah, and A. Othmane, *"An Efficiency PI Speed Controller for Future Electric Vehicle in Several Topology,"*Procedia Technology,pp.501-508, 2016.

[13] A. K Yadav, P. Gaur, S. K. Jha, J. R. P. Gupta, and A. P. Mittal, *"Optimal speed control of hybrid electric vehicles,"* Journal of Power Electronics,vol. 11, pp. 393-400, 2011.

[14] A. K. Yadav, and P. Gaur, *"An optimized and improved STF-PID speed control of throttle controlled HEV,"* The Arabian Journal for Science and Engineering, vol. 41, pp. 3749-3760, 2016.

[15] K. C. Prajapati, R. R. Patel, and R. Sagar, *"Hybrid vehicle: A study on technology,"* International Journal of Engineering Research & Technology," vol. 3, pp. 1076-1082, 2014.

[16] H. C. Chih, *"Adaptive Fuzzy Control Strategy for a Single-Wheel Transportation Vehicle*," IEEE Access. pp. 3272- 3283, 2019.

[17] T. Daisuke, A. Takashi, and M. Shigeyuki, *"An analytical method of EV velocity profile determination from the power consumption of electric vehicles,"* September 3-5, 2008.

[18] Manga Sravani, K.Padma Priya *"Antiwindup Design for Fuzzy PID Controlled DC Motor under Nonlinearities and Load Variations"* International Journal of Engineering Trends and Technology 61.2 (2018)