ORIENTATION PROJECT REPORT

Prepared for: Solarillion Foundation

Project Title: Closed loop control of DC Motor using Arduino

Prepared by: Aswin Natesh

23rd November 2015

Executive Summary

Objective

To Design a Closed loop speed control system for a 12 volt DC motor, along with a Speed sensor designed to measure the instantaneous speed in RPM, and to be sent as input to a Microcontroller. Based on the difference in sensed and reference speeds, An Algorithm is to be developed and implemented on a prototype to minimize the response time to achieve the speed.

Introduction

The efficiency of a motor in real time is anywhere between 50% - 70%. This Efficiency could be improvised by stabilizing the speed depending on load conditions. This usually done by three methods namely terminal voltage control, armature rheostat control method and flux control method.

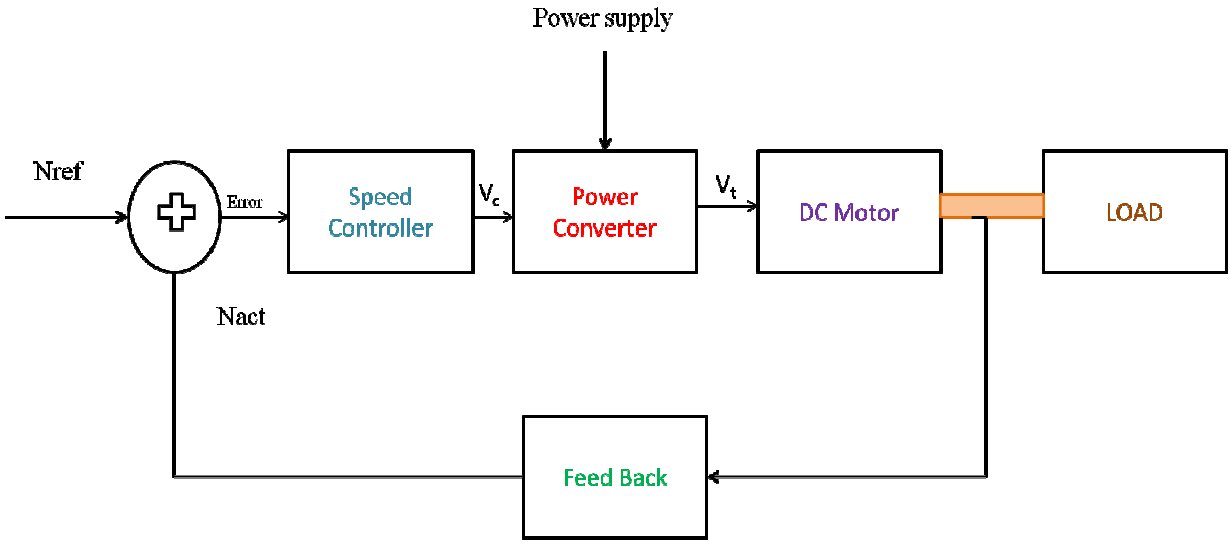
Here in this project terminal voltage control method is employed, to maintain a control over speed.

The Speed of the Motor is controlled by a Arduino module, powered by Atmega 328 Microcontroller, which is programmed with a PID Algorithm to provide the desired system response. The PID Algorithm would be further discussed in detail.

Strategies Adopted

- A simple algorithm to improvise the response rate of the motor, with minimum software lag is the primary strategy.
- Low Cost IR Sensor Modules are used to detect and measure time taken for each revolution made by the rotor.
- Arduino platform for the processing of feedback samples, and terminal voltage control of the transistor circuit, which in turn drives the motor.

The Closed Loop System



Hardware Components:

Motor 1 Specification | 500 Rpm

This is the first motor that is being tested with system developed. This motor is specifically designed for robotic applications, and comes with a plastic cased gearbox. These gearboxes convert the transmission speed and torque required. These gearboxes are readily available with huge range of RPMs.

Specifications:

Shaft speed	: 500 Rpm
Shaft Diameter `	: 6mm
Torque	: 2Kgcm
Voltage	: 12V DC
No Load Current	: 60 mA (Max)
Loaded current	: 800 mA (Max)

Motor Specification | 1000 Rpm

This is the second motor that is tested with system develop. This belongs to the family of high torque industrial motor manufactured by Johnson Electric, and used in control applications. This comes attached with a metal cased gearbox.

Specifications:

Shaft speed	: 1000 Rpm
Shaft Diameter	: 6mm
Torque	: 10Kgcm
Voltage	: 12V DC
No Load Current	: 800 mA (Max)
Loaded current	: 5 A (Max)

Power Supply Unit (SMPS)

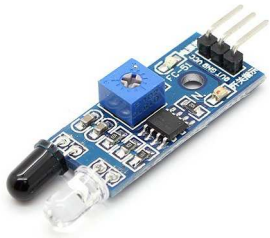
This is an AC to DC power converter rated at 25W. The switched-mode power supply (SMPS) incorporates a switching regulator to convert electrical power efficiently. As in this project the, actual characteristics of motor is being studied, the source voltages must match values required by motors and microcontrollers, thus a SMPS does this most effectively.

Specifications:

Input AC Voltage	: 110V – 230V 50Hz
Output DC Voltage	: 12V, 2 Amps

Feedback | IR Modules

This module consist of a IR Transmitter and Receiver, which transmits continuous IR Rays, and once a reflecting medium is brought within range, the IR beam gets reflected and sensed by the Receiver, which in turn sends a High pulse to the controller. This module is powered by a 5V Source. The difference in time between pulses is computed by the microcontroller. This time when divided by 60 seconds, we get the RPM of the motor.



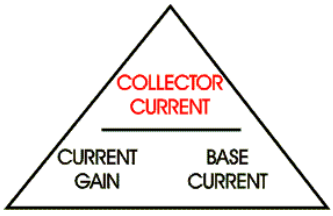
$$RPM = \frac{(60 \times 1000)}{Time\ Interval}$$

Power Converter | Transistor (TIP 122)

Transistors are used to amplify current. The leads are divided into BASE, COLLECTOR and EMITTER or GAIN. The GAIN is simply the amount of amplification.

Selection Criteria

- Base current of Transistor should be below 30mA, as Arduino absolute maximum rating is 40mA.
- Should drive high current DC Load of 5 Amps.



A solution for this is TIP 122, which is a NPN power Darlington Transistor used as a switching device.

Characteristics:

Collector Base Voltage (V_{CBO})	= 100V
Collector Emitter Voltage (V_{CEO})	= 100V
Emitter Base Voltage (V_{EBO})	= 5V
Collector Current (I_C)	= 10 A

$$\text{BASE CURRENT} = \frac{\text{COLLECTOR CURRENT}}{\text{CURRENT GAIN}} = 5mA$$

The base of the transistor (PIN 1) is connected to the Arduino output pin (D6) through a 1K Ohm Resistor. The emitter (PIN3) is connected to ground and the collector (PIN2) is connected to one end of the coil of the device being driven. The other end of the coil is connected to the +12VDC external power supply (the ground from this power supply is connected to a common ground with the Arduino - this is necessary for the transistor to function).

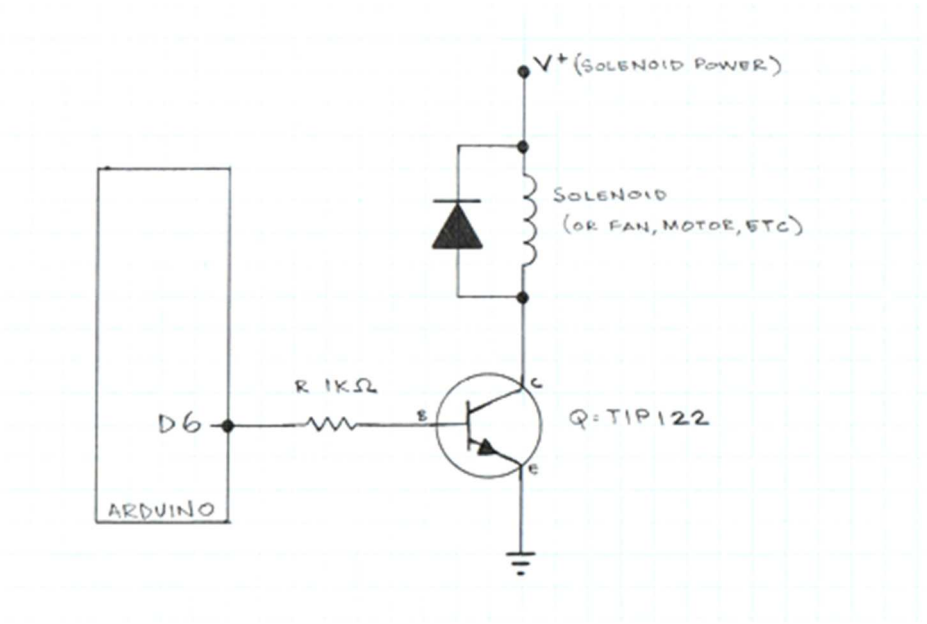
Controller | Arduino Uno

The Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller.

Technical Specifications:

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Circuitry:



Algorithm:

Step 1. Start

Step 2. Fetch Set point from User through Serial Monitor

Step 3. Compute PWM Value through PID controller

Step 4. Pass data to drive the motor

Step 5. Measure the rotor speed

Step 6. Compute error, and redefine PWM through PID controller

Step 7. Pass redefined value to the motor

Step 8. Loop Statements 3 -7 till required speed is obtained

What is PID?

From Wikipedia: "A PID controller calculates an 'error' value as the difference between a measured [Input] and a desired set point. The controller attempts to minimize the error by adjusting [an Output]."

There are 3 Tuning Parameters (or "Tunings"): Kp, Ki & Kd. Adjusting these values will change the way the output is adjusted. Fast? Slow? God-awful? All of these can be achieved depending on the values of Kp, Ki, and Kd.

$$\text{Output} = K_P e(t) + K_I \int e(t)\,dt + K_D \frac{d}{dt} e(t)$$

$$\text{Where : } e = \text{Setpoint - Input}$$

Not all of the terms in the PID equation are necessarily used. Here it is just the PD controller is used, and not PID.

Tuning Methodology

The Kp Kd values in this project is tuned manually because auto-tuning function would not suit this case as when trade-off between least deviation, and time is concerned. Auto tuning might fetch accurate results in a longer span of time. With practice, the tuning is brought closer to the ideal value.

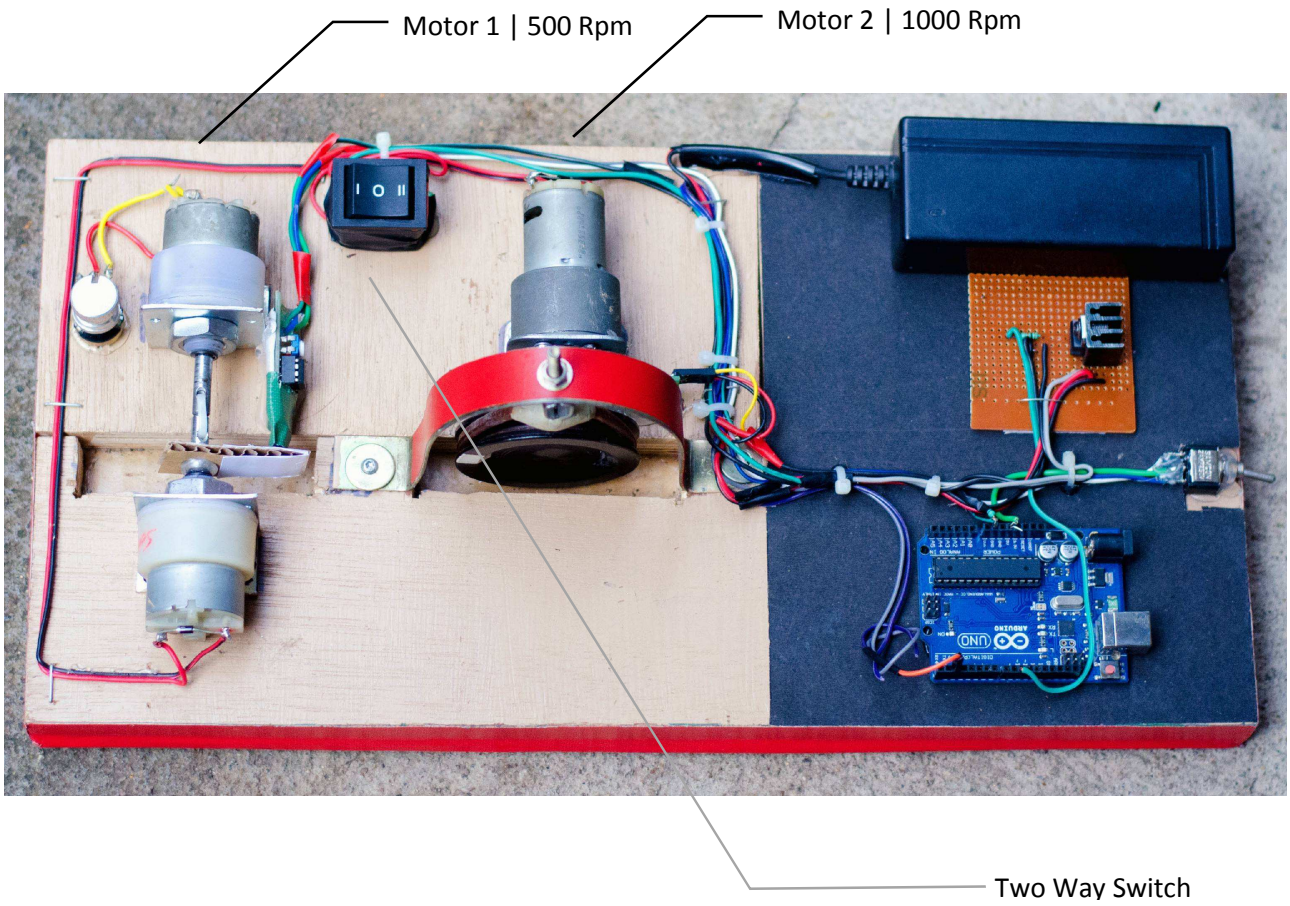
A simple empirical approach is used here starting by zeroing the integral and derivative gains, and just using the proportional term. Setting the proportional gain increasingly higher will finally cause the system to oscillate. This is bad behavior. Now by proportional gain is reduced till below the point of incipient oscillation is obtained. Right after this the derivative gain is slowly increased in steps which should act to forestall the start of oscillatory behavior.

Program Coding | Self Explanatory

```
                                                        //VARIABLE DECLATATION
#define LOOPTIME  100                                    // Main loop execution time
unsigned long PID_Delay = 0;                            // PID Delay time
unsigned long CycleTime;                                // Revolution time
unsigned long WhiteArea;
unsigned long BlackArea;
int User_RPM = 0, PWM_val = 0;
int Error, Value;
float RPM;

float Kp = 0.42;                                         // PID Proportional control Gain
float Kd = 0.1525;                                      // PID Derivative control Gain
int IR_pin = 2;                                         // IR Read Sensor Pin
int PWM1 = 10;                                          // Motor PWM Control Pin
                                                                  // INITIAL SETUP

void setup()

  {
    Serial.begin(9600);
    pinMode(IR_pin, INPUT);
    analogWrite(PWM1, PWM_val);
    digitalWrite(IR_pin, HIGH);
  }
                                                              // FUNCTION DECLARATIONS
void Measure()                                          // Revolution Time
  {
     float Old_RPM;
     WhiteArea = pulseIn(IR_pin, LOW);
     BlackArea = pulseIn(IR_pin, HIGH);
     CycleTime = WhiteArea + BlackArea;                 // Time taken per Revolution
     RPM = ((60*1000000)/CycleTime);                    // RPM Calculation
   }

int UpdatePid(int command, int targetValue, int currentValue)    // PID correction
  {
    float Pid_Term = 0;
```

```
    static int last_Error=0;
    Error = abs(targetValue) - abs(currentValue);
    Pid_Term = (Kp * Error) + (Kd * (Error - last_Error));
    last_Error = Error;
    return constrain(command + int(Pid_Term), 0, 255);          // Mapping to PWM Equivalent
  }
void Print_RPM()
{
      Serial.print("\t USER RPM: ");  Serial.print(User_RPM);
      Serial.print("\t RPM: ");       Serial.print(RPM);
      Serial.print("\t ERROR: ");     Serial.print(Error);
      if(User_RPM == RPM || User_RPM == (RPM + 1)|| User_RPM == (RPM -1))
       { Serial.println("\t Speed Acheived ! "); }
      else
      {  Serial.println(""); }
   }
                                                                                  // MAIN LOOP
void loop()
 {
   if (Serial.available() >0 )
     { User_RPM = Serial.parseInt(); }                          // Read Required RPM
   Measure();                                                   // Measure Current RPM
   if((millis()-PID_Delay) >= LOOPTIME)                         // PID Steady State Time delay
     {
       PID_Delay = millis();
       PWM_val= UpdatePid(PWM_val, User_RPM, RPM);              // Calculate PWM
       analogWrite(PWM1, PWM_val);
     }
   Print_RPM();                                                 // Print Live Values

}
```

Handling Instructions:



Step 1. Connect the power cord between ac mains and the SMPS

Step 2. Establish a USB Serial communication with the Computer

Step 3. Select the motor to be tested using the Two way switch

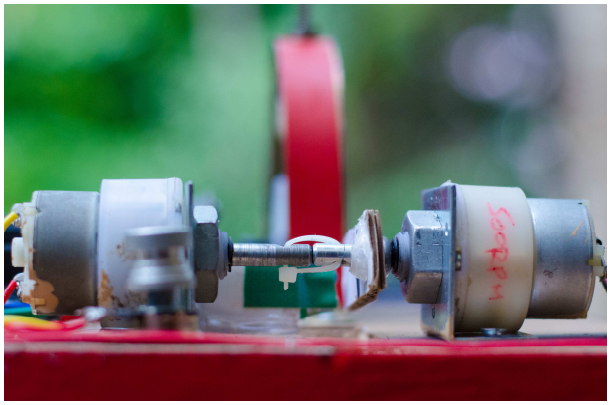
Step 4. Apply necessary load to the motors independently

Step 5. Monitor speed through the on screen data displayed

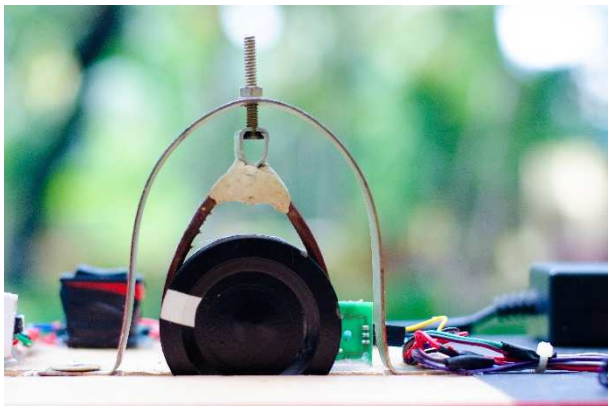
Applying Loads:

Motor 1 | 500 RPM

Apply required load to the motor by adjusting the potentiometer knob placed beside the driving motor. Release loads by zeroing the potentiometer.



Motor 2 | 1000 RPM

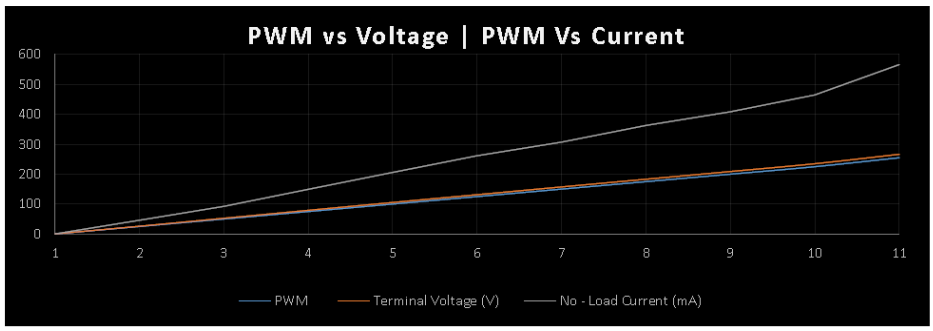


Apply required load to the brake drum by fastening the nut and bolt placed over the clamp setup. Release load by loosening the nut.

Results & Graphs

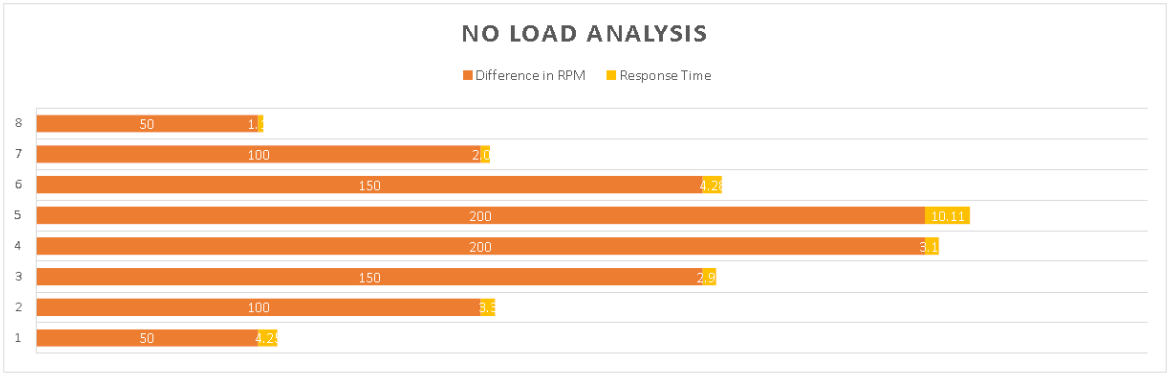
PWM Analysis

PWM	Terminal Voltage (V)	No - Load Current (mA)
0	0.05	0
25	0.83	20
50	2.51	40
75	4.09	70
100	5.42	100
125	6.46	130
150	7.38	150
175	8.19	180
200	8.99	200
225	10.01	230
255	11.5	300



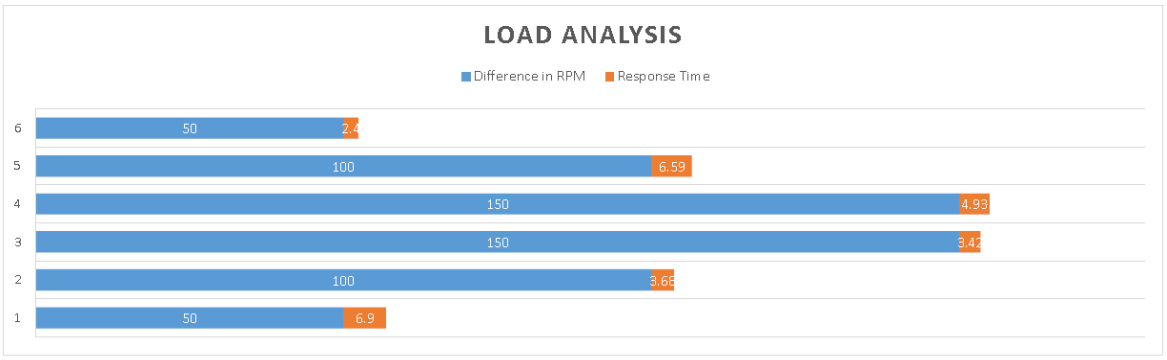
No Load Analysis

S.no	Current Rpm	Required Rpm	Difference in RPM	Achieved Rpm	Response Time	Rising or Falling
1	300	350	50	349	4.25	Rising
2	300	400	100	399	3.3	
3	300	450	150	448	2.98	
4	300	500	200	502	3.13	
5	500	300	200	301	10.11	Falling
6	500	350	150	350	4.28	
7	500	400	100	399	2.09	
8	500	450	50	452	1.13	
Average Response Time					3.91	



Load Analysis

S.no	Current Rpm	Required Rpm	Difference in RPM	Achieved Rpm	Response Time	Rising or Falling
1	300	350	50	350	6.9	Rising
2	300	400	100	399	3.68	
3	300	450	150	448	3.42	
4	450	300	150	299	4.93	Falling
5	450	350	100	349	6.59	
6	450	400	50	399	2.4	
Average Response Time					4.65	



How did it all happen?

Now coming across the learnings from this project, initially, being too concerned about the hardware setup of the project, and power supplies, I went on sourcing branded motors. At this phase of the project, it was a mistake choosing a high torque branded motor just because lower torque motors weren' t available in the stores. A true factor that branded motors are far linear that non branded china motors, they did also consume higher currents. Continuing further, no load operations did work smoothly, whereas applying load to this motor was a challenge.

Being a high torque motor, applying hand loads was extremely difficult and non-uniform. To resolve this issue, I further went on attaching a brake drum to the rotor for applying pulley loads. The next added complication, source for the motor. On applying loads, the motor draws excessive current from the power supply unit, because of which the voltage drops, leading to insufficient terminal voltage. This made me to source a higher power supply unit, and on powering the motor, the transistor behavior was abnormal. This was a deadlock situation, and to overcome this, transistor driver circuits are to be designed which does not fall under the project scope, and since the deadline period for this project has also crossed, I did jump to a lower rating motor.

When this motor was replaced with a non-branded robotic motor with lower speed, the entire system did happen to work perfectly. With a very slight tuning of PID parameters, the system response did improve and the scale of error was around +/- 1. The maximum no load time to achieve this was 3 Seconds, and on loaded condition 6 Seconds.

Since this being the final result required, I would like to consider this project to be completed and would request you to accept this report.

- Aswin Natesh

Acknowledgement:

I Thank Mr. Vineeth Vijayaraghavan, for giving me this opportunity and motivation to gain knowledge through these type of assignments & projects. He was extremely considerate and I did cherish ever moment conversing with him.

I Thank Mr.Raja Shekar for having explained the entire project scope, and he was very eager to listen to my questions and answered them patiently.

I Thank Mr.Vikram Vel, for helping me on the selection of motors and sensors for this project. He truly impressed me through his coding and debugging skills on Arduino Programming.

Finally, I am thankful to all my friends who have directly and indirectly advised me on addressing the intermediate problems, fine tuning of parameters, and preparing the report.

Conclusion:

A Generic code has been developed which uses PD Algorithm to control the Speed of a DC motor at an average response time of **3.91 Seconds** during No Load operation and **4.65 Seconds** during Loaded Operation.

The Accuracy of the motor speed is observed to be (+/-) 1 Rpm