# Final Seminar Report

# DC servo motor speed control using PID

**By**


**Sidharth Sahdev** **2011AAPS098H**
**Rahul Dubey** **2011A3PS338H**


**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI**
**(November, 2013)**

# BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
# HYDERABAD CAMPUS


**Students:** **Rahul Dubey, B.E. Electrical & Electronics**
**Sidharth Sahdev,B.E. Electronics & Communications**

**Expert:** **Assoc. Prof. Alivelu Manga Parimi**

**Guiding Faculty:**


**Duration:** 15th August, 2013 to 21st November, 2013

**Date of Submission**: 25th November, 2013

**Title of the Project**: DC Servo motor control using PID controller

**Course** Lab Oriented project


**Key Words:** Servo, PID, Arduino, L293D, H-bridge, Simulink, feedback

**Project Area :** Control System and Design


AT

**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI**

# Table of Contents


ACKNOWLEDGEMENTS ..... 4

Abstract ..... 5

INTRODUCTION ..... 6

Servo Motor ..... 6

PID Control Systems ..... 7

Model used ..... 9

Without controller: (open loop) ..... 9

With controller ( Closed loop) ..... 11

ARDUINO ..... 12

Technical specifications: ..... 12

Power: ..... 13

Input and Output ..... 14

How to use Arduino: ..... 15

Motor Driver -L293D IC ..... 15

H-Bridge ..... 17

General: ..... 17

Operation: ..... 18

Work-flow ..... 20

DC Motor control with GUI ..... 21

DC small servo-Motor control GUI with PID controller ..... 22

Servo motor position control: ..... 23

Servo motor speed control: ..... 24

DC big servo motor: ..... 26

Interfacing with Simulink ..... 26

Runnig the simulink block diagram: ..... 29

Problems faced: ..... 32

INFERENCE/CONCLUSION ..... 35

References: ..... 36

Appendix: ..... 37

# ACKNOWLEDGEMENTS

Firstly, we are grateful to **Department of Electrical Engineering, BITS-Pilani Hyderabad Campus** for offering us Lab Oriented Project (LOP) as a course in our curriculum.

We would like to express our sincere thanks to Assoc P**rof. Alivelu Manga Parimi**, Department Of Electrical Engineering, BITS –Pilani Hyderabad Campus for giving us the opportunity to carry out a project in this esteemed organization.

We would also like to thank ma'am, our mentor and our Project Guide for suggesting us the project and providing us valuable guidance and support throughout our work.

We would like to extend our gratitude to our friends Kashyap, Alind, Suchita and Aalhad who helped us out in every possible manner and supported us from time to time.

# Abstract

We developed the controller model for a DC Servo Motor. We start with simulating the open loop model in Matlab for which we referred a few papers. Then, we designed the controller that is the closed loop representation of the model in an effort to minimize the error signal. After this target was successfully achieved, we interfaced the Arduino with Matlab. To begin with, we made a GUI (Graphical user interface) to correctly control the speed of a small DC Motor and now currently doing the same for a DC Servo. Next we show how exactly to interface arduino with Simulink.

# INTRODUCTION

## Servo Motor

A **servomotor** is a rotary actuator that allows for precise control of angular position, velocity and acceleration. It consists of a suitable motor coupled to a sensor for position feedback. It also requires a relatively sophisticated controller, often a dedicated module designed specifically for use with servomotors.
Servomotors are not a different class of motor, on the basis of fundamental operating principle, but uses servomechanism to achieve closed loop control with a generic open loop motor.
Servo motors are used for angular positioning, such as in radio control airplanes. They typically have a movement range of 180 deg but can go up to 210 deg.
The output shaft of a servo does not rotate freely, but rather is made to seek a particular angular position under electronic control.
They are typically rated by torque and speed.

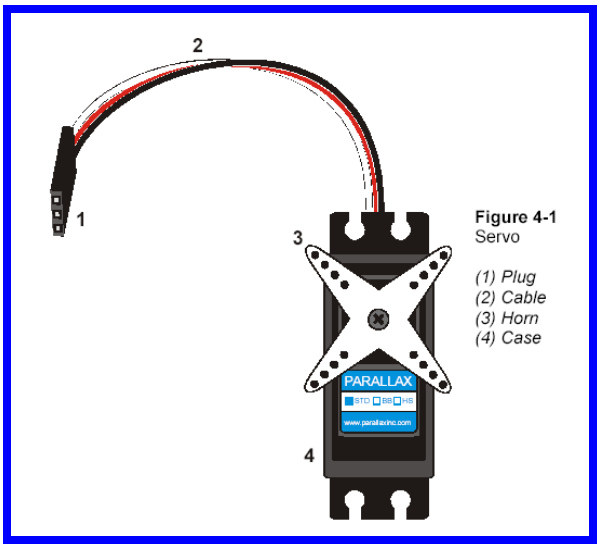
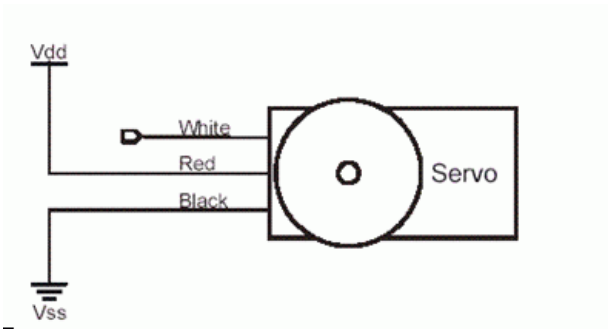



**Figure 4-1**
Servo

*(1) Plug*
*(2) Cable*
*(3) Horn*
*(4) Case*



# PID Control Systems

The Proportional Integral Derivative (PID) control function shown in Figure 8.6 is the most popular choice in industry. In the equation given the 'e' is the system error, and there are three separate gain constants for the three terms. The result is a control variable value.

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

Block diagram

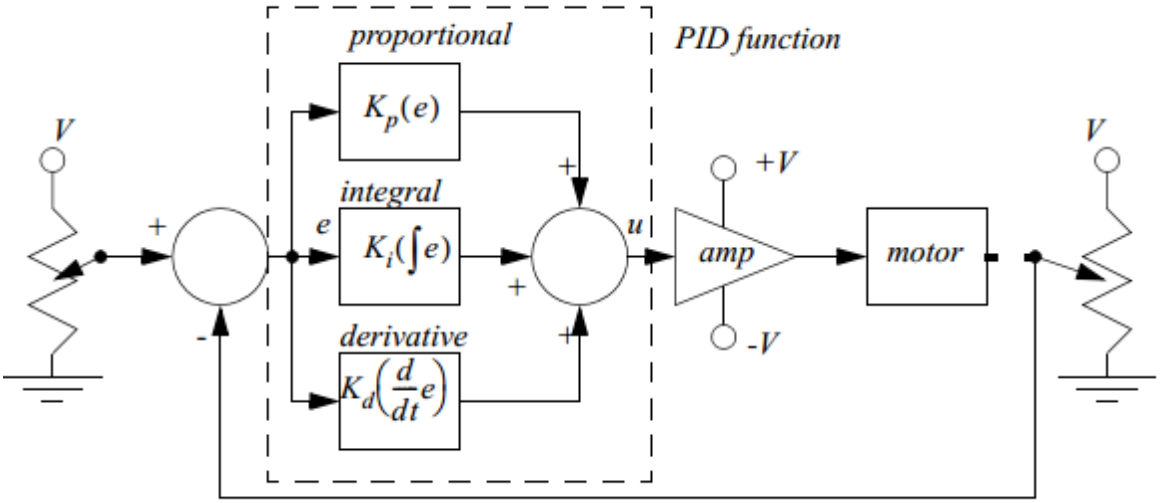



Figure shows a basic PID controller in block diagram form. In this case the potentiometer on the left is used as a voltage divider, providing a setpoint voltage. At the output the motor shaft drives a potentiometer, also used as a voltage divider. The voltages from the setpoint and output are subtracted at the summation block to calculate the feedback error. The resulting error is used in the PID function. In the proportional branch the error is multiplied by a constant, to provide a longterm output for the motor (a ballpark guess). If an error is largely positive or negative for a while the integral branch value will become large and push the system towards zero. When there is a sudden change occurs in the error value the differential branch will give a quick response. The results of all three branches are added together in the second summation block. This result is then amplified to drive the motor. The overall performance of the system can be changed by adjusting the gains in the three branches of the PID function.

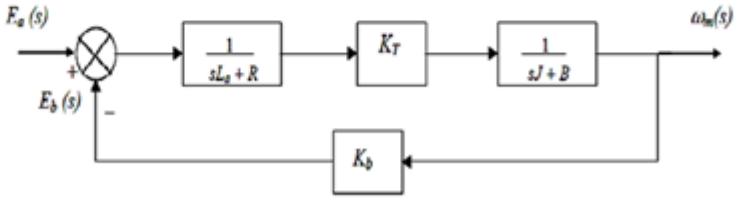
*Tip*: *The manual process for tuning a PID controlled is to set all gains to zero. The proportional gain is then adjusted until the system is responding to input changes without excessive overshoot. After that the integral gain is increased until the longterm errors disappear. The differential gain will be increased last to make the system respond faster.*

Standard Controller types:

| *Type* | *Transfer Function* |
|---|---|
| *Proportional (P)* | $G_c = K$ |
| *Proportional-Integral (PI)* | $G_c = K\left(1 + \frac{1}{\tau D}\right)$ |
| *Proportional-Derivative (PD)* | $G_c = K(1 + \tau D)$ |
| *Proportional-Integral-Derivative (PID)* | $G_c = K\left(1 + \frac{1}{\tau D} + \tau D\right)$ |
| *Lead* | $G_c = K\left(\frac{1 + \alpha\tau D}{1 + \tau D}\right) \quad \alpha > 1$ |
| *Lag* | $G_c = K\left(\frac{1 + \tau D}{1 + \alpha\tau D}\right) \quad \alpha > 1$ |
| *Lead-Lag* | $G_c = K\left[\left(\frac{1 + \tau_1 D}{1 + \alpha\tau_1 D}\right)\left(\frac{1 + \alpha\tau_2 D}{1 + \tau_2 D}\right)\right] \quad \alpha > 1,\ \tau_1 > \tau_2$ |

# EQUATIONS USED

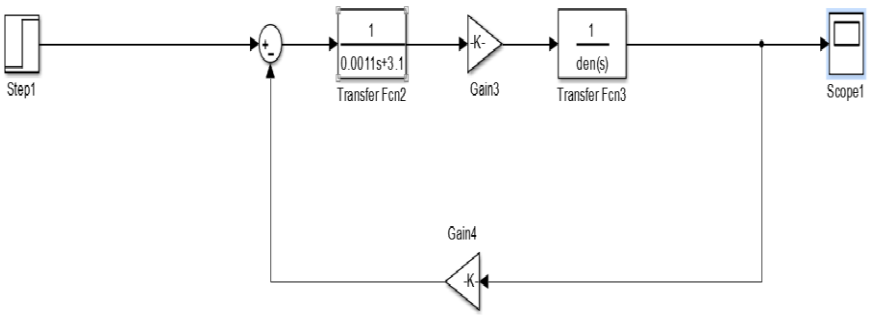
- $E^a(s) = R^a.I^a(s) + sL^a.I^a(s) + E^b(s)$
- $T^m(s) = sJ.\omega^m(s) + B.\omega^m(s)$
- $T^m(s) = K^T.I^a(s)$
- $E^b(s) = K^b.\ \omega^m(s)$

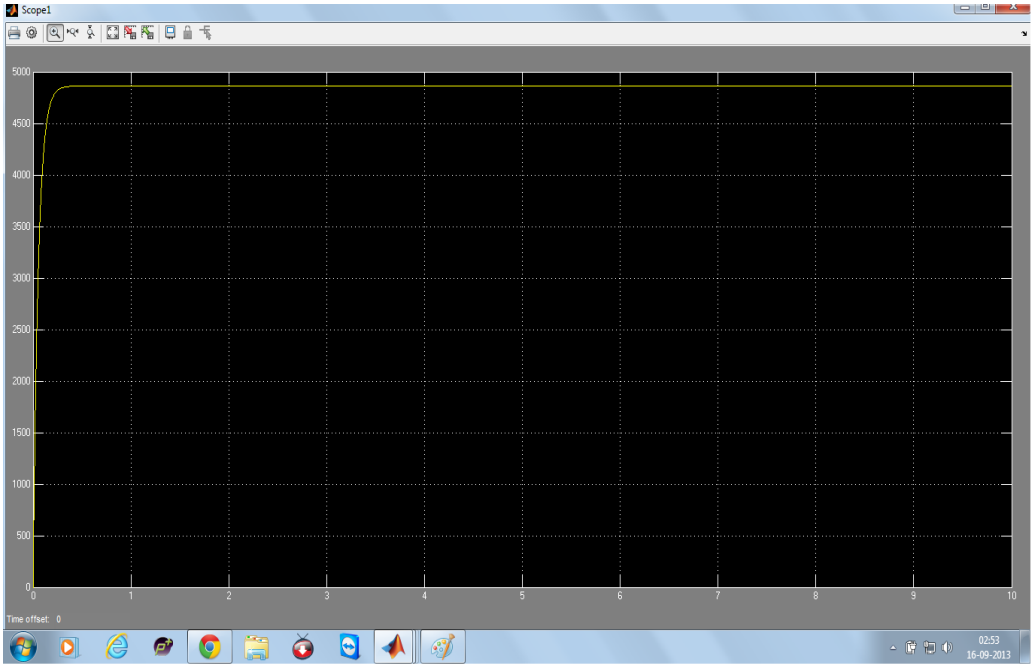



## Model used

### Without controller: (open loop)

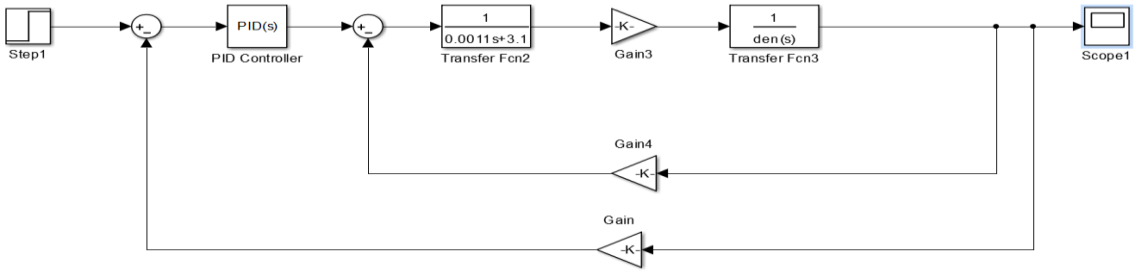
To begin with, we referred the IEEE papers to study about the working of the Dc Servo motors. Using the equations that govern the working of the dc Servo motor, we came up to the following model.

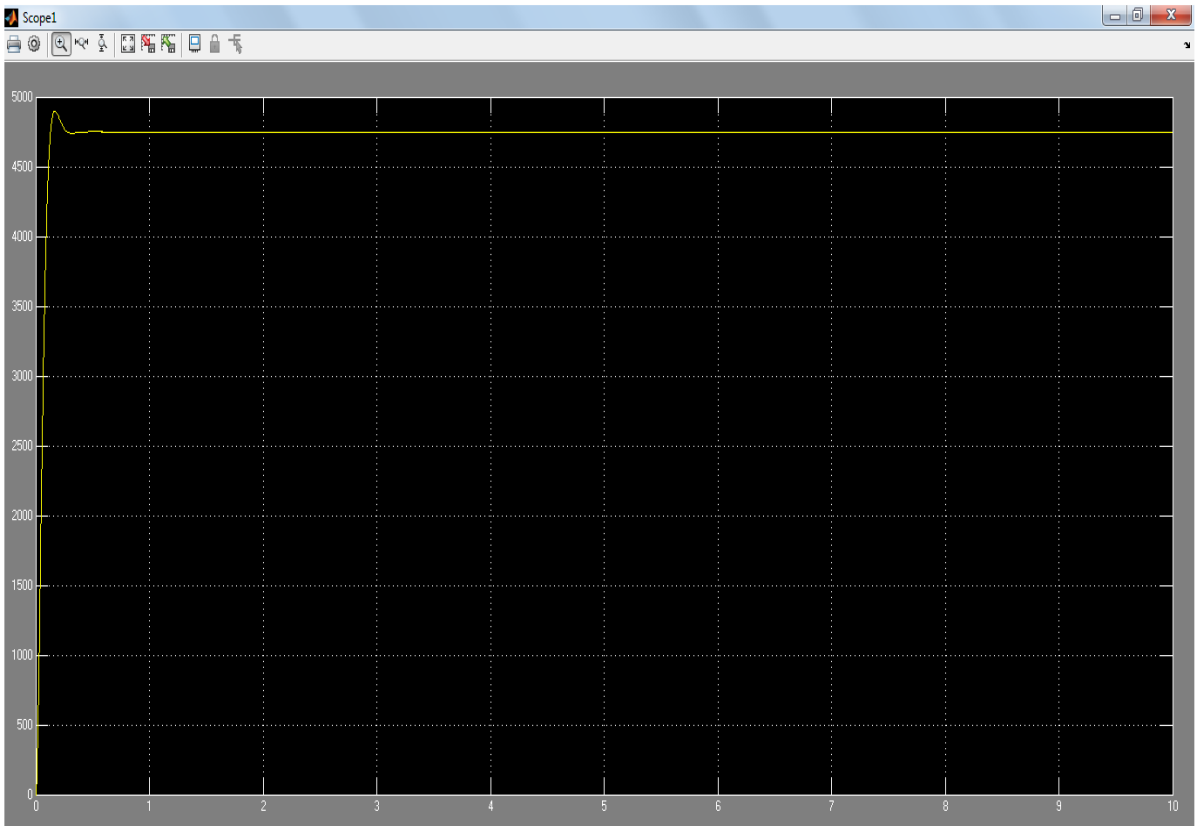



The motor was rated at 3750 rpm. Without the controller, we achieved a steady state value of 3800 rpm with a rise time of about 0.5 seconds.
Next, our target was to design the PID Controller to get an even accurate value with a reduced rise time.

## With controller ( Closed loop)




As we can observe, we got the steady state value of 3750 rpm with a rise time of about 0.2 seconds!!!
Now, we were left with the actual hardware implementation part. The components required are :-

1. Dc Servo
2. Arduino (the micro controller)
3. Motor Driver IC (to amplify the output current by the Arrduino)
4. Breadboard ( to make connections)
5. 9V/12V battery ( in case we do not wish to power through the laptop)
6. Other smaller components

Now, Arduino is described here, which is the main processing unit of the system ( after the laptop ofcourse)

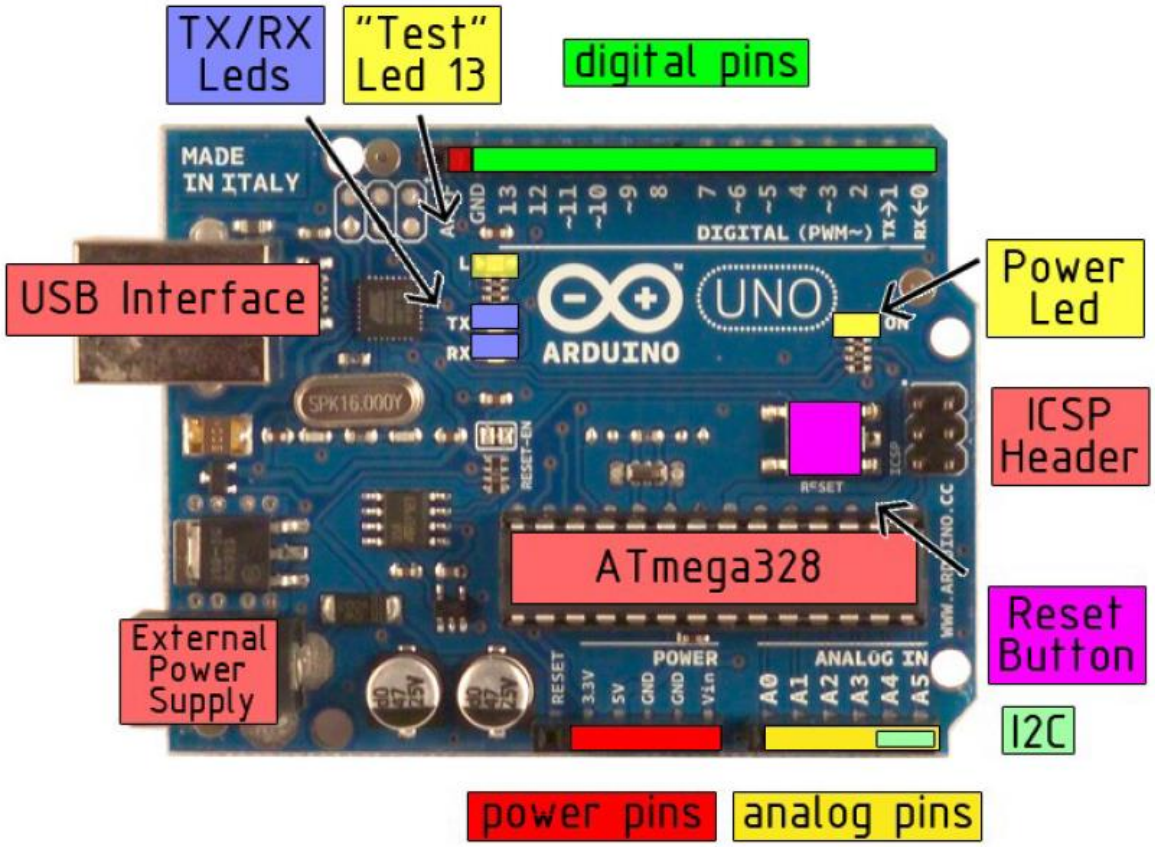
# ARDUINO

The Arduino Uno is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial converter.
"Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduno, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform

## Technical specifications:

Microcontroller ATmega328
Operating Voltage 5V
Input Voltage (recommended) 7-12V
Input Voltage (limits) 6-20V
Digital I/O Pins 14 (of which 6 provide PWM output)

Analog Input Pins 6
DC Current per I/O Pin 40 mA
DC Current for 3.3V Pin 50 mA
Flash Memory 32 KB of which 0.5 KB used by bootloader
SRAM 2 KB
EEPROM 1 KB
Clock Speed 16 MHz




## Power:

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.
The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

## Input and Output

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. TThese pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip .
- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Uno has 6 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:

- **I2C: 4 (SDA) and 5 (SCL).** Support I2C (TWI) communication using the Wire library.

There are a couple of other pins on the board:

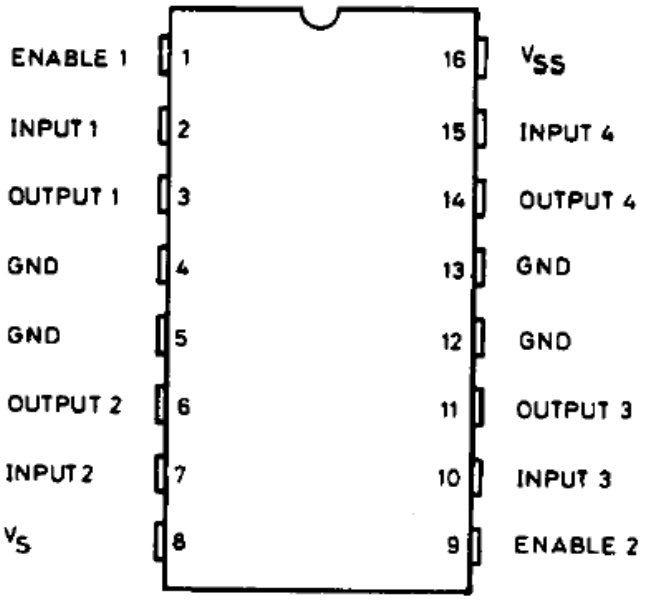
- **AREF.** Reference voltage for the analog inputs. Used with analogReference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

## How to use Arduino:

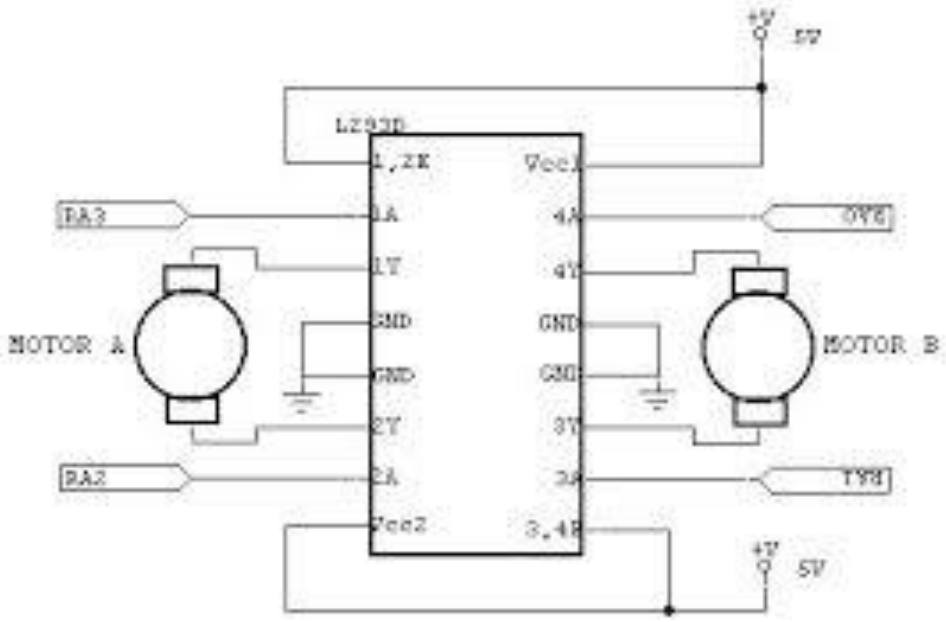
Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino projects can be stand-alone or they can communicate with software on running on a computer (e.g. Flash, Processing, MaxMSP).
Arduino is a cross-platform program. You'll have to follow different instructions for your personal
OS.

# Motor Driver -L293D IC

The current drawn from the output pins of the arduino is in mA which is insufficient to drive the motor. Here we need to basically amplify the current using a amplifier. The circuitry used is provided by the L293D IC that consists of Op-Amps and integrated H bridge logic to deliver the required current to drive a motor.




| Enable 1 | Input 1 | Input 2 | Function |
|---|---|---|---|
| High | Low | High | Clockwise |
| High | High | Low | Anti-clockwise |
| High | Low | Low | Stop |
| High | High | High | Stop |
| Low | N/A | N/A | Stop |



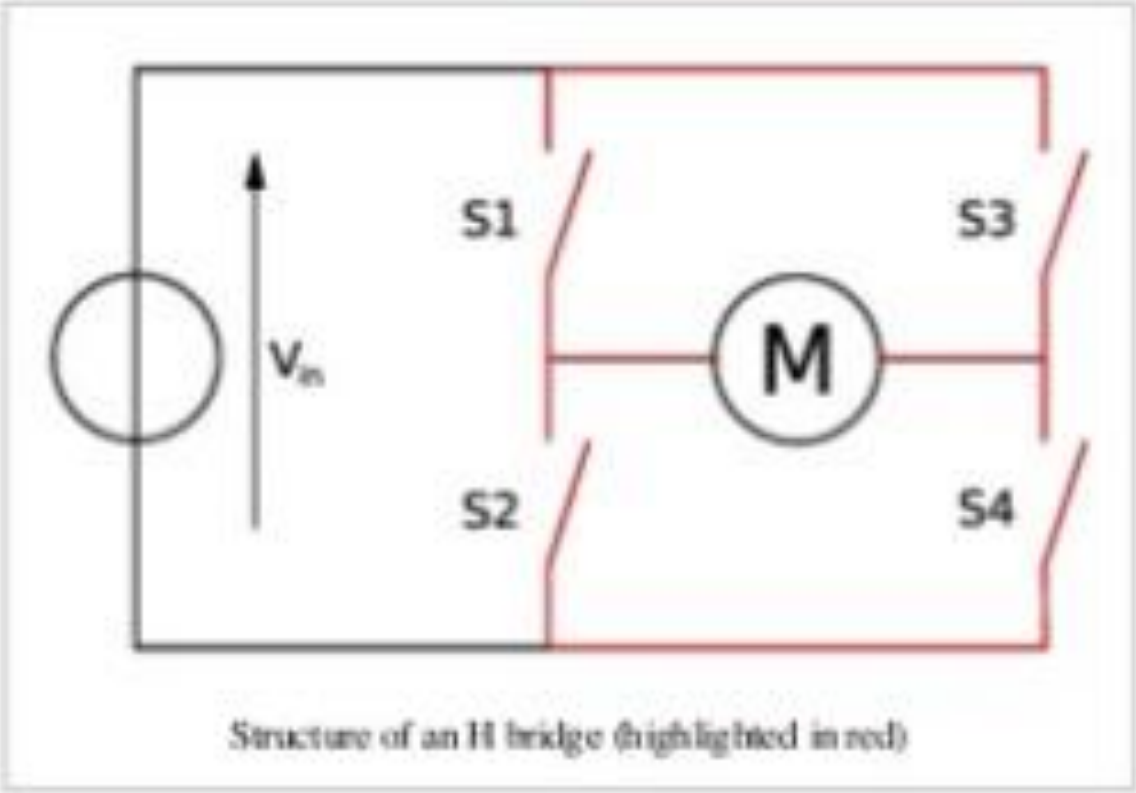
# H-Bridge

An H-bridge is an electronic circuit that enables a voltage to be applied across a load direction. These circuits are often used in robotics and other applications to allow DC motors to run forwards and backwards. H bridges are available as integrated circuits, or can be built from discrete components.

## General:

The term H Bridge is derived from the typical graphical representation of such a circuit. An H bridge is built with four switches (solid-state or mechanical). When the switches S1 and S4 are closed (see figure) and S2 and S3 open, a positive voltage will be applied across the motor. By opening S1 and S4 and closing S2 and S3, this voltage is reversed allowing reverse operation of the motor.

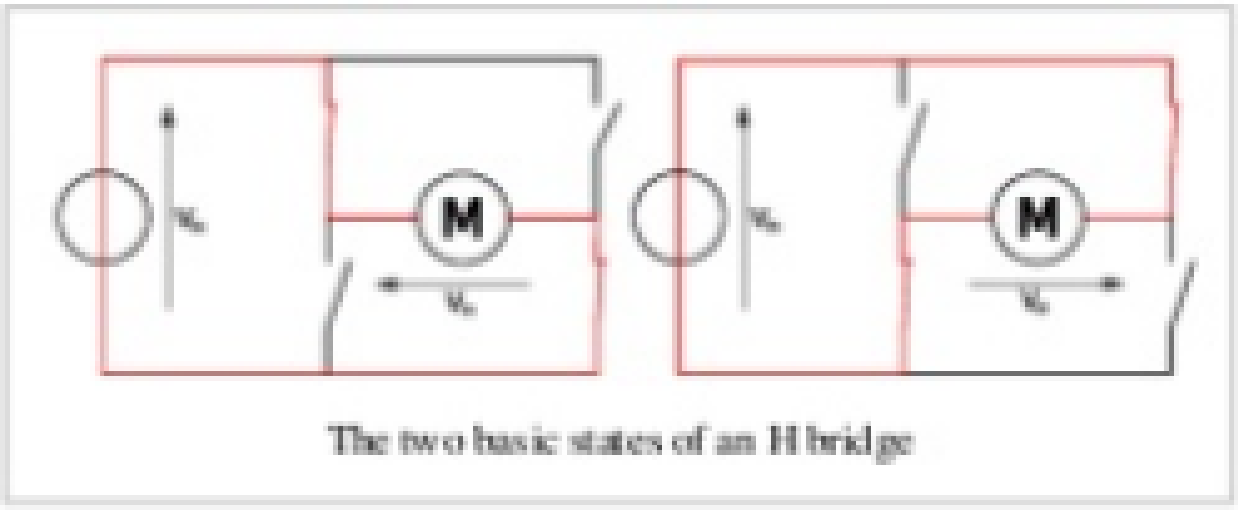
Using the nomenclature above, the switches S1 and S2 should never be closed at the same time, as this would cause a short circuit on the input voltage source. The same applies to the switches S3 and S4. This condition is known as shoot-through.



Structure of an H bridge (highlighted in red)

## Operation:

The H bridge arrangement is generally used to reverse the polarity of the motor, but can be used to brake the motor, where the motor comes to sudden stop, as the motor's terminals are shorted, or let the motor 'free run' to stop, as the motor is effectively disconnected from the circuit. The following table summarizes operation, with S1-S4 being the 4 switches.




The two basic states of an H bridge

| S1 | S2 | S3 | S4 | Result |
|---|---|---|---|---|
| 1 | 0 | 0 | 1 | Motor moves right |
| 0 | 1 | 1 | 0 | Motor moves left |
| 0 | 0 | 0 | 0 | Motor free runs |
| 0 | 1 | 0 | 1 | Motor brakes |
| 1 | 0 | 1 | 0 | Motor brakes |

# Work-flow

After implementing the controller model on Matlab, we worked on the Hardware part of it. We aim to interface the Simulink and Arduino. However, we faced issues in that.

Expectation :

In order to link the Simulink with Arduino, we need an Encoder Software. This software converts the Simulink code to C code which then has to be converted to the respective Matlab code. This code is then dumped onto the Arduino which hence controls the motor.

Reality :

Apparantly, the Encoder works well with only Arduino Uno and Arduino Atmega 2560. We had Arduino Duemilanove, which did not respond to this software. We researched for the possible alternatives so that we could get around the problem.

Here are few of the feasible options available that can be explored. :-

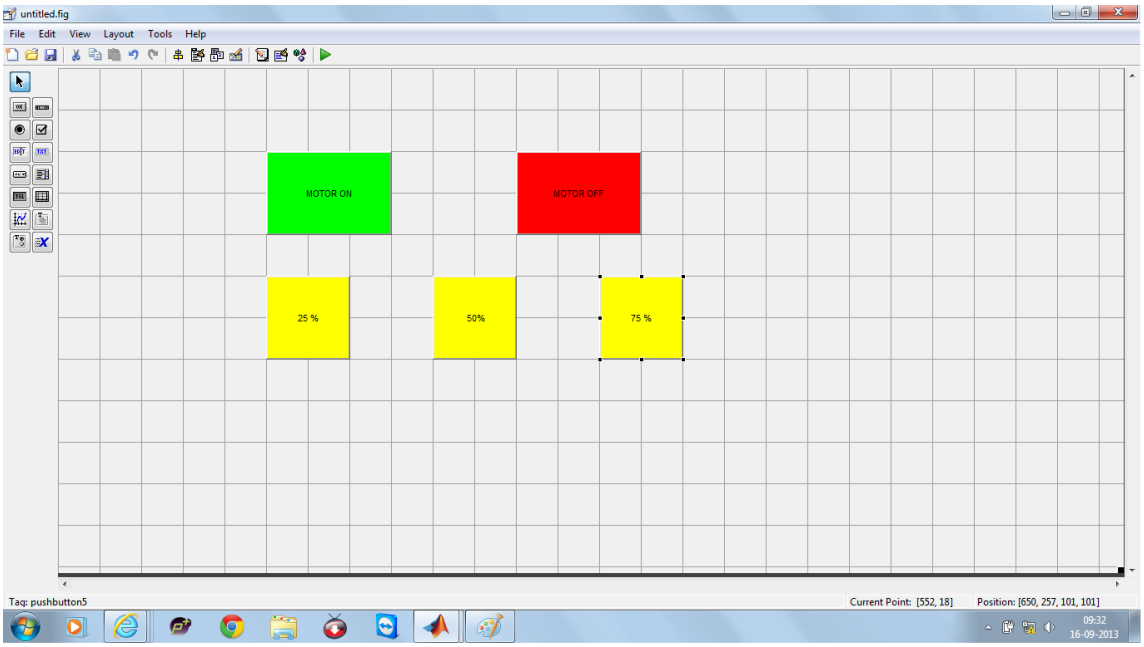
1. Purchase the required Arduino Board ( Uno or Atmega 2560 ). Arduino Uno costs about 1100/- and Atmega costs about 3600/-. Though costly, buying one would greatly simplify the effort as these boards are compatible with the Available software.

2. Use a PIC that costs about 300/-. It has the advantage of the cost factor. Also, this is a tried and tested method that would guarantee success. But, this will be a totally different approach to the problem as Arduino will have to part its way. The PIC uses the Ladder logic coding which is new to us and hence take time to seep in.

3. The last option is to code for the complete controller! This would not involve Simulink anywhere. We will do the coding and come up with a GUI (Graphical User Interface) similar to what we have already done for the Dc Motor. All that remains in this approach is to code for the PID controller and link it with Arduino.

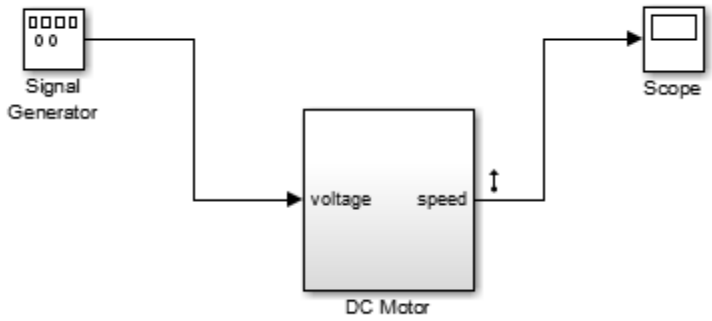
After discussion with others, we arrived at the decision of doing the coding for PID controller to check if we are getting the desired result. In case we are able to implement it successfully, the other groups will also code their respective models for uniformity. Else, we will buy the Arduino Uno and link it with Simulink using the Encoder software we already have.

## DC Motor control with GUI

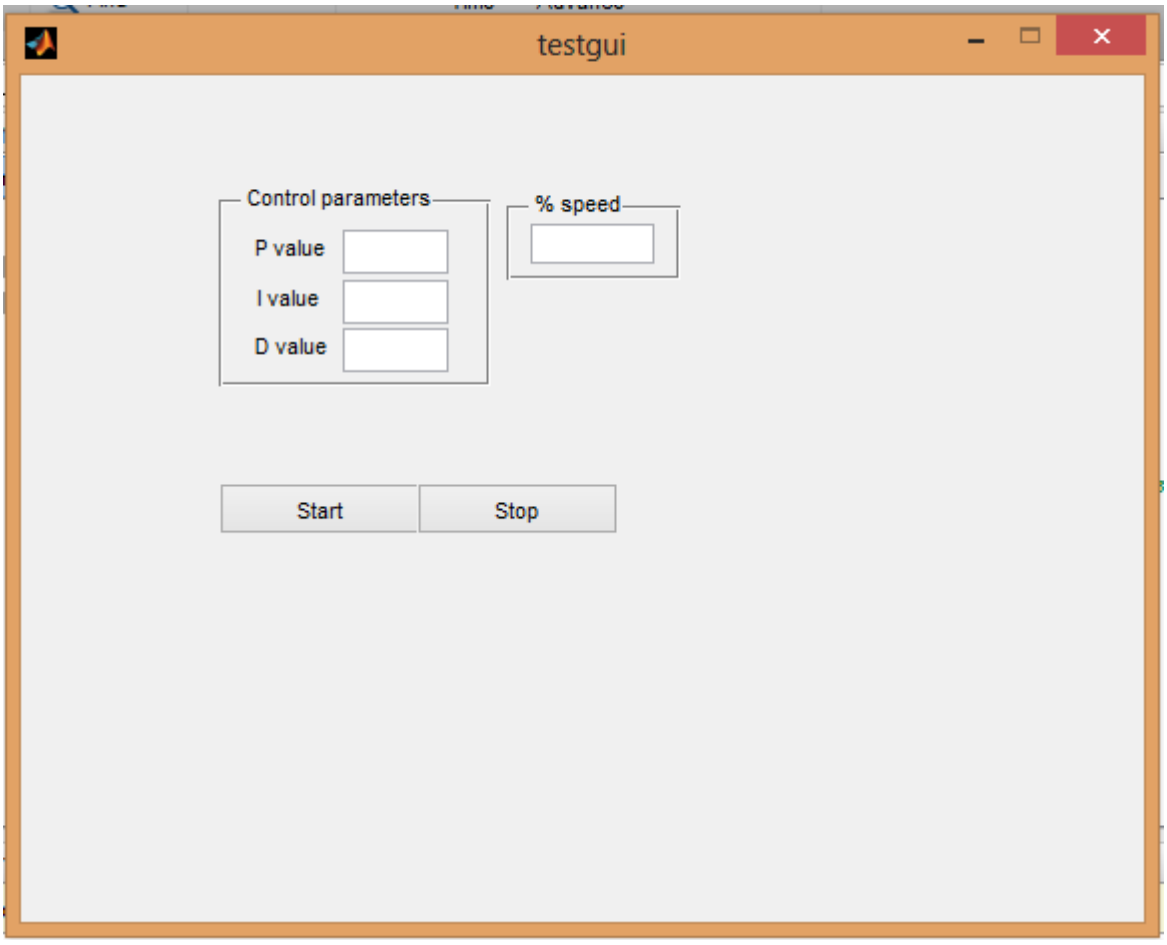



This is a simple GUI to control the speed of a dc toy motor. The various buttons given provide us with option to run the motor at 25%, 50% or 75% of the rated speed. When we click on “motor on” button the motor starts to run at rated(100%) speed.

We controlled the motor with a potentiometer using MATLAB and Arduino to check for the proper interface between the two.




## DC small servo-Motor control GUI with PID controller




Here, the GUI allows you to feed values in the control parameters and also to specify the speed at which you want the motor to run (in %).

By pressing start button, the servo motor runs at the specifed speed and using the given P,I,D values to the controller.
We searched a lot on the internet and in hyderabad to get a tachometer for this motor, but we couldn't get it. Actually tachometers are not made for small dc servo motors. So we decided to come up with a circuitry that would incorporate or mimic the use of a tachometer feedback.
What we proposed is to make an IR sensor based tachometer that would compute the position of the motor by interception method. A long stick was mounted on the motor and a sensor was place on top of the motor, as soon as the stick crosses the sensor, a dip in the analog value of the sensor is obtained and that is noted. Keepin a threshold on the dip value, we measured the time of interception each time. Now as we have the position value and the time between two sucecive intercepts, we can compute the speed at which the motor is rotating. This analog speed value is then fed to the controller as a feedback.

Output/Results obtained:

1.) Highly inaccurate readings
2.) The sensor would at times miss the interception.
3.) Also since the small servo motor rotates only by 180 degrees, therefore it is difficult to come up with a formula to compute the speed.

## Servo motor position control:

We wrote a code to control the position of a small- servo motor. The amount of turn in given to the potentiometer would result in an equivalent change in the position of the motor. Here the feedback was coming manually.

**Code:**

```
// Controlling a servo position using a potentiometer
(variable resistor)
// by Michal Rinott <http://people.interaction-
ivrea.it/m.rinott>

#include <Servo.h>

Servo myservo;  // create servo object to control a
servo
```

```
int potpin = 0;  // analog pin used to connect the
potentiometer
int val;    // variable to read the value from the
analog pin

void setup()
{
  myservo.attach(9);  // attaches the servo on pin 9 to
the servo object
}

void loop()
{
  val = analogRead(potpin);            // reads the
value of the potentiometer (value between 0 and 1023)
  val = map(val, 0, 1023, 0, 179);     // scale it to
use it with the servo (value between 0 and 180)
  myservo.write(val);                  // sets the
servo position according to the scaled value
  delay(15);                           // waits for the
servo to get there
}
```

## Servo motor speed control:

We controlled the speed of the servo motor by using a potentiometer as our feedback (manual). By rotating the potentiometer in one direction the motor speed would increase and by rotating it in opposite direction the speed would decrease.

**Code:**

```
#include <Servo.h>

Servo myservo;  // create servo object to control a
servo
                // a maximum of eight servo objects can
be created
```

```
int pos = 0;    // variable to store the servo position
int potpin = 5;  // analog pin used to connect the
potentiometer
int val;    // variable to read the value from the
analog pin
 int a=0;
void setup()
{
  myservo.attach(9);  // attaches the servo on pin 9 to
the servo object
}

void loop()
{
  if(Serial.available()>0)
  {
    a=Serial.read();
    Serial.println(a);
  }
     // scale it to use it with the servo (value
between 0 and 180)

  for(pos = 0; pos < 90; pos += a)  // goes from 0
degrees to 180 degrees
  {
  //val = analogRead(potpin);            // reads the
value of the potentiometer (value between 0 and 1023)
  //val = map(val, 0, 1023, 1, 30);    // in steps of 1
degree
    myservo.write(pos);              // tell servo to
go to position in variable 'pos'
    delay(40);                       // waits 15ms for
the servo to reach the position
  }

    for(pos = 60; pos < 180; pos += a)  // goes from 0
degrees to 180 degrees
  {
  //val = analogRead(potpin);            // reads the
value of the potentiometer (value between 0 and 1023)
```

```
  //val = map(val, 0, 1023, 1, 30);    // in steps of 1
degree
    myservo.write(pos);              // tell servo to
go to position in variable 'pos'
    delay(40);                       // waits 15ms for
the servo to reach the position
  }

  for(pos = 180; pos>=1; pos-= a)     // goes from 180
degrees to 0 degrees
  {
  //val = analogRead(potpin);            // reads the
value of the potentiometer (value between 0 and 1023)
  //val = map(val, 0, 1023, 1, 30);
    myservo.write(pos);              // tell servo to
go to position in variable 'pos'
    delay(40);                       // waits 15ms for
the servo to reach the position
  }

  delay(20);
}
```

## DC big servo motor:

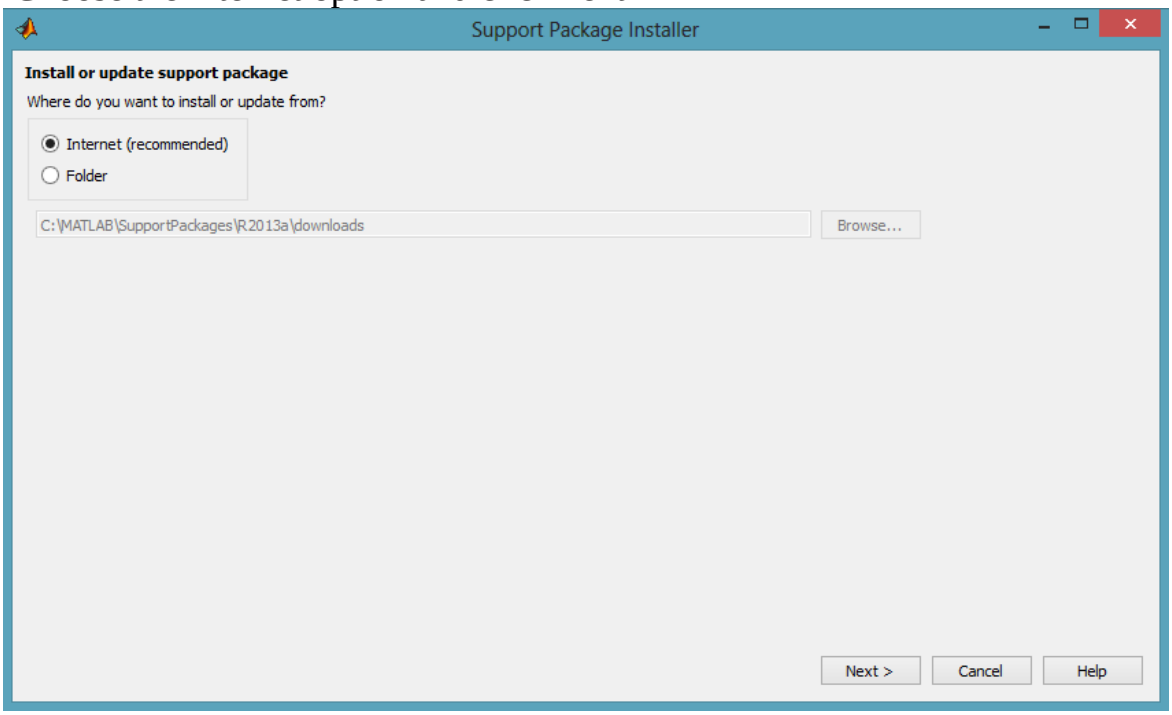
As mentioned above the tachometer is not available for a small 5V dc servo motor, so we have to go for a bigger high end dc servo motor. We managed to get a 24V dc motor and a 36V dc servo motor. Now to drive thesemotors we needed a constant high dc power source supply. We could not get a 24V constant supply, so we decided to run it at half the rated speed at 12V using a lead acid battery.

To investigate more about the motors we went to check the output on a Digital Signal Oscilloscope. There we could actually see the discontinuous output voltage wave form.

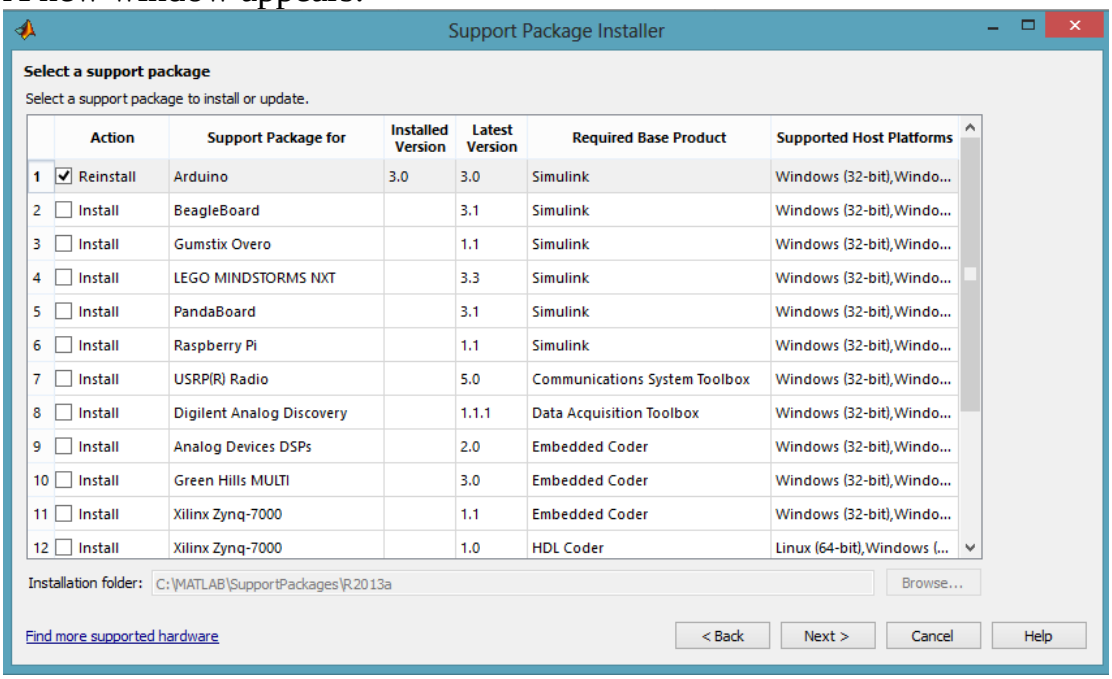
Now our next task is to interface this motor with Simulink.

**Interfacing with Simulink** involves to download the arduino package for Simulink from the internet. Follow the steps:

1.) Type targetinstaller in the command window.
2.) A pop up window will appear.
3.) Choose the internet option and click next



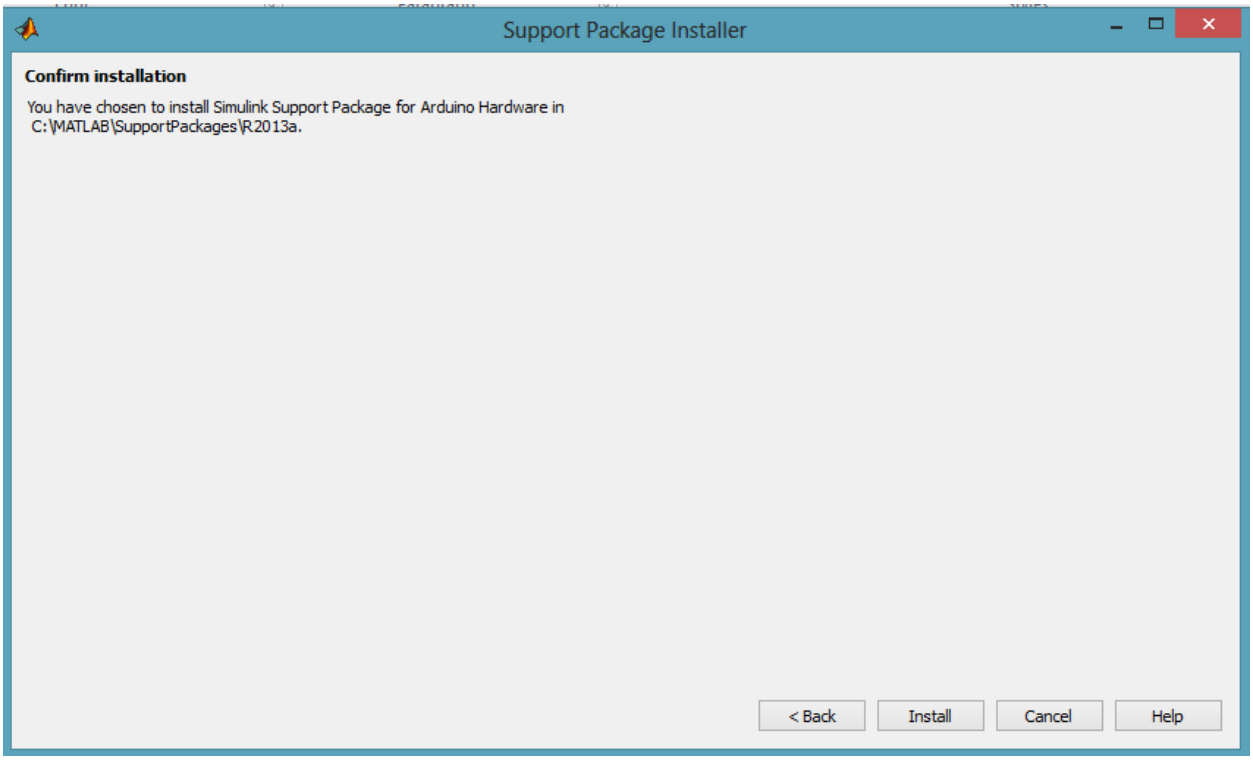

4.) A new window appears:




select arduino package and click on next.

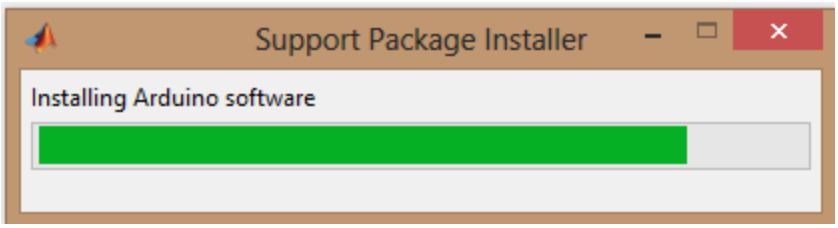
5.) Click on I agree and next

6.)

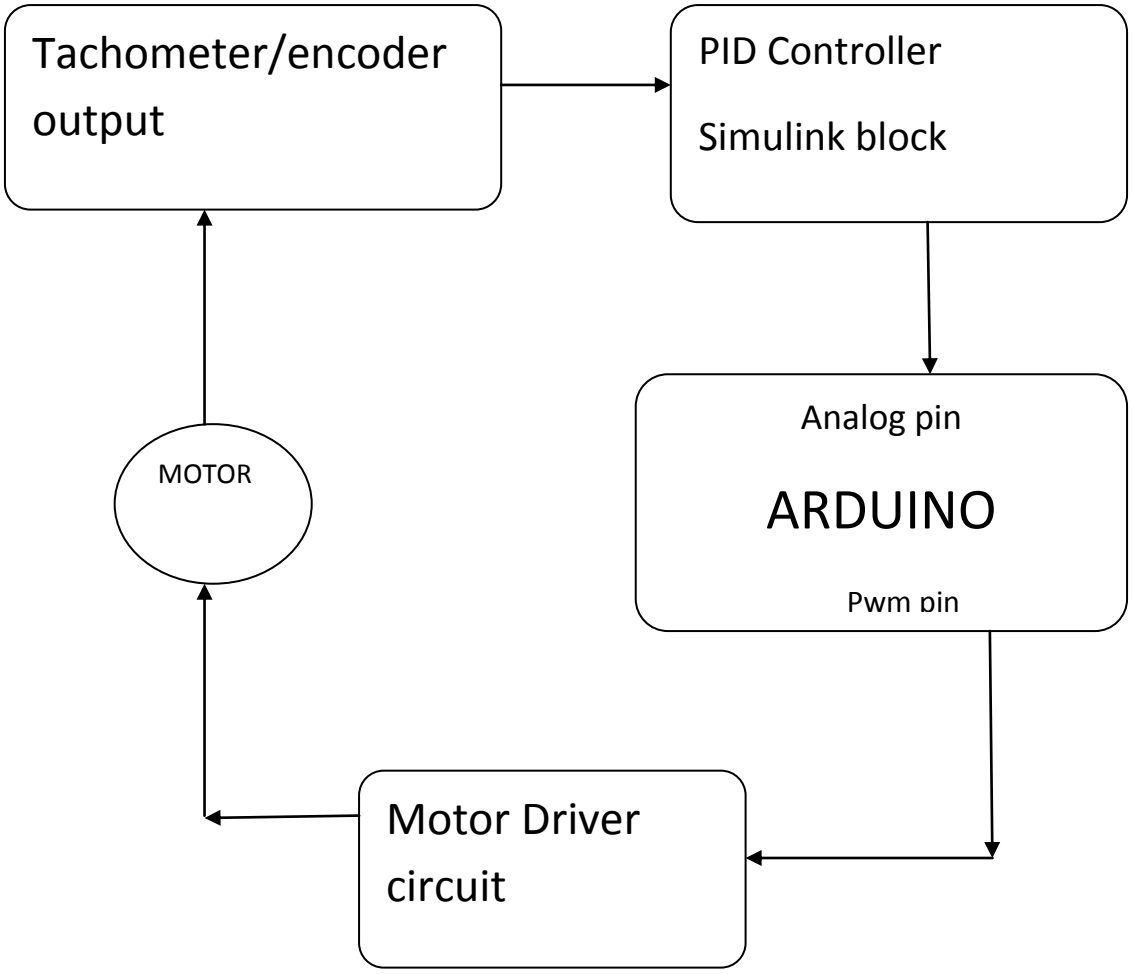



click on install to start installation

7.)

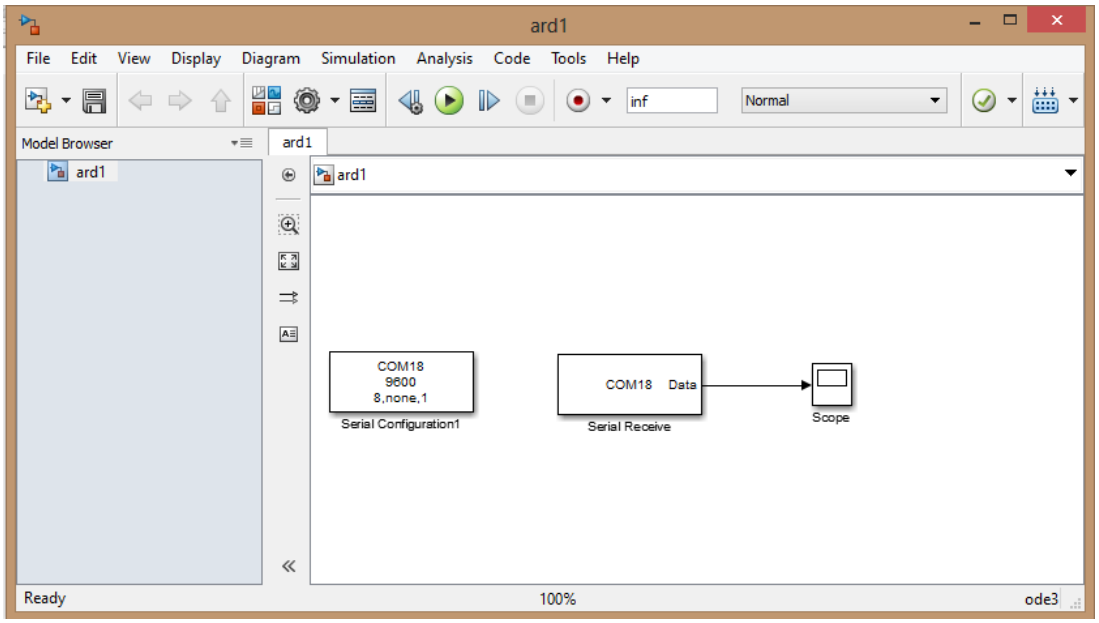


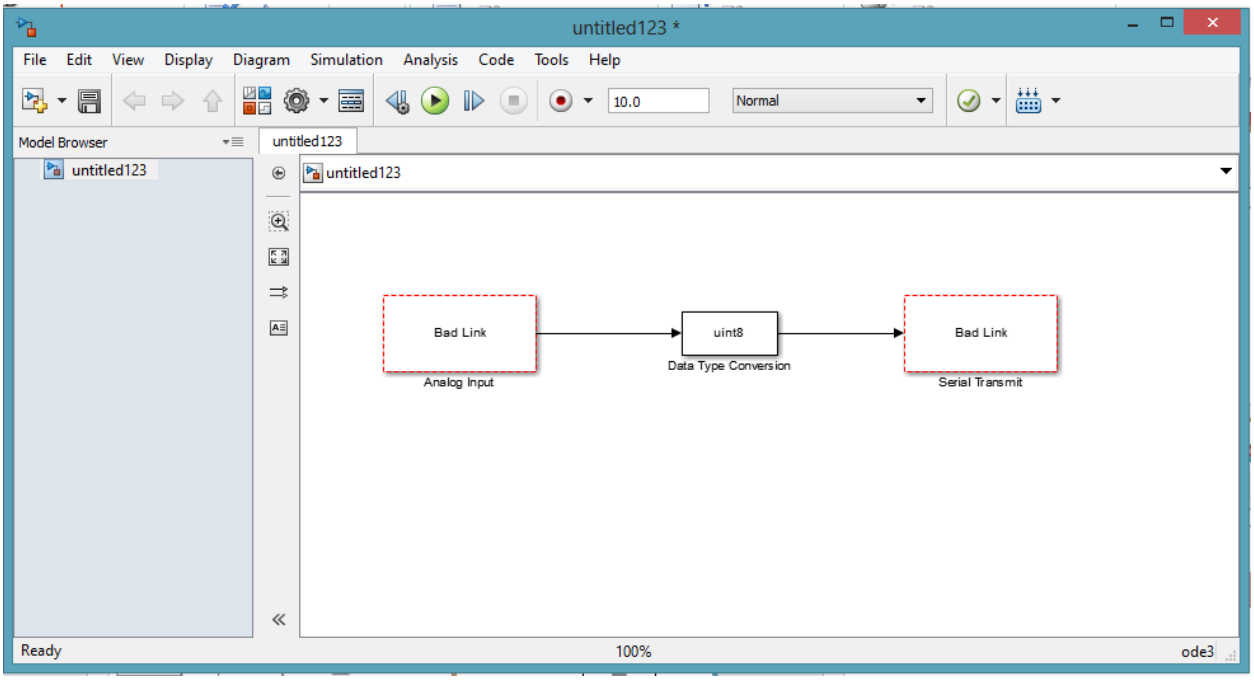
**Runnig the simulink block diagram:**



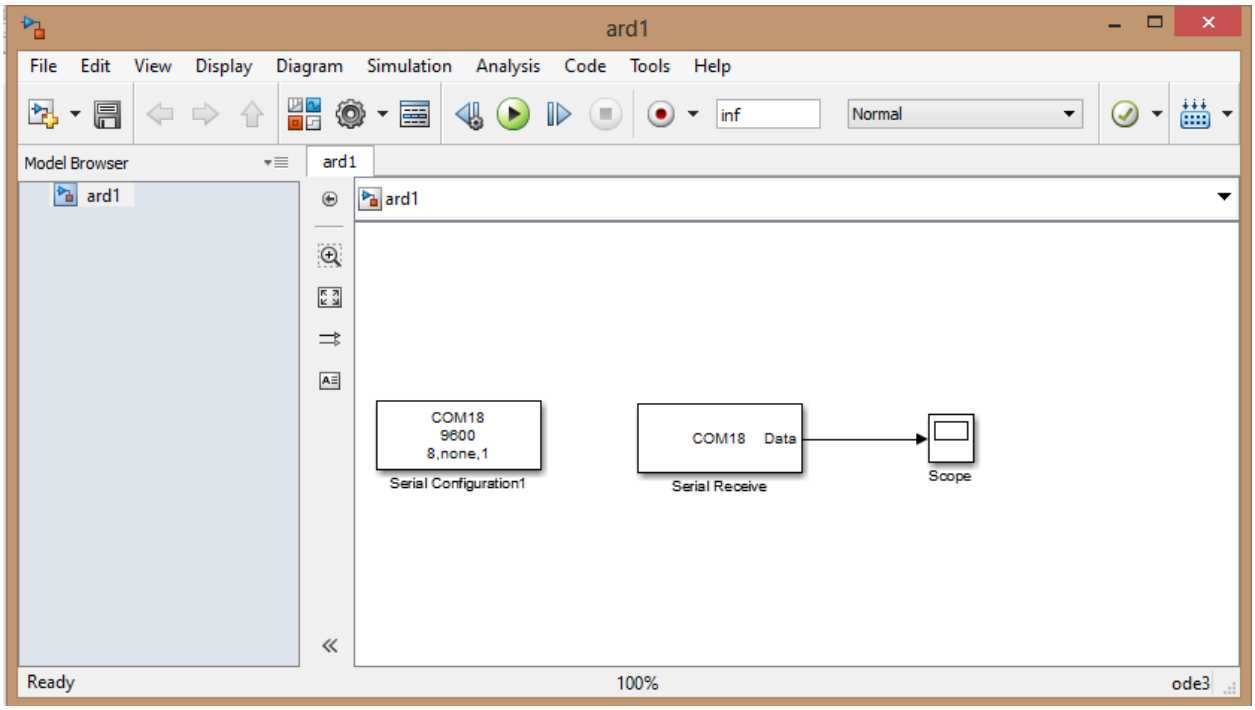
# Circuit connection:

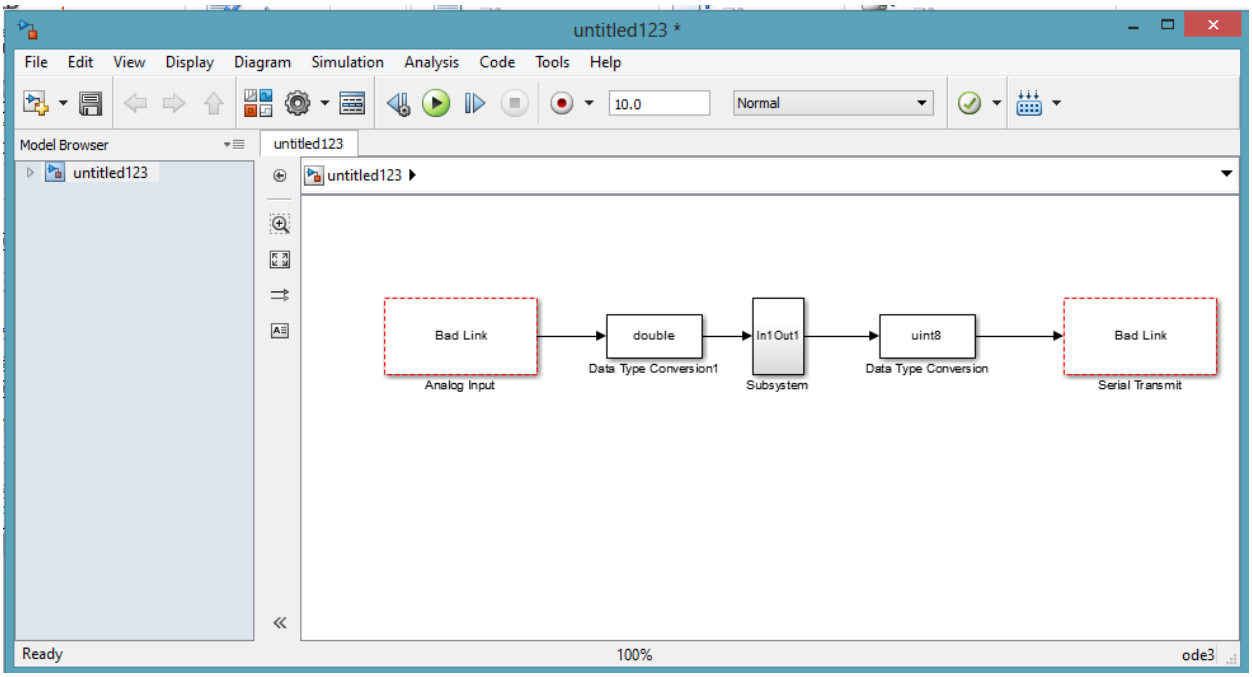
Without controller:






With controller:

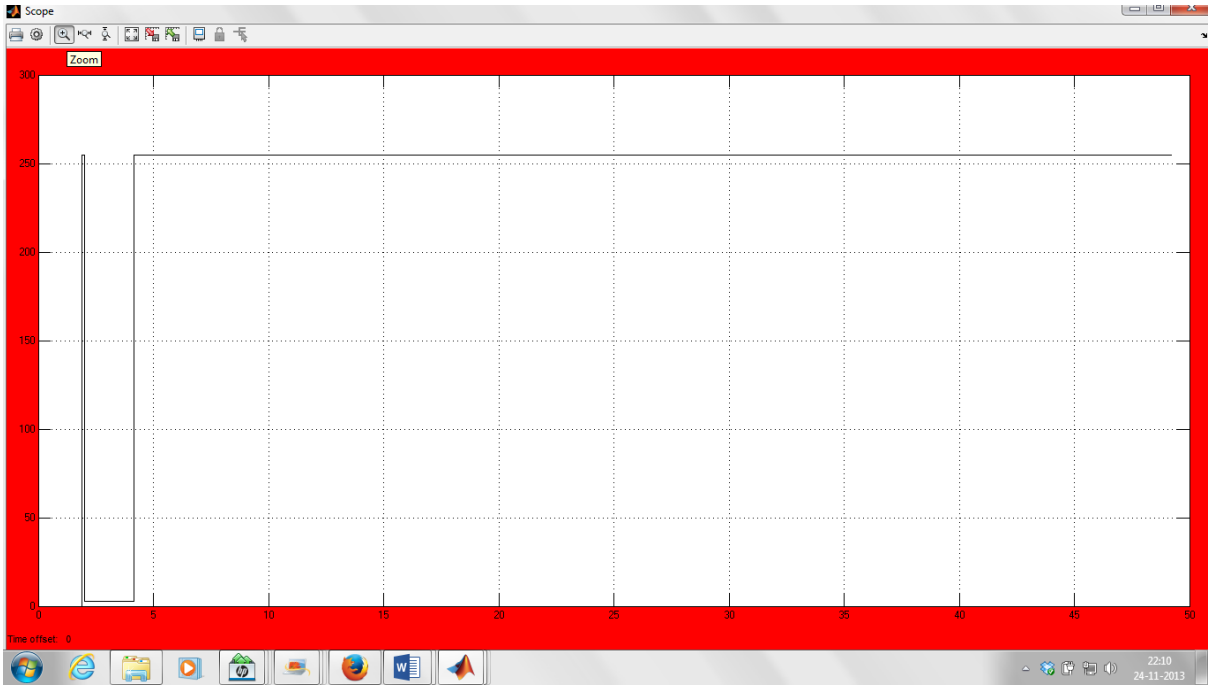




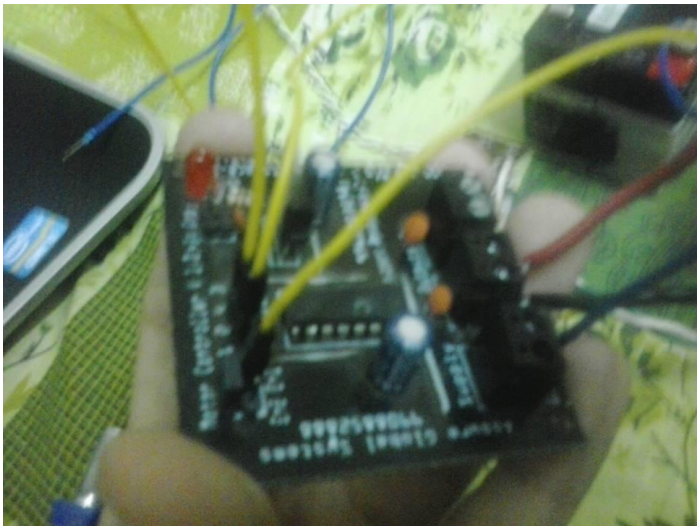

# Problems faced:

Tried the tacho output at the digital input, we dint get any values, it was a waveform with short impulses.
Tried the tacho output through the Rx pin of the arduino, but it would not upload the code as transmission and recption cannot happen simultaneiously in arduino.
Then finally we went for the analog pin read of the arduino. But here also it wouldn't accept directly.
So we thought of a way to send it serially through the output COM port of our laptop.

Next we ran the block diagram of a simple open loop real time system, we got results in the form of wave impluses of vey short duration.
We introduced our PID controller and then tried, here we got a result of the form:




This output is without the feedback but using the controller. Actually the problem we are facing is that the big motor requires a very large current. So we used the motor driver circuit to control the current, but in the motor driver each H-bridge can tolerate a max of around 700mA. So on trying to power the motor the L293D ic got burst!!

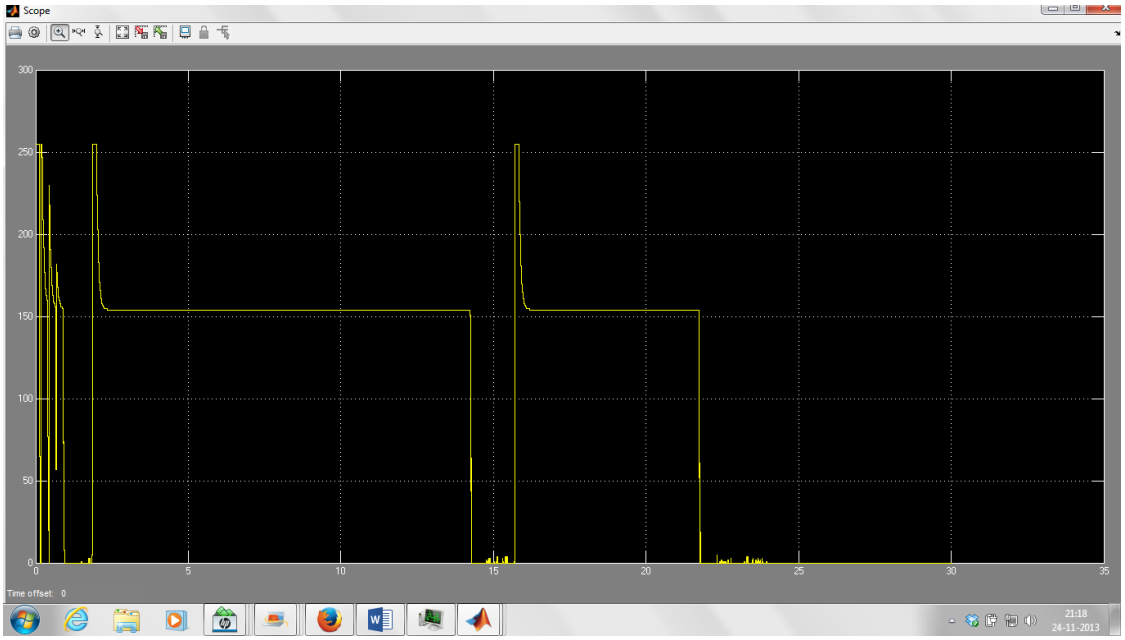


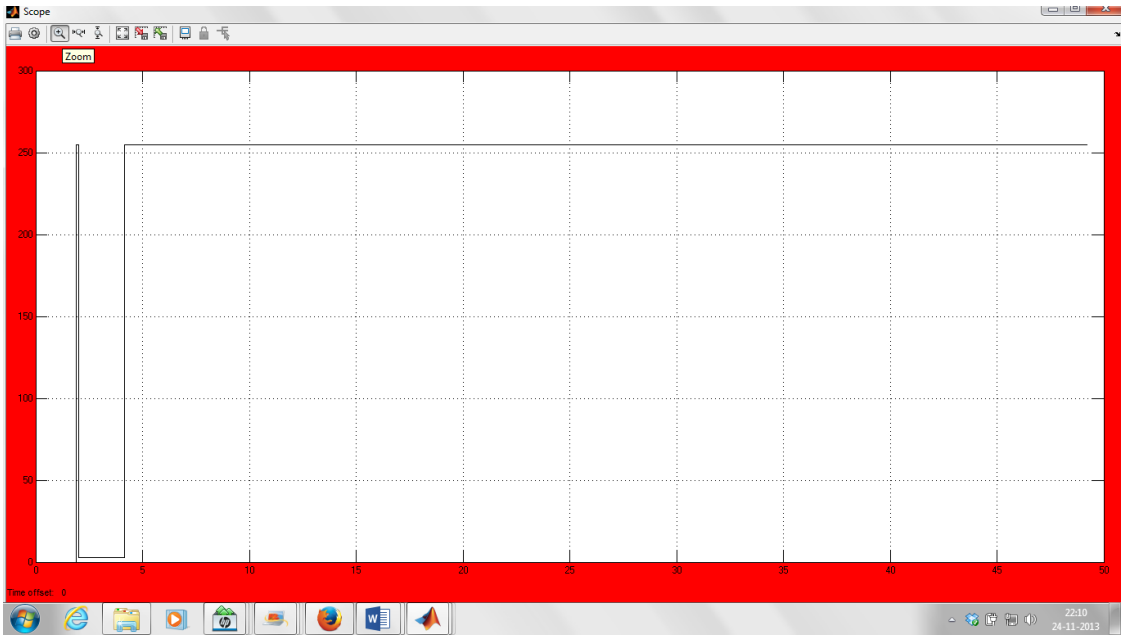
Hence we could not control the motor with such high current directly from arduino, neither by using a L293D. We have to use a higher current rating motor driver IC. We could not find one of 2A current capacity Ic. So we decided to make

the H-bridge circuitry using npn transistors, but it did not work..
The only way out is to Make h-bridge using high current rating MOSFETs. We couldn’t arrange for high MOSFETs so we did not test the motor with a controlled feed back loop for the controller. The results that we got are not for rated speed because we did not have a continuous 24V dc supply and more over the motor parameters were unkown.

# INFERENCE/CONCLUSION

Comparison with lead lag controller and PID



# References:


- A MATLAB/Simulink-Based Interactive Module for Servo Systems Learning (by Noudine Aliane)
- Speed Control of D. C. Servo Motor By Fuzzy Controller Dipraj, Dr. A. K. Pandey
- PID CONTROLLER DESIGN FOR CONTROLLING DC MOTOR SPEED USING MATLAB APPLICATION MOHAMED FARID BIN MOHAMED FARUQ
- Probably the best simple PID tuning rules in the world Norwegian University of Science and Technology Sigurd Skogestad Department of Chemical Engineering N–7491 Trondheim Norway Submitted to Journal of Process Control July 3, 2001This version: September 12, 2001
- Mathworks.com
- Wikipedia
- Google

# Appendix:

***Matlab arduino GUI code***:

```
function varargout = testgui(varargin)

% TESTGUI MATLAB code for testgui.fig
%      TESTGUI, by itself, creates a new TESTGUI or raises the existing
%      singleton*.
%
%      H = TESTGUI returns the handle to a new TESTGUI or the handle to
%      the existing singleton*.
%
%      TESTGUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in TESTGUI.M with the given input
arguments.
%
%      TESTGUI('Property','Value',...) creates a new TESTGUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before testgui_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to testgui_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 testgui

% Last Modified by GUIDE v2.5 18-Oct-2013 19:07:15

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @testgui_OpeningFcn, ...
                   'gui_OutputFcn',  @testgui_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 testgui is made visible.
function testgui_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 testgui (see VARARGIN)

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

% Update handles structure
guidata(hObject, handles);

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


% --- Outputs from this function are returned to the command line.
function varargout = testgui_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;


% --- Executes on button press in button_start.
function button_start_Callback(hObject, eventdata, handles)
% hObject    handle to button_start (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (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 edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as a double
p = str2double(get(handles.edit4,'String'));
p



function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (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 edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5 as a double
i = str2double(get(handles.edit5,'String'));
i



function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (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 edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6 as a double
d = str2double(get(handles.edit6,'String'));
d


function edit7_Callback(hObject, eventdata, handles)
% hObject    handle to edit7 (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 edit7 as text
%        str2double(get(hObject,'String')) returns contents of edit7 as a double

com=serial('COM14','BaudRate',9600);
fopen(com);
disp('hi')
speed = str2double(get(handles.edit7,'String'))
fprintf(com,speed)
%fclose(com);
```

***MATLAB support package for arduino:***

```
% This files installs the MATLAB support package for Arduino (ArduinoIO
package).

%   Copyright 2011 The MathWorks, Inc.

% look for arduino.m
wa=which('arduino.m','-all');

% make sure we are in the right folder and there are no other arduino.m files
if length(wa) < 1,
   msg=' Cannot find arduino.m, please run this file from the folder containing
arduino.m';
   error(msg);
end
if length(wa) > 1,
   msg=' There is at least another arduino.m file in the path, please delete any other
versions before installing this one';
```

```
   error(msg);
end

% get the main arduino folder
ap=wa{1};ap=ap(1:end-10);

% Add target directories and save the updated path
addpath(fullfile(ap,''));
addpath(fullfile(ap,'simulink',''));
addpath(fullfile(ap,'examples',''));
disp(' Arduino folders added to the path');

result = savepath;
if result==1
   nl = char(10);
   msg = [' Unable to save updated MATLAB path (<a
href="http://www.mathworks.com/support/solutions/en/data/1-
9574H9/index.html?solution=1-9574H9">why?</a>)' nl ...
         ' On Windows, exit MATLAB, right-click on the MATLAB icon, select
"Run as administrator", and re-run install_arduino.m' nl ...
         ' On Linux, exit MATLAB, issue a command like this: sudo chmod 777
usr/local/matlab/R2011a/toolbox/local/pathdef.m' nl ...
         ' (depending on where MATLAB is installed), and then re open MATLAB
and re-run install_arduino.m' nl ...
         ];
   error(msg);
else
   disp(' Saved updated MATLAB path');
   disp(' ');
end

clear wa ap result nl msg
```