UNIVERSIDAD de VALLADOLID

ESCUELA de INGENIERÍAS INDUSTRIALES

INDUSTRIAL ENGINEERING

FINAL PROJECT WORK

PART I : READING IN A BAR'S POSITION BY A MAGNETIC ENCODER AND AN ARDUINO UNO MICROCONTROLLER PART II : POSITION CONTROL OF A ROTATING BAR WITH ENGINE AND PROPELLER USING LABVIEW AND ARDUINO

Author

Luyck, Robin

Mentor:

García Ruiz, Francisco Javier

System engineering and automation

June — 2013

Forword

First of all I want to thank my mentor at the University F. Javier García, who was always there to help me if I had a problem with the final project work.

Also a thanks to my promoter-docent-mentor at the KHLim in Belgium Wim Claes to do the weekly check-ups.

Also I want to thank the following people at the University: Luis A. Santos Dominguez for giving me all the software and tools to create the project, Sánchez Báscones, Ma Isabel to help me with all the documents for Erasmus and Jesús Zamarreño for helping me with some troubleshooting.

Then a big thanks to the following people who made this Erasmus possible for me:

Greet Raymaekers, Lecturer / Coordinator International Relations Engineering at the KHLim in Belgium to do all the research on the available Erasmus places and all the e-mails and paper work.

My parents, family and friends for giving me this opportunity and supporting me.

My girlfriend Alice for supporting and helping me always.

I also want to say thanks to all the docents at the KHLim for teaching and helping me during the 3 years of my Bachelor electro-mechanics.

Robin Luyck

Erasmus Valladolid February – June 2013

In February of 2012 we had an information hour at the KHLim (Limburg Catholic University College (LCUC)) about doing our internship abroad.

I am a very motivated and enthusiastic person, I like challenges, experiencing the technologies and the culture all over the world. Therefore I was very interested in the concept of Erasmus. I was pleased to do my Erasmus in Valladolid, Spain. During this Erasmus I had to do my *final project work* for my bachelor degree electro-mechanics with option electro-mechanic.

The Erasmus Program (**E**u**R**opean Community **A**ction **S**cheme for the **M**obility of **U**niversity **S**tudents) is a European Union (EU) student exchange program and gives students the chance to explore different countries in Europe and explore the technologies and culture abroad.

Valladolid is the capital of the province of the same name, and is part of the region of Castile-Leon in Spain.

Valladolid is a reasonably large industrial city to the **northwest of Madrid**. It is one of the major regional center in Castile-Leon. The city has an older core with some interesting buildings. However it has focused more on industry than tourism. Nowadays the population of Valladolid is near **322,000 inhabitants**.

I learned a lot during this Erasmus, I got to know the Spanish mentality and culture, visited some other parts and beautiful places in Spain. It was a positive experience for me.

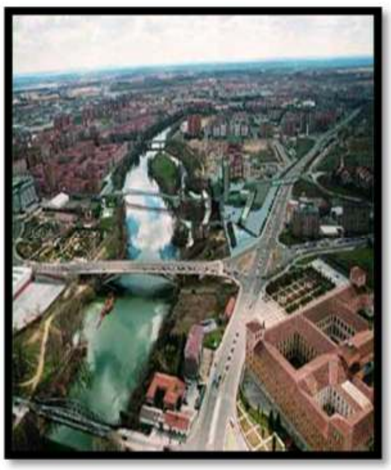


Table of contents

Introduction

This thesis is about my final project work I did at the industrial engineering and technology school from the university of Valladolid (UVa) from February 2013 until June 2013.

The thesis is divided in different chapters. First off I will give the solutions and programs that were my task in this final project work and explain the most important thing, which is how reading in the bars angular from plant n°1 and then the PID control on plant n°2.

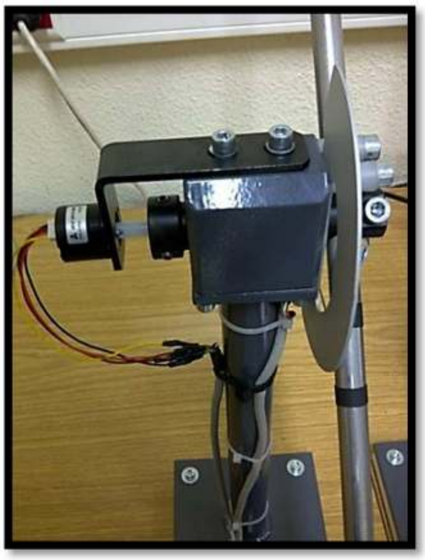
After this follow chapters on the software, the hardware, sources and datasheets.

➔ *The project is divided in 2 parts:*

Goals part I

Implementing a 10-bits magnetic encoder on a plant its rotating shaft and writing a firmware (sketch) in Arduino to read in the bars angular. The firmware (sketch) will be able to turn the binary signal into the decimal physical angular from the bar, which then later can be used to detect the bars position.

I added something extra on this **part I**, which is implementing the program Processing, this to visualize the angular in a graph, instead of just getting the angular out on the serial monitor from the Arduino program.



Plant n°1

- Knowledge of the Arduino Uno microcontroller
- Knowledge of the 10 bits Magnetic Encoder AEAT-6010
- Knowledge of writing C/C++ code
- Knowledge of the Processing open source program
- Knowledge of the steps for installing the software to run on a computer system.

Goals part II

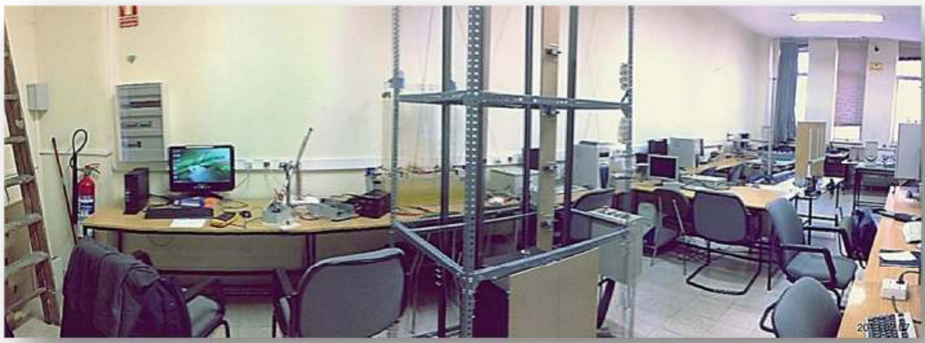
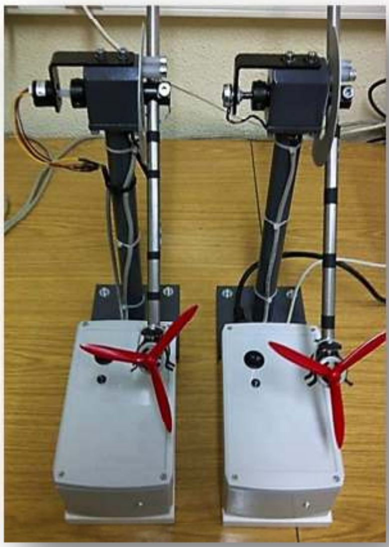
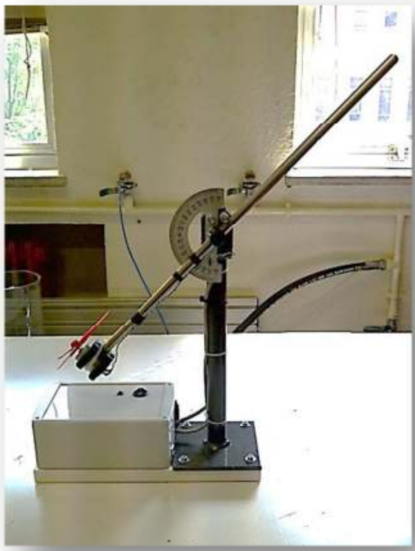
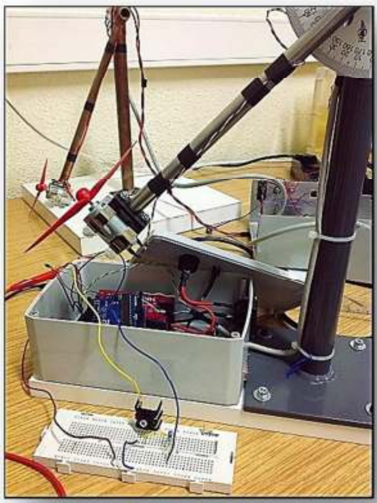
Position control on a bar with motor and propeller, this by implementing PID control on the plant. All this by using the graphical program LabVIEW, Arduino Uno microcontroller and all the necessary electronic parts for position measuring and power control. I was already familiar with the Arduino microcontroller doing hobby electronic projects in my free time and had read a lot of books about it, which gave me a good basic knowledge about the Arduino open-source-platform.

The plant consists of two bars anchored to swing mode: at one end of the rod sits a propeller engine that by his force will adjust the bars position. The position is measured by a potentiometer attached to the shaft and is proportional to the rotation of the propeller and will be read in by the Arduino Uno microcontroller analog input pin.

Plant nº2

- Knowledge of the Arduino Uno microcontroller
- Knowledge of writing C/C++ code
- Knowledge of the graphical program LabVIEW + it's toolkits
- Knowledge to make the Arduino communicate with LabVIEW
- Knowledge of the open-source hardware Fritzing to create the electronic drawings
- Obtaining of the mathematical model of the system through physical laws
- Development of an open loop system that allows us the get the PID values
- Development of a closed loop system that controls the correct position of the bar against possible disturbances outside the system
- Improving the system to make it more stable and reliable.
- Knowledge of Pulse Width Modulation (PWM)
- Knowledge of the steps for installing the software to run on the computer system.

Pictures at the laboratory

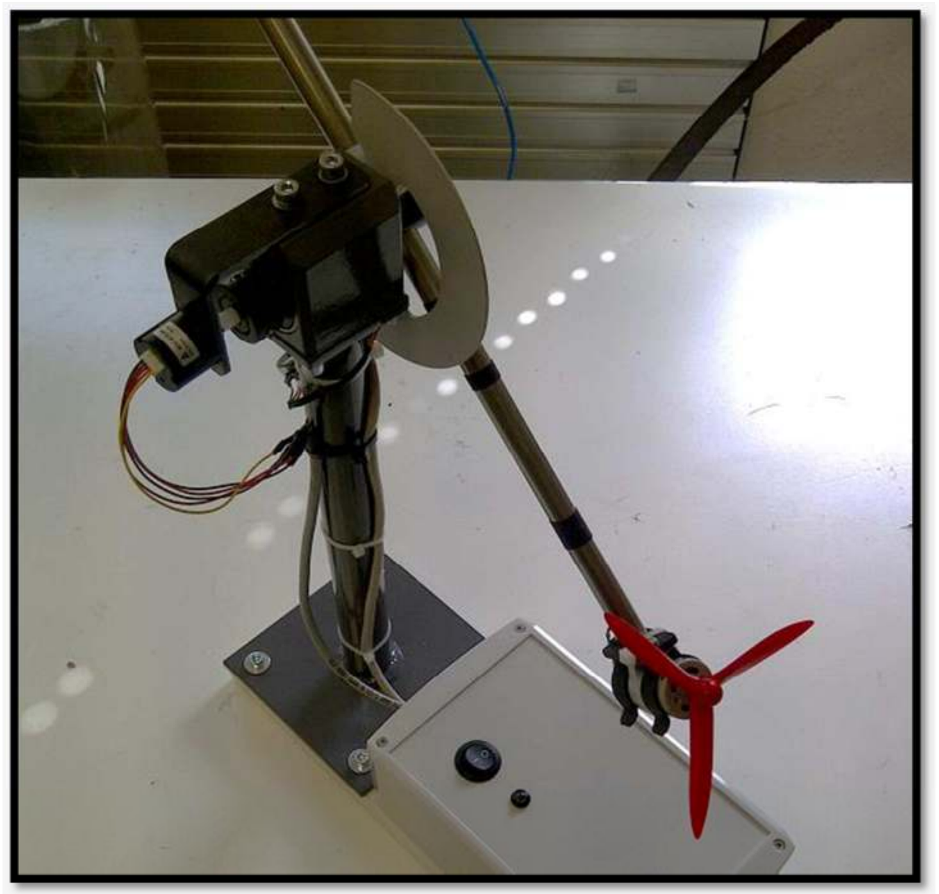
1. CHAPTER 1: Part I : reading in a bar's position by a magnetic encoder and an Arduino Uno microcontroller

1.1 Intro

At the University there is a laboratory for control techniques with +- 15 plants like the one I use here, the students all have to do PID control on it with an Arduino microcontroller communicating with Matlab and Simulink. On all the plants shaft there is a potentiometer attached to detect the bars angular and give this value to the PID controller as process value, which will depending on this value and the set point change the controller its output.

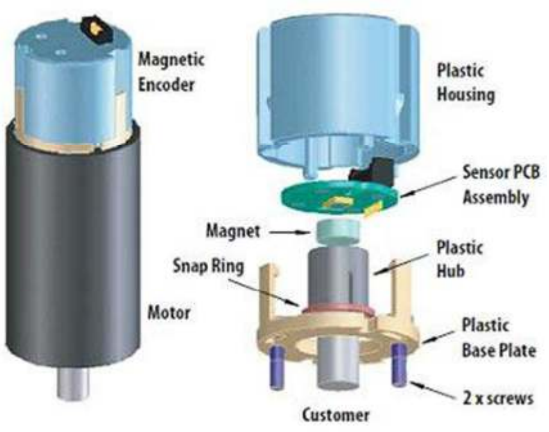
My task for the final project work was to implement a magnetic encoder on the shaft instead of the potentiometer and read in the angular from the bar into the Arduino software, so they later can use it to do PID control.

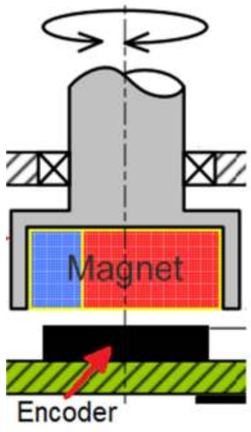
I added an *extra program* to this Part I : implementing the program *Processing* which is an open source programming language. Processing makes it possible to create images, animations, and interactions. I will create a graph to show the bars angular.



Plant n°1

1.2 Magnetic encoder AEAT-6010





The AEAT-6010 serie is a high performance, low cost, optical absolute magnetic encoder module, designed to detect absolute angular position for precise motor feedback.

This magnetic encoders is ideal for angular detection within 360°. Based on magnetic technologies, the device is non-contact and ensures reliable operations. It is able to provide absolute angle detection upon power-up, with a resolution of 0.35°(10 bits version), which is equivalent to 1024 positions (0-1023) per revolution respectively. The positional data is provided in serial bit stream. There is no upper speed limit; the only restriction is that there will be fewer samples per revolution as the speed increases.

In addition, the magnetic encoder comes complete with housing that enables easy assembly and integration to customers' applications.

With a 3-wire serial interface, the AEAT-6010 serie enables the provision of absolute position data in binary format. The encoder is compatible with a wide range of industrial automation applications with a single 5V power supply input, which are accessed through four 0.025 inch square pins located on 0.1 inch centers.

1.2.1 Magnetic encoder's mean features

- **10 bits** resolution
- **Contactless sensing** technologies
- Wide temperature range from -40° to 125°C
- **Absolute angular position detection**
- **Synchronous Serial Interface** (SSI) output for absolute position data (**binary format**)
- Single **5V** supply
- Easy assembly, no signal adjustment required
- Direct connectivity through PCB (**P**rinted **C**ircuit **B**oard)
- Small form factor (23mm diameter x 19mm height)
- 6mm shaft

1.2.2 Electrical connections encoder

Electrical Connections

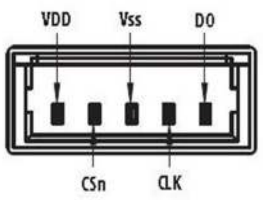


Figure 4. Electrical Connections

Pin	Symbol	Description
1	VDD	5V Supply Voltage
2	CSn	Chip Select – Input (See Figure 2)
3	VSS	Supply Ground
4	CLK	Serial Clock - Input (See Figure 2)
5	DO	Serial Data - Output. (See Figure 2)

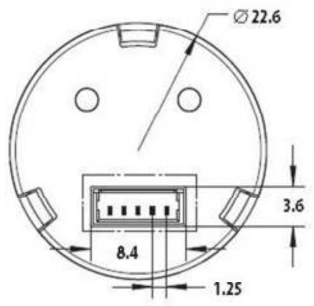
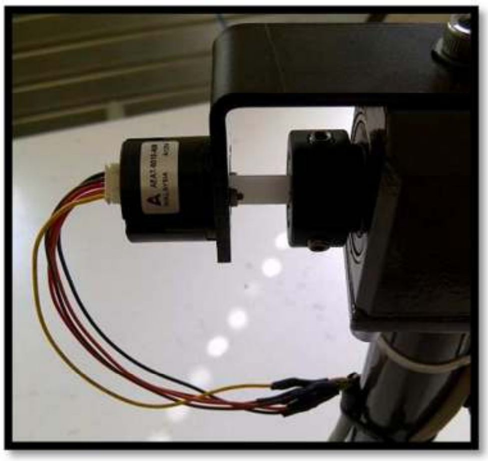


Figure 5. Basic connector dimensions

Connections between the encoder connector and the Arduino Uno



Wires leaving the encoder and entering the Arduino with different colors

I soldered the 5-pins female connector to a 5 wires multi bus, therefore the colors change like you can see on the next page.

Pin connection:

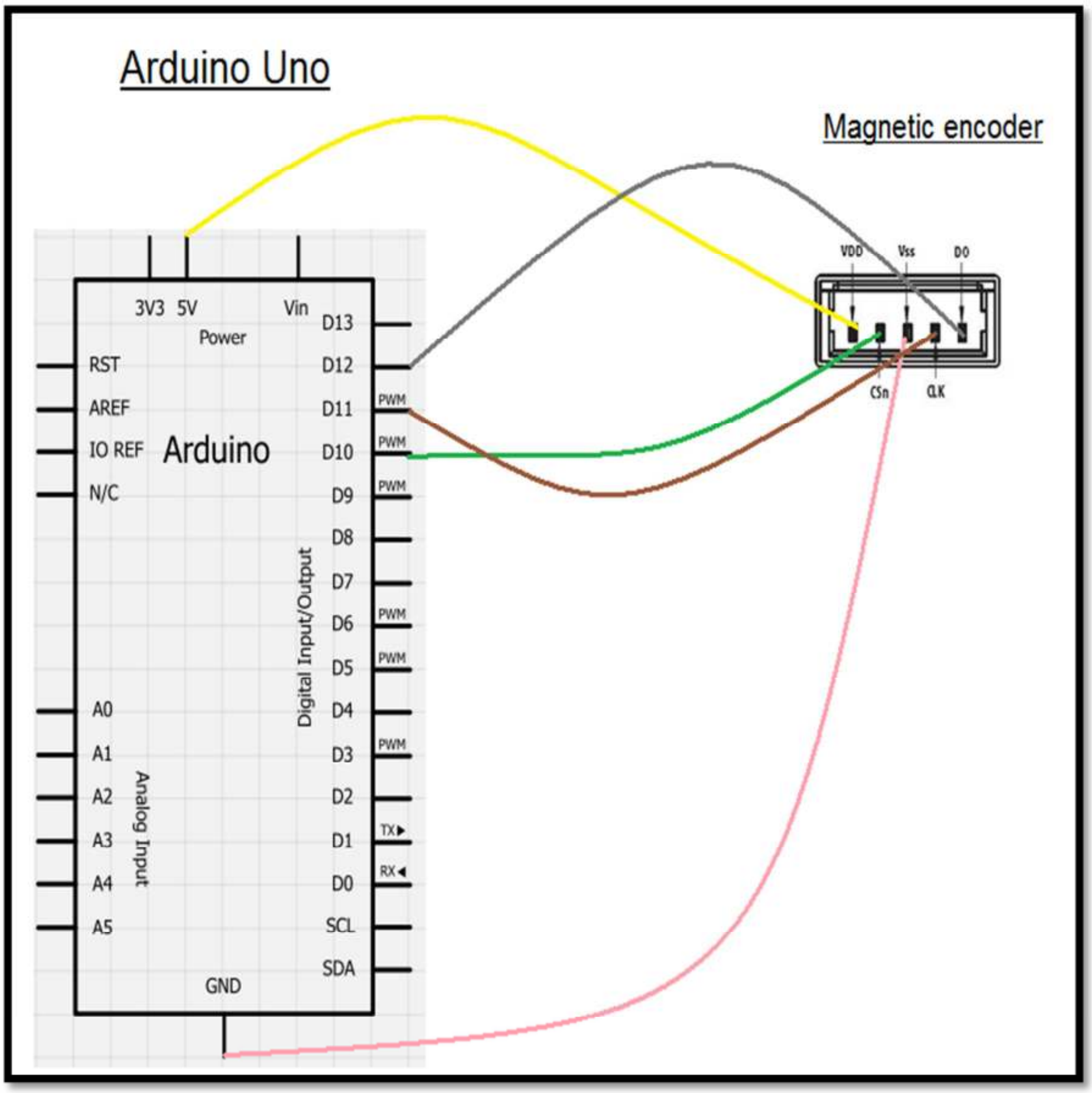
1: wire color in white box: **Yellow**: **VDD** (5V supply voltage): 5V power supply Arduino

2: wire color in white box: **Green**: **CSn** (Chip select): Digital output 10 Arduino

3: wire color in white box: **Pink: VSS** (GND): GND Arduino

4: wire color in white box: **Brown**: **CLK** (Serial clock-input): Digital output 11 Arduino

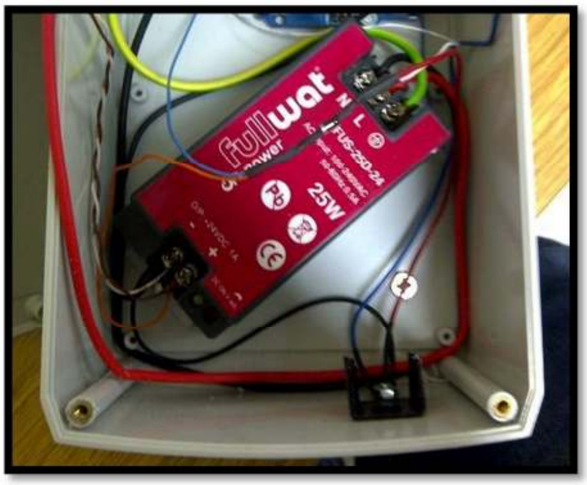
5: wire color in white box: **Grey**: **DO** (Serial Data-Output): Digital input 12 Arduino



Wire colors to know entering the Arduino

1.3 Electronic circuit plant

In this part I, I will only be using the Arduino Uno in the white box. The features about this one are explained in the hardware chapter. There is a transformer and transistor attached in the box, but I will not be using them. The Arduino will be powered through USB.



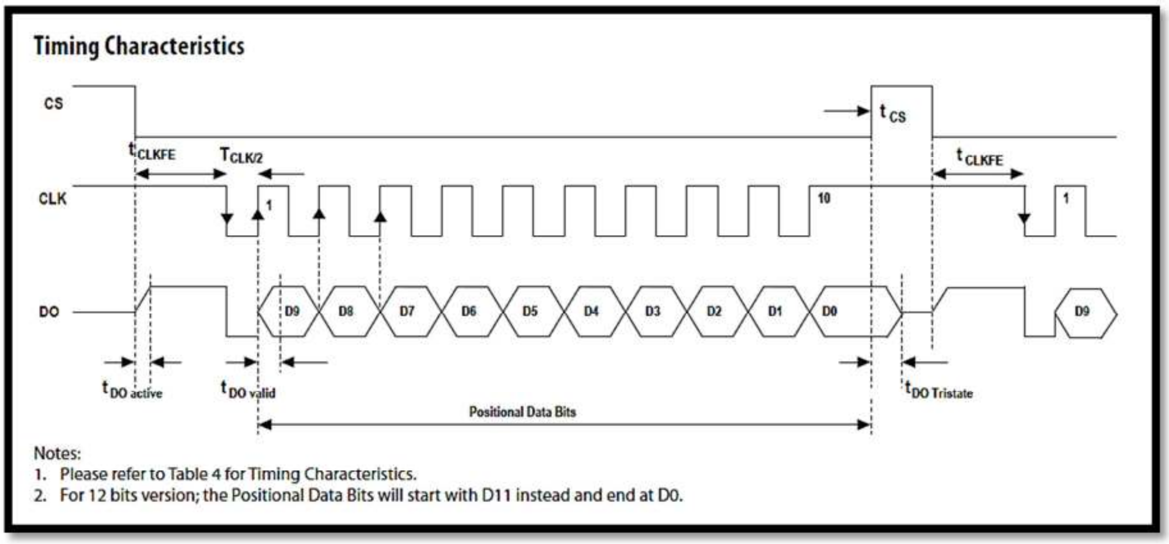
The transformer and transistor will not be used in this part I

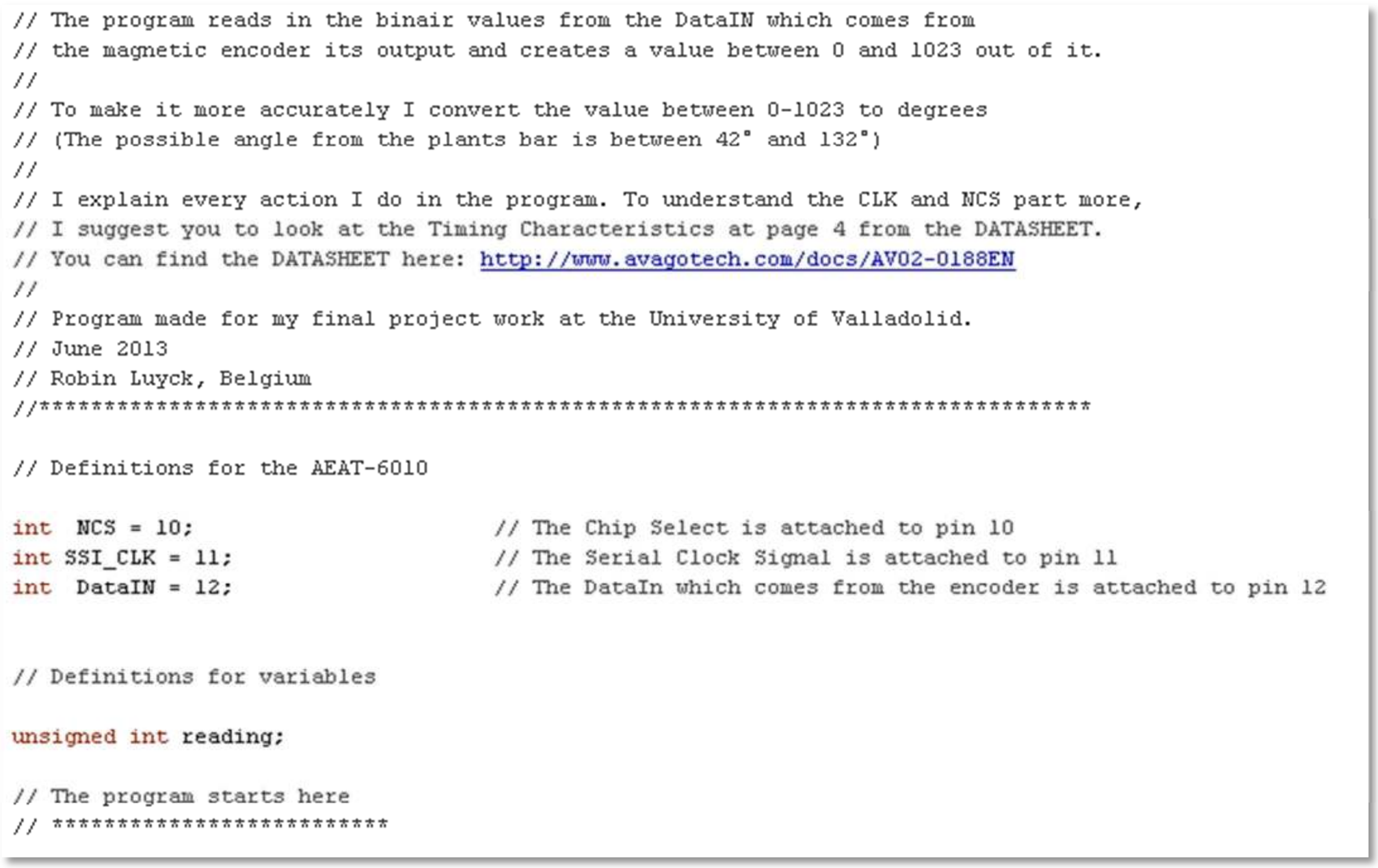
1.4 Arduino sketch

1.4.1 Explanation firmware

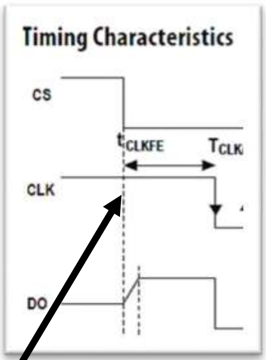
The following firmware is written to control the **S**ynchronous **S**erial **I**nterface for the AEAT-6010 magnetic encoder, to read in a bars angular in degrees.

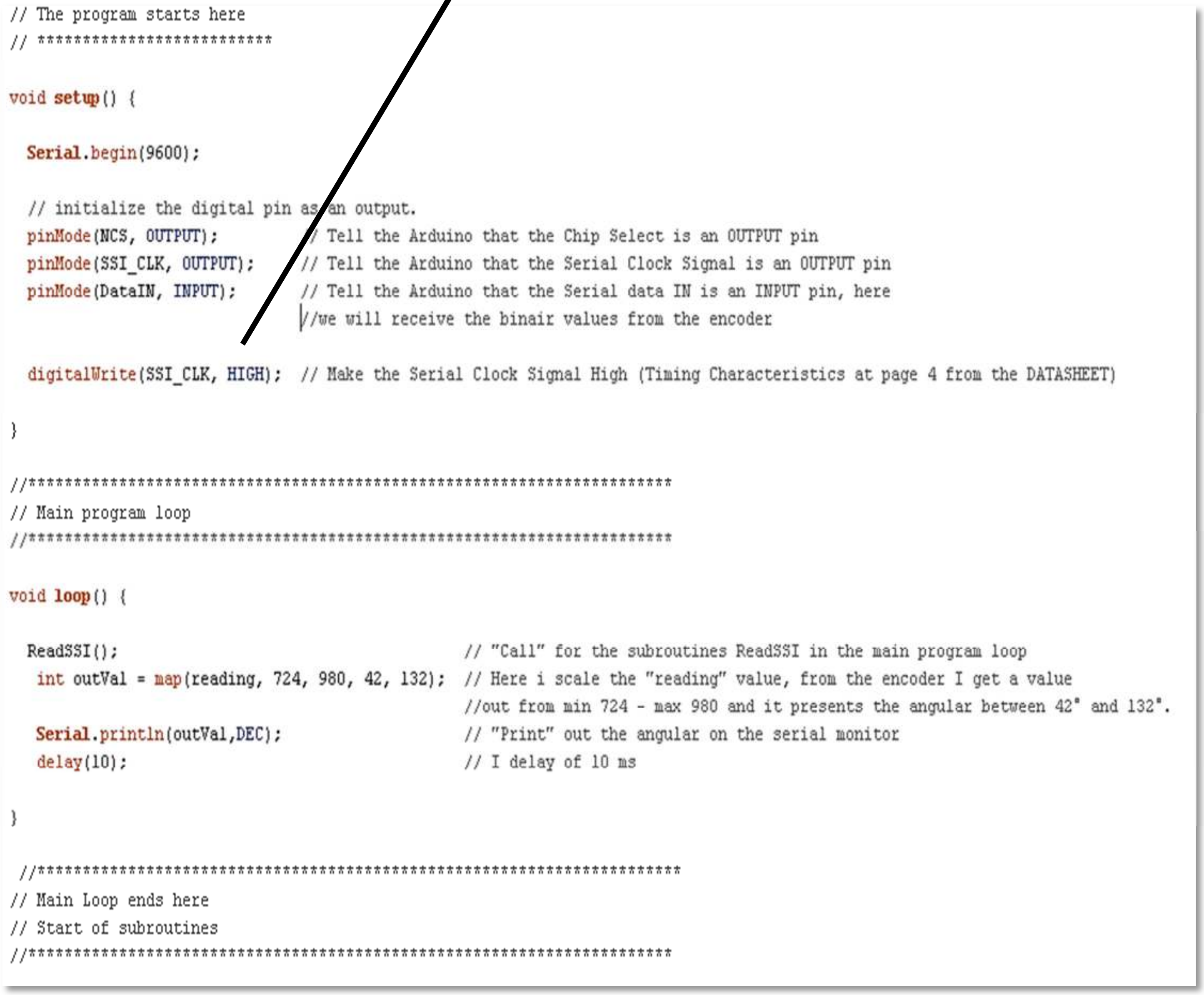
To understand the CLK, NCS and Serial data coming from the encoder better, take a look at the following Timing Characteristics from the encoders datasheet:


Timing Characteristics
CS
CLK
DO
t CLKFE
T CLK/2
1
10
t CS
t CLKFE
1
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
D9
t DO active
t DO valid
Positional Data Bits
t DO Tristate
Notes:
1. Please refer to Table 4 for Timing Characteristics.
2. For 12 bits version; the Positional Data Bits will start with D11 instead and end at D0.

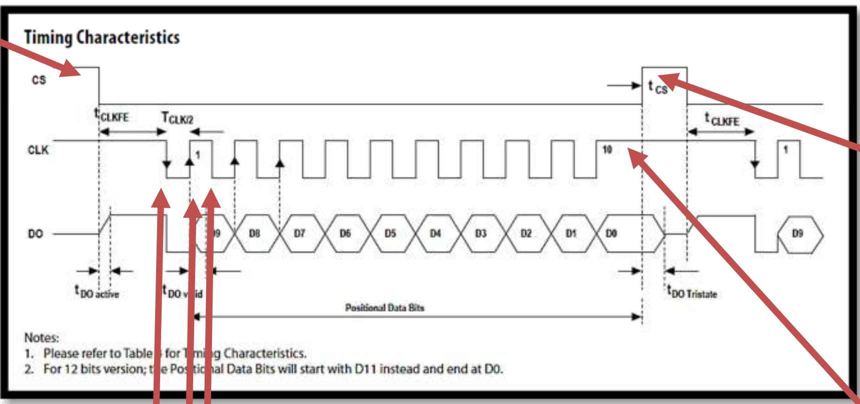


```
// The program reads in the binair values from the DataIN which comes from
// the magnetic encoder its output and creates a value between 0 and 1023 out of it.
//
// To make it more accurately I convert the value between 0-1023 to degrees
// (The possible angle from the plants bar is between 42° and 132°)
//
// I explain every action I do in the program. To understand the CLK and NCS part more,
// I suggest you to look at the Timing Characteristics at page 4 from the DATASHEET.
// You can find the DATASHEET here: http://www.avagotech.com/docs/AV02-0188EN
//
// Program made for my final project work at the University of Valladolid.
// June 2013
// Robin Luyck, Belgium
//*********************************************************************************

// Definitions for the AEAT-6010

int  NCS = 10;                      // The Chip Select is attached to pin 10
int SSI_CLK = 11;                   // The Serial Clock Signal is attached to pin 11
int  DataIN = 12;                   // The DataIn which comes from the encoder is attached to pin 12


// Definitions for variables

unsigned int reading;

// The program starts here
// **************************
```





```
// The program starts here
// **************************

void setup() {

  Serial.begin(9600);

  // initialize the digital pin as an output.
  pinMode(NCS, OUTPUT);          // Tell the Arduino that the Chip Select is an OUTPUT pin
  pinMode(SSI_CLK, OUTPUT);      // Tell the Arduino that the Serial Clock Signal is an OUTPUT pin
  pinMode(DataIN, INPUT);        // Tell the Arduino that the Serial data IN is an INPUT pin, here
                                 //we will receive the binair values from the encoder

  digitalWrite(SSI_CLK, HIGH);   // Make the Serial Clock Signal High (Timing Characteristics at page 4 from the DATASHEET)

}

//**************************************************************************
// Main program loop
//**************************************************************************

void loop() {

  ReadSSI();                                  // "Call" for the subroutines ReadSSI in the main program loop
   int outVal = map(reading, 724, 980, 42, 132); // Here i scale the "reading" value, from the encoder I get a value
                                              //out from min 724 - max 980 and it presents the angular between 42° and 132°.
   Serial.println(outVal,DEC);                // "Print" out the angular on the serial monitor
   delay(10);                                 // I delay of 10 ms

}

 //**************************************************************************
// Main Loop ends here
// Start of subroutines
//**************************************************************************
```

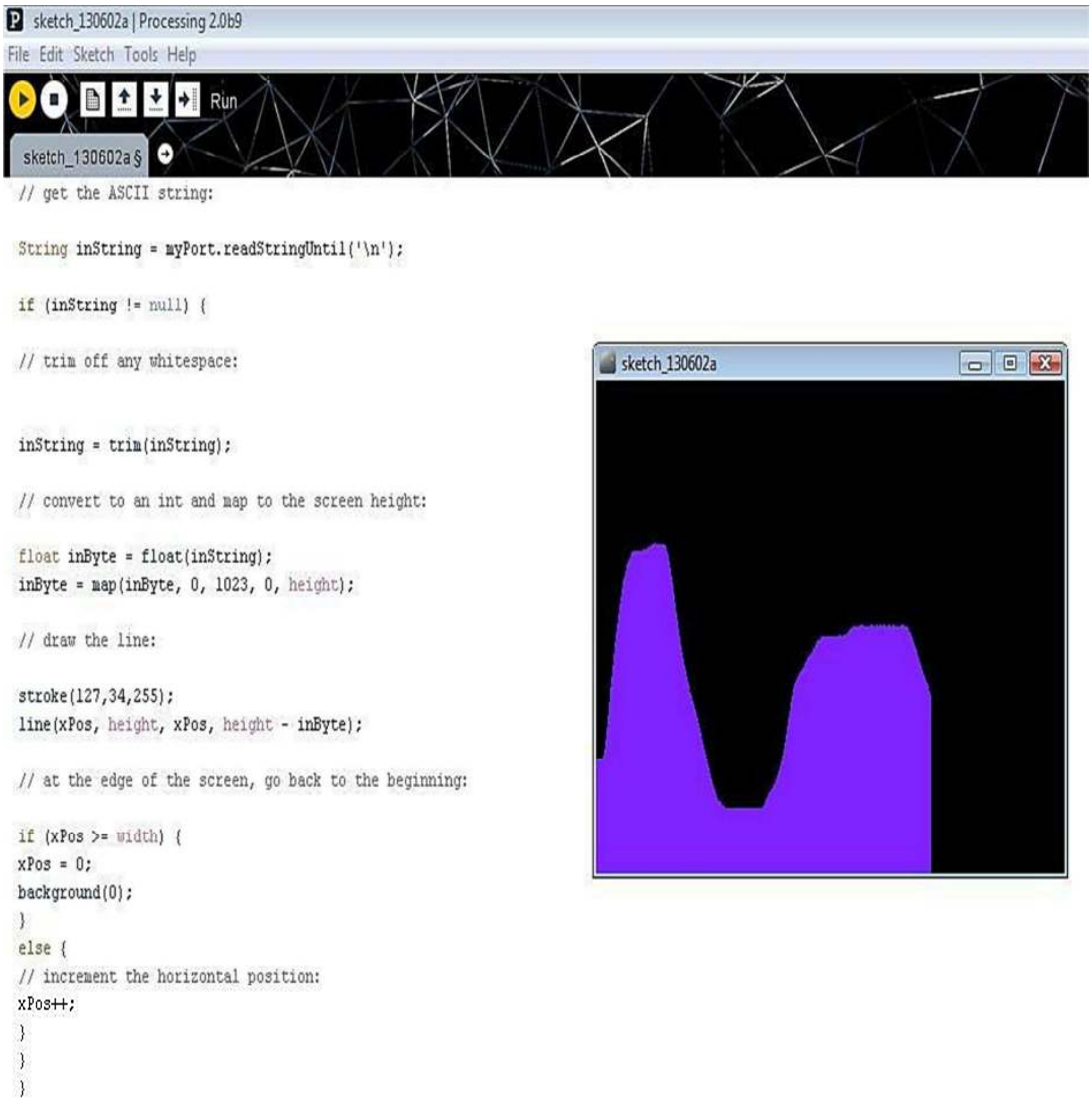

Timing Characteristics
CS
CLK
DO
tCLKFE
TCLK/2
tCS
tCLKFE
1
10
1
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
D9
tDO active
tDO valid
tDO Tristate
Positional Data Bits
Notes:
1. Please refer to Table for Timing Characteristics.
2. For 12 bits version; the Positional Data Bits will start with D11 instead and end at D0.

```
void ReadSSI(void)                  // Create a subroutine
{
int i;                              // "i" will be used to count untill 10, int stores a 16-bit (2-byte) value.
                                    //This yields a range of -32,768 to 32,767.
char Res = 10;                      // "Res" will be used to start from 10 and go down untill 1. char is a data
                                    //type that takes up 1 byte of memory that stores a character value.
unsigned int mask;                  // unsigned also stor 2-bytes but instead of storing negative numbers however they
                                    //only store positive values, yielding a useful range of 0 to 65,535.


reading = 0;
        mask = 0x0200;
digitalWrite(NCS, LOW);             // Like you can see in the time characteristics NCS needs to be made LOW first
delayMicroseconds(1);               // Wait at least 500ns for: tCLK FE = First data shifted to output register
digitalWrite(SSI_CLK, LOW);         // After that I make the CLK signal LOW for the first time

for(i=(Res-1);i>0;i--)              // Creating the for statement which will create the 10 bits signal
{
  digitalWrite(SSI_CLK, HIGH);      // Making the CLK signal HIGH (which will be done 10 times again)
  delayMicroseconds(1);             // Wait at least 500ns for: TCLK/2 = Start of data output
if (digitalRead(DataIN)) reading |= mask;
digitalWrite(SSI_CLK, LOW);         // Make the CLK signal back LOW after receiving the first bit
mask = mask >> 1;

if (i == 1)
          {
          digitalWrite(SSI_CLK, HIGH);      // After receiving the 10th bit, make the CLK signal HIGH
   if (digitalRead(DataIN)) reading |= mask;
          }
}

digitalWrite(NCS, HIGH);            // After receiving the 10th bit and making the CLK back HIGH, also make the
                                    //NCS signal HIGH. After this the program can start over again from above!

}
```

1.5 Processing sketch

1.5.1 Explanation Processing sketch

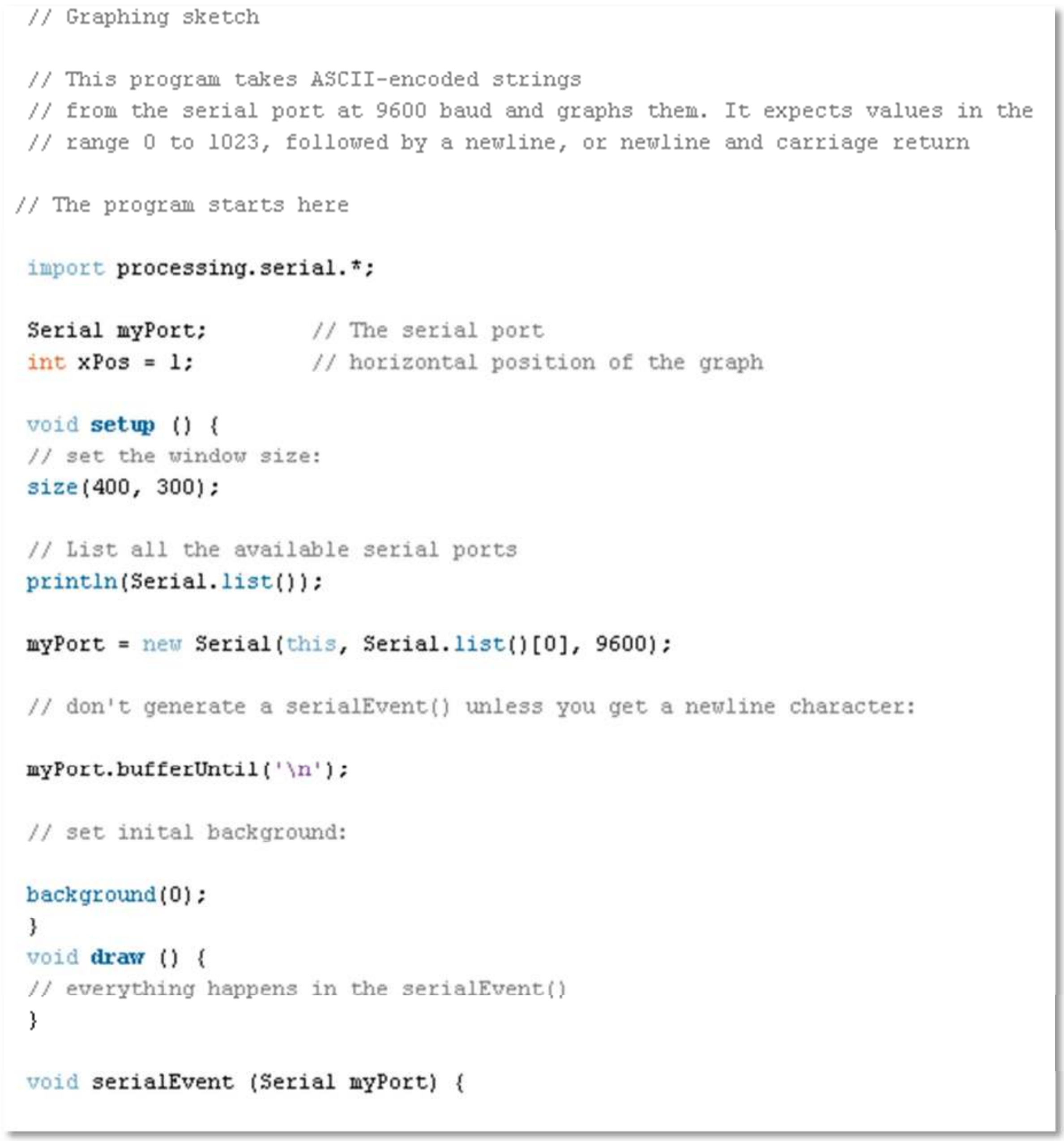
The Processing sketch will generate a graph depending on the bars angular (32º-132) The explanation about the different functions is shown behind the "//" in the code.


sketch_130602a | Processing 2.0b9
File Edit Sketch Tools Help
Run
sketch_130602a §
sketch_130602a

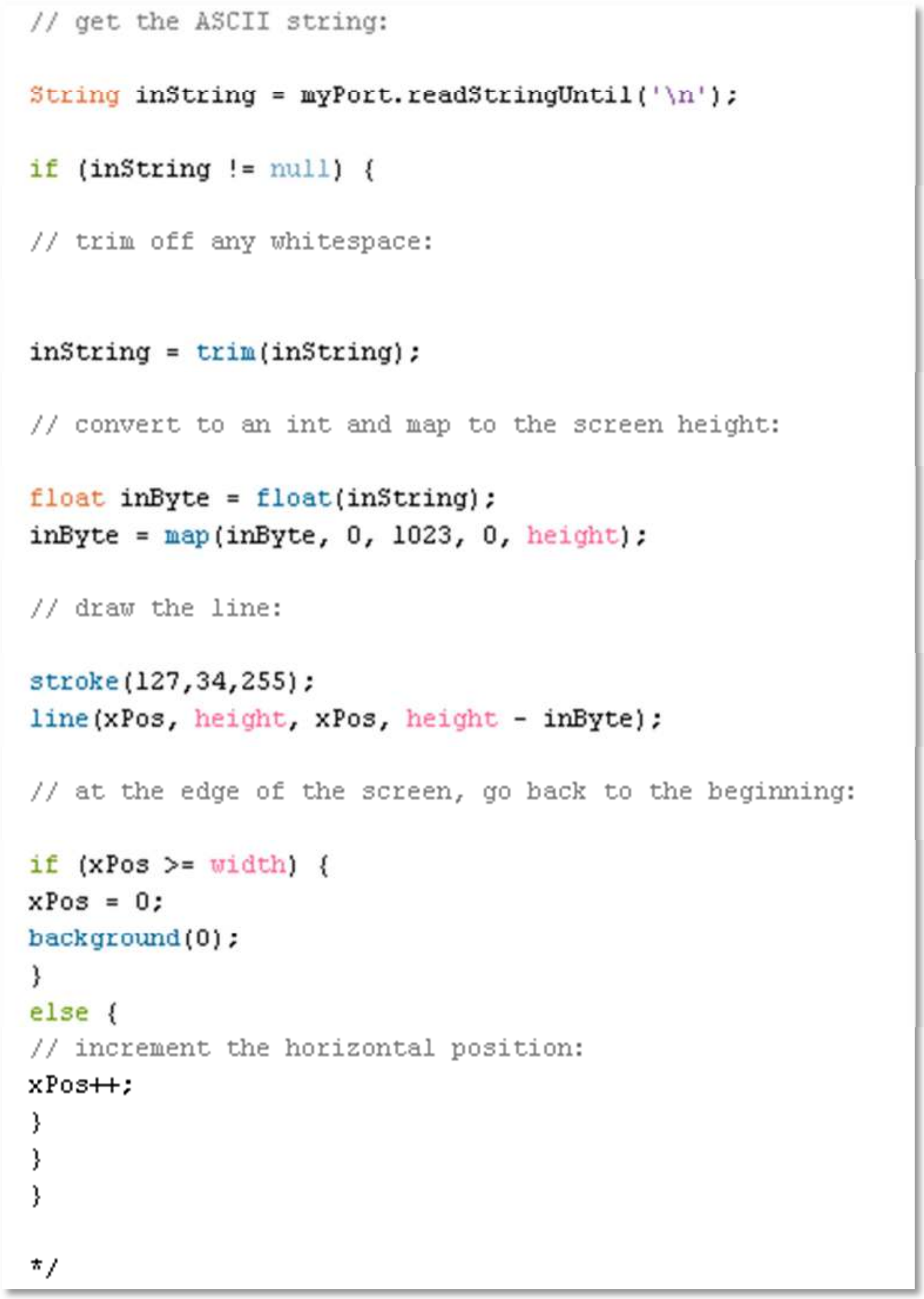
```
// get the ASCII string:

String inString = myPort.readStringUntil('\n');

if (inString != null) {

// trim off any whitespace:


inString = trim(inString);

// convert to an int and map to the screen height:

float inByte = float(inString);
inByte = map(inByte, 0, 1023, 0, height);

// draw the line:

stroke(127,34,255);
line(xPos, height, xPos, height - inByte);

// at the edge of the screen, go back to the beginning:

if (xPos >= width) {
xPos = 0;
background(0);
}
else {
// increment the horizontal position:
xPos++;
}
}
}
```

Testing the Processing program

<u>The main Processing sketch, that will create a graph from the Arduino it's serial port</u>:



```
// Graphing sketch

// This program takes ASCII-encoded strings
// from the serial port at 9600 baud and graphs them. It expects values in the
// range 0 to 1023, followed by a newline, or newline and carriage return

// The program starts here

import processing.serial.*;

Serial myPort;         // The serial port
int xPos = 1;          // horizontal position of the graph

void setup () {
// set the window size:
size(400, 300);

// List all the available serial ports
println(Serial.list());

myPort = new Serial(this, Serial.list()[0], 9600);

// don't generate a serialEvent() unless you get a newline character:

myPort.bufferUntil('\n');

// set inital background:

background(0);
}
void draw () {
// everything happens in the serialEvent()
}

void serialEvent (Serial myPort) {
```



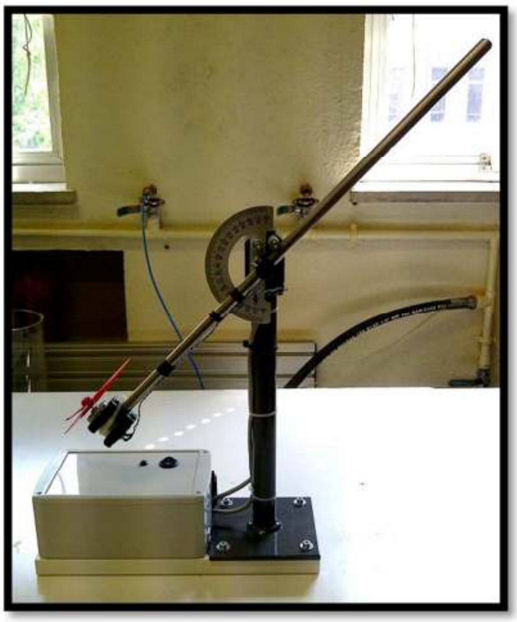
```
// get the ASCII string:

String inString = myPort.readStringUntil('\n');

if (inString != null) {

// trim off any whitespace:


inString = trim(inString);

// convert to an int and map to the screen height:

float inByte = float(inString);
inByte = map(inByte, 0, 1023, 0, height);

// draw the line:

stroke(127,34,255);
line(xPos, height, xPos, height - inByte);

// at the edge of the screen, go back to the beginning:

if (xPos >= width) {
xPos = 0;
background(0);
}
else {
// increment the horizontal position:
xPos++;
}
}
}

*/
```

2. CHAPTER 2: Part II : Position control of a rotating bar with engine and propeller using LabVIEW and Arduino

2.1 Intro

In this chapter I will describe the main program, the plants components, the theoretical part on PID control and the tuning method of Ziegler-Nichols that I used to get my P and I value.

2.2 Plant description



The plant consists of a system with two bars, one totally anchored and one with a degree of freedom around an axis with a DC motor with a propeller attached. The white box contains all the electronics.

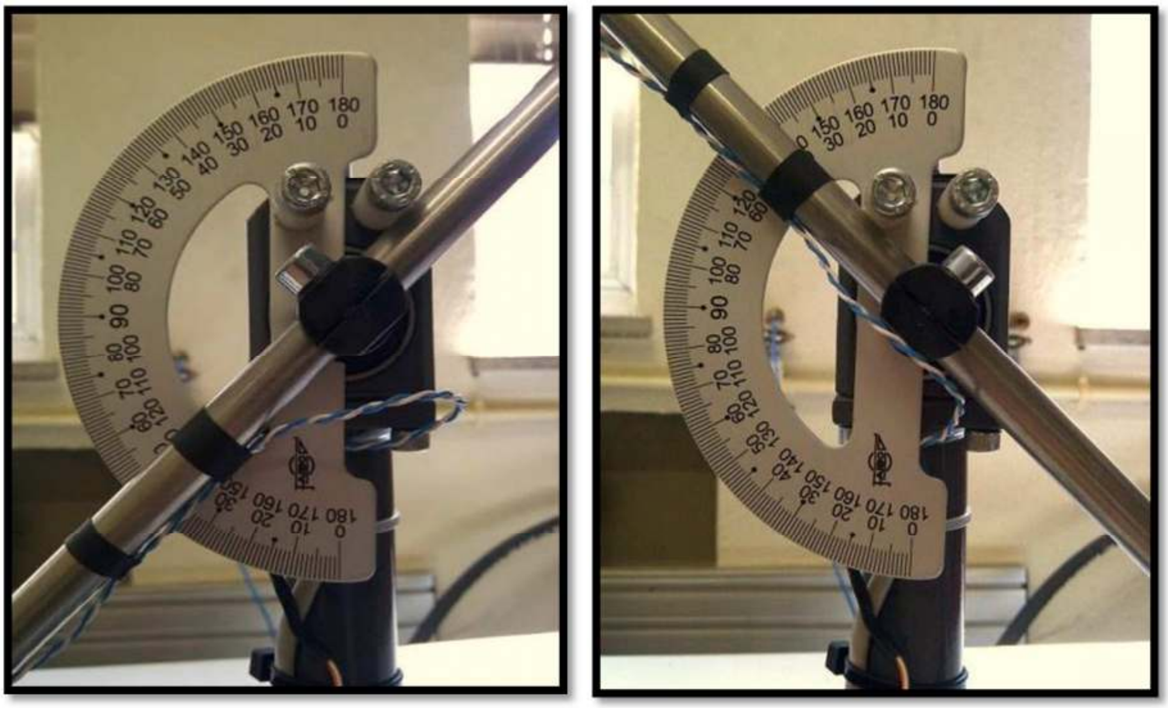
On the rotation shaft from the plant is a potentiometer attached which will give the right position of the bar at every moment. The electric DC motor with propeller attached controls the movement of the bar. The higher the motor voltage the faster the propeller will turn and the bar will change position.

So inside the box we find the **electronic circuit**. Formed by an Arduino Uno microcontroller, an NPN transistor (BD 139) with a radiator which overlaps it to dissipate heat. This transistor will be used to control the speed of the electrical motor by using PWM (PWM is explained in another chapter). Also in the box we have a 230V AC to 24V DC transformer which we need to power the motor. This because the Arduino cannot deliver the amount of voltage and current we need to power it.

There is one “main” wire which contains 5 wires inside, which are 2 from the motor and 3 from the potentiometer. On the outside of the box you can plug in the 230V socket to power the transformer and the USB cable to reprogram the Arduino.

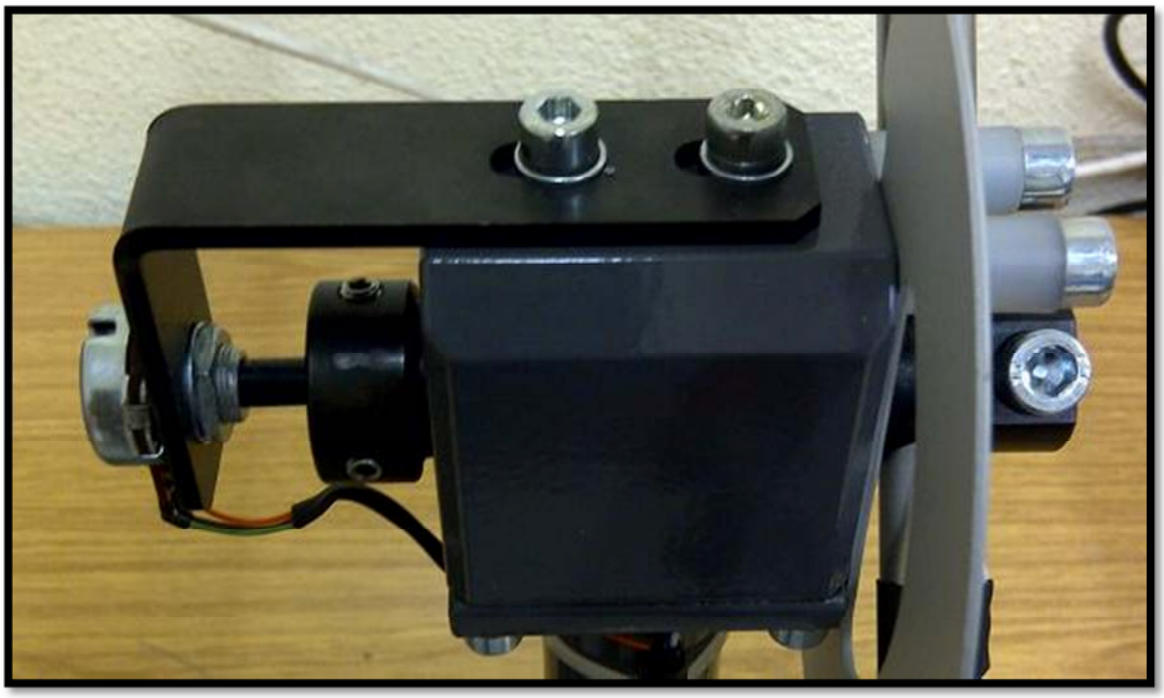
2.3 Basic function

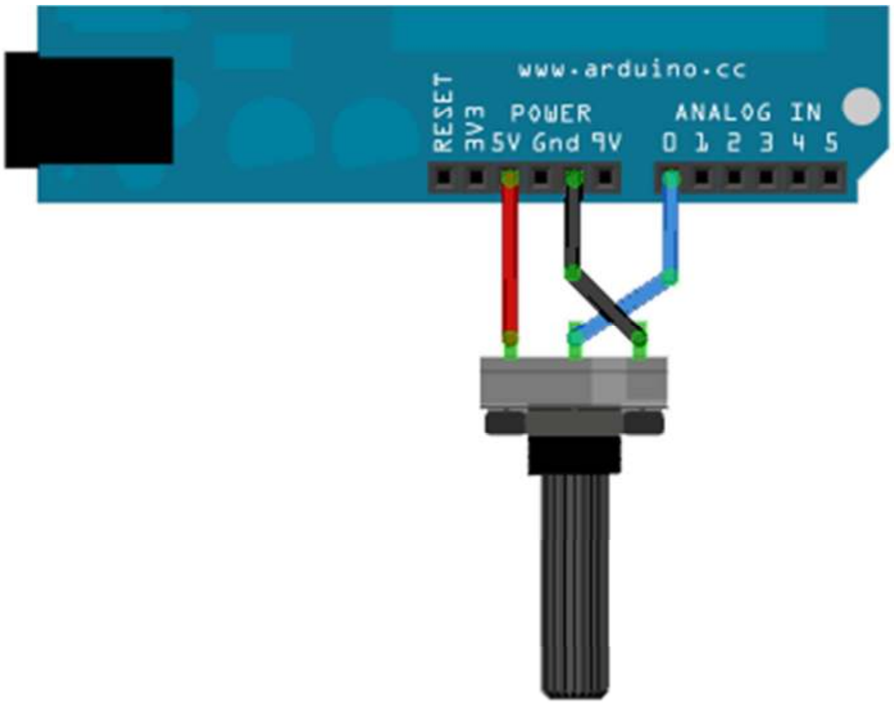
The operation is based on the rotation of the motor and consequently of the propeller, the rotating propeller creates a force that rotates the movable bar (this process will be studied carefully in section mathematical model, to know accurately the forces that influence the plant). The motor turns depending on the voltage received, faster or slower, letting our bar which is attached with a degree of freedom move with a curve at its ends. There is a semicircle attached to the plant, to at all times be able to visualize the positions angle. The mechanic minimum and maximum angular positions are 42° and 135°.



Minimum and Maximum angular

On the rotation shaft is a potentiometer attached which changes its value of resistance depending on the angle of the moving bar. Now we connect **the right pin** to the Arduino ground (**GND**) and **the left pin** to the **5V** pin, we then connect the **center pin** to the **analog input** of the Arduino board. This connection is called "voltage regulator" and what it does is, depending on the position of the potentiometer it will have different voltages, always between the limits set upper limit of 5 volts and 0 volts the bottom.





Potentiometer attached to the shaft

So through the Arduino digital output (PWM) we can move the bar and through an analog input we know the position of the bar. The engine needs a higher potential than the Arduino board, which is 5 Volt. Therefore we need a power circuit, which is explained in depth in the "The electronic circuit".

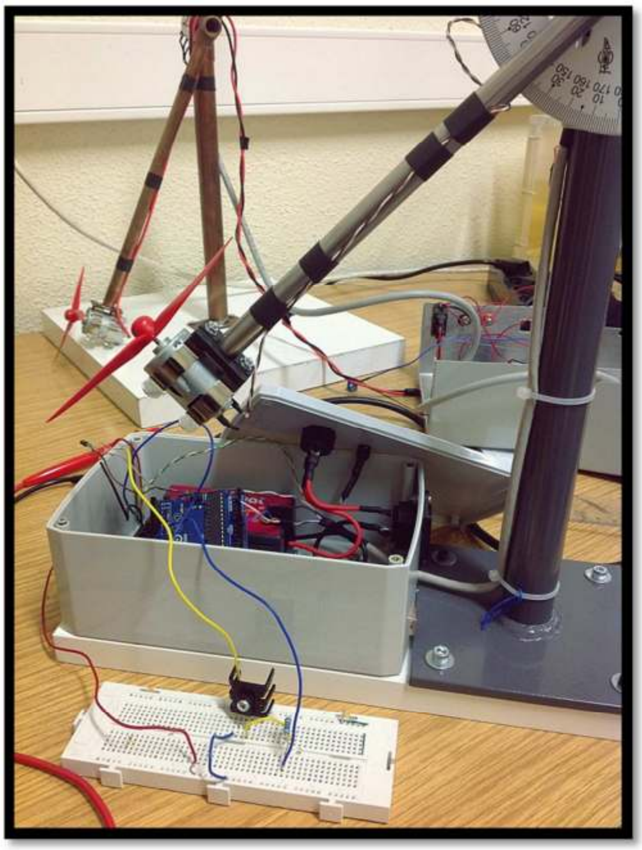
2.4 Electronic circuit

In this section of the thesis I will explain in a detailed way the parts of the electronic circuit. Within this "electronic circuit" there are three sections:

I: ***The controller***: The Arduino Uno microcontroller communicating with the LabVIEW software will control the whole system depending on the set point and the actual process value from the analog input (Pot meter). Doing this by sending out a PWM signal to the power circuit by a digital PWM output.

II: ***The power circuit***: This circuit will deliver the power to the actuator. The transistor will adjust the transformer's 24V depending on the PWM value from the PID controller its output. A PID output value between 0-255 = voltage between 0-24V.

III: ***The actuator***: The electrical DC motor with the propeller attached.

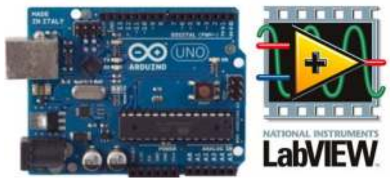


Testing the PWM with the transistor

These three sections are not separated from each other. For example to make the engine run we have to use PWM on the transistor and this is controlled by the PWM output pin from the Arduino and using the LabVIEW program with the PID controller.

2.4.1 The controller

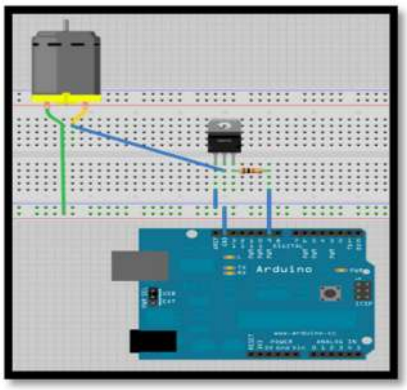
The controller controls the system through the Arduino its internal communicating sketch, input-output signal and the LabVIEW program with the PID controller. The control part uses PWM on the power circuit which allows the motor to turn faster or slower. The Arduino has inputs and outputs for analog and digital type, specifically our output is of a digital type (PWM), as Arduino is powered by a 5 volt signal (via USB) this signal will vary from 0-5 volts. The analog input pin has a 10 bits resolution for the internal converter, it means the Arduino board receives a signal of 10 bits of accuracy and it is able to send a signal of 8 bits of precision being always from 0 to 5 volts.



2.4.2 The power circuit

The goal of the power circuit is powering the DC motor, this because the Arduino board is powered via USB with a voltage of 5V and the motor needs a higher potential, therefore an external source is needed (230VAC-24VDC transformer).

To allow the engine to operate in the range of its required potential, I use a power supply of 24 volts ,which is the maximum for the motor. This source is connected to an ordinary plug we have at home (230V AC) and the output will always be 24VDC. As the output of our power supply is constant 24VDC we need a "device" that allows us to vary the amount of voltage sent to the motor, therefore I use the bipolar transistor. There are two requirements for the chosen transistor: the collector-emitter voltage must be greater than 24 volts and the base current has to support more than 40 mA, which is the current leaving from the Arduino because of the base resistor. It's because of these two datasheet values that I use the chosen transistor 139 which has a collector emitter voltage up to 100 volts and base current of 0.5 Amperes.



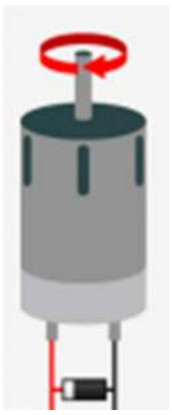
Power circuit

2.4.3 The actuator

The third part of the circuit is the actuator, the motor, this part is not related to an electrical circuit itself, but it depends on the power deliverer and the controller.

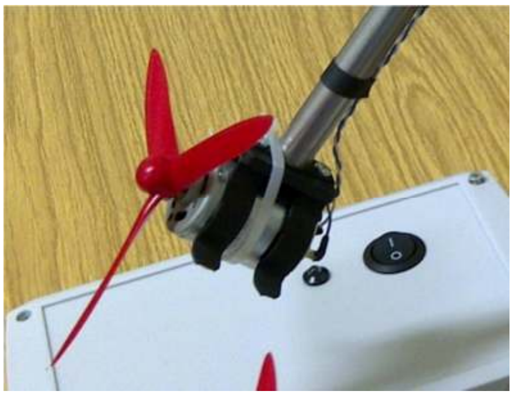
I have designed a control circuit and a power deliverer for it. The higher the voltage the more rapidly the engine will turn and therefore the propeller attached to the external part of the motor will exercise more power to lift the bar. We must remember that the maximum potential that we let go through the ends of the diode is 24 volts, which matches with our max. voltage.

When mounting the motor we must place a diode in antiparallel, if we do not do this, we run the risk of damaging the transistor we have on the power circuit.



Motor with an antiparallel diode

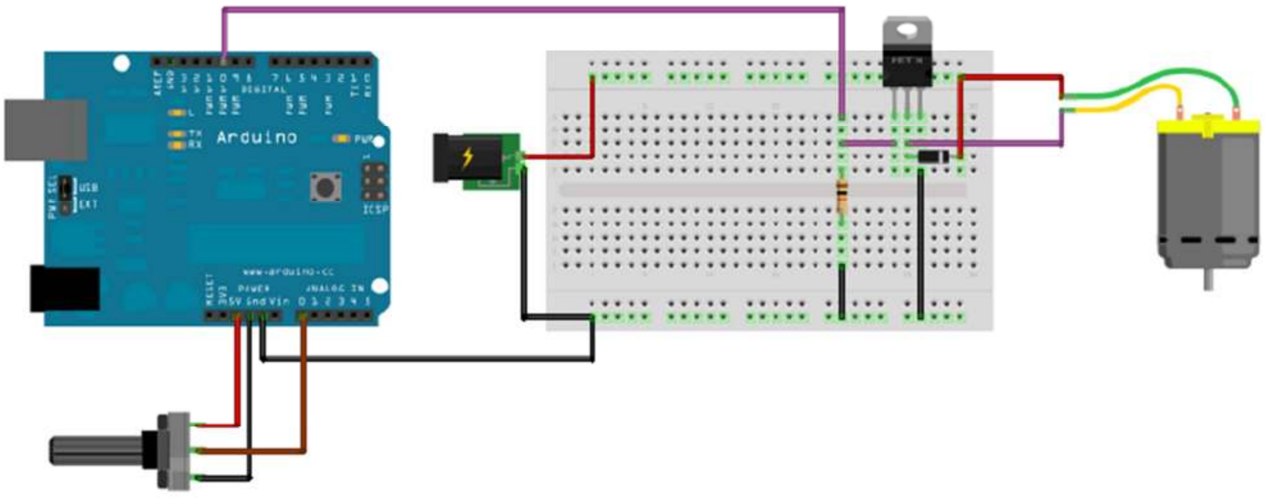
As you can see in the picture the cathode of diode is connected to the positive of the engine (the engine does not have a positive side and a negative but to feed this way the engine rotates in the direction we want) and the anode is connected to the negative side.

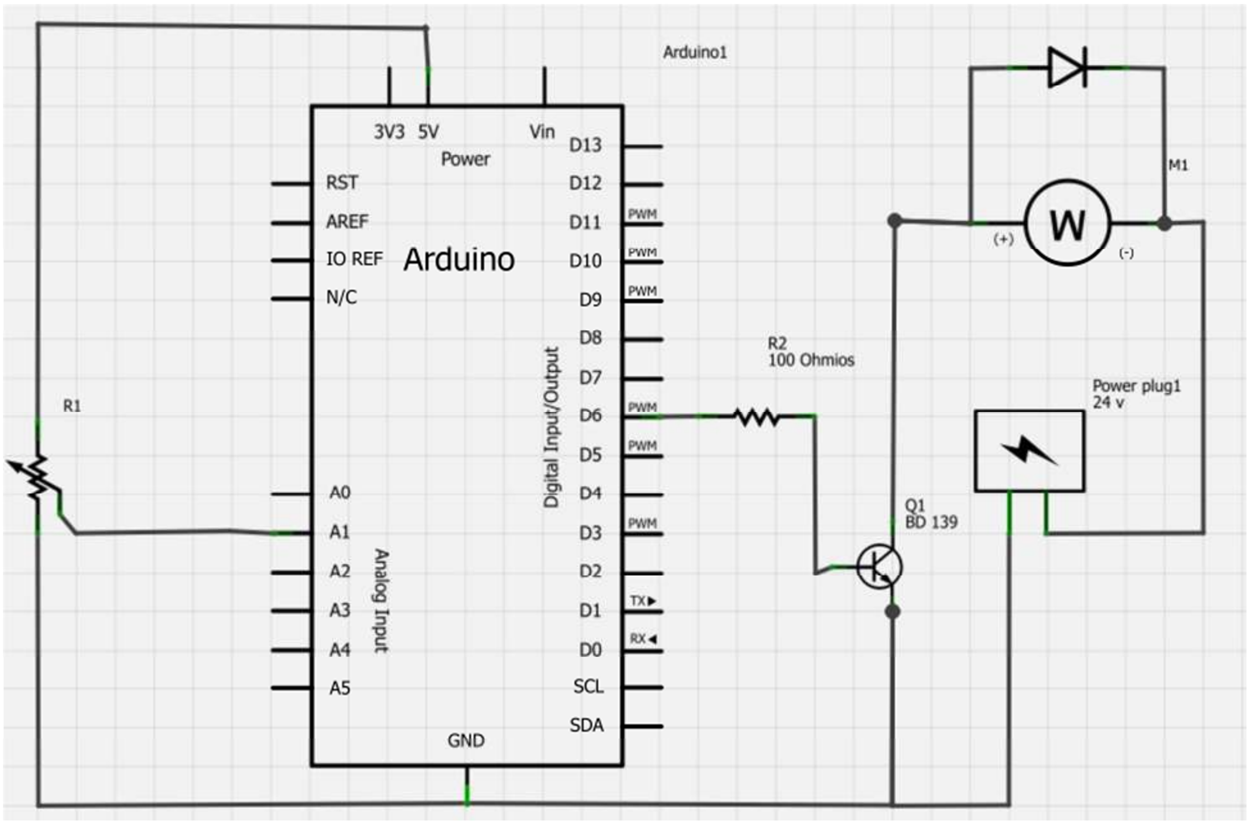


Motor + propeller

2.4.4 Total electronic circuit

So the circuit must be powered in two ways, we need to plug in the power into a standard 230V outlet for the transformer to power the DC motor and we need the Arduino board to insert by USB cable to the computer to feed the board and to program it.





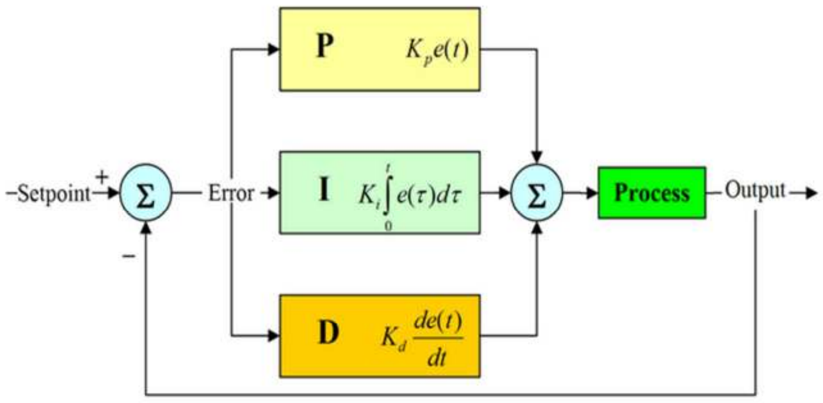
Electrical scheme

2.5 Theoretical part

2.5.1 Intro

In this theoretical part I will explain the different P, I and the D response on a system, open loop, closed loop, the two Ziegler-Nichols tuning methods , the orders in systems and the mathematical model of the system. For the controller I designed I didn't use the D-action. This because the system works fine using a PI-controller.

2.5.2 PID



Proportional-Integral-Derivative (PID) control is the most common control algorithm used in industry and has been universally accepted in industrial control. The popularity of PID controllers can be attributed partly to their robust performance in a wide range of operating conditions and partly to their functional simplicity, which allows engineers to operate them in a simple, straightforward manner.

As the name suggests, PID algorithm consists of three basic coefficients; **proportional**, **integral** and **derivative** which are varied to get optimal response.

2.5.3 Control System:

The basic idea behind a PID controller is to read a sensor, then compute the desired actuator output by calculating proportional, integral, and derivative responses and summing those three components to compute the output. Before I start to define the parameters of a PID controller, I will show what a closed loop system is and some of the terminologies associated with it.

2.5.4 P action

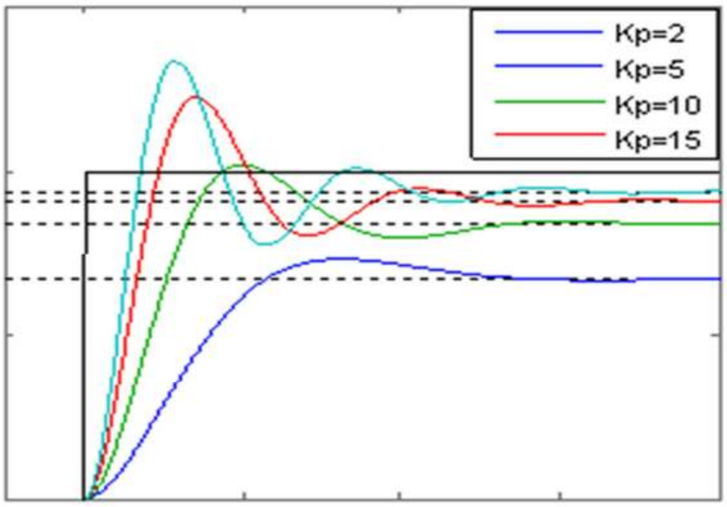
The proportional component depends only on the **difference between the set point and the process variable**. This difference is referred to as the **Error term**. The *proportional gain (K_c)* determines the ratio of output response to the error signal. For instance, if the error term has a magnitude of 10, a proportional gain of 5 would produce a proportional response of 50. In general, increasing the proportional gain will increase the speed of the control system response. However, if the proportional gain is too large, the process variable will begin to oscillate. If K_c is increased further, the oscillations will become larger and the system will become unstable and may even oscillate out of control.

The goal of this action is that after having set the steady-state error it will be zero relative to a reference.

The output we get from it is proportional to the error being this $u_{(t)} = K_p \cdot e_{(t)}$therefore the transfer function of the proportional action is nothing more than an adjustable gain.

$$C_p(s) = K_p$$

This action does not correct the error in the steady state. The following figure shows the operation of a P controller.



Answer of a system with different K_p

As you can see in the figure the **steady state error decreases as the constant increases**, the speed of this is slower but also the overshoot and oscillations increase taking a longer time to oscillate.

2.5.5 I action

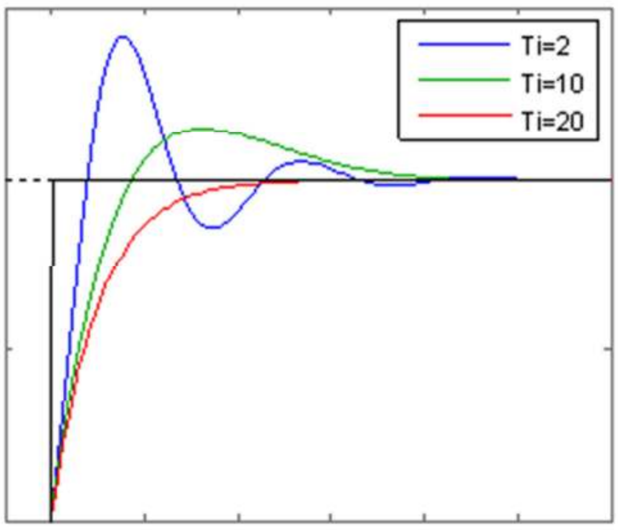
The integral component **sums the error term over time**. The result is that even a small error term will cause the integral component to increase slowly. The integral response will continually increase over time unless the error is zero, so the effect is to drive the Steady-State error to zero. Steady-State error is the final difference between the process variable and set point. A phenomenon called integral windup results when integral action saturates a controller without the controller driving the error signal toward zero.

The output of this controller is proportional to the accumulated error, the response will be much slower. The fdt of the controller output and this are:

$$u_{(t)} = K_i \int_0^t e_{(t)}\, dt \qquad C_i(s) = \frac{K_i}{s}$$

The purpose of this action is the output to match the reference steady state, this can change without having to change the Ki unlike the proportional control.

The characteristics of the integral action can be seen in the figure below, where we represent the PI control in which the portion is varied.



Answer of a system for different values of Ti

Being T$_1$=$\frac{1}{k_i}$, therefore **the bigger the constant the higher the speed of the system**, but also the bigger will be its damping, as it could destabilize if it is too big. As we can see the steady error tends to 0.

2.5.6 D action

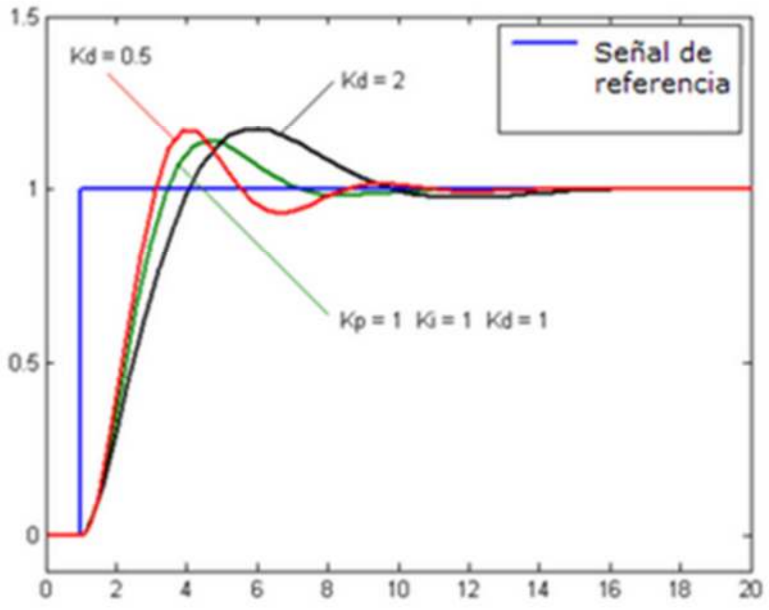
D-action is **NOT USED** !

The derivative component causes the output to decrease if the process variable is increasing rapidly. The derivative response is proportional to the rate of change of the process variable. Increasing the *derivative time* *(T_d)* parameter will cause the control system to react more strongly to changes in the error term and will increase the speed of the overall control system response. Most practical control systems use very small derivative time (T_d), because the Derivative Response is highly sensitive to noise in the process variable signal. If the sensor feedback signal is noisy or if the control loop rate is too slow, the derivative response can make the control system unstable

This action works when there is a change in absolute value of the error. Therefore it is not going to be used alone, as it only corrects single errors in the transitional stage. It's a predictable action therefore a quick action.

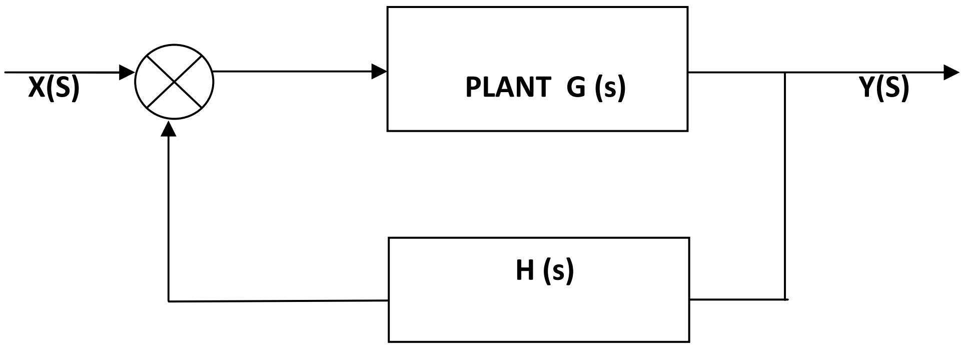
Its aim is to correct the error signal before it becomes too big. The prediction is made by extrapolating the control error in the direction of the tangent to the corresponding curve, as shown in the figure.

The disadvantage is that it enlarges the noise signals and it could cause saturation in the controller. It can be used in systems with significant delay, because it allows rapid impact of the variable after introducing a disturbance in the process. The characteristics of the integral action can be seen in the figure below, in which we represent the PI control varying the portion.

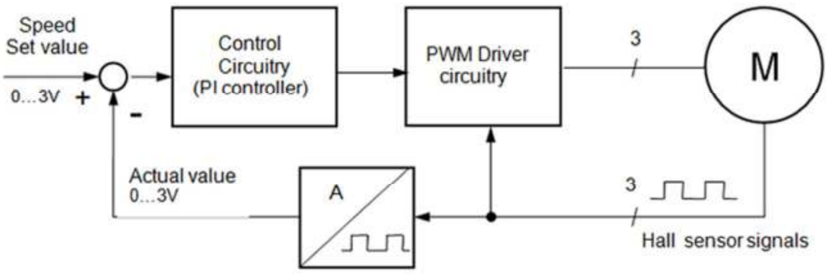


Answer of a system for different values of Ti

2.5.7 Closed loop



A practical example for a ***closed loop system***:



Closed loop speed control:

Hall sensors are used to measure the actual speed of the rotor. This signal is the actual value input for a Control Circuitry, usually based on PI control algorithm.

The voltage level for motor is adjusted depending on the difference between the actual value and the set value by implementing PWM.

These circuits are different because the output depends not only on the input, but on both the input and the output. These circuits are ideal for the regulation because we can compare the output with the input. We must add that the upper line will be not only the plant but it may have many other functions such as controllers etc.., like the line below, but from here we will consider the above line as G (s) and below as H (s).

At this point we can state that the transfer function is simply the mathematical function that relates the ratio between output and input:

Transfer function: F (s) = Y (s) / X (s)

The transfer function obviously consists of **a numerator** where the zeros of the system (are the values that make the numerator zero), and **a denominator** where **the poles** of the system (are the values that make the denominator zero).

The way of classifying the different systems is through the degree of the polynomial that has the characteristic equation (the denominator of the transfer function), this approach obtains three groups of **first order** systems (polynomial of degree 1), **second systems** order (polynomial of degree 2), and **higher order** systems (polynomial of degree 3 or higher).

2.5.8 Tuning rules Ziegler-Nichols

The three **PID** controller actions are not always used, the proportional action normally appears accompanied only with the integral action or with the derivative one. This way they conform the **PI** controller and the **PD** one. These together with the PID controller are the one we **most commonly find**.

For its proper functioning, the tuning must be correct. This can be done in several ways, today there are programs (auto tuning) that will automatically adjust it, but it can also be done **manually**.

To do this there are rules that facilitate this work. We will see the rules proposed by Ziegler-Nichols:

The process of setting the optimal gains for P, I and D to get an ideal response from a control system is called ***tuning***. There are different methods of tuning, of which the "guess and check" method and the "Ziegler Nichols" method.

The gains of a PID controller can be obtained by trial and error method. Once an engineer understands the significance of each gain parameter, this method becomes relatively easy. In this method, the I and D terms are set to zero first and the proportional gain is increased until the output of the loop oscillates.

As one increases the proportional gain, the system becomes faster, but care must be taken to not make the system unstable. Once P has been set to obtain a desired fast response, the integral term is increased to stop the oscillations. **The integral term reduces the steady state error**, ***but increases overshoot***.

Some amount of overshoot is always necessary for a fast system so that it could respond to changes immediately. The integral term is tweaked to achieve a minimal steady state error. Once the P and I have been set to get the desired fast control system with minimal steady state error, the derivative term is increased until the loop is acceptably quick to its set point. Increasing derivative term decreases overshoot and yields higher gain with stability but would cause the system to be highly sensitive to noise.

The Ziegler-Nichols method is another popular method of tuning a PID controller. It is very similar to the trial and error method wherein I and D are set to zero and P is increased until the loop starts to oscillate. Once oscillation starts, the critical gain K_c and the period of oscillations P_c are noted. The P, I and D are then adjusted as per the tabular column shown below.

Control	P	Ti	Td
P	0.5Kc	-	-
PI	0.45Kc	Pc/1.2	-
PID	0.60Kc	0.5Pc	Pc/8

Ziegler-Nichols tuning, using the oscillation method

PID controller tuning rules for Ziegler-Nichols

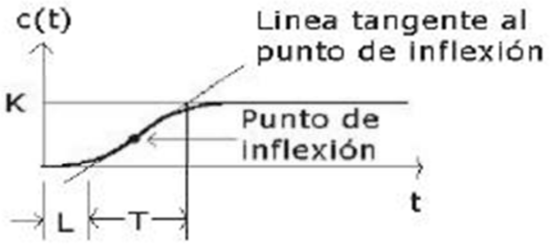
Ziegler and Nichols proposed some rules to determine the values of proportional gain K_p, integral time T_i and derivative time T_d.

There are **two methods** called rules Ziegler-Nichols tuning. In both it is requested to obtain a 25% maximum overshoot in the response step.

2.5.9 First method (not used in this final project)

The plant response to a unit step input is obtained experimentally. If the plant does not contain integrators or complex conjugate dominant poles, the unit step response curve can be in the form of S (if the response exhibits an S-shaped curve, this method is not applicable). Such step-response curves is generated experimentally or from a dynamic simulation of the plant.

The S-shaped curve is characterized by two parameters: the delay time L and the time constant T. The delay time and the time constant are determined by drawing a tangent line at the inflection point of the S-shaped curve and determining the intersection of this tangent with the time axis and the line c (t) = K, as be seen below.



Calculation of delay time and the time constant

In this case, the transfer function C (s) / U (s) is approximated through a first order system with a transport delay as it follows:

$$\frac{C(s)}{U(s)} = \frac{Ke^{-Ls}}{Ts+1}$$

Ziegler and Nichols established the values of K_p, T_i y T_d agreeing with the following table.

Type of control	K_p	T_i	T_d
P	$\frac{T}{L}$	∞	0
PI	$0.9\frac{T}{L}$	$\frac{T}{0.3}$	0
PID	$1.2\frac{T}{L}$	2L	0.5L

2.5.10 Second method (USED)

In the second method, we first set $T_i = \infty$ and $T_d = 0$. Thanks to the use of only the proportional control action, K_p increases from 0 to a critical value where the output K_{cr} first exhibits sustained oscillations (if the output has not sustained oscillations for any value you can take K_p, this method does not apply). Therefore, the critical gain K_{cr} and P_{cr} corresponding period are determined experimentally. Ziegler-Nichols suggested to establish the values of the parameters K_p, T_i and T_d according to the formula shown in the table.

Type of control	K_p	T_i	T_d
P	0.5 K_{cr}	∞	0
PI	0.45 K_{cr}	$\frac{1}{1.2}P_{cr}$	0
PID	0.6 K_{cr}	$0.5P_{cr}$	$0.125P_{cr}$

2.5.11 Kind of system

In the field of automatic control systems the plants are studied in a theoretical way because the plants are characterized by mathematical equations, in order to use them in a simpler way especially for the possibility of using computers to solve problems.

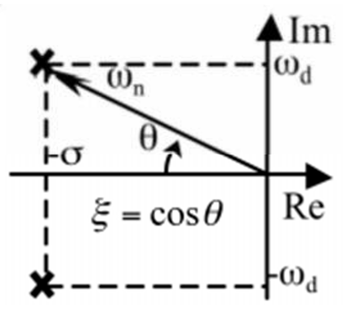
Second order

We have a characterized plant with a mathematical function:

The second order systems have a function of transfer which has this form:

$$F(s) = \frac{K{\omega_n}^2}{s^2 + 2\xi\omega_n s + {\omega_n}^2}$$

Where "ω_n " is the natural frequency, "ξ" is the damp-coefficient and "K" keeps on being the system's gain. The natural frequency and the damp-coefficient indicate the poles' position, being it the arcsine of ξ the angle which form the poles and ω_n their wideness, as for example:



Poles of a second order system

Second-order systems are divided into **four groups** depending on the value of the damping coefficient, these are called:

*** Underdamped system**

*** Overdamped system**

*** Critically damped system**

*** Critically undamped system**

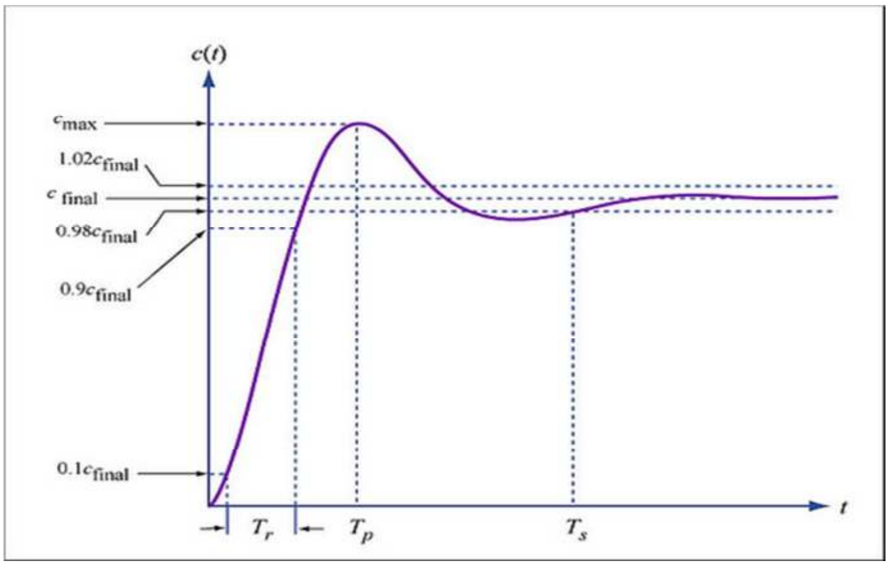
For the second-order systems we specify graphic **characteristics** in order to establish better their characteristics, these are:

• **Rise time** (t_r): rise time is the time taken by a signal to change from a specified low value (10%) to a specified high value (90%).

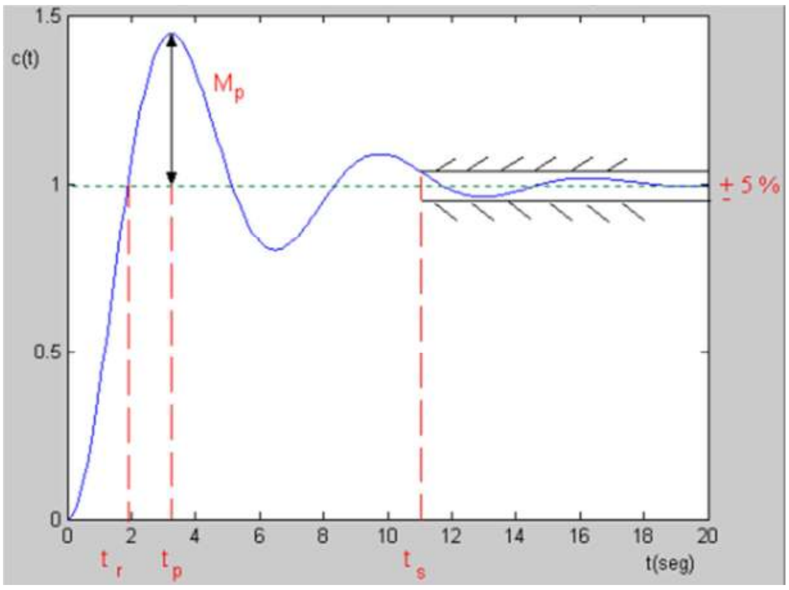
• **Peak time** (t_p): is the time it takes the system response to reach the first peak or maximum value.

• **Overshoot** (M_p): is the percentage value of the maximum or peak value with respect to the final value.

• **Settling Time or stabilization** (t_s): is the time required for the system response is in a band of plus or minus 5% (also frequently used 2%).



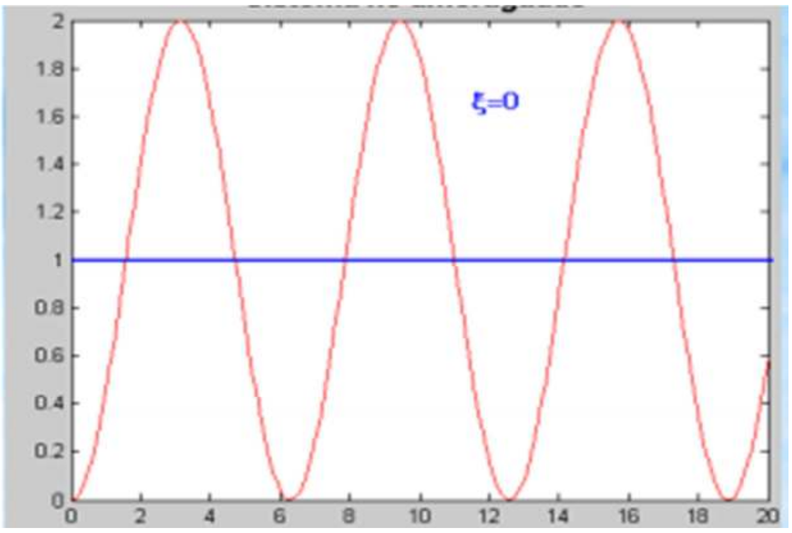
An underdamped 2nd order system



Specifications in the time domain

Not damped movement *(ξ=0)*

This movement occurs when the damping coefficient is zero. It's a bad movement because, as its name suggests, does not dampen the signal. Your answers in the system have the following graphics:

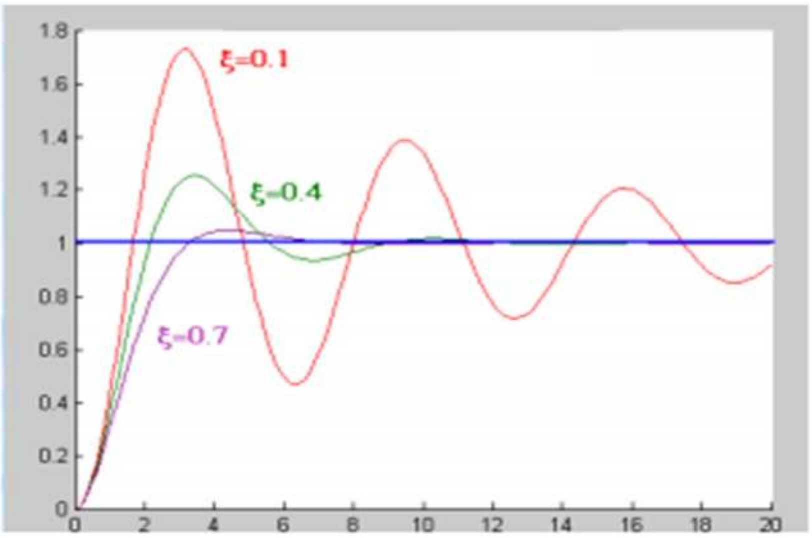


Not damped movement

Underdamped movement $(0 < \xi < 1)$

When the damping ratio is bigger than zero and less than one it is said to be an underdamped motion. This type of movement is one of the most effective because it is usually fast but has an overshoot that could generate a problem.

A graph of the response of this type of motion is:

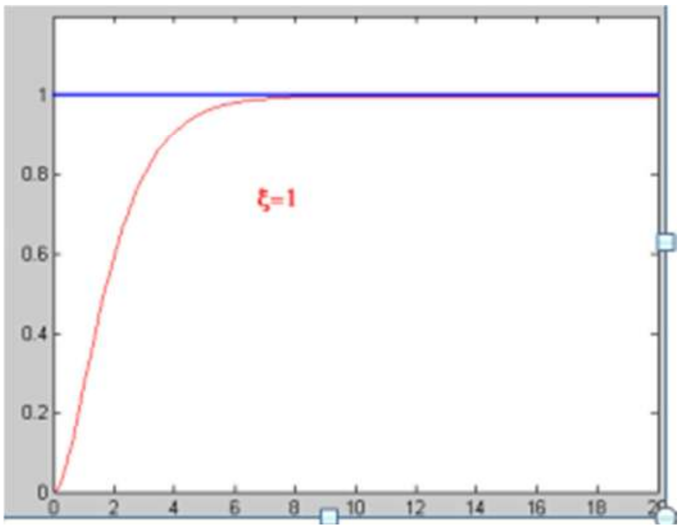


Underdamped system

As shown in the graph when the damping coefficient is close to zero the overshoot is bigger, this overshoot is descending as we approach the unit.

Critically damped movement *($\xi = 1$)*

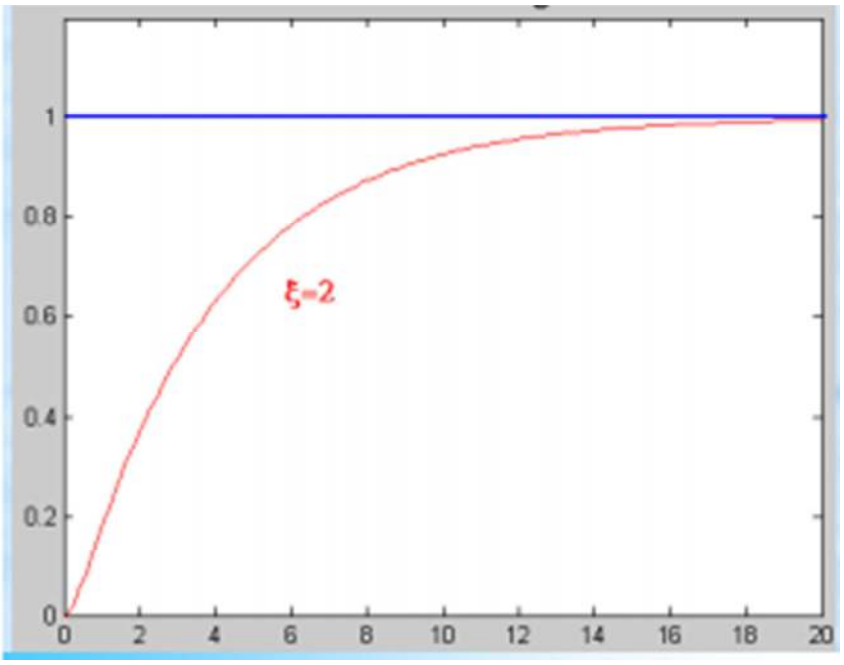
When the damping ratio is exactly equal to 1 it is said to be a critically damped motion



Critical system

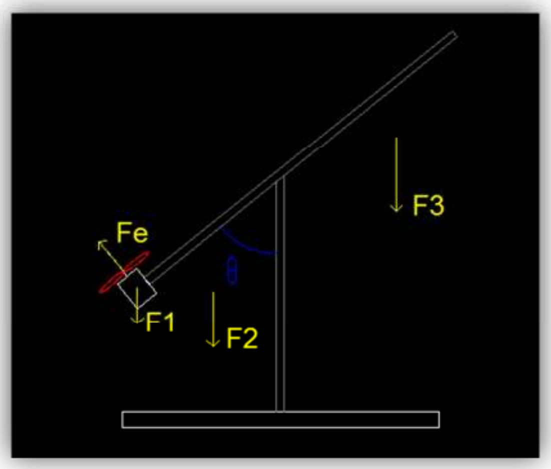
Overdamped movement *($\xi > 1$)*

If the damping coefficient is larger than the unit it is said to be a movement overdamped



Overdamped system

2.5.12 Mathematical model



In order to get the mathematical model of the system we must have a complete knowledge about it and the physical laws that affect it. Our system may be defined as two anchored bars, one totally fixed (the vertical permanent one) and the other one with the motor attached which has a degree of freedom that performs with rotational movement. The center of rotation is the point where both bars are anchored.

I am now going to study the system for the mathematic model.

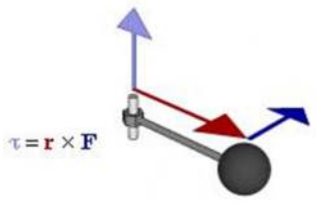
Analysis of the movement

The movement made by the moving bar with the motor attached makes the **rigid-body rotation** around a fixed axis.

In physics, a **rigid-body** means the distance between any two given points of a rigid body remains constant in time regardless of external forces exerted on it. The rotation (motion) which changes the bar its angular is the torque, another words for torque is moment of force. The torque is the tendency of a force to rotate an object about an axis.

The torque is a product of the distance between the applied force (r) and the applied force itself (F).

$$\tau = r \times F$$



T is the torque vector and the magnitude of the torque.

r is the displacement vector (a vector from the point from which torque is measured to the point where force is applied), and **r** is the length (or magnitude) of the lever arm vector.

F is the force vector and the magnitude of the force

× denotes the cross product

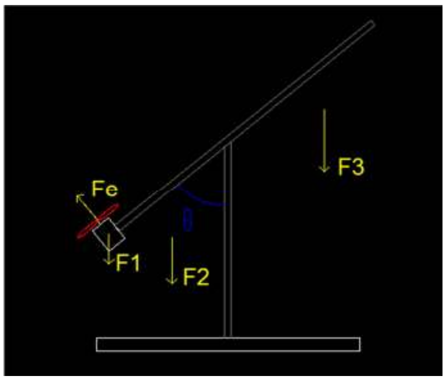
Moment of inertia involved:

The torque is related to the moment of inertia, so another important term in turning movements, as every rotating element has an inertia moment. The relationship **formula** is:

$$\sum \tau = I \propto$$

Where $\propto$ is the angular acceleration on the body.

Starting from these formulas I can obtain the equations of the system. First I am going to calculate the inertia moment of the bar and the motor and then the torque. In the picture we can see a summary of the forces exposed in our system.



The system's forces

The systems forces:

Fe is the force that provides the rotation of the motor and the charge to change the bar position.

All the other forces are due to gravity:

F1 is the total gravitational of the mass of the motor

F2 is the total gravitational force for the left side of the bar

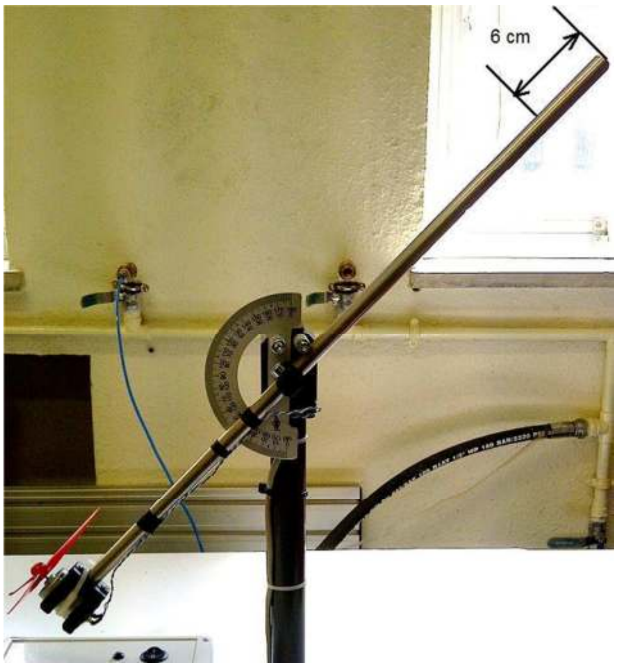
F3 is the total gravitational force for the right side of the bar

Force of gravity

The force of gravity acts on the bar and the engine, therefore we calculate them separately.

If the center of gravity of the bar would be in the center of gravity, the total torque which produces the gravity on the rod would be zero because the torques of one side of the rotation axis would compensate the other one. In this case the center of gravity would be displaced **6 cm** therefore if the gravity will produce a torque it will be:

$$\sum \tau_b = \int_{0.22}^{0.28} g\frac{M}{L}xdx = \int_{0.22}^{0.28} 9.81\frac{0.454}{0.5}xsin\theta(t)dx = 0.1335sin\theta(t)$$



Here you can see that the ***right side*** *is* ***6 cm longer***

In order to calculate the torque of the motor and the propeller, we assume that the whole set is a point mass located in the midpoint of the motor. Therefore, the torque of this set is:

$\tau_m = r\ \times F_g = -0.245 \cdot 9.81 \cdot 0.076 \cdot sin\theta(t) = -0.1825\ sin\theta(t)\ Fg$

Movement of inertia

is a property of rotating bodies that defines its resistance to a change in angular velocity about an axis of rotation. It is how rotation of a body is affected by Newton's law of inertia, which states "Every body perseveres in its state of being at rest or of moving uniformly straight forward except insofar as it is compelled to change its state by forces impressed."

The moment of inertia only depends on the body geometry and position of the rotational axis but it does not depend on the forces involved in movement.

Physically if we take a particle system and a random axis, the moment of inertia is defined as the sum of the products of the masses of the particles by the square of the distance *r* of each particle to said axis. Mathematically it is expressed as:

$$I = \sum m_i \cdot r_i^2$$

For a continue mass body it is generalized:

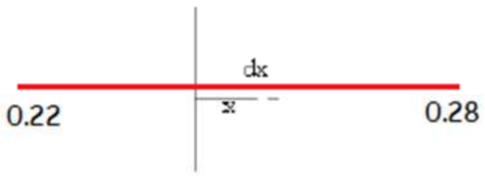
$$I = \int r^2 dm$$

Now I will calculate the inertia moment of the bar and the propeller motor.

Inertia moment of the bar

Now I apply the inertia formula to the inertia moment of our bar, which has a mass M = 454 g and a length L = 50 cm = 0.5 m

In the following picture is a scheme of our bar which shows that the right side is 6 cm longer then the left side.



The bar's dividing

The *dm* mass of the element of longitude of the bar is between x and x+dx and is:

$$dm = \frac{M}{L} dx$$

Therefore the inertia moment is:

$$I_b = \int_{-0.22}^{0.28} \frac{0.454}{0.5} x^2 dx = 9.867 \cdot 10^{-3} kg/\, m^2$$

Inertia moment of the propeller

When calculating the moment of inertia of the propeller engine I will take it as if all its mass was concentrated at a point. To calculate it I use the point mass formula:

$$I = \sum m_i \cdot r_i^2$$

The set has a mass m = 0.76 and is at a distance r = 0.245, therefore the moment of inertia will be:

$I = 0.076 * 0.245^2 = 4.562 \cdot 10^{-3}\ \mathrm{kg/\ m^2}$

Inertia moment in total

It is the sum of the previous two elements:

$I_T = 1.443 \cdot 10^{-2}\ \mathrm{kg/\ m^2}$

Transferfunction plant

In the transfer function the input will be the driving force from the motor and the output is the angle of the moving bar. Therefore the transfer function of the form remain $\frac{\theta}{F_e}$.

We will obtain the equation of our system with the help of this formula. But like any system the system has a friction so you have to attach this term. Therefore the formula remains:

$$\sum \tau - B\dot{\theta} = I \propto$$

As we already have the torques of the calculated system and the moments of inertia, we only have to replace $\propto$ which is the angular acceleration thus it can be put as $\frac{d^2\theta}{dt}$.

$$\tau_{b+}\tau_{m+}\tau_e - B\frac{d\theta}{dt} = I\frac{d^2\theta}{dt}$$

$$0.1335sin\theta(t) - 0.1825sin\theta(t) + 0.245F_g - B\frac{d\theta}{dt} = 0.01443\frac{d^2\theta}{dt}$$

$$0.245F_g = 0.049sin\theta(t) + B\dot{\theta} + 0.01443\ddot{\theta}$$

If we find derivatives in our equation we will use the Laplace transform to obtain the transfer function $\frac{\theta}{F_e}$. Since the equation is nonlinear, we will linearize it using the Taylor series about a point of equilibrium ($\dot{\theta}_0 = \ddot{\theta}_0 = 0, \theta = 45°, F_g$=0.049sin(45)/0.245=0.141N)

$$f = 0.245F_g - 0.049sin\theta(t) - B\dot{\theta} - 0.01443\ddot{\theta} = 0$$

$$\Delta f = \frac{\partial f}{\partial\ddot{\theta}}\Delta\ddot{\theta} + \frac{\partial f}{\partial\dot{\theta}}\Delta\dot{\theta} + \frac{\partial f}{\partial\theta}\Delta\theta + \frac{\partial f}{\partial F_g}\Delta F_g = 0$$

$$\Delta f = 0.245\Delta F_g - 0.049cos45°\Delta\theta - B\Delta\dot{\theta} - 0.01443\Delta\ddot{\theta}$$

The linearized equation will be:

$$0.245\Delta F_g = 0.01443\Delta\ddot{\theta} + B\Delta\dot{\theta} + 0.0346\Delta\theta$$

Now we can use the transformed of Laplace:

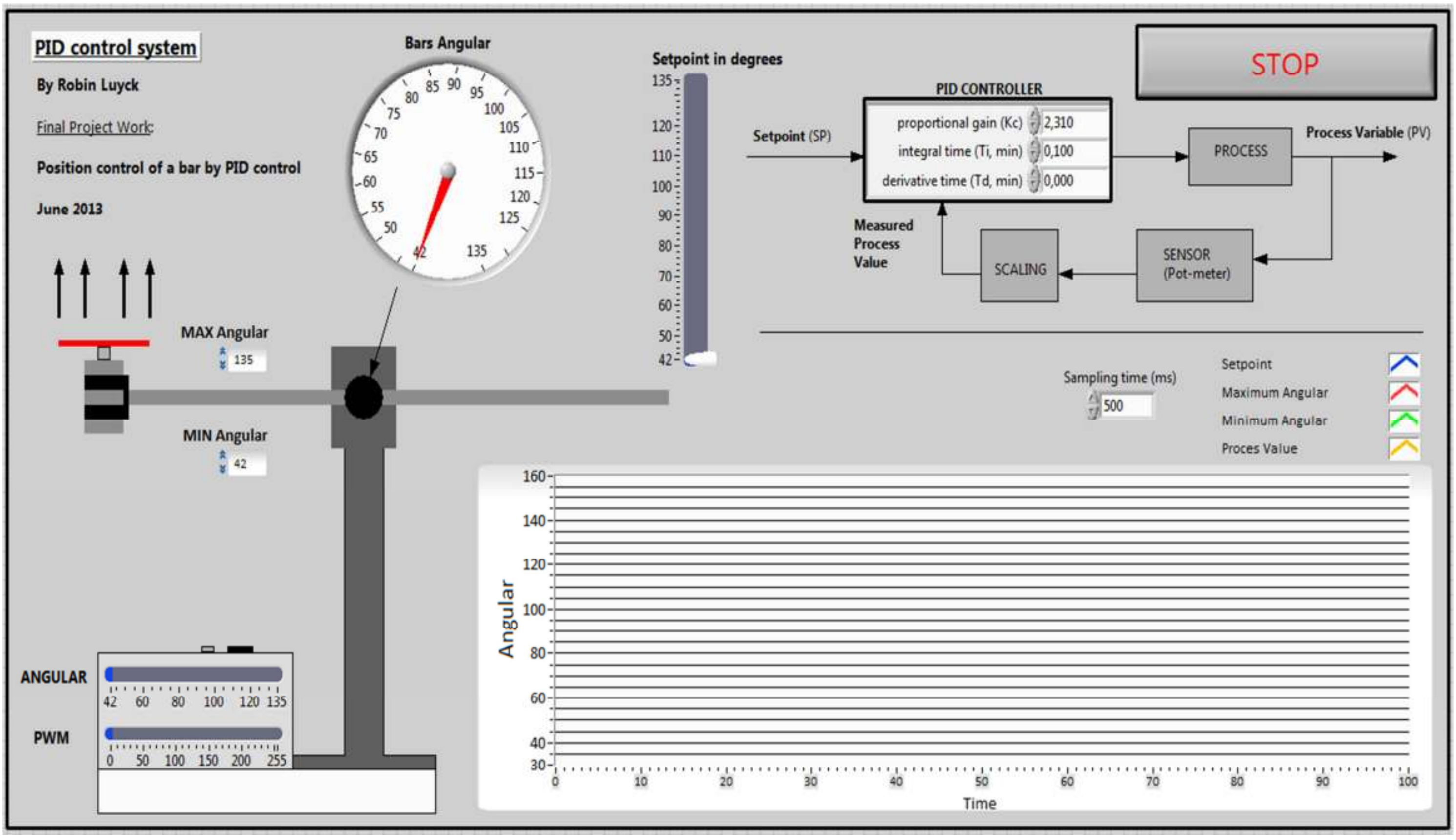
$$0.245\Delta F_g(s) = 0.01443s^2\Delta\theta(s) + Bs\Delta\theta(s) + 0.0346\Delta\theta(s)$$

So the transfer function of our system, supposed that the starting conditions are null, and taking ΔF=0, it will be:

$$\frac{\Delta\theta(s)}{\Delta F_g(s)} = \frac{0.245}{0.01443s^2 + Bs + 0.0346}$$

The friction coefficient B will change from one device to another, as the friction of each plant is a bit different.

2.6 Main LabVIEW controlling program

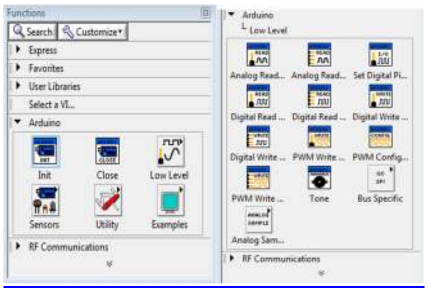


Front panel Main LabVIEW controlling program

2.6.1 The code from zero to controlling the bars position

The goal was creating a program (Front panel + Block diagram) in LabVIEW that communicates with the Arduino Uno its Inputs/Outputs to control the bars position.

➔ To communicate with the Arduino I found a “Arduino toolkit” which makes it possible to communicate between the LabVIEW software and the Arduino its I/O. This part is explained in the software chapter of this thesis.

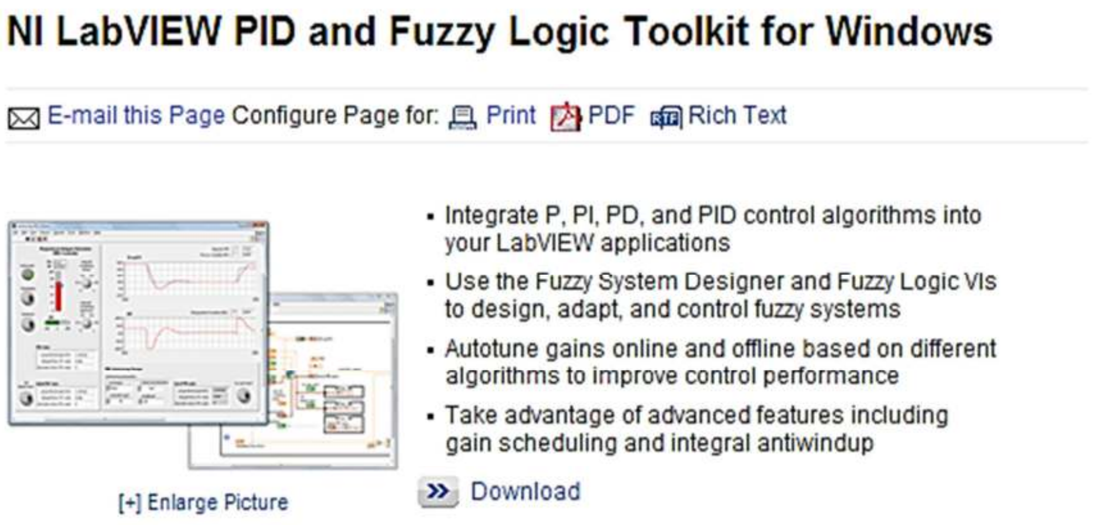


➔ LINK: http://vishots.com/getting-started-with-the-labview-interface-for-arduino/

➔ National Instruments offers the **PID Control toolkit** for LabVIEW, which was perfect for this project.

The toolkit can be downloaded from the NI website or installed by the NI Cd-roms:

LINK: http://sine.ni.com/nips/cds/view/p/lang/en/nid/209054



NI LabVIEW PID and Fuzzy Logic Toolkit for Windows

E-mail this Page Configure Page for: Print PDF Rich Text

- Integrate P, PI, PD, and PID control algorithms into your LabVIEW applications
- Use the Fuzzy System Designer and Fuzzy Logic VIs to design, adapt, and control fuzzy systems
- Autotune gains online and offline based on different algorithms to improve control performance
- Take advantage of advanced features including gain scheduling and integral antiwindup

[+] Enlarge Picture

» Download

2.6.1.1 The PID algorithm by National Instruments:

The PID controller compares the setpoint (SP) to the process variable (PV) to obtain the error (e).

In this case, the controller will compare **the angular we have chosen in the front panel** with the **actual angular from the bar**.

$$e = SP - PV$$

Then the PID controller calculates the controller action, u(t), where Kc is controller

$$u(t) = K_c\left(e + \frac{1}{T_i}\int_0^t e\,dt + T_d\frac{de}{dt}\right)$$

gain.

If the error and the controller output have the same range, –100% to 100%, controller gain is the reciprocal of proportional band. Ti is the integral time in **minutes**, also called the reset time, and Td is the derivative time in minutes, also called the rate time. The derivative action will not be used in this project, because with a PI-controller the not super complex plant can already be controlled.

The following formula represents the **proportional action**:

$$u_p(t) = K_c e$$

The following formula represents the **integral action**:

$$u_I(t) = \frac{K_c}{T_i}\int_0^t e\,dt$$

The following formula represents the **derivative action:** (Not used in this project)

$$u_D(t) = K_c T_d \frac{de}{dt}$$

2.6.1.2 Implementing the PID Algorithm with the PID Vis

This section describes how the PID VIs implement the positional PID algorithm. The subVIs used in these VIs are labelled so you can modify any of these features as necessary.

Error Calculation

The following formula represents the current error used in calculating proportional, integral, and derivative action.

$$e(k) = (SP - PV_f)$$

Proportional Action

Proportional Action is the controller gain times the error, as shown in the following formula.

$$u_P(k)=(K_c * e(k))$$

Trapezoidal Integration

Trapezoidal Integration is used to avoid sharp changes in integral action when there is a sudden change in PV or SP. Use nonlinear adjustment of integral action to counteract overshoot. The larger the error, the smaller the integral action, as shown in the following formula.

$$u_I(k)=\frac{K_c}{T_i}\sum_{i=1}^{k}\left[\frac{e(i)+e(i-1)}{2}\right]\Delta t$$

Partial Derivative Action

NOT USED

Controller Output

Controller output is the summation of the proportional, integral, and derivative action, as shown in the following formula:

$$u(k) = u_P(k) + u_I(k) + u_D(k)$$

Output Limiting

The actual controller output is limited to the range specified for control output.

$$\text{If } u(k) \geq u_{max} \text{ then } u(k) = u_{max}$$

$$\text{and}$$

$$\text{if } u(k) \leq u_{min} \text{ then } u(k) = u_{min}$$

The following formula shows the practical model of the PID controller:

$$u(t) = K_c\left[(SP - PV) + \frac{1}{T_i}\int_0^t (SP - PV)dt - T_d\frac{dPV_f}{dt}\right]$$

2.6.1.3 The Autotuning Algorithm

The toolkit also offers a Autotuning Algorithm:

I used autotuning to improve the stability of my system. Before you begin autotuning, you must establish a stable controller!

I used the following Ziegler-Nichols method to get a stable controller:

2.6.1.4 Closed-Loop (Ultimate Gain) Tuning Procedure

The PID toolkit also offers a Autotuning Algorithm:

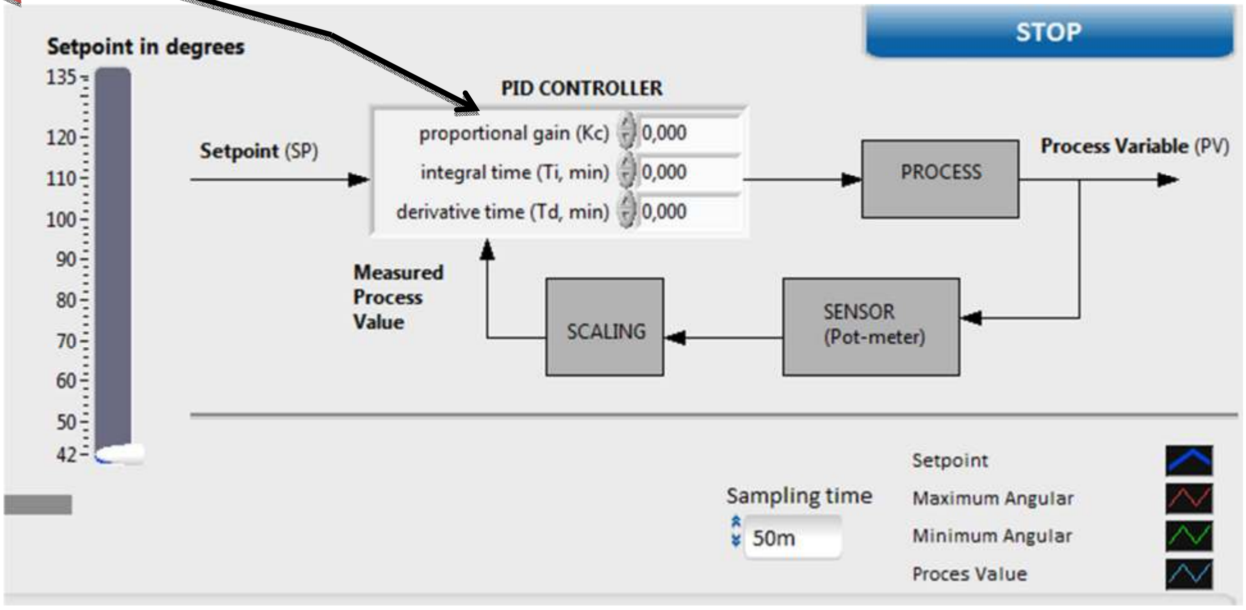
I used autotuning to improve the stability of my system. Before you begin autotuning, you must establish a stable controller!

I used the following Ziegler-Nichols method to get a stable controller:

Although the closed-loop (ultimate gain) tuning procedure is very accurate, you must put your process in steady-state oscillation and observe the PV on a strip chart. I completed the following steps to perform the closed-loop tuning procedure:

Step 1: Set both the **derivative time** and the **integral time (does not matter, because I don't use it in my controller)** on your PID controller to 0.

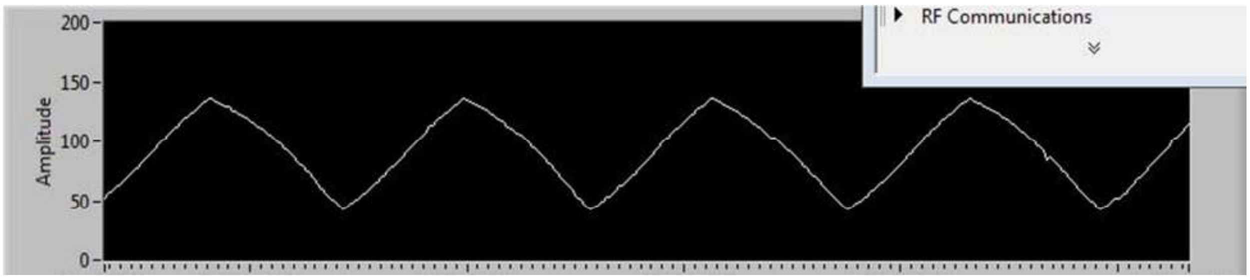
Step 2: With the controller in automatic mode, slowly and carefully increase the **proportional gain** (Kc) in **small increments**. Make a small change in SP (Angular wished) to disturb the loop after each increment. As you increase Kc, the value of PV should begin to oscillate. Keep making changes until the oscillation is **sustained, neither growing nor decaying over time**.

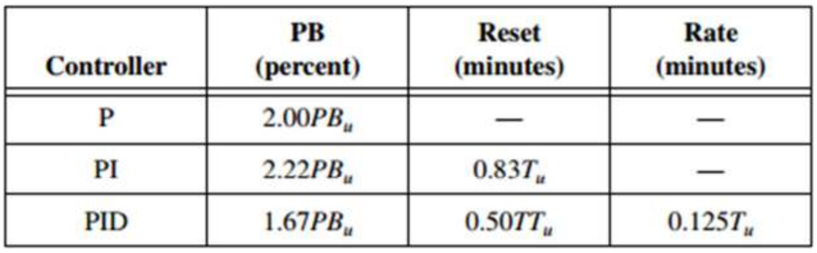


Step 3: Record the controller proportional band (PBu) as a percent, where PBu= 100/Kc.

Step 4: Record the period of oscillation (Tu) in minutes.

Step 5: Multiply the measured values by the factors shown in the table on this page and enter the new tuning parameters into your controller. The table provides the proper values for a quarter-decay ratio. If you want less overshoot, increase the gain Kc.





Controller	PB (percent)	Reset (minutes)	Rate (minutes)
P	$2.00PB_u$	—	—
PI	$2.22PB_u$	$0.83T_u$	—
PID	$1.67PB_u$	$0.50TT_u$	$0.125T_u$

Closed-Loop-Quarter-Decay Ratio Values

My values voor **P** and **I** after this Closed-Loop (Ultimate Gain) Tuning Procedure methode were:

P = 2.9

I = 0.4

➔ Now I will use the PID with Autotuning VI to get the process even more stable:

To use the Autotuning Wizard to improve your controller performance, you must first create your control application and determine PID parameters that produce **stable control of the system**. For my system those are the values I just calculated by the closed loop tuning procedure (P=2.9 and I=0.4)

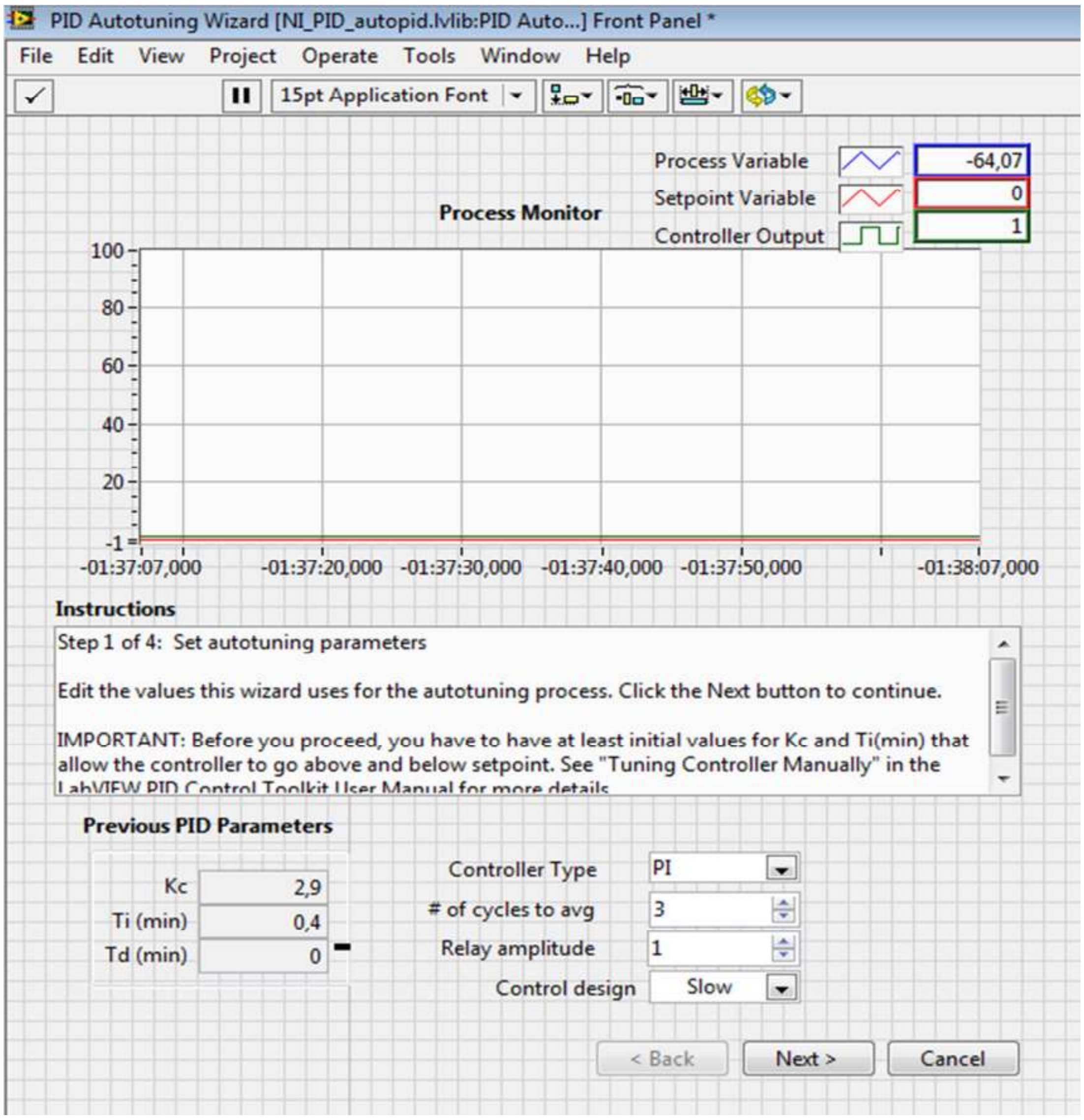
You can develop the control application using either the PID VI, the PID Gain Schedule VI, or the PID with Autotuning VI.

Because the PID with Autotuning VI has input and output consistent with the other PID VIs, you can replace any PID VI with it. The PID with Autotuning VI has several additional input and output values to specify the autotuning procedure.

The two additional input values are autotuning parameters and autotune?. autotuning parameters is a cluster of parameters that the VI uses for the autotuning process.

Because the Autotuning Wizard allows you to specify all of these parameters manually, you can leave the autotuning parameters input unwired. The autotune? input takes a Boolean value supplied by a user control. Wire a Boolean control on the front panel of your application to this input. When you press the Boolean control, the Autotuning Wizard opens automatically.

Set the Boolean control mechanical action to Latch When Released so that the Autotuning Wizard does not open repeatedly when the user presses the control. The Autotuning Wizard steps the user through the autotuning process.



Process Variable -64,07
Setpoint Variable 0
Controller Output 1

Instructions

Step 1 of 4: Set autotuning parameters

Edit the values this wizard uses for the autotuning process. Click the Next button to continue.

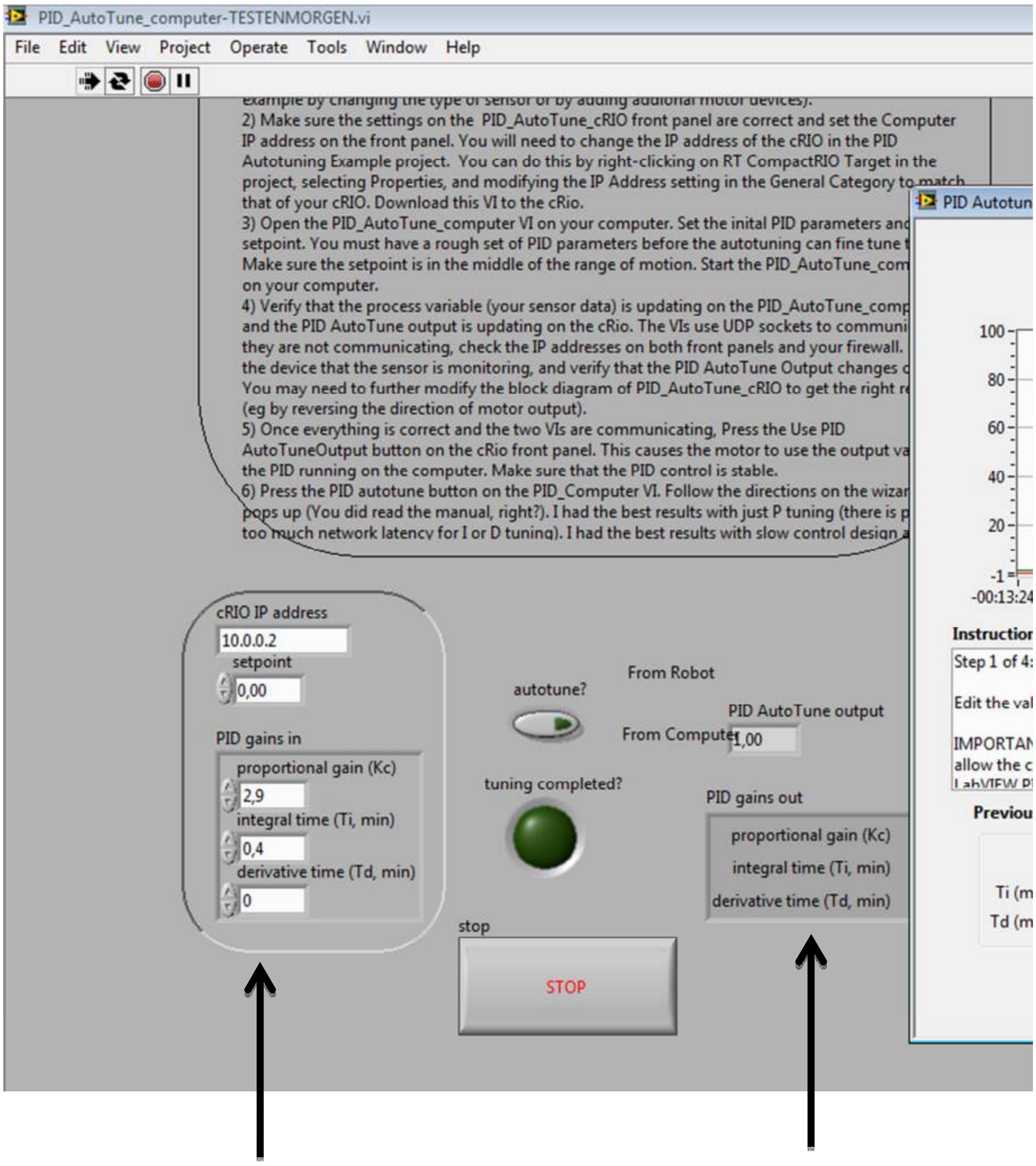
IMPORTANT: Before you proceed, you have to have at least initial values for Kc and Ti(min) that allow the controller to go above and below setpoint. See "Tuning Controller Manually" in the LabVIEW PID Control Toolkit User Manual for more details

Previous PID Parameters

Kc 2,9
Ti (min) 0,4
Td (min) 0

Controller Type: PI
# of cycles to avg: 3
Relay amplitude: 1
Control design: Slow

The Autotuning wizard



PID_AutoTune_computer-TESTENMORGEN.vi

example by changing the type of sensor or by adding additional motor devices).
2) Make sure the settings on the PID_AutoTune_cRIO front panel are correct and set the Computer IP address on the front panel. You will need to change the IP address of the cRIO in the PID Autotuning Example project. You can do this by right-clicking on RT CompactRIO Target in the project, selecting Properties, and modifying the IP Address setting in the General Category to match that of your cRIO. Download this VI to the cRio.
3) Open the PID_AutoTune_computer VI on your computer. Set the inital PID parameters and setpoint. You must have a rough set of PID parameters before the autotuning can fine tune Make sure the setpoint is in the middle of the range of motion. Start the PID_AutoTune_com on your computer.
4) Verify that the process variable (your sensor data) is updating on the PID_AutoTune_comp and the PID AutoTune output is updating on the cRio. The VIs use UDP sockets to communi they are not communicating, check the IP addresses on both front panels and your firewall. the device that the sensor is monitoring, and verify that the PID AutoTune Output changes You may need to further modify the block diagram of PID_AutoTune_cRIO to get the right r (eg by reversing the direction of motor output).
5) Once everything is correct and the two VIs are communicating, Press the Use PID AutoTuneOutput button on the cRio front panel. This causes the motor to use the output va the PID running on the computer. Make sure the PID control is stable.
6) Press the PID autotune button on the PID_Computer VI. Follow the directions on the wizar pops up (You did read the manual, right?). I had the best results with just P tuning (there is p too much network latency for I or D tuning). I had the best results with slow control design a

cRIO IP address: 10.0.0.2
setpoint: 0,00
PID gains in
proportional gain (Kc): 2,9
integral time (Ti, min): 0,4
derivative time (Td, min): 0

autotune?
tuning completed?
From Robot
From Computer
PID AutoTune output: 1,00
PID gains out
proportional gain (Kc)
integral time (Ti, min)
derivative time (Td, min)

The PID gains in, before the autotuning and the PID gains out, after going through the autotuning wizard

After doing the AutoTuning I got the following values for P and I:

P = 2.4

I = 0.2

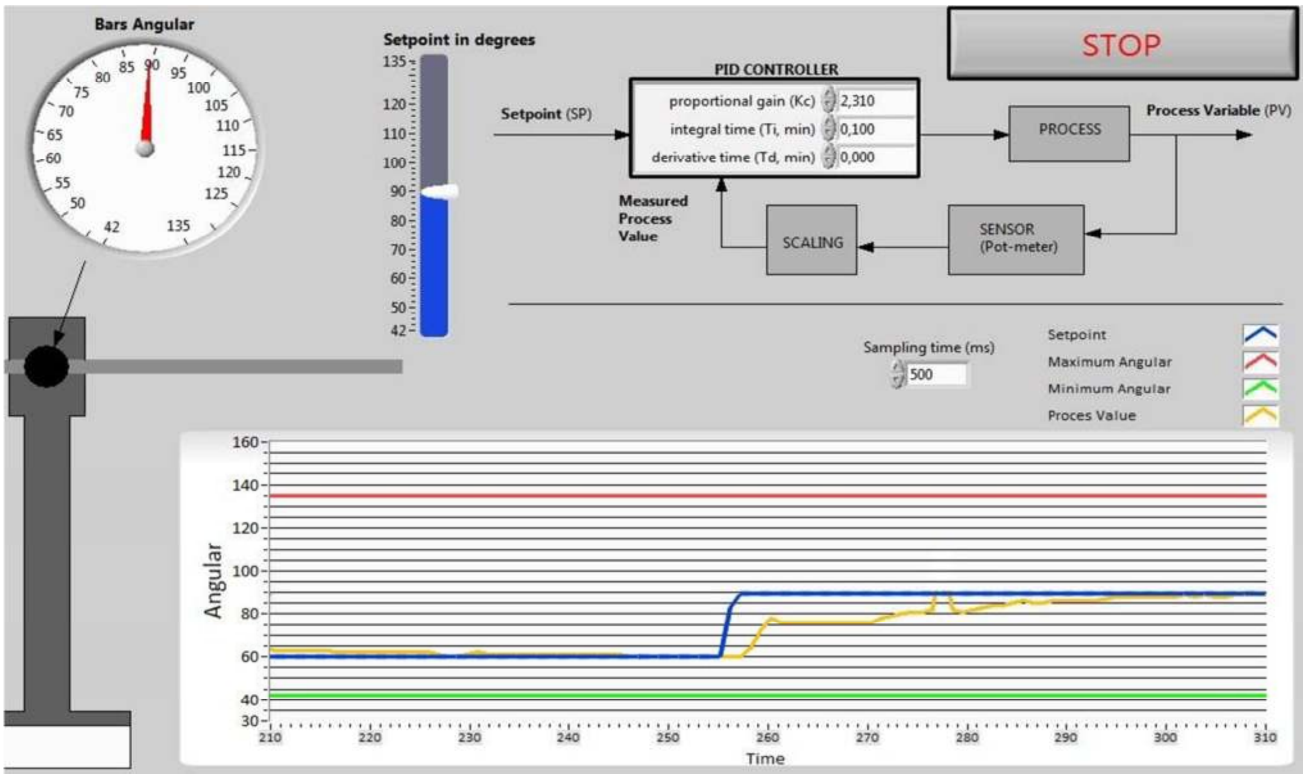
After doing the trail and error by myself for values close to the AutoTuning outputs, **my final results (value) for P and I** are:

P = 2.31

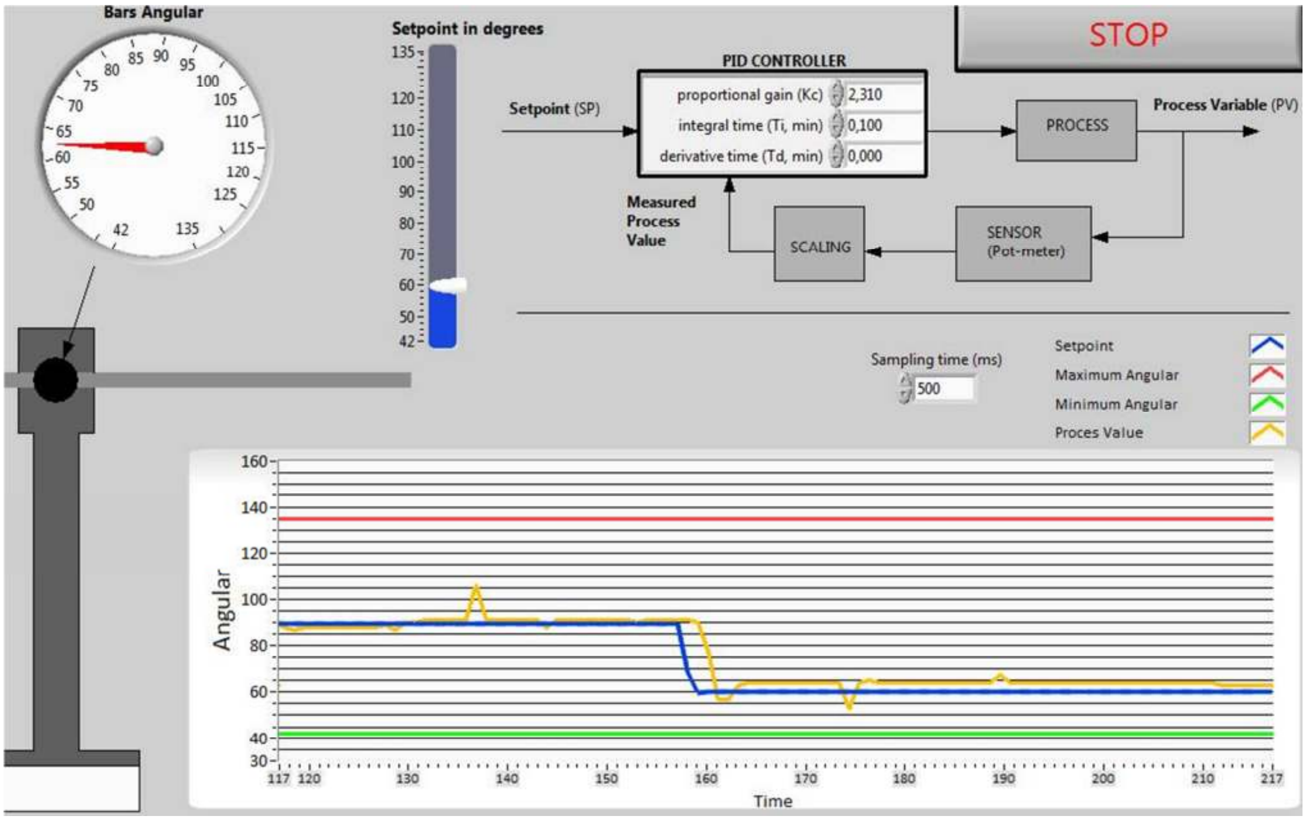
I = 0.1

Testing with the final PI values:

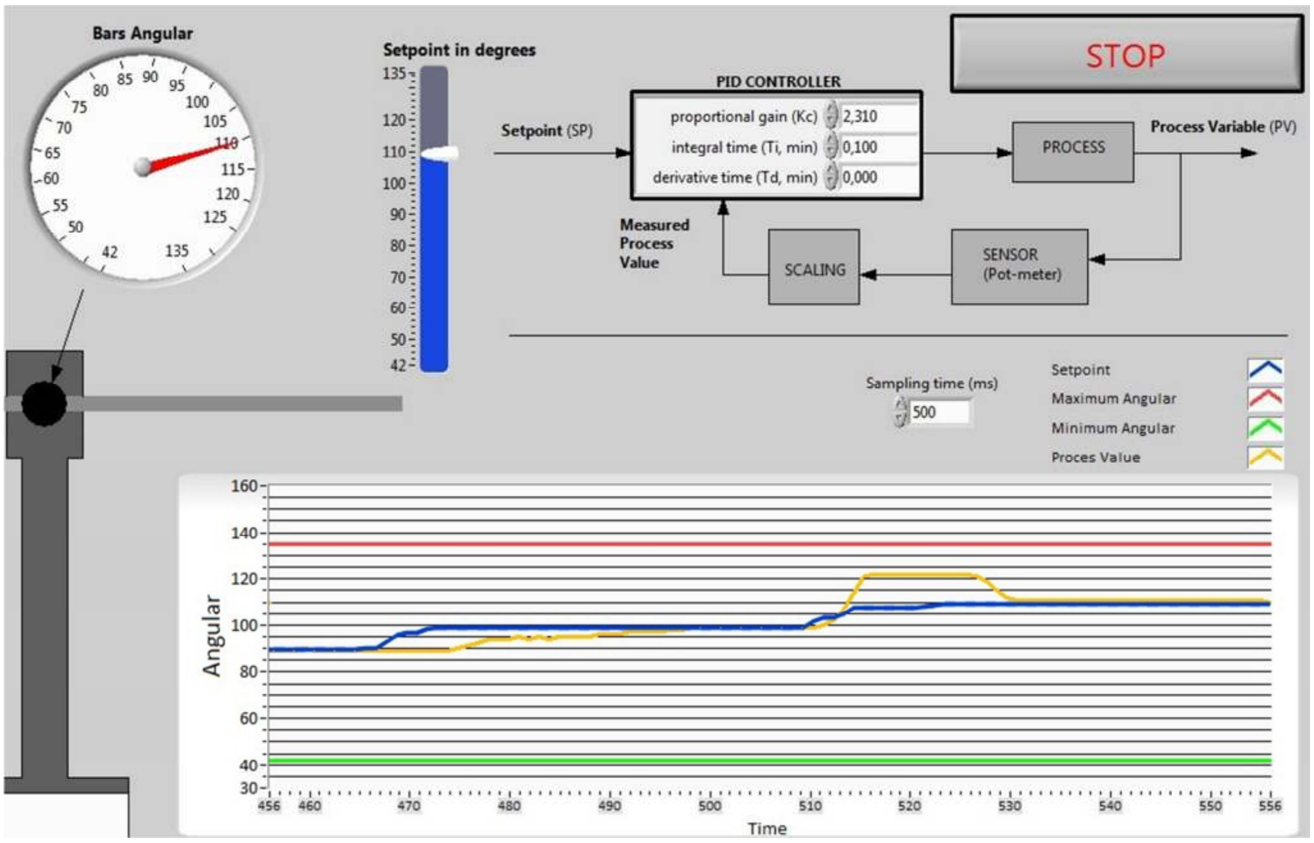
➔ 60 to 90 °



➔ 90° to 60°

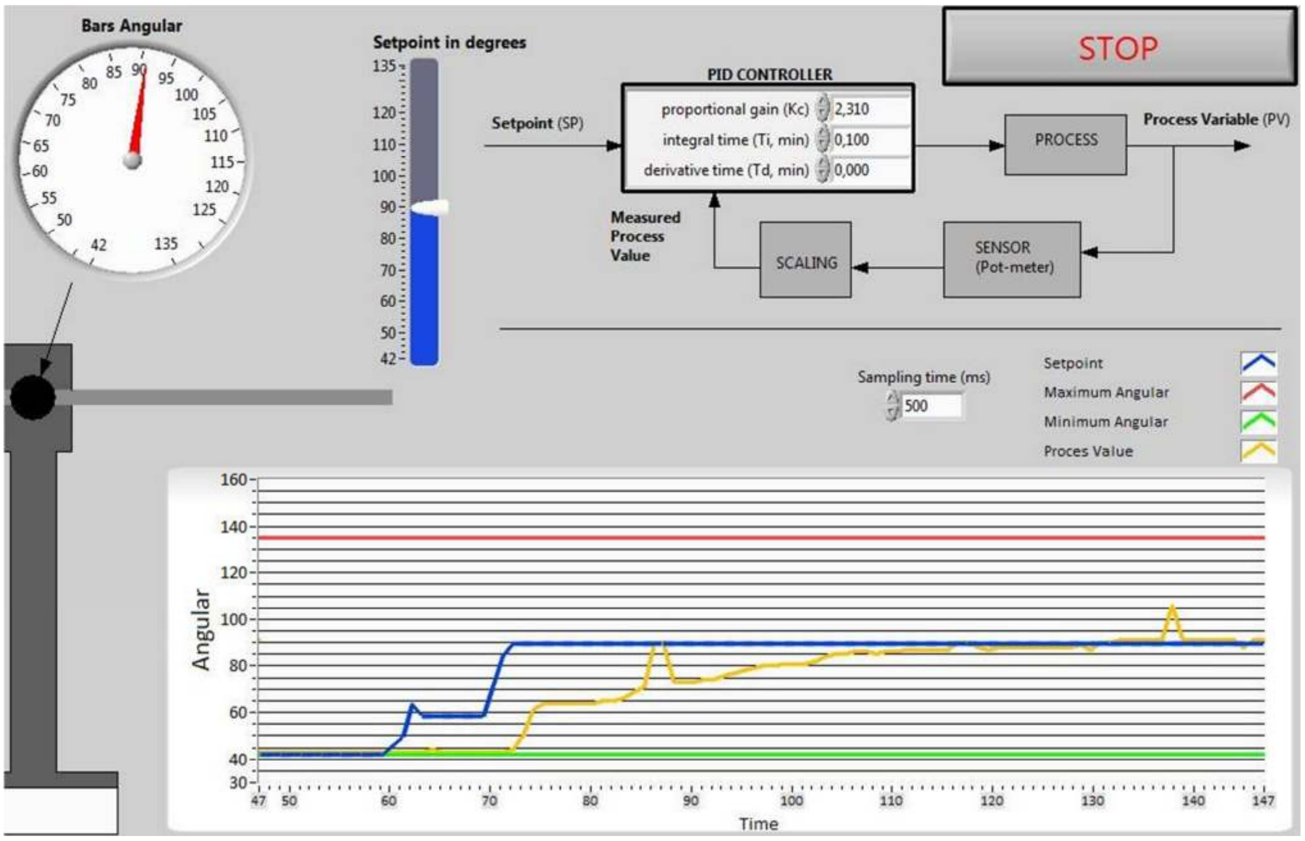


➔ 90° to 100° to110°



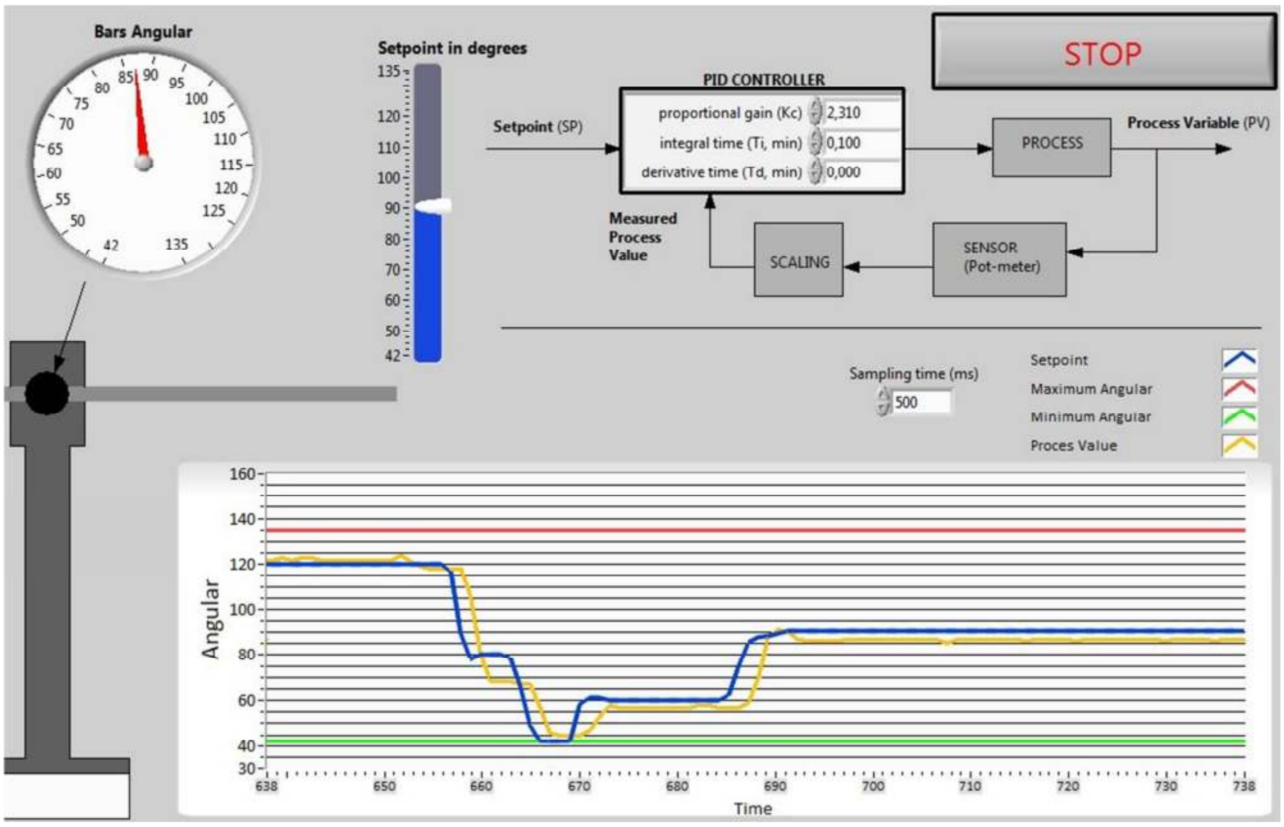
proportional gain (Kc) 2,310
integral time (Ti, min) 0,100
derivative time (Td, min) 0,000
Sampling time (ms) 500

➔ Minimul angular (42°) to 90°

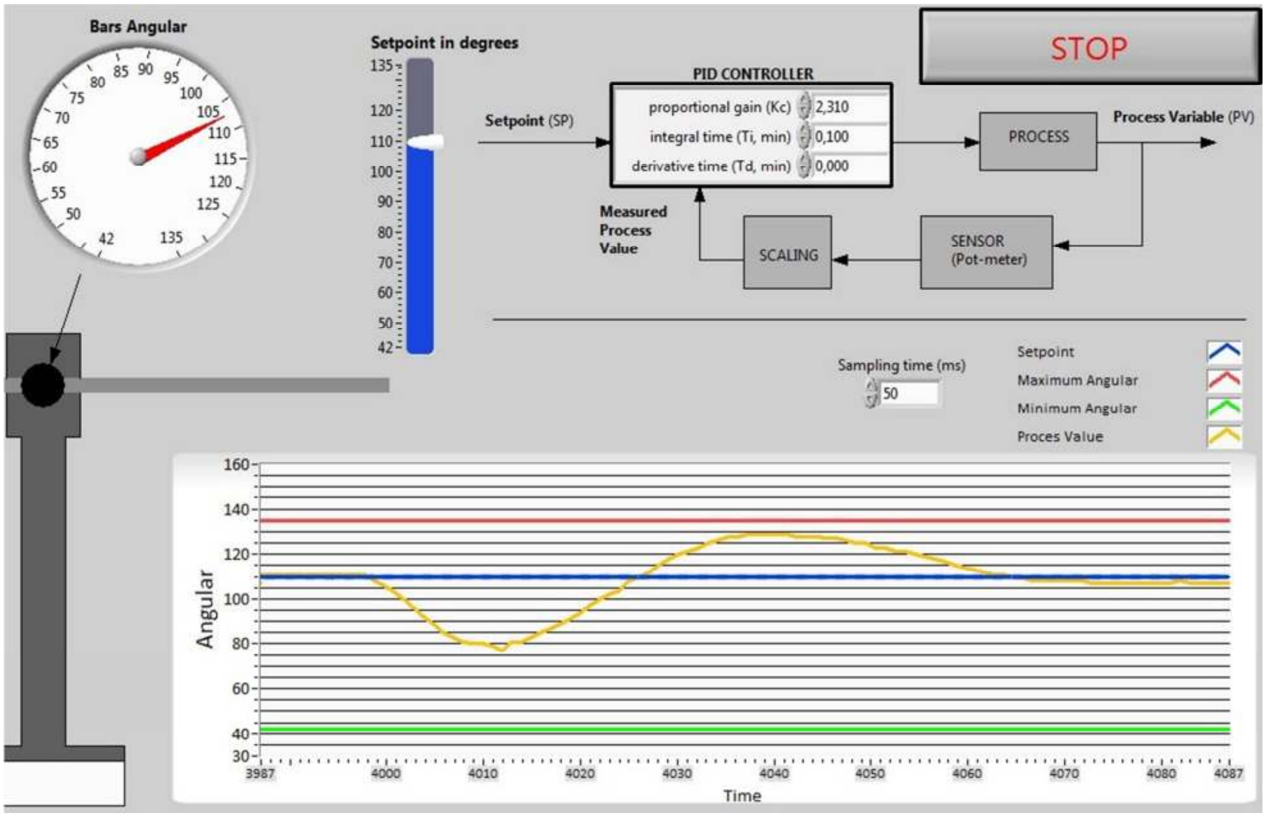


proportional gain (Kc) 2,310
integral time (Ti, min) 0,100
derivative time (Td, min) 0,000
Sampling time (ms) 500

➔ Random set points



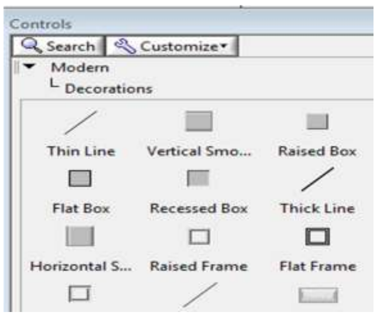
➔ Adding an error to the system (pushing the bar down)

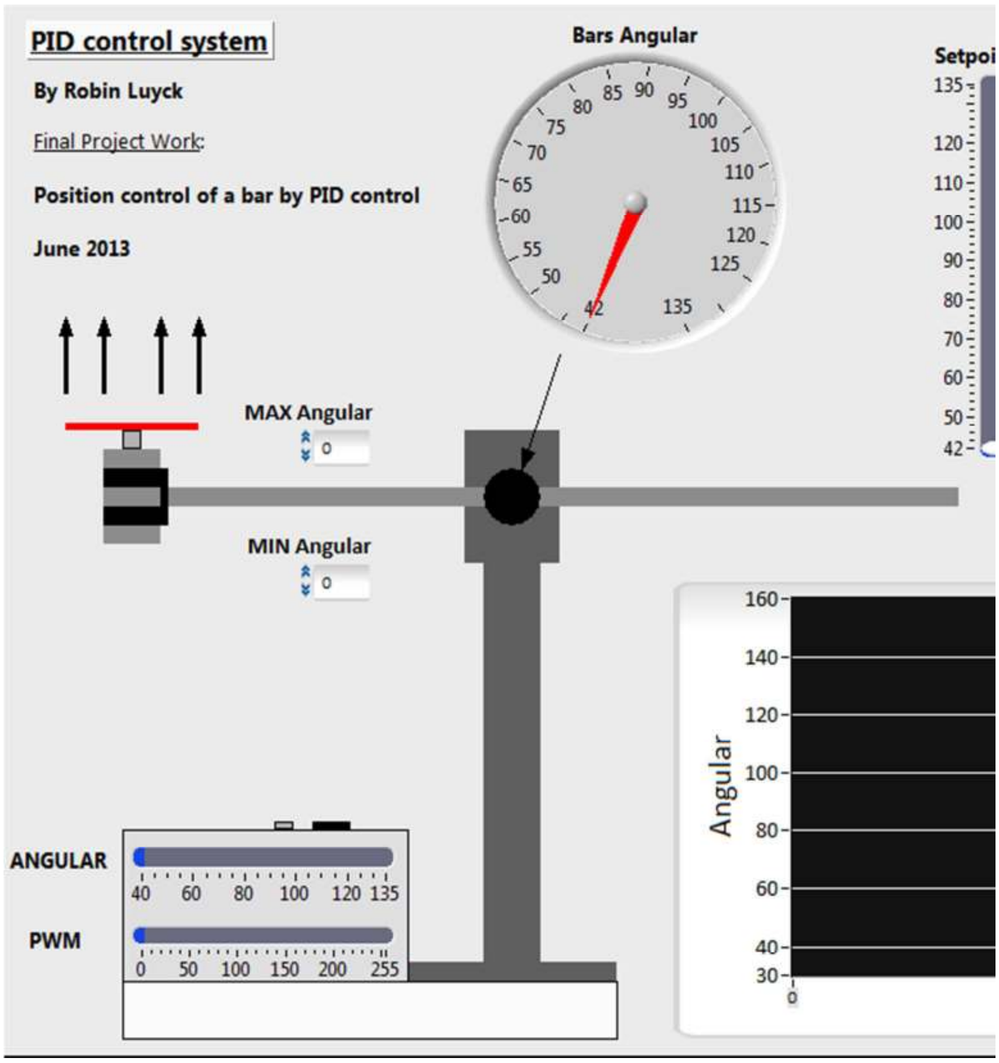


2.6.2 Creating the front panel

Now that I have the PI values, I can make the definitive front panel.

I used the decorations under **Controls Palette – Modern – Decorations** to create a "drawing" that looks almost the same as the PLANT in real.




PID control system

By Robin Luyck

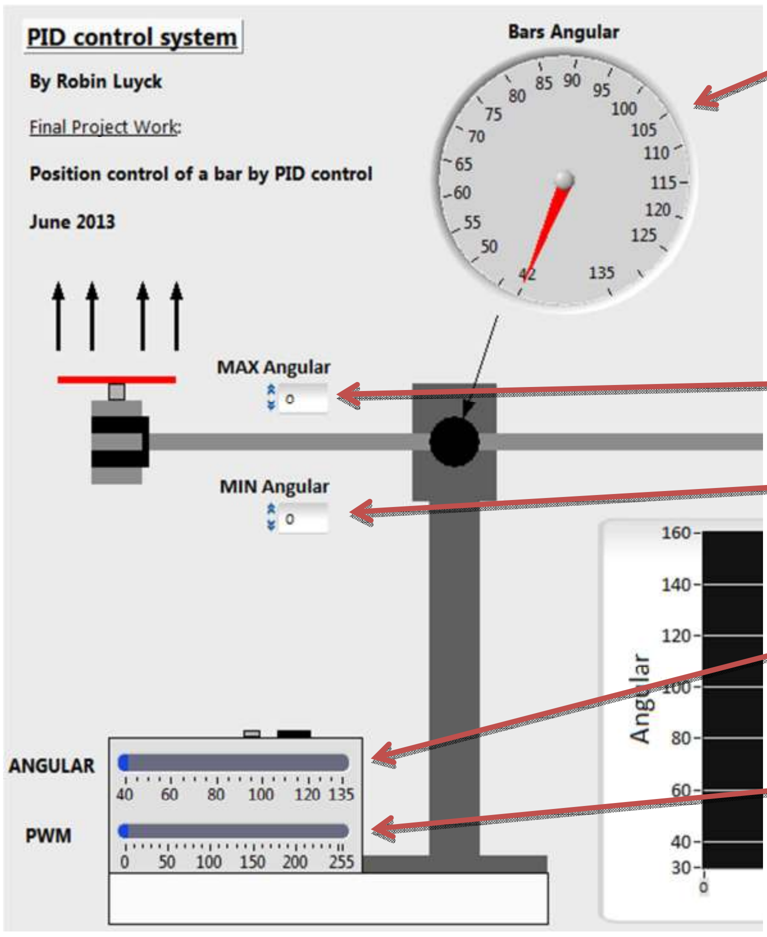
Final Project Work:

Position control of a bar by PID control

June 2013

The PLANT created in the front panel

Information about the frontpanel:



Displays the bars angular

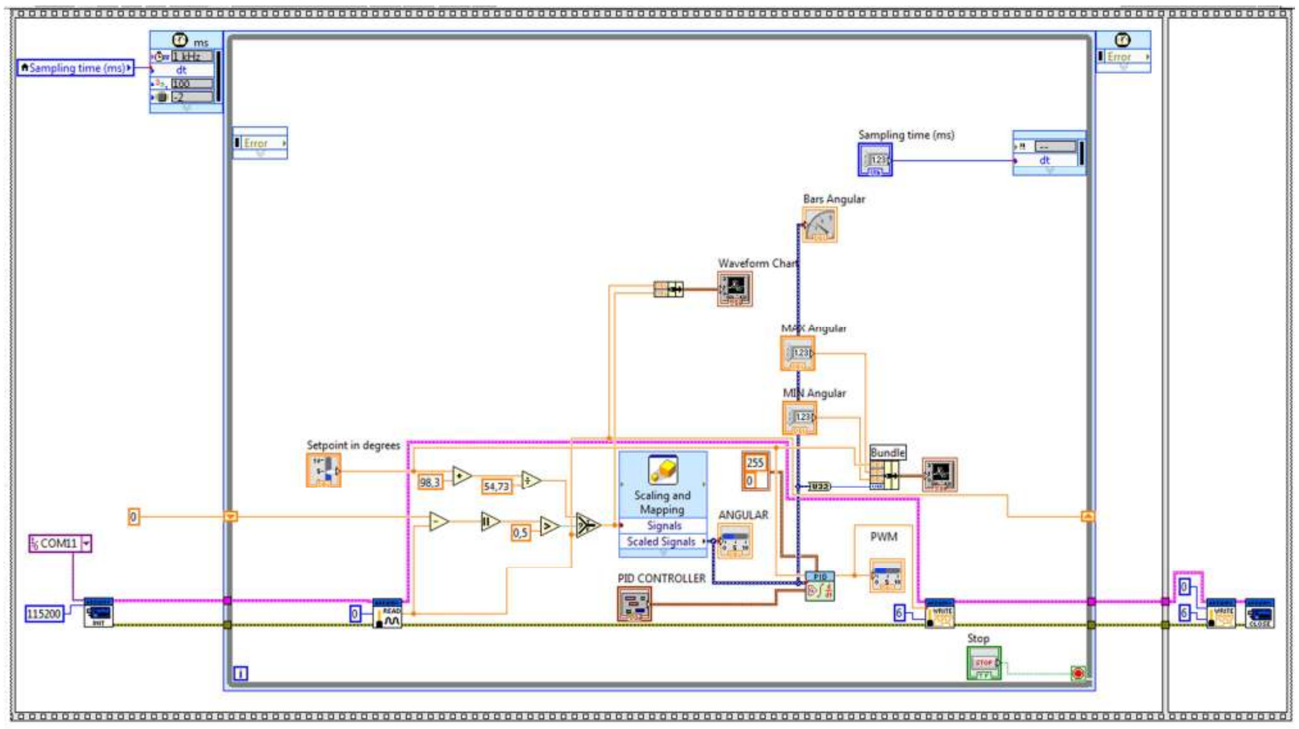
Insert the mechanical max angular

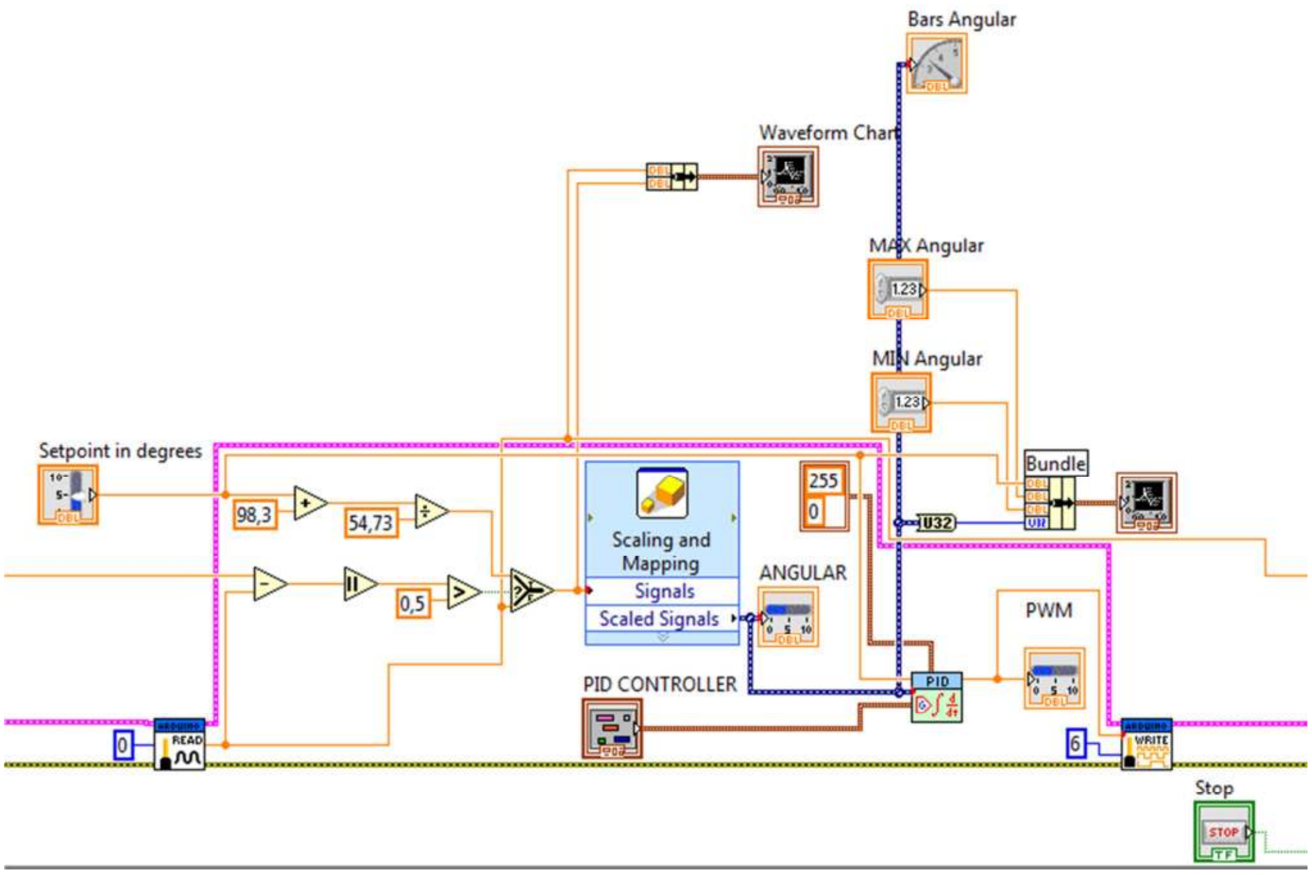
Insert the mechanical min angular

Display the angular on the "white box"

Display the PWM that is done on the transistor at the moment

2.6.3 Creating the block diagram





Explenation from left to right:

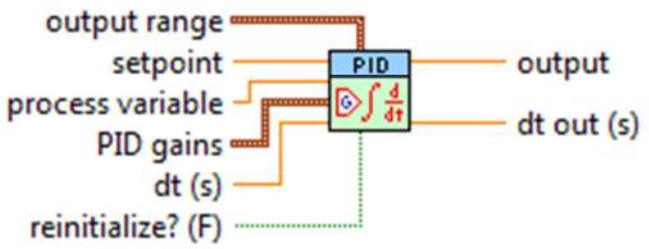
"Everything" is placed in a flat sequence structure, were in the first frame a while loop is created which will run when running the VI. On the left side you can see the "init" vi from the Arduino toolkit, which is explained in the software part. Here I give up the COM port for the USB, to communicate with the Arduino + the baud rate.

Then the next thing you can see is the "*Analog read pin*" vi from the toolkit. The Analog pin number is given up by a constant, the potentiometer its "output" is attached to analog input pin 0…

Then the following step I do is scaling the 0-5V signal to the mechanical angular, after this I pass through a low pass filter to filter out the created noise by the propeller. Now we have the Process value after the filter, this one will enter the PID vi its "process value" input.

Other connections made with the PID vi are:

- Set point by a slider in the front panel
- The PID values (gains)
- The min and max value the PID controller can send on its output (0-255 for the PWM vi from the Arduino toolkit)
- The output from the PID controller is connected to the PWM vi from Arduino to control the motor his speed.



Implements a PID controller using a PID algorithm for simple PID applications or high speed control applications that require an efficient algorithm. The PID algorithm features control output range limiting with integrator anti-windup and bumpless controller output for PID gain changes. Use the DBL instance of this VI to implement a single control loop. Use the DBL Array instance to implement parallel multi-loop control.

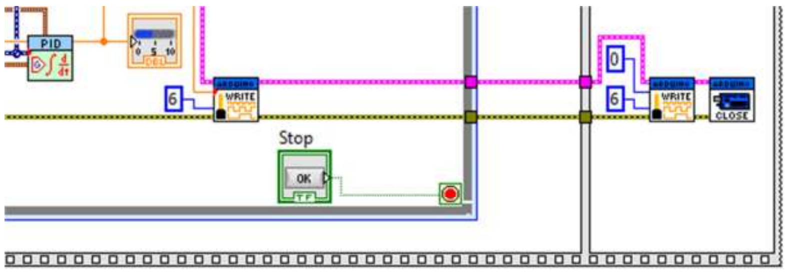
The PID vi block

2.6.4 Troubleshooting the main LabVIEW program

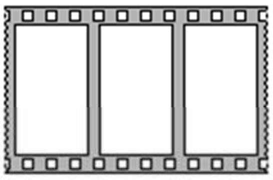
➔ Problem **1**:

While running the while loop and I would like to stop the program by pressing the **main stop button**, the PWM vi kept running it's last value and so the DC motor kept running which was not the idea! Therefore I inserted a **Flat Sequence Structure**.

The idea behind using this structure is that when the while loop has been stopped which takes place in the first frame (frame 1) and where the PWM vi will get the PID controller its output value for the duty cycle on pin 6, the second frame (frame 2) will get activated and the first frame deactivated. In this second frame, the PWM vi takes place again, but here it writes a "0" to the duty cycle input and so the main program will stop + the motor will stop turning.



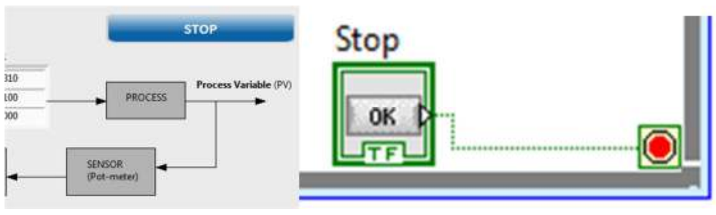
*On the left side **frame 1** and on the right side **frame 2***



Consists of one or more subdiagrams, or frames, that execute sequentially. Use the Flat Sequence structure to ensure that a subdiagram executes before or after another subdiagram.

Data flow for the Flat Sequence structure differs from data flow for other structures. Frames in a Flat Sequence structure execute from left to right and when all data values wired to a frame are available. The data leaves each frame as the frame finishes executing. This means the input of one frame can depend on the output of another frame.

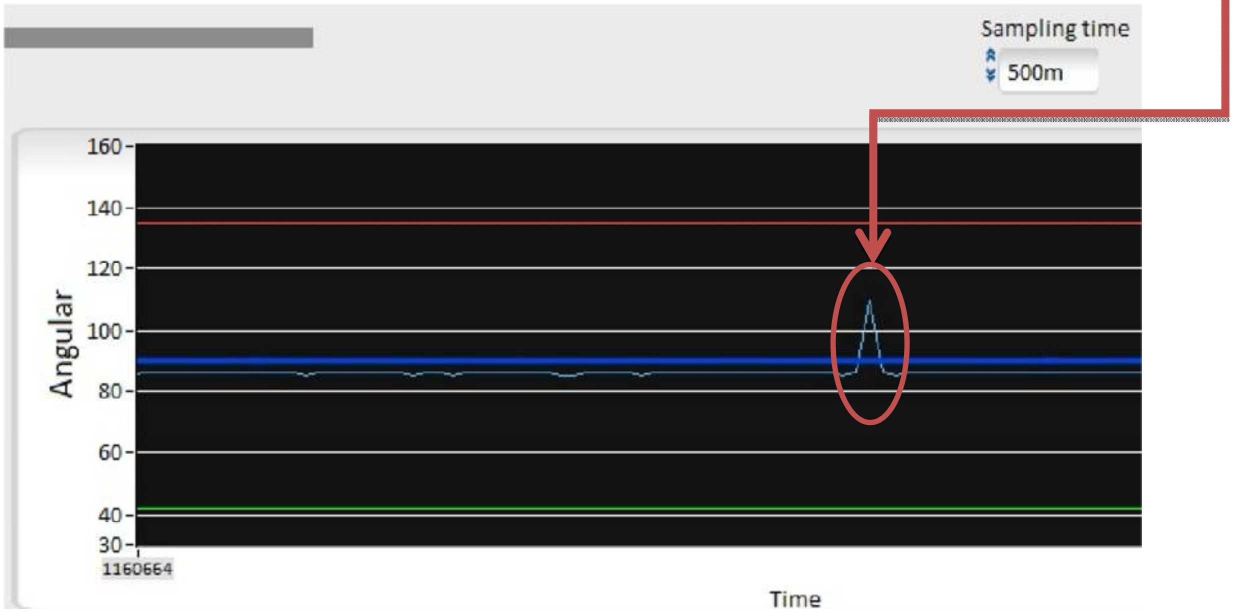
Flat Sequence Structure (Flat Sequence)



The main STOP button

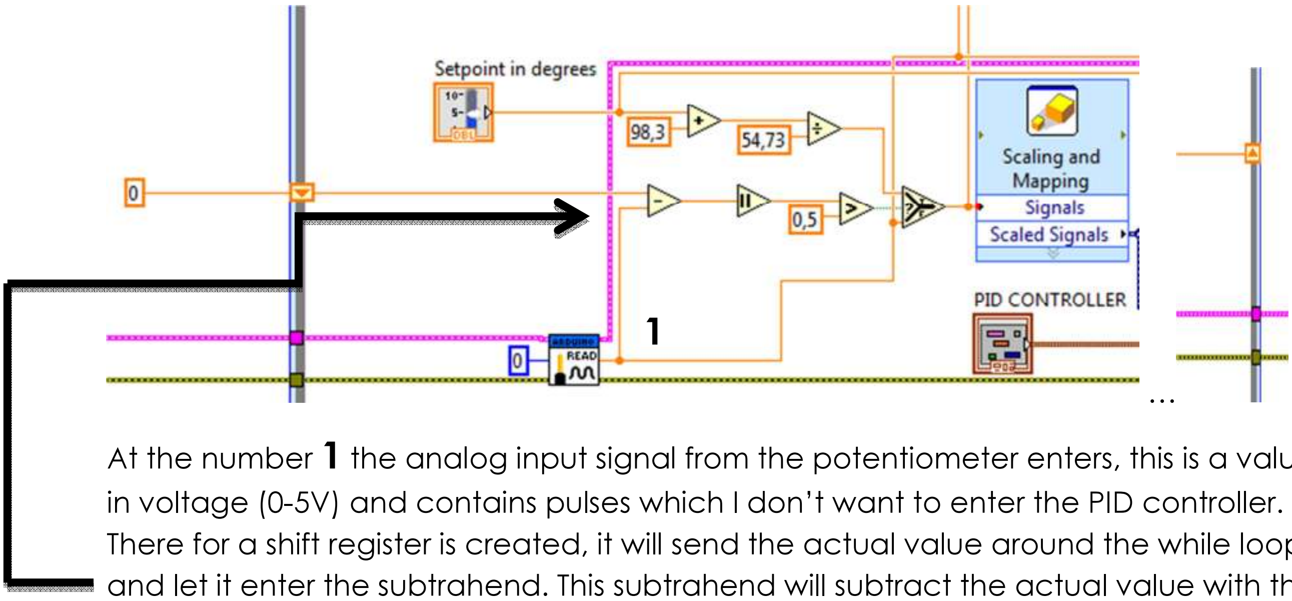
➔ Problem **2**:

When the program is running, the motor is running, the propeller will create disturbing on the pot-meter value (pulses) entering the LabVIEW program. The PID controller will react on these pulses.



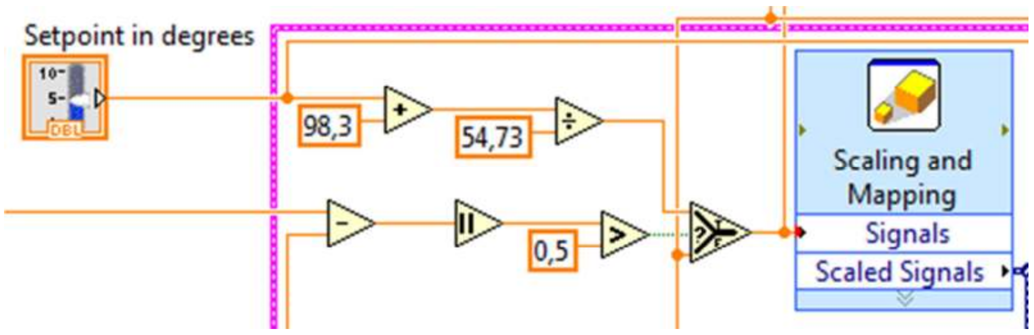
The proportional component depends only on the **difference between the set point and the process variable**. This difference is referred to as the **Error term**. So the PID controller will change its output because of these pulses, which is a problem!

Therefore the following solution:



At the number **1** the analog input signal from the potentiometer enters, this is a value in voltage (0-5V) and contains pulses which I don't want to enter the PID controller. There for a shift register is created, it will send the actual value around the while loop and let it enter the subtrahend. This subtrahend will subtract the actual value with the previous value.

After this the subtracted value will be compared with "0.5". This 0.5 is in voltages. I did measuring on the pulses before and the pulses that disturb the system the most are the pulses greater 0.5 Volts. When the disturbing pulse is greater as 0.5V, the comparison is true. After the comparison U can see the wire entering a select block, which will compare the signal entering the "?" input with the TRUE and FALSE value.



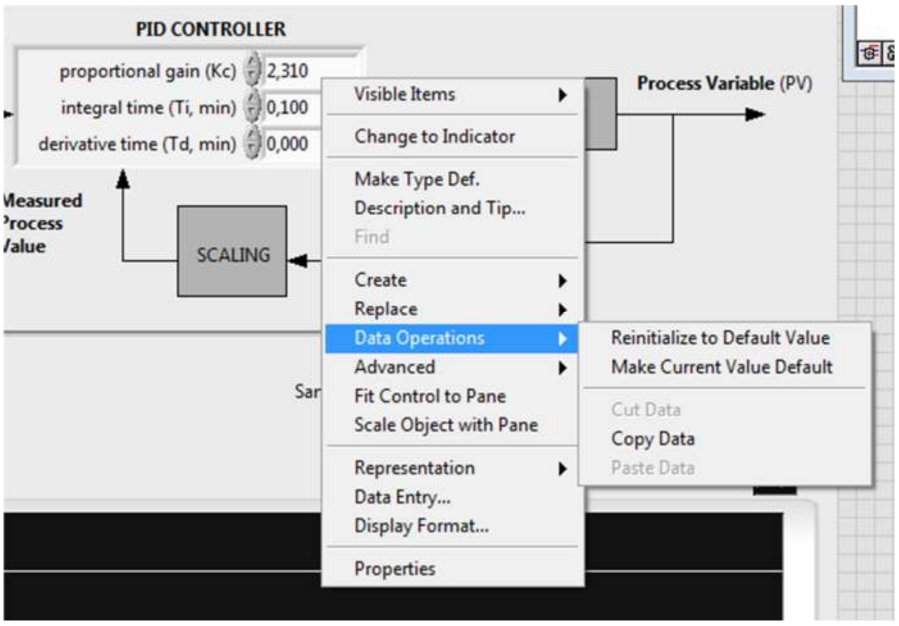
The FALSE value presents the actual sensed value coming from the analog input.

The TRUE value represents the set point in degrees converted to a voltage signal.

If the PULSE is greater as 0.5V the TRUE statement gets executed, if not and the PULSE is smaller as 0.5, which presents the FALSE statement, the sensed value from the analog input will enter the Scaling function and go to the PID controller with no pulse or a pulse smaller then 0.5V.

➔ Problem **3**:

Every time when I started up the LabVIEW program and the main program, my PI values were back to "0". Therefore I needed to right click on the numeric controller and make the Current Value Default.



3. CHAPTER 3: Software

In this chapter I will give information on all the software which is used in the two parts (Info software, installation and examples)

3.1 Arduino

3.1.1 Intro to Arduino

Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It is intended for designers, hobbyists, and anyone interested in creating interactive objects or environments.

Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling motors, lights and other actuators. The microcontroller on the board is programmed using the Arduino programming language based on C/C++ and the Arduino development environment (based on Processing). Arduino projects can be stand-alone or they can communicate with software running on a computer (In my case LabVIEW!).

The boards can be built by hand or purchased preassembled; the software can be downloaded for free.

Its goal is based on small projects, for fans and lovers of electronics. Its main feature is the ease with which we can program something, unlike other plates with microcontrollers on the market, whose programming is much more laborious. Moreover Arduino belongs to a free software based company (Open source platform), this allows us to use it in any environment and project.

3.1.2 Installation Arduino software

The first thing that should be noted is that Arduino has a free software program.

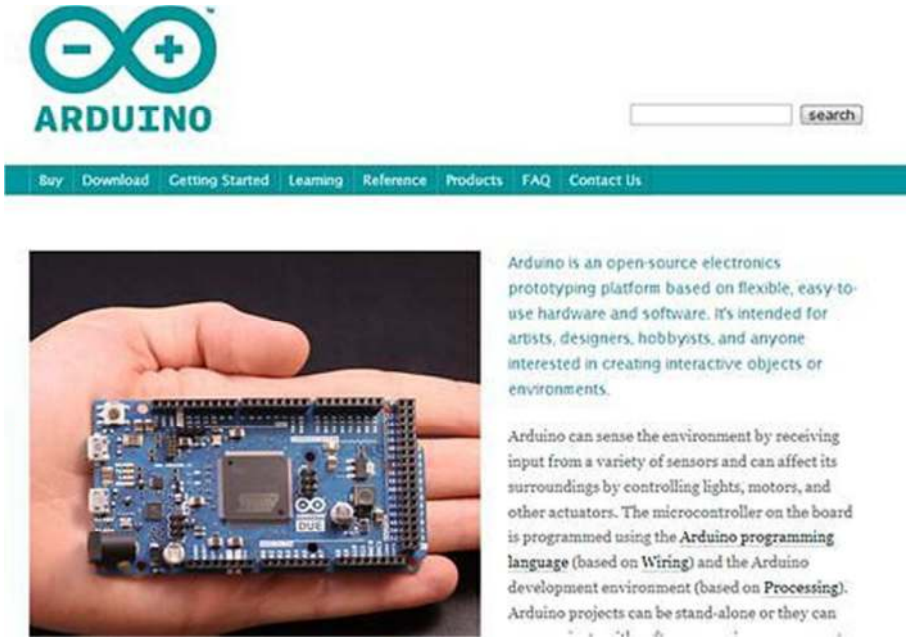
In this case go to the official website for Arduino: http://arduino.cc

The website show you all sorts of information about Arduino, it has a forum where users can exchange experiences, projects, blog, questions, etc… It also has an online store where you can buy Arduino products.

DOWNLOAD THE ARDUINO SOFTWARE:

The open-source Arduino environment makes it easy to write code and upload it to the I/O board.

At the website there is a section where you can download the free software, it's the section named "download".



Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments.

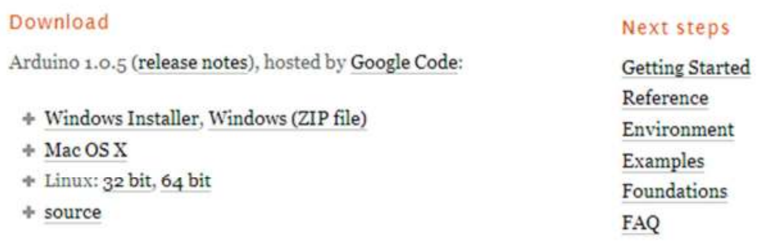
Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the **Arduino programming language** (based on **Wiring**) and the Arduino development environment (based on **Processing**). Arduino projects can be stand-alone or they can

Arduino has developed an interactive environment for development, open source course called IDE (Integrated Development Environment), the IDE to program the Arduino is an easy, based on C/C++, processing and wiring that is the language the Arduino uses.

Arduino provides your IDE in various formats, depending on the operating system that your computer has, Arduino currently reached by the version number 19, but you can download any of the IDE prior to this in case you have some sort of compatibility issue.

In my case as I have to install the software on a computer equipped with Windows I choose the latest version out there for Windows.

In each version of Arduino the functions have been improved, the improvement are also explained every time. So it is worth going the website from time to time to check for a new version.


Download

Arduino 1.0.5 (release notes), hosted by Google Code:

- Windows Installer, Windows (ZIP file)
- Mac OS X
- Linux: 32 bit, 64 bit
- source

Next steps

Getting Started
Reference
Environment
Examples
Foundations
FAQ

Once downloaded into a folder, we have to unzip it and put it in a place easily accessible as we will need it repeatedly. In this folder we find the libraries, examples, drivers and the Arduino program itself.

By now we have the program for starting writing a program in the Arduino environment but first we have to set up the connection between the PC (or laptop) and the Arduino UNO. This is explained in the following chapter.

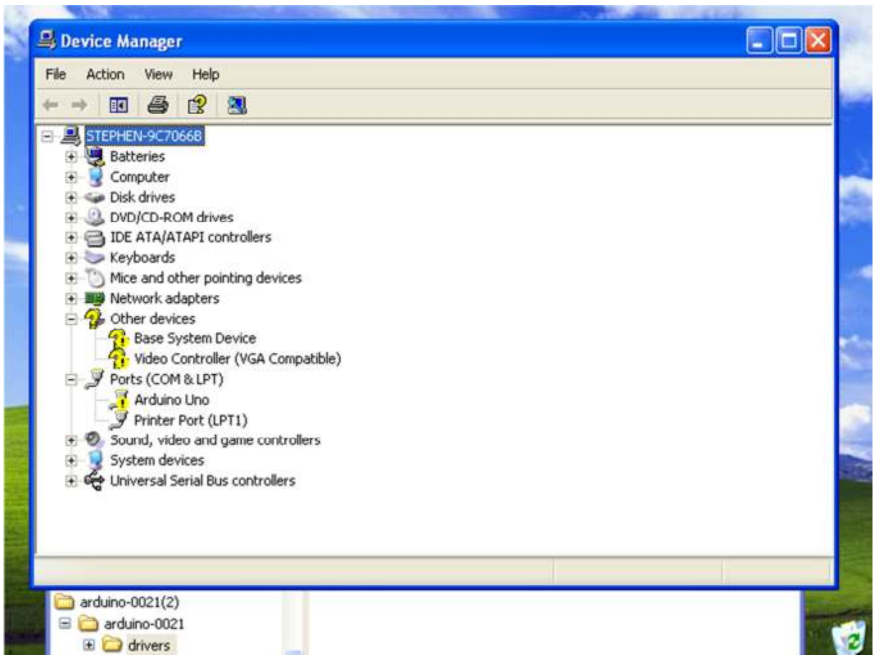
3.1.3 Installation Arduino drivers

Plug in your board and wait for Windows to begin its driver installation process. After a few moments, the process will fail, despite its best efforts..

Then click on the Start Menu, and open up the Control Panel.

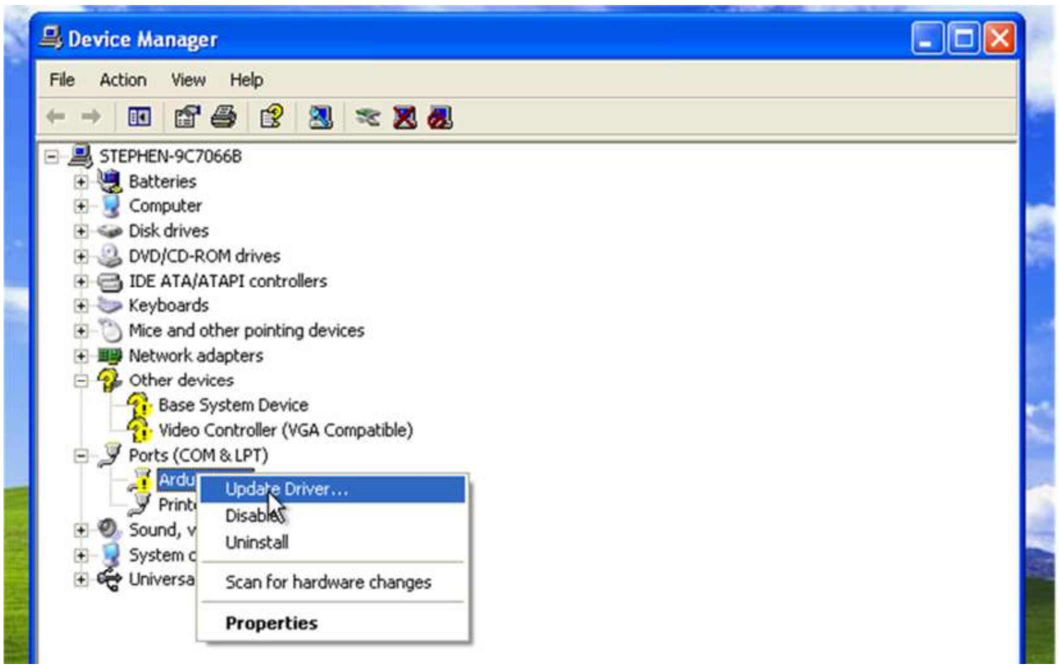
While in the Control Panel, navigate to System and Security. Next, click on System. Once the System window is up, open the Device Manager.

Look under Ports (COM & LPT). You should see an open port named "Arduino UNO (COMxx)"

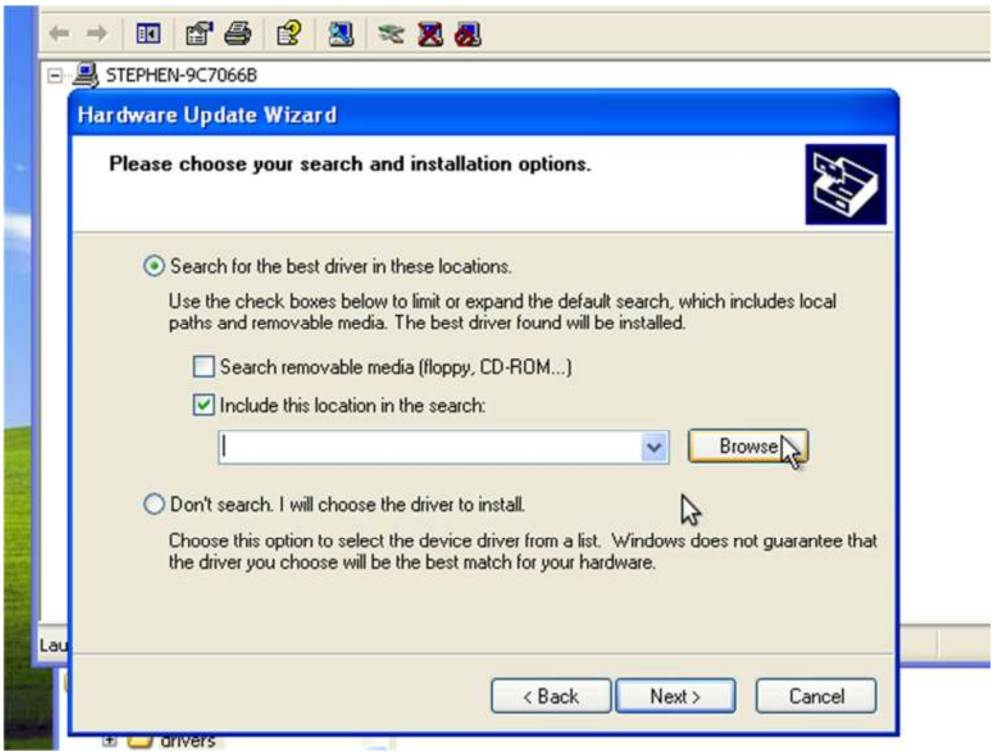


STEPHEN-9C7066B
Batteries
Computer
Disk drives
DVD/CD-ROM drives
IDE ATA/ATAPI controllers
Keyboards
Mice and other pointing devices
Network adapters
Other devices
Base System Device
Video Controller (VGA Compatible)
Ports (COM & LPT)
Arduino Uno
Printer Port (LPT1)
Sound, video and game controllers
System devices
Universal Serial Bus controllers
arduino-0021(2)
arduino-0021
drivers

Right click on the "Arduino UNO (COmxx)" port and choose the "Update Driver Software" option.



Next, choose the "Browse my computer for Driver software" option.



Please choose your search and installation options.

(•) Search for the best driver in these locations.

Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.

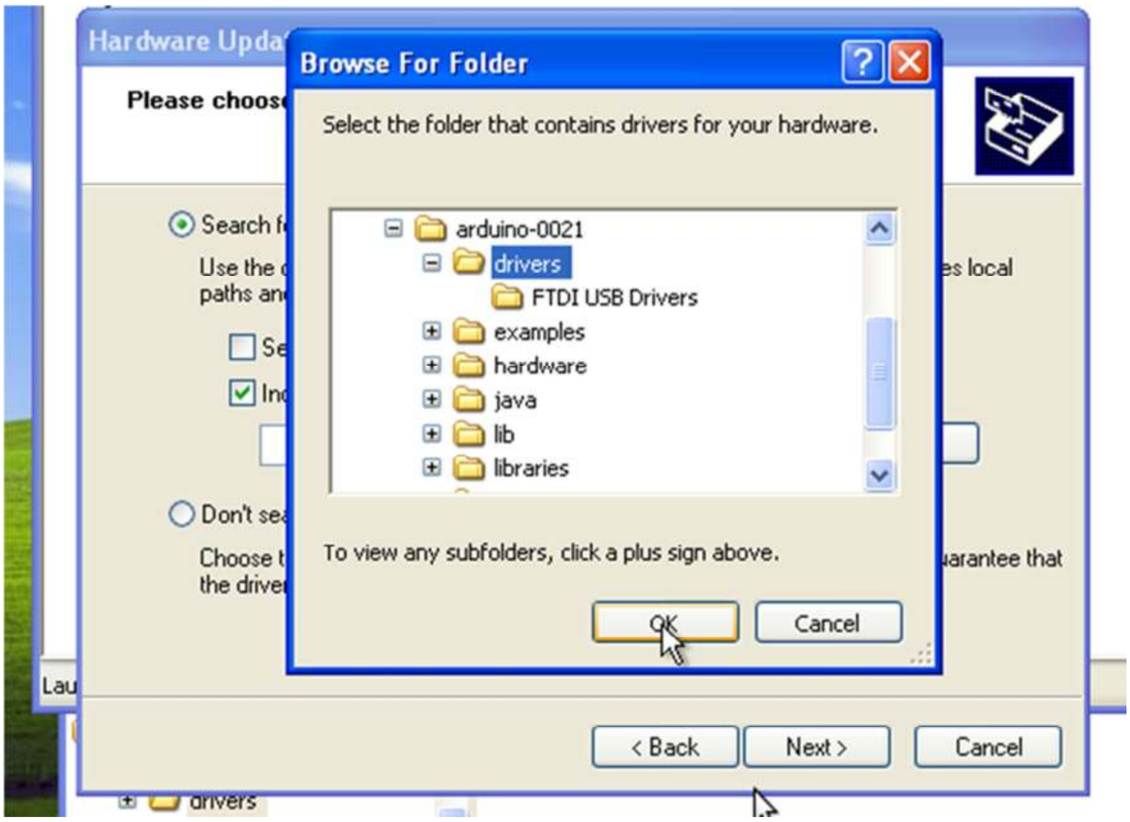
☐ Search removable media (floppy, CD-ROM...)

☑ Include this location in the search:

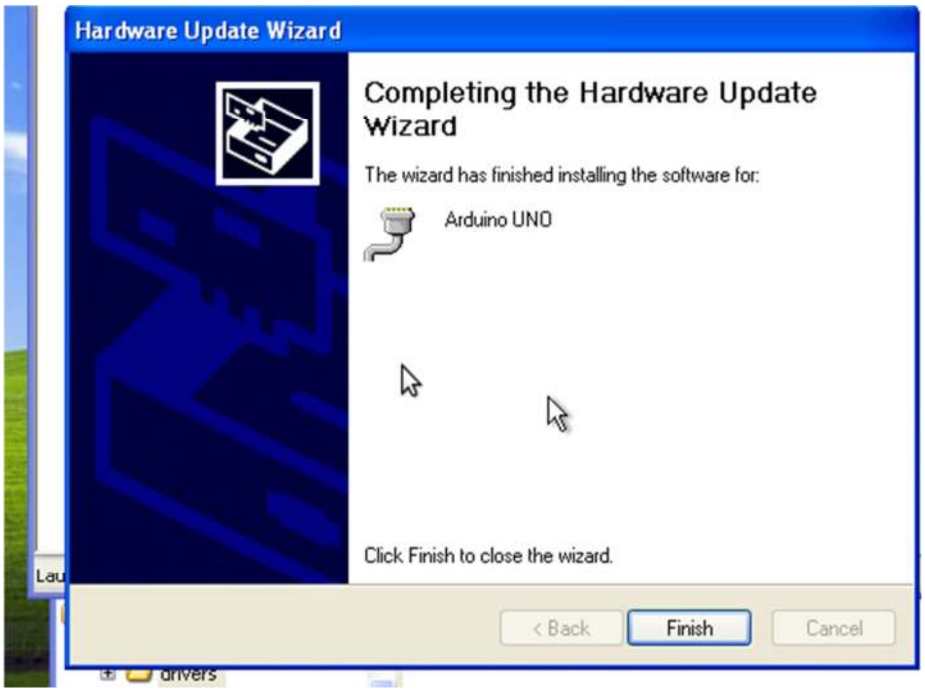
( ) Don't search. I will choose the driver to install.

Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.

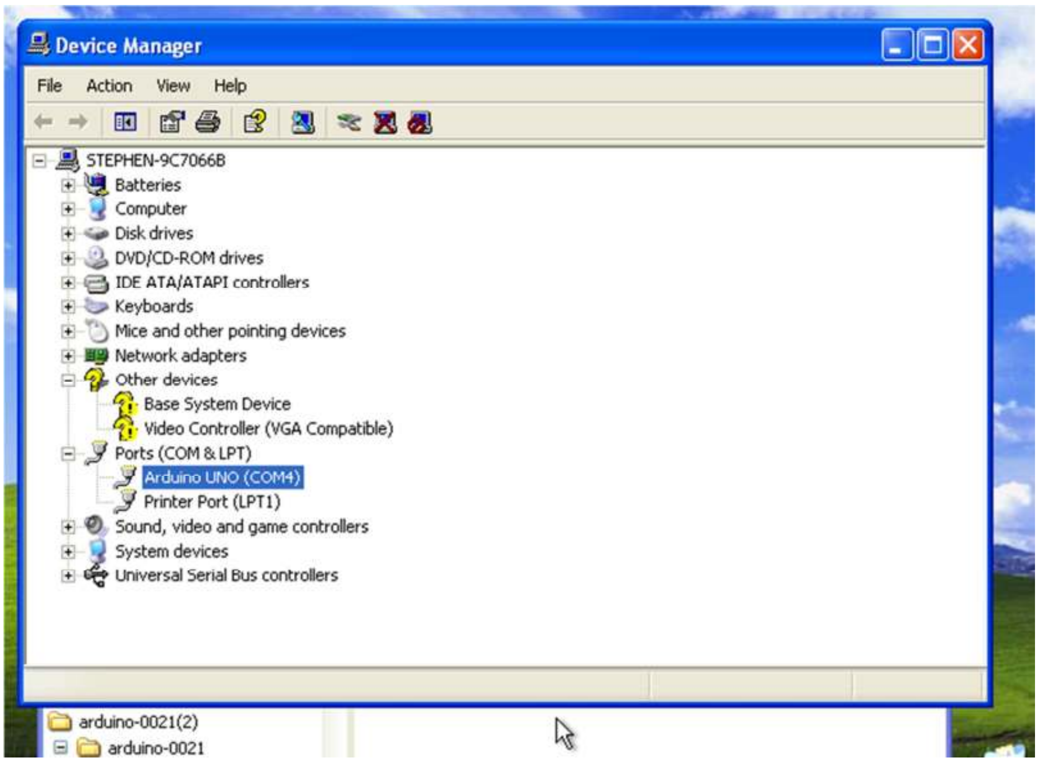
Finally, navigate to and select the driver file named "arduino.inf", located in the "Drivers" folder of the Arduino Software download (not the "**FTDI USB Drivers**" sub-directory). If you are using an old version of the IDE (1.0.3 or older), choose the Uno's driver file named "**Arduino UNO.inf**"



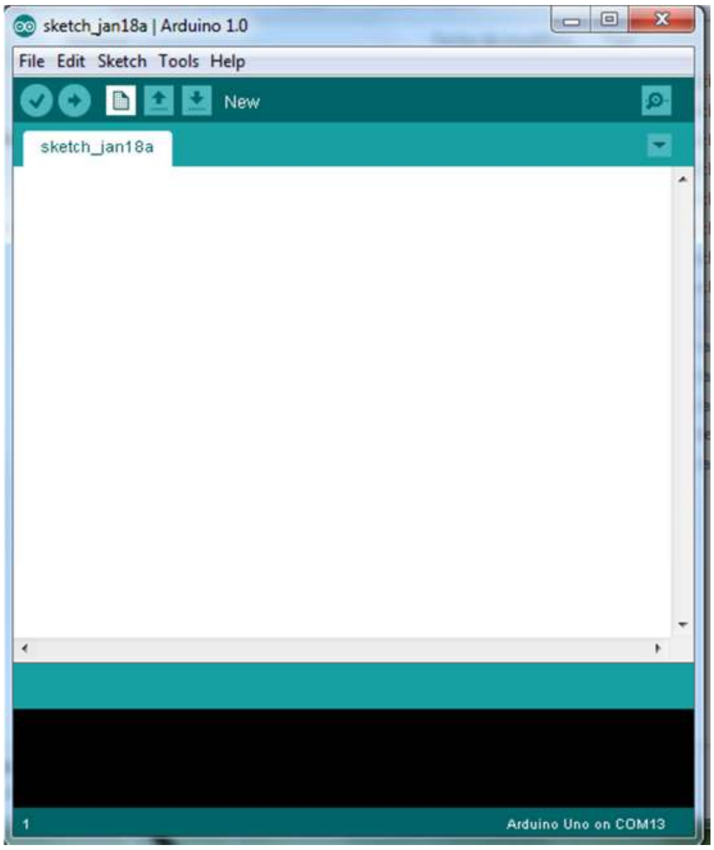
Windows will finish up the driver installation from there.


Hardware Update Wizard
Completing the Hardware Update Wizard
The wizard has finished installing the software for:
Arduino UNO
Click Finish to close the wizard.
< Back
Finish
Cancel

Now the COM port is ready to use for the Arduino and the communication between the PC (or laptop) is OK.

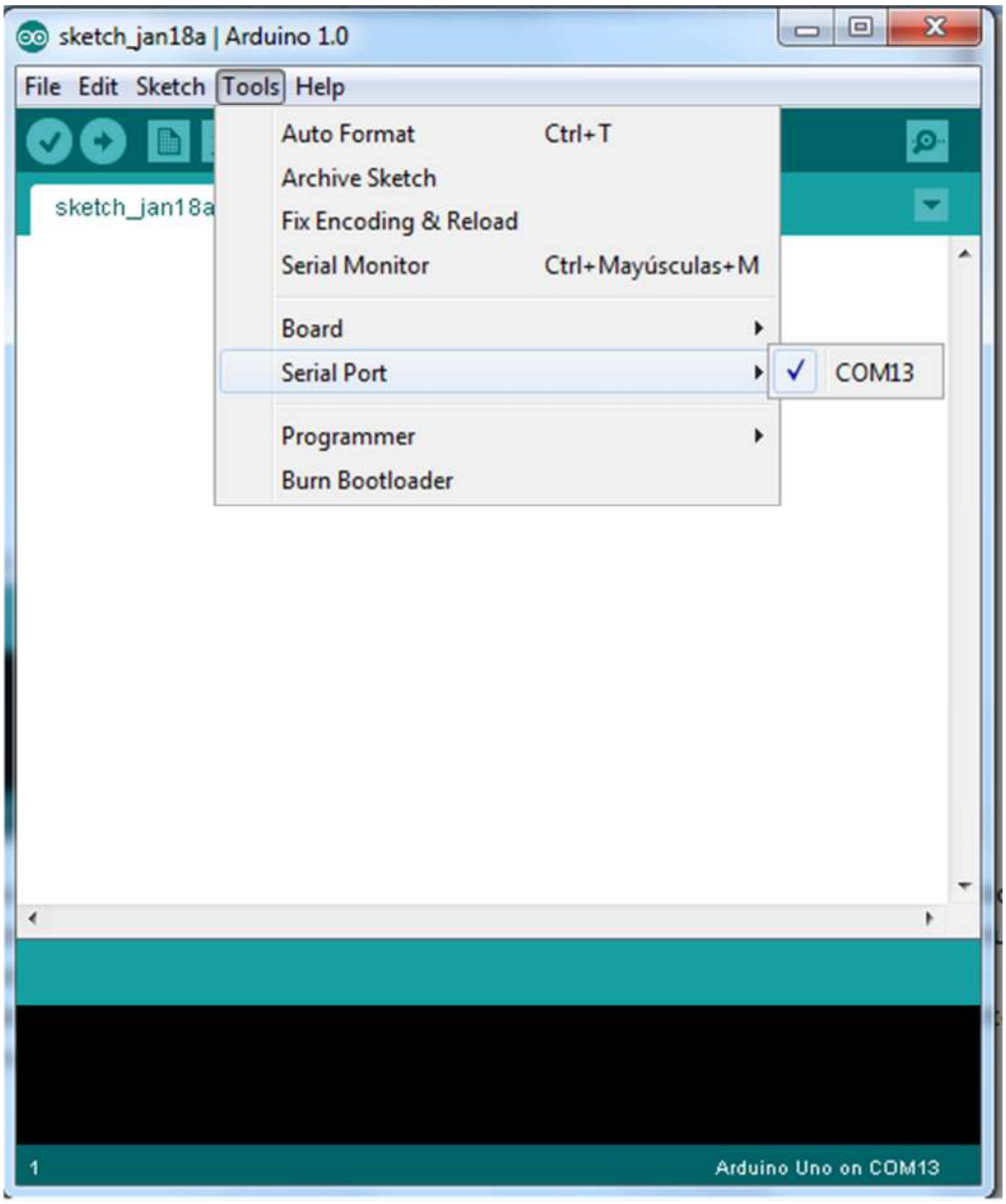


Once everything is installed it is time to open the program : go to the folder **arduino-1.0.2**, on which there is symbol named **Arduino**. When you click on it, it will be like this:



IDE of Arduino

Make sure that you are using the correct Arduino board in the program! To check this click on Tools -> Board and look that the option selected is Arduino UNO board (The Arduino model used in my case). Also make sure that you are working with the correct port number, to check this click on Tools -> Serial Port and it will show our port number. For example: COM13



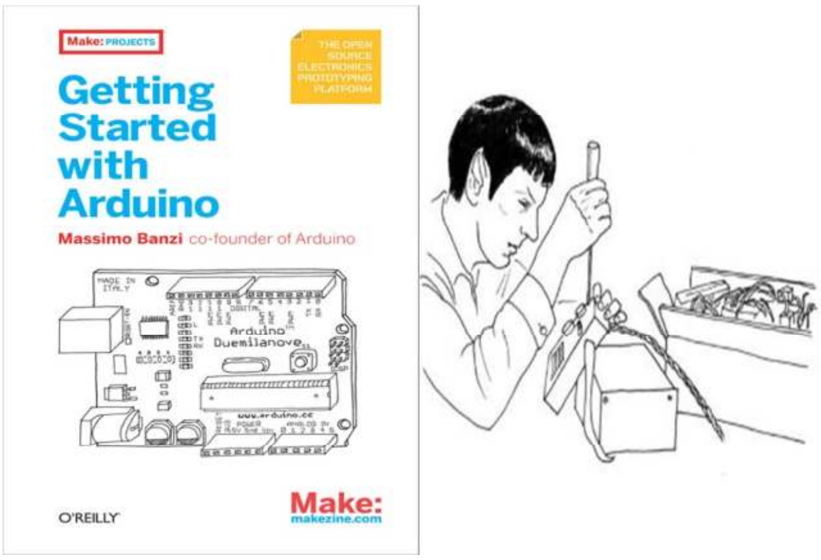
Selecting the Serial Port

Once we know that we are working with the correct port, we can start writing the program we want, we use the samples that come in the IDE or get off a program to verify that our board is working correctly.

3.1.4 Creating a sketch in Arduino

I learned programming in Arduino a couple of years ago, because of hobby electronics in my free time. Back then I bought the book "Getting Started with Arduino" which is a perfect guide to learn programming in Arduino. I also used the book during this final project work.

It is programmed through a free program that can be downloaded from the Arduino website (http://www.arduino.cc). Through the computer the code can be written, these programs use a proprietary programming language which is based on C/C++. But we can also use Arduino with other programs, such as LabVIEW, Processing (See part II of the thesis), Matlab Simulink libraries… For LabVIEW you need a interface with the Arduino toolkit.


Make: PROJECTS
THE OPEN SOURCE ELECTRONICS PROTOTYPING PLATFORM
Getting Started with Arduino
Massimo Banzi co-founder of Arduino
O'REILLY
Make: makezine.com

Getting Started with Arduino

The Arduino **integrated development environment** (IDE) is a **cross-platform** application written in **Java**, and is derived from the IDE for the **Processing programming language** . It is designed to introduce programming to artists and other newcomers unfamiliar with software development. It includes a code editor with features such as syntax highlighting and automatic indentation, and is also capable of compiling and uploading programs to the board with a single click. There is typically no need to edit **makefiles** or run programs on **a command-line interface**. A program or code written for Arduino is called a sketch.

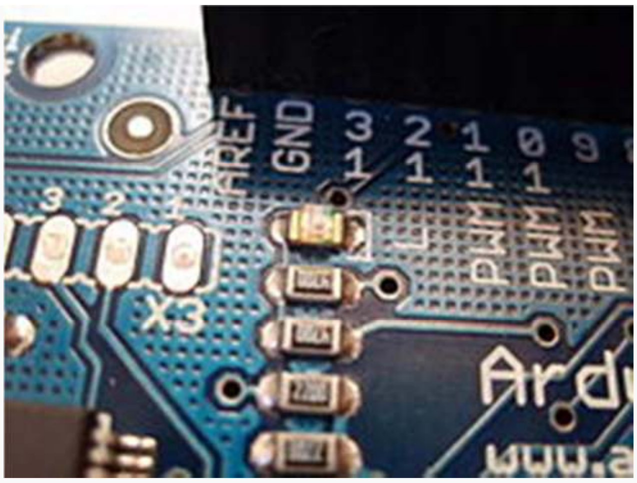
Arduino programs are written in **C** or **C++**. The Arduino IDE comes with a **software library** called "Wiring" from the original Wiring project, which makes many common input/output operations much easier. Users only need to define two functions to make a runnable **cyclic executive** program:

- `setup()`: a function run once at the start of a program that can initialize settings
- `loop()`: a function called repeatedly until the board powers off

3.1.5 Blink LED example

Example

A typical first program for a microcontroller simply blinks an LED on and off. In the Arduino environment, the user might write a program like this:



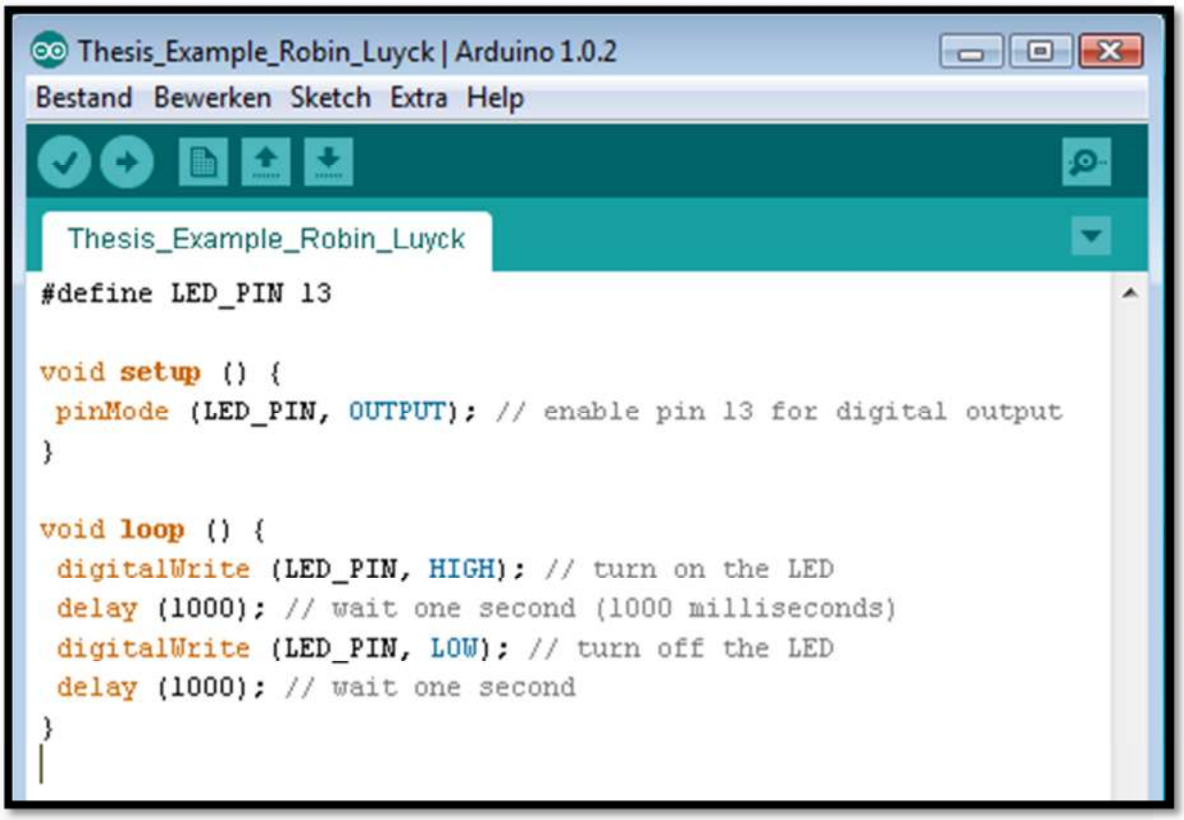
Here you can see the integrated pin 13 LED

It is a feature of most Arduino boards that they have an LED and load resistor connected between pin 13 and ground, a convenient feature for many simple tests.

The Arduino code to make the LED blink:

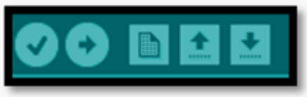
```
#define LED_PIN 13

void setup () {
 pinMode (LED_PIN, OUTPUT); // enable pin 13 for digital output
}

void loop () {
 digitalWrite (LED_PIN, HIGH); // turn on the LED
 delay (1000); // wait one second (1000 milliseconds)
 digitalWrite (LED_PIN, LOW); // turn off the LED
 delay (1000); // wait one second
}
```

The sketch to make a LED blink:


Thesis_Example_Robin_Luyck | Arduino 1.0.2
Bestand Bewerken Sketch Extra Help
Thesis_Example_Robin_Luyck

```
#define LED_PIN 13

void setup () {
 pinMode (LED_PIN, OUTPUT); // enable pin 13 for digital output
}

void loop () {
 digitalWrite (LED_PIN, HIGH); // turn on the LED
 delay (1000); // wait one second (1000 milliseconds)
 digitalWrite (LED_PIN, LOW); // turn off the LED
 delay (1000); // wait one second
}
```

The program its buttons:



1 2 3 4 5

1: Compile the sketch
2: Upload the sketch to the Arduino microcontroller
3: New sketch
4: Open the sketch which is on the Arduino microcontroller
5: Save the sketch

The previous code would not be seen by a standard C++ compiler as a valid program, so when the user clicks the "Upload to I/O board" button in the IDE, a copy of the code is written to a temporary file with an extra include header at the top and a very simple main() function at the bottom, to make it a valid C++ program. So with this you can see that it is not 100% C++ code.

3.2 LabVIEW

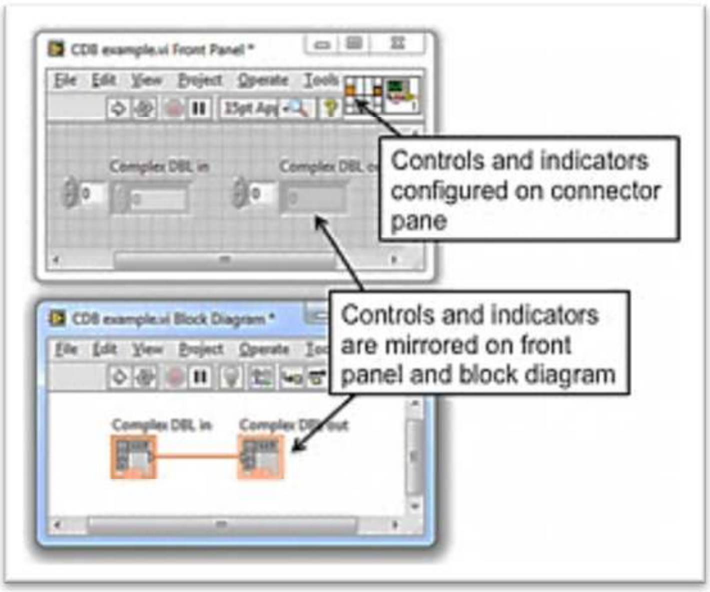
3.2.1 Intro to LabVIEW

LabVIEW (short for **Lab**oratory **V**irtual **I**nstrument **E**ngineering **W**orkbench) is a system design platform and development environment for a visual programming language from National Instruments.

Graphical programming:

LabVIEW ties the creation of user interfaces (called front panels) into the development cycle. LabVIEW programs/subroutines are called virtual instruments (**VIs**).

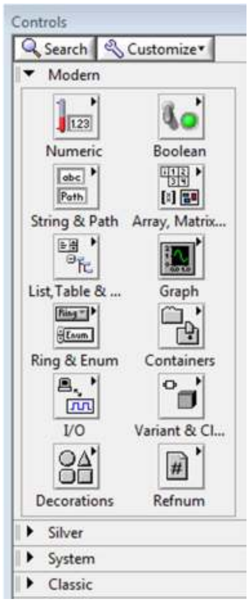
Each VI has three components: a block diagram, a front panel and a connector panel!


CDB example.vi Front Panel *
Complex DBL in
Complex DBL out
Controls and indicators configured on connector pane
CDB example.vi Block Diagram *
Controls and indicators are mirrored on front panel and block diagram
Complex DBL in
Complex DBL out

a block diagram, a front panel and a connector panel

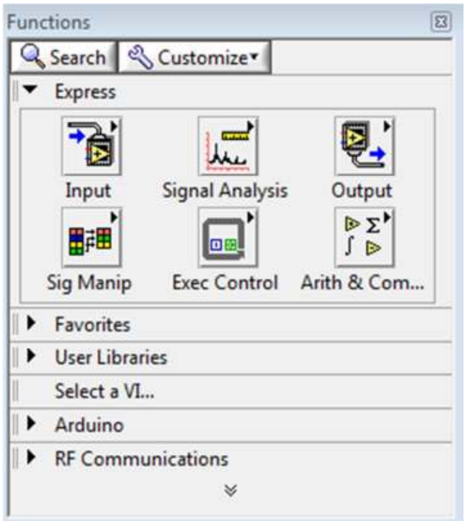
The last is used to represent the VI in the block diagrams of other, calling VIs. For example when you create a PID controller in a VI, you can insert this VI then in another VI where you need also a PID controller…

The ***front panel*** is built using controls and indicators. **Controls** are **inputs** – they allow a user to supply information to the VI. **Indicators** are **outputs** – they indicate, or display, the results based on the inputs given to the VI.

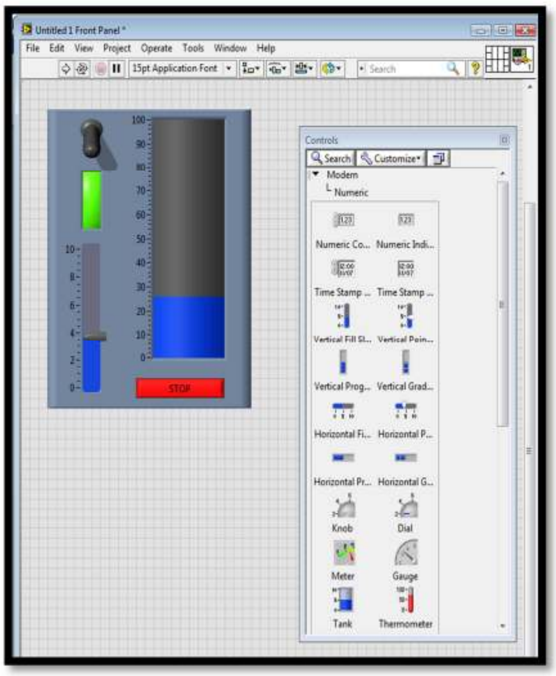


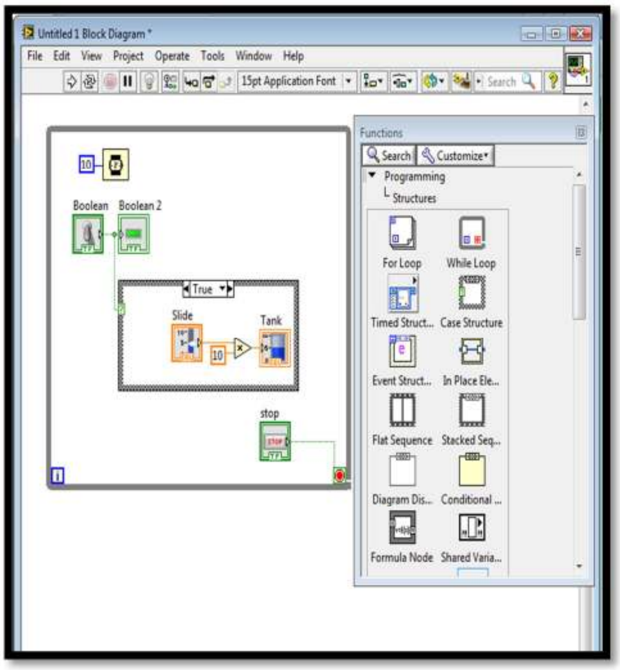
The back panel, which is a ***block diagram***, contains the **graphical source code**. All of the objects placed on the front panel will appear on the back panel as terminals. The back panel also contains structures (like a while loop) and functions which perform operations on controls and supply data to indicators.

The structures and functions are found on the **Functions palette** and can be placed on the back panel. Collectively controls, indicators, structures and functions will be referred to as nodes. Nodes are connected to one another using wires – e.g. two controls and an indicator can be wired to the addition function so that the indicator displays the sum of the two controls. Thus a virtual instrument can either be run as a program, with the front panel serving as a user interface, or, when dropped as a node onto the block diagram, the front panel defines the inputs and outputs for the given node through the connector panel. This implies each VI can be easily tested before being embedded as a subroutine into a larger program.



The graphical approach also allows non-programmers to build programs by dragging and dropping virtual representations of lab equipment with which they are already familiar. The LabVIEW programming environment, with the included examples and documentation, makes it simple to create small applications.





LabVIEW Front panel *LabVIEW Block diagram*

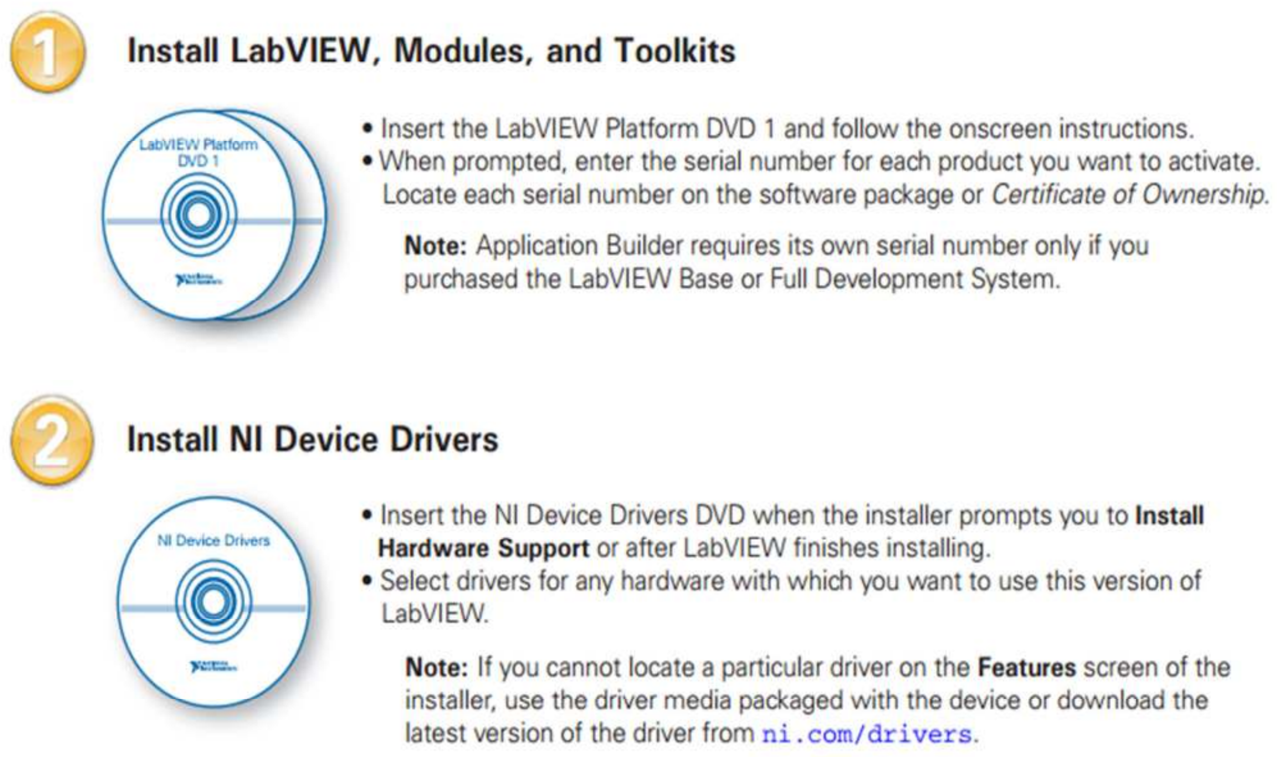
The goal of **part II** is creating a PID controller in the LabVIEW software to control the plant. Normally National Instruments offers a DAQ module (which also has analog and digital In/Outputs), which automatically can communicate with the LabVIEW software through its I/O pins.

My goal was to control the plant through an Arduino Uno microcontroller. Therefore I needed to find something to communicate between LabVIEW and the Arduino Uno.. I found a NI LabVIEW Interface for Arduino Toolkit, which makes it possible to communicate between the two. I will explain this part later, first I will give some info on the LabVIEW software itself.

3.2.2 Installation LabVIEW software

To install the LabVIEW 2012 software : the University of Valladolid owned the National Instruments Academic Software, Fall 2012, which includes LabVIEW 2012. The package contained 7 cds with the necesairry cd to install the NI LabVIEW 2012 software, the drivers to connect a NI device, NI Academic Site License (4cds) and the FPGA Module. The installing of the software is an easy step by step installation.

If there are problems with the installation, you can find answers on the National Instruments website or refer to the LabVIEW Installation Troubleshooting Guide (troubleshooting_guide.html) on the LabVIEW Platform DVD1.



1 **Install LabVIEW, Modules, and Toolkits**

- Insert the LabVIEW Platform DVD 1 and follow the onscreen instructions.
- When prompted, enter the serial number for each product you want to activate. Locate each serial number on the software package or *Certificate of Ownership*.

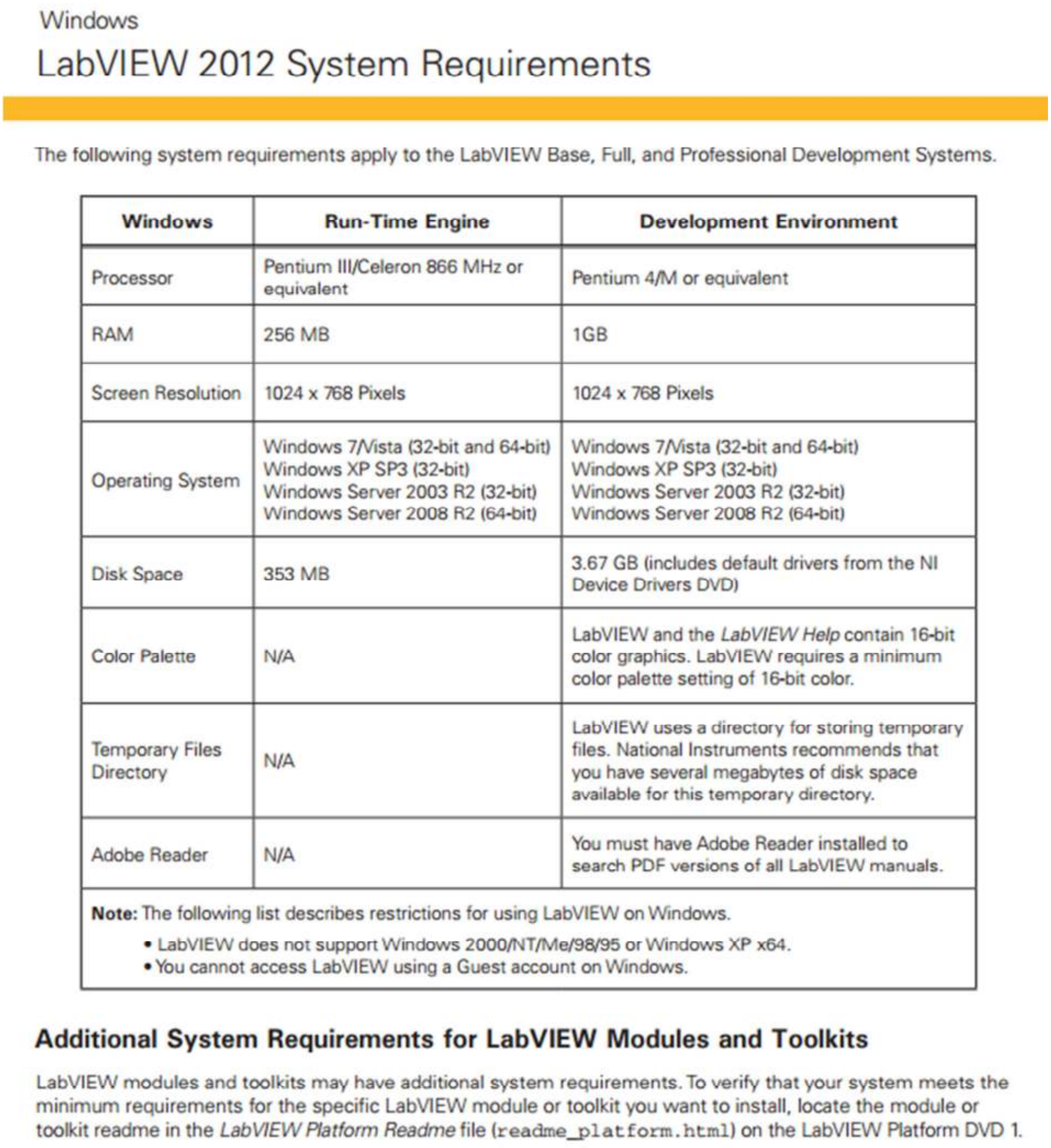
Note: Application Builder requires its own serial number only if you purchased the LabVIEW Base or Full Development System.

2 **Install NI Device Drivers**

- Insert the NI Device Drivers DVD when the installer prompts you to **Install Hardware Support** or after LabVIEW finishes installing.
- Select drivers for any hardware with which you want to use this version of LabVIEW.

Note: If you cannot locate a particular driver on the **Features** screen of the installer, use the driver media packaged with the device or download the latest version of the driver from `ni.com/drivers`.

The system requirements:



Windows

LabVIEW 2012 System Requirements

The following system requirements apply to the LabVIEW Base, Full, and Professional Development Systems.

Windows	Run-Time Engine	Development Environment
Processor	Pentium III/Celeron 866 MHz or equivalent	Pentium 4/M or equivalent
RAM	256 MB	1GB
Screen Resolution	1024 x 768 Pixels	1024 x 768 Pixels
Operating System	Windows 7/Vista (32-bit and 64-bit) Windows XP SP3 (32-bit) Windows Server 2003 R2 (32-bit) Windows Server 2008 R2 (64-bit)	Windows 7/Vista (32-bit and 64-bit) Windows XP SP3 (32-bit) Windows Server 2003 R2 (32-bit) Windows Server 2008 R2 (64-bit)
Disk Space	353 MB	3.67 GB (includes default drivers from the NI Device Drivers DVD)
Color Palette	N/A	LabVIEW and the *LabVIEW Help* contain 16-bit color graphics. LabVIEW requires a minimum color palette setting of 16-bit color.
Temporary Files Directory	N/A	LabVIEW uses a directory for storing temporary files. National Instruments recommends that you have several megabytes of disk space available for this temporary directory.
Adobe Reader	N/A	You must have Adobe Reader installed to search PDF versions of all LabVIEW manuals.

Note: The following list describes restrictions for using LabVIEW on Windows.

- LabVIEW does not support Windows 2000/NT/Me/98/95 or Windows XP x64.
- You cannot access LabVIEW using a Guest account on Windows.

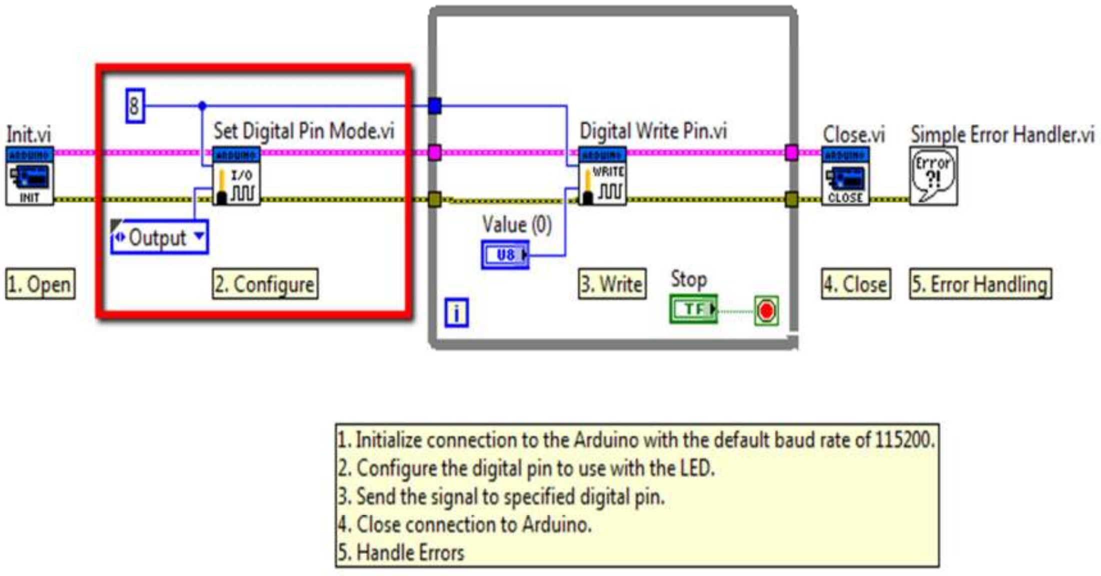
Additional System Requirements for LabVIEW Modules and Toolkits

LabVIEW modules and toolkits may have additional system requirements. To verify that your system meets the minimum requirements for the specific LabVIEW module or toolkit you want to install, locate the module or toolkit readme in the *LabVIEW Platform Readme* file (`readme_platform.html`) on the LabVIEW Platform DVD 1.

3.3 NI LabVIEW interface for Arduino toolkit

3.3.1 Intro to the toolkit

The NI LabVIEW Interface for Arduino Toolkit helps you easily interface with the Arduino microcontroller using LabVIEW. With this toolkit and LabVIEW, you can control or acquire data from the Arduino microcontroller. Once the information is in LabVIEW, analyze it using the hundreds of built-in LabVIEW libraries, develop algorithms to control the Arduino hardware.



1. Initialize connection to the Arduino with the default baud rate of 115200.
2. Configure the digital pin to use with the LED.
3. Send the signal to specified digital pin.
4. Close connection to Arduino.
5. Handle Errors

An example of using the toolkit from Arduino in LabVIEW

A sketch for the Arduino microcontroller acts as an I/O engine that interfaces with LabVIEW VIs through a serial connection. This helps you quickly move information from Arduino pins to LabVIEW without adjusting the communication, synchronization, or even a single line of C code. Using the common Open, Read/Write, Close convention in LabVIEW, you can access the digital, analog, pulse-width-modulated of the Arduino microcontroller.

3.3.2 Step by step starterguide for the Installation

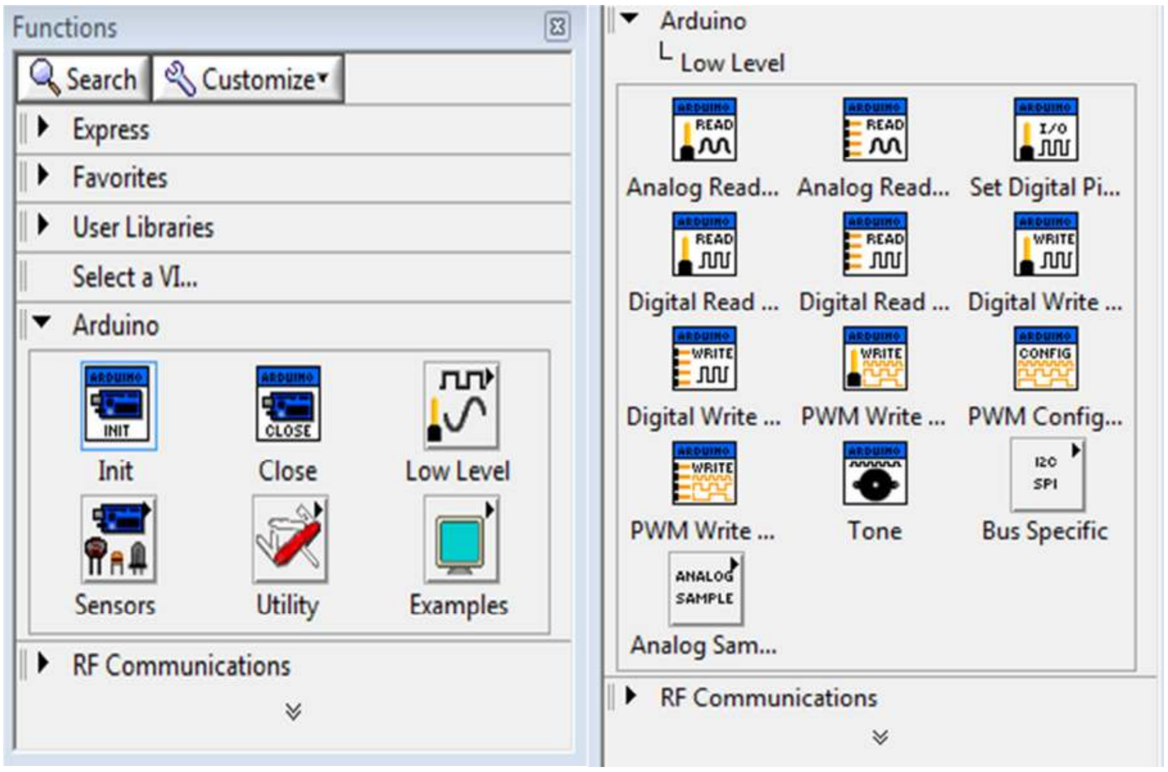
After some research I found a really good website which explains all the installations that have to be done to make the connection. It contains **a video tutorial** and **a step by step startup guide** where everything is explained.

The link is: http://vishots.com/getting-started-with-the-labview-interface-for-arduino/

The labview interface for arduino is a vi based API that was written and distributed by national instruments. The code also includes and arduino embedded program which must be downloaded to the device. This program which runs on the Arduino, responds to commands sent on the USB bus from the LabVIEW program. It then sends back data to the the computer via the USB.

The LabVIEW VIs provided, allow you to read back the analog inputs, control the digital IO lines and use several other features of the Arduino hardware.

Note: The Arduino microcontroller must be connected to the computer with LabVIEW through USB.



Arduino Toolkit in LabVIEW

3.4 Step by Step Startup Guide

3.4.1 Steps

Here is a step by step process to get up and running with Arduino and LabVIEW:

1. **Purchase an Arduino board.** Make sure you have a USB cable.

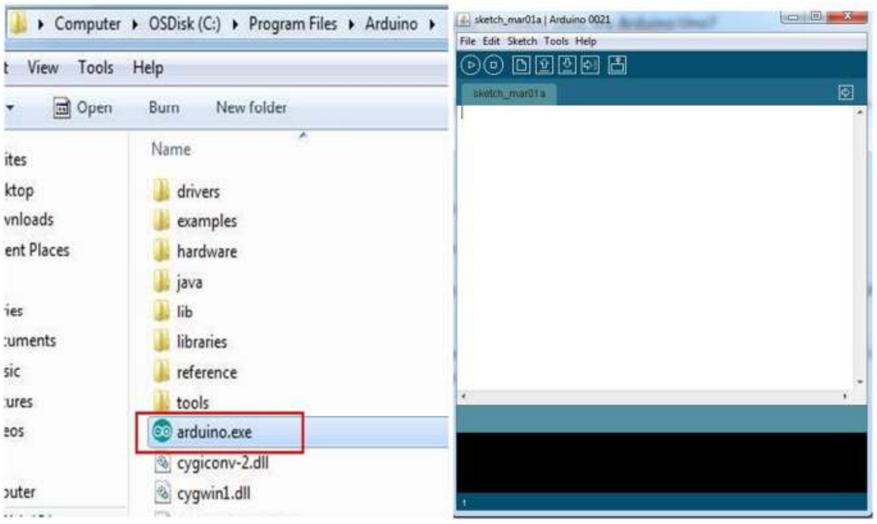
2. **Make sure you have LabVIEW 2009 or newer installed.** The VIs that are included in the LIFA are saved in LV 2009, so this is the version of LV that you must have to be able to use the LIFA

3. **Install NI-VISA Drivers.** To LabVIEW, the arduino appears as a serial instrument device. To communicate with serial instruments in LabVIEW, you need to have the latest version of the NI-VISA driver. You can get the latest NI-VISA drivers here: (http://search.ni.com/nisearch/app/main/p/bot/no/ap/tech/lang/en/pg/1/sn/catnav:du,n8:3.1637,ssnav:sup/)
   Make sure to select the latest Windows version.

4. **Install the Arduino IDE and drivers for Windows.** You can download them from the Arduino website here:
   http://arduino.cc/en/Main/Software
   - Step-by-step instructions for setting up the Arduino software on Windows can be found here:
   http://arduino.cc/en/Guide/Windows

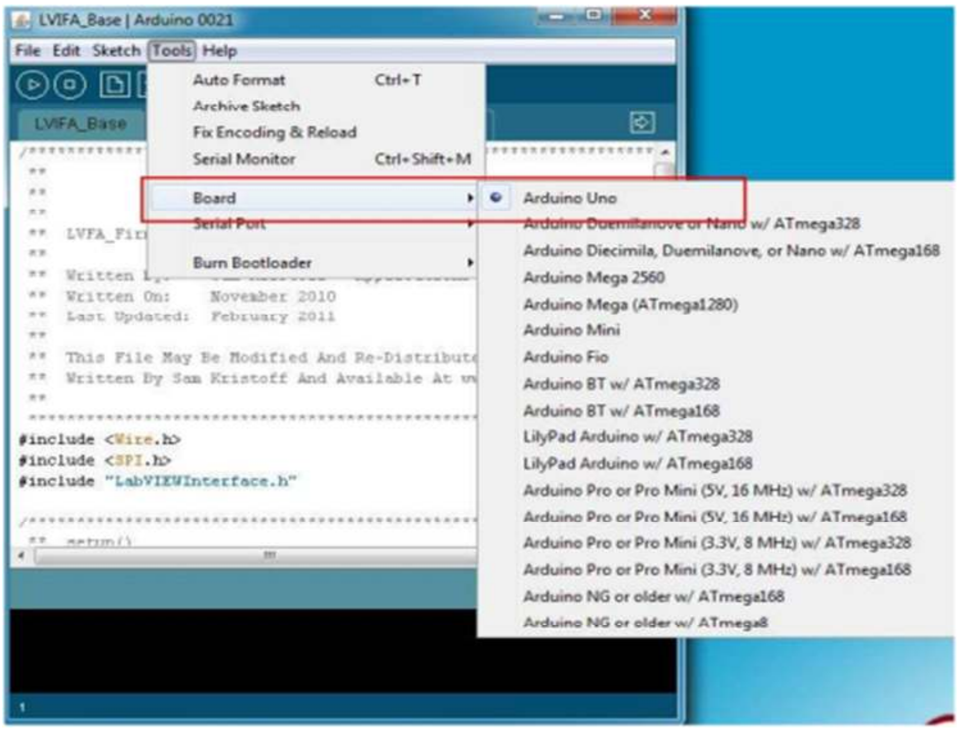
5. **Install the LIFA.** The LIFA (LabVIEW Interface for Arduino) is available as a VI package through the LabVIEW Tools Network. You must first install VIPM:
   http://jki.net/vipm/download
   Once VIPM is installed, click on this link:
   https://lumen.ni.com/nicif/us/evaltlktlvardio/content.xhtml
   to get and install the LIFA under LabVIEW 2009+.

6. **Upload the sketch 'LIFA_Base.pde' to the Arduino.** The LIFA comes with a sketch program that must be uploaded to the Arduino before you can use the VIs to communicate with it. You must use the Arduino IDE software (which you installed in step 4) to do this:



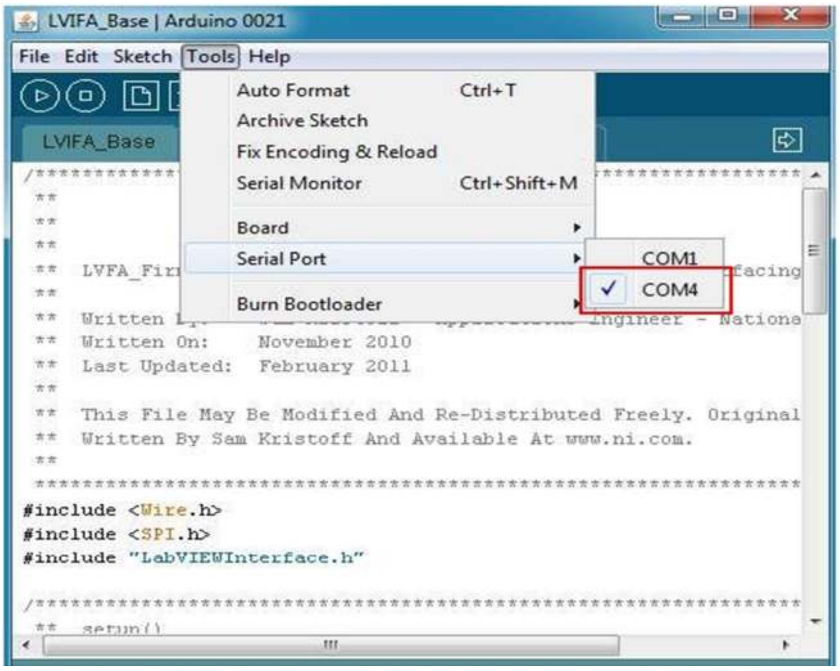
The Arduino IDE will open. Click File»Open and browse to LVIFA_Base.pde found in C:\Program Files\National Instruments\LabVIEW 2010\vi.lib\LabVIEW Interface for Arduino\Firmware\LVIFA_Base

Choose the Arduino board type by clicking **Tools»Board»Arduino Uno**


LVIFA_Base | Arduino 0021
File Edit Sketch Tools Help
LVIFA_Base
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Board
Serial Port
Burn Bootloader
Arduino Uno
Arduino Duemilanove or Nano w/ ATmega328
Arduino Diecimila, Duemilanove, or Nano w/ ATmega168
Arduino Mega 2560
Arduino Mega (ATmega1280)
Arduino Mini
Arduino Fio
Arduino BT w/ ATmega328
Arduino BT w/ ATmega168
LilyPad Arduino w/ ATmega328
LilyPad Arduino w/ ATmega168
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega168
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega168
Arduino NG or older w/ ATmega168
Arduino NG or older w/ ATmega8

```
/************
**
**
**
**  LVFA_Fir
**
**  Written
**  Written On:    November 2010
**  Last Updated:  February 2011
**
**  This File May Be Modified And Re-Distribute
**  Written By Sam Kristoff And Available At w
**
*********************************************
#include <Wire.h>
#include <SPI.h>
#include "LabVIEWInterface.h"

/*********************************************
**  setup()
```

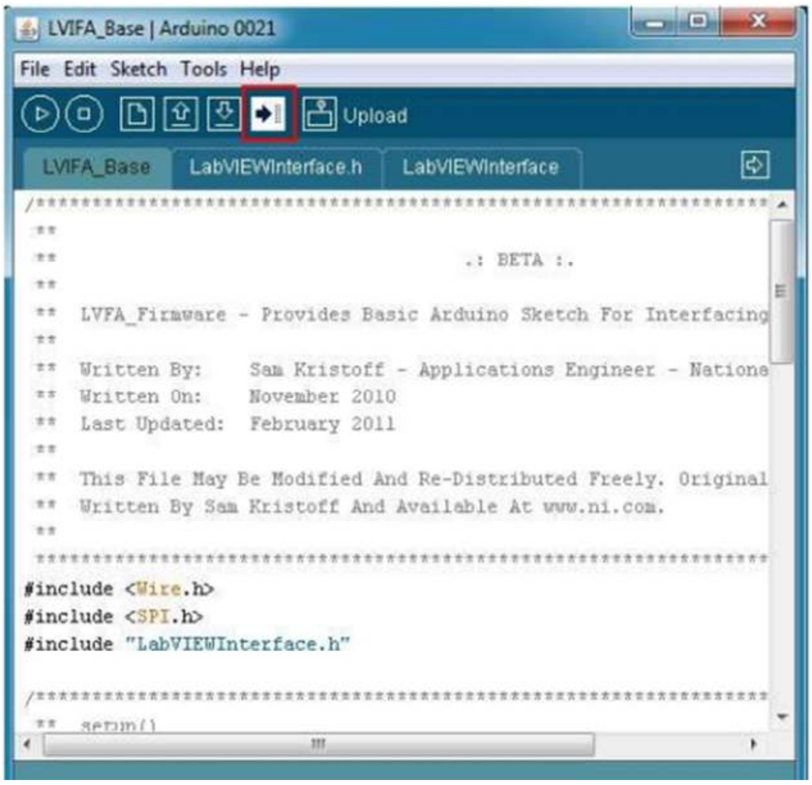
Choose the COM port by clicking **Tools»Serial Port** and choosing the COM port that corresponds to your Arduino Uno.



```
/************
**
**
**
**  LVFA_Fir                                                       facing
**
**  Written                                       Engineer - Nationa
**  Written On:      November 2010
**  Last Updated:    February 2011
**
**  This File May Be Modified And Re-Distributed Freely. Original
**  Written By Sam Kristoff And Available At www.ni.com.
**
*********************************************************************
#include <Wire.h>
#include <SPI.h>
#include "LabVIEWInterface.h"

/********************************************************************
**  setup()
```

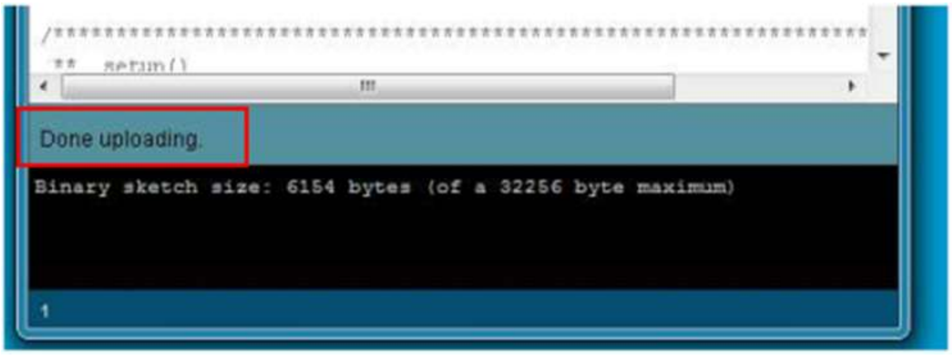
You can determine the COM port that corresponds to your Arduino Uno using the device manager by **clicking Start»Control Panel»Device Manager and expanding Ports(LTP & COM).**

Click the Upload button to upload the firmware to the Arduino Uno.



```
/********************************************************************
**
**                          .: BETA :.
**
**  LVFA_Firmware - Provides Basic Arduino Sketch For Interfacing
**
**  Written By:     Sam Kristoff - Applications Engineer - Nationa
**  Written On:     November 2010
**  Last Updated:   February 2011
**
**  This File May Be Modified And Re-Distributed Freely. Original
**  Written By Sam Kristoff And Available At www.ni.com.
**
*********************************************************************
#include <Wire.h>
#include <SPI.h>
#include "LabVIEWInterface.h"

/********************************************************************
**  setup()
```

The Arduino IDE should report Done Uploading when the firmware has been successfully uploaded to the Arduino.



```
/*********************************************************************
**  setup()
```

Done uploading.

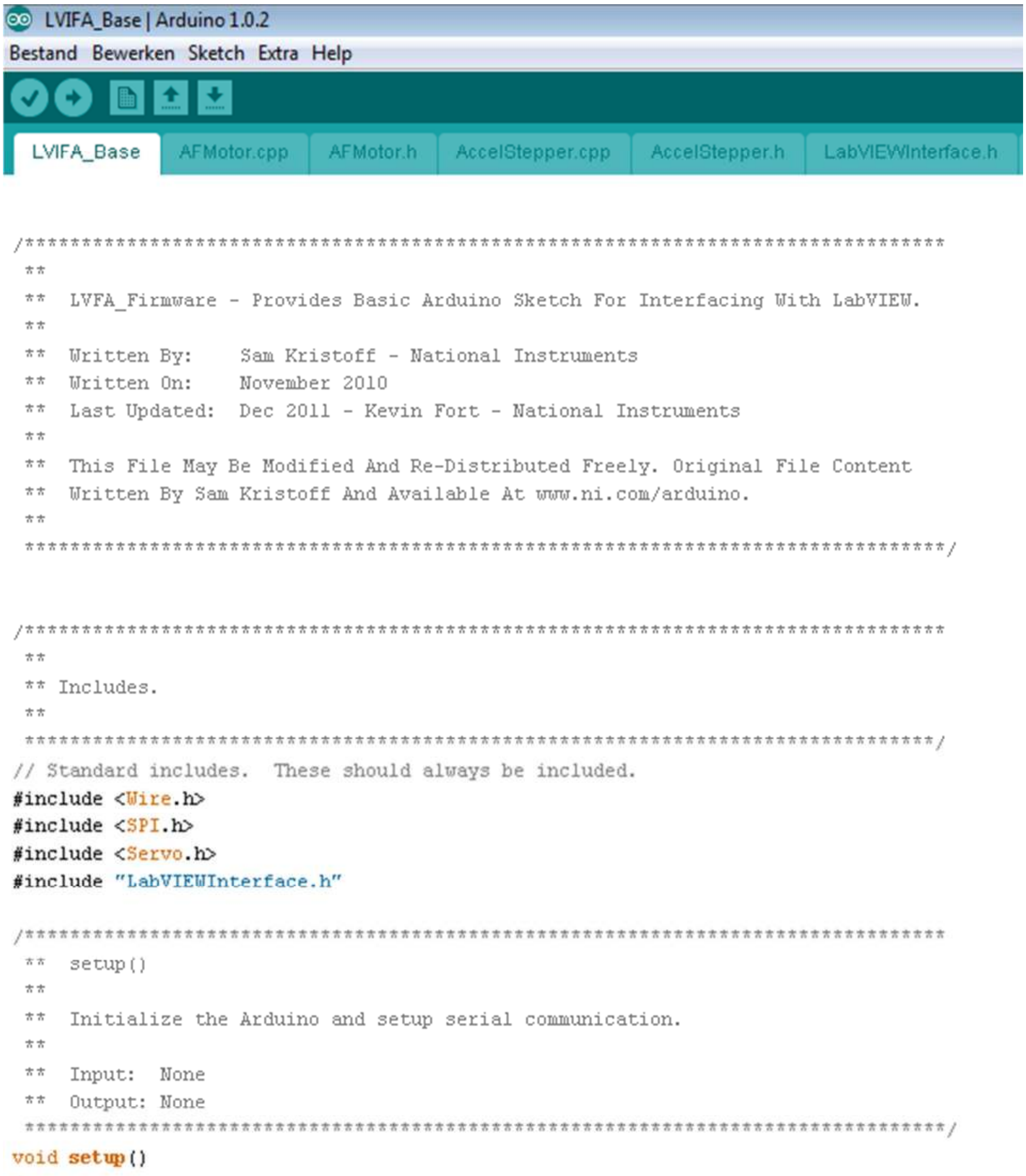
```
Binary sketch size: 6154 bytes (of a 32256 byte maximum)
```

1

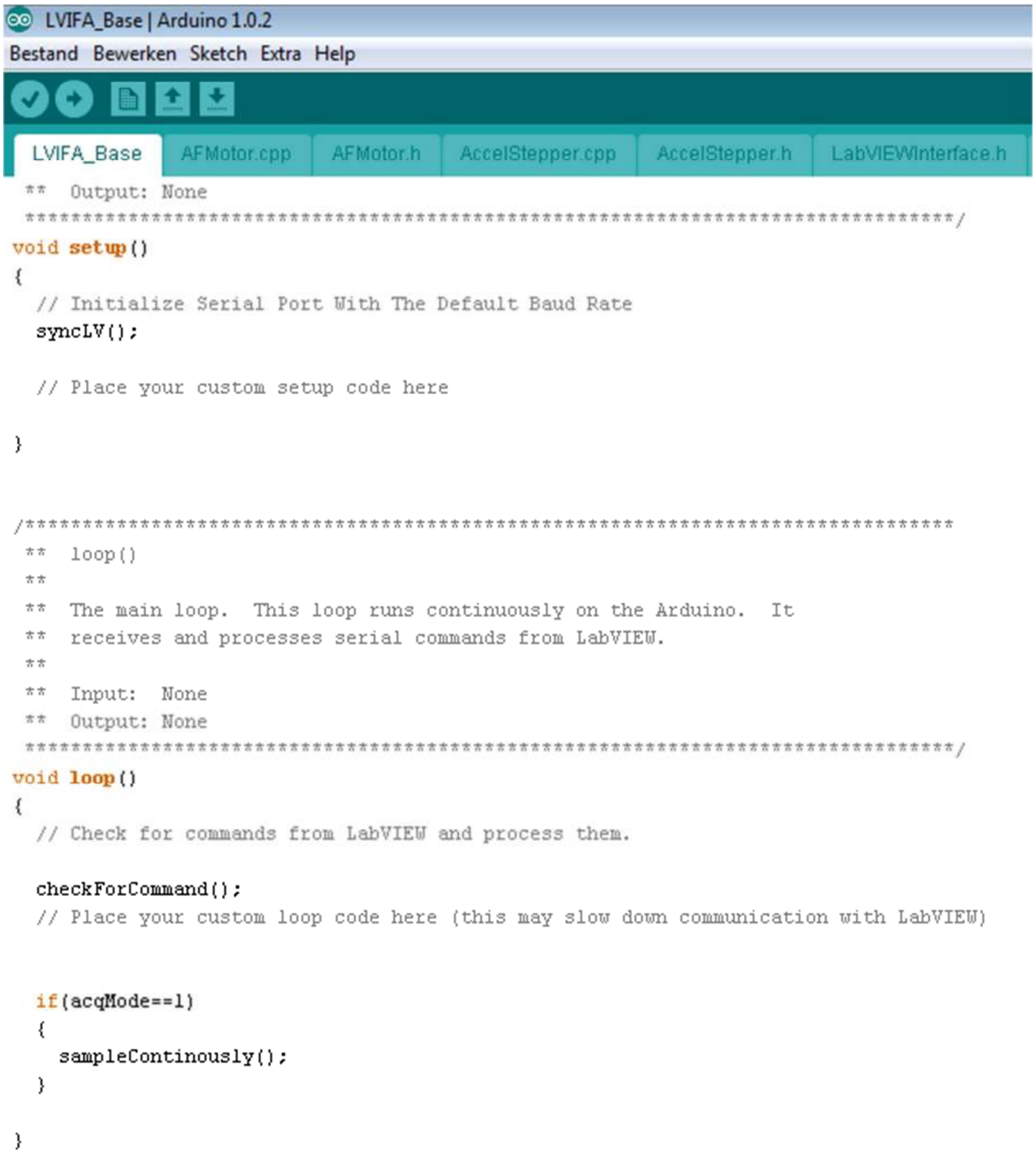
7. The sketch is located at:
   - C:\Program Files\National Instruments\LabVIEW 2010\vi.lib\LabVIEW Interface for Arduino\Firmware\LVIFA_Base\LVIFA_Base.pde
   -
8. **Start Playing.** For help, you can join the Arduino LabVIEW community at ni.com/arduino

3.4.2 Connection setup with the Arduino sketch

The sketch that you have to upload to communicate between the Arduino I/O and LabVIEW:



```
/*********************************************************************************
 **
 **  LVFA_Firmware - Provides Basic Arduino Sketch For Interfacing With LabVIEW.
 **
 **  Written By:    Sam Kristoff - National Instruments
 **  Written On:    November 2010
 **  Last Updated:  Dec 2011 - Kevin Fort - National Instruments
 **
 **  This File May Be Modified And Re-Distributed Freely. Original File Content
 **  Written By Sam Kristoff And Available At www.ni.com/arduino.
 **
 *********************************************************************************/


/*********************************************************************************
 **
 ** Includes.
 **
 ********************************************************************************/
// Standard includes.  These should always be included.
#include <Wire.h>
#include <SPI.h>
#include <Servo.h>
#include "LabVIEWInterface.h"

/*********************************************************************************
 **  setup()
 **
 **  Initialize the Arduino and setup serial communication.
 **
 **  Input:  None
 **  Output: None
 *********************************************************************************/
void setup()
```



```
 **  Output: None
 ********************************************************************************/
void setup()
{
  // Initialize Serial Port With The Default Baud Rate
  syncLV();

  // Place your custom setup code here

}


/*********************************************************************************
 **  loop()
 **
 **  The main loop.  This loop runs continuously on the Arduino.  It
 **  receives and processes serial commands from LabVIEW.
 **
 **  Input:  None
 **  Output: None
 ********************************************************************************/
void loop()
{
  // Check for commands from LabVIEW and process them.

  checkForCommand();
  // Place your custom loop code here (this may slow down communication with LabVIEW)


  if(acqMode==1)
  {
    sampleContinously();
  }

}
```

→ When the sketch is compiled and uploaded, the communication between the Arduino I/O and the toolkit in LabVIEW should work perfectly.

3.4.3 Practical example with the toolset

I made This LabVIEW program example to control 4 relays through LabVIEW and an Arduino shields output. How the program works is specified on the front panel.

As you can see, I created the following things on the **front panel**:

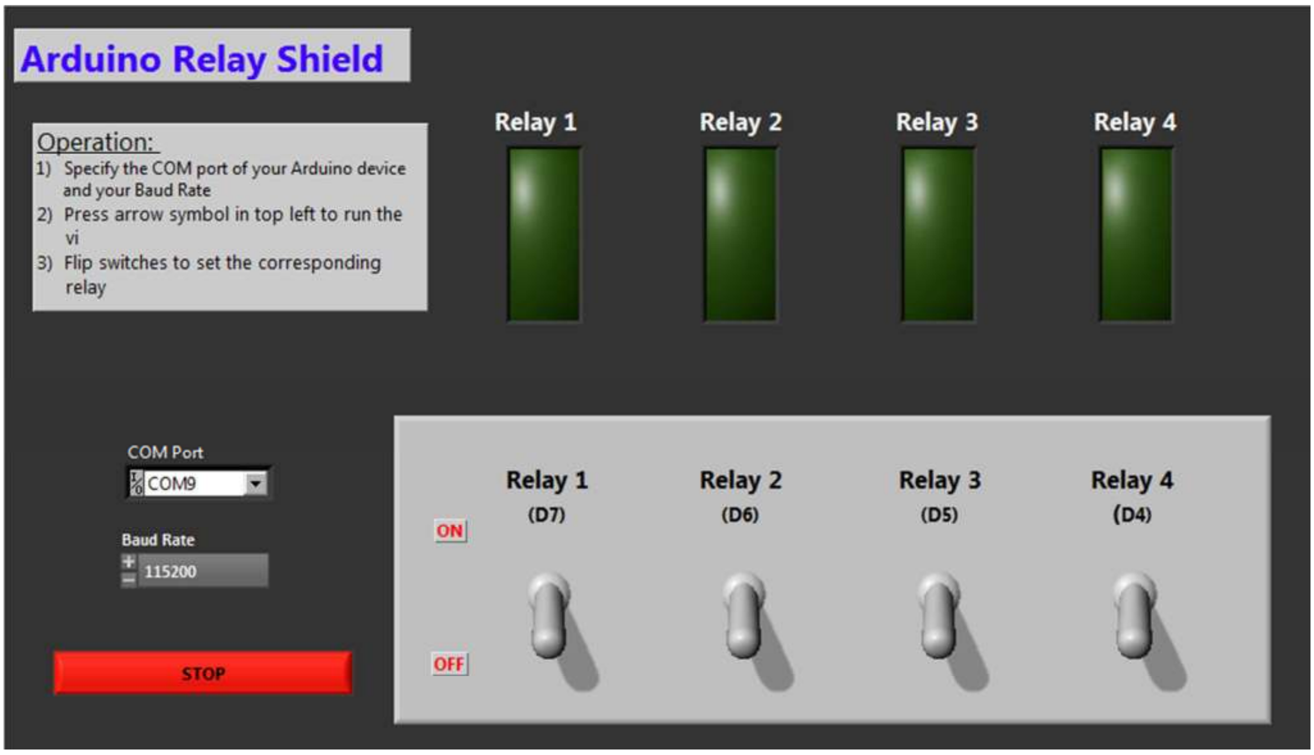
➔ 4 green indicater LED's to show the relay status;

➔ 4 switches to control the relays individual;

➔ A numeric controller to choose the COM port;

➔ A numeric controller to choose the Baud Rate;

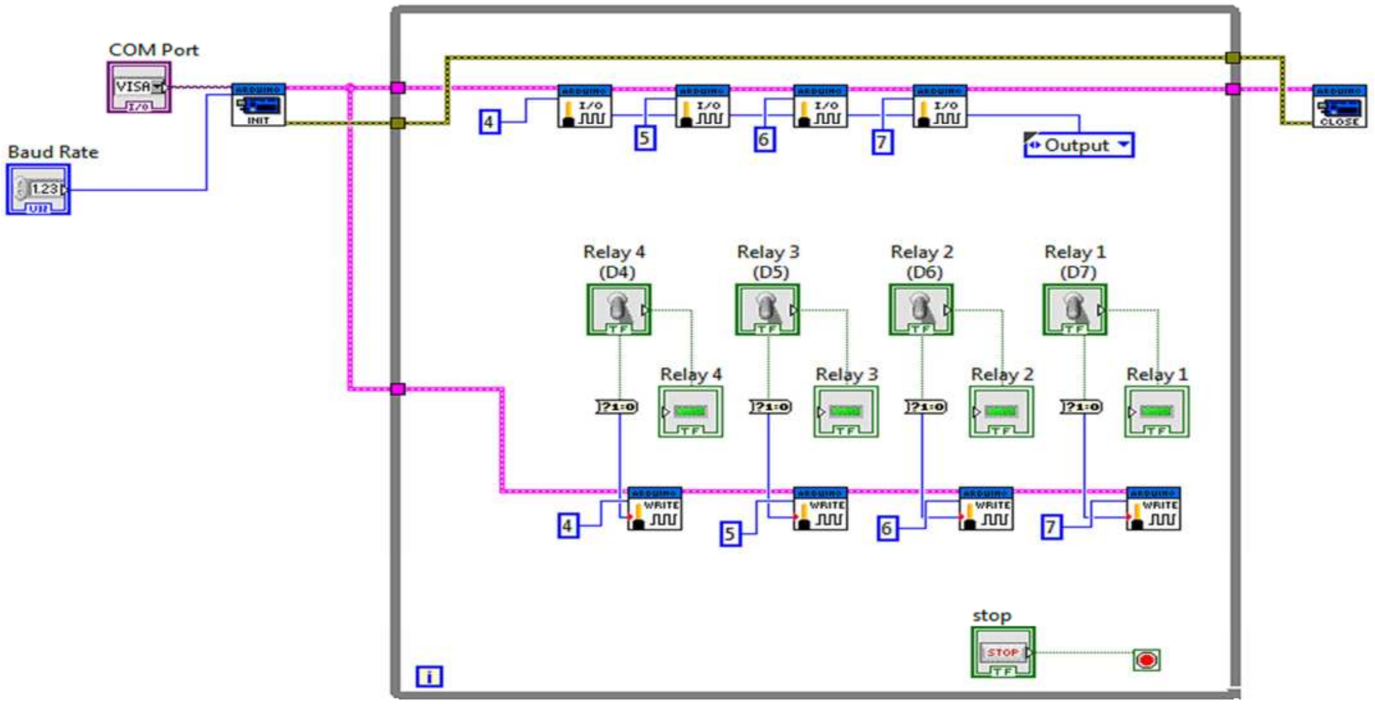
➔ A red STOP button to stop the "while loop" and so the system;


Arduino Relay Shield

Operation:
1) Specify the COM port of your Arduino device and your Baud Rate
2) Press arrow symbol in top left to run the vi
3) Flip switches to set the corresponding relay

The front panel

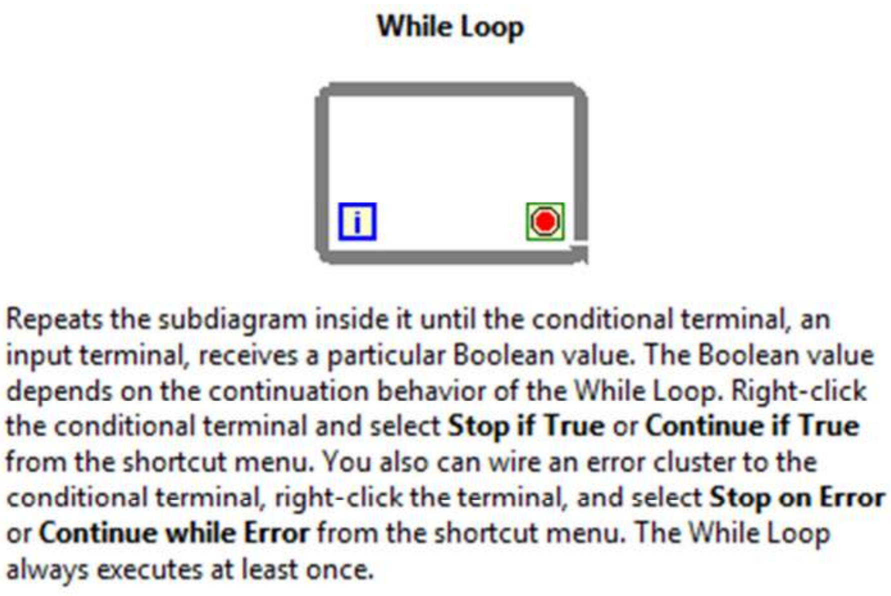
Because of adding the Controllers and Indicators on the front panel, they will automatically be created in the **block diagram**:



The block diagram

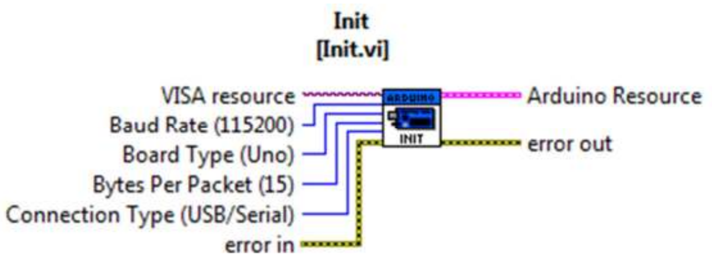
The things I had to add in the block diagram after creating the front panel are:

➔ ***A while loop***


While Loop

Repeats the subdiagram inside it until the conditional terminal, an input terminal, receives a particular Boolean value. The Boolean value depends on the continuation behavior of the While Loop. Right-click the conditional terminal and select **Stop if True** or **Continue if True** from the shortcut menu. You also can wire an error cluster to the conditional terminal, right-click the terminal, and select **Stop on Error** or **Continue while Error** from the shortcut menu. The While Loop always executes at least once.

➔ ***Arduino INIT vi***

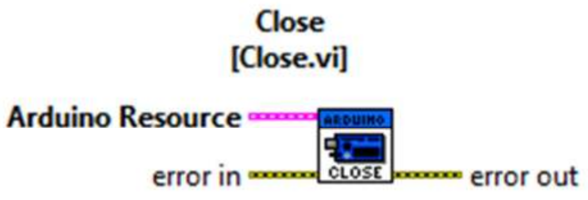


Initializes a connection to an Arduino running the LabVIEW Interface for Arduino sketch.

Note: The baud rate specified must match the baud rate defined by DEFAULTBAUDRATE in the Arduino firmware.

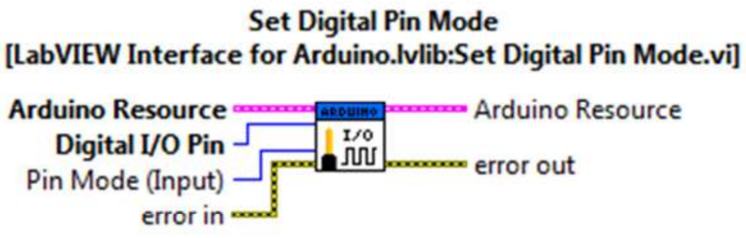
Note: The Bytes Per Packet input must match the COMMANDLENGTH specified in the Arduino firmware.

➔ ***Arduino CLOSE vi***



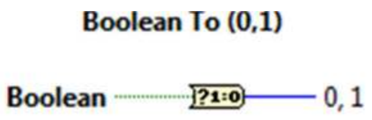
Closes the active connection to an Arduino.

➔ ***Set Digital Pin Mode vi***



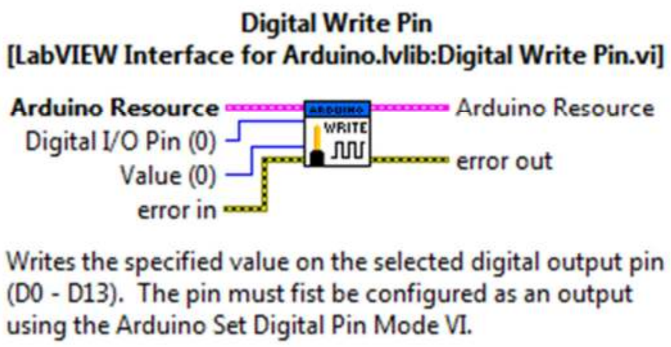
Configures the specified digital I/O pin (D0 - D13) as either input or output.

➔ ***Boolean To (0,1) converter***



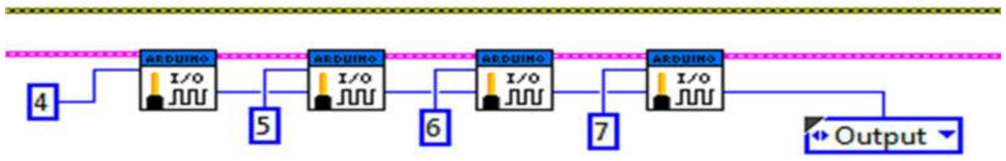
Converts a Boolean FALSE or TRUE value to a 16-bit integer with a value of 0 or 1, respectively.

➔ ***Digital Write Pin vi***

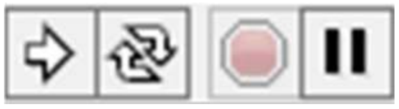

Digital Write Pin
[LabVIEW Interface for Arduino.lvlib:Digital Write Pin.vi]

Writes the specified value on the selected digital output pin (D0 - D13). The pin must fist be configured as an output using the Arduino Set Digital Pin Mode VI.

➔ ***Create constants to tell which pin from the Arduino + indicator to choose Input or Output***



To run the program, make sure the Arduino sketch to communicate between LabVIEW and Arduino is uploaded into the Arduino and running, then run the LabVIEW VI by clicking on the Run Continuously button in the left corner of the screen:



1 **2** **3** **4**

1: Run

2: Run continuously

3: Stop

4: Pauze

3.5 Processing

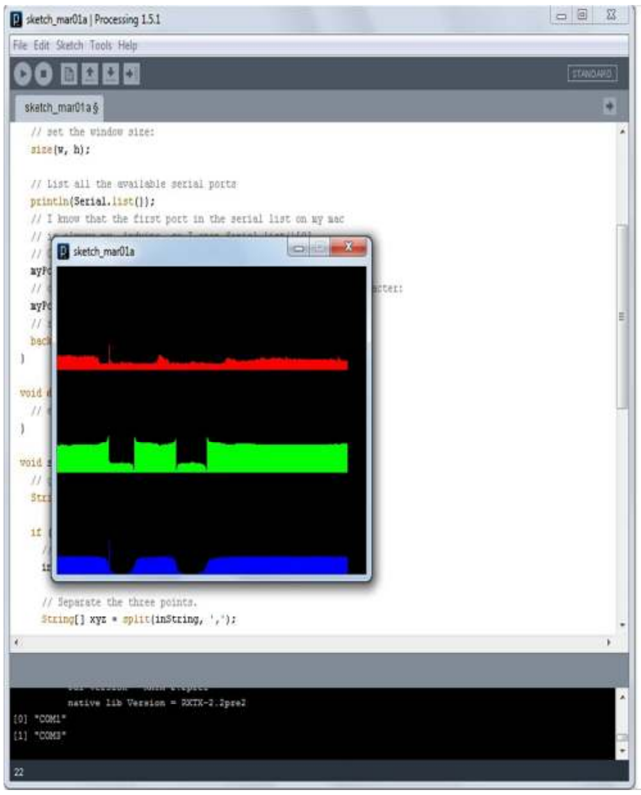
3.5.1 Intro to Processing

Processing is an open source programming language and environment for people who want to create images, animations, and interactions. Initially developed to serve as a software sketchbook and to teach fundamentals of computer programming within a visual context, Processing also has evolved into a tool for generating finished professional work. Today, there are tens of thousands of students, artists, designers, researchers, and hobbyists who use Processing for learning, prototyping, and production

» Free to download and open source

» Interactive programs using 2D, 3D or PDF output

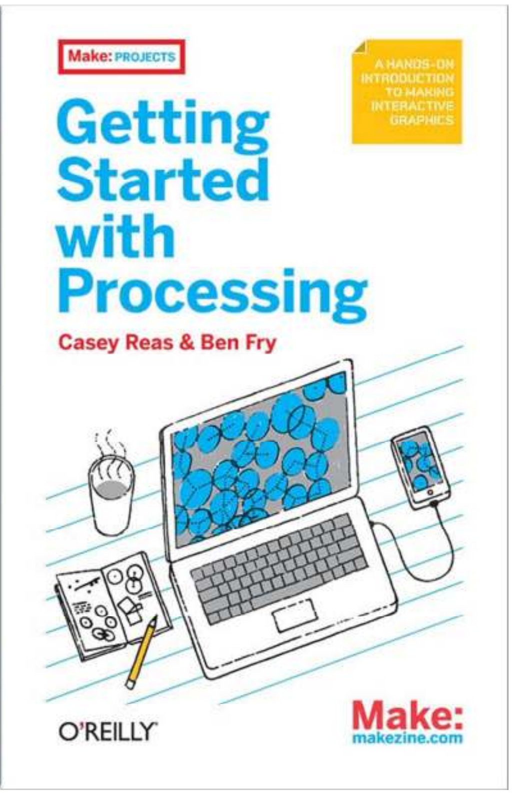
Example + the book on using Processing (“**Getting Started with Processing**”):



```
  // set the window size:
  size(w, h);

  // List all the available serial ports
  println(Serial.list());
  // I know that the first port in the serial list on my mac
  //
  //
  myPo
  //                                              acter:
  myPo
  //
  back
}

void d
  //
}

void s
  //
  Stri

  if (
    //
    in

    // Separate the three points.
    String[] xyz = split(inString, ',');
```

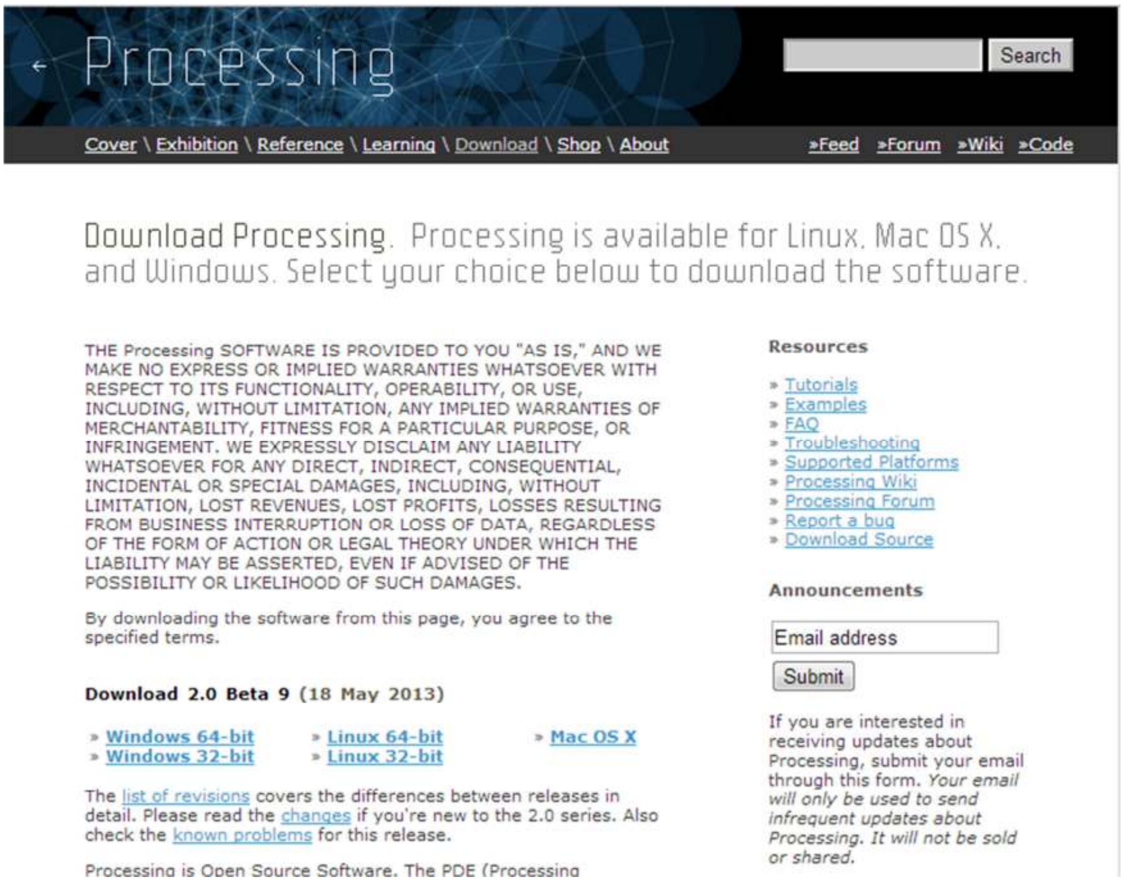
```
        native Lib Version = RXTX-2.2pre2
[0] "COM1"
[1] "COM3"
```

22


Make: PROJECTS
A HANDS-ON INTRODUCTION TO MAKING INTERACTIVE GRAPHICS
Getting Started with Processing
Casey Reas & Ben Fry
O'REILLY
Make:
makezine.com

3.5.2 Installation Processing software

Go the Processing website and go to "Download":
http://processing.org/download/


Processing
Search
Cover \ Exhibition \ Reference \ Learning \ Download \ Shop \ About
»Feed »Forum »Wiki »Code

Download Processing. Processing is available for Linux, Mac OS X, and Windows. Select your choice below to download the software.

THE Processing SOFTWARE IS PROVIDED TO YOU "AS IS," AND WE MAKE NO EXPRESS OR IMPLIED WARRANTIES WHATSOEVER WITH RESPECT TO ITS FUNCTIONALITY, OPERABILITY, OR USE, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR INFRINGEMENT. WE EXPRESSLY DISCLAIM ANY LIABILITY WHATSOEVER FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR SPECIAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST REVENUES, LOST PROFITS, LOSSES RESULTING FROM BUSINESS INTERRUPTION OR LOSS OF DATA, REGARDLESS OF THE FORM OF ACTION OR LEGAL THEORY UNDER WHICH THE LIABILITY MAY BE ASSERTED, EVEN IF ADVISED OF THE POSSIBILITY OR LIKELIHOOD OF SUCH DAMAGES.

By downloading the software from this page, you agree to the specified terms.

Download 2.0 Beta 9 (18 May 2013)

» Windows 64-bit
» Windows 32-bit
» Linux 64-bit
» Linux 32-bit
» Mac OS X

The list of revisions covers the differences between releases in detail. Please read the changes if you're new to the 2.0 series. Also check the known problems for this release.

Processing is Open Source Software. The PDE (Processing

Resources

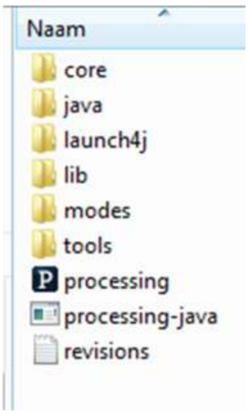
» Tutorials
» Examples
» FAQ
» Troubleshooting
» Supported Platforms
» Processing Wiki
» Processing Forum
» Report a bug
» Download Source

Announcements

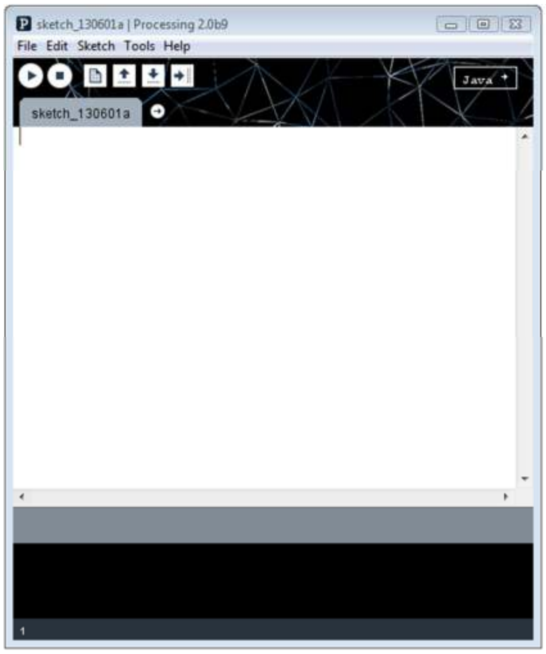
Email address
Submit

If you are interested in receiving updates about Processing, submit your email through this form. *Your email will only be used to send infrequent updates about Processing. It will not be sold or shared.*

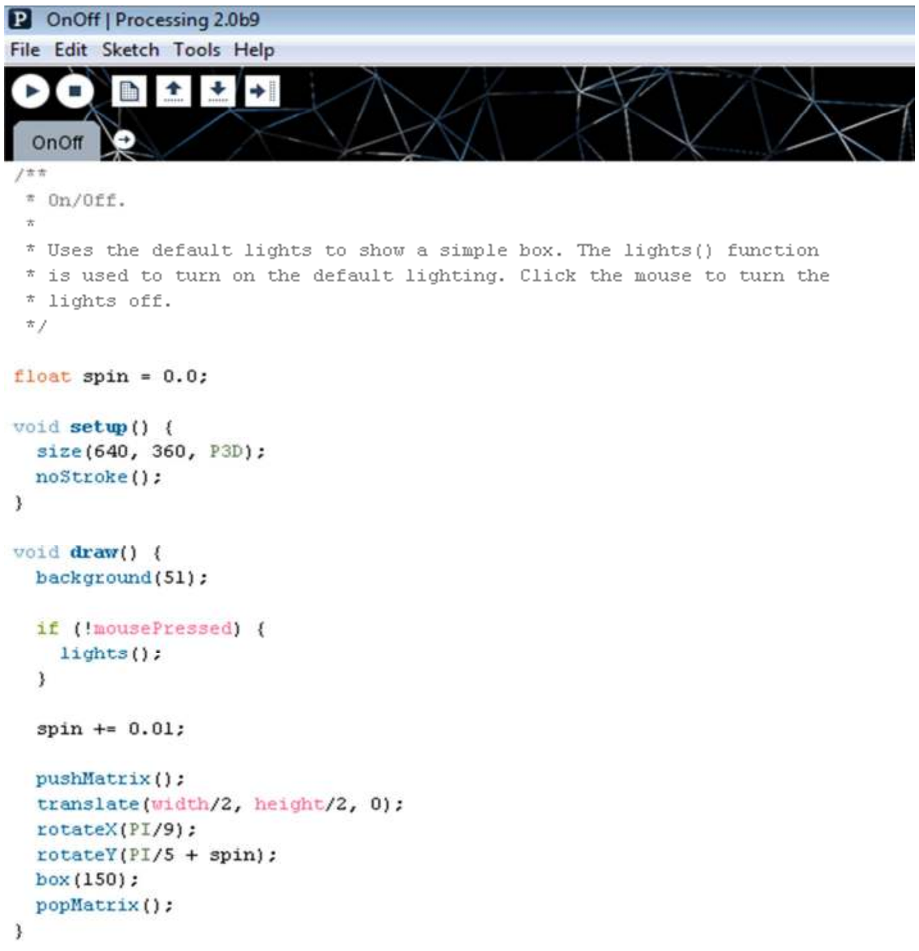
- ➔ Then choose the system you are using on your laptop or pc, a zipped map will automatically download now.
- ➔ Unzip the map in a directory that you prefer.
- ➔ When unzipped you can click on the Processing logo and the software will open.



➔ Now you can start writing your program:


sketch_130601a | Processing 2.0b9
File Edit Sketch Tools Help
Java
sketch_130601a
1

3.5.3 Practical example



```
/**
 * On/Off.
 *
 * Uses the default lights to show a simple box. The lights() function
 * is used to turn on the default lighting. Click the mouse to turn the
 * lights off.
 */

float spin = 0.0;

void setup() {
  size(640, 360, P3D);
  noStroke();
}

void draw() {
  background(51);

  if (!mousePressed) {
    lights();
  }

  spin += 0.01;

  pushMatrix();
  translate(width/2, height/2, 0);
  rotateX(PI/9);
  rotateY(PI/5 + spin);
  box(150);
  popMatrix();
}
```

4. CHAPTER 4: Hardware

In this chapter I will give information on all the hardware I used.

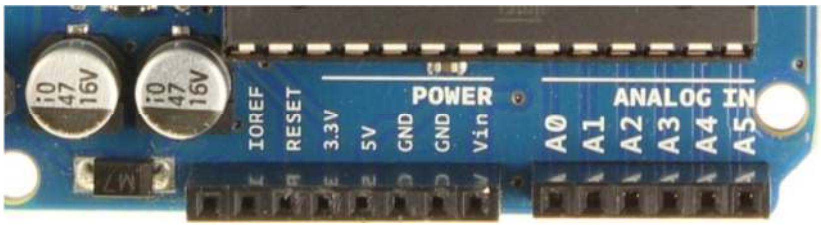
4.1 The Arduino Uno microcontroller

Arduino UNO

Arduino boards can be used in different ways, whether through USB powered through the computer, an adaptar or with a small battery without connecting it to the computer. 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. This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.

3V3. A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.

GND. Ground pins.

IOREF. This pin on the Arduino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs for working with the 5V or 3.3V.

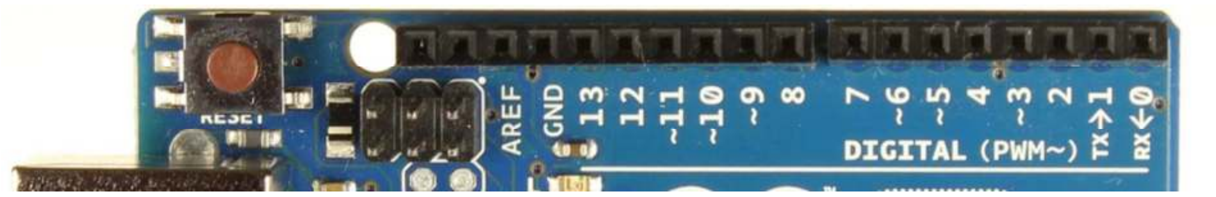
4.1.1 Mean features

The Arduino Uno is a microcontroller board based on the ATmega328 chip (you can find the datasheet at the end of the thesis). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, 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.

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 (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

4.1.2 Inputs and outputs

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:



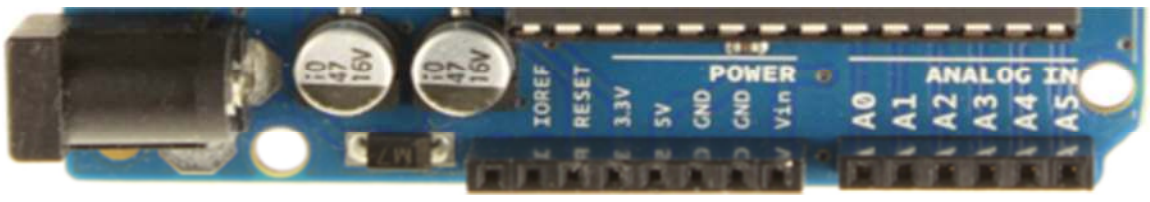
Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.

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 using the SPI library.

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.



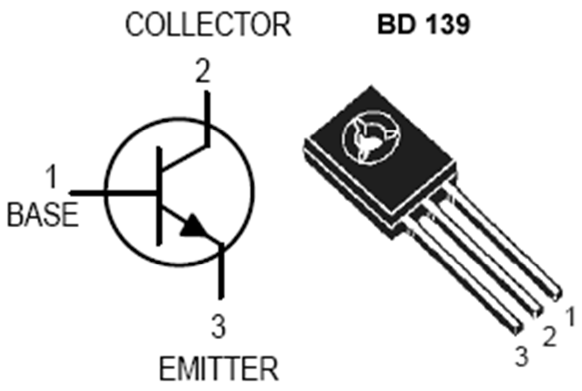
The Uno has **6 analog inputs**, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function.

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.

4.2 NPN Epitaxial silicon transistor BD 139



This transistor allows me to vary the amount of voltage sent to the motor, by applying PWM on it.

There are two requirements for the chosen transistor:

- ➔ the collector-emitter voltage must be greater than 24 volts
- ➔ The base current has to support more than 40 mA, which is the current leaving from the Arduino because of the base resistor.

It's because of these two datasheet values that I use the chosen transistor 139 which has a collector emitter voltage up to 100 volts and base current of 0.5 Amperes.

You can find the BD139 transistor datasheet in CHAPTER 6.

4.3 Transformer 24V DC out power supply

This transformer from the company *fullwat* creates a constant 24 Voltage DC on its output, this 24V will be controlled between 0-24V by the PWM that the Arduino uses on the transistor, to control the speed of the motor. This 0-24V depending on the PID controller its output (0-255).

You can find the fullwat FUS-25D-24 datasheet in CHAPTER 6.

5. CHAPTER 5: Sources

http://www.arduino.cc/

http://netherlands.ni.com/

http://vishots.com/getting-started-with-the-labview-interface-for-arduino/%E2%80%8F

http://www.google.com

http://www.ni.com/pdf/manuals/372192a.pdf

http://www.processing.org/

http://fritzing.org/

http://www.avagotech.com/pages/en/motion_control_encoder_products/magnetic_encoders/aeat-6010-a06/

6. CHAPTER 6: Datasheets

6.1 AEAT-6010-A06 10-bits Magnetic Encoder

AEAT-6010/6012 Magnetic Encoder
10 or 12 bit Angular Detection Device

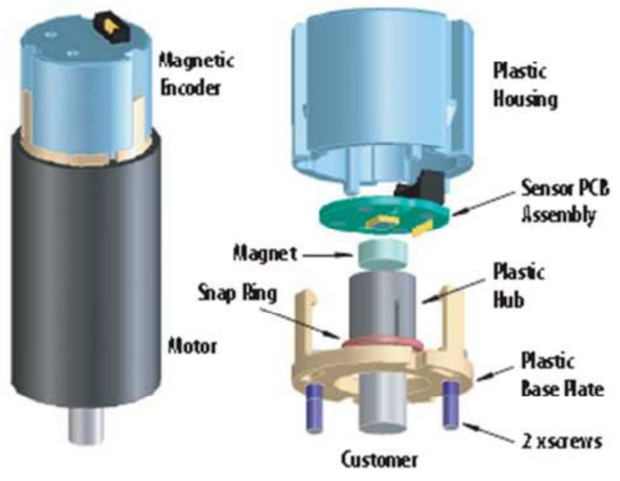
AVAGO TECHNOLOGIES

Data Sheet

Description

Avago Technologies' AEAT-60xx series of magnetic encoders provides an integrated solution for angular detection. With ease of use in mind, these magnetic encoders are ideal for angular detection within 360°. Based on magnetic technologies, the device is non-contact and ensures reliable operations. It is able to provide absolute angle detection upon power-up, with a resolution of 0.0879°(12 bits version) or 0.35°(10bits version), which is equivalent to 4096 and 1024 positions per revolution respectively. The positional data is provided in serial bit stream. There is no upper speed limit; the only restriction is that there will be fewer samples per revolution as the speed increases.

Exploded View



Features

- 10 or 12 bits resolution
- Contactless sensing technologies
- Wide temperature range from -40° to 125°C
- Absolute angular position detection
- Synchronous serial interface (SSI) output for absolute position data (binary format)
- Code monotony error = ± 1 LSB
- 5V supply
- Easy Assembly, No Signal Adjustment required
- RoHS compliant

Applications

- Flow meter
- Angular detection
- Knob control
- Rotary encoder

Device Selection Guide [1]

Part Number	Resolution (bit)	Operating Temperature (°C)	Output Communication	DC Supply Voltage (V), V_{DD}
AEAT-6012-A06	12	-40 to +125	Serial	+5.0
AEAT-6010-A06	10	-40 to +125	Serial	+5.0

Notes:
1. For other options of Magnetic Encoder, please refer to factory.

Table 1. Absolute Maximum Ratings [2, 3]

Parameter	Symbol	Limits	Units	Notes
DC Supply Voltage at pin V_{DD} = 5V	V_{DD}	-0.3 to + 7	V	
Input Voltage	V_i	-0.3 to VDD+0.3	V	
Storage Temperature	T_{STG}	-40 to 125	°C	

Notes:
2. Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.
3. Exposure to absolute maximum rating conditions for extended periods may affect reliability.

Table 2. Recommended Operating Condition

Parameter	Symbol	Values	Units	Notes
DC Supply Voltage at pin V_{DD} = 5V	V_{DD}	+4.5 / +5.5	V	
Ambient Temperature	T_{amb}	−40 to +125	°C	
Maximum Read-out Frequency	f_{CLK}	≤1	MHz	>0 MHz

Table 3. DC Characteristics

DC Characteristics over Recommended Operating Range, typical at 25 °C

			Values				
Parameter	Symbol	Condition	Min	Typ.	Max	Units	Notes
VDD Supply Current	I_{DD}			16	20	mA	
Output High Voltage D0	V_{OH}		V_{DD}-0.5			V	
Output Low Voltage D0	V_{OL}				V_{SS}+0.4	V	
Output Current D0	I_O				4	mA	V_{DD} pin = 4.5V
Input High Voltage CLK, CSn	V_{IH}		0.7*V_{DD}				4
Input Low Voltage CLK, CSn	V_{IL}				0.3*V_{DD}		

Note:
4. CSn is internal pull-up.

Package Dimensions

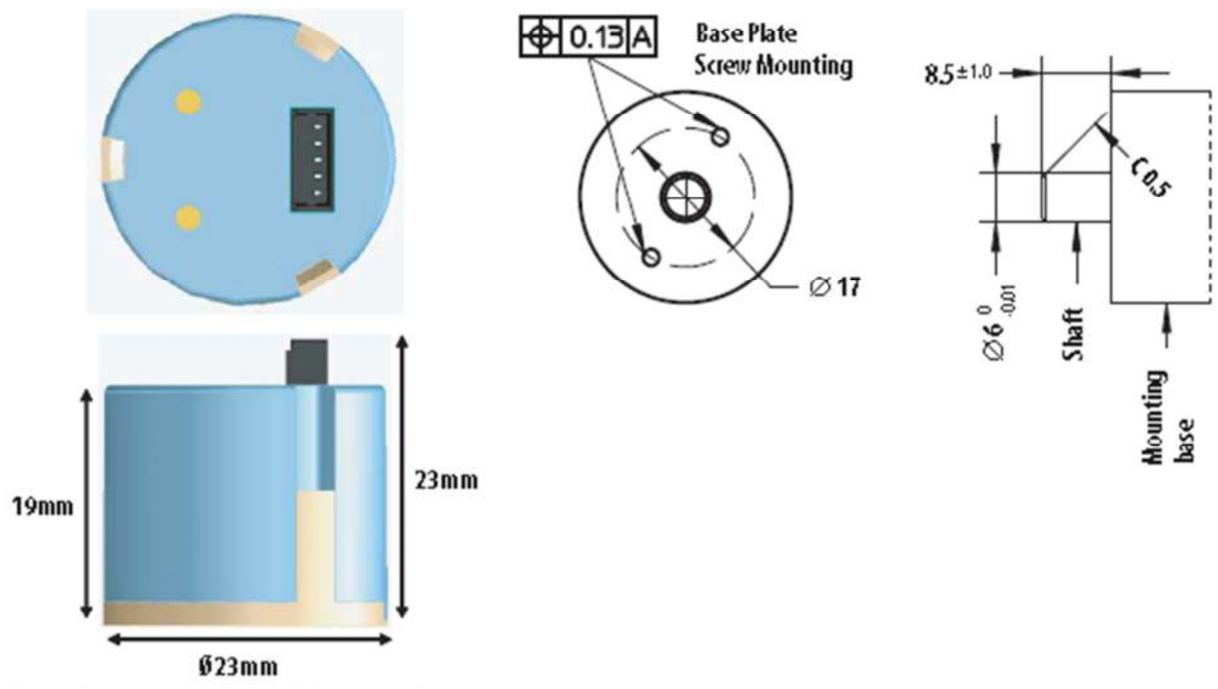


Figure 1. Package and recommended mounting dimension

Parameters

No.	Parameter	Value
1	Operating Temp(°C)	- 40 to +125
2	Shaft axial play (mm)	± 0.08
3	Shaft TIR (mm)	0.05
4	Mechanical speed (rpm)	12,000
5	Shaft diameter (mm)	6 + 0 / -0.01
6	Moment inertia (g-cm^2)	0.104
7	Shaft length – (mm)	8.5 ±1.0
8	Mounting screw size (mm)	M2 x 0.4 x 8 (socket head cap screw, head Ø3.8 ± 0.18 mm)
9	Recommended screw torque	0.6 lb.inch
10	Encoder base plate thickness (mm)	2
11	Bolt circle	± 0.13

* Note:- For high temperature application, it is highly recommended that adhesive be applied at least to the screw and the base plate interface. Refer Application Note for further details.

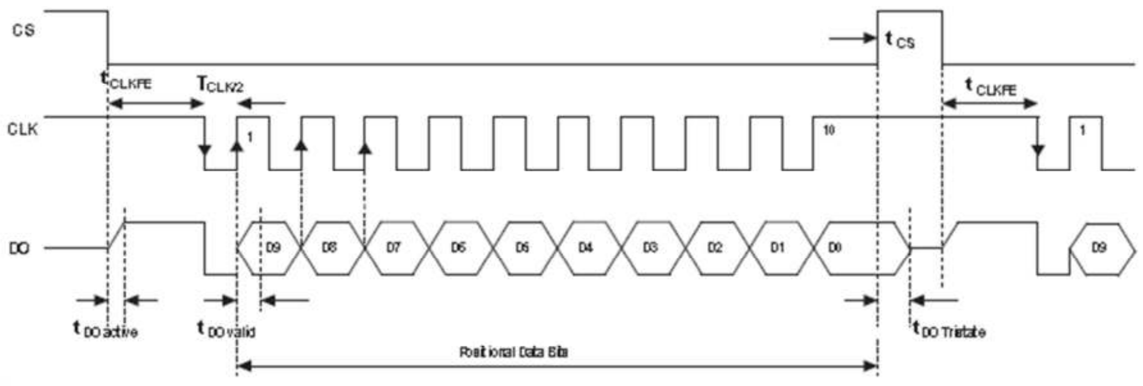
Table 4. Timing Characteristics

Timing Characteristics over Recommended Operating Range, typical at 25 °C

			Values				
Parameter	**Symbol**	**Condition**	**Min**	**Typ.**	**Max**	**Units**	**Notes**
Data output activated (logic high)	$T_{DO\,active}$				100	ns	1
First data shifted to output register	$t_{CLK\,FE}$		500			ns	2
Start of data output	$T_{CLK/2}$		500			ns	3
Data output valid	$T_{DO\,valid}$				375	ns	4
Data output tristate	$T_{DO\,tristate}$				100	ns	5
Pulse width of CSn	T_{CSn}		500			ns	6
Sampling rate for absolute output	f_{abs}		9.9	10.42	10.94	kHz	7
Power-up time	t_{CF}						8
10-bit version			-	-	50	ms	
12-bit version			-	-	20	ms	

Notes:
1. Time between falling edge of CSn and data output activated
2. Time between falling edge of CSn and first falling edge of CLK
3. Rising edge of CLK shifts out one bit a time
4. Time between rising edge of CLK and data output valid
5. After the last bit DO changes back to "tristate"
6. CSn=high; To initiate read-out of next angular position
7. Internal sampling rate.
8. Until internal compensation finished

Timing Characteristics



Notes:
1. Please refer to Table 4 for Timing Characteristics.
2. For 12 bits version; the Positional Data Bits will start with D11 instead and end at D0.

Figure 2. Timing Diagram for 10 bit Magnetic Encoder

Table 5. Linearity

Parameter	Symbol	Min.	Typ.	Max	Units	Notes
Integral Non-Linearity	INL	-	± 0.8 [1]	± 2.4 [2]	Deg.	
Differential Non-Linearity						
10-bit version	DNL	-	-	± 0.176	Deg.	No missing codes
12-bit version		-	-	± 0.044	Deg.	No missing codes

Notes:
1. Average value at typical operating and mounting conditions.
2. Maximum value over recommended operating range and over radial & axial mounting tolerances.

Linearity Definitions

Integral non-linearity

Integral non-linearity (INL) is the maximum deviation between actual angular position and the position indicated by the encoder's output count, over one revolution. It is defined as the most positive linearity error +INL or the most negative linearity error –INL from the best fit line, whichever is larger.

Differential non-linearity

Differential non-linearity (DNL) is the maximum deviation of the step length from one position to the next.

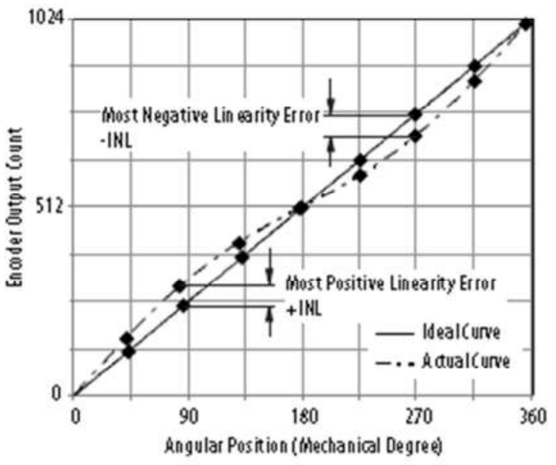


Figure 3. Integral non-linearity

Table 6. Environmental Specifications

Parameter	Reference Standard	Test Conditions	Level
Electromagnetic Compatibility (EMC) [1]			
Electrostatic discharge (ESD) immunity	IEC/EN 61000-4-2	8kV	
Power frequency magnetic field immunity	IEC/EN 61000-4-8	30 A/m (continuous field)	
		300 A/m (short duration field)	Level 4
Pulse magnetic field immunity	IEC/EN 61000-4-97	1000 A/m	Level 5
Damped oscillatory magnetic field immunity	IEC/EN 61000-4-10	100 A/m	Level 5
Mechanical Durability			
Vibration (Operating)	IEC/EN 60068-2-6	10-500Hz at 5G	
Shock	IEC/EN 60068-2-27	6ms at 200G	

Electrical Connections

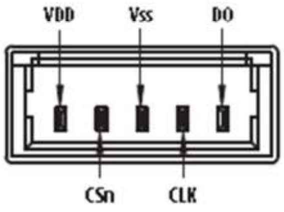


Figure 4. Electrical Connections

Pin	Symbol	Description
1	VDD	5V Supply Voltage
2	CSn	Chip Select – Input (See Figure 2)
3	VSS	Supply Ground
4	CLK	Serial Clock - Input (See Figure 2)
5	DO	Serial Data - Output. (See Figure 2)

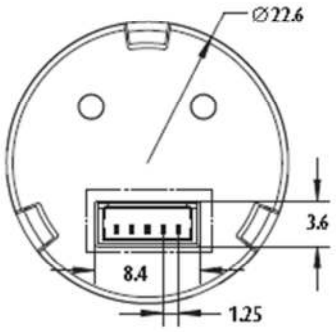
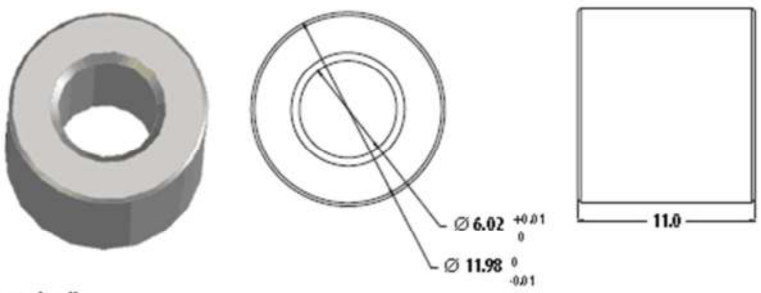


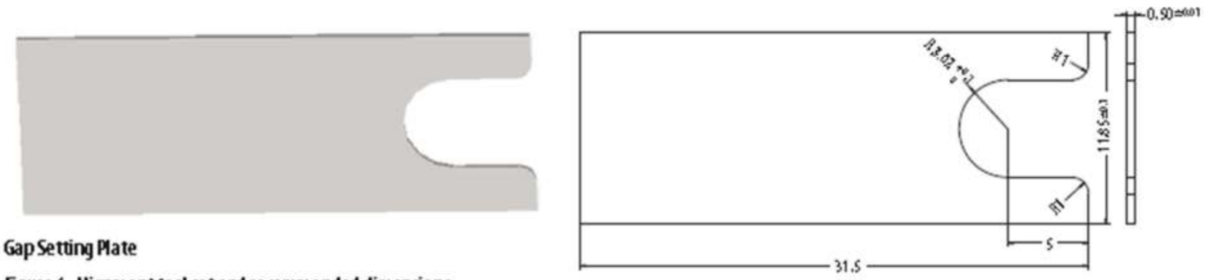
Figure 5. Basic connector dimensions

Alignment Tool Set - Part number HEDS-8934

This optional alignment tool set consists of a gap setting plate and a centering jig. Refer to Application Note 5317 for the assembly guide.



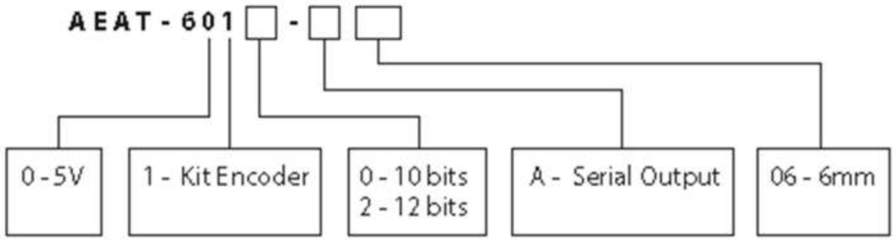
Centering Jig



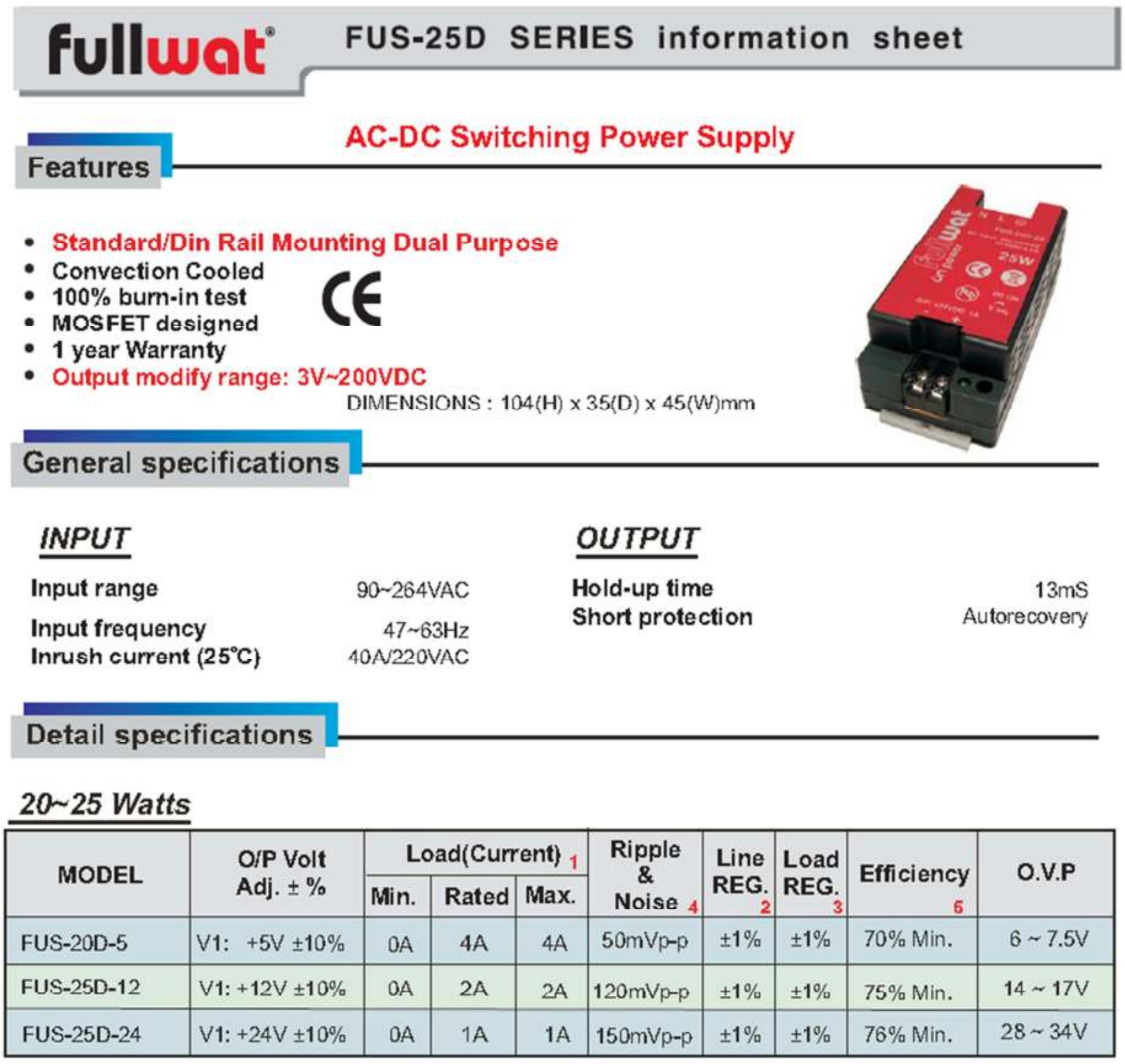
Gap Setting Plate

Figure 6. Alignment tool set and recommended dimensions

Ordering Information



6.2 fullwat FUS-25D-24 Transformer



fullwat® FUS-25D SERIES information sheet

AC-DC Switching Power Supply

Features

- Standard/Din Rail Mounting Dual Purpose
- Convection Cooled
- 100% burn-in test
- MOSFET designed
- 1 year Warranty
- Output modify range: 3V~200VDC

CE

DIMENSIONS : 104(H) x 35(D) x 45(W)mm

General specifications

INPUT

Input range	90~264VAC
Input frequency	47~63Hz
Inrush current (25°C)	40A/220VAC

OUTPUT

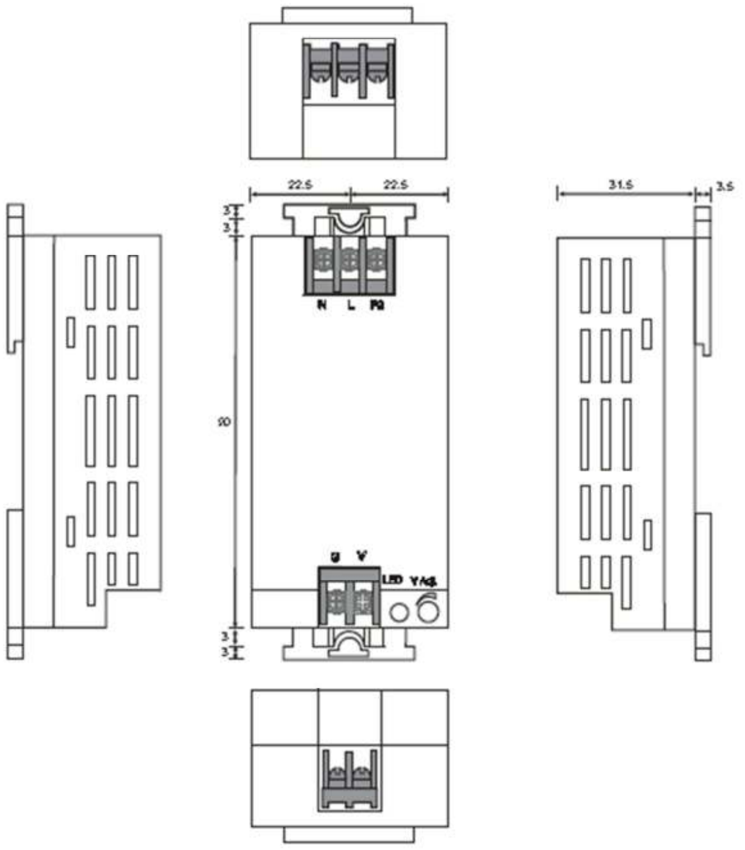
Hold-up time	13mS
Short protection	Autorecovery

Detail specifications

20~25 Watts

MODEL	O/P Volt Adj. ± %	Load(Current) 1			Ripple & Noise 4	Line REG. 2	Load REG. 3	Efficiency 6	O.V.P
		Min.	Rated	Max.					
FUS-20D-5	V1: +5V ±10%	0A	4A	4A	50mVp-p	±1%	±1%	70% Min.	6 ~ 7.5V
FUS-25D-12	V1: +12V ±10%	0A	2A	2A	120mVp-p	±1%	±1%	75% Min.	14 ~ 17V
FUS-25D-24	V1: +24V ±10%	0A	1A	1A	150mVp-p	±1%	±1%	76% Min.	28 ~ 34V

Mechanical Details

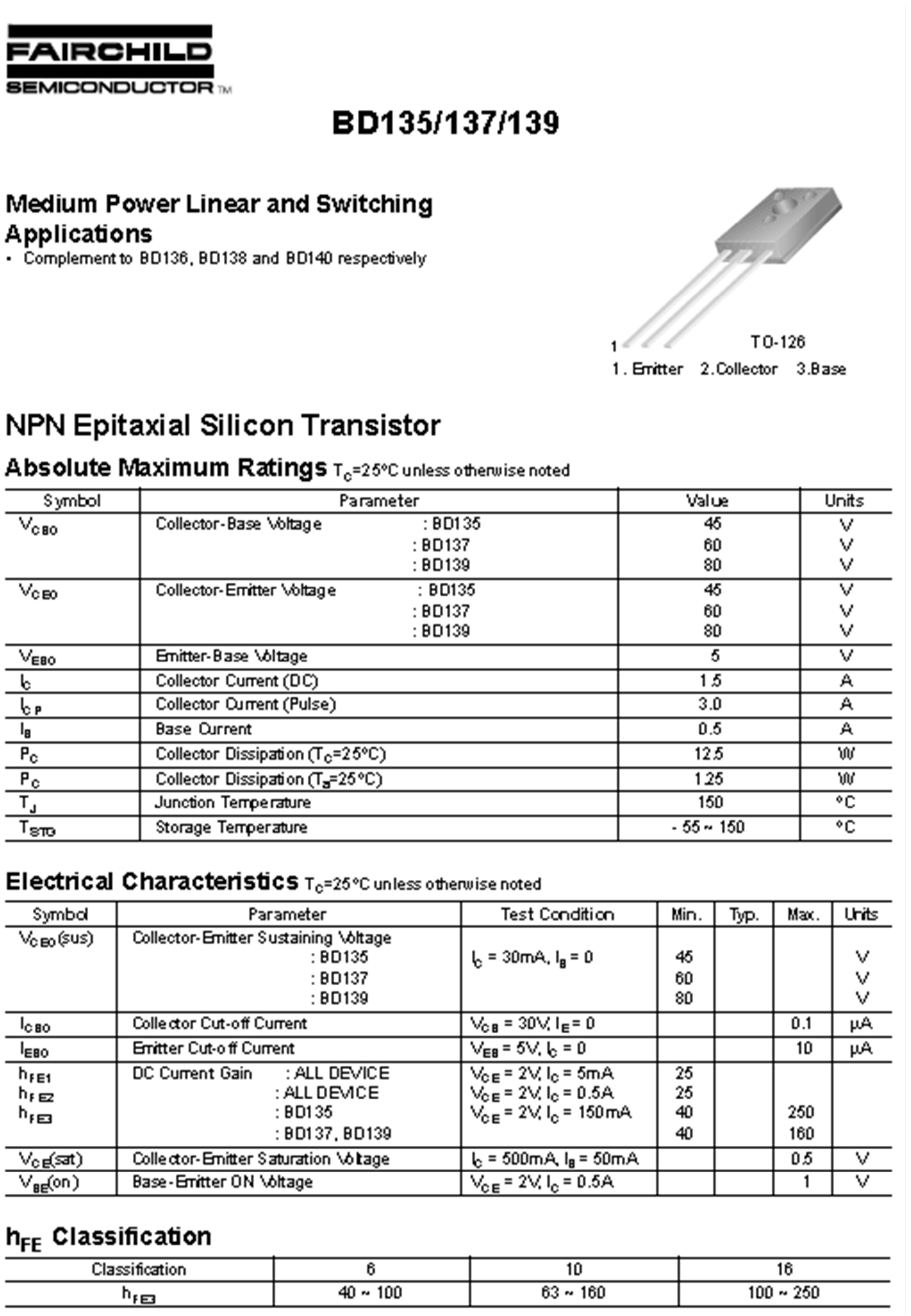


Panel Designation

CASE NO.:CS025
UNIT:mm
DIMENSIONS:104(H)*35(D)*45(W)

Single	Description
L	L:Line Terminal of Input
N	N:Neutral Teminal of Input
FG	FG:Ground(Earth)
G	G:DC Output Ground Terminal
V	V:DC Output Terminal

6.3 BD 139 Transistor

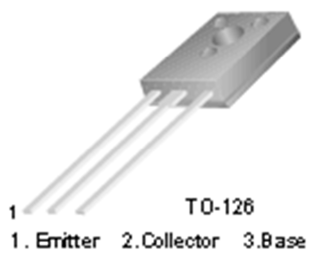


FAIRCHILD SEMICONDUCTOR ™

BD135/137/139

Medium Power Linear and Switching Applications

- Complement to BD136, BD138 and BD140 respectively



NPN Epitaxial Silicon Transistor

Absolute Maximum Ratings T_C=25°C unless otherwise noted

Symbol	Parameter	Value	Units
V_{CBO}	Collector-Base Voltage : BD135 : BD137 : BD139	45 60 80	V V V
V_{CEO}	Collector-Emitter Voltage : BD135 : BD137 : BD139	45 60 80	V V V
V_{EBO}	Emitter-Base Voltage	5	V
I_C	Collector Current (DC)	1.5	A
I_{CP}	Collector Current (Pulse)	3.0	A
I_B	Base Current	0.5	A
P_C	Collector Dissipation (T_C=25°C)	12.5	W
P_C	Collector Dissipation (T_a=25°C)	1.25	W
T_J	Junction Temperature	150	°C
T_{STG}	Storage Temperature	- 55 ~ 150	°C

Electrical Characteristics T_C=25°C unless otherwise noted

Symbol	Parameter	Test Condition	Min.	Typ.	Max.	Units
V_{CEO}(sus)	Collector-Emitter Sustaining Voltage : BD135 : BD137 : BD139	I_C = 30mA, I_B = 0	45 60 80			V V V
I_{CBO}	Collector Cut-off Current	V_{CB} = 30V, I_E = 0			0.1	μA
I_{EBO}	Emitter Cut-off Current	V_{EB} = 5V, I_C = 0			10	μA
h_{FE1} h_{FE2} h_{FE3}	DC Current Gain : ALL DEVICE : ALL DEVICE : BD135 : BD137, BD139	V_{CE} = 2V, I_C = 5mA V_{CE} = 2V, I_C = 0.5A V_{CE} = 2V, I_C = 150mA	25 25 40 40		 250 160	
V_{CE}(sat)	Collector-Emitter Saturation Voltage	I_C = 500mA, I_B = 50mA			0.5	V
V_{BE}(on)	Base-Emitter ON Voltage	V_{CE} = 2V, I_C = 0.5A			1	V

h_{FE} Classification

Classification	6	10	16
h_{FE3}	40 ~ 100	63 ~ 160	100 ~ 250

Typical Characteristics

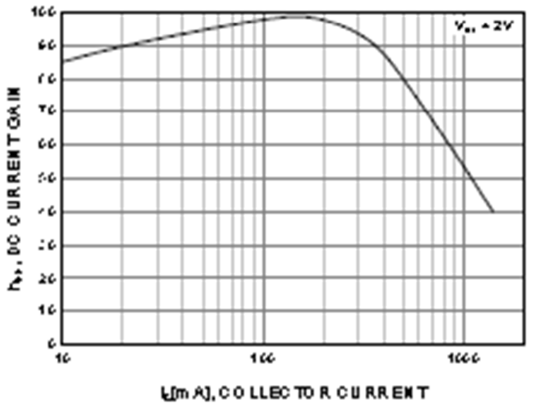


Figure 1. DC current Gain

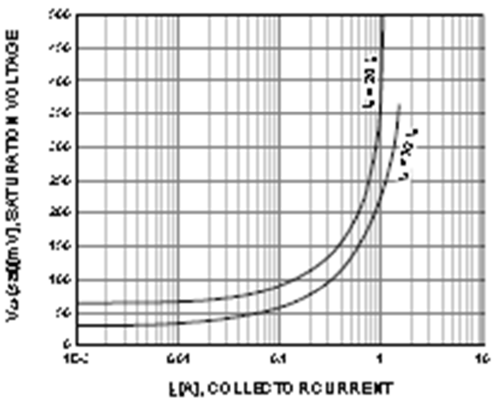


Figure 2. Collector-Emitter Saturation Voltage

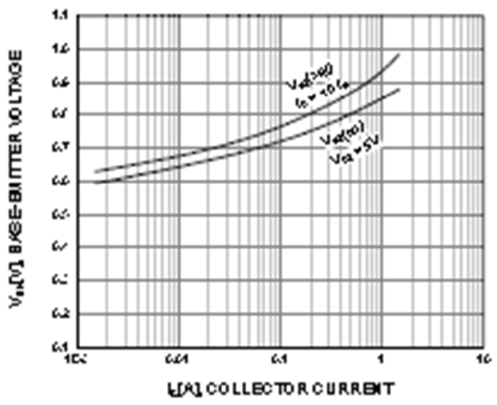


Figure 3. Base-Emitter Voltage

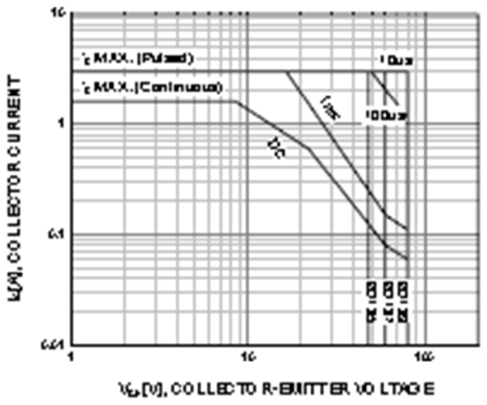


Figure 4. Safe Operating Area

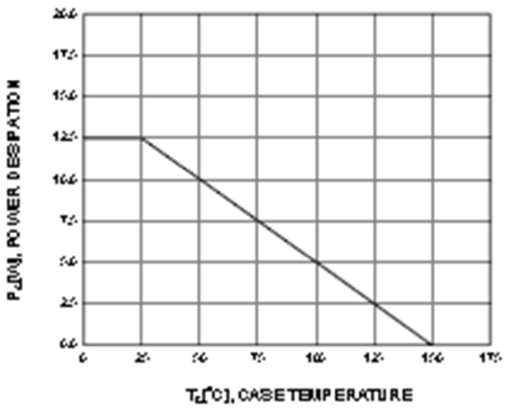
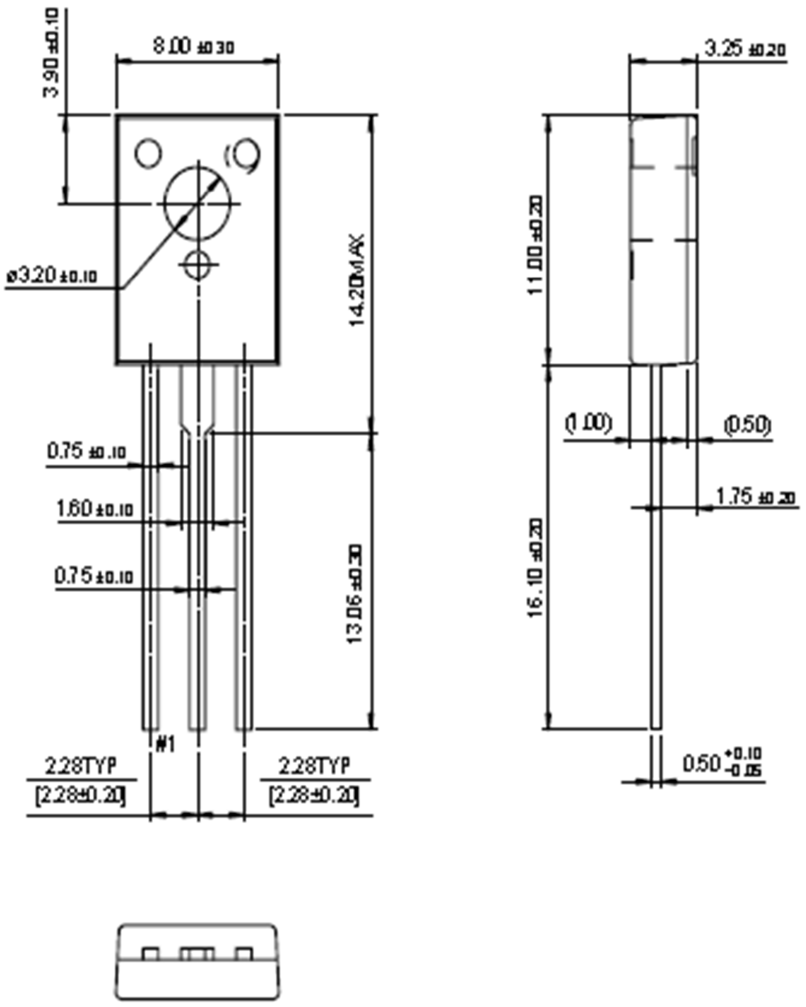


Figure 5. Power Derating

Package Demensions

TO-126


8.00 ±0.30
3.90 ±0.10
ø3.20 ±0.10
14.20MAX
0.75 ±0.10
1.60 ±0.10
0.75 ±0.10
13.06 ±0.30
#1
2.28TYP
[2.28±0.20]
2.28TYP
[2.28±0.20]
3.25 ±0.20
11.00 ±0.20
(1.00)
(0.50)
1.75 ±0.20
16.10 ±0.20
0.50 +0.10 -0.05