Report

Subject: Embedded Systems

Topic: Control

**Student: Marusic Diana, FAF-171**

**Teacher: Bragarenco Andrei**

**Content**

1. **Domain**
2. **Component description**
3. **Implementation**
4. **Proof**
5. **Annex**

**Domain**

**Control system**

A control system manages, commands, directs, or regulates the behavior of other devices or systems using [control loops](https://en.wikipedia.org/wiki/Control_loop).



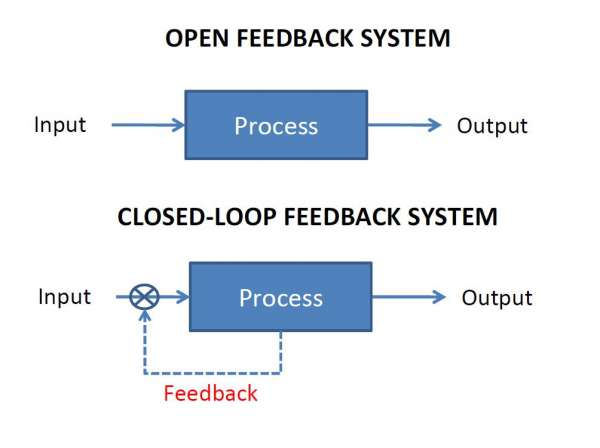
It can range from a single home heating controller using a [thermostat](https://en.wikipedia.org/wiki/Thermostat) controlling a domestic boiler to large [Industrial control systems](https://en.wikipedia.org/wiki/Industrial_control_system) which are used for controlling [processes](https://en.wikipedia.org/wiki/Process_(engineering)) or machines.

****

**Open-loop control system**

In an **open-loop** control system, the control action from the controller is independent of the process variable.

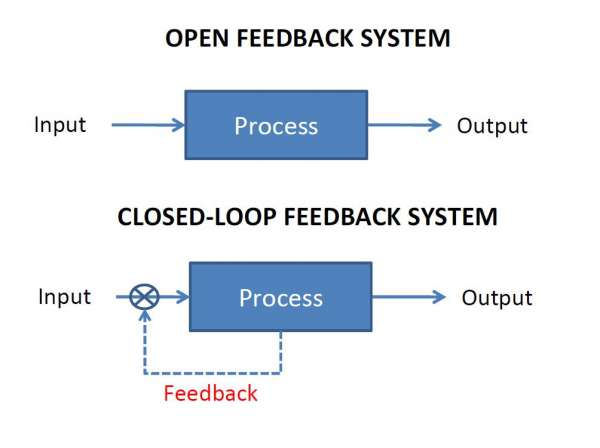
An example of this is a central heating boiler controlled only by a timer. The control action is the switching on or off of the boiler. The process variable is the building temperature. This controller operates the heating system for a constant time regardless of the temperature of the building.



**Closed-loop control system**

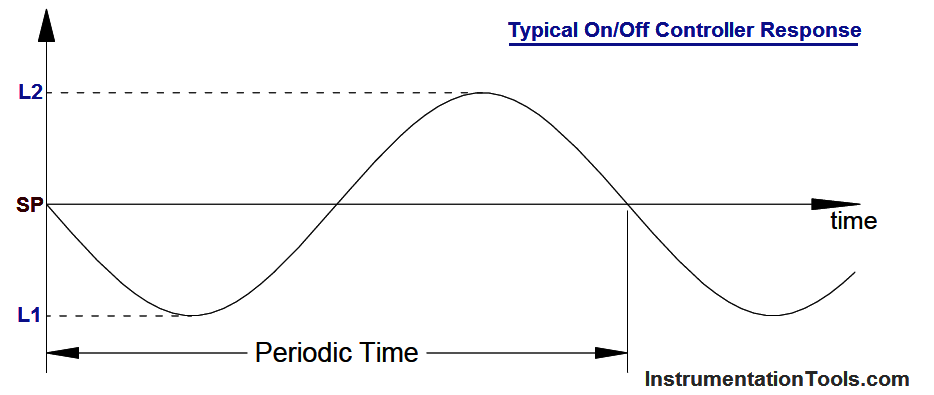
In a closed-loop control system, the control action from the controller is dependent on the desired and actual process variable.

In the case of the boiler analogy, this would utilise a thermostat to monitor the building temperature, and feed back a signal to ensure the controller output maintains the building temperature close to that set on the thermostat. A closed loop controller has a feedback loop which ensures the controller exerts a control action to control a process variable at the same value as the setpoint. For this reason, closed-loop controllers are also called feedback controllers.



**On/Off control**

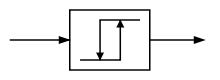
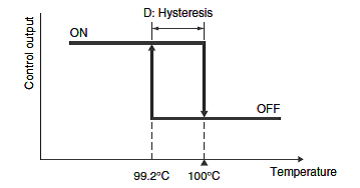
On–off control uses a feedback controller that switches abruptly between two states. A simple bi-metallic domestic thermostat can be described as an on-off controller. When the temperature in the room (PV) goes below the user setting (SP), the heater is switched on. Another example is a pressure switch on an air compressor. When the pressure (PV) drops below the setpoint (SP) the compressor is powered. Simple on–off control systems like these can be cheap and effective.

****

**On/Off Hysteresis control (bang-bang controller)**

ON/OFF control action turns the output ON or OFF based on the set point. The output frequently changes according to minute temperature changes as a result, and this shortens the life of the output relay or unfavorably affects some devices connected to the Temperature Controller. To prevent this from happening, a temperature band called hysteresis is created between the ON and OFF operations.

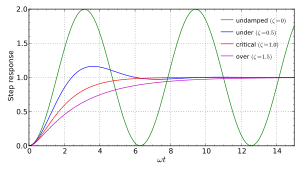
Due to the discontinuous control signal, systems that include on/off hysteresis controllers are variable structure systems, and hysteresis controllers are thus variable structure controllers.

****

**Proportional control**

Proportional control is a type of linear feedback control system in which a correction is applied to the controlled variable which is proportional to the difference between the desired value (SP) and the measured value (PV). Two classic mechanical examples are the toilet bowl float proportioning valve and the fly-ball governor.

The proportional control system is more complex than an on–off control system, but simpler than a proportional-integral-derivative (PID) control system used, for instance, in an automobile cruise control.



**PID(proportional–integral–derivative) or three-term controller**

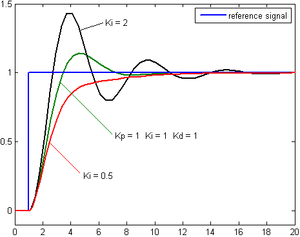
It is a control loop mechanism employing feedback that is widely used in industrial control systems and a variety of other applications requiring continuously modulated control. A PID controller continuously calculates an *error value* e(t) as the difference between a desired setpoint (SP) and a measured process variable (PV) and applies a correction based on proportional, integral, and derivative terms (denoted *P*, *I*, and *D* respectively).

#### **Derivative action**

The derivative is concerned with the rate-of-change of the error with time: If the measured variable approaches the setpoint rapidly, then the actuator is backed off early to allow it to coast to the required level; conversely, if the measured value begins to move rapidly away from the setpoint, extra effort is applied—in proportion to that rapidity to help move it back.

On control systems involving motion-control of a heavy item like a gun or camera on a moving vehicle, the derivative action of a well-tuned PID controller can allow it to reach and maintain a setpoint better than most skilled human operators. If derivative action is over-applied, it can, however, lead to oscillations.

#### **Integral action**



Change of response of second-order system to a step input for varying Ki values.

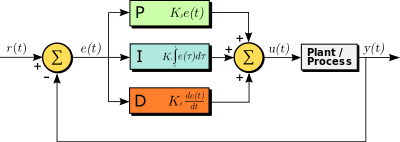
The integral term magnifies the effect of long-term steady-state errors, applying an ever-increasing effort until error is removed. In the example of the furnace above working at various temperatures, if the heat being applied does not bring the furnace up to setpoint, for whatever reason, [integral](https://en.wikipedia.org/wiki/Integral) action increasingly *moves* the proportional band relative to the setpoint until the PV error is reduced to zero and the setpoint is achieved.

#### **Ramp up % per minute**

Some controllers include the option to limit the "ramp up % per minute". This option can be very helpful in stabilizing small boilers (3 MBTUH), especially during the summer, during light loads. A utility boiler "unit may be required to change load at a rate of as much as 5% per minute.

**Mathematical form:**





**Applicability:**

The use of the PID algorithm does not guarantee optimal control of the system or its control stability (see § Limitations of PID control, below). Situations may occur where there are excessive delays: the measurement of the process value is delayed, or the control action does not apply quickly enough. In these cases lead–lag compensation is required to be effective. The response of the controller can be described in terms of its responsiveness to an error, the degree to which the system overshoots a setpoint, and the degree of any system oscillation. But the PID controller is broadly applicable, since it relies only on the response of the measured process variable, not on knowledge or a model of the underlying process.

**Component description**

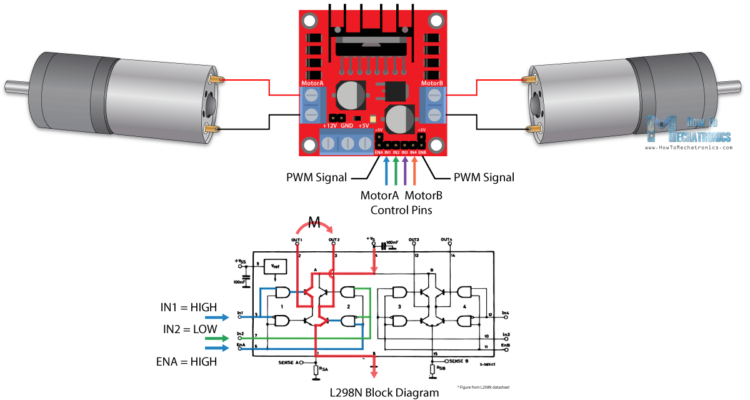
**DC motor**

****

A DC(direct current) motor is any of a class of rotary [electrical motors](https://en.wikipedia.org/wiki/Electrical_motor) that converts direct current electrical energy into mechanical energy. The most common types rely on the forces produced by magnetic fields. Nearly all types of DC motors have some internal mechanism, either electromechanical or electronic, to periodically change the direction of current in part of the motor.

A DC motor consists of an stator, an armature, a rotor and a commutator with brushes. Opposite polarity between the two magnetic fields inside the motor cause it to turn. DC motors are the simplest type of motor and are used in household appliances, such as electric razors, and in electric windows in cars.

**L298 motor driver**

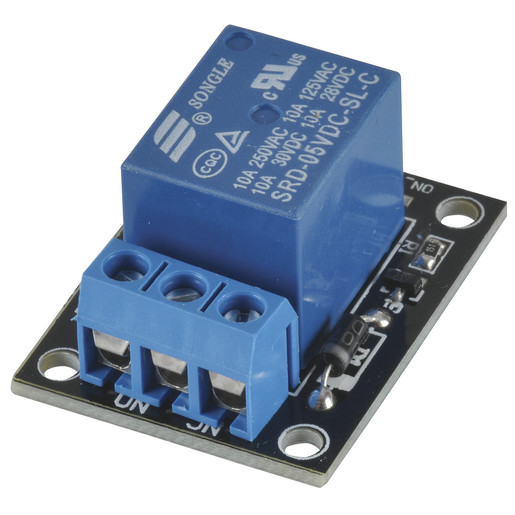
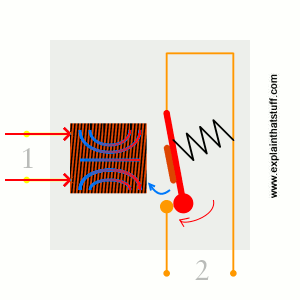
****

The L298N is a dual H-Bridge motor driver which allows speed and direction control of two DC motors at the same time. The module can drive DC motors that have voltages between 5 and 35V, with a peak current up to 2A.

The L298N module has two screw terminal blocks for the motor A and B, and another screw terminal block for the Ground pin, the VCC for motor and a 5V pin which can either be an input or output. This depends on the voltage used at the motors VCC. The module has an onboard 5V regulator which is either enabled or disabled using a jumper. If the motor supply voltage is up to 12V we can enable the 5V regulator and the 5V pin can be used as output, for example for powering our Arduino board. But if the motor voltage is greater than 12V we must disconnect the jumper because those voltages will cause damage to the onboard 5V regulator. In this case the 5V pin will be used as input as we need to connect it to a 5V power supply in order for the IC to work properly.

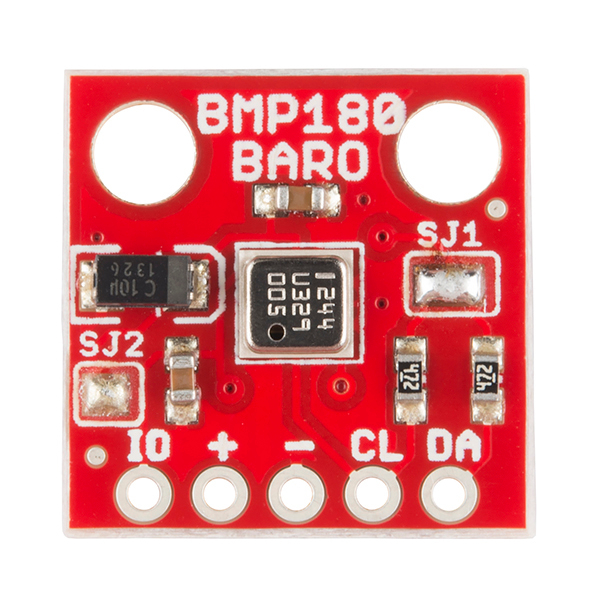
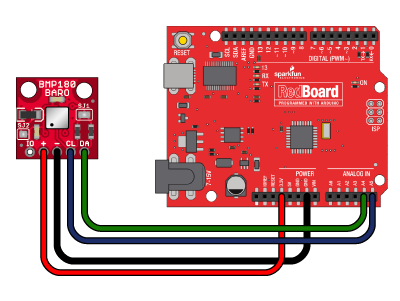
Next are the logic control inputs. The Enable A and Enable B pins are used for enabling and controlling the speed of the motor. If a jumper is present on this pin, the motor will be enabled and work at maximum speed, and if we remove the jumper we can connect a PWM input to this pin and in that way control the speed of the motor. If we connect this pin to a Ground the motor will be disabled.

**Relay**

** **

A relay is an [electromagnetic](https://www.explainthatstuff.com/magnetism.html) switch operated by a relatively small [electric](https://www.explainthatstuff.com/electricity.html) current that can turn on or off a much larger electric current. The heart of a relay is an electromagnet (a coil of wire that becomes a temporary [magne](https://www.explainthatstuff.com/magnetism.html)t when electricity flows through it). It is common to think of a relay as a kind of electric [lever](https://www.explainthatstuff.com/toolsmachines.html): switch it on with a tiny current and it switches on ("leverages") another appliance using a much bigger current. This is useful, because, as the name suggests, many sensors are incredibly sensitive pieces of [electronic](https://www.explainthatstuff.com/electronics.html) equipment and produce only small electric currents. But often we need them to drive bigger pieces of apparatus that use bigger currents. Relays bridge the gap, making it possible for small currents to activate larger ones. That means relays can work either as switches (turning things on and off) or as amplifiers (converting small currents into larger ones).

**Temperature sensor - BMP180 Barometric Pressure, Temperature, Altitude sensor**

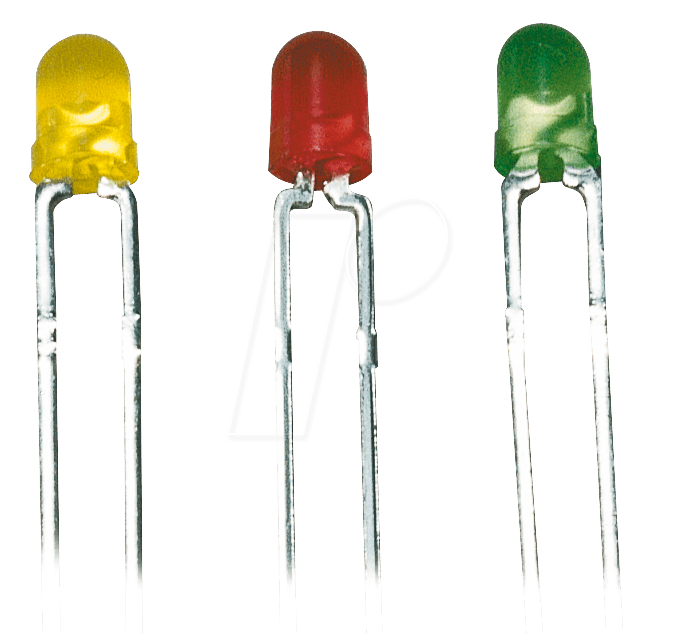
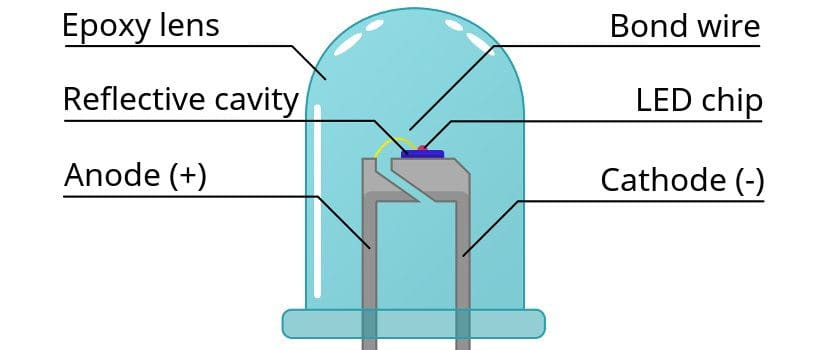
****

# The BMP180 Breakout is a barometric pressure sensor with an I2C ("Wire") interface.

# Barometric pressure sensors measure the absolute pressure of the air around them. This pressure varies with both the weather and altitude. Depending on how you interpret the data, you can monitor changes in the weather, measure altitude, or any other tasks that require an accurate pressure reading.

It is also possible to measure temperature with this sensor.

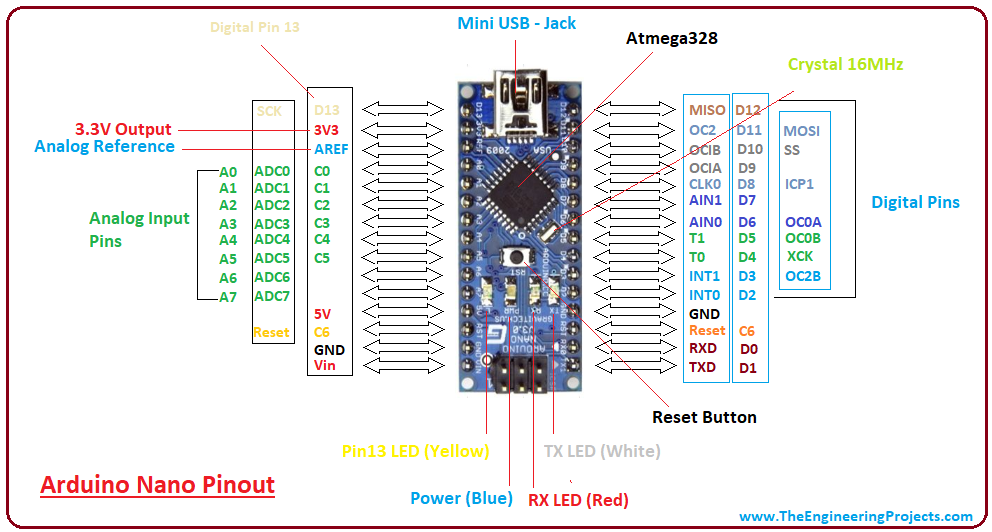
**LED (Light-Emitting Diode)**

** **

LED is a [semiconductor](https://en.wikipedia.org/wiki/Semiconductor) [light sourc](https://en.wikipedia.org/wiki/Light_source)e that emits light when [current](https://en.wikipedia.org/wiki/Electric_current) flows through it. [Electrons](https://en.wikipedia.org/wiki/Electron) in the semiconductor recombine with [electron holes](https://en.wikipedia.org/wiki/Electron_hole), releasing energy in the form of [photons](https://en.wikipedia.org/wiki/Photon). The color of the light (corresponding to the energy of the photons) is determined by the energy required for electrons to cross the [band gap](https://en.wikipedia.org/wiki/Band_gap) of the semiconductor. White light is obtained by using multiple semiconductors or a layer of light-emitting phosphor on the semiconductor device.

There are some advantages of LEDs over incandescent light bulbs. Unlike [incandescent bulbs](https://home.howstuffworks.com/light-bulb.htm), LEDs don't have filaments that burn out, they use less electricity, and they don't get very hot. LEDs last just as long as a standard transistor. The lifespan of an LED surpasses the short life of an incandescent bulb by thousands of hours

**Arduino Nano**

****

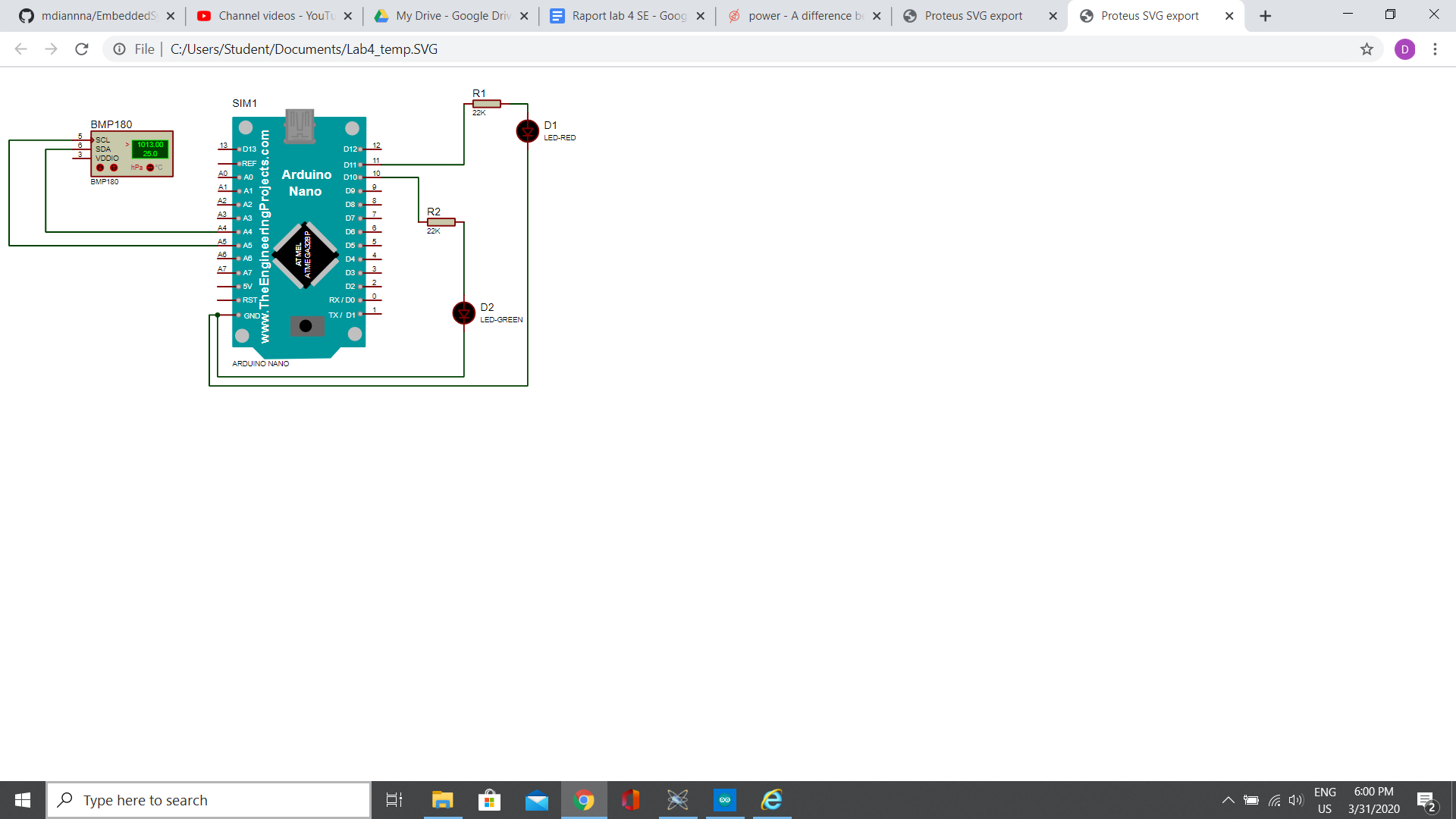
The **Arduino Nano** is a small, complete, and breadboard-friendly board based on the ATmega328P (Arduino Nano 3. x). It has more or less the same functionality of the Arduino Duemilanove, but in a different package. It lacks only a DC power jack, and works with a Mini-B USB cable instead of a standard one. It has 14 digital pins (D0-D13) and 8 analog input pins (D0-D7). Its small dimensions make it a good board for compact projects.

**Implementation**

1. **Temperature control with On/Off hysteresis:**

****

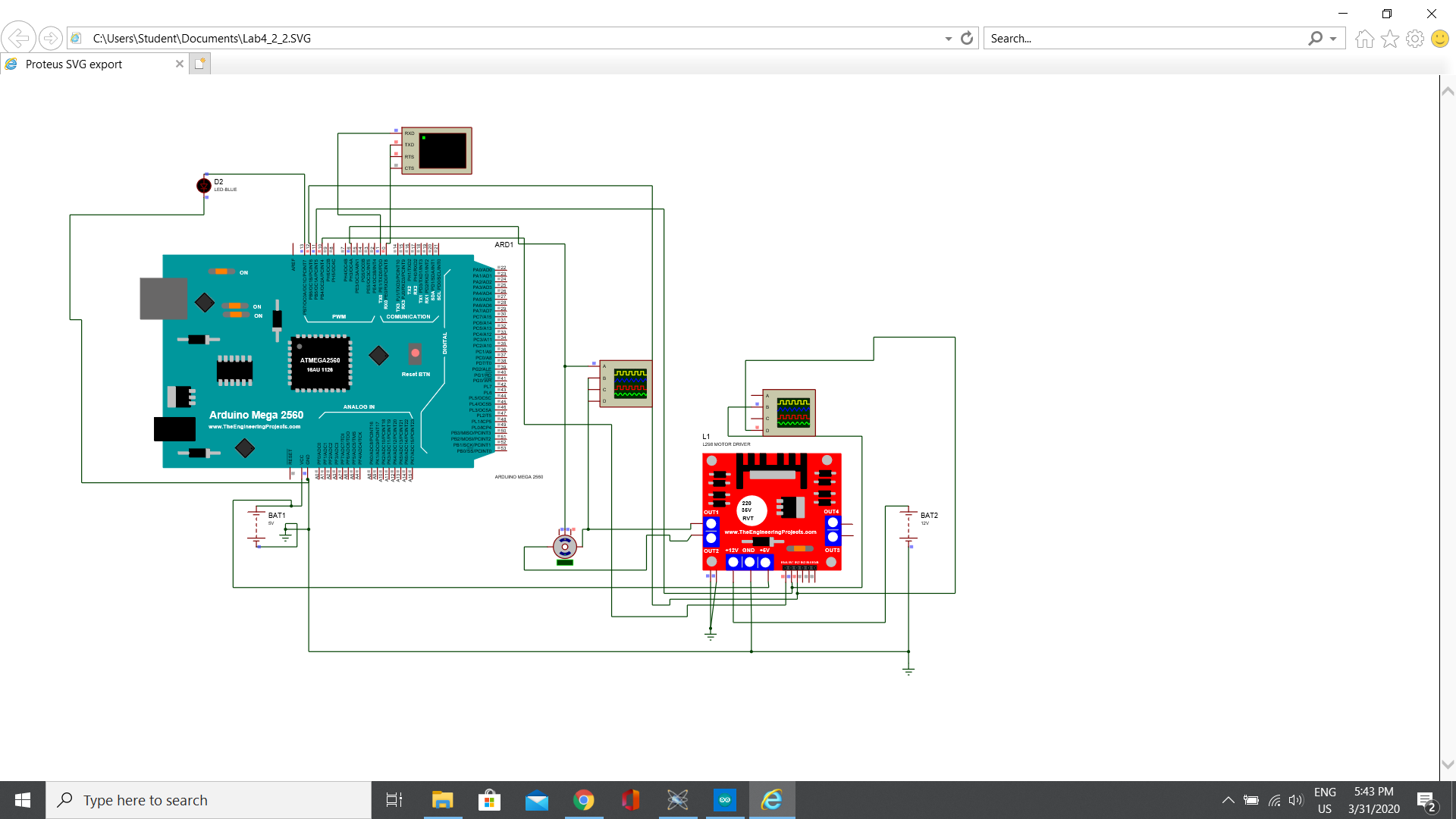
In the first part of the laboratory work, the temperature On/Off with hysteresis was demonstrated using the BMP180 sensor - that is used to measure air pressure, but also temperature. When the temperature goes below 25 - 1 degrees, the red LED is turned on, meaning that the conditioner is turned on, and when the temperature exceeds 25+1 degrees, the conditioneer is turned off (the red LED will turns and the green LED will turn on)

****

1. **Motor with PID control:**

****

In the second part of the laboratory work, the PID control is used to control a DC motor with L298n driver. Initially the encoder calculates the time in which the motor rotates 1 rotation, and then PID control is setup and start to be used. The encoder will output 1 only when there is a complete circle of the motor, and 0 in other cases. The circuit is shown below:

****

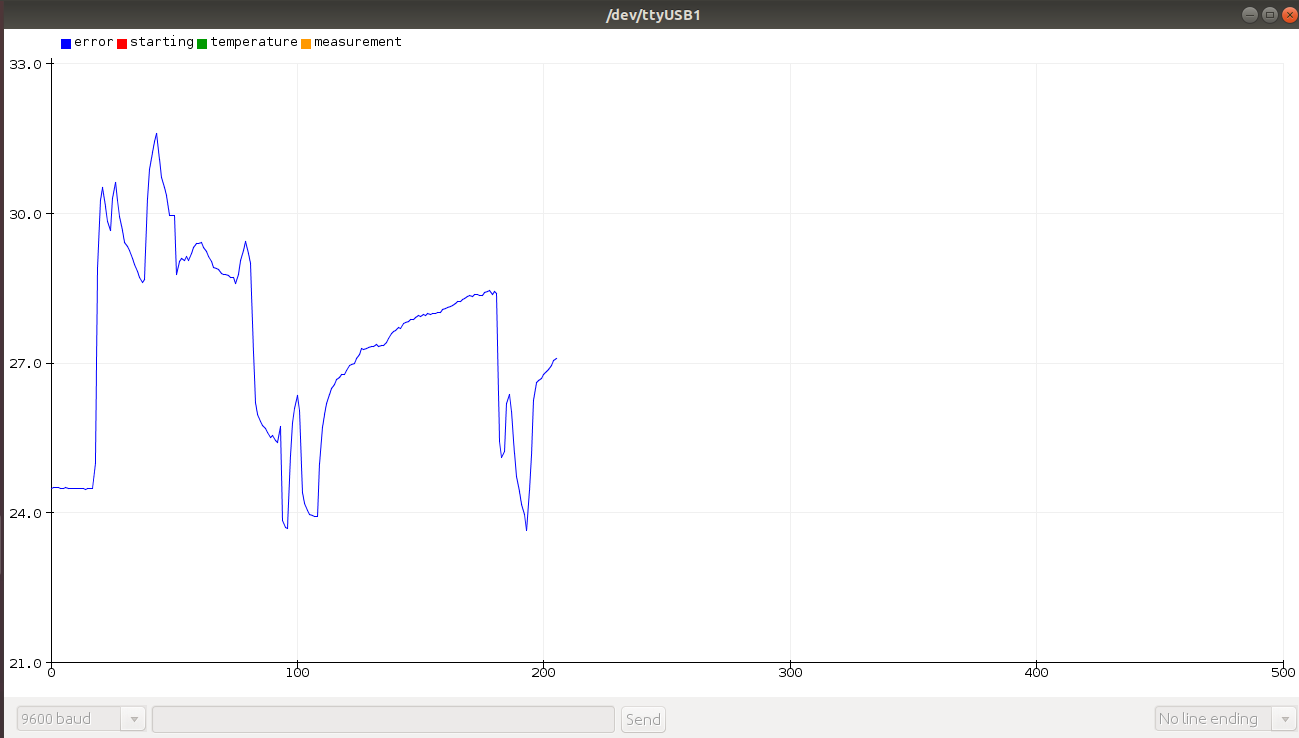
**Proof**

1. **Temperature On/Off with hysteresis**

[**https://www.youtube.com/watch?v=RhDfSjxMg50**](https://www.youtube.com/watch?v=RhDfSjxMg50)

**Video with serial plotter:**

[**https://www.youtube.com/watch?v=1uXtSl\_9nuQ**](https://www.youtube.com/watch?v=1uXtSl_9nuQ)

****

1. **Motor PID Control**

[**https://www.youtube.com/watch?v=USqHfumYLX4**](https://www.youtube.com/watch?v=USqHfumYLX4)

[**https://www.youtube.com/watch?v=n\_dLGJ2Foyk**](https://www.youtube.com/watch?v=n_dLGJ2Foyk)

**Annex - source code**

1. **Temperature On/Off hysteresis control**

-----------------------

Main file:

-----------------------

#include "temperature.h"

#include "conditioner.h"

#include <Arduino.h>

#define TIME\_INTERVAL\_MS 2000 //Read sensor each 2 seconds

#define MAX\_TEMP 26

#define MIN\_TEMP 24

long previousMillis = 0;

void SerialInit() {

Serial.begin(9600);

}

unsigned long CompareTimeInterval(unsigned long currentMillis, unsigned long previousMillis) {

return currentMillis - previousMillis;

}

int TimeIntervalElapsed(unsigned long currentMillis, unsigned long previousMillis) {

int interval = CompareTimeInterval(currentMillis, previousMillis);

if(interval > TIME\_INTERVAL\_MS) {

return 1;

}

return 0;

}

int TemperatureExceedsMaxTemp(double t) {

if(t>=MAX\_TEMP) {

return 1;

}

return 0;

}

int TemperatureBelowMinTemp(double t) {

if(t<=MIN\_TEMP) {

return 1;

}

return 0;

}

void setup() {

SerialInit();

InitTemperatureSensor();

ConditionerInit();

Serial.println("Init complete");

}

void loop()

{

double t = 0.0;

unsigned long currentMillis = millis(); //time elapsed

if(TimeIntervalElapsed(currentMillis, previousMillis))

{

previousMillis = currentMillis; //"Last time is now"

t = GetTemperature();

// Open serial plotter

Serial.println(t);

if(TemperatureExceedsMaxTemp(t) && IsConditionerTurnedOff())

{

TurnConditionerOn();

}

else if(TemperatureBelowMinTemp(t)&&IsConditionerTurnedOn())

{

TurnConditionerOff();

}

}

}

-----------------------

temperature.h:

-----------------------

#ifndef TEMPERATURE\_H\_

#define TEMPERATURE\_H\_

void InitTemperatureSensor();

double GetTemperature();

#endif

-----------------------

temperature.cpp

-----------------------

#include "temperature.h"

#include <Arduino.h>

#include <Wire.h>

// Sparkfun BMP180 library

#include "SFE\_BMP180.h"

SFE\_BMP180 sensor;

void InitTemperatureSensor()

{

if (sensor.begin())

Serial.println("Temperature sensor init success");

else

{

Serial.println("Temperature sensr PMB180 init fail\n\n");

while(1); // Pause forever.

}

}

double GetTemperature(){

char status;

double T;

status = sensor.startTemperature();

if (status != 0)

{

// Wait for the measurement to complete:

delay(status);

// Retrieve the completed temperature measurement:

// The measurement is stored in the variable T.

// Function returns 1 if successful, 0 if failure.

status = sensor.getTemperature(T);

if (status != 0)

{

// Print out the measurement:

// Serial.print("temperature: ");

// Serial.print(T,2);

// Serial.print(" deg C, ");

}

else Serial.println("error retrieving temperature measurement\n");

}

else Serial.println("error starting temperature measurement\n");

return T;

}

-----------------------

led.h

-----------------------

#ifndef LED\_H\_

#define LED\_H\_

// builtin pin is 13 - LED\_BUILTIN

#define RED\_LED\_PIN 10

#define GREEN\_LED\_PIN 11

#define LED\_ON 1

#define LED\_OFF 0

void LEDs\_Init();

int Is\_RED\_LED\_On();

int Is\_GREEN\_LED\_On();

void Turn\_LED\_On(int led\_pin);

void Turn\_LED\_Off(int led\_pin);

#endif /\* LED\_H\_ \*/

-----------------------

led.cpp

-----------------------

#include "led.h"

#include <Arduino.h>

#include <Wire.h>

void LEDs\_Init() {

pinMode(GREEN\_LED\_PIN, OUTPUT);//Change to output my pins

pinMode(RED\_LED\_PIN, OUTPUT);

digitalWrite(GREEN\_LED\_PIN,LED\_OFF);//Turn off LED

digitalWrite(RED\_LED\_PIN,LED\_OFF);//Turn off LED

}

int Is\_RED\_LED\_On() {

int LEDState = digitalRead(RED\_LED\_PIN);

if(LEDState==LED\_ON) {

return 1;

}

//else

return 0;

}

int Is\_GREEN\_LED\_On() {

int LEDState = digitalRead(GREEN\_LED\_PIN);

if(LEDState==LED\_ON) {

return 1;

}

//else

return 0;

}

void Turn\_LED\_On(int led\_pin) {

digitalWrite(led\_pin,LED\_ON);

}

void Turn\_LED\_Off(int led\_pin) {

digitalWrite(led\_pin,LED\_OFF);

}

-----------------------

conditioner.h

-----------------------

#ifndef CONDITIONER\_H\_

#define CONDITIONER\_H\_

void ConditionerInit();

void TurnConditionerOn();

void TurnConditionerOff();

int IsConditionerTurnedOn();

int IsConditionerTurnedOff();

#endif

-----------------------

conditioner.cpp

-----------------------

#include "led.h"

void ConditionerInit() {

LEDs\_Init();

}

void TurnConditionerOn() {

Turn\_LED\_Off(GREEN\_LED\_PIN);

Turn\_LED\_On(RED\_LED\_PIN);

}

void TurnConditionerOff() {

Turn\_LED\_Off(RED\_LED\_PIN);

Turn\_LED\_On(GREEN\_LED\_PIN);

}

int IsConditionerTurnedOn() {

if(Is\_RED\_LED\_On()==1 && Is\_GREEN\_LED\_On()==0) {

return 1;

}

return 0;

}

int IsConditionerTurnedOff() {

if(Is\_RED\_LED\_On()==0) {

return 1;

}

return 0;

}

For BMP180 sensor - Library “SFE\_BMP180” from sparkfun

1. **Motor PID control**

---------------

motor.h

---------------

// Motor A

#define enA 10

#define in1 11

#define in2 12

void InitMotor();

void MotorForward(int speed);

void MotorBackward(int speed);

void MotorStop();

---------------

motor.cpp

---------------

void InitMotor() {

pinMode(enA, OUTPUT);

pinMode(in1, OUTPUT);

pinMode(in2, OUTPUT);

}

void MotorForward(int speed) {

digitalWrite(in1, LOW);

digitalWrite(in2, HIGH);

analogWrite(enA,speed);

}

void MotorBackward(int speed) {

digitalWrite(in1, HIGH);

digitalWrite(in2, LOW);

// analogWrite(enA,PowerToSpeed(power));

analogWrite(enA,speed);

}

void MotorStop() {

digitalWrite(in1, LOW);

digitalWrite(in2, LOW);

analogWrite(enA,0);

}

-------------------

encoder.h

-------------------

#define ENCODER\_PIN 6 // change

int maxTimeOneRotation;

void EncoderInit() {

pinMode(ENCODER\_PIN,INPUT);

MotorForward(165);

delay(1000);

MotorForward(255);

delay(3000);

while(ReadEncoderValue()!=1);

unsigned long startTime = millis();

delay(50);

while(ReadEncoderValue()!=1);

unsigned long elapsedTime = millis()-startTime;

Serial.println("max time 1 rotation:");

Serial.println(elapsedTime);

maxTimeOneRotation = elapsedTime;

delay(3000);

}

double ReadEncoderValue() {

int value = digitalRead(ENCODER\_PIN);

return value;

}

--------------------

Main program

--------------------

#include “motor.h”

#include “encoder.h”

#include <PID\_AutoTune\_v0.h>

#include <PID\_v1.h>

double Setpoint, Input, Output;

//Specify the links and initial tuning parameters

// Kp, Ki, Kd

PID myPID(&Input, &Output, &Setpoint,600,100,10, DIRECT);

PID\_ATune aTune(&Input, &Output);

void PID\_Init() {

//turn the PID on

myPID.SetMode(AUTOMATIC);

}

void SerialInit() {

Serial.begin(9600);

}

unsigned long currentTime;

unsigned long prevTime;

void TimeInit() {

currentTime = millis();

prevTime = millis();

}

void setup()

{

TimeInit();

SerialInit();

Setpoint = 230;

//AutoTuneInit();

// TunePID();

EncoderInit();

PID\_Init();

}

double CalculateMotorSpeed(unsigned long time\_1\_rotation) {

//return 1000.0/time\_1\_rotation;

return 10000/time\_1\_rotation;

}

unsigned long CalculateElapsedTime(unsigned long prevTime, unsigned long currentTime) {

return currentTime - prevTime;

}

int AdjustKp(int kp, int ki, int kd) {

Serial.println("Adjust kp");

Serial.println(kp);

myPID.Compute();

//Wait for overshooting

while(Output <Setpoint) {

Serial.println("kp:");

Serial.println(kp);

Serial.println("Output:");

Serial.println(Output);

Serial.println("Input:");

Serial.println(Input);

Serial.println("Setpoint");

Serial.println(Setpoint);

kp+=100;

myPID.SetTunings(kp,ki,kd);

myPID.Compute();

}

Serial.println("kp overshooting");

Serial.println(kp);

kp = kp/2;

Serial.println(kp);

return kp;

}

int AdjustKi(int kp, int ki, int kd) {

myPID.SetTunings(kp,ki,kd);

myPID.Compute();

while(Output <Setpoint) {

ki=10;

myPID.SetTunings(kp,ki,kd);

myPID.Compute();

Serial.println("ki overshooting");

Serial.println(ki);

ki = ki/2;

AdjustKp(kp,ki,kd);

Serial.println(ki);

}

return ki;

}

int AdjustKd(int kp, int ki, int kd) {

kd++;

return kd;

}

void TunePID() {

Serial.println("Tune PID");

int kp = 1;

int ki = 1;

int kd = 1;

Output = 0;

Setpoint = 200;

Input = maxTimeOneRotation/2;

kp = 10;

myPID.SetTunings(kp,ki,kd);

kp = AdjustKp(kp, ki, kd);

myPID.SetTunings(kp,ki,kd);

ki = 10;

myPID.SetTunings(kp,ki,kd);

kp = AdjustKp(kp,ki,kd);

myPID.SetTunings(kp,ki,kd);

ki = AdjustKi(kp, ki, kd);

myPID.SetTunings(kp,ki,kd);

kd = 3;

myPID.SetTunings(kp,ki,kd);

kp = AdjustKp(kp,ki,kd);

myPID.SetTunings(kp,ki,kd);

ki = AdjustKi(kp, ki, kd);

myPID.SetTunings(kp,ki,kd);

kd = AdjustKd(kp, ki, kd);

myPID.SetTunings(kp,ki,kd);

Serial.println("--------");

Serial.println("test after adjunsting:");

myPID.Compute();

Serial.println("Output:");

Serial.println(Output);

Serial.println("Setpoint:");

Serial.println("Setpoint");

}

int motorSpeed;

void loop()

{

// O rotatie completa

if(ReadEncoderValue()==1) {

prevTime = currentTime;

currentTime = millis();

// motorSpeed = 110;

unsigned long elapsedTime = CalculateElapsedTime(prevTime, currentTime);

Serial.println("Elapsed Time:");

Serial.println(elapsedTime);

motorSpeed = CalculateMotorSpeed(elapsedTime);

Serial.println("---Motor speed:----");

Serial.println(motorSpeed);

}

Input = motorSpeed;

Serial.println("Input:");

Serial.print(Input);

Input = map(Input, 0, maxTimeOneRotation, 0, 255);

myPID.Compute();

Serial.println("Input:");

Serial.print(Input);

Serial.println("Output:");

Serial.print(Output);

MotorForward(Output);

}