STR (online)
Real Time Systems

Lab 4:
Proportional-Integral-Derivative position control of a DC-motor with encoder using Tinkercad

FIB Barcelona School of Informatics

UPC Technical University of Catalonia

Antonio Camacho Santiago

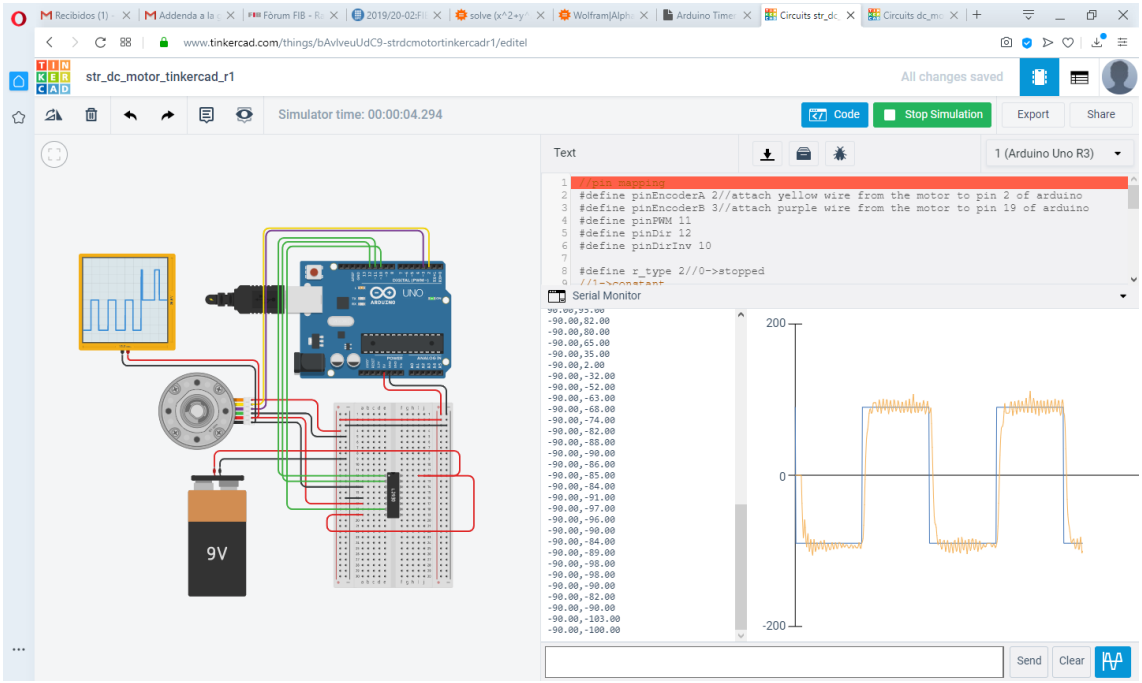
April 2020

Table of contents

Objective of the lab

The objective of the online lab is to control a dc motor using the Arduino Uno board.

A screenshot of the expected results is shown below



The work to be done is to get the encoder data using external interrupts, and to develop a proportional+integral+derivative task to control the motor position.

The motor should be regulated to follow a reference path turning from -90 degrees to +90 degrees each second. In the above figure, the reference is plotted in blue, while the motor angle is plotted in orange.

The main parts of the system are

-Arduino Uno rev3 microcontroller

-DC-motor with encoder

-9Volts battery to feed the motor

-L293D motor driver

How it works:

The motor can rotate in both directions by using some output pins of the microcontroller. Also, the speed of rotation can be effectively controlled by changing the duty cycle of any pulse-width-modulator capable pin of the Arduino board.

The motor driver consists of several power switches that can manage the voltage and current needed by the motor. Note that the driver is feed by the 9V battery.

Each time that the motor rotates, it generates some pulses in the encoder. These pulses are attached to some pins of the microcontroller. By counting the number and type of pulses, the motor position (or the angle) can be computed.

Once the reference and the angle are obtained, a control action need to be computed. This control value is the duty cycle and the direction to be translated to the motor driver.

TODO list

The work must be done individually.

The deadline of this lab is May 14th of 2020.

A complete list of the work to be done is presented below.

TODO #1: Build and connect the hardware parts.

TODO #2: Develop the code inside the ISREncoderA and ISREncoderB functions to get the motor angle.

TODO #3: Write a PID controller tasks running at 100Hz (0.01 seconds) to compute the duty cycle.

TODO #4: Join everything together to mimic the results shown in the first figure of this report.

Sent code, screenshot and share link to the raco.

TODO #5: Convert the code into tasks using the FreeRTOS template available at https://raco.fib.upc.edu/avisos/veure.jsp?id=105181&assig=GRAU-STR.

Sent *.cpp code to the raco

TODO #6 (optional): Include a User Datagram Protocol (UDP) communication task into the previous FreeRTOS based code to connect a host PC with the Arduino Uno board using the Ethernet shield.

Deliverables:

-The tinkercad code to implement the functionality in a separate file

-A screenshot of the project in a separate image file

-A share link from tinkercad to test the project

-The eclipse code to convert the previous functionality into tasks

-(Optional) The eclipse code to communicate via udp

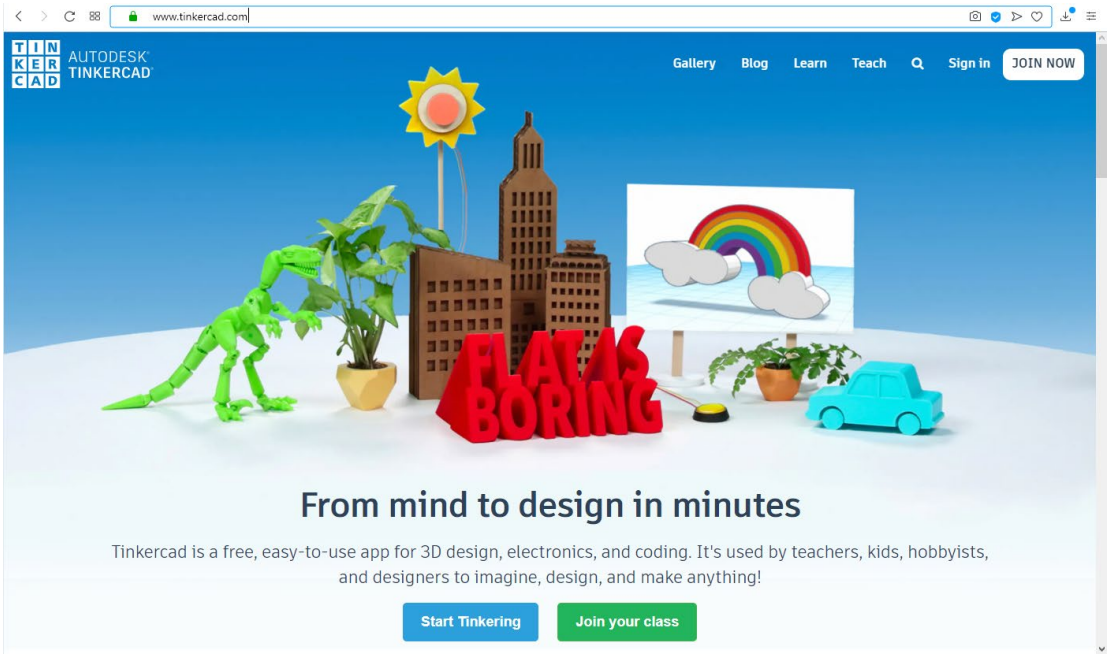
Sent everything to

https://raco.fib.upc.edu/practiques/practica.jsp?espai=270071&action=view&id=74323

Introduction to tinkercad

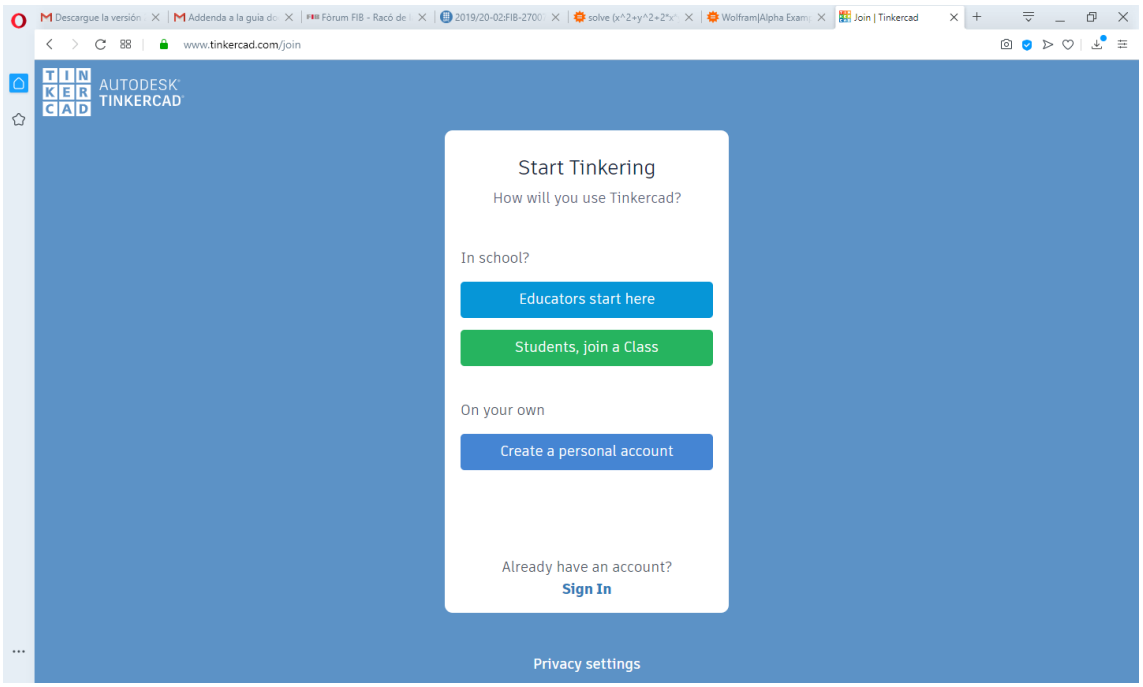
A brief demo to start working with tinkercad is presented below.

Tinkercad is a web based application. To start a new project go to https://www.tinkercad.com

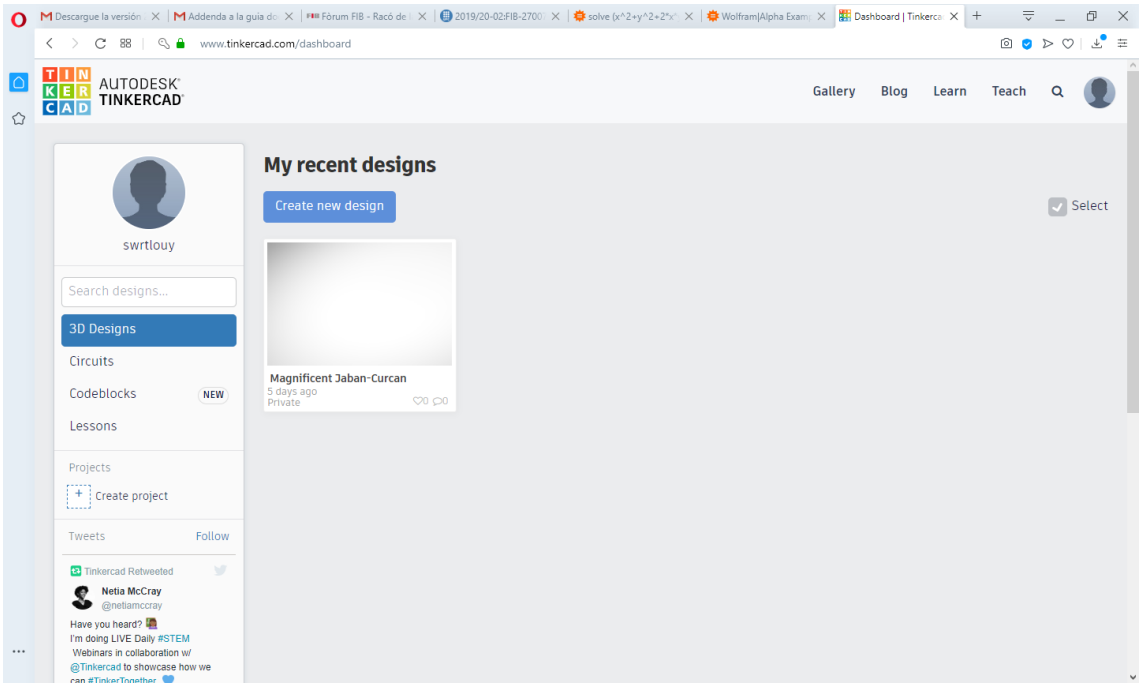


Click Start Tinkering.

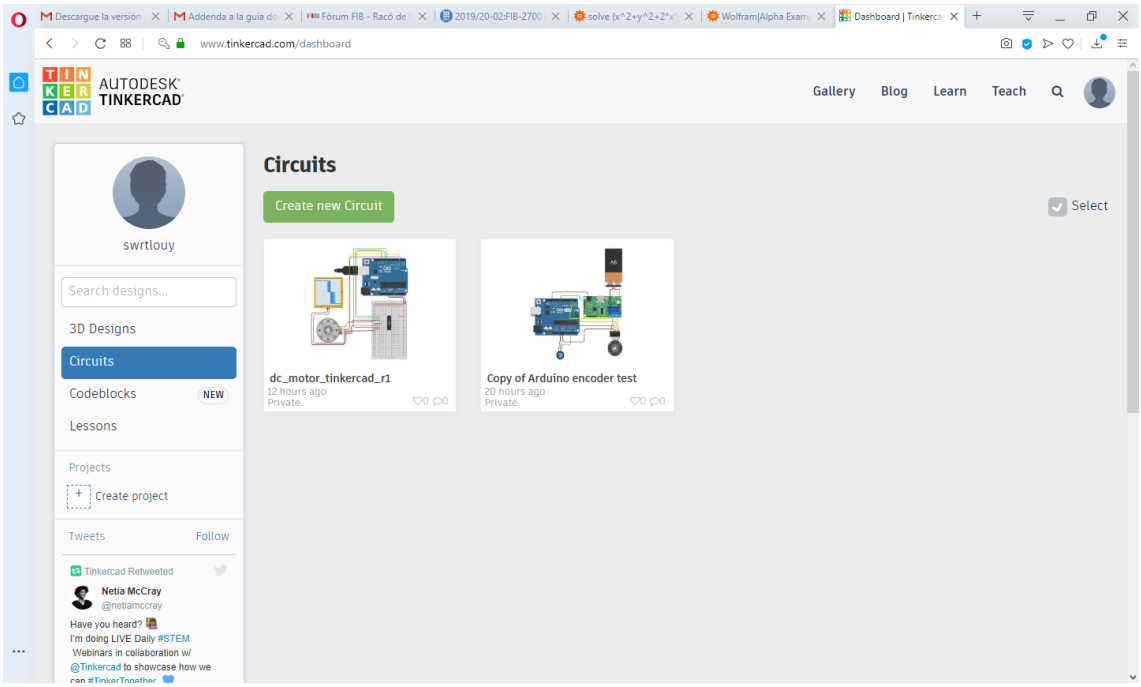
Log in with a personal account.



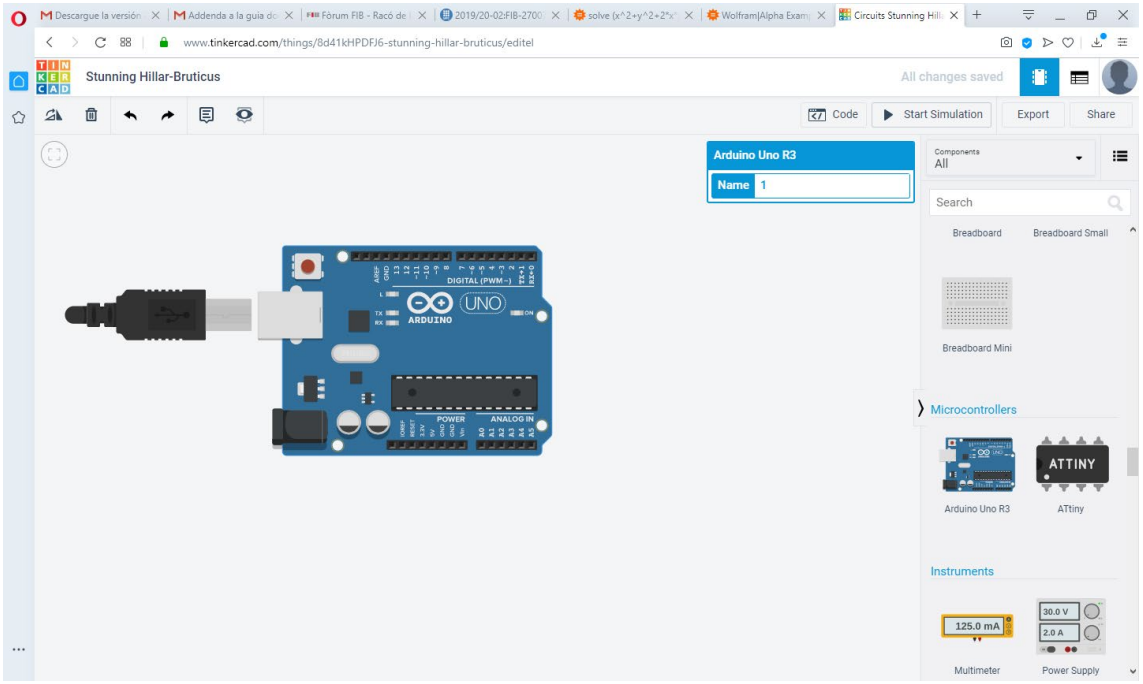
The main page is as follows



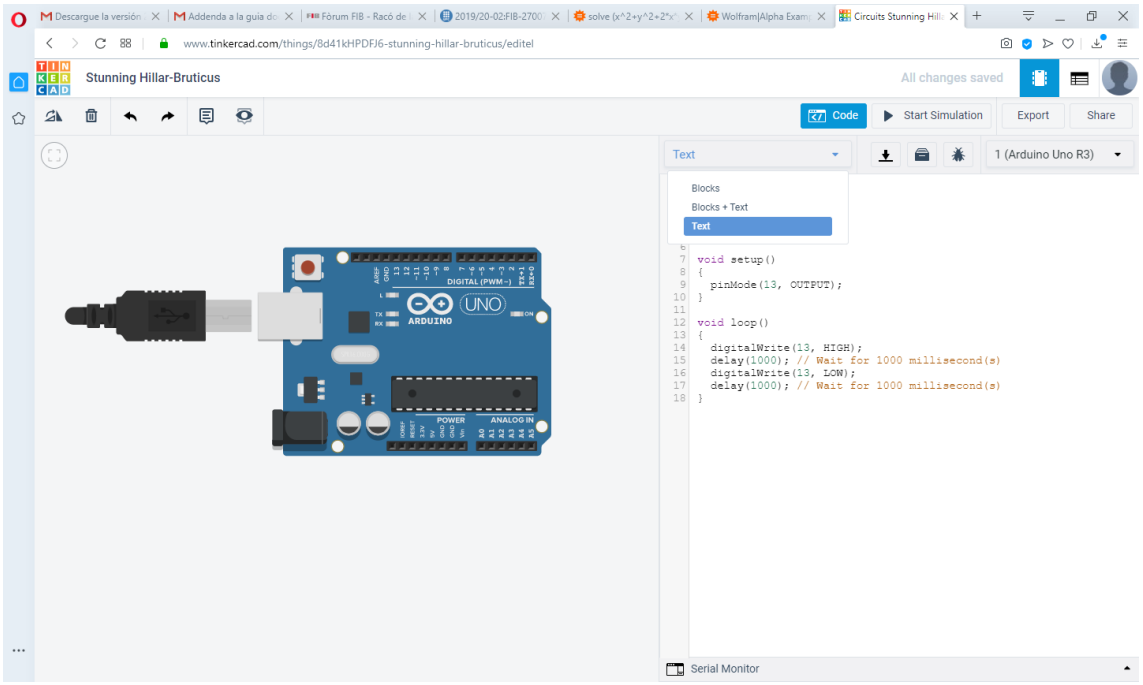
Click on Circuits button and Create new Circuit



The first project to start with tinkercad will be a blinking led. For doing so, we just need an Arduino microcontroller. Go to Components list item and select all. Drag and drop the Arduino Uno R3 device.



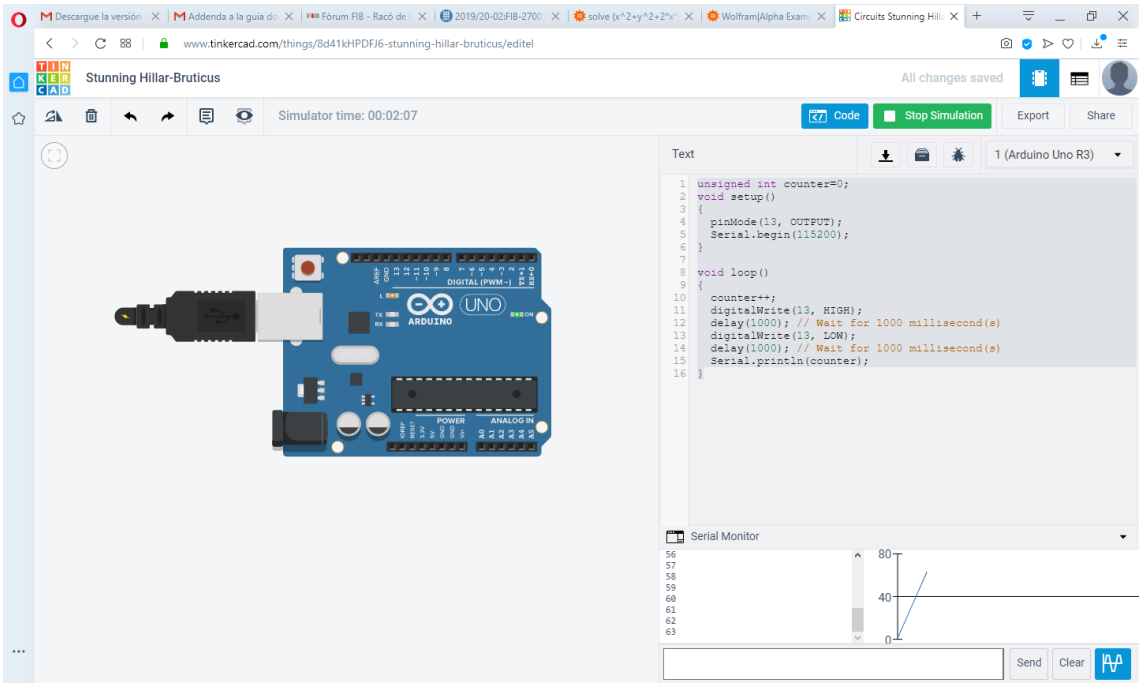
Select the Code button and change from Blocks to Text to start typing your code



Let’s add some communication functionality by using the serial monitor. We will send the counter value over the serial link with the following code.

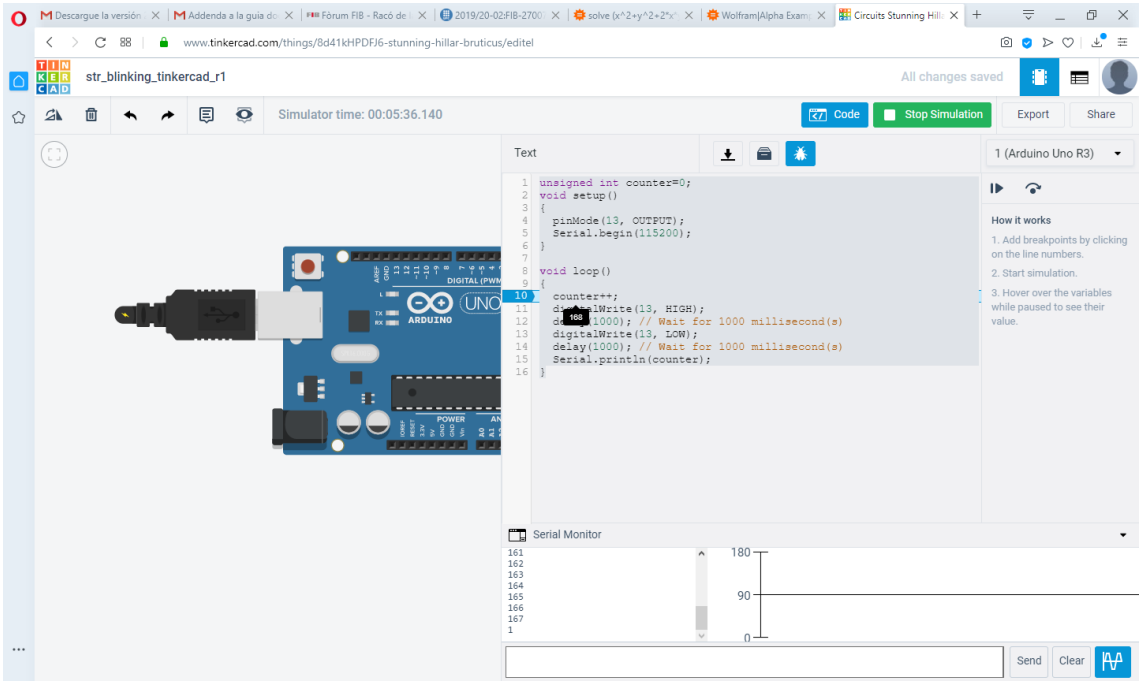
```
unsigned int counter=0;
void setup()
{
  pinMode(13, OUTPUT);
  Serial.begin(115200);
}

void loop()
{
  counter++;
  digitalWrite(13, HIGH);
  delay(1000); // Wait for 1000 millisecond(s)
  digitalWrite(13, LOW);
  delay(1000); // Wait for 1000 millisecond(s)
  Serial.println(counter);
}
```

To start the simulation, click the play button and the Serial Monitor button. Also the Serial Plotter is available by clicking the graph button on the bottom-right part of the screen.



Note that the built-in led of the Arduino board is blinking. It is labelled as “L” close to the Arduino logo.

It is possible to debug the code by clicking on the debug button and then click over the line number where the breakpoint will be located.



To continue the code execution, just click the Resume execution button. To use the step by step debugging functionality, click on the step over next function button.

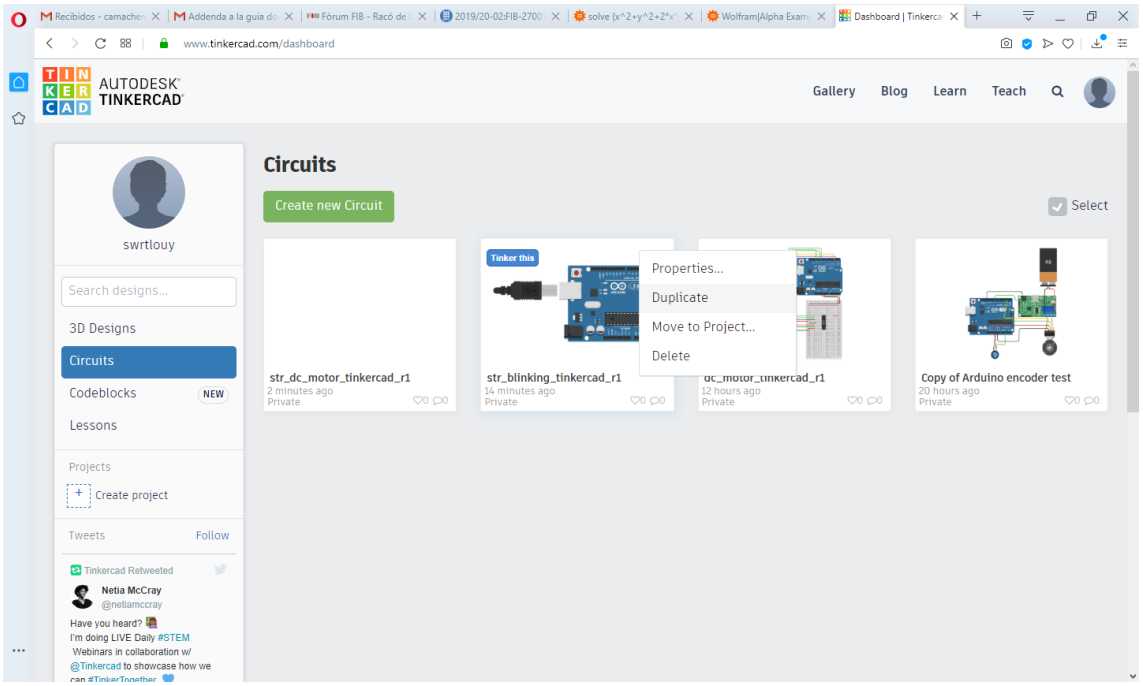
A link to the project is available at:

https://www.tinkercad.com/things/8d41kHPDFJ6-strblinkingtinkercadr1/editel?sharecode=VsmGRYmGDzutWoDIRv6bcs3D9XeDqaBTWMhGuybzSAc

Assembling parts

Until now, we have presented the basics for a new project. Now let's start with our project. It is based on controlling the position of a dc motor (actuator) with encoder (sensor).

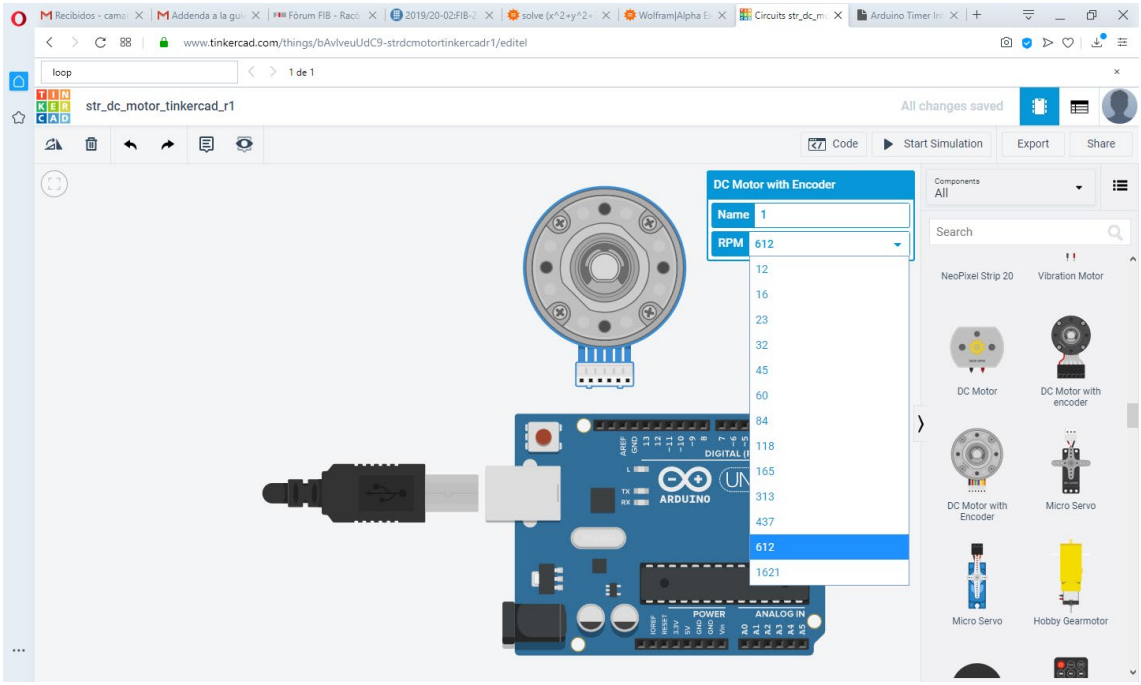
Let's duplicate the previous project to start a new one. Go to the main page by clicking on the top-left icon of tinkercad. Move around the project and click the options button. Select duplicate.



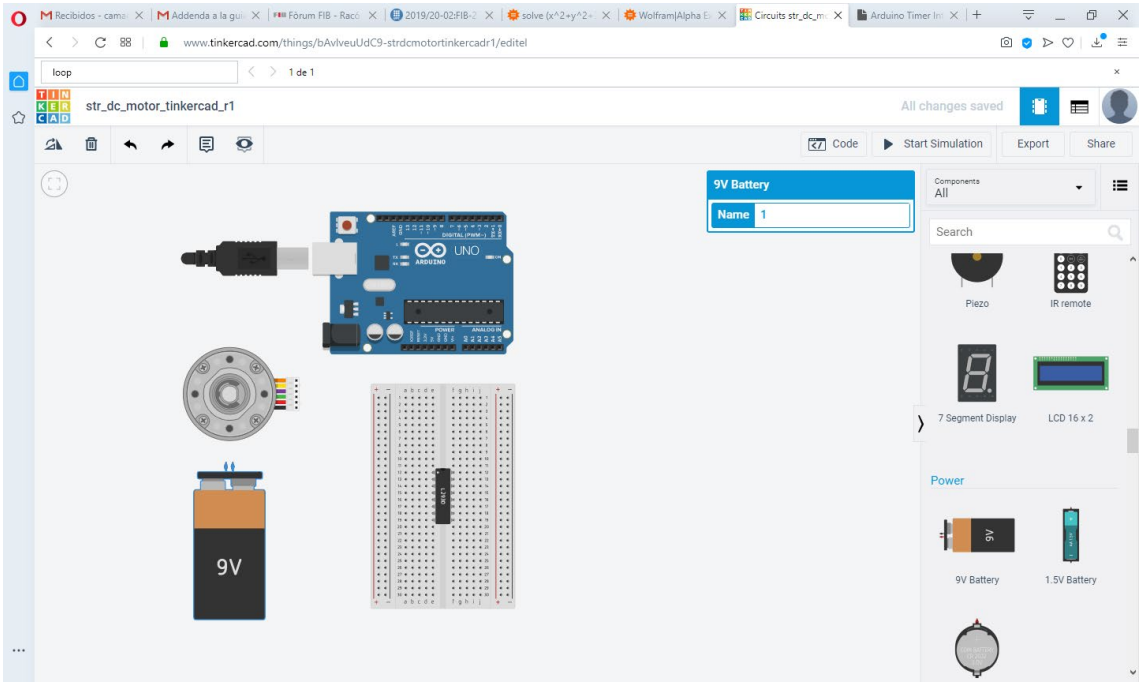
The project is based on controlling a motor with encoder. We will do this by using a single task executed each 10 milliseconds. The task will be triggered by a timer1 interrupt and basically it will execute a PID algorithm. The skeleton is as follows

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);

  Serial.begin(115200);

  // TIMER 1 for interrupt frequency 100 Hz:
  cli(); // stop interrupts
  TCCR1A = 0; // set entire TCCR1A register to 0
  TCCR1B = 0; // same for TCCR1B
  TCNT1  = 0; // initialize counter value to 0
  // set compare match register for 100 Hz increments
  OCR1A = 19999; //=16000000/(8*100)-1(must be <65536)
  // turn on CTC mode
  TCCR1B |= (1 << WGM12);
  // Set CS12, CS11 and CS10 bits for 8 prescaler
  TCCR1B |= (0 << CS12) | (1 << CS11) | (0 << CS10);
  // enable timer compare interrupt
  TIMSK1 |= (1 << OCIE1A);
  sei(); // allow interrupts

}

unsigned long isr_timer1_count=0;
ISR(TIMER1_COMPA_vect)
{
  digitalWrite(LED_BUILTIN, digitalRead(LED_BUILTIN) ^ 1);

  isr_timer1_count++;

}

void loop()
{
  // put your main code here, to run repeatedly:
}
```

To include the motor with encoder go to components→all→DC Motor with Encoder. We will change the RPM configuration value from 16 to 612 to mimic the real motor.



We need a motor driver, the L293D H-bridge is the same as the one we use in the lab. Include it, a 9V battery and a breadboard to easily connect everything. Remember to connect the motor driver in between the breadboard!



The connections are as follows

Breadboard + to Arduino 5V

Breadboard - to Arduino GND

Encoder Power (Orange) to 5V

Encoder Channel A (Yellow) to Arduino pin 2

Encoder Channel B (Purple) to Arduino pin 3

Encoder Ground (Green) to GND

Motor Positive (Red) to L293D Output 2

Motor Negative (Black) to L293D Output 1

Battery 9V + to L293D Power 1

Battery 9V + to L293D Power 2

Battery 9V - to L293D GND

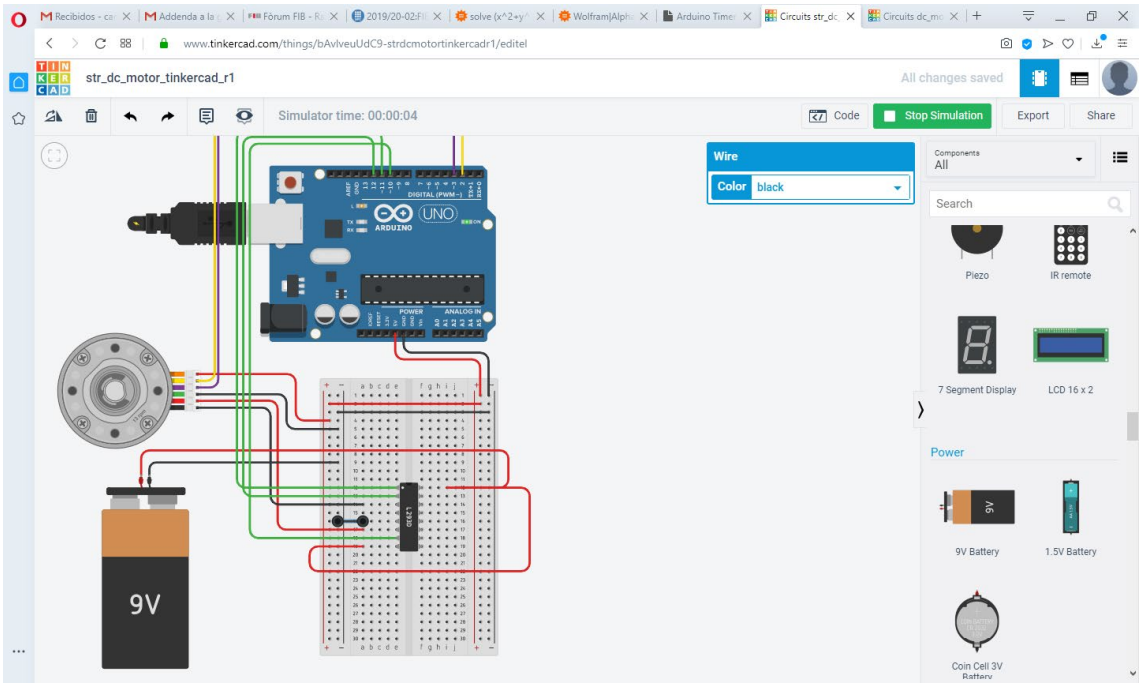
Battery 9V - to Arduino GND

Arduino pin 11 to L293D Enable 1&2

Arduino pin 12 to L293D Input 1

Arduino pin 10 to L293D Input 2

The connections are shown in the next figure.



The next code implements the basic functionality

```
//pin mapping
#define pinEncoderA 2//attach yellow wire from the motor to pin 2 of arduino
#define pinEncoderB 3//attach purple wire from the motor to pin 19 of arduino
#define pinPWM 11
#define pinDir 12
#define pinDirInv 10

float u=0;
signed int ISRCounter=0;//used inside the ISR

//function prototypes
void ISREncoderA(void);
void ISREncoderB(void);

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(LED_BUILTIN,OUTPUT);
  pinMode(pinEncoderA, INPUT_PULLUP);
  pinMode(pinEncoderB, INPUT_PULLUP);
  pinMode(pinDir, OUTPUT);
  pinMode(pinDirInv, OUTPUT);
  pinMode(pinPWM, OUTPUT);
  attachInterrupt(digitalPinToInterrupt(pinEncoderA), ISREncoderA, CHANGE);
  attachInterrupt(digitalPinToInterrupt(pinEncoderB), ISREncoderB, CHANGE);
  Serial.begin(115200);

  // TIMER 1 for interrupt frequency 100 Hz:
  cli(); // stop interrupts
  TCCR1A = 0; // set entire TCCR1A register to 0
  TCCR1B = 0; // same for TCCR1B
  TCNT1  = 0; // initialize counter value to 0
  OCR1A = 19999; // = 16000000 / (8 * 100) - 1 (must be <65536)
  TCCR1B |= (1 << WGM12);
  TCCR1B |= (0 << CS12) | (1 << CS11) | (0 << CS10);
  TIMSK1 |= (1 << OCIE1A);
  sei(); // allow interrupts
}

unsigned long isr_timer1_count=0;
ISR(TIMER1_COMPA_vect)
{
  digitalWrite(LED_BUILTIN, digitalRead(LED_BUILTIN) ^ 1);
  isr_timer1_count++;
  if (isr_timer1_count%200>100)
  {
    u=10;
  }else{
    u=-10;
  }
  //check for motor direction
  if (u>=0)
  {
    digitalWrite(pinDir,0);
    digitalWrite(pinDirInv,1);
  }else{
    digitalWrite(pinDir,1);
    digitalWrite(pinDirInv,0);
  }
  //check for saturation
  if (u>255.0)
  {
    u=255.0;
  }
  if (u<-255.0)
  {
    u=-255.0;
  }
  analogWrite(pinPWM,abs(u));//update output

  Serial.println(ISRCounter);
}

void ISREncoderA(void)
{
//Fill this with your code
}

void ISREncoderB(void)
{
//Fill this with your code
}


void loop()
{
  // put your main code here, to run repeatedly:
}
```

The first step for developing the lab is to fill the code in the ISREncoderA and ISREncoderB functions. These functions are attached to interrupts when an external pin changes. Therefore, when the encoder A or B generates a rising or falling edge, the interrupt is triggered, and then it is possible to detect the position of the motor. A quick search on the web will help you to develop this part of the code. Multiple solutions exist, however the idea is to don't miss any pulse.

The above presented code turns the motor clock and counter-clock wise each second in order start rotating the motor. However, after checking that the angle measurement works properly, the computation of the control action `u=10` and `u=-10` after the `if (isr_timer1_count%200>100)` line should be modified by the proportional+integral+derivative (PID) computation. For doing this, you need to implement the following equations

$$error = reference\ value - encoder\ position$$

$$u = k_p \cdot error + k_i \cdot \int error\ dt + k_d \cdot \frac{derror}{dt}$$

Where k_p, k_i and k_d are control parameters that must be determined by trial and error

Note that when selecting the control gains, the following is observed:

Increasing k_p yields to faster dynamics.

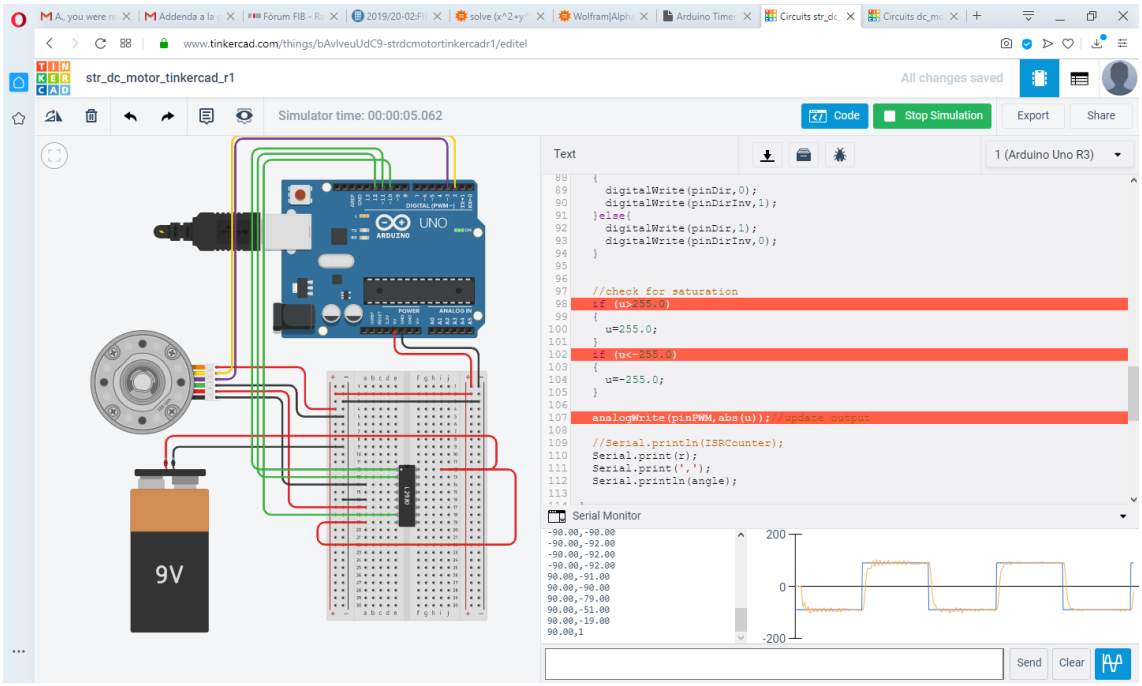
Increasing too much k_p yields to unstable dynamics. Therefore, there is a trade-off between the values that this gain can achieve.

Whenever k_i is zero, the error will never vanish since the integral part tends to cancel the steady-state error as time evolves.

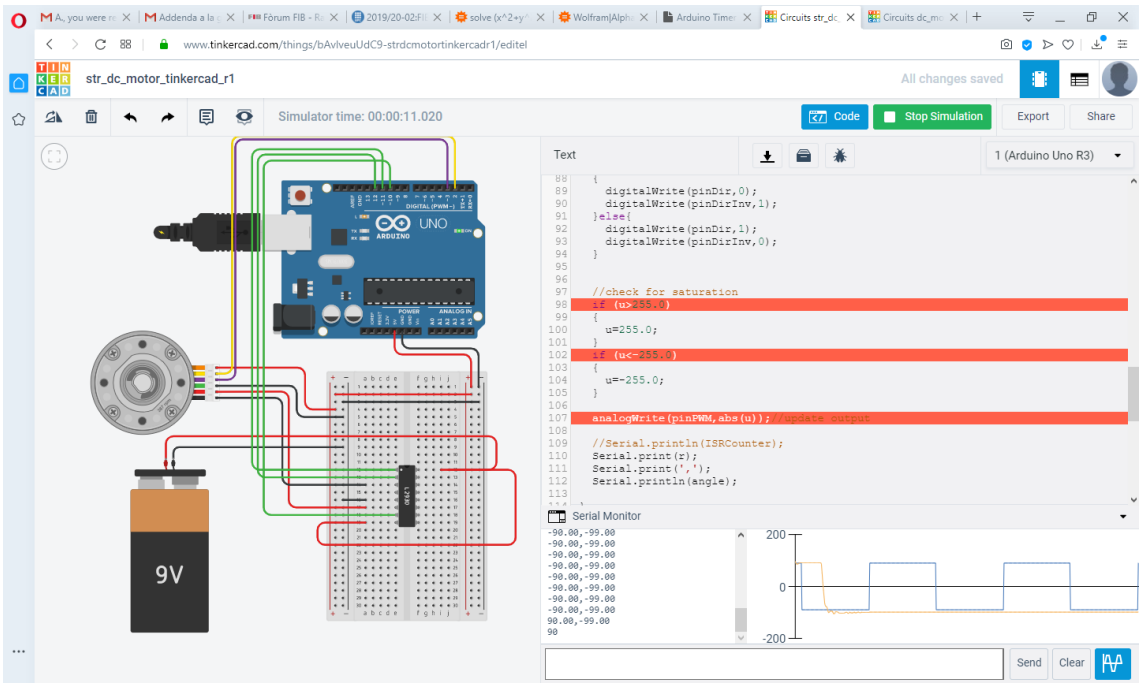
A small value of k_d tends to improve the motor dynamics during the reference change, avoiding large excursions and overshoots.

To implement the PID algorithm, you have to compute the discrete integral and derivative inside the microcontroller.

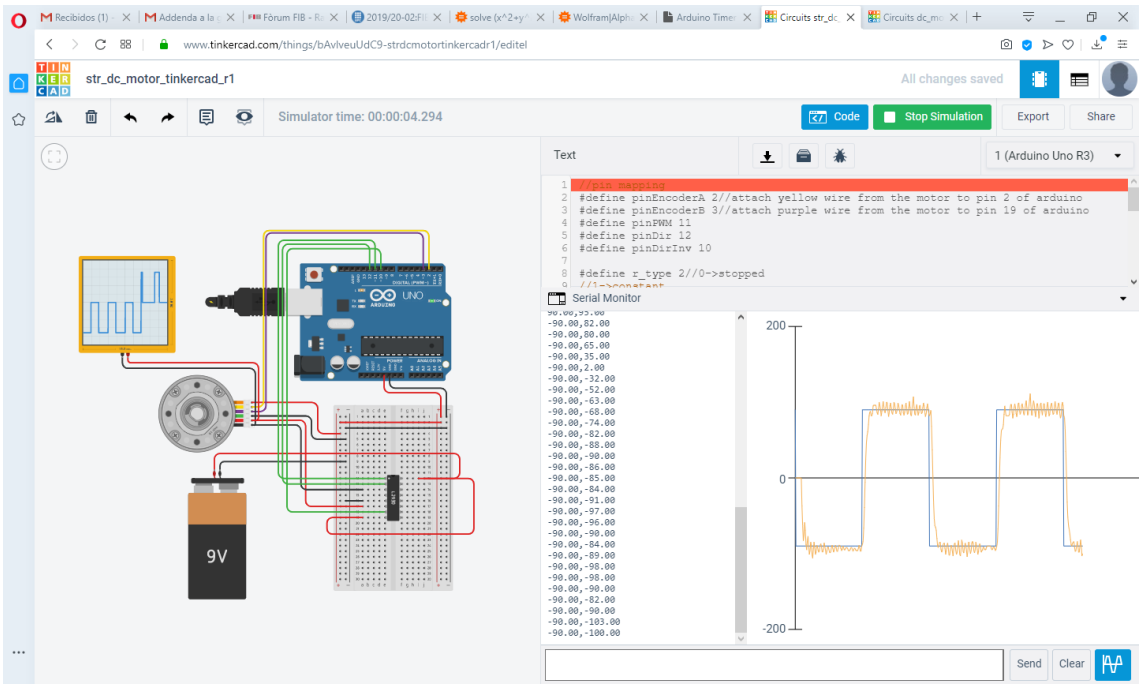
The result should be something similar to



KNOWN BUG: Sometimes the motor simulator stops working, and there are no more encoder pulses. This seems to be a problem with the motor simulator, not with the code. So, whenever this happens, stop the simulation and start again. The screenshot below shows the bug.



The expected results for the lab are shown below. Note that the motor dynamics is not of interest. By adjusting the control gains k_p, k_i and k_d they can be improved but it is not really needed to spend too much time adjusting these gains.



Once this has been obtained, the next step is to translate these tasks to FreeRTOS. However this cannot be done in tinkercad. So, the code must be translated into eclipse, the tasks need to be created, and the whole code have to be sent to the raco. I will test it and sent you back feedback to improve the operation of the dc-motor.

Also, udp communication between the arduino and a host pc will be nice to have. A new task performing this functionality inside the eclipse template with FreeRTOS will be of interest. This last part is optional.

Do not hesitate to contact me at antonio.camacho.santiago@upc.edu for further information.