**UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS**

(Universidad del Perú, DECANA DE AMÉRICA)

**FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA**



**“SISTEMA DE CONTROL DE VELOCIDAD PARA UN VEHÍCULO ELÉCTRICO”**

**CURSO** **:** INGENIERÍA DE CONTROL

**PROFESOR** **:** RAUL MARCELO ARMAS CALDERÓN

**INTEGRANTES** **:**

* Cabello Cordova Ronal Jairo 17200040
* Artica Villarroel Jose Luis 17200035
* Alderete Cruzatt Miguel 17200131
* Alegre Saenz Diego 09200002

**LIMA – PERÚ**

**2021**

**INDICE**

[**INTRODUCCIÓN 4**](#_Toc81166957)

[**OBJETIVOS 4**](#_Toc81166958)

[**DIAGRAMA DE BLOQUES 4**](#_Toc81166959)

[**DESCRIPCIÓN DE COMPONENTES 5**](#_Toc81166960)

[**Arduino UNO 5**](#_Toc81166961)

[**Controladora 5**](#_Toc81166962)

[**Motor DC 5**](#_Toc81166963)

[**Vehículo eléctrico 6**](#_Toc81166964)

[**HARDWARE 7**](#_Toc81166965)

[**Sensores 7**](#_Toc81166966)

[**Encoder 7**](#_Toc81166967)

[**Encoder de Cuadratura 7**](#_Toc81166968)

[**Funcionamiento 7**](#_Toc81166969)

[**Generación de desfasamiento de ambas señales 8**](#_Toc81166970)

[**Controlador 8**](#_Toc81166971)

[**L298N 8**](#_Toc81166972)

[**ESPECIFICACIONES TÉCNICAS 8**](#_Toc81166973)

[**PULSE WIDTH MODULATION (PWM) 9**](#_Toc81166974)

[**Actuadores 9**](#_Toc81166975)

[**Motor DC 9**](#_Toc81166976)

[**Ley de Lorentz 9**](#_Toc81166977)

[**Componentes 9**](#_Toc81166978)

[**Tipos de motores DC 10**](#_Toc81166979)

[**Motores de imán permanente 10**](#_Toc81166980)

[**Motores serie 10**](#_Toc81166981)

[**Shunt 11**](#_Toc81166982)

[**SOFTWARE 11**](#_Toc81166983)

[**PROTEUS DESIGN SUITE 11**](#_Toc81166984)

[**ARDUINO IDE 11**](#_Toc81166985)

[**SIMULACIÓN 12**](#_Toc81166986)

[**Motor-encoder proteus 12**](#_Toc81166987)

[**COMPIM 12**](#_Toc81166988)

[**Controlador PID del sistema de control 14**](#_Toc81166989)

[**Diseño del algoritmo 14**](#_Toc81166990)

[**Ajuste del algoritmo 14**](#_Toc81166991)

[**Obtención de datos desde proteus 14**](#_Toc81166992)

[**Lectura de datos en Matlab 15**](#_Toc81166993)

[**Función ident 15**](#_Toc81166994)

[**Implementación en simulink 16**](#_Toc81166995)

[**Ajuste de las ganancias PID 16**](#_Toc81166996)

[**Verificación 17**](#_Toc81166997)

[**Valores finales del kp,ki y kd 18**](#_Toc81166998)

[**CONCLUSIONES 19**](#_Toc81166999)

[**PRUEBA DEL SENSOR ENCODER 20**](#_Toc81167000)

[**Código Arduino 20**](#_Toc81167001)

[**Diagrama esquemático en proteus 20**](#_Toc81167002)

[**Resultados: 21**](#_Toc81167003)

[**CÁLCULO DE RPM 22**](#_Toc81167004)

[**Código en Arduino 22**](#_Toc81167005)

[**Diagrama esquemático en proteus 23**](#_Toc81167006)

[**Resultados: 24**](#_Toc81167007)

[**ADQUISICIÓN DE DATOS 24**](#_Toc81167008)

[**Código Arduino 24**](#_Toc81167009)

[**Diagrama esquemático en proteus 25**](#_Toc81167010)

[**CONTROL DE VELOCIDAD RPM DE UN MOTOR DC 26**](#_Toc81167011)

[**Código Arduino 26**](#_Toc81167012)

[**Diagrama esquemático en proteus 28**](#_Toc81167013)

# **INTRODUCCIÓN**

Los sistemas de control son un conjunto de dispositivos o componentes encargados de administrar, ordenar, dirigir o regular el comportamiento de otro sistema, con el fin de evitar las probabilidades de fallo y poder obtener los resultados deseados. El siguiente trabajo está basado en la implementación de un sistema de control de velocidad de un vehículo eléctrico, el sistema se encargará de aumentar o reducir la velocidad controlándola y acercándola a la velocidad deseada a partir de una velocidad de referencia, para la implementación del sistema de control se han utilizado diversos componentes tales como el arduino uno, un controlador y un motor DC.

## OBJETIVOS

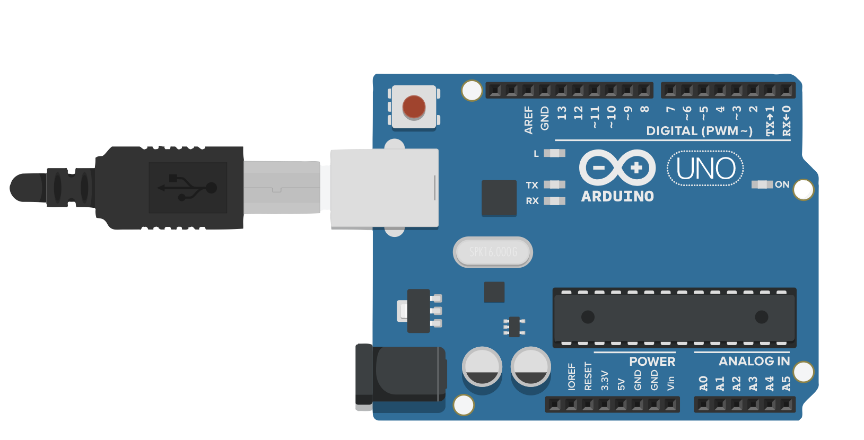
El presente trabajo se centra en el control de velocidad. Partiendo de una velocidad de referencia que nos facilitará un sistema de control inteligente, debemos obtener y enviar hacia el sistema de aceleración y de freno las señales que mejor adaptan la velocidad actual a la deseada.

De lo anterior se deduce la necesidad de una realimentación, de esta forma el sistema de control dispondrá tanto de la velocidad actual del vehículo como de la deseada, y podrá actuar en consecuencia.

## **DIAGRAMA DE BLOQUES**

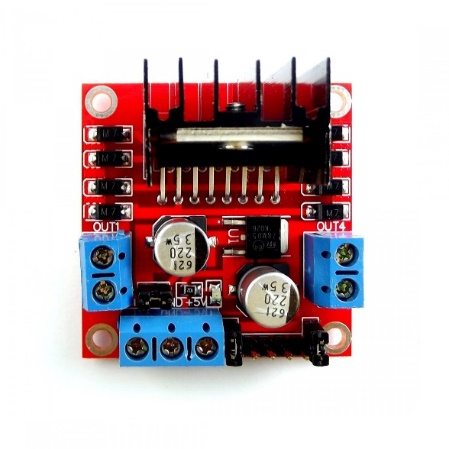
# DESCRIPCIÓN DE COMPONENTES

## Arduino UNO

Es una placa de microcontrolador de código abierto. La placa está equipada con conjuntos de pines de E/S digitales y analógicas que pueden conectarse a varias placas de expansión y otros circuitos. La placa tiene 14 pines digitales, 6 pines analógicos y programables con el Arduino IDE (Entorno de desarrollo integrado) a través de un cable USB.

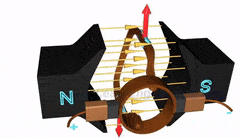
## Controladora

Componente que permite activar a los motores eléctricos en un sentido u otro y al mismo tiempo permite controlar variables como: velocidad y torque de los mismos. La controladora va ser el cerebro de nuestro vehículo.



## Motor DC

Es una máquina capaz de transformar energía eléctrica, suministrada en forma de corriente continua, en energía mecánica. Provoca un movimiento rotatorio, gracias a la acción de un campo magnético. Un motor de corriente continua está compuesto de un estator y un rotor. En muchos motores c.c., generalmente los más pequeños, el estator está compuesto de imanes para crear un campo magnético. El rotor es el dispositivo que gira en el centro del motor y está compuesto de arrollamientos de cable conductores de corriente continua.



## Vehículo eléctrico

Los vehículos eléctricos son aquellos cuyo motor es capaz de convertir la energía de una batería en energía mecánica, posibilitando el movimiento del vehículo en cuestión. Un vehículo eléctrico es un vehículo propulsado por uno o más motores eléctricos. Un vehículo eléctrico puede alimentarse a través de una fuente externa que suministre energía eléctrica, o pueden ser autónomos al tener instalados baterías, paneles solares, o un generador eléctrico que transforme un combustible en electricidad.

# HARDWARE

## Sensores

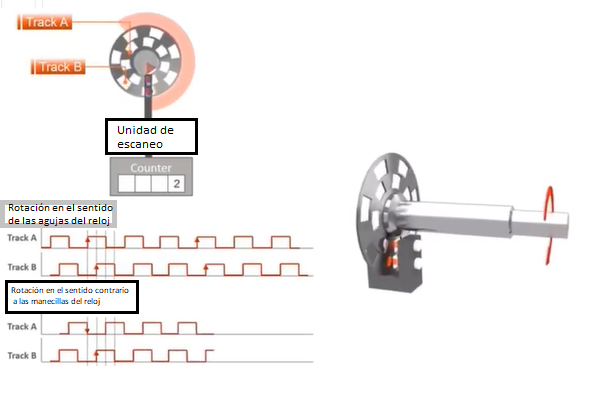
### **Encoder**

Los encoders son componentes que se añaden a un motor de corriente continua para convertir el movimiento mecánico en pulsos digitales que puedan ser interpretados por el sistema de electrónica de control integrado. El principal objetivo de los distintos tipos de encoders es el de transformar información de un formato a otro, con el propósito de estandarización, adecuación de la velocidad o control de la seguridad. Los motores DC tienen un complejo control de posición y de la velocidad, su comportamiento es no lineal y depende mucho de la carga que soportan; por este motivo necesitan de la aplicación de un encoder (que puede estar integrado o no) que permita conocer y asegurar la correcta posición del eje.

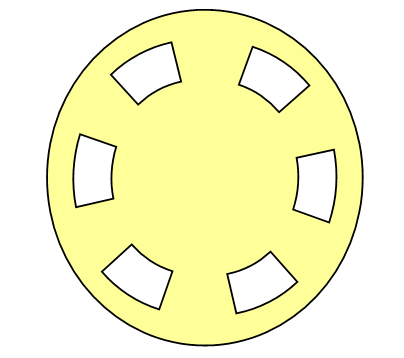
#### Encoder de Cuadratura

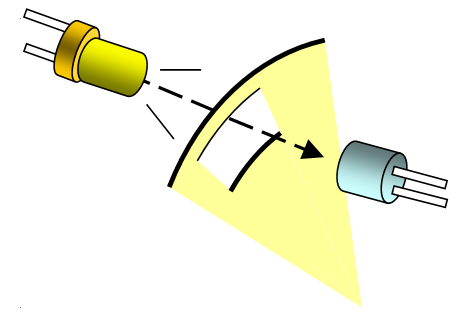
Es un tipo de encoder rotativo incremental (mide un desplazamiento angular con respecto a una posición de referencia) el cual tiene la capacidad para indicar la posición, dirección y velocidad del movimiento. En relación a sus aplicaciones, podemos encontrarlos en muchos productos eléctricos de consumo y en una infinidad de aplicaciones comerciales. Entre sus principales ventajas destaca su flexibilidad, sencillez y durabilidad.

Tiene al menos dos sensores ópticos



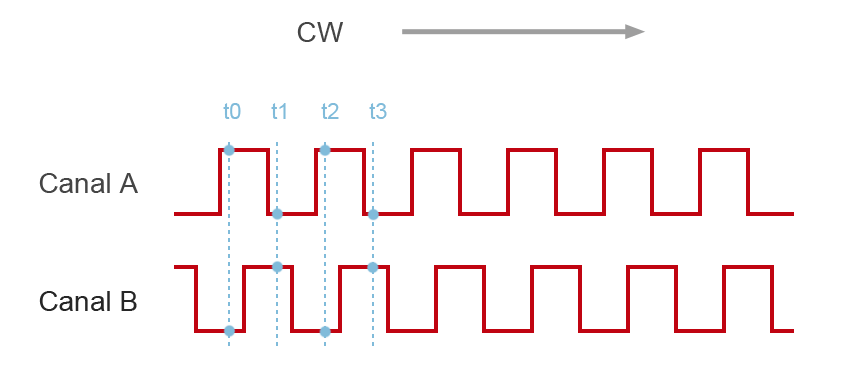
#### Funcionamiento

El disco que tiene el encoder de manera interna viene a ser de la siguiente forma. Dicho disco está compuesto con “ventanas” transparentes equidistantes del centro del disco de tal manera que dichas “ventanas” dejan pasar la luz. El número de agujeros determina la resolución del encoder. Conectado a la estructura fija va un fotodiodo o fototransistor iluminado de una fuente de luz puesta al frente e interrumpida por el disco como muestra.



#### Generación de desfasamiento de ambas señales

El estado del sensor determinará cuál es el sentido de giro. Los dos sensores se usarán como A y B:



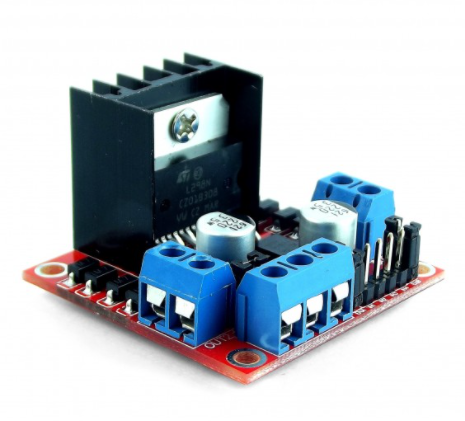
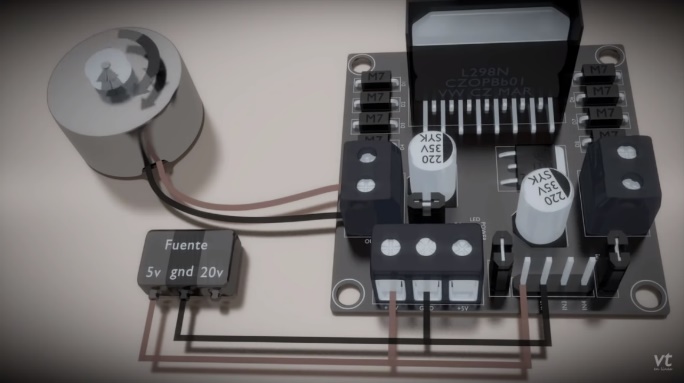
## Controlador

### L298N

El driver puente H L298N es el módulo más utilizado para manejar motores DC de hasta 2 amperios. El chip L298N internamente posee dos puentes H completos que permiten controlar 2 motores DC. El módulo permite controlar el sentido y velocidad de giro de motores que se pueden obtener de microcontroladores y tarjetas de desarrollo como Arduino o Raspberry Pi. El control del sentido de giro se realiza mediante dos pines para cada motor, la velocidad de giro se puede regular haciendo uso de modulación por ancho de pulso (PWM por sus siglas en inglés). Tiene integrado un regulador de voltaje LM7805 de 5V encargado de alimentar la parte lógica del L298N, el uso de este regulador se hace a través de un Jumper y se puede usar para alimentar la etapa de control.

#### ESPECIFICACIONES TÉCNICAS

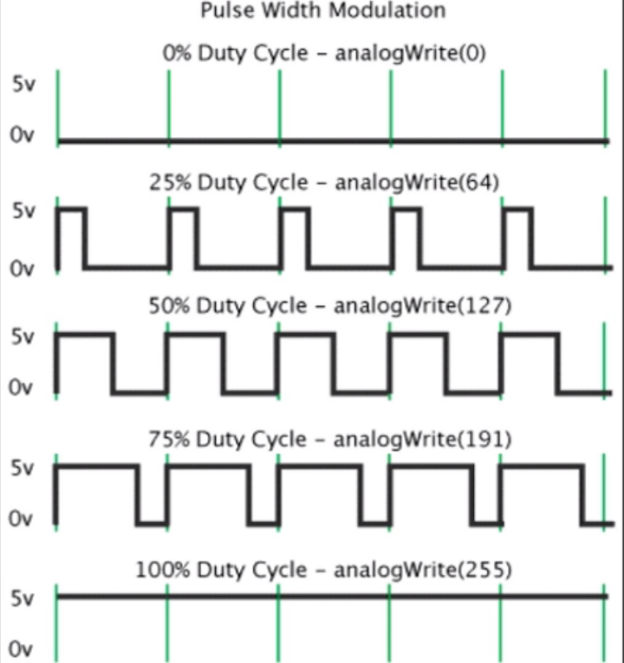
* Chip: L298N
* Canales: 2 (soporta 2 motores DC o 1 motor PAP)
* Voltaje lógico: 5V
* Voltaje de potencia (V motor): 5V - 35V DC
* Consumo de corriente (lógico): 0 a 36mA
* Capacidad de corriente: 2A (picos de hasta 3A)
* Potencia máxima: 25W
* Dimensiones: 43 \* 43 \* 27 mm
* Peso: 30g

****

#### PULSE WIDTH MODULATION (PWM)

La modulación por ancho de pulsos de una señal o fuente de energía es una técnica en la que se modifica el ciclo de trabajo de una señal periódica, ya sea para transmitir información a través de un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga.

Es una técnica para modificar el ancho de pulso y nos va permitir variar el voltaje de salida de un PIN y esto nos permite controlar la luminosidad de un LED o la velocidad de un MOTOR.



## Actuadores

### Motor DC

#### **Ley de Lorentz**

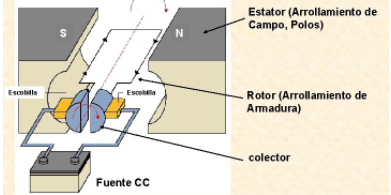
Una carga eléctrica en movimiento dentro de un campo magnético sufre una fuerza. Experimentalmente se comprueba que esta fuerza magnética ejercida por el campo es proporcional al valor de la carga y a su velocidad, y que la dirección de la fuerza es perpendicular a la velocidad de la carga.

#### **Componentes**

* Estator:

El estator es la parte inductora, la que produce el campo magnético. Está formado por una culata, en donde se coloca la bobina encargada de producir el campo magnético CC.

* Rotor:

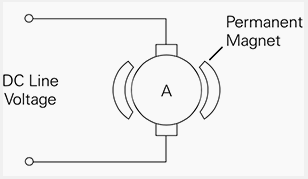
Se trata de la parte que gira o rota dentro de una máquina eléctrica, ya sea un motor o un generador eléctrico

#### 

### **Tipos de motores DC**

#### Motores de imán permanente

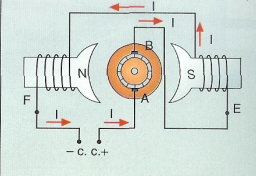
El motor de imán permanente lo utiliza. un imán para suministrar flujo de campo. Los motores de imán permanente DC tienen excelente capacidad de par de arranque con buena regulación de velocidad. Una desventaja de los motores de imán permanente de CC es que están limitados a la cantidad de carga que pueden manejar. Estos motores se pueden encontrar en aplicaciones de baja potencia.



#### Motores serie

En un motor de CC de serie, el campo se conecta en serie con la armadura. El campo se enrolla con algunas vueltas de cable grande porque debe llevar toda la corriente de armadura. Una característica de los motores de serie es el motor. Desarrolla una gran cantidad de par de arranque. Sin embargo, la velocidad varía ampliamente entre la no carga y la carga completa. Los motores de la serie no se pueden usar donde se requiere una velocidad constante bajo cargas variables.

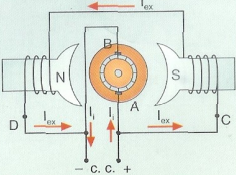
Adicionalmente, La velocidad de un motor en serie sin carga. Aumenta hasta el punto donde el motor puede dañarse. Alguna carga debe estar siempre conectada a un motor conectado en serie. Los motores conectados en serie generalmente no son adecuados para el uso en la mayoría de las aplicaciones de variadores de velocidad variable.



#### Shunt

En un motor de derivación el campo está conectado en paralelo (shunt) con los devanados de la armadura. El motor conectado en derivación ofrece una buena regulación de la velocidad. El devanado de campo puede ser excitado por separado o conectado a la misma fuente que la armadura.

Una ventaja para un campo de derivación excitado por separado. es la capacidad de un variador de velocidad para proporcionar un control independiente de la armadura y el campo.



# SOFTWARE

## PROTEUS DESIGN SUITE

Proteus es una aplicación para la ejecución de proyectos de construcción de equipos electrónicos en todas sus etapas: diseño del esquema electrónico, programación del software, construcción de la placa de circuito impreso, simulación de todo el conjunto, depuración de errores, documentación y construcción.

En el proyecto sirvió para diseñar y simular el sistema retroalimentado con el uso de arduino, los sensores, y actuadores.

## **ARDUINO IDE**

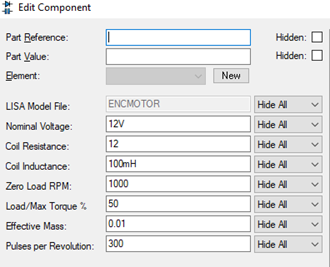
El Arduino IDE es una aplicación multiplataforma que está escrita en el lenguaje de programación Java. Se utiliza para escribir y cargar programas en placas compatibles con Arduino, pero también, con la ayuda de núcleos de terceros, se puede usar con placas de desarrollo de otros proveedores.

En este trabajo, se usó este software para la programación de las acciones que hará el Arduino Uno en base a los datos recogidos por el encoder y enviando las señales al L298N para controlar al motor DC

# **SIMULACIÓN**

## Motor-encoder proteus

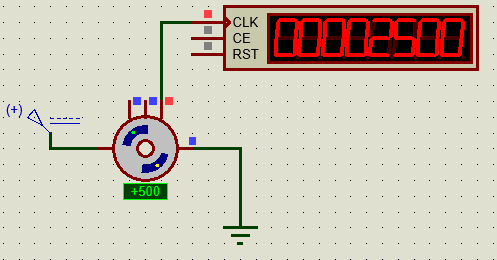
Las RPM de carga cero son 360 12v de suministro (voltaje nominal), y los pulsos por revolución en los pines de salida se establecen en 300 (resolución). Y la carga del motor se establece en 50%.



Esto es lo que se ve cuando se abre las propiedades del motor.

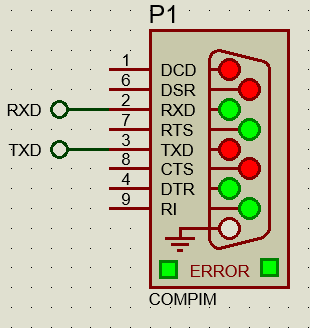
Si se aplica suministro de 12v al motor, la pantalla debajo del motor debe mostrar 500 RPM (que es el 50% de 1000 RPM) y los pulsos en la salida deben ser:

Eso es de hecho lo que verá si se simula el circuito con 12v conectado al motor y un medidor de frecuencia para mostrar la salida:

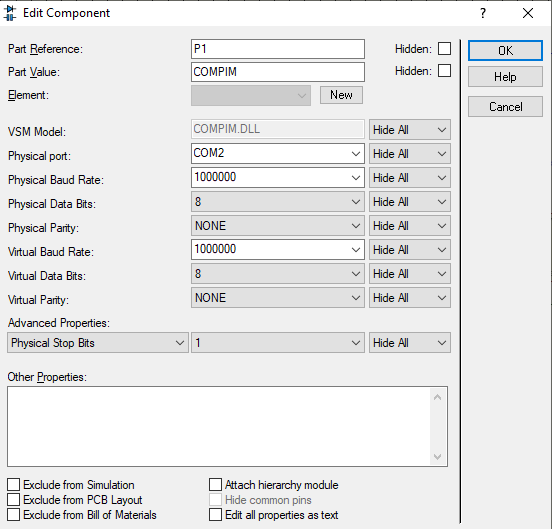


## **COMPIM**

Se utiliza COMPIM para activar la comunicación serial en proteus. Se logra creando un puerto serie virtual en Proteus que simule la interacción con Arduino. Si agregamos un modelo de puerto serie y conectamos el RXD y el TXD, los datos transmitidos deben devolverse inmediatamente al ordenador. Otra característica del modelo COMPIM es que proporciona conversión de velocidad de transmisión.

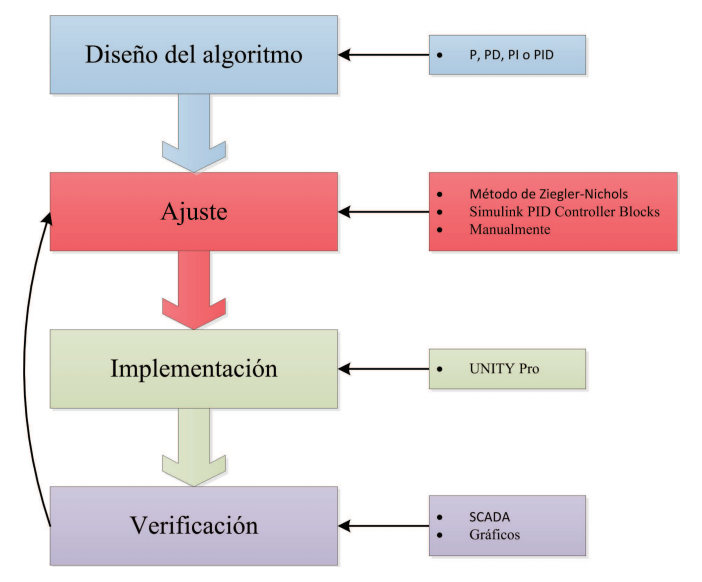


Configuración del COMPIM:



Controlador PID del sistema de control

El controlador PID será diseñado siguiendo el diagrama de flujo de la siguiente figura.



## Diseño del algoritmo

El diseño de un controlador PID comienza con la elección de la estructura adecuada: P, PI, PD o PI. Por ello se tiene que evaluar el beneficio de las ganancias proporcionales, derivativas e integrales. La ganancia Proporcional es necesaria porque reacciona directamente con el error obtenido entre el valor deseado y la variable controlada. Pero un controlador P con valores altos puede provocar un problema de sobreoscilación y para corregirlo se tiene que agregar una ganancia Derivativa (D) pues provee información acerca de cómo el error va cambiando con respecto al tiempo. Y, por último, para eliminar el error de estado estacionario, se agrega una ganancia Integral (I).

## Ajuste del algoritmo

Para el cálculo de estas constantes que van a definir el comportamiento del sistema de control se va a partir de una aproximación que proporciona el APP PID Tuner de Matlab. Se usará la aplicación PID Tuner porque sintoniza automáticamente las ganancias de un controlador PID para una planta SISO (Single-input single-output system. Se puede especificar el tipo de controlador, como PID, PI, PD o P.

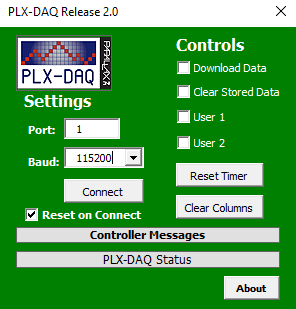
El sintonizador PID se utilizará con una planta representada por un modelo LTI numérico, con una función de transferencia (tf); sin embargo, no se cuenta con la función de transferencia del motor-encoder del proteus pues para obtenerla primero habría que determinar las constantes que utilizan las ecuaciones diferenciales que definen su comportamiento. Por ello, también se usar el software System Identification Toolbox para poder estimar la planta a partir de datos los datos simulados con proteus.

### Obtención de datos desde proteus

Para estimar la función de transferencia del motor-encoder de proteus se tiene que hallar las señales que estimulan o excitan al motor (señal PWM) y la salida del sistema (velocidad RPM). El cálculo de la velocidad RPM se tiene en el CÓDIGO DEL CÁLCULO RPM.

Los datos serán generados por el sensor durante la simulación en proteus para ser almacenados en un archivo de Excel que pueda ser leído por Matlab.

Los envíos de datos desde proteus a Excel se conseguirá con el software Paralaje Data Acquisition herramienta (PLX-DAQ) que es un complemento de Microsoft Excel que proporciona una forma fácil de recolección de datos de sensores en tiempo real en una hoja de cálculo.



### Lectura de datos en Matlab

Con los datos pwm y rpm almacenados en un archivo Excel se procede a leerlos en Matlab

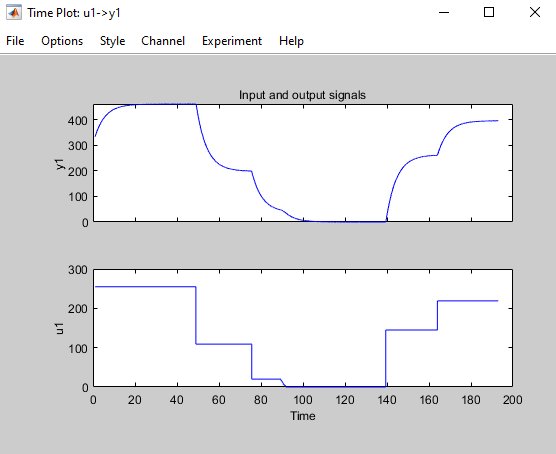
T = readtable('PLX-DAQ\_R2.xlsm','sheet','Simple Data');

pwm = table2array(T(:,3));

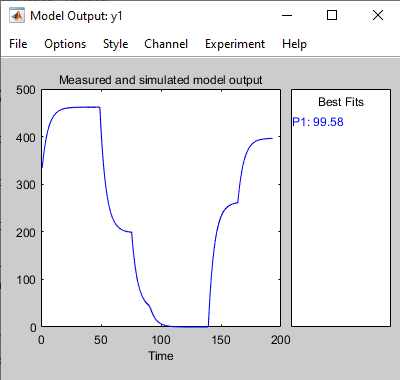
rpm = table2array(T(:,4));

### Función ident

Los datos medidos del dominio del tiempo son importados a Matlab. Ambas señales están medidas en el dominio tiempo. La señal y1 corresponde a la velocidad RPM y la señal u1, al pwm.

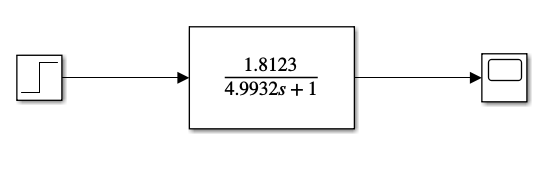


La función de transferencia ya puede ser estimada. Y se observa que la función estimada tiene un 99.58% de coincidencia con la señal original.

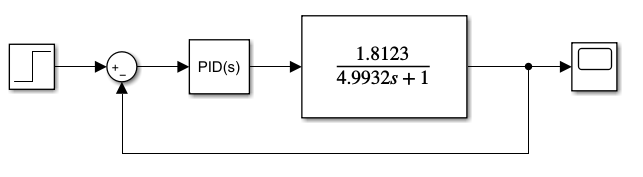


### Implementación en simulink

Diseño en simulink del sistema de lazo abierto.

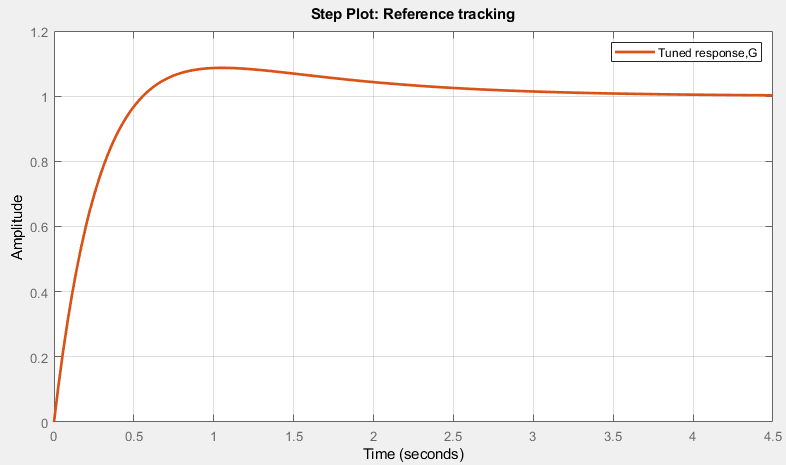


Diseño en simulink del sistema de lazo cerrado.

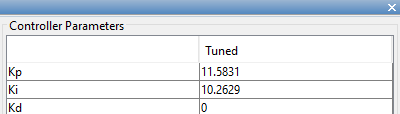


## Ajuste de las ganancias PID

Primero importamos la función de transferencia de la planta y luego ajustamos los tiempos de respuesta y el comportamiento transitorio.

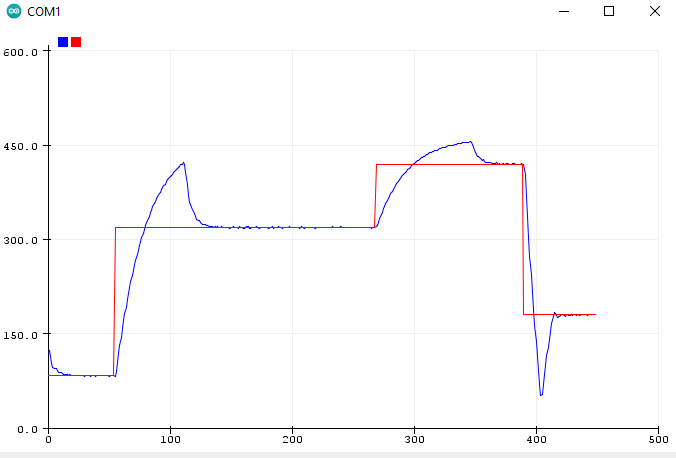


Las constantes aproximadas son:



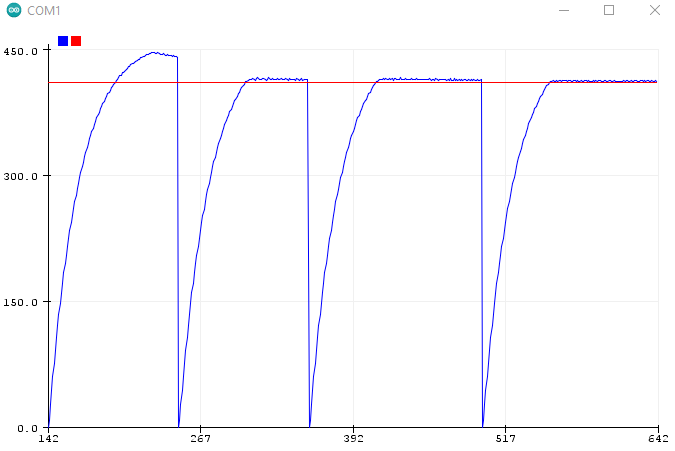
## Verificación

Resultados



Nótese que el valor de los rpm se estabiliza muy rápidamente y eso ocurre porque el valor del ki es alto; sin embargo, esta ganancia también induce a sobreimpulsos.

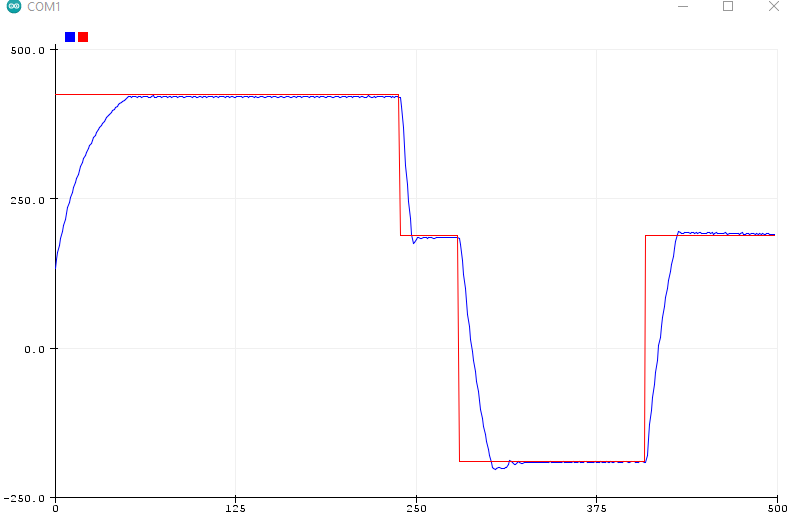
Por lo tanto, se va a reducir el valor ki. Se mostrará la evolución de la variable de control con los siguientes valores de ki: 3.701, 0.17008, 0.1685, 0.1585. Y las constantes kp y kd son, respectivamente, 7.8 y 1.3715. En la siguiente imagen se observa cómo afecta el valor de ki sobre el control.



### Valores finales del kp,ki y kd

* float aggkp = 10.8, aggkd = 0.5714, aggki = 0.831;
* float conskp = 10.8, conskd = 0.7714, conski = 0.971;

Estos resultados se obtienen con un PID con ganancias adaptativas



# **CONCLUSIONES**

* Hay un ruido inmanente en el cálculo de la velocidad RPM pues a pesar de que se mida en un intervalo de la misma longitud y para una misma entrada PWM las velocidades fluctúan alrededor de un valor. Durante las pruebas notamos que este ruido era mucho mayor cuando se disminuía la velocidad de comunicación serial o cuando se calculaba las RPM en intervalos de tiempo muy cortos. Por ello, la velocidad de comunicación serial es de 1000000 baudios y el tiempo de muestreo o cálculo, 20 ms (un tiempo lo suficientemente pequeño como para aproximar la derivada e integral del error sin perjudicar el cómputo de la velocidad RPM).
* La condición fluctuante de la velocidad de referencia, a través de un potenciómetro, repercutió en el control PID con el fenómeno derivative-kick pero se pudo mejorar el control considerando la derivada del error igual a la derivada del output.
* La señal de control solo opera sobre el motor cuando toma valores de 0 a 255 y si sale del intervalo se evidencian el fenómeno wind-up que consiste en retardos indeseados por estar fuera del rango de acción; por ello, la señal PID debe mantenerse entre los límites de operación. Para ello se limitó la salida del PID y la acción integral al rango 0-255.
* Las modificaciones para suprimir los efectos de los fenómenos wind-up y derivative-kick mejoraron la aproximación de la velocidad de salida a la de referencia (en el sentido de que se obtuvo una respuesta rápida y robusta) y los resultados finales fueron satisfactorios.

# PRUEBA DEL SENSOR ENCODER

## Código Arduino

#define SB 3

#define SA 2

void setup() {

// put your setup code here, to run once:

Serial.begin(115200);

pinMode(SA, INPUT);

pinMode(SB, INPUT);

}

void loop() {

// put your main code here, to run repeatedly:

int a = digitalRead(SA);

int b = digitalRead(SB);

Serial.print(a\*5);

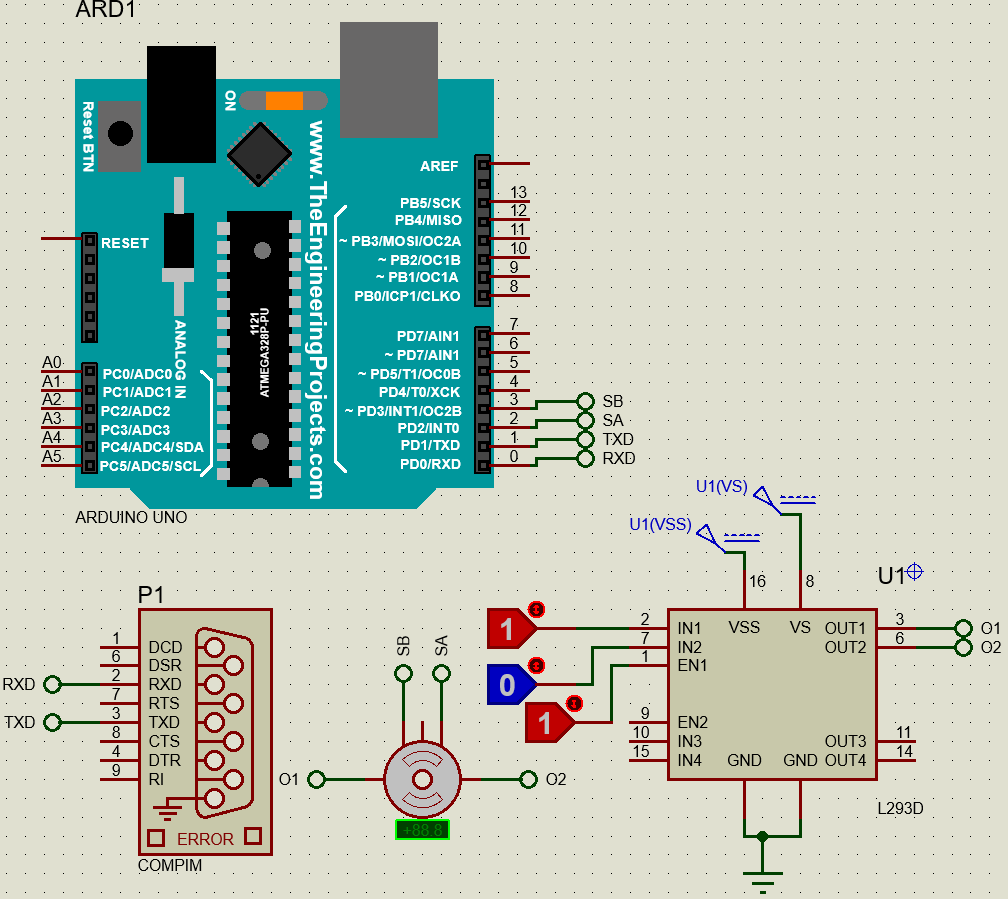
Serial.print(" ");

Serial.print(b\*5);

Serial.println();

}

## Diagrama esquemático en proteus



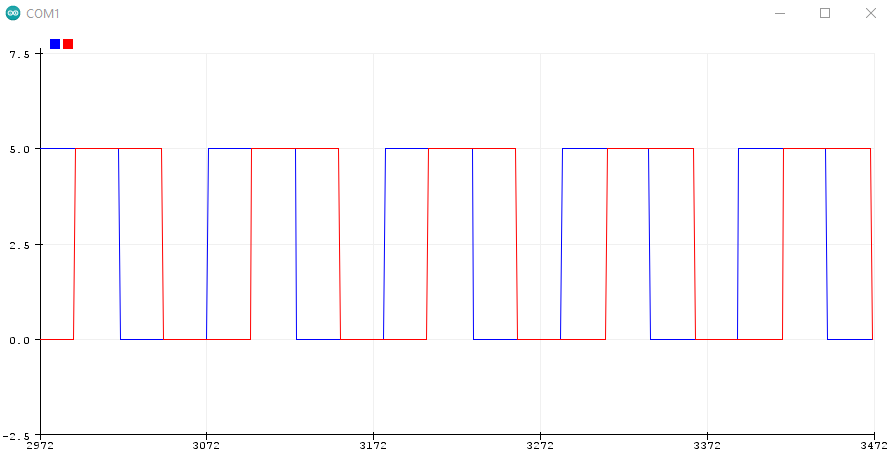
## Resultados:

|  |  |  |  |
| --- | --- | --- | --- |
| ENA | IN1 | IN2 | Sentido |
| 0 | - | - | - |
| - | 1 | 1 | - |
| - | 0 | 0 | - |
| 1 | 0 | 1 | Levógiro |
| 1 | 1 | 0 | Dextrógiro |

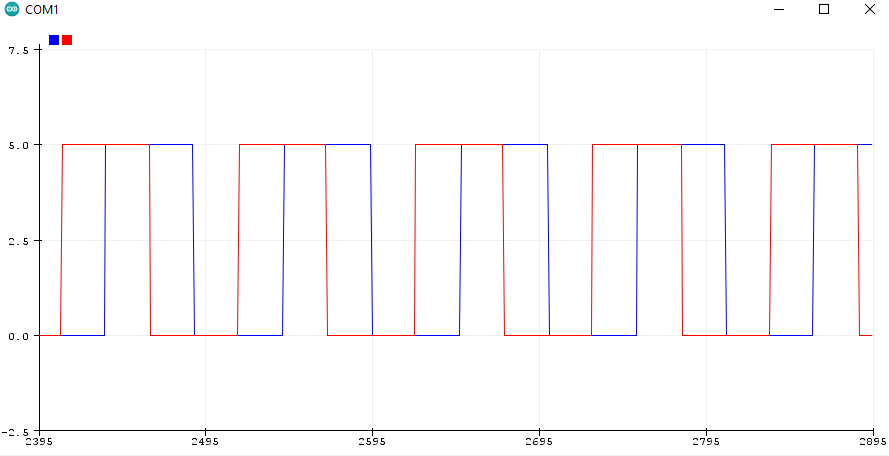
Gráfico de las señales A y B

* Señal A: Azul
* Señal B: Roja

Sentido levógiro: La señal A está adelantada a la señal B



Sentido dextrógiro: La señal B está adelantada a la señal A



# CÁLCULO DE RPM

## Código en Arduino

#include <util/atomic.h>

#include <LiquidCrystal.h>

LiquidCrystal lcd(13,12,11,10,9,8);

#define ENA 5

#define IN1 7

#define IN2 6

#define SA 2

#define SB 3

float resolution = 300.0;

long prevT = 0;

volatile int pulses = 0;

void setMotor(int dir, int pwmVal) {

analogWrite(ENA, pwmVal);

if(dir == 1){ // CW

digitalWrite(IN1, HIGH);

digitalWrite(IN2, LOW);

}

else if(dir == -1){ // CCW

digitalWrite(IN1, LOW);

digitalWrite(IN2, HIGH);

}

else{

digitalWrite(IN1, LOW);

digitalWrite(IN2, LOW);

}

}

void readEncoder() {

pulses++;

}

void setup() {

// put your setup code here, to run once:

Serial.begin(115200);

pinMode(IN1, OUTPUT);

pinMode(IN2, OUTPUT);

pinMode(SA, INPUT);

pinMode(SB, INPUT);

attachInterrupt(0, readEncoder, RISING);

prevT = micros();

}

void loop() {

// put your main code here, to run repeatedly:

lcd.setCursor(0,0);

lcd.print("RPM:");

float pwm = map(analogRead(0),0,1023,0,255);

setMotor(1, pwm);

float rpm;

float currT = micros();

float deltaT = currT-prevT;

if(deltaT>=2.0e5){

ATOMIC\_BLOCK(ATOMIC\_RESTORESTATE){

rpm = (float)pulses/deltaT/resolution\*60\*1.0e6;

pulses = 0;

}

prevT = micros();

attachInterrupt(0, readEncoder, RISING);

lcd.print(rpm);

lcd.print(" ");

lcd.setCursor(0,1);

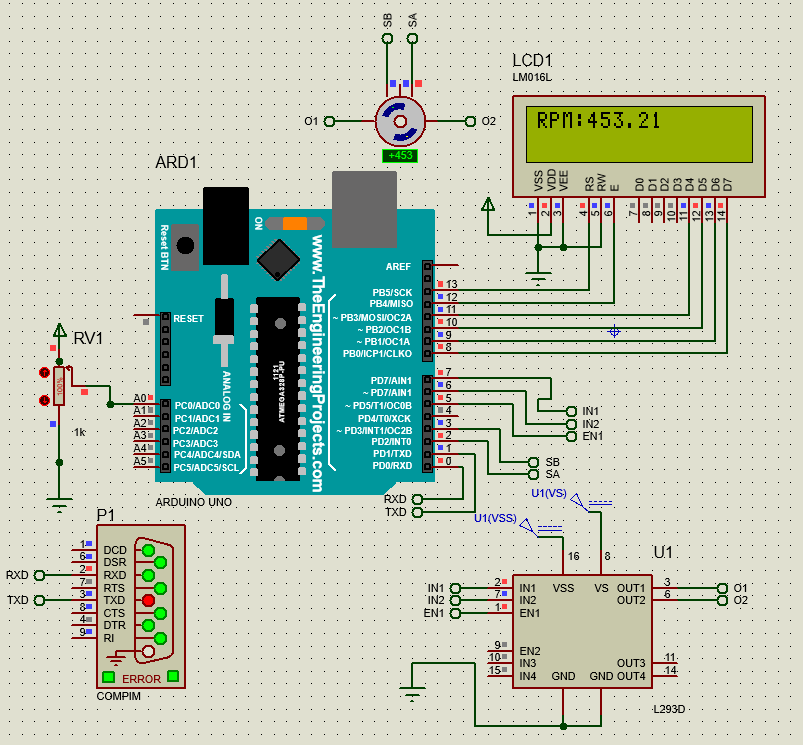
}

Serial.print(rpm, 3);

Serial.println();

}

## Diagrama esquemático en proteus



## Resultados:

La pantalla debajo del motor muestra el RPΜ (rondas por minuto). Y la pantalla lcd muestra el mismo valor de RPM. La velocidad de los pulsos que se obtiene en los pines de salida (codificador) depende de la propiedad de pulsos por revolución del motor.

Formula de transformación de pulsos por segundo a RPM:

# ADQUISICIÓN DE DATOS

## Código Arduino

#include <util/atomic.h>

#define IN1 13

#define IN2 12

#define EN1 5

#define SA 3

#define SB 2

float prevT;

float pwm;

float resolution = 300.0;

volatile int pulses = 0;

void readEncoder(){

pulses++;

}

void setMotor(int dir, int pwm){

analogWrite(EN1, pwm);

if(dir>0){

digitalWrite(IN1, HIGH);

digitalWrite(IN2, LOW);

}

else if(dir==0){

digitalWrite(IN1, LOW);

digitalWrite(IN2, HIGH);

}

else{

digitalWrite(IN1, LOW);

digitalWrite(IN2, LOW);

}

}

void setup() {

// put your setup code here, to run once:

Serial.begin(115200);

pinMode(IN1, OUTPUT);

pinMode(IN2, OUTPUT);

pinMode(EN1, OUTPUT);

pinMode(SA, INPUT);

pinMode(SB, INPUT);

attachInterrupt(1, readEncoder, RISING);

Serial.println("CLEARSHEET");

Serial.println("LABEL, Hora,Segundos,pwm, rpm");

Serial.println("RESETTIMER");

prevT = micros();

}

void loop() {

// put your main code here, to run repeatedly:

pwm = map(analogRead(0),0,1023,0,255);

setMotor(1,pwm);

float rpm;

float currT = micros();

float deltaT = currT-prevT;

if(deltaT>=2.0e5){

ATOMIC\_BLOCK(ATOMIC\_RESTORESTATE){

rpm = (float)pulses/deltaT/resolution\*60\*1.0e6;

pulses = 0;

}

prevT = micros();

Serial.print("DATA,TIME,TIMER,");

Serial.print(pwm);

Serial.print(",");

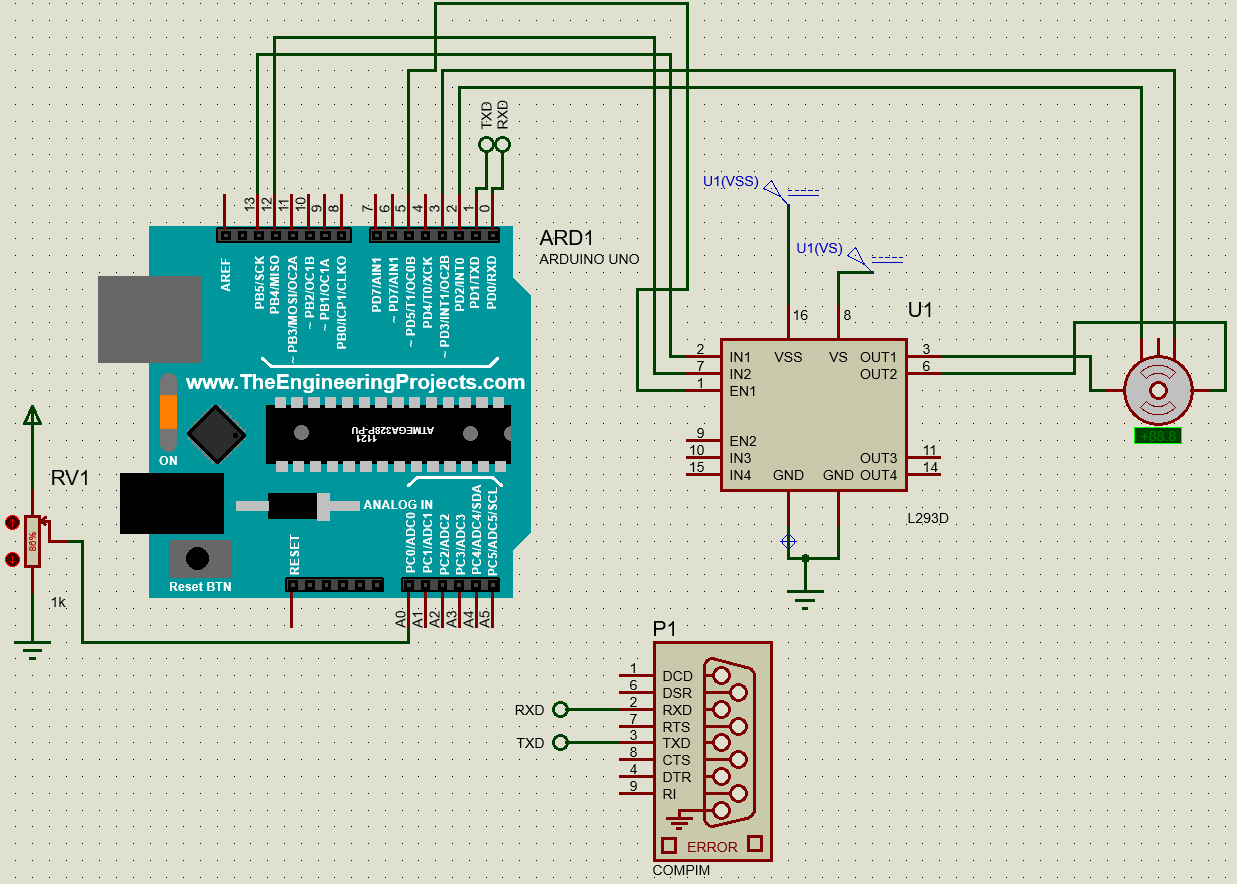
Serial.print(rpm);

Serial.println("");

}

}

## Diagrama esquemático en proteus



# CONTROL DE VELOCIDAD RPM DE UN MOTOR DC

## Código Arduino

#include <util/atomic.h>

#define ENA 5

#define IN1 13

#define IN2 12

#define SA 2

#define SB 3

#define button 4

float resolution = 300.0;

float lastInput = 0.0;

float lastTime;

float prevT;

float outMin = 0;

float outMax = 462;

volatile int pulses = 0;

volatile int theta = 0;

int pos = 0;

float eprev = 0.0;

float eintegral = 0.0;

float target;

float kp,kd,ki;

float aggkp = 12.8, aggkd = 0.5714, aggki = 0.731;

float conskp = 12.8, conskd = 0.5714, conski = 0.731;

void setTunings(float p, float i ,float d){

kp = p;

ki = i;

kd = d;

}

void setMotor(int dir, int pwmVal) {

analogWrite(ENA, pwmVal);

if(dir == 1){ // CW

digitalWrite(IN1, HIGH);

digitalWrite(IN2, LOW);

}

else if(dir == -1){ // CCW

digitalWrite(IN1, LOW);

digitalWrite(IN2, HIGH);

}

else{

digitalWrite(IN1, LOW);

digitalWrite(IN2, LOW);

}

}

void readEncoder() {

pulses++;

int b = digitalRead(SB);

if(b>0){ //CW

theta = 1;

}

else{ //CCW

theta = -1;

}

}

void setup() {

// put your setup code here, to run once:

Serial.begin(1000000);

pinMode(IN1, OUTPUT);

pinMode(IN2, OUTPUT);

pinMode(SA, INPUT);

pinMode(SB, INPUT);

pinMode(button, INPUT);

attachInterrupt(0, readEncoder, RISING);

target = analogRead(0)\*(2\*digitalRead(4)-1);

prevT = micros();

}

void loop() {

// put your main code here, to run repeatedly:

target = map(analogRead(0),0,1023,outMin,outMax)\*(2\*digitalRead(4)-1);

float rpm;

float currT = micros();

float deltaT = (currT-prevT);

int dir;

if(deltaT>=2.0e5){

ATOMIC\_BLOCK(ATOMIC\_RESTORESTATE){

rpm = (float)pulses/deltaT/resolution\*60\*1.0e6;

pulses = 0;

dir = theta;

}

prevT = micros();

float e = target - rpm\*dir;

float gap = fabs(e);

if (gap<10)

setTunings(conskp,conski,conskd);

else{

setTunings(aggkp,aggki,aggkd);

}

float dedt = -(rpm-lastInput)/(deltaT/1.0e6);

eintegral = eintegral + e\*(deltaT/1.0e6);

if (eintegral>255) eintegral = 255;

else if (eintegral < -255) eintegral = -255;

float u = kp\*e + kd\*dedt + ki\*eintegral;

float pwm = fabs(u);

if(pwm>255) pwm = 255;

lastInput = rpm;

setMotor(2\*(u>0)-1, pwm);

Serial.print(rpm\*dir);

Serial.print(" ");

Serial.print(target);

Serial.println();

}

}

## Diagrama esquemático en proteus

