Universidad Nacional Autónoma de México

Facultad de Ingeniería
Departamento de Control y Robótica
Instrumentación Virtual
2018-1

Control de velocidad con PID digital para un Motor de CD y conexión por TCP/IP

Torres Galicia Daniel
Zazueta Cruz Gibrán Alfonso
Grupo 2

Entrega Experimental: 7 de diciembre de 2017
Entrega Reporte: 10 de diciembre de 2017

1. Introducción

El método de control PID es el más utilizado en la industria debido a su relativa facilidad de implementación y a su fiabilidad para actuar ya que no se necesita un modelo matematico. Este controlador proporciona una salida u(t) en función de una señal de error e(t) y en base a los valores de la ganancia proporcional Kp, ganancia integral Ki y ganancia deivativa Kd. Su unico inconveniente resulta en la seleccion de los parametros necesarios para el controlador, existiendo metodos que te permiten calcular y aproximar esos parametros.

La comunicación y visualización en la computadora es algo necesario en la actualidad, ya que los sistemas de monitoreo son muy aplicados en la industria. Por lo cual un objetivo claro es tener una implementación de un sistema de monitoreo. Otro objetivo que se tiene es tener una comunicación con la computadora que se puede hacer de múltiples maneras, por lo cual se prevé la utilización de la comunicación TCP/IP ya que en la actualidad esta permeando en todas las areas de la industria.

2. Desarrollo

Se plantea la programación de un control PID digital para controlar la velocidad en RPM de un motor de corriente directa con reducción y un sistema de monitoreo con ajuste de señal de referencia por medio de comunicación TCP/IP con NI Labview.

2.1. Motor

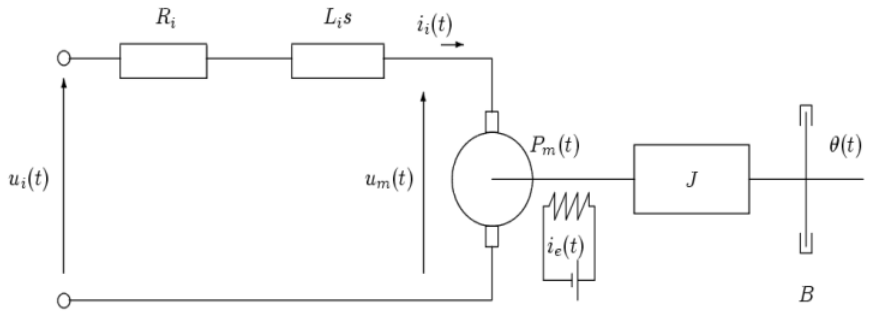

Figura 1: Esquema eléctrico-mécanico del motor

El motor a controlar elegido fue un motorredutor namiki de 12v de 120 rpm con un encoder integrado de 8 puntos por vuelta incremental, con una reducción de 1:80. El modelado del motor se obtiene a partir del modelo de la figura 1, el modelo se comienza con la parte electrica y despues con la parte mecanica.

$$u(t) = R * i(t) + L * i'(t) + k_E * \omega(t)$$

$$K_T * i(t) = B * \omega + J * \omega' +$$

La parte resistiva se puede obtener a partir de la medicion de las borneras ya que el motor es de imanes permanentes, por lo cual solo el alambre es del rotor. La inductancia se puede calcular con un ciruito básico RLC, junto con el generador de señales y un osciloscopio. Las demas constantes fueron obtenidas por medio de libros.

Para el proyecto se implemetÓ un control por PWM, por lo que se tenÍa que crear un diseño elÉctrico acorde a eso, ya que se utilizÓ un arduino para generar la señal de control y tomar la lectura del encoder incremental que posee el motor. En la figura 8 de los anexos se puede visualizar el diseño elÉctrico utilizado para el control.

2.2. Arduino

El control fue implementado con una tarjeta Arduino Mega 2560 que hacÍa solamente de control, mientras se comunicaba con el nodeMCU para que este al mismo tiempo se comunicara con el servidor en la computadora por medio de TCP/IP. El programa del control se logró por medio del programa que se encuentra en el anexo A.1.

El programa consiste de realizar un control PID por el metodo de euler para calcular la derivada y la integral, se establecieron las constantes a partir del punto de equilibrio de 80 [rpm]. Para calcular la velocidad del motor, se debio de hacer de forma implicita ya que el encoder solo calcula el numero de pasos que se dieron. POr lo tanto a partir de la diferencia entre los pasos se puede conocer la velocidad aproximada. Esta parte se logro a partir de interrupciones de cambio de estado en los pines 2 y 3.

En el loop principal del programa se utiliza una espera para enviar datos cada determinado momento, mientras ocurren las interrupciones. Para calcular el PID, anteriromrente mencionado, se utilizo una interrupcion cada 50 ms, ya que la velocidad maxima que se lograba no permitia un tiempo de ciclo menor.

2.3. Conexión TCP/IP con nodeMCU

Para lograr la comunicación TCP/IP se optó por implementar un çlienteütilizando la tarjeta de desarrollo libre nodeMCU v2 con el microprocesador ESP8266 integrado. La conexión entre el nodeMCU y el arduino Mega se realizó por comunicación serie a través del bloque UART de ambos dispositivos. En cuanto al sevidor, este fue configurado en Labview, utilizando los bloques de comunicación TCP/IP y una computadora personal con conexión Wifi. El diagrma de conexiones se muestra en la figura 9, en los anexos de este documento

2.4. Programación nodeMCU

Para cumplir con nuestro fin el ESP8266 debe conectarse, primero, con el arduino MEGA por comunicación serial, despues a la red de internet local (pro-

porcionando la información del SSID y la contraseña), y finalmente al servidor por medio de la ip del "host".

Una vez la comunicación se ha efectuado el programa se dedicará a recibir y enviar datos entre nuestro microcontrolador con el PID programadao y la cmputadora con Labview. Esto se hace a través de los comando "Serial.print" y "client.print", para escritura, al igual que "Serial.read" y "client.readStringUntil" para lectura.
El programa completo se muestra en el apartado de anexos.

3. Labview VI

El panel frontal y el diagrama de bloques del VI completo se presentan en las figuras 6 y 7 de la sección de Anexos de este documento.

Para facilitar el entendimiento y visualización del programa este se explicará por partes:

Se empieza por inicializar la conexión con el cliente deseado; para esto se utiliza el bloque Tcp Listen que espera hasta el "timeout" especificado (30 seg) para realizar una conexión con algún dispositivo sobre el puerto especificado (port: 6340). En caso de no encontrar ningún dispositivo el bloque entrega un cluster de error que hace al "Server loop" iterar en una ocasión; por otro lado, si el dispositivo logra conectarse, del bloque saldrá por "connection ID" un número referenciado a la comunicación TCP que se está realizando.

Una vez iniciada la conexión se utiliza el bloque "TCP Read" para leer la cadena de datos que esté mandando el ESP8266. En la figura 3 se puede observar como se progamó esta operación, se utilizó un ciclo while para ir formando la cadena de carácteres que era leída 1 bit ala vez. Estos carácteres se concatenaban utilizando un shift register hasta encontrar el carácter "!", que funcionaba como nuestro delimitador de "fin de cadena".
Se decidió optar por esta forma de lectura ya que no podíamos saber el tamaño de la cadena que iba a enviar nuestro microcontrolador y si teníamos un valor fijo de bytes de lectura los datos serían cortados. Una solución que s eprobó fue utilizar otro bloque Read para saber primero el número de bytes entrantes para después leer nuestor dato de interés, sin embargo, la programación de este algoritmo hacía que nuestra lectura de datos fuera muy lenta y provocaba errores de conexión.

Una vez obtenida la cadena entrante se procede a graficarla. Debido a que estos datos llegan del microcontrolador como un string con formato

$$''datoPV + datoCV!''$$

la cadena tenía que ser separada. Para esto se utilizó el bloque "Match pattern", que divide un string en relación a un delimitador especificado, en la figura 4 se puede observar esta implementación.

Primero se utilizá un Match pattern para quitar el caracter de terminación "!" y después para separar nuestros dos datos de interés con con la expresión

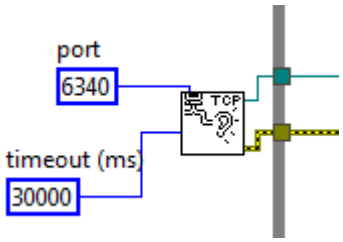

Figura 2: Inicio de conexión

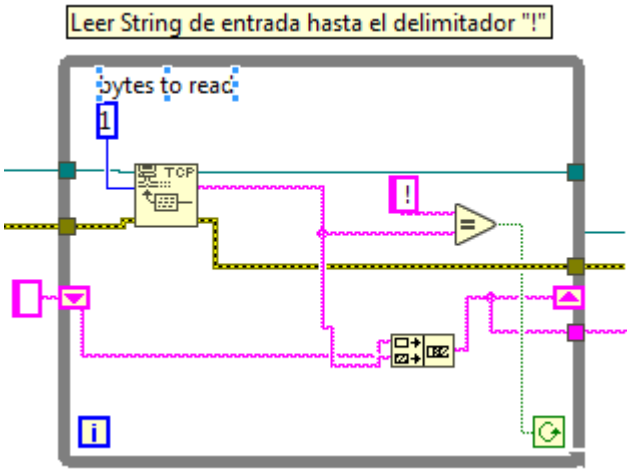

Figura 3: Lee la cadena recibida

regular "+". Finalmente los datos son convertidos a números y graficados con "Waveform chart".

Por último, en la figura 5 se muestra la implementación del Setpoint en nuestor programa. Para esto fue necesario utilizar otra función de la librearía TCP/IP: "TCP Write". Se utilizó una estructura case con un control para ordenar la detención del programa y de la comunicación, en caso de que el control enviara un True, el bloque TCP Write recibe un 200 que para nuestra progamación del ESP8266 significa detener la comunicación con el servidor.
Por otro lado, mientras el botón no sea presionado lo que el programa enviará por TCP será el dato del Setpoint, que puede ser especificado de dos maneras: por un control numérico o de perilla.

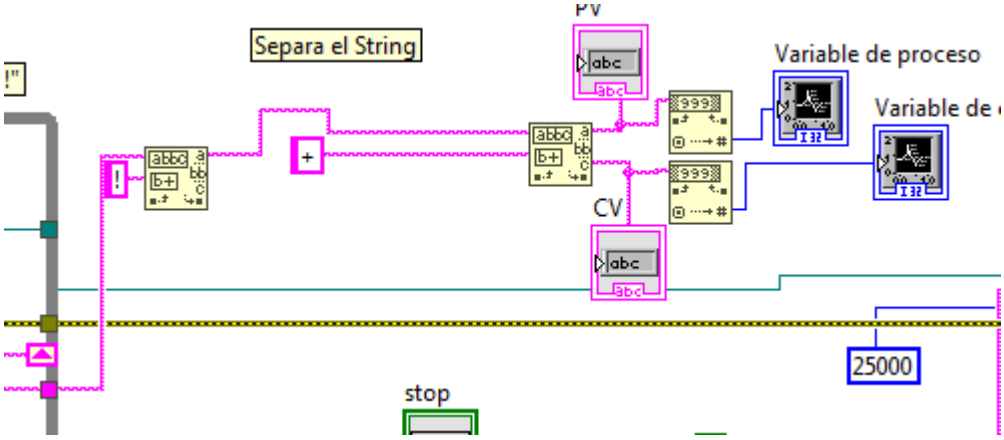

Figura 4: Separar datos

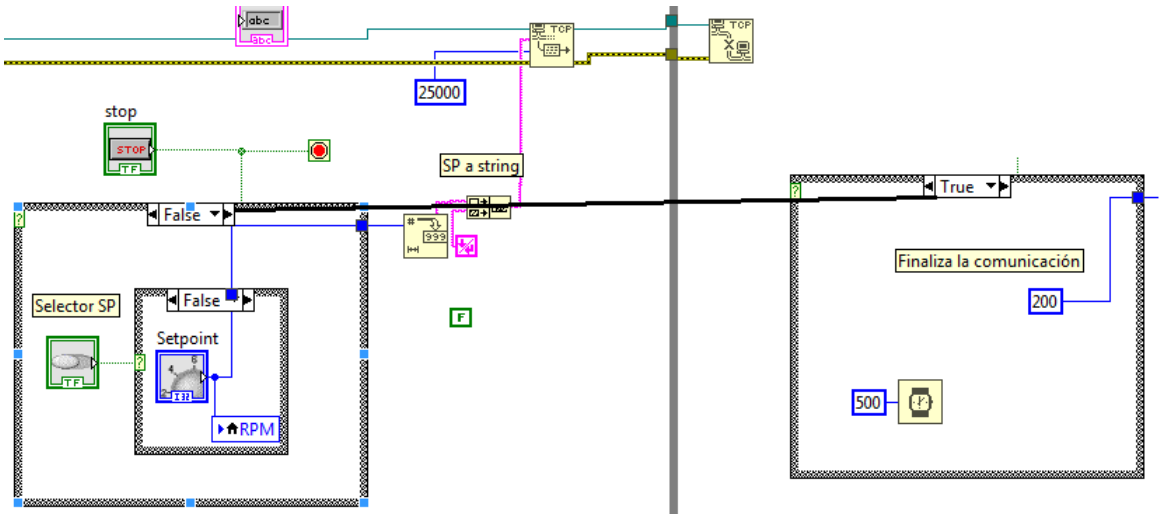

Figura 5: Escritura de datos

4. Conclusiones

4.1. Torres Galicia Daniel

El control PID es necesario en muchos caso, siendo utilizado en muchos aspectos de la industria y especialmente fácil en comparación con otros controles que son hechos a partir del modelo y no son modificables fácilente. Solamente se tiene que obtener los parametros de sintonización para los valores proporcional, integral y derivativo. En el programa de arduino lo que fue más fácil de programar, fue la parte proporcional ya que solo depende del error; en esta parte se comenzo a sintonizar el PID, siendo en este momento un control proporcional que cumplia la funcion de acercarse a la referencia.

La parte integral fue mas compleja de construir, se realizo a partir del metodo de Euler, quedando solamente una expresion. Mientras la parte derivativa, la mas compleja de contruir por que las señales presenta ruido y no se puede utilizar la definicion de la derivada; se debe de hacer por medio de euler para tener una buena aproximacion o de otro tipo de aproximacion númerica, al obtener la derivada (que seria la aceleracion), se puede aproximar por los metodos numericos.

Al terminar de crear el PID, se tenia el problema de calcular la velovidad, que al hacerla igual que la aceleracion, no se encontraba, mientras que al calcular por medio de la de fincion de derivada se podia encontrar facilment, pero necesitaba una constante que fue encontrada de manera experimental.

Al terminar de hacer el control de la velocidad, se procedio con la comunicacion hacia la computadora mientras de manera directa, ya que nuestro objetivo era poder monitorerarla velocidad angular en la computadora. Ya se había realizado alguna vez la comunicación directa entre LabView y arduino, por lo que se facilito en gran medidad la este paso. Y nuestro siguiente objetivo era la comunicacion por TCP/IP.

AL tener programa todo en el Arduino, entonces se procedio a modificar el codigo y que el NodeMCU fungiera como modem, ya que solo se iba a encargar de la comunicacion hacia la computadora y el arduino del control, como en algunos procesos se realiza, seprarar acciones del procesador principal y relegarlass a otro pocesador. Realizar la comunicacon fue sencillo solo que dependia demasiado del modem en la red que nos estuvieras comunicando ya que muchas veces se desconectaba. Se asemejaba mucho a otras comunicaciones que se realizaban anteriormente.

4.2. Zazueta Cruz Gibrán Alfonso

Las complicaciones para el proyecto comenzaron al momento de elegir el motor a controlar, se necesitaba de un motor pequeño que no demandará demasiada corriente, pero que a su vez fuera capaz de alcanzar velocidades lo suficientemente grandes para poder poner aprubea nusestro control. De igual manera, era necesario encontrar un encoder con pecio fuera accesible y que mantuviera una precisión que ayudara a nuestra señal de error. Al final el motor que conseguí-

mos cumplió adecuadamente todos estos requerimentos, además de que venía con el encoder ya montado.
Referente a la programación del PID, esta resultó mas sencilla de lo que esperaba, debido a que ya habíamos programado este tipo de controlador en otras plataformas, por lo que nuesto algoritmo dio resultados aceptables desde un principio. Por otro lado. donde si se tuvo alguna complicación fue en encontrar el parámetro a controlar, la velocidad. En un principio hacíamos control de posición y al querer implementarlo paa la velocidad (con una derivada) tuvimos que agregar un valor experimental.
Otro problema referente a esto fue la sintonización del PID que, de igual manera, tuvo que ser experimental, debido a la dificultad de ver la respuesta del modelo con el método de comunicación que teníamos
En cuanto a la comunicación TCP/IP esta fue la primera vez que tuve oportunidad tanto este protocolo de comunicación como la tarjeta nodeMCU. una vez entendí su funcionamiento y probé ejemplos básicos su implementación para nuestras necesidades no fue muy complicada; el único problema surgió al momento de leer los datos, ya que, no lograbamos hacer que se leyeran las cadenas completas. La solución a esto es la que se detalló en el desarrollo del reporte.
Aún cuando la comunicación fue sencilla de implementar, en nuestras pruebas no era extraño que hubiera perdida de información en algunos momentos. Esto se lo atribuyó a al conexión con el router, lo que me hace dudar en que este dispositivo sea adecuado para este tipo de aplicaciones de control.

A. Programas

A.1. Programa para control (Arduino MEGA 2560)

```
#include <TimerOne.h>

const int channelPinA = 2;
const int channelPinB = 3;
const int channelPinC = 8;
const int channelPinD = 9;

const long maxSteps = 100000000;
const long minSteps = -100000000;
volatile long ISRCounter = 0;
long paspos1 = 0;
float pasrec = 0;
//float paspos = 0;
//float provel = 0;
float refvel = 0;
float errvel = 0;
int control = 0;
float cK1=1.5;    //Constante proporcional0.9
float cI1=0.08  ;     //Constante integral0.05
float pasint=0;     //Valor pasada para la integral
float cD1=0.002;       //Constante derivativa0.001
float pasder=0;     //Valor pasado de la derivada
float Lam1=20;      //Frecuencia de corte5
float k1=0;         //Ayuda para el metodo de euler

float dt1=0.020;    //Constante de tiempo
int env1;
int env2;

//bool IsCW = true;

void setup()
{
  //pinMode(channelPinA, INPUT_PULLUP);
   Serial.begin(57600);        //Maneja la velocidad de la transmision serial
   pinMode(channelPinA,INPUT);  //Activa el bit del canal 1
   pinMode(channelPinB,INPUT);  //Activa el bit del canal 2
   pinMode(channelPinC,OUTPUT);  //Activa el bit del canal 1
   pinMode(channelPinD,OUTPUT);  //Activa el bit del canal 2
  attachInterrupt(digitalPinToInterrupt(channelPinA), doEncodeA, CHANGE);
//Activa la interrupcion del channelPinA
  attachInterrupt(digitalPinToInterrupt(channelPinB), doEncodeB, CHANGE);
//Activa la interrupcion del channelPinB
   Timer1.initialize(20000);    //20 ms de muestreo
   Timer1.attachInterrupt(PID); //Maneja la señal de control
}

void loop()
{
```

```
   env1 = pasrec;//(provel);
   env2 = (control*100)/255;
  if(env1<100)Serial.print(0);
  if(env1<10)Serial.print(0);
  Serial.print(env1);
  Serial.write('+');
  if(env2<100)Serial.print(0);
  if(env2<10)Serial.print(0);
  Serial.print(env2);
  Serial.println("!");
    if(Serial.available())
     {
      refvel = Serial.parseInt();
      refvel = refvel;
      //control = Serial.parseInt();
     }
    if(refvel>180)
     {
      if(refvel==200) refvel=0;
      refvel=180;
     }
    if(refvel<-180)
     {
      refvel=-180;
     }
  delay(200);
}

void doEncodeA()
{
     if ( digitalRead(channelPinA) == digitalRead(channelPinB))
      {
        if (ISRCounter + 1 <= maxSteps) ISRCounter++;
        else ISRCounter=0;
      }
      else
      {
        if (ISRCounter - 1 > minSteps) ISRCounter--;
        else ISRCounter=0;
      }
}

void doEncodeB()
{
     if ( digitalRead(channelPinA) != digitalRead(channelPinB))
      {
        if (ISRCounter + 1 <= maxSteps) ISRCounter++;
        else ISRCounter=0;
      }
```

```
      else
      {
        if (ISRCounter - 1 > minSteps) ISRCounter--;
        else ISRCounter=0;
      }
}

void PID()
{
//Calculo de la velocidad
  pasrec = 4*(ISRCounter - paspos1);
  paspos1 = ISRCounter;
//  provel = -Lam1*paspos+Lam1*pasrec;//dt1;
//  paspos = paspos+dt1*provel;

  errvel = refvel-pasrec;

//Calculo de la integral
  pasint = pasint + dt1*errvel;

//calculo de la derivada
k1 = -Lam1*pasder+Lam1*errvel;
pasder = pasder+dt1*k1;

//Calculo del PID
  control = cK1*errvel+cI1*pasint+cD1*k1;
  if(control>=255)
   {
    control=255;
   analogWrite(channelPinC,control);
   analogWrite(channelPinD,0);
   }
  else if(control>0)
   {
   analogWrite(channelPinC,control);
   analogWrite(channelPinD,0);
   }
  else if(control>-255)
   {
    analogWrite(channelPinC,0);
   analogWrite(channelPinD,-1*control);
   }
  else
   {
    control=-255;
   analogWrite(channelPinC,0);
   analogWrite(channelPinD,-1*control);
   }
}
```

A.2. Programa nodeMCU

```
1
2int x=0;
3int setp=1;
4char linea[8];
5#include <ESP8266WiFi.h>      //cabecera para programar el
      ESP8266
6#include <ESP8266WiFiMulti.h>
7
8ESP8266WiFiMulti WiFiMulti;
9
10void setup() {
11    Serial.begin(57600);       //Inicia comunicacion serial
          con el Arduino Mega
12    Serial.setTimeout(3000);
13    delay(10);
14
15    // Conexi n a Red wifi
16    WiFi.mode(WIFI_STA);
17    WiFiMulti.addAP("SSID", "Contrase a"); //Se establece
          la conexi n con la red especificada
18
19    Serial.println();
20    Serial.println();
21    Serial.print("Estableciendo conexi n WIFI ");
22
23    while(WiFiMulti.run() != WL_CONNECTED) {
24        Serial.print(".");
25        delay(500);
26    }
27
28    Serial.println("");
29    Serial.println("WiFi conectado");
30    Serial.println("IP address: ");
31    Serial.println(WiFi.localIP());
32
33    delay(500);
34}
35
36
37void loop() {
38    const uint16_t port = 6340;          //El puerto con el
          que se comunicar  nuestro servidor
39    const char * host = "192.168.1.129"; // ip o dns del
          servidor, en este caso la direcci n IP entregado
          por el router a nuestra computadora
40
41
42    Serial.print("Conect ndose a: ");
43    Serial.println(host);
44
45    // Use WiFiClient class to create TCP connections
46    WiFiClient client;
47
48    if (!client.connect(host, port)) {
49        Serial.println("Fall  ...");
```

```
        Serial.println("Espere 5 seg...");
        x=0;
        delay(5000);
        return;
    }

 // Despu s conectarse inicia un ciclo while que solo
      termina al recibir un dato igualk a 200
while(setp !=200){

  Serial.readBytes(linea, 8);   //Lee del UART

   client.print(linea);       //Envia la cadena al servidor
       (Labview)
   String line = client.readStringUntil('\r'); //Lee ddel
       lservidor
   setp = line.toInt();                        //convierte
      cadena a entero
  // Serial.print("SP= ");
   Serial.println(setp);

   delay(500);   //Delay sincronizado con servidor
   }

   Serial.println("Conexi n terminada");
   client.stop();
   setp=0;
  return;

}
```

A.3. Programa Labview

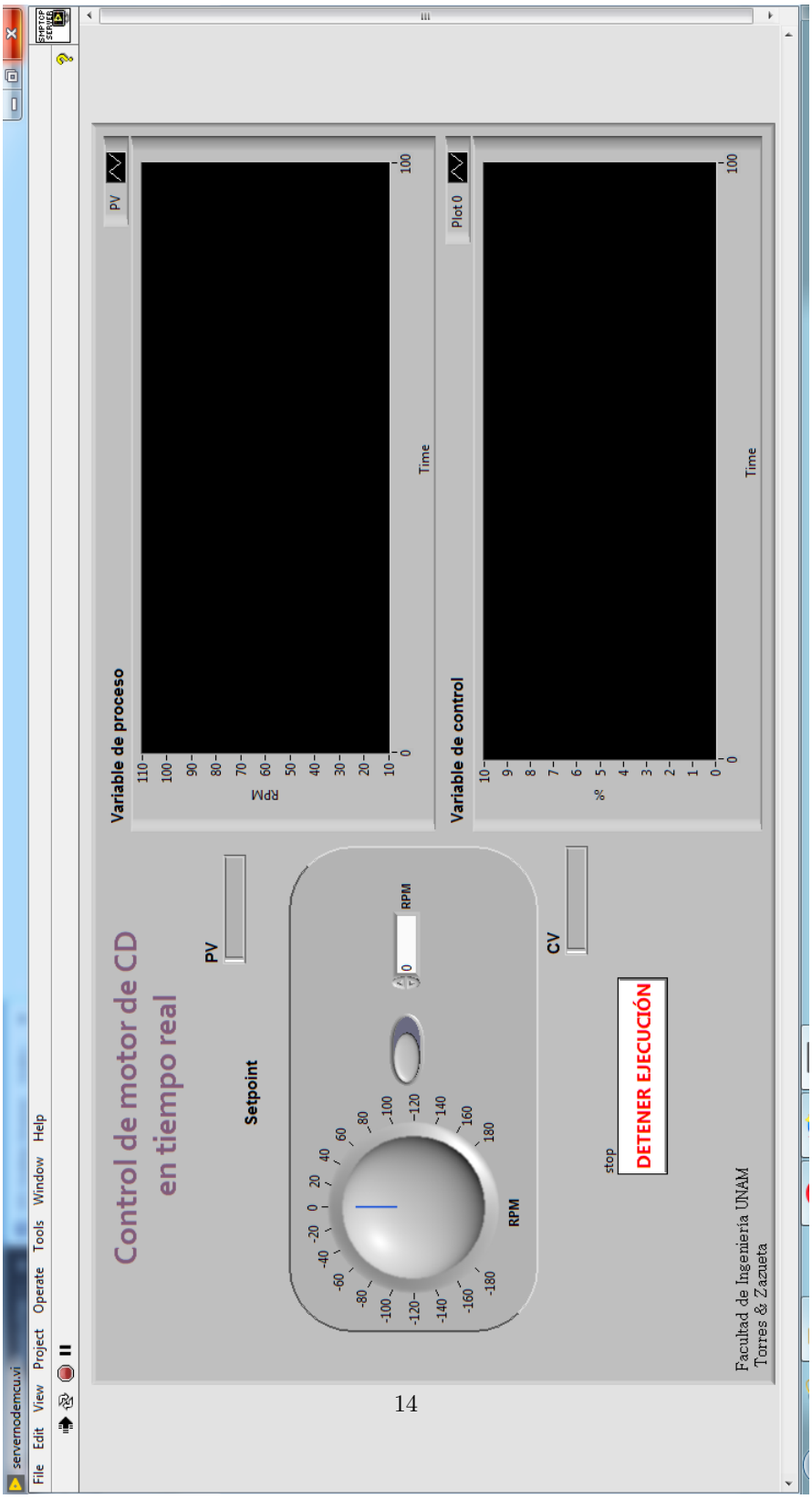

Figura 6: Panel frontal VI Labview

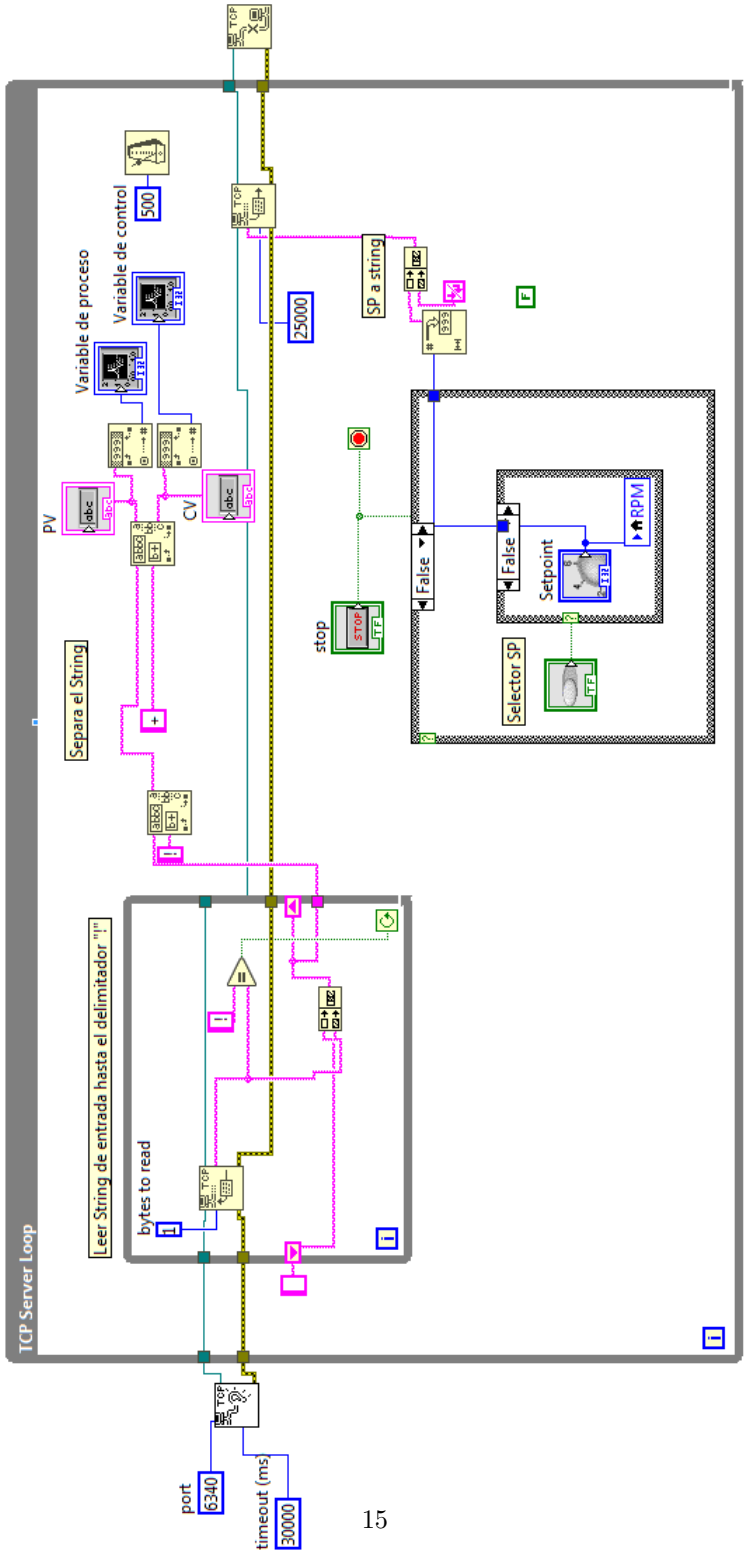

Figura 7: Diagrama de bloques Labview

B. Diagramas de conexiones

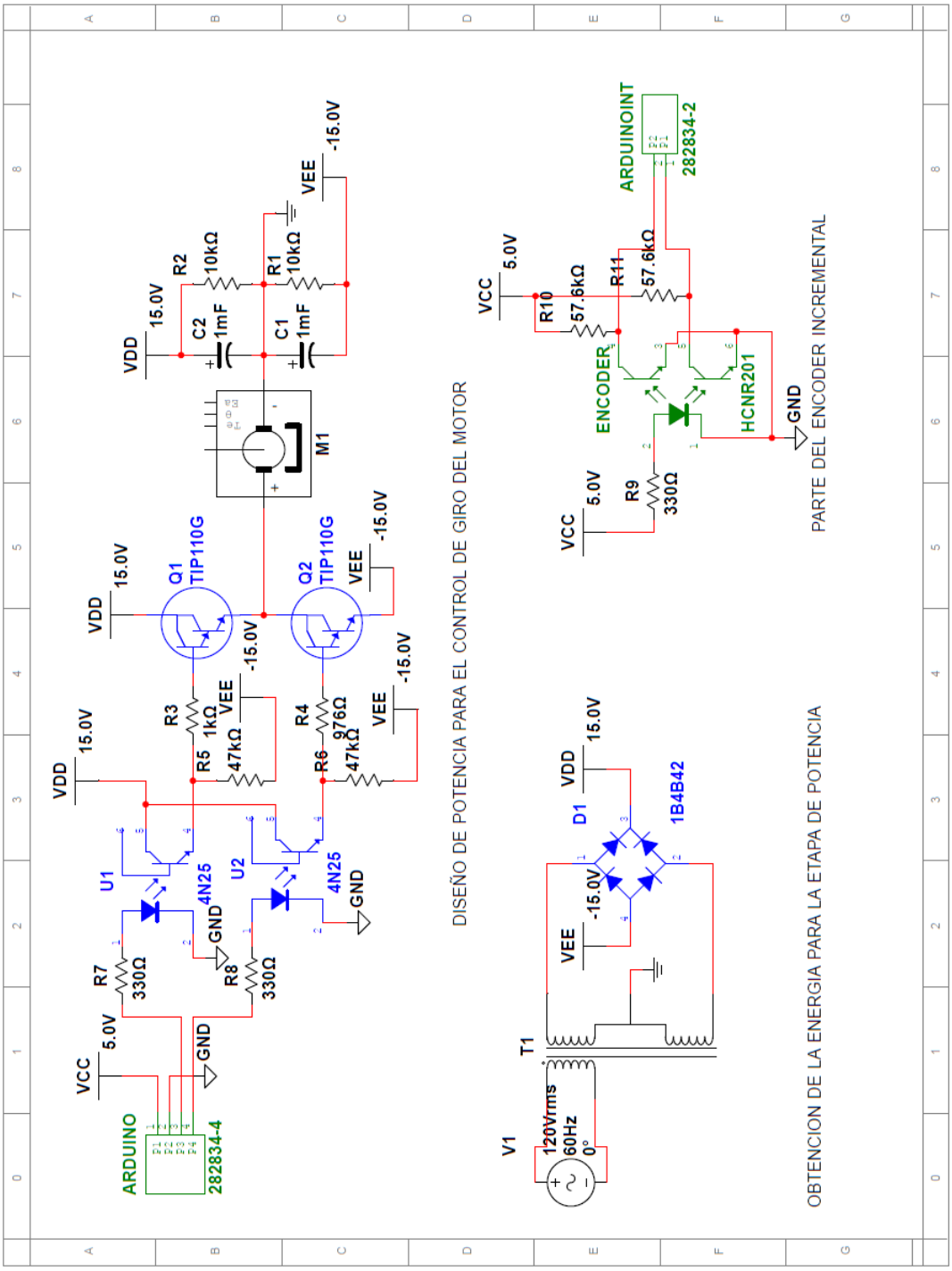

Figura 8: Diseño electrónico

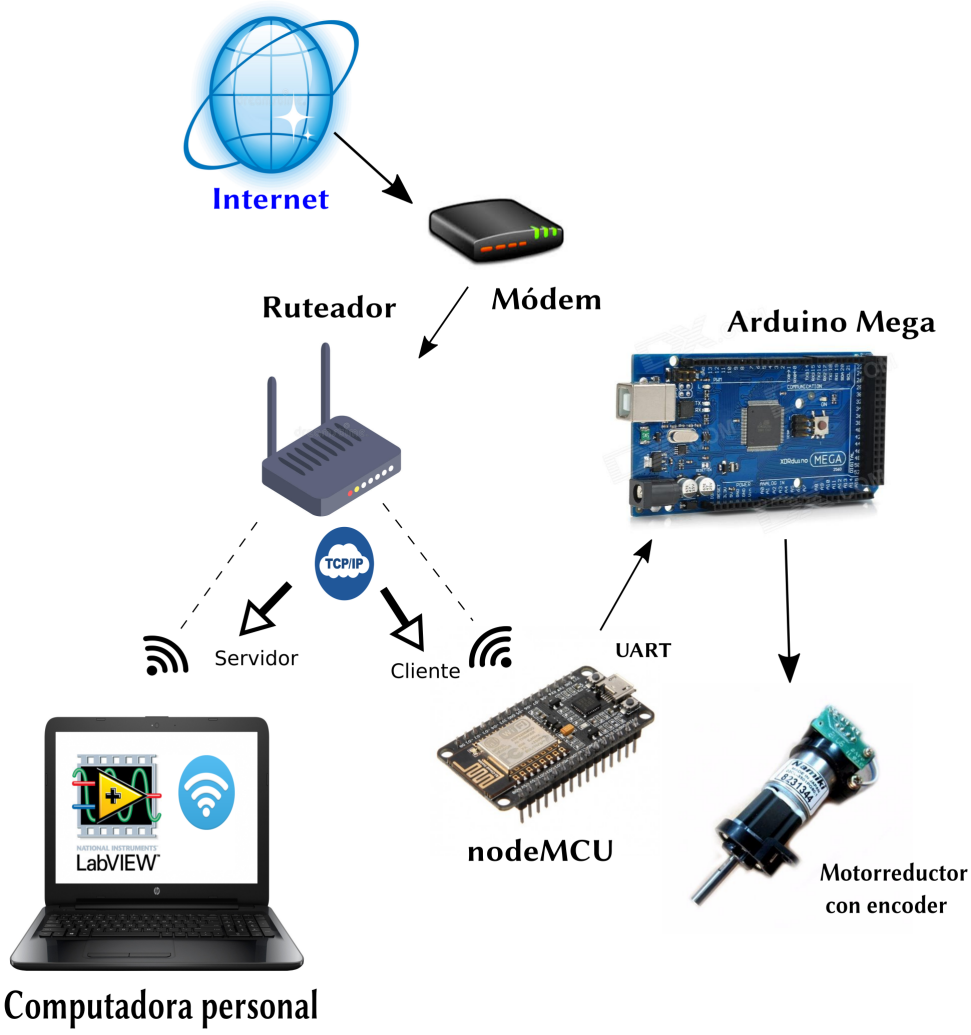

Figura 9: Esquema de conexiones