UNIVERSIDAD AUTONOMA METROPOLITANA
UNIDAD AZCAPOTZALCO

LICENCIATURA EN INGENIERÍA ELECTRÓNICA

Modalidad: **Proyecto Tecnológico**

Prototipo para el control de posición de un vehículo sobre riel como apoyo a la docencia e investigación.

Alumno:
Galicia Irianda Jorge Eduardo

Matrícula:
2132004036

Asesores:
Dr. Jesús Ulises Liceaga Castro
Dra. Irma Irasema Siller Alcalá

Trimestre: 19-I

Fecha: 30-Agosto-2019

Declaratoria.

Yo, Irma Irasema Siller Alcalá, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Yo, Jesús Ulises Liceaga Castro, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Yo, Jorge Eduardo Galicia Irianda, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

RESUMEN

En este proyecto se realiza el prototipo para el control de posición de un vehículo sobre un riel, de tal modo que dicho prototipo se utilice para futuras aplicaciones de investigación, así como apoyo para la docencia.

Se caracterizó un sensor infrarrojo Sharp para la medición de las distancias del vehículo sobre el riel, pero al no obtener distancias tan precisas se cambió dicho sensor por un sensor ultrasónico.

Para la realización de todo el proyecto, se usó el microcontrolador Arduino Uno el cual contiene diversas entradas y salidas digitales, así como también analógicas, en donde se usó una entrada analógica para caracterizar el sensor infrarrojo mientras que para el sensor ultrasónico se utilizaron dos salidas digitales.

Para desarrollar la etapa de potencia se usó un motor de DC con motorreductor, y se diseñó un puente H basado en transistores 2N2222, diodos 1N4001 y resistencias de 1kΩ, aunque dicho diseño se cambió por el circuito integrado L293D ya que el voltaje que recibía el motor de DC no era suficiente para poder moverse hacia ningún sentido.

Se diseñó un control P en programación Arduino Uno para el control de posición y velocidad de un vehículo sobre un riel.

Se realizó la instalación de los paquetes de Arduino en Matlab 2018b con el propósito de poder utilizar la tarjeta Arduino como una tarjeta de adquisición de datos, además de configurar parámetros en dicho software que permitieran el uso correcto de la placa Arduino.

Se diseño un control PI en Matlab 2018b (Simulink) utilizando como tarjeta de adquisición de datos la placa Arduino Uno, el cual contendrá todos los bloques necesarios para poder llevar acabo el control PI del vehículo sobre el riel, en este caso se utilizaron bloques de comparación que permitieran el giro tanto a la derecha como izquierda del motor de DC, así como la diferencia entre una referencia y la posición inicial del móvil para encontrar el valor de error, además de los bloques necesarios para llevar a cabo el control PI, como fue el caso de la ganancia de KP como de KI que fueron bloques de control variables, es decir, que se podrá colocar cualquier valor en dichas ganancias.

Finalmente se obtuvieron gráficas en tiempo real de la señal de referencia, posición, error y de PWM en las cuales se pudieron observar los distintos cambios debidos a los valores de las ganancias KP y KI, así como de su referencia y de la posición inicial del vehículo sobre el riel.

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Hoy en día en el laboratorio de la UEA de Control Digital de la Universidad Autónoma Metropolitana unidad Azcapotzalco, en el cual se realizan prácticas que implican el uso de tarjetas de adquisición de datos para ser usadas en la implementación de controladores PID, cuenta con pocos dispositivos o prototipos a los cuales se les puedan diseñar e implementar controladores, en particular tipo PID´s, por lo que en este proyecto se diseñará un prototipo más, que ayudará al estudiante a acortar la brecha que existe entre la teoría y la práctica.

Por definición el controlador Proporcional-Integral-Derivativo (PID) está dado por la ecuación diferencial:

$$U(\mathrm{t}) = Kp[\, e(t) + \frac{1}{Ti}\int e(t)dt + Td\ \frac{d}{dt}e(t)];\ Kp, Ti\ y\ Td\ \in R\ y\ constantes$$

$$\text{Donde } e(t) = r(t) - y(t)$$

Con: y (t) salida del proceso.

r (t) señal de referencia.

u (t) variable de control.

En función de transferencia el PID está dado por:

$$\frac{U(s)}{E(s)} = Kp[1 + \frac{1}{Tis} + Tds]$$

El PID tienes muchísimas aplicaciones en el control de procesos. En particular, el caso de sistemas de bajo grado el PID resulta ser el controlador ideal. Por otro lado, su diseño es un problema semiabierto que ha generado un gran número de procedimientos para su diseño. Para el caso que se pretende atacar se hará uso de los métodos de Ziegler/Nichols y diseño de compensadores.

Por otro lado, los controladores PID pueden implementarse a través de diferentes dispositivos tales como:

Microcontrolador, FPGA, PLC, DSP, etc. Pero para fines didácticos se usará una PC junto con tarjetas de adquisición de datos, y el paquete computacional MATLAB, con la intención de observar las gráficas obtenidas en tiempo real y posteriormente compararlas con las obtenidas por simulación, logrando con esto comparar la teoría con la práctica.

Algunas de las aplicaciones más comunes de los controladores PID son:

Lazos de nivel (Tanques de líquidos con agua entre algunos otros más), Lazos de presión (Tubos, recipientes, etc.), Lazos de temperatura (Calentadores, Refrigeradores, etc.) entre algunas aplicaciones más.

En conclusión, no se busca mejorar ningún diseño previo ni proponer uno nuevo, sino construir un prototipo particular que pueda ser utilizado en las UEA de Control Digital y Laboratorio de Control, y que a la vez pueda ser utilizado para fines de investigación en Control.

ANTECEDENTES

Algunos antecedentes que demuestran no sólo la utilidad de disponer de prototipos para el estudio y compresión del controlador PID, sino que es factible que sean diseñados y construidos por las propias universidades se muestran a continuación:

Instituto Politécnico Nacional-*ESIME*

Autor: Fernando Ávila Herrera.

Jorge Isaac Carbajal Bernal.

Josué Mares Olivares.

Tesis: "Control de velocidad y dirección de un robot de carreras autónomo" [1].

Año: 2011.

En esta tesis se hace mención del desarrollo de la velocidad y dirección para un robot de carreras autónomo, utilizando un control PID. El prototipo tenía como principal objetivo poder ser capaz de seguir una trayectoria, así mismo en el presente proyecto se pretende seguir una única dirección para la posición del prototipo. Se utilizaron tres subsistemas (sensores, control y actuadores) y un móvil con cuatro ruedas. Cabe mencionar que el sistema de control fue en tiempo real y que también para el control de dirección del robot se ejecutó el algoritmo de control PID para que de esa manera se determinara el error en la posición y por ende se modificará la señal PWM que era enviada al servomotor, el cual modificaba la posición de las llantas delanteras. El control PID en este proyecto busca corregir la velocidad del robot y a diferencia del proyecto propuesto que será para posición, ambos ocupan de sensores que les permiten el control de distancia y velocidad.

Universidad Nacional Autónoma de México

Autor: Isaac Ramírez Enríquez

Tesis: "Control PID de temperatura con PLC Siemens S7-300 y Allen Bradley SLC 500" [2].

Año: 2017.

*En esta tesis implementaron un control PID para llevar a cabo el control automático de un sistema de temperatura, al igual que en el proyecto propuesto se llevará a cabo el diseño de un control PID, pero con la diferencia de que será para posición y no para temperatura. La principal importancia de la tesis fue el comparar dos PLC´s (*SIEMENS S7-300 Y ALLEN BRADLEY SLC 500) *para que de esa forma pudieran demostrar cuál de los dos PLC tenía un mayor desempeño al utilizar un controlador PID de temperatura.*

Instituto Politécnico Nacional-*ESIME*

Autor: Juan Pablo López Franco

Jaime Eduardo Morales Martínez

Tesis: "Implementación de un sistema de domótica para el control de temperatura e iluminación de un hogar" [3].

Año: 2017.

En este proyecto se diseñó y construyó un sistema automatizado para el control de condiciones de temperatura e iluminación de un hogar utilizando un control PID, ya que como mencionan se tiene un mayor ahorro de energía eléctrica con el control PID, además de permitir seleccionar la cantidad de luz que se requiera. Para el presente proyecto se busca el control de posición, más no de temperatura, pero tomando en cuenta que ambos proyectos están con base a un control PID siendo ambos de gran importancia, uno para ser usado en el hogar y por nuestro lado para el apoyo a la docencia e investigación.

Universidad Industrial de Santander

Autor: Claudia Yaneth Peñaranda

Wilson Reinaldo Silva

Eulices Gómez Arias

Tesis: "Instrumentación y control de nivel para un sistema de tanques acoplados en el laboratorio de control e instrumentación de la E3T/UIS" [4].

Año: 2014.

Para este proyecto se utilizó un controlador PID empleando el primer método de Ziegler & Nichols para el llenado de tanques, aunque no optaron por el desarrollo y diseño del controlador PID ya que utilizaron un controlador UDC1200, es por eso que en este proyecto se busca el diseño del controlador PID ya que de esa manera se tiene una mayor visión del control digital además de ser una versión original, en un futuro se pueden hacer las modificaciones necesarias para un mayor desempeño o en su defecto se puede ocupar para otro de tipo de controles a diferencia de tener un controlador ya elaborado del cual no se puede modificar para otra aplicaciones.

JUSTIFICACIÓN

Un problema que se presenta en el laboratorio de control de la Universidad Autónoma Metropolitana unidad Azcapotzalco es el hecho de que no cuente con prototipos para llevar a cabo prácticas con controladores PID en tiempo real. Al momento de ingresar al ámbito laboral existe un gran problema para el ingeniero recién egresado, por eso es necesario añadir más prototipos para el control de variables que comúnmente se presentan en la industria tales como: presión, nivel, temperatura, velocidad, posición, etc.

En este proyecto se propone diseñar un prototipo con el cual los alumnos puedan controlar y al mismo tiempo puedan ver físicamente como se puede analizar y diseñar el control del sistema; de esta forma podrán adquirir experiencia práctica.

Es de vital importancia que se trabaje con nuevos prototipos de control dentro de la universidad ya que no somos la única institución que incluye en su currícula laboratorios de control, como es el caso de la UNAM y el IPN, que cuentan con prototipos comerciales, los cuales consideramos costosos tanto en su adquisición como en su mantenimiento. Es por esta razón que se propone el diseño y construcción de un prototipo que sea económico y de fácil mantenimiento. Además, este prototipo puede ser usado en las UEA de Control Digital y Laboratorio de Control.

OBJETIVOS

General

Diseñar e implementar un prototipo para la docencia e investigación que permita el control de posición y velocidad de un carro guiado por riel con base a un controlador PID.

Particulares

- Seleccionar el tipo de sensor infrarrojo a trabajar para la distancia que abarca el riel.
- Diseñar y programar un controlador PID mediante el software MATLAB (Para la UEA de Control Digital). De esta forma, la interfaz de usuario será el paquete de simulación MATLAB/Simulink.
- Diseñar y construir una etapa de potencia para el funcionamiento correcto de un motor de D.C.
- Realizar las pruebas experimentales para asegurar el funcionamiento total del sistema de control.
- Mostrar las gráficas de las respuestas obtenidas en tiempo real a través de la tarjeta de adquisición de datos, quien será la interfaz de comunicación.

CAPÍTULO 1. MARCO TEÓRICO

Este capítulo tendrá un enfoque en todo el marco teórico que se vendrá implementando a través del proyecto y así tener bases sólidas de los conceptos.

CONTROL DIGITAL

La siguiente introducción está basada en [5].
En la figura 1.1 se puede observar un sistema de lazo cerrado en continuo.

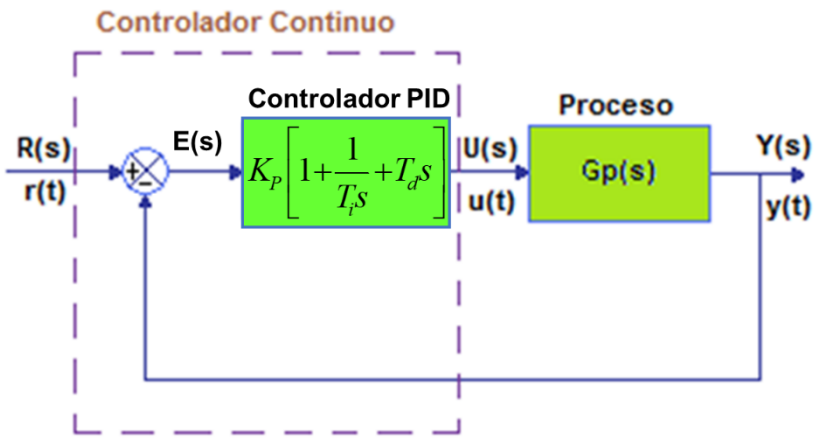

Figura 1.1. Diagrama de un controlador analógico.

El controlador analógico, encerrado en el cuadrado con líneas punteadas, puede reemplazarse por un controlador digital como se muestra en la figura 1.2, el cual hace la misma tarea de control que el controlador analógico. La diferencia básica entre estos controladores es que el sistema digital opera con señales discretas (o muestras de la señal sensada) en lugar de operar con señales continuas.

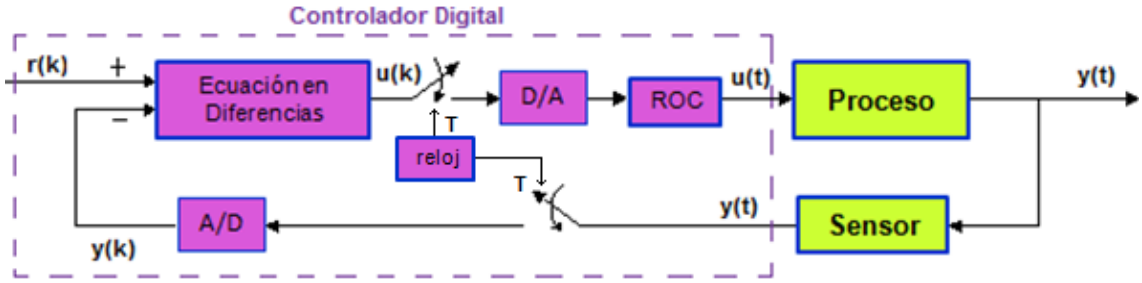

Figura 1.2. Diagrama de un controlador digital.

El control digital es llevado a cabo por una computadora, un PLC, un microcontrolador o un DSP. Para poder ingresar la señal o variable a controlar *y(t),* es necesaria una tarjeta de adquisición de datos, la cual usa un conversor analógico digital (A/D) para procesar datos analógicos para presentarlos a una computadora.

Un ADC (Convertidor Analógico - Digital por sus siglas en inglés) se encarga de convertir un valor analógico de voltaje a su correspondiente combinación binaria. Este voltaje es proporcional al valor de la variable física, esta es la forma en que un sensor proporciona la medición de la variable a controlar, como se muestra en la figura 1.1 *y(t)*.

El bloque marcado con ecuación en diferencias, corresponde a la computadora, el PLC, etc. quien realizará el PID ó ley de control, (Será la señal de control *u(t)* necesaria que hará que la señal a controlar *y(t)* llegue al valor deseado) que en discreto, se reduce a una ecuación en diferencias. La señal de control u(k) tendrá

que pasar de una combinación binaria, que es lo que podemos obtener de la computadora, a una señal analógica u(t), esto lo llevará a cabo el DAC (convertidor Digital – Analógico) que está contenido en la tarjeta de adquisición de datos.

En la figura 1.4, se puede observar la descripción grafica de un sistema de control digital. El objetivo del control para este caso es hacer que la temperatura del módulo llegue a 80°, esta vendría siendo la referencia o setpoint, en otras palabras el objetivo de control es hacer que la salida llegue a una señal de referencia. Para obtener el objetivo es necesario manipular la energía u(t), de tal forma que obtengamos la temperatura deseada o señal deseada.

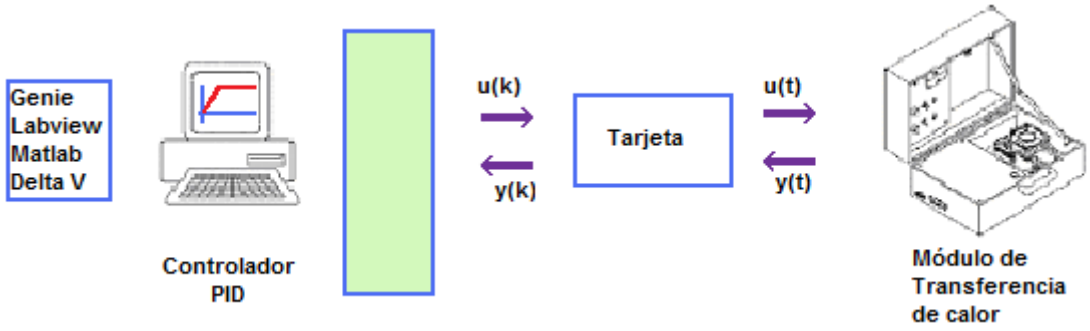

Figura 1.3. Diagrama de la implementación de un controlador digital.

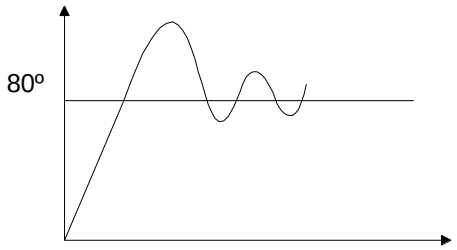

Figura 1.4. Salida y(t) del módulo de transferencia de calor.

En el sistema estamos viendo que la señal de control (u(t) o u(k)) es obtenida de una computadora, la cual contiene el controlador y de ahí es obtenida la señal de control. Esta señal es obtenida de paquetes tales como Genie, Labviews, Matlab, Delta V (para nuestro caso particular Matlab) los cuales estamos acostumbrados a verlos en bloques de control, que contienen la ecuación en diferencias, que representan la señal de control.

¿Qué es lo que necesita esta ecuación para ser implementada? Necesita la señal de control calculada pasada u(k-1), la señal de control calculada antepasada u(k-2), el error actual e(k) entre el valor deseado r(k) (llamado setpoint ó señal de referencia) y la señal a controlar y(k). También necesitará e(k-1) error pasado y e(k-2) error antepasado. Esto lo vamos a deducir a continuación.

PID Discreto
Parte Integral:

La parte integral se desarrolla sumando el área bajo la curva, aproximándola a través de rectángulos.

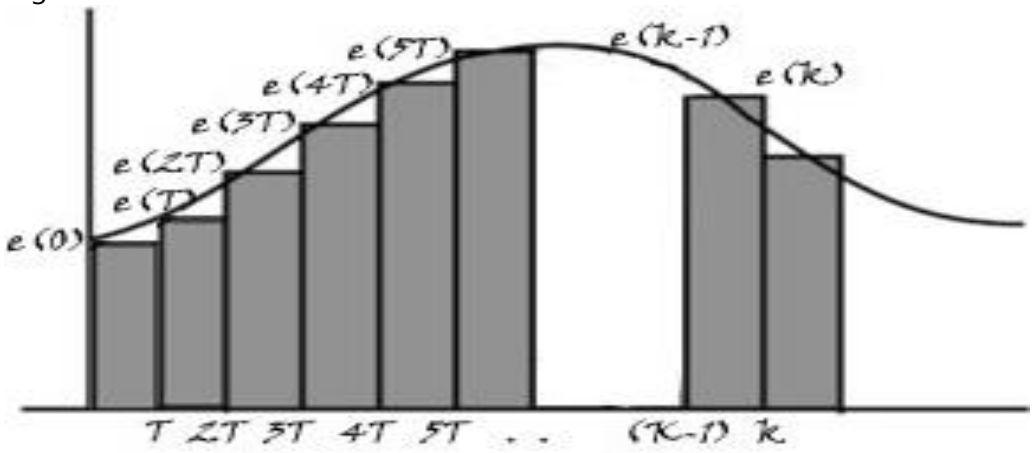

Figura 1.5. Aproximación a la Integral.

$$I(0) = e(0)T$$
$$I(T) = e(T)T$$
$$I(2T) = I(T) + e(2T)T$$
$$\vdots$$
$$I(kT) = I\left[(k-1)\right] + e(kT)T$$
$$I(z) = z^{-1}I(z) + E(z)T$$
$$I(z) = \frac{T}{1 - z^{-1}}E(z)$$

Parte Derivativa:

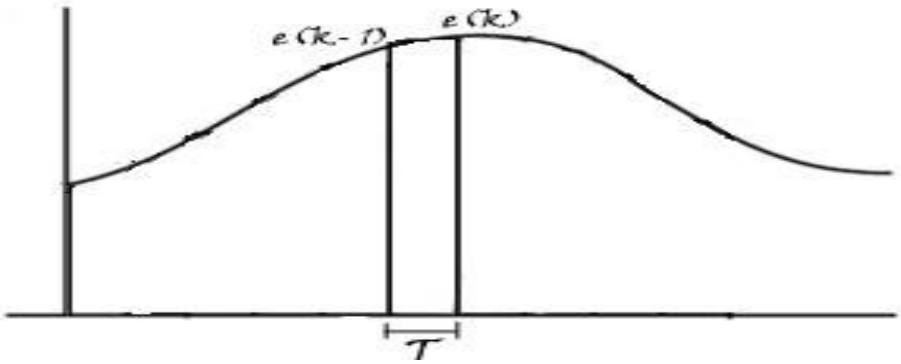

Figura 1.6. Aproximación a la derivada.

$$D(kT) = \frac{e(k) - e(k-1)}{T}$$
$$D(z) = \frac{E(z) - z^{-1}E(z)}{T}$$
$$D(z) = \frac{1 - z^{-1}}{T}E(z)$$

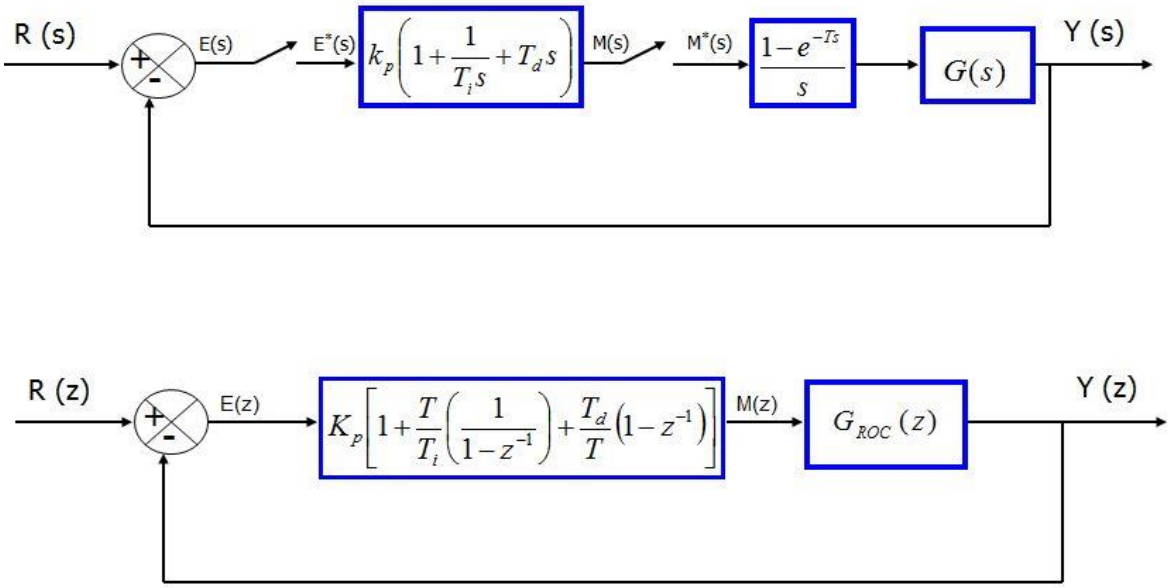

Figura 1.7. Diagrama de control general de un sistema digital.

Por lo que el controlador PID digital basado en las aproximaciones a la derivada e Integral, está dado por:

$$M(z)=k_p\left[1+\frac{T}{T_i}\left(\frac{1}{1-z^{-1}}\right)+\frac{T_d}{T}\left(1-z^{-1}\right)\right]E(z)$$

$$\left(1-z^{-1}\right)M(z)=k_p\left[\left(1-z^{-1}\right)+\frac{T}{T_i}+\frac{T_d}{T}\left(1-z^{-1}\right)^2\right]E(z)$$

$$\left(1-z^{-1}\right)M(z)=k_p\left[\left(1-z^{-1}\right)+\frac{T}{T_i}+\frac{T_d}{T}\left(1-2z^{-1}+z^{-2}\right)\right]E(z)$$

$$M(z)-z^{-1}M(z)=k_p\left(1+\frac{T}{T_i}+\frac{T_d}{T}\right)E(z)-k_p\left(1+\frac{2T_d}{T}\right)z^{-1}E(z)+k_p\frac{T_d}{T}z^{-2}E(z)$$

$$m(k)=m(k-1)+k_p\left(1+\frac{T}{T_i}+\frac{T_d}{T}\right)e(k)-k_p\left(1+\frac{2T_d}{T}\right)e(k-1)+k_p\frac{T_d}{T}e(k-2)$$

CAPÍTULO 2. DESARROLLO DEL PROYECTO

En este proyecto se realiza el prototipo para el control de posición de un vehículo sobre un riel basado en un control PI mediante el software Matlab con apoyo de Simulink.

En la figura 2.1 se muestra el esquema del prototipo el cual maneja los siguientes bloques:

Posición del carro

Se coloca a una distancia de 6.25 cm después del sensor, para que esta sea la posición inicial del carro (0 cm).

Sensor

Se trata de un sensor infrarrojo el cual está posicionado en forma vertical en frente del carro, de esa forma no le afectará ningún tipo de objeto que se encuentre alrededor del riel y se conecta a la placa Arduino Uno, enviando la posición del carro.

Controlador

En este bloque nos encontramos con la placa Arduino Uno la cual es conectada a la PC mediante un cable tipo USB.

Etapa de potencia

Contamos con un puente H que permitirá un giro para el motor tanto a la derecha como a la izquierda el cual estará conectado al Arduino Uno y estará conectado al motor de DC que mueve al carro.

Motor del carro

Se tiene un motor de DC con una caja reductora, la cual permite conectar un engrane que permitirá el movimiento sobre el riel, este motor se alimenta con 5 volts.

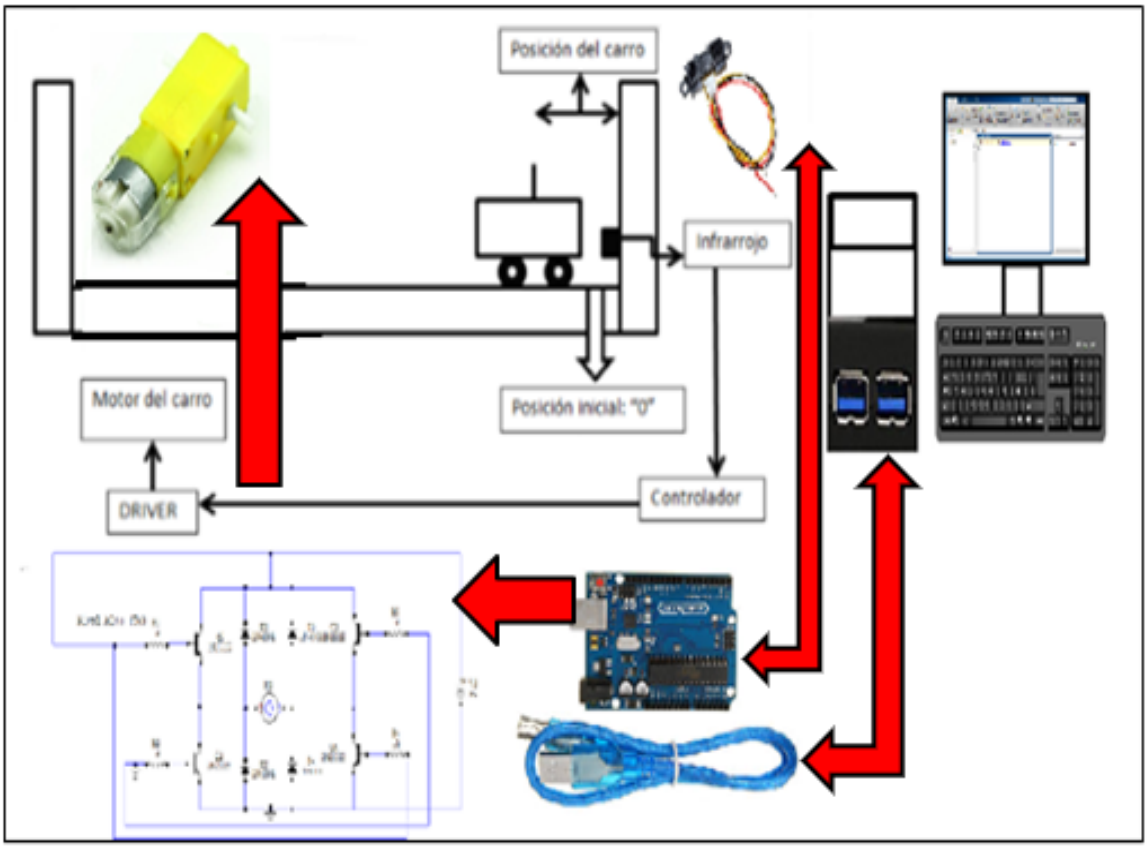

Figura 2.1. Esquema del prototipo.

En este capítulo también describiremos como se realizó cada componente del prototipo.

SENSOR INFRARROJO.

Esta etapa consiste en caracterizar el sensor infrarrojo seleccionado (Sharp GP2Y0A21YK0F) como se muestra en la figura 2.2 el cual a diferencia de otros sensores permite obtener distancias desde los 10 hasta los 80 cm y con la ventaja de que este mismo no es afectado en su funcionamiento por cuestiones de temperaturas. Cuenta con 3 pines de conexión los cuales como se muestra en la figura 2.2 el color negro corresponde a GND, el color rojo a VCC y el color amarillo es el voltaje de salida (Vo).

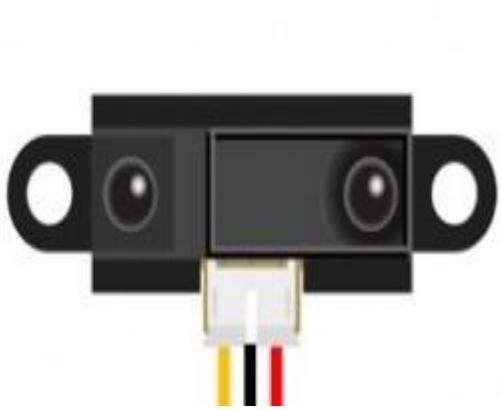

Figura 2.2. Sensor Sharp GP2Y0A21YK0F.

Para llevar acabo mediciones más precisas se optó por trabajar en el rango estable del sensor que se muestra en la figura 2.3 que es a partir de los 10 cm, ya que de lo contrario las mediciones resultaban erróneas.

Figura 2.3. Representación del rango mínimo del sensor Sharp.

MICROCONTROLADOR ARDUINO UNO.

La tarjeta Arduino Uno cuenta con un total de 14 entradas/salidas digitales de las cuales 6 se pueden utilizar como salidas PWM (Modulación por ancho de pulso) y otras 6 son entradas analógicas como se la muestra en la figura 2.4.

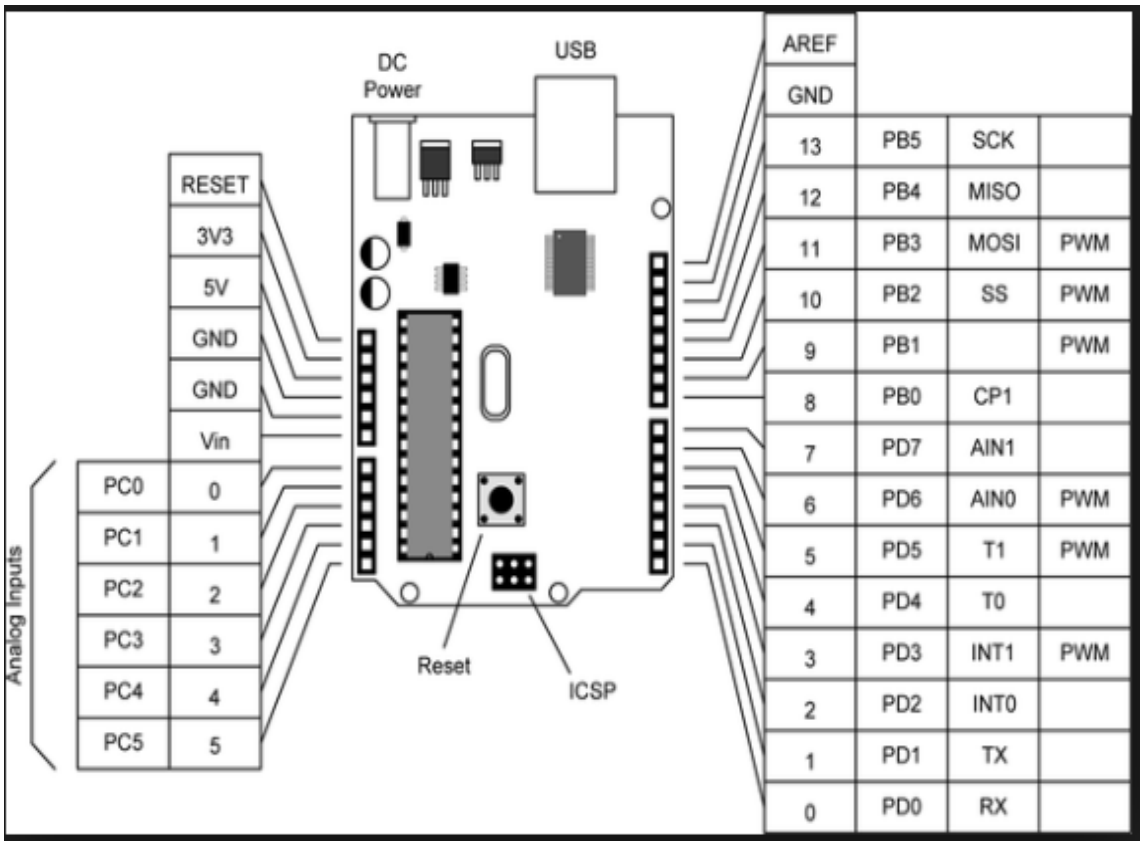

Figura 2.4. Esquema de pines del microcontrolador Arduino UNO.

TABLILLA PROTOBOARD.

La protoboard no es más que una placa de prueba en la cual se pueden colocar elementos electrónicos y cables que permitan la unión de uno o varios componentes sin la necesidad de soldar ninguno de los componentes y con la ventaja, de que si algún componente llega a quemarse, este se pueda remplazar rápidamente, sin necesidad de desoldar y nuevamente soldar la placa.

En muchas ocasiones llega a suceder que los cables que se insertan en la protoboard no quedan completamente introducidos en la placa por lo que a veces puede generar ruido, solo es cuestión de observar cuidadosamente que ningún cable o componente estén fuera de la placa.

En la figura 2.5 se muestra un ejemplo de una protoboard, aunque cabe resaltar que existen diferentes modelos de placa dependiendo el fabricante, pero su funcionamiento es el mismo.

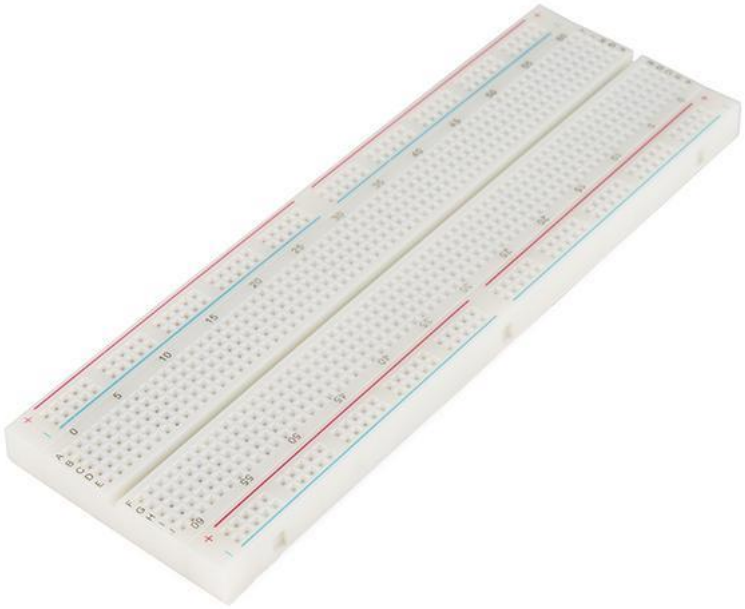

Figura 2.5. Placa Protoboard.

INTERCONEXIÓN DEL SENSOR SHARP CON EL ARDUINO UNO.

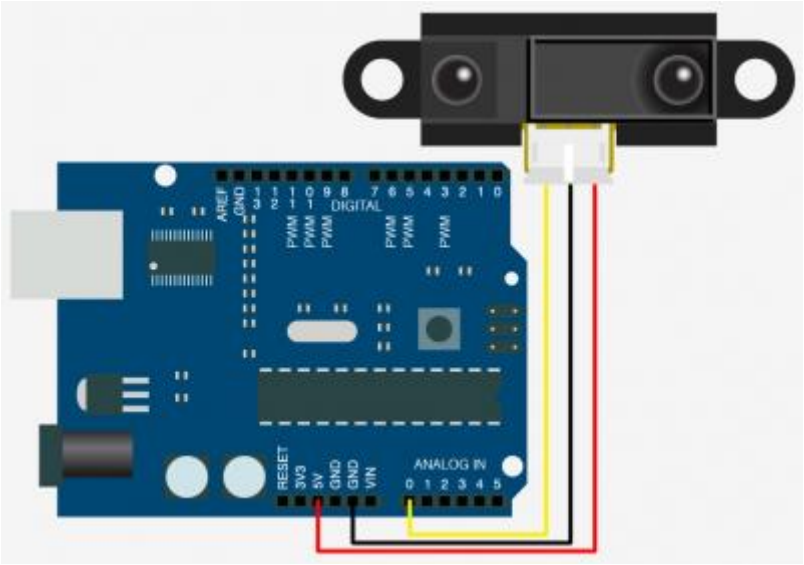

Figura 2.6. Interconexión de Arduino UNO con sensor Sharp.

En la figura 2.6 se muestra la interconexión del sensor Sharp con el microcontrolador (Arduino uno) y como se puede observar el sensor se alimenta directamente del Arduino conectando el cable rojo del sensor a 5v y el cable negro a GND y su pin amarillo (Vo) a una entrada analógica en este caso A0.

PRIMERA CARACTERIZACIÓN DEL SENSOR.

Este tipo de sensor entrega una salida de voltaje, correspondiente a la distancia de detección, por lo que se llevó acabo su caracterización tomando valores a partir de los 10 cm (rango estable) y así sucesivamente de 10 cm en 10 cm hasta llegar a una distancia total de 80 cm y sus respectivas lecturas de ADC como se muestra en la tabla 1, con apoyo del software Excel graficamos y obtuvimos la fórmula de la curva.

Distancia(cm)	Lectura de ADC
10	450
20	242
30	185
40	154
50	145
60	129
70	117
80	113

Tabla 2.2. Distancia vs Lectura de ADC.

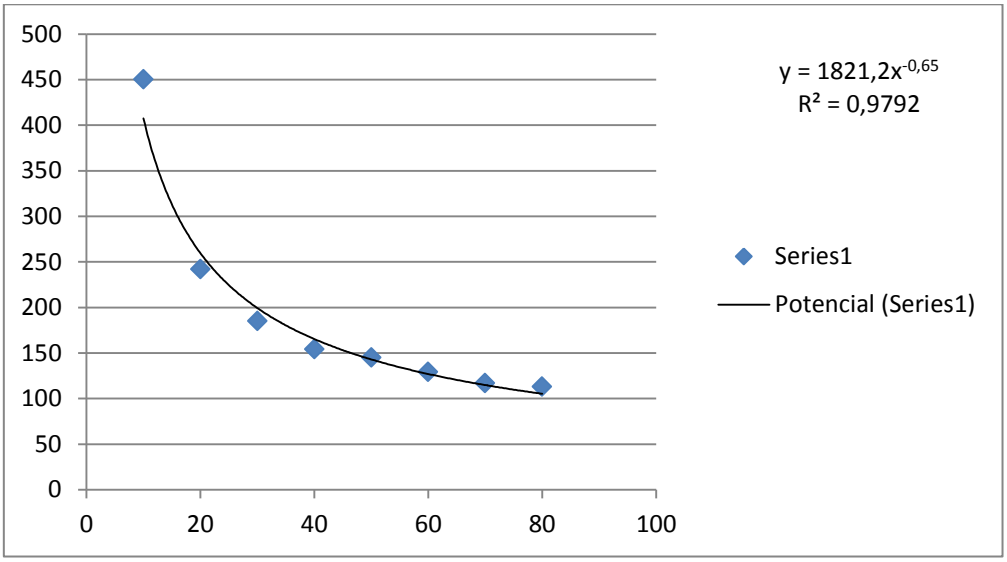

Figura 2.7. Gráfica de distancia vs Lectura de ADC.

SEGUNDA CARACTERIZACIÓN DEL SENSOR.

Se llevó a cabo una segunda caracterización del sensor ya que la primera no resultó ser óptima para el cálculo de la distancia del sensor, esta vez tomando

valores desde los 10 cm hasta los 25 cm ya que el rango de trabajo era más estable como se muestra en la tabla 2.

Distancia(cm)	Lectura de ADC
10	501
11	444
12	413
13	389
14	361
15	341
16	312
17	301
18	285
19	273
20	261
21	249
22	241
23	228
24	220
25	212

Tabla 2.2. Distancia vs Lectura de ADC.

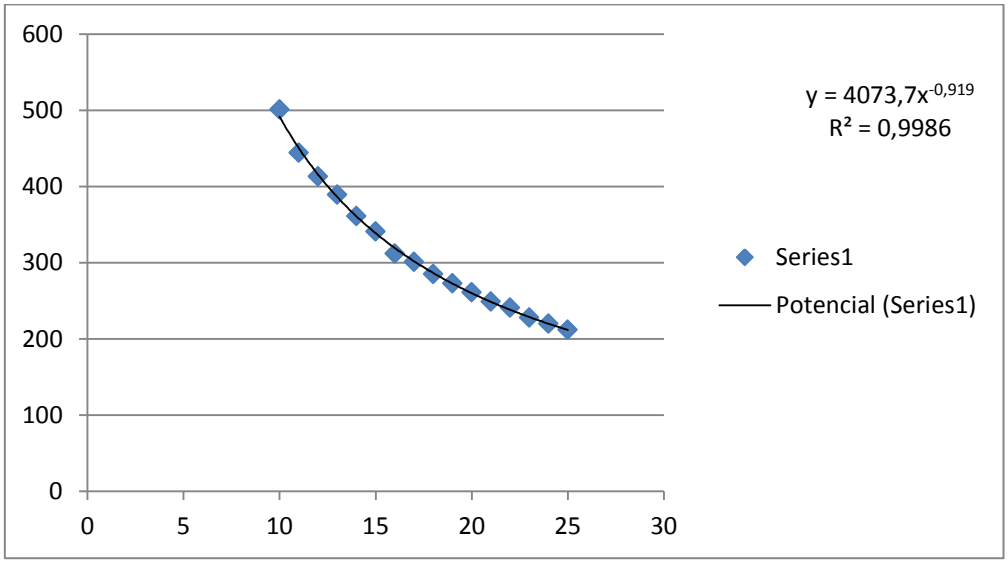

Figura 2.8. Gráfica de distancia vs Lectura de ADC.

Después de haber realizado diferentes pruebas con el sensor Sharp, se mantenía el problema de no poder registrar valores exactos de distancias, ya que pese no ser afectado el sensor por cuestiones de temperatura, se encontró una

desventaja la cual es que al haber adquirido el producto, el sensor no contaba con una protección en su pequeña placa como se muestra en la figura 2.9 por lo que al estar trasladando el sensor de un lado a otro para las diferentes pruebas este fue dañado, como consecuencia ya no se tuvo un buen funcionamiento para el prototipo además de solo poder registrar una distancia máxima de 20 cm por el mismo problema y se optó por cambiar el sensor Sharp por un sensor ultrasónico, quien será usado en todo el Proyecto Terminal.

Figura 2.9. Placa de sensor Sharp.

SENSOR ULTRASÓNICO.

Al tener muchas lecturas erróneas por parte del sensor Sharp se optó por cambiarlo por un sensor ultrasónico como se muestra en la figura 2.10 (modelo HC-SR04) el cual permite medir distancias a través de los ultrasonidos ya que a diferencia del sensor infrarrojo (Sharp) que utiliza propiedades de la luz para calcular la distancia, el sensor ultrasónico utiliza las propiedades de propagación del sonido para medir distancias.

Cuenta con 4 pines de conexión los cuales no son representados por colores, pero no es inconveniente ya que el sensor indica cada uno de sus pines como se puede observar en la figura 2.10, en este caso VCC es conectado a la tarjeta Arduino para su alimentación al igual que GND, los pines TRIG y ECHO no tienen un orden específico de conexión en el Arduino, pero se pueden utilizar cualquiera de las salidas digitales.

Figura 2.10. Sensor Ultrasónico HC-SR04.

INTERCONEXIÓN DEL SENSOR ULTRASÓNICO CON EL ARDUINO UNO.

En la siguiente figura se muestra la interconexión del sensor ultrasónico con el Arduino Uno, pero como detalle fundamental cabe resaltar que los pines TRIG y ECHO del sensor ultrasónico no tienen un orden específico de conexión como se muestra en la sección MICROCONTROLADOR ARDUINO UNO, ya que se pueden utilizar cualquiera de sus 12 salidas digitales.

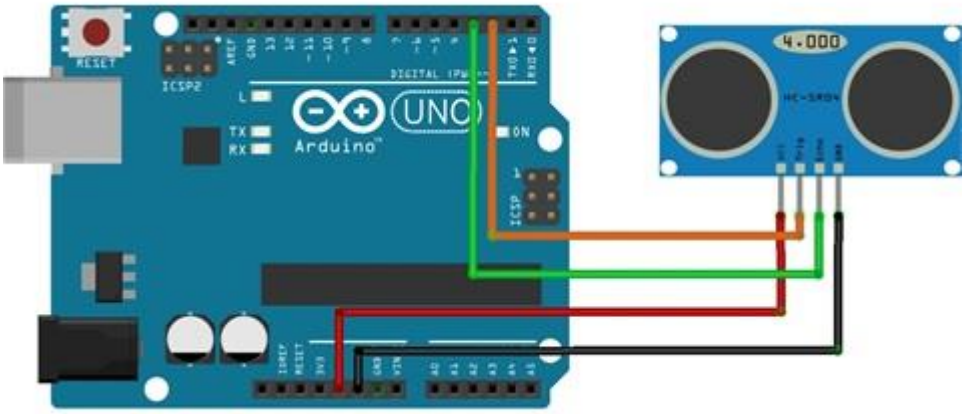

Figura 2.11. Interconexión del sensor ultrasónico y Arduino Uno.

CÓDIGO PARA MEDIR DISTANCIAS CON SENSOR ULTRASÓNICO.

El código que se muestra a continuación fue programado en Arduino con la finalidad de medir distancias con el sensor ultrasónico.

Primero se declararon dos variables de valor entero, una con el nombre de pin_TRIG destinada a la salida digital 8 del Arduino y otra con el nombre de pin_ECHO de igual forma destina a la salida digital 12, estas salidas no son exclusivas, se pueden utilizar cualquiera de las 12 permitidas incluyendo las PWM como ya se había mencionado anteriormente.

Se declararon otras tres variables dos de tipo entero (tiempoSenal y distanciaCM) y una de tipo flotante con el nombre de distanciaM, los nombres tampoco son exclusivos ya que las variables pueden tener cualquier otro nombre, la cuestión está en recordar la función que tiene cada uno de ellos.

```
int pin_TRIG = 8; //Pin para enviar pulsos
int pin_ECHO = 12; //Pin para recibir la señal de rebote
int tiempoSenal;
int distanciaCM;
float distanciaM;
```

En el menú de void setup se inicializan las variables pin_TRIG y pin_ECHO una como salida y la otra como entrada respectivamente y Serial.begin (9600) abre

el puerto serie con la finalidad de poder observar los valores de distancia en el monitor serial del Arduino.

```
void setup() {
  Serial.begin(9600);
  pinMode(pin_TRIG, OUTPUT);
  pinMode(pin_ECHO, INPUT);
}
```

En el menú de void loop primero se estabiliza el sensor con un nivel bajo (pin_TRIG, LOW) y con un delay de 5 microsegundos a la variable pin_TRIG y enseguida se envía un pulso para iniciar el sensor en este caso se mandó a un nivel alto (pin_TRIG, HIGH) con un delay de 10 microsegundos, luego en tiempoSenal se guarda el tiempo que dura el pin_ECHO en HIGH para que finalmente se realice la siguiente conversión "*((0.034 * tiempoSenal) / 2)-5"* y el resultado de la conversión nos dará como resultado la distancia en centímetros guardando ese valor en la variable distanciaCM, ese dato de la distancia se manda al monitor serial mediante *Serial.println(distanciaCM)* durante un delay de 100 microsegundos (0.1 segundos) para poder leerlo y así se repite el mismo proceso hasta que se quiera detener el programa.

```
void loop() {
  digitalWrite(pin_TRIG, LOW);
  delayMicroseconds(5); //Estabiliza el sensor
  digitalWrite(pin_TRIG, HIGH); //Envía el pulso para iniciar el sensor
  delayMicroseconds(10);
  tiempoSenal = pulseIn(pin_ECHO, HIGH); //Tiempo que dura el pin ECHO en HIGH
  distanciaCM=((0.034 * tiempoSenal) / 2)-5; //Convierte el tiempo en distancia (cm)
  Serial.println(distanciaCM);
  delay(100);
}
```

Finalmente, con el sensor ultrasónico hubo un menor error en las mediciones de las distancias.

ETAPA DE POTENCIA.

DISEÑO DE PUENTE H.

En esta etapa se llevó a cabo el diseño de un puente H el cual permitirá el movimiento en sentido horario y antihorario de un motor de DC, se propuso el diseño de la figura 2.12 para el puente H:

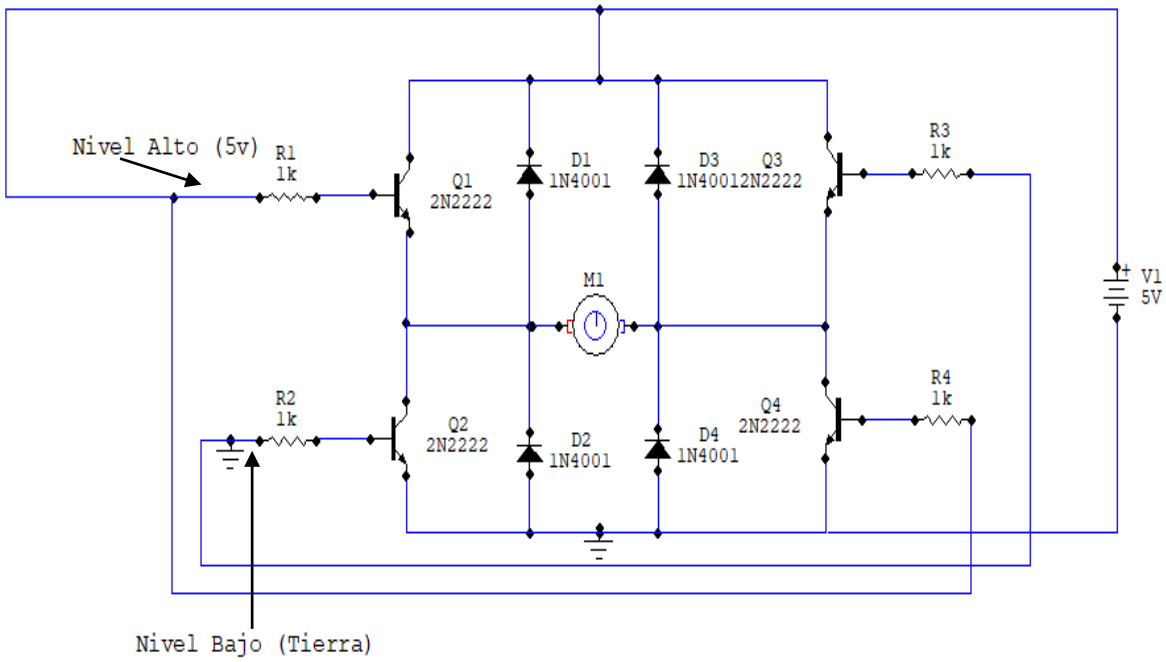

Figura 2.12. Diagrama de puente H en sentido horario.

PRIMERA PRUEBA DEL PUENTE H.

Una vez armado el circuito se llevaron a cabo diferentes pruebas para comprobar que el funcionamiento del puente H era el correcto como se muestra en la figura 2.12, el cual hace que el motor se mueva en sentido horario, ya que se tenía un nivel alto de voltaje (5 Volts) en dos transistores (Q1 y Q4) mientras que los otros transistores (Q2 y Q3) estaban en un nivel bajo es decir estaban referidos a tierra para que de esa forma el movimiento del motor fuera hacia un solo sentido (a la derecha).

SEGUNDA PRUEBA DEL PUENTE H.

Como segunda prueba sobre el mismo circuito se invirtieron los niveles de voltajes de los transistores, es decir, los transistores (Q2 y Q3) estaban a un nivel alto de voltaje en este caso de 5v y por ende Q1 y Q4 ahora estaban referidos a tierra, permitiendo el movimiento del motor en un sentido antihorario como se muestra en la figura 2.13.

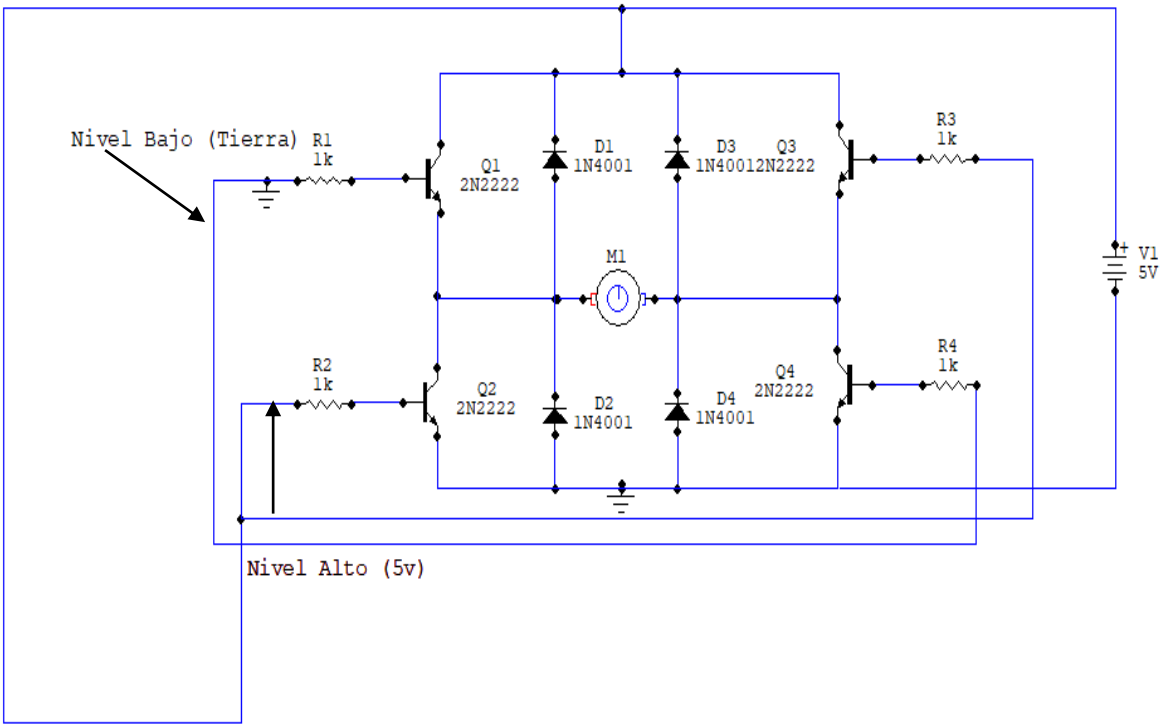

Figura 2.13. Diagrama de puente H en sentido antihorario.

En un principio el movimiento del motor en ambos sentidos fue exitoso, pero con la peculiaridad de que al registrar el voltaje que se estaba recibiendo del motor no correspondía a 5v sino a 3,6v por lo que había pérdida de voltaje dentro del puente H, por lo que se realizaron mediciones con un multímetro digital en el circuito y se encontró que la perdida de voltaje se encontraba en los transistores, ya que un transistor está compuesto por un diodo entre su base y emisor el cual tiene una caída de voltaje de 0.6v a 0.7v y siendo que se tenían dos transistores en funcionamiento, al girar en cualquiera de los dos sentidos se perdían 1.4v aproximadamente, por lo cual el motor no giraba de la manera más óptima y después de varios análisis del circuito se podía compensar esa pérdida de voltaje en el circuito, pero esto requería de agregar una cantidad más grande de componentes electrónicos, que muchas veces conllevan a tener un circuito que genere más ruido y al mismo tiempo pérdidas de voltaje en el circuito, por lo que se optó por trabajar con un circuito integrado, que ya contara con un puente H para que de esa forma el motor trabajara correctamente.

CIRCUITO INTEGRADO L293D.

Al tratarse de un prototipo se propuso el circuito integrado L293D, ya que este cuenta con dos configuraciones de puente H como se muestra en la figura 2.14 además de ser un producto electrónico muy comercial y de costo razonable (aproximadamente de 35 pesos mexicanos).

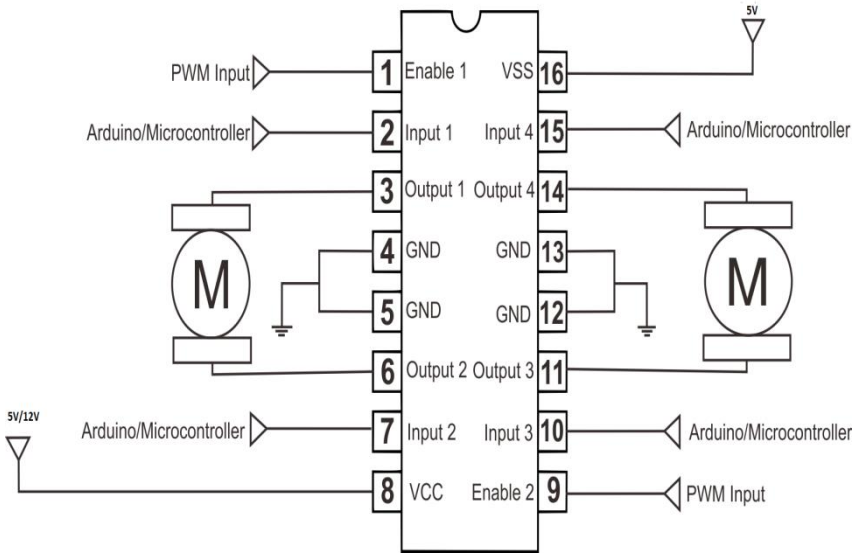

Figura 2.14. Circuito L293D con control de motor DC Bidireccional.

PROTECCION DEL L293D.

Una vez montado el circuito integrado L293D se procedió a revisar en el datasheet una configuración necesaria para conectar el motor de DC, encontrándose que si se conectaba directamente a la salida 3 y 6 del integrado, podría generar una descarga al no estar protegido, por lo que el datasheet indica en la figura 2.15 como se debe proteger el motor.

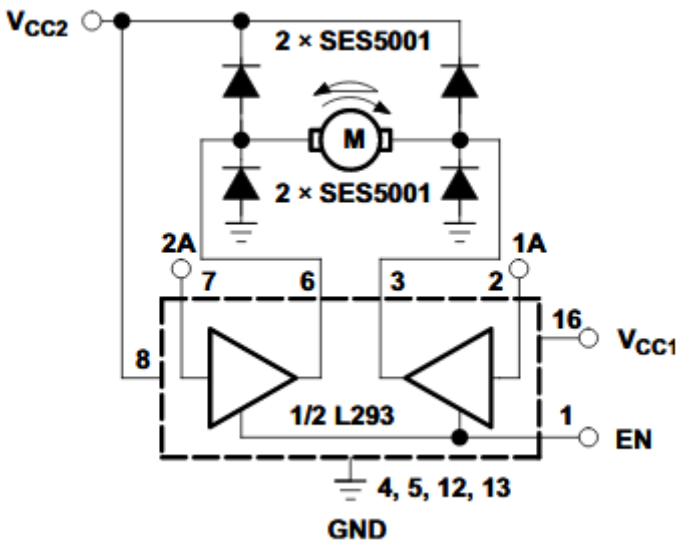

Figura 2.15. Conexión de protección del motor DC.

CIRCUITO CON GIRO A LA DERECHA CON POTENCIÓMETRO.

Se volvió armar el circuito pero esta vez conectando el integrado L293D y agregando un potenciómetro para el control de velocidad del motor, se utilizó solo el movimiento hacia la derecha del motor para realizar diferentes pruebas, se conectó el enable1 al potenciómetro y la entrada 1 y 2 del integrado (pin2 y pin7 respectivamente) para poder realizar el movimiento hacia la derecha se dejó en nivel bajo el pin2 (entrada1) y a nivel alto el pin7 (entrada2), así al alimentar el motor, realizaba movimiento hacia la derecha al girar el potenciómetro, a la izquierda bajaba la velocidad y al girarlo a la derecha aumentaba su velocidad, el circuito quedó armado como se muestra en la figura 2.16.

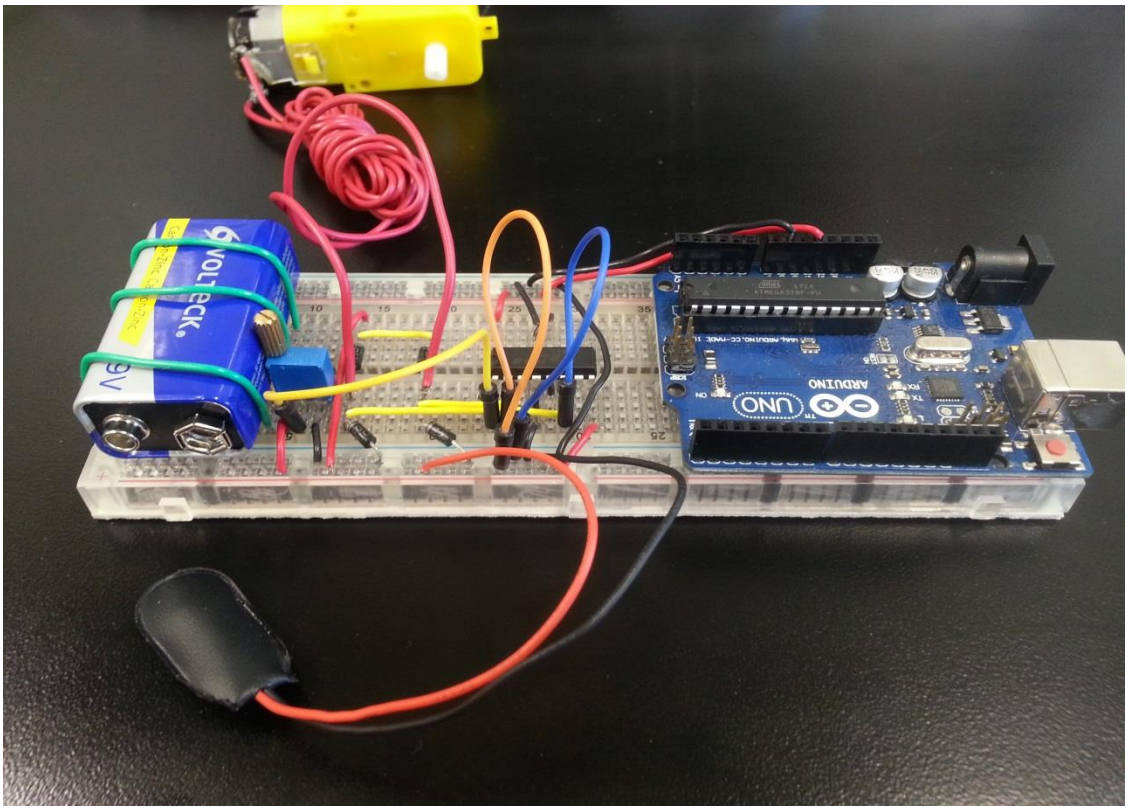

Figura 2.16. Circuito con giro a la derecha, regulando velocidad con potenciómetro.

Al tener un funcionamiento correcto del motor, se procedió a remover el potenciómetro, para que la velocidad del motor se controlara con una señal PWM, la cual fue generada por el microcontrolador. Las entradas del circuito integrado se conectaron a las salidas digitales 4 y 7 del microcontrolador como se muestra en la figura 2.17 después se procedió a escribir el código en el microcontrolador para que este funcionara a un ciclo de trabajo del 98,50 y 25% girando a la derecha y a la izquierda, al querer utilizar el 100% del ciclo de trabajo resultaba bastante difícil de analizar en el osciloscopio ya que solo mostraba la parte negativa en el osciloscopio, pero gracias a que el circuito

integrado cuenta con dos configuraciones de puente H, se cambiaron todas las conexiones del circuito integrado y al analizar las señales estas se mostraban correctamente.

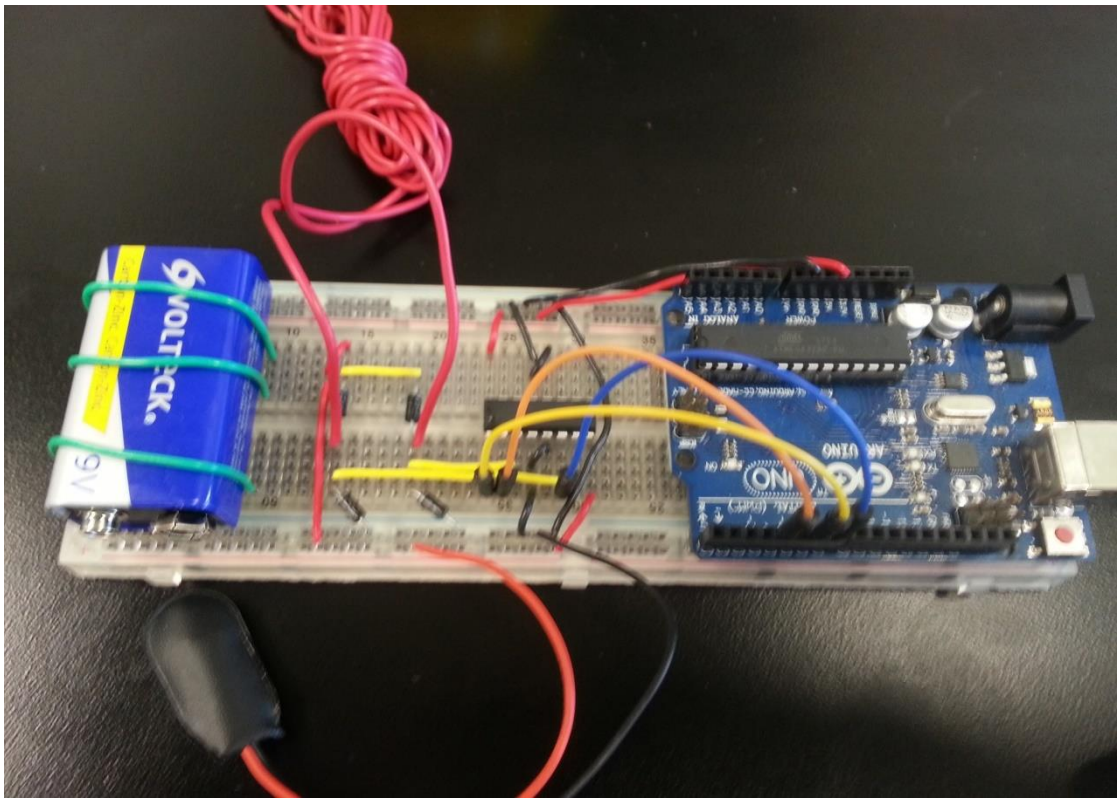

Figura 2.17. Circuito con giro a la derecha e izquierda, regulando velocidad con señal PWM.

BATERIA EXTERNA.

Se coloco una batería externa de 9 volts del fabricante VOLTECK, la cual se muestra en la figura 2.18, aunque cualquier batería que entregue 9 volts también puede servir, ya que el arduino no proporciona el suficiente voltaje para poder alimentar el circuito integrado L293D(la bateria es conectada a VCC2 de la figura 2.15), un detalle curioso y muy importante que cabe recordar a cada momento, es que cualquier bateria tiene un límite de vida, esto debido al desgaste o uso que se le de, por eso es necesario cambiarla al menos una vez por mes, en el caso de haberla usado continuamente todos los dias.

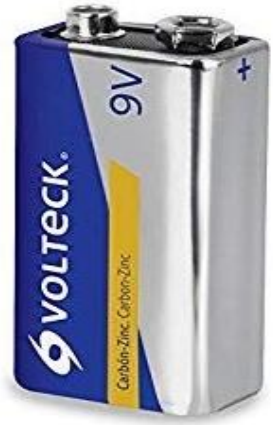

Figura 2.18. Batería de 9 volts.

CODIGO EN MICROCONTROLADOR PARA CONTROL DE MOTOR DC.

Primero se declararon las variables a ocupar en el programa y en este caso tenemos las variables de valor entero para SignalPWM (salida digital 6), MotorEntrada1 (salida digital 4) y MotorEntrada2 (salida digital 7). En el void loop

```
const int SignalPWM = 6;                // PWM se conecta al pin 9 del LM239D
const int MotorEntrada1 = 4;            // Entrada 10 del LM239D
const int MotorEntrada2 = 7;            // Entrada 15 del LM239D
```

En el void setup se inicializaron como salidas SignalPWM, MotorEntrada1 y MotorEntrada2.

```
void setup()
{
  pinMode(SignalPWM,OUTPUT);            //Configura SignalPWM como salida
  pinMode(MotorEntrada1, OUTPUT);       //Configura MotorEntrada1 como salida
  pinMode(MotorEntrada2, OUTPUT);       //Configura MotorEntrada2 como salida
}
```

En el void loop se colocó en un nivel alto a MotorEntrada1 mientras que MotorEntrada2 se colocó en un nivel bajo para después colocar a SignalPWM a un 98% de ciclo de trabajo (la señal PWM registra valores desde 0 hasta 255, siendo este su 100%, por lo que se utilizó un porcentaje menor para no saturar la señal PWM) durante 5 segundos con la finalidad de lograr un giro a la derecha con el motor de DC como se muestra en la figura 2.19.
Después se colocaron MotorEntrada1 y MotorEntrada2 en niveles altos con un retardo de 0.2 segundos.

```
void loop()
{
Movimiento a la derecha para el motor de DC
```

```
digitalWrite(MotorEntrada1,HIGH);        //Se coloca en un nivel alto a MotorEntrada1
digitalWrite(MotorEntrada2,LOW);         //Se coloca en un nivel bajo a MotorEntrada2
analogWrite(SignalPWM,250);              //El ciclo de trabajo a un 98%
delay(5000);                             //Durante 5 segundos

digitalWrite(MotorEntrada1,HIGH);
digitalWrite(MotorEntrada2,HIGH);
delay(200);
```

Enseguida se procedió a detener el motor de DC para colocar un ciclo de trabajo distinto por lo que MotorEntrada1 y MotorEntrada2 se colocaron en niveles bajos durante 3 segundos, después se colocó en un nivel alto a MotorEntrada1 mientras que MotorEntrada2 se colocó en un nivel bajo para después colocar a SignalPWM a un 50% de ciclo de trabajo durante 5 segundos con la finalidad de lograr un giro a la derecha con el motor de DC como se muestra en la figura 2.20.
Después se colocaron MotorEntrada1 y MotorEntrada2 en niveles altos con un retardo de 0.2 segundos.

```
digitalWrite(MotorEntrada1,LOW);
digitalWrite(MotorEntrada2,LOW);
delay(3000);

digitalWrite(MotorEntrada1,HIGH);        //Se coloca en un nivel alto a MotorEntrada1
digitalWrite(MotorEntrada2,LOW);         //Se coloca en un nivel bajo a MotorEntrada2
analogWrite(SignalPWM,127);              //El ciclo de trabajo a un 50%
delay(5000);                             //Durante 5 segundos

digitalWrite(MotorEntrada1,HIGH);
digitalWrite(MotorEntrada2,HIGH);
delay(200);
```

Enseguida se procedió nuevamente a detener el motor de DC para colocar un ciclo de trabajo distinto por lo que MotorEntrada1 y MotorEntrada2 se colocaron en niveles bajos durante 3 segundos, después se colocó en un nivel alto a MotorEntrada1 mientras que MotorEntrada2 se colocó en un nivel bajo para después colocar a SignalPWM a un 25% de ciclo de trabajo durante 5 segundos con la finalidad de lograr un giro a la derecha con el motor de DC como se muestra en la figura 2.21.
Después se colocaron MotorEntrada1 y MotorEntrada2 en niveles altos con un retardo de 0.2 segundos.

```
digitalWrite(MotorEntrada1,LOW);
digitalWrite(MotorEntrada2,LOW);
delay(3000);
```

```
digitalWrite(MotorEntrada1,HIGH);       //Se coloca en un nivel alto a MotorEntrada1
digitalWrite(MotorEntrada2,LOW);        //Se coloca en un nivel bajo a MotorEntrada2
analogWrite(SignalPWM,67);              //El ciclo de trabajo a un 25%
delay(5000);                            //Durante 5 segundos

digitalWrite(MotorEntrada1,HIGH);
digitalWrite(MotorEntrada2,HIGH);
delay(200);
```

Después se procedió nuevamente a detener el motor de DC para colocar un ciclo de trabajo distinto por lo que MotorEntrada1 y MotorEntrada2 se colocaron en niveles bajos durante 3 segundos, después se colocó en un nivel alto a MotorEntrada1 mientras que MotorEntrada2 se colocó en un nivel bajo para después colocar a SignalPWM a un 0% de ciclo de trabajo durante 5 segundos, pero al tener un ciclo de trabajo de 0% no habría movimiento para ningún lado como se muestra en la figura 2.22.
Después se colocaron MotorEntrada1 y MotorEntrada2 en niveles altos con un retardo de 0.2 segundos.

```
digitalWrite(MotorEntrada1,LOW);
digitalWrite(MotorEntrada2,LOW);
delay(3000);

digitalWrite(MotorEntrada1,HIGH);       //Se coloca en un nivel alto a MotorEntrada1
digitalWrite(MotorEntrada2,LOW);        //Se coloca en un nivel bajo a MotorEntrada2
analogWrite(SignalPWM,0);           //El ciclo de trabajo a un 0%(motor Sin movimiento)
delay(5000);                        //Durante 5 segundos

digitalWrite(MotorEntrada1,HIGH);
digitalWrite(MotorEntrada2,HIGH);
delay(200);
```

Finalmente se colocaron a niveles bajos MotorEntrada1 y MotorEntrada2 con un retardo de 3 segundos para colocar otro ciclo de trabajo.

```
digitalWrite(MotorEntrada1,LOW);
digitalWrite(MotorEntrada2,LOW);
delay(3000);
```

Movimiento a la izquierda para el motor de DC

Se colocó en un nivel bajo a MotorEntrada1 mientras que MotorEntrada2, se colocó en un nivel alto para después colocar a SignalPWM a un 25% de ciclo de trabajo durante 5 segundos, con la finalidad de lograr un giro a la izquierda con el motor de DC como se muestra en la figura 2.23.

Después se colocaron MotorEntrada1 y MotorEntrada2 en niveles altos con un retardo de 0.2 segundos.

```
digitalWrite(MotorEntrada1,LOW);       //Se coloca en un nivel bajo a MotorEntrada1
digitalWrite(MotorEntrada2,HIGH);      //Se coloca en un nivel alto a MotorEntrada2
analogWrite(SignalPWM,67);             //El ciclo de trabajo a un 25%
delay(5000);                           //Durante 5 segundos

digitalWrite(MotorEntrada1,HIGH);
digitalWrite(MotorEntrada2,HIGH);
delay(200);
```

MotorEntrada1 y MotorEntrada2 se colocaron en niveles bajos durante 3 segundos, después se colocó en un nivel bajo a MotorEntrada1 mientras que MotorEntrada2, se colocó en un nivel alto para después colocar a SignalPWM a un 50% de ciclo de trabajo durante 5 segundos, con la finalidad de lograr un giro a la izquierda con el motor de DC como se muestra en la figura 2.24.
Después se colocaron MotorEntrada1 y MotorEntrada2 en niveles altos con un retardo de 0.2 segundos.

```
digitalWrite(MotorEntrada1,LOW);
digitalWrite(MotorEntrada2,LOW);
delay(3000);

digitalWrite(MotorEntrada1,LOW);       //Se coloca en un nivel bajo a MotorEntrada1
digitalWrite(MotorEntrada2,HIGH);      //Se coloca en un nivel alto a MotorEntrada2
analogWrite(SignalPWM,127);            //El ciclo de trabajo a un 50%
delay(5000);                           //Durante 5 segundos

digitalWrite(MotorEntrada1,HIGH);
digitalWrite(MotorEntrada2,HIGH);
delay(200);
```

MotorEntrada1 y MotorEntrada2 se colocaron en niveles bajos durante 3 segundos, se colocó en un nivel bajo a MotorEntrada1, mientras que MotorEntrada2 se colocó en un nivel alto para después colocar a SignalPWM a un 98% de ciclo de trabajo durante 5 segundos con la finalidad de lograr un giro a la derecha con el motor de DC como se muestra en la figura 2.25.
Después se colocaron MotorEntrada1 y MotorEntrada2 en niveles altos con un retardo de 0.2 segundos.

```
digitalWrite(MotorEntrada1,LOW);
digitalWrite(MotorEntrada2,LOW);
delay(3000);

digitalWrite(MotorEntrada1,LOW);       //Se coloca en un nivel bajo a MotorEntrada1
```

```
  digitalWrite(MotorEntrada2,HIGH);       //Se coloca en un nivel alto a MotorEntrada2
  analogWrite(SignalPWM,250);             //El ciclo de trabajo a un 98%
  delay(5000);                            //Durante 5 segundos

  digitalWrite(MotorEntrada1,HIGH);
  digitalWrite(MotorEntrada2,HIGH);
  delay(200);
```

MotorEntrada1 y MotorEntrada2 se colocaron en niveles bajos durante 3 segundos y se colocó en un nivel bajo a MotorEntrada1 mientras que MotorEntrada2 se colocó en un nivel alto para después colocar a SignalPWM a un 0% de ciclo de trabajo durante 5 segundos, pero al tener un ciclo de trabajo de 0% no habría movimiento para ningún lado como se muestra en la figura 2.22.

Después se colocaron MotorEntrada1 y MotorEntrada2 en niveles altos con un retardo de 0.2 segundos.
Y finalmente MotorEntrada1 y MotorEntrada2 se colocaron en niveles bajos con un retardo de 3 segundos.

```
  digitalWrite(MotorEntrada1,LOW);
  digitalWrite(MotorEntrada2,LOW);
  delay(3000);

  digitalWrite(MotorEntrada1,LOW);        //Se coloca en un nivel bajo a MotorEntrada1
  digitalWrite(MotorEntrada2,HIGH);       //Se coloca en un nivel alto a MotorEntrada2
  analogWrite(SignalPWM,0);               //El ciclo de trabajo a un 0%
  delay(5000);                            //Durante 5 segundos

  digitalWrite(MotorEntrada1,HIGH);
  digitalWrite(MotorEntrada2,HIGH);
  delay(200);

  digitalWrite(MotorEntrada1,LOW);
  digitalWrite(MotorEntrada2,LOW);
  delay(3000);
}
```

PRUEBAS EN OSCILOSCOPIO.

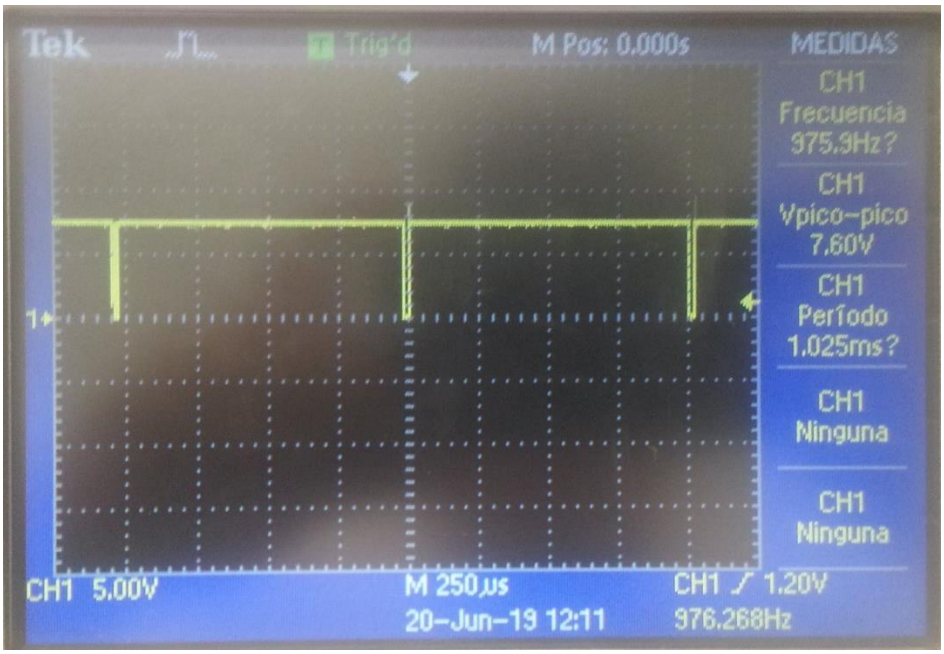

Figura 2.19. Ciclo de trabajo del 98% giro a la derecha (positivo).

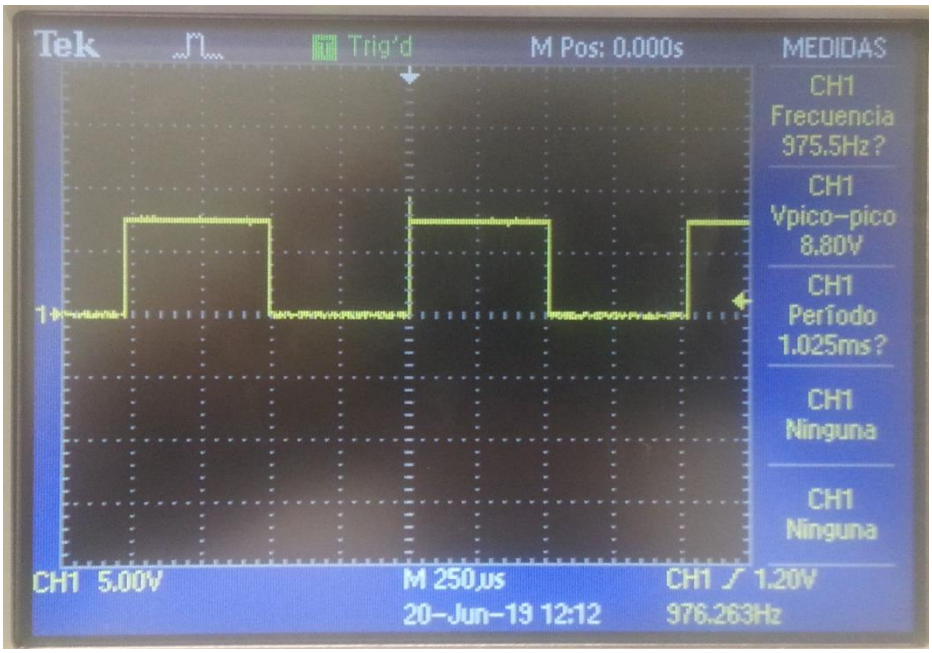

Figura 2.20. Ciclo de trabajo del 50% giro a la derecha (positivo).

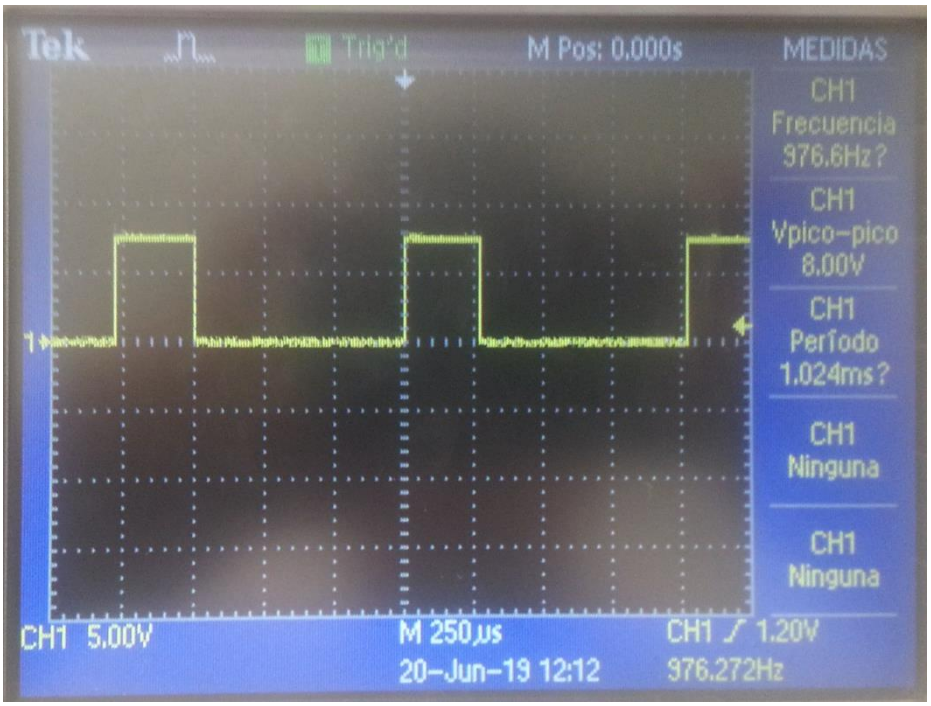

Figura 2.21. Ciclo de trabajo del 25% giro a la derecha (positivo).

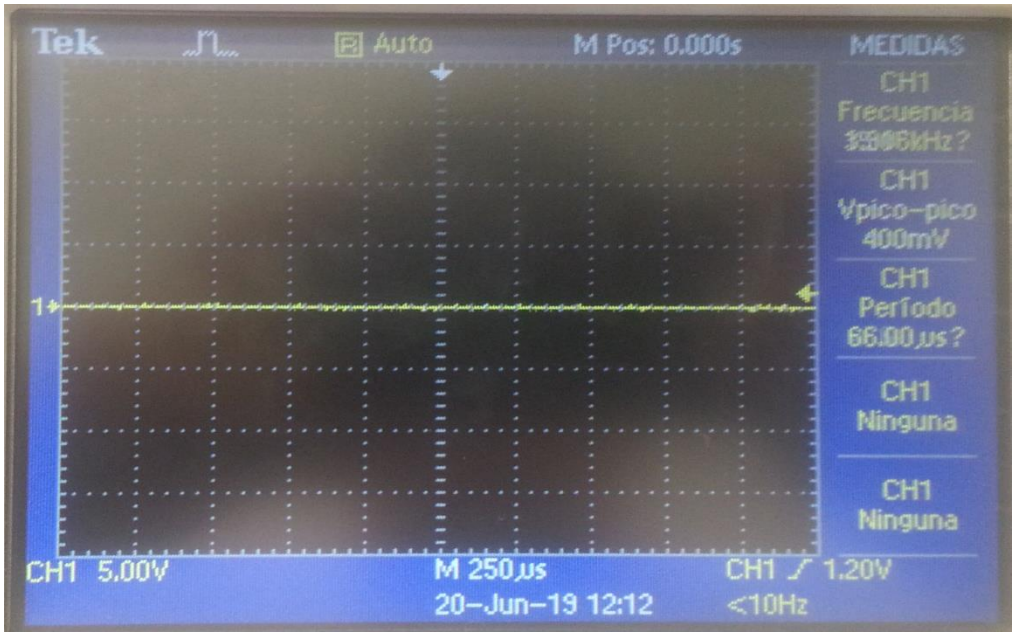

Figura 2.22. Ciclo de trabajo del 0% sin movimiento.

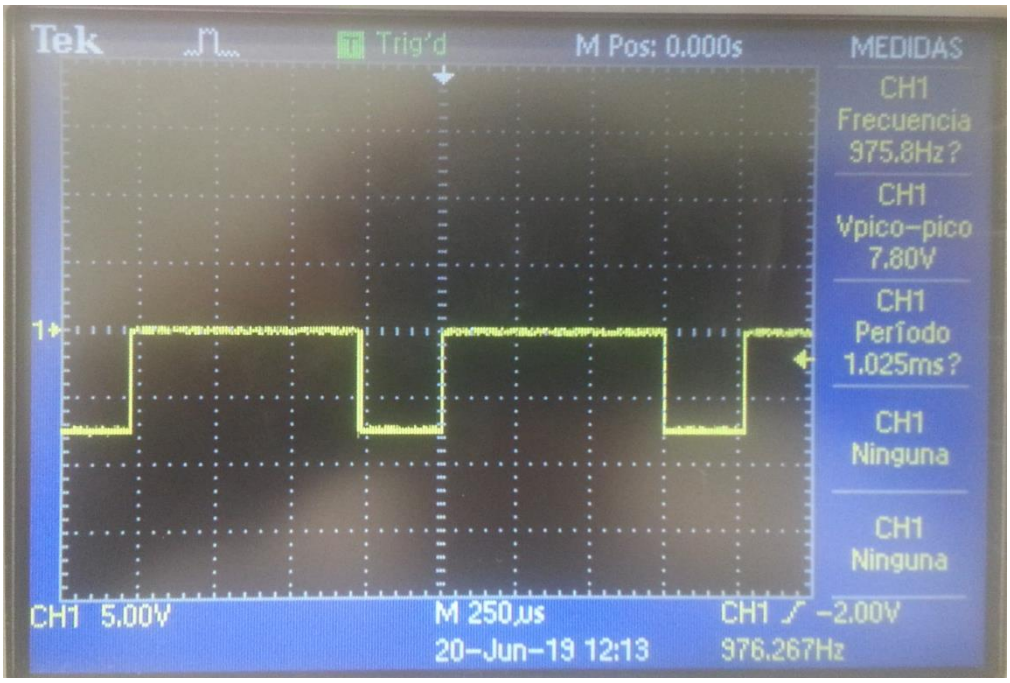

Figura 2.23. Ciclo de trabajo del 25% giro a la izquierda (negativo).

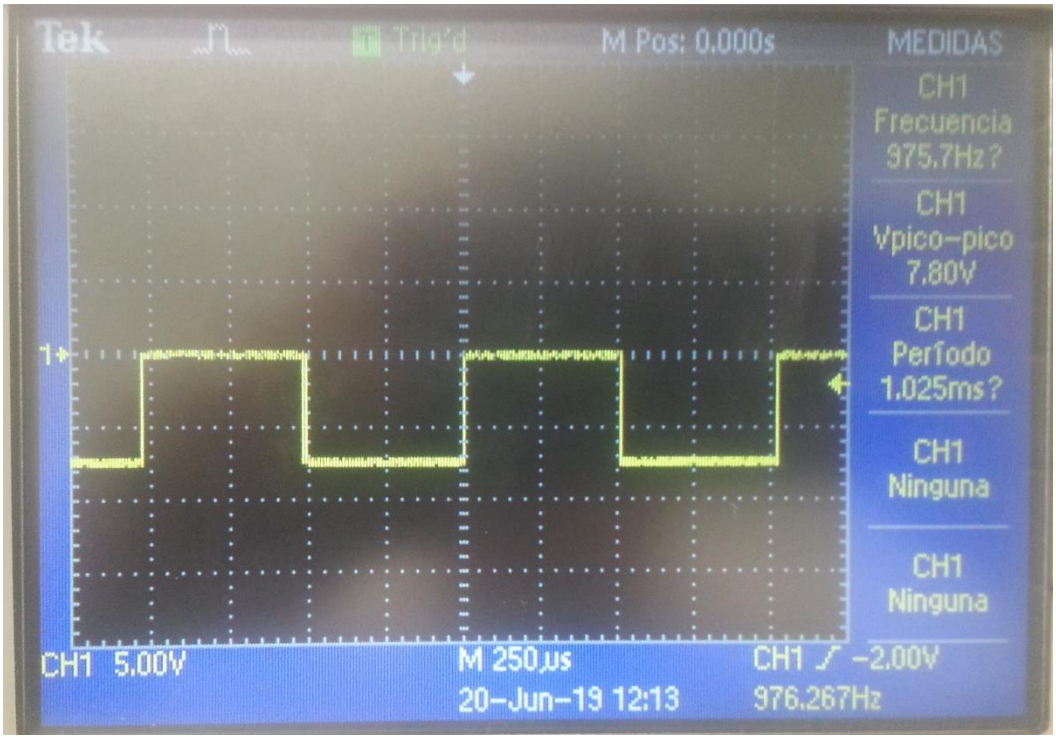

Figura 2.24. Ciclo de trabajo del 50% giro a la izquierda (negativo).

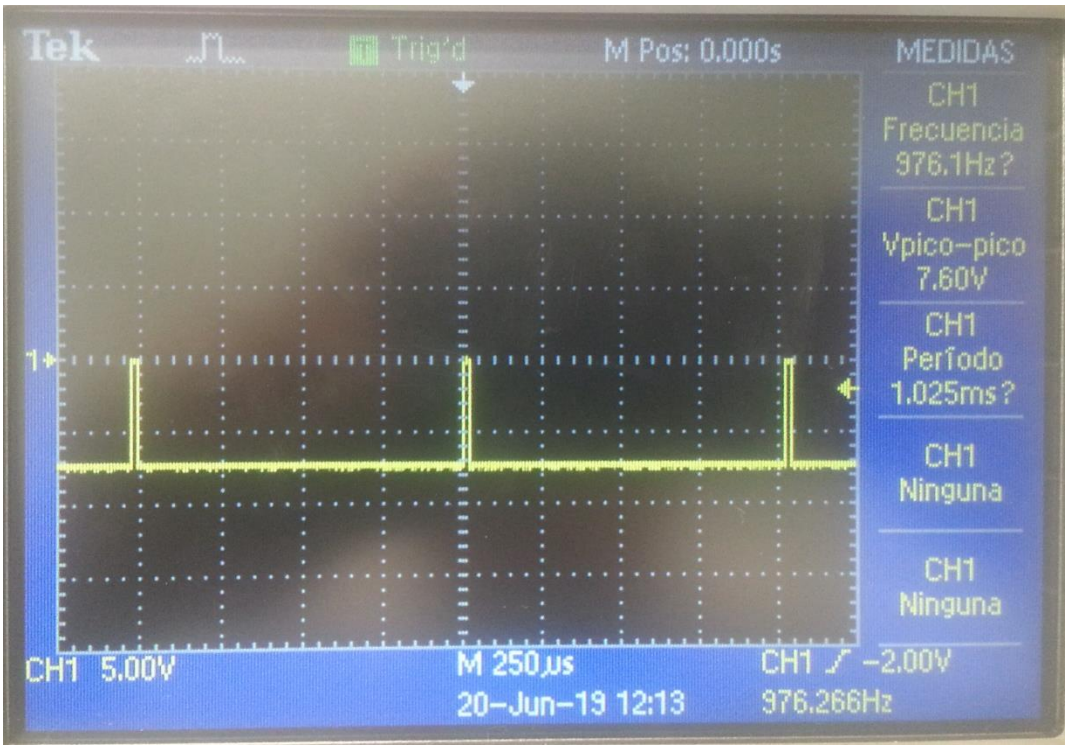

Figura 2.25. Ciclo de trabajo del 98% giro a la izquierda (negativo).

Para poder tener una referencia respecto a las señales obtenidas se colocaron en un generador de señales, una señal cuadrada con un ciclo de trabajo del 95% como se muestra en la figura 2.26 al igual que una señal cuadrada con un ciclo de trabajo del 50% como en la figura 2.27 y finalmente una señal cuadrada con un ciclo de trabajo del 25% como se puede observar en la figura 2,28 de esta forma se pudieron comparar la señales generadas con las señales obtenidas respecto a su ciclo de trabajo.

Figura 2.26. Referencia del generador de señales del 95%.

Figura 2.27. Referencia del generador de señales del 50%.

Figura 2.28. Referencia del generador de señales del 25%.

En la figura 2.29 se muestra el circuito completo para generar todos los ciclos de trabajo remplazando el motor de DC por una resistencia de 1 KΩ para poder observar perfectamente las señales en el osciloscopio

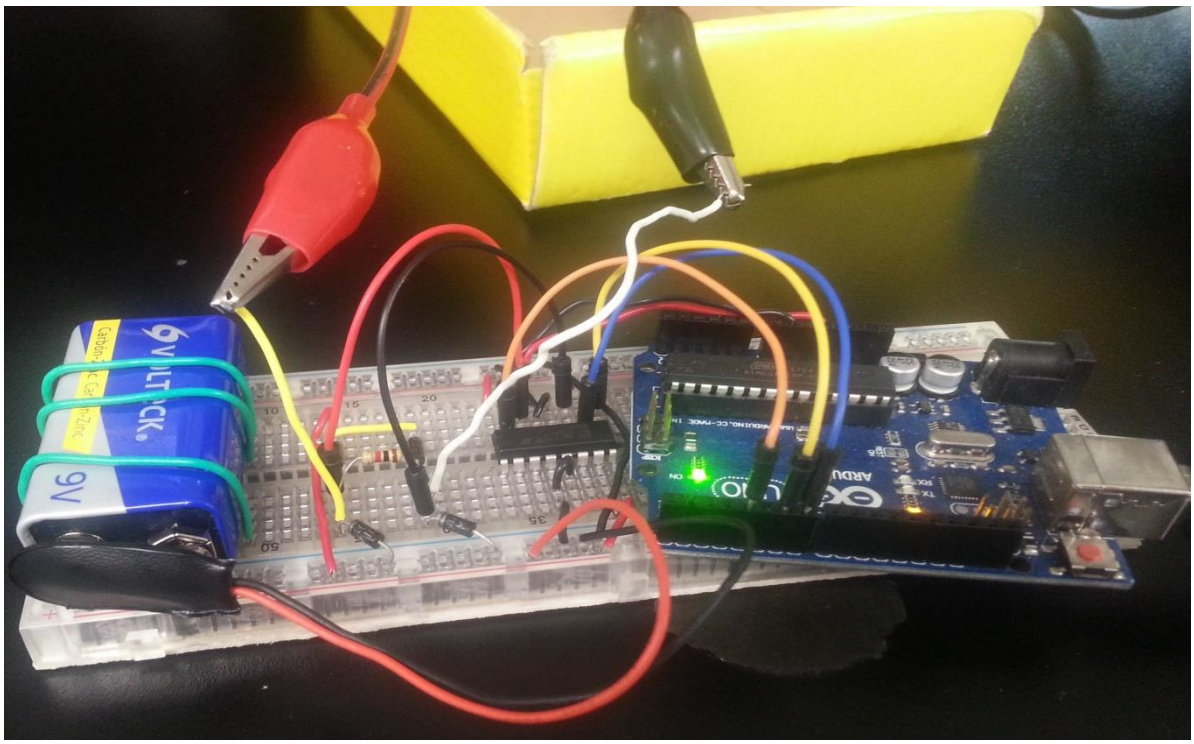

Figura 2.29. Análisis del circuito conectando salida al osciloscopio.

CODIGO DE CONTROL PROPORCIONAL EN ARDUINO.

Para este programa primero se declararon las variables a utilizar, declarando una variable tipo flotante con el nombre de "kp" con un valor de 0.2, una variable tipo flotante con el nombre "cont_p" con un valor de 0, tres variables de tipo entero, una con nombre de "error", que guardará el valor de error, otra variable con el nombre "ref" que contendrá la referencia del sistema que posteriormente, podrá tener diferentes valores, otra variable con el nombre de "y" que representa la posición inicial del carrito, además de tres variables de valor entero con el nombre de "der", que estará conectado al pin 7 del Arduino Uno, otra con el nombre de "izq" que estará conectado al pin 4 del Arduino Uno, otra con el nombre de "pwm" con un valor cero que después tomara otros valores.

```
float kp = 0.2;
float cont_p = 0;
int error;
int ref;
int y = 0;
int der = 7;
int izq = 4;
int pwm = 0;
```

Se declararon dos variables de valor entero, una con el nombre de pin_TRIG destinada a la salida digital 8 del Arduino y otra con el nombre de pin_ECHO destinada a la salida digital 12.

Se declararon otras tres variables, dos de tipo entero (tiempoSenal y distanciaCM) y una de tipo flotante con el nombre de distanciaM.

```
int pin_TRIG = 8; //Pin para enviar pulsos
int pin_ECHO = 12; //Pin para recibir la señal de rebote

int tiempoSenal;
int distanciaCM;
float distanciaM;
```

En el menú de void setup se inicializan las variables pin_TRIG y pin_ECHO una como salida y la otra como entrada respectivamente y Serial.begin (9600) abre el puerto serie con la finalidad de poder observar los valores de distancia en el monitor serial del Arduino además de declarar como salidas las variables "der" e "izq".

```
void setup() {
  Serial.begin(9600);
```

```
  pinMode(pin_TRIG, OUTPUT);
  pinMode(pin_ECHO, INPUT);
  pinMode(der, OUTPUT);
  pinMode(izq, OUTPUT);
}
```

En el menú de void loop primero se estabiliza el sensor con un nivel bajo (pin_TRIG, LOW) y con un delay de 5 microsegundos a la variable pin_TRIG y enseguida se envía un pulso para iniciar el sensor, en este caso se mandó a un nivel alto (pin_TRIG, HIGH) con un delay de 5 microsegundos, luego en tiempoSenal se guarda el tiempo que dura el pin_ECHO en HIGH, para que finalmente se realice la siguiente conversión "*((0.034 * tiempoSenal) / 2)-5"* y el resultado de la conversión nos dará como resultado la distancia en centímetros guardando ese valor en la variable distanciaCM.

```
void loop() {
  digitalWrite(pin_TRIG, LOW);
  delayMicroseconds(5); //Estabiliza el sensor

  digitalWrite(pin_TRIG, HIGH); //Envia el pulso para iniciar el sensor
  delayMicroseconds(10);

  tiempoSenal = pulseIn(pin_ECHO, HIGH); //Tiempo que dura el pin ECHO en HIGH
  distanciaCM = ((0.034 * tiempoSenal) / 2)-5; //Convierte el tiempo en distancia (cm)

  Serial.println(distanciaCM);
```

Después se colocó la variable "ref" es decir la referencia a una distancia de 1 cm, colocando en error la operación de "ref – distanciaCM" esto con la finalidad de saber el valor del error, enseguida con la variable "cont_p" se almacena el valor obtenido de la operación multiplicando la variable "error" por "kp" con un valor absoluto para no afectar el mapeo, enseguida en la variable "pwm" se realizó un mapeo para poder almacenar un valor de señal PWM desde 0 hasta 255 respecto de una distancia de 30 cm como máximo y una mínima de 0 cm.
Y los valores del motor se colocaron a niveles bajos "der e izq".

```
  ref = 1;
  error = ref - distanciaCM ;
  cont_p = abs(error * kp);
  pwm = map(cont_p, 0, 30, 0, 255);
  digitalWrite(der, LOW);
  digitalWrite(izq, LOW);
```

Se colocaron tres condiciones distintas empezando la primera con una condición en la que si el error es mayor a cero el motor se mueva a la derecha colocando la variable "der" a un nivel alto y la variable "izq" a un nivel bajo y

con una señal PWM conectada a la salida 6 del Arduino Uno con un cierto valor generado del mapeo, la segunda con una condición en la que si el error es menor a cero el motor se mueva a la izquierda colocando la variable "der" a un nivel bajo y la variable "izq" a un nivel alto y con una señal PWM conectada a la salida 6 del Arduino Uno con un cierto valor generado del mapeo y finalmente como ultima condición en caso de que el valor del error sea cero el motor no se moverá a ningún lado esto colocando los valores de "der e izq" a un nivel bajo.

```
  if (error > 0) {
    digitalWrite(der, HIGH);
    digitalWrite(izq, LOW);
    analogWrite(pwm, 6);
  }

  else if (error < 0) {
    digitalWrite(der, LOW);
    digitalWrite(izq, HIGH);
    analogWrite(pwm, 6);
  }
  else {
    digitalWrite(der, LOW);
    digitalWrite(izq, LOW);
    analogWrite(pwm, 6);
  }
}
```

INSTALACION DE LIBRERÍA ARDUINO EN MATLAB 2018B.

A continuación, se muestra la instalación de los paquetes de Arduino en Matlab 2018b los cuales son muy importantes para poder conectar nuestra tarjeta Arduino Uno a Matlab de lo contrario no se podrá conectar Arduino Uno.

Primero se abrirá el programa Matlab 2018b que desplegará la ventana principal como se observa en la figura 2.30.

A continuación, se debe pinchar en la pestaña "Add-Ons" como se muestra en la figura 2.31 y desplegará su menú, luego seleccionamos la opción "Get Hardware Support Packages".

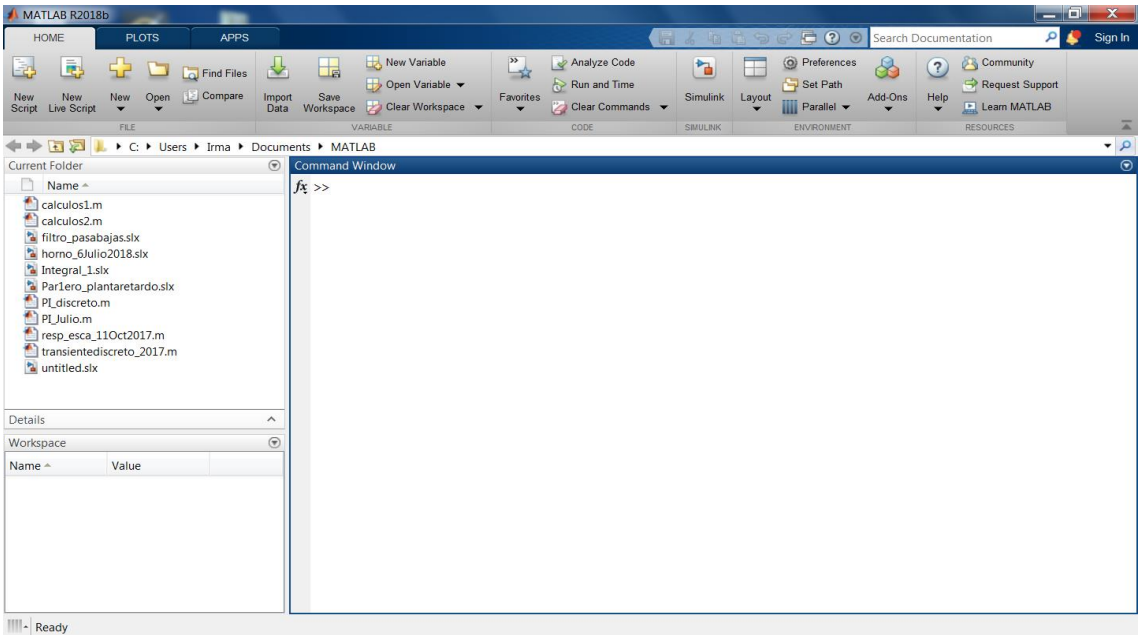

Figura 2.30. Ventana principal de Matlab 2018b.

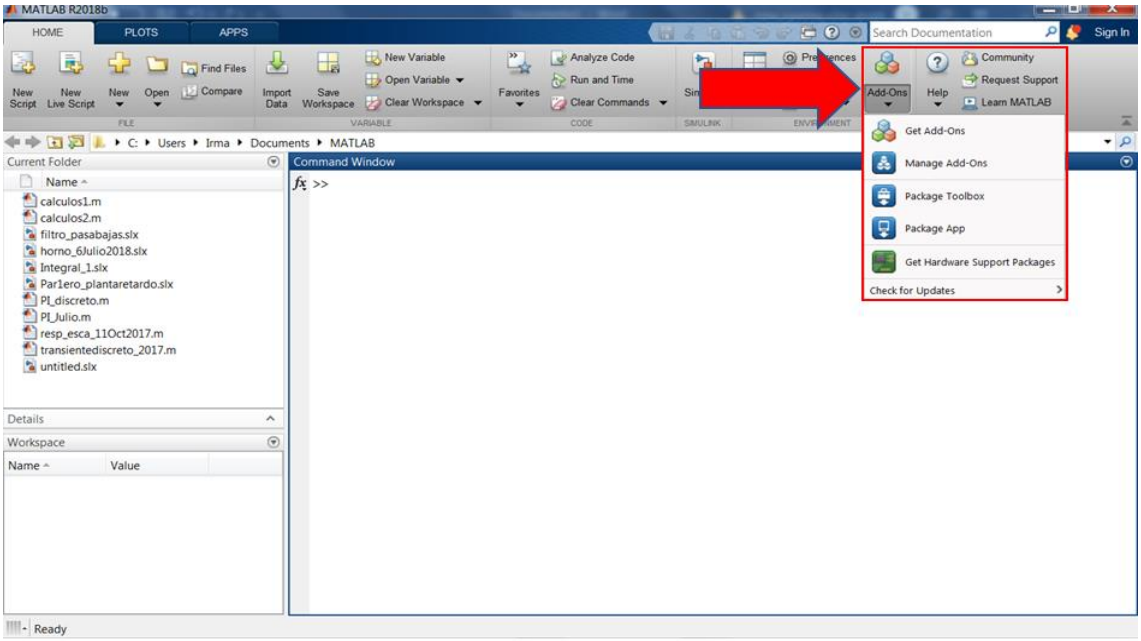

Figura 2.31. Menú de "Add Ons".

Ahora se nos muestra la ventana de "Add Ons Explorer" como se muestra en la figura 2.32 y como se puede observar los paquetes de instalación de Arduino no vienen preinstalados en Matlab por lo que tendremos que llevar a cabo la instalación de cada uno de ellos.

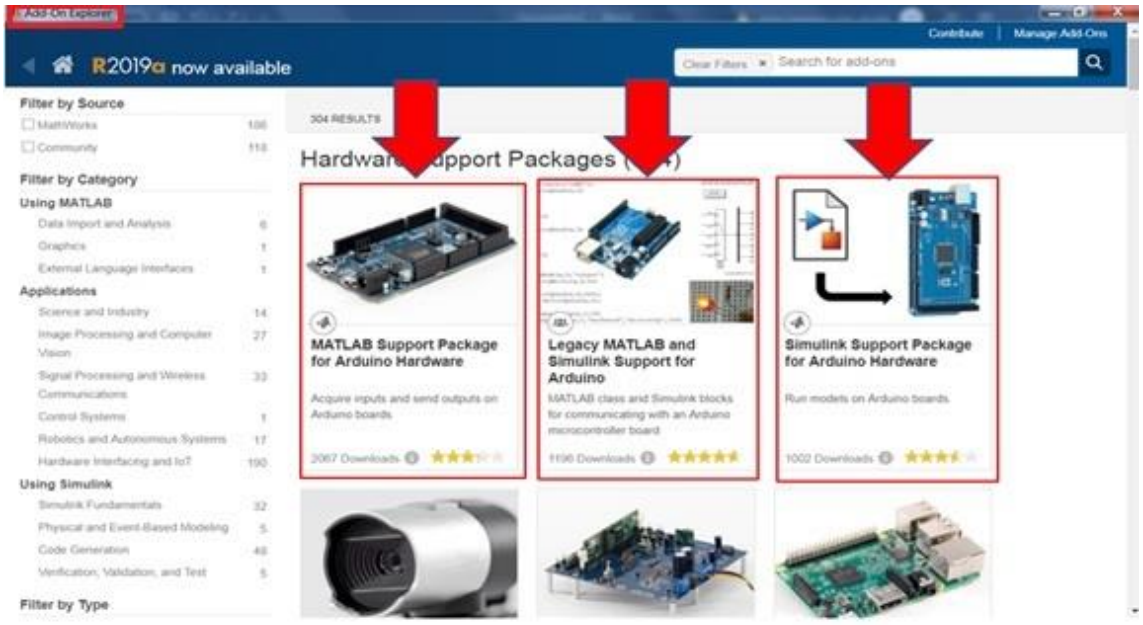

Figura 2.32. Ventana principal de "Add Ons Explorer".

Pinchamos ahora sobre "MATLAB Support Package for Arduino Hardware" lo cual nos desplegara otra ventana como se muestra en la figura 2.33, a continuación, pinchamos la opción que dice "Install" y nos mostrará dos opciones (Install y Download Only) como se observa en la figura 2.34 y seleccionaremos la opción "Install".

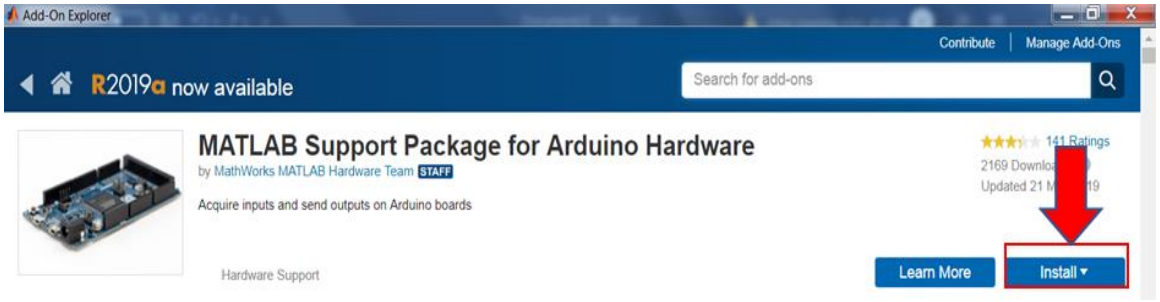

Figura 2.33. Ventana de "Support Package for Arduino Hardware".

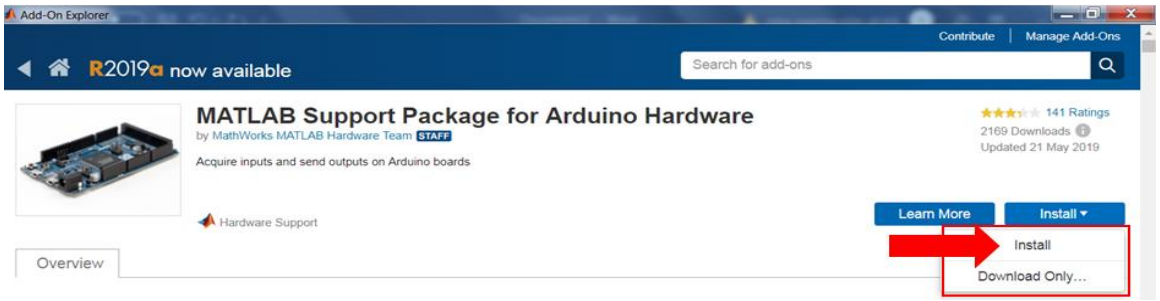

Figura 2.34. Menú de "Install".

Se nos mostrará la ventana de registro e inicio de sesión como se muestra en la figura 2.35, en caso de no contar con una cuenta se tendrá que pinchar sobre crear cuenta en Matlab, mientras que en el caso contrario en el que ya contamos con una cuenta, solo es cuestión de ingresar nuestra dirección de correo electrónico y contraseña. Es de suma importancia llevar al cabo cualquiera de los dos pasos ya que de lo contrario Matlab no permitirá la instalación de ningún paquete.

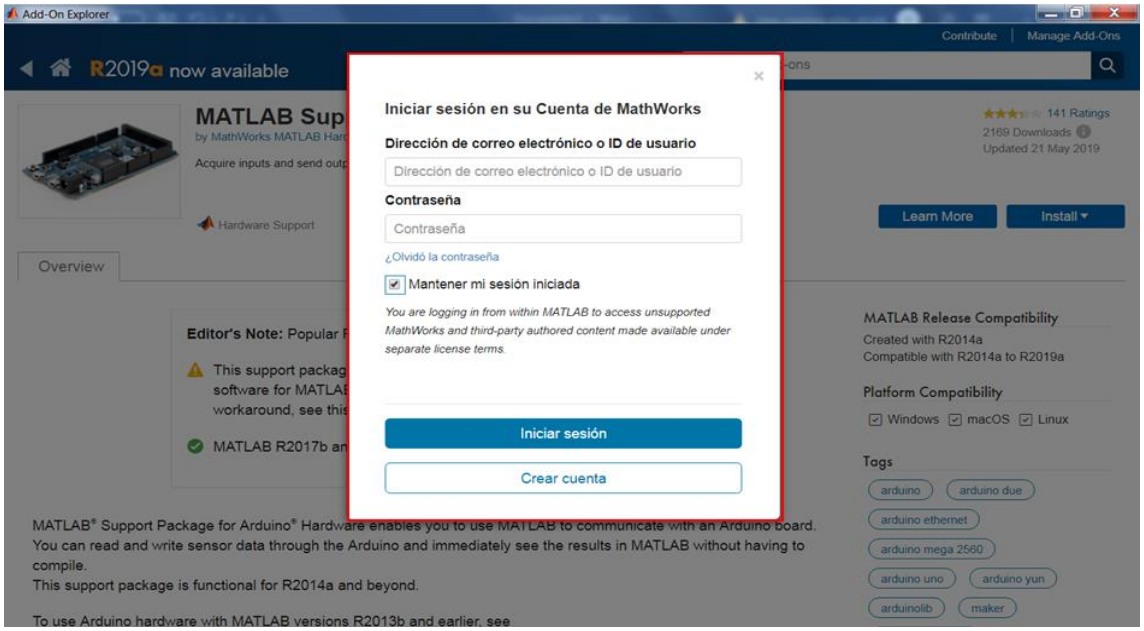

Figura 2.35. Ventana de registro e inicio de sesión.

A continuación, se nos muestra la ventana instalación como se muestra en la figura 2.36 en ella solo tendremos que pinchar en "Next" pasa seguir con la instalación.

Después se desplegará otra ventana, en la cual podremos ver que los paquetes se están descargando e instalando como se muestra en la figura 2.37, ahora solo es cuestión de esperar unos cuantos minutos en lo que termina la instalación.

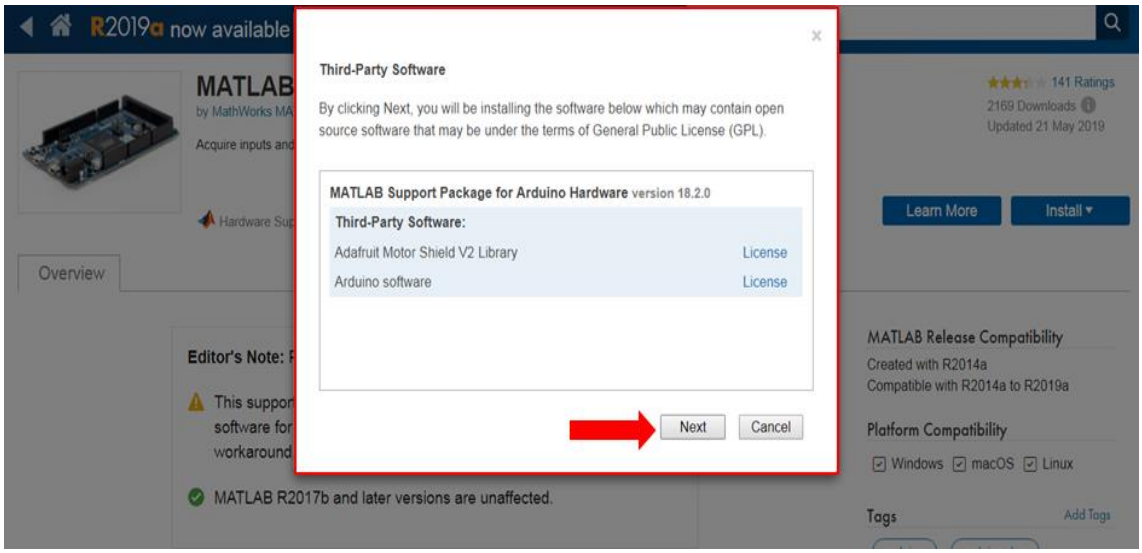

Figura 2.36. Ventana de instalación.

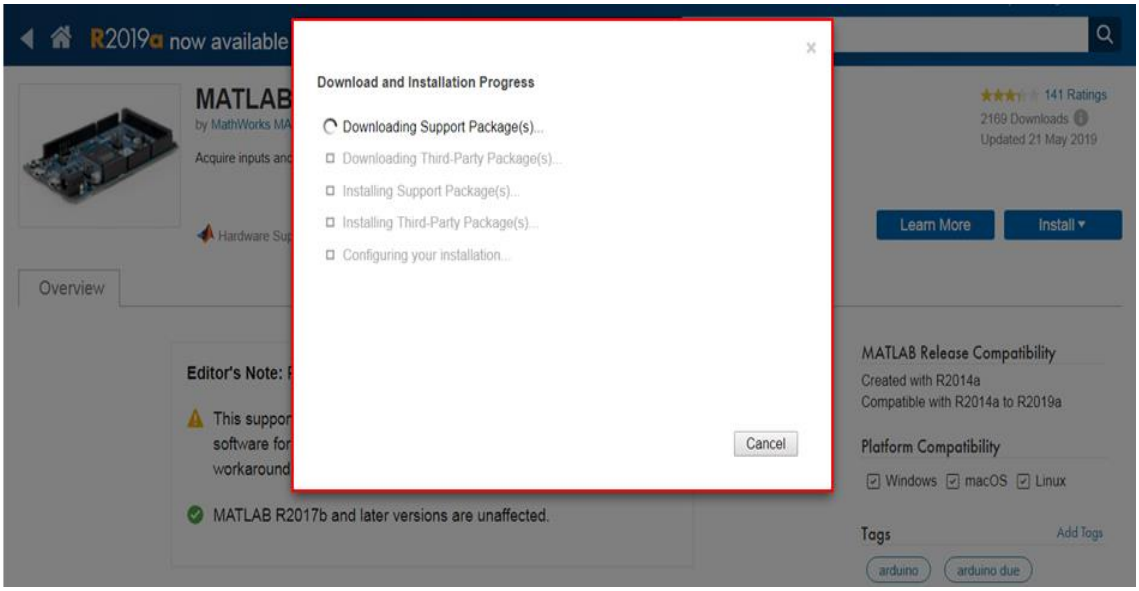

Figura 2.37. Completando instalación del paquete.

Al terminar el proceso, nos regresará nuevamente a la ventana "MATLAB Support Package for Arduino Hardware" como se muestra en la figura 2.38 y como se puede apreciar ya no permite instalar el paquete, ya que este se encuentra instalado en nuestro Matlab, vea la figura 2.39. Aún faltan otros dos paquetes por instalar, proseguiremos a instalar el paquete "Legacy Matlab and Simulink Support for Arduino".

Figura 2.38. Ventana de "Support Package for Arduino Hardware" 2.

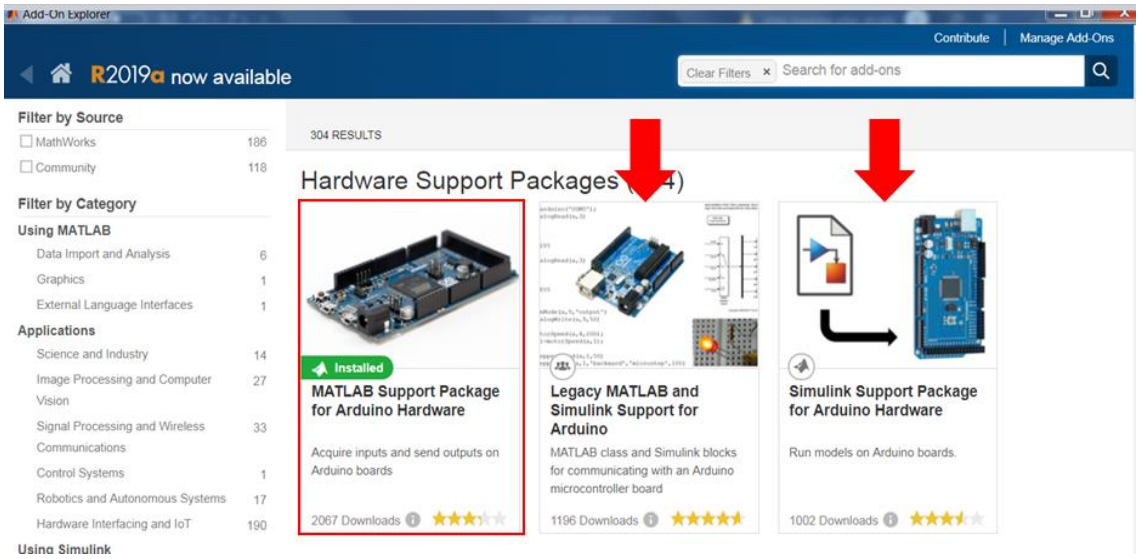

Figura 2.39. Ventana principal de "Add Ons Explorer" 2.

Se desplegará la ventana de "Legacy Matlab and Simulink Support for Arduino" como se muestra en la figura 2.40 y seleccionamos el cuadro azul "Add" que mostrará dos opciones distintas (Add to MATLAB y Download Only) y seleccionaremos "Add to MATLAB" para continuar con la instalación como se observa en la figura 2.41.

Figura 2.40. Ventana de "Legacy Matlab and Simulink Support for Arduino".

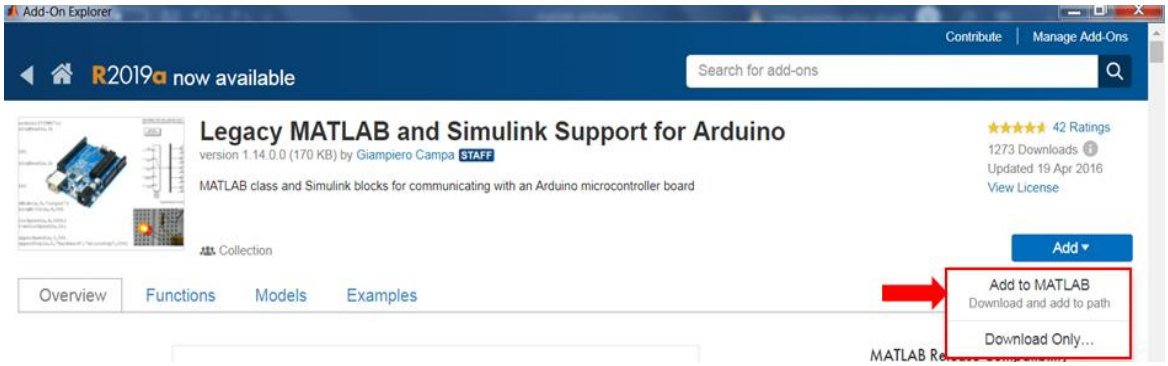

Figura 2.41. Menú de "Add".

Se mostrará la ventana de licencia "Legacy Matlab and Simulink" como se muestra en la figura 2.42, seleccionaremos la opción "Agree", después nos regresará a la ventana "Legacy Matlab and Simulink Support for Arduino" como se muestra en la figura 2.43 y podemos observar que se instaló el paquete con éxito.

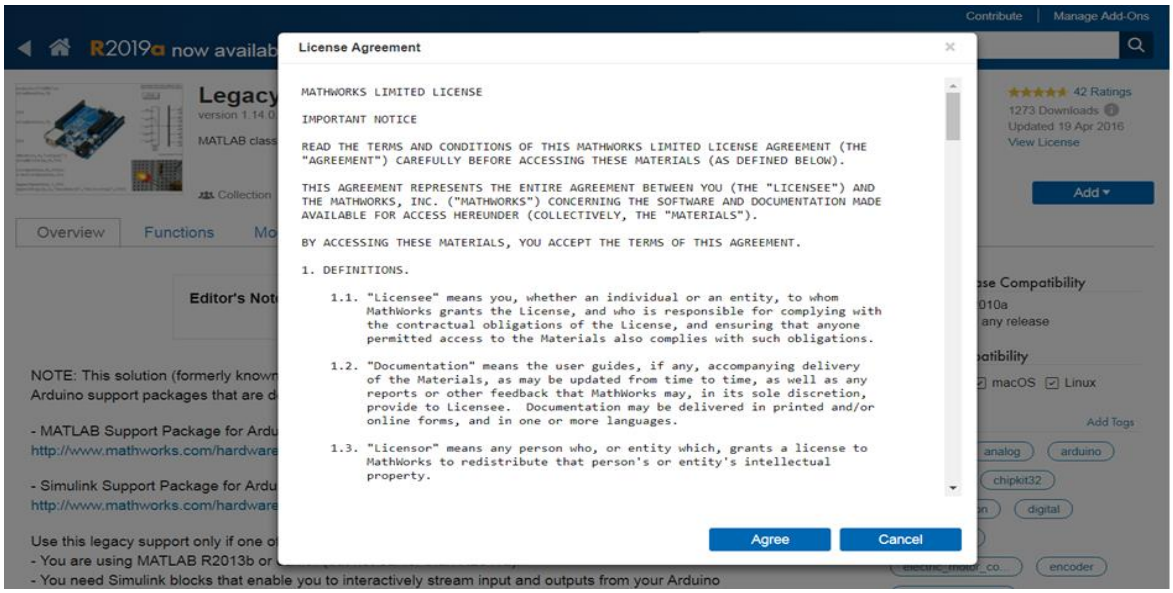

Figura 2.42. Ventana de licencia "Legacy Matlab and Simulink".

Figura 2.43. Ventana de "Legacy Matlab and Simulink Support for Arduino" 2.

Por último regresamos a la ventana de "Add Ons Explorer" como se muestra en la figura 2.44 y como se puede observar, ya solo falta por instalar el paquete de

Arduino "Simulink Support Package for Arduino Hardware" y lo seleccionamos, el cual nos mostrará su ventana de instalación como se muestra en la figura 2.45, seleccionaremos la opción "Install" que nos mostrará dos opciones (Install y Download Only) y seleccionaremos "Install", a continuación se nos mostrará la ventana de licencia "Simulink Support Package" y seleccionaremos la opción "I Accept" como se muestra en la figura 2.46.

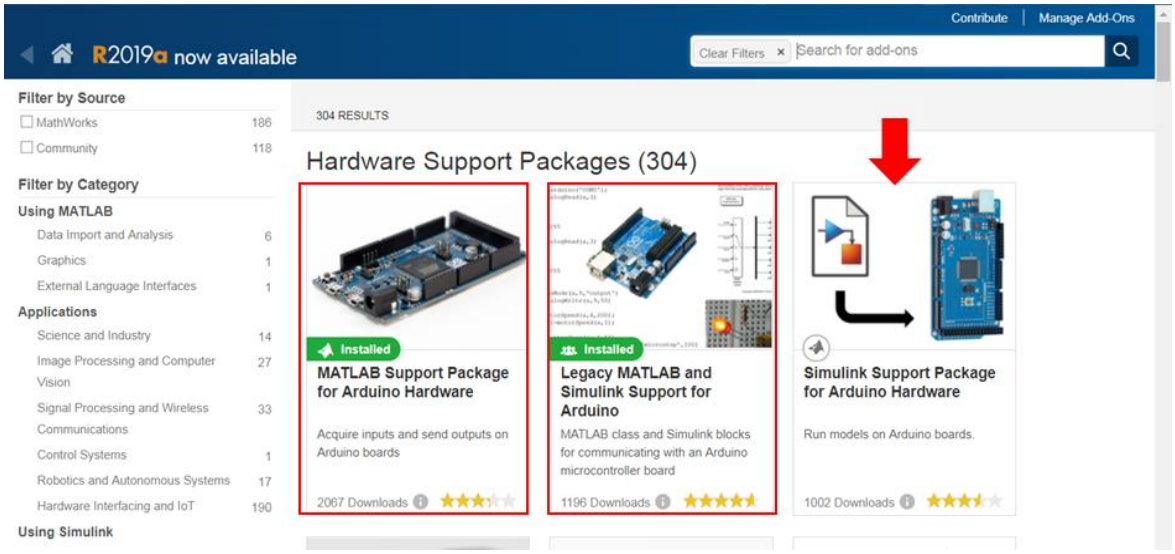

Figura 2.44. Ventana principal de "Add Ons Explorer" 3.

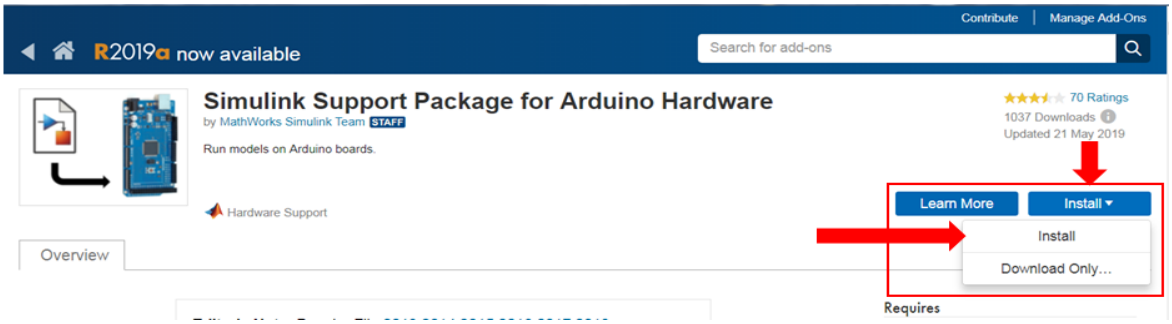

Figura 2.45. Ventana de "Simulink Support Package for Arduino Hardware".

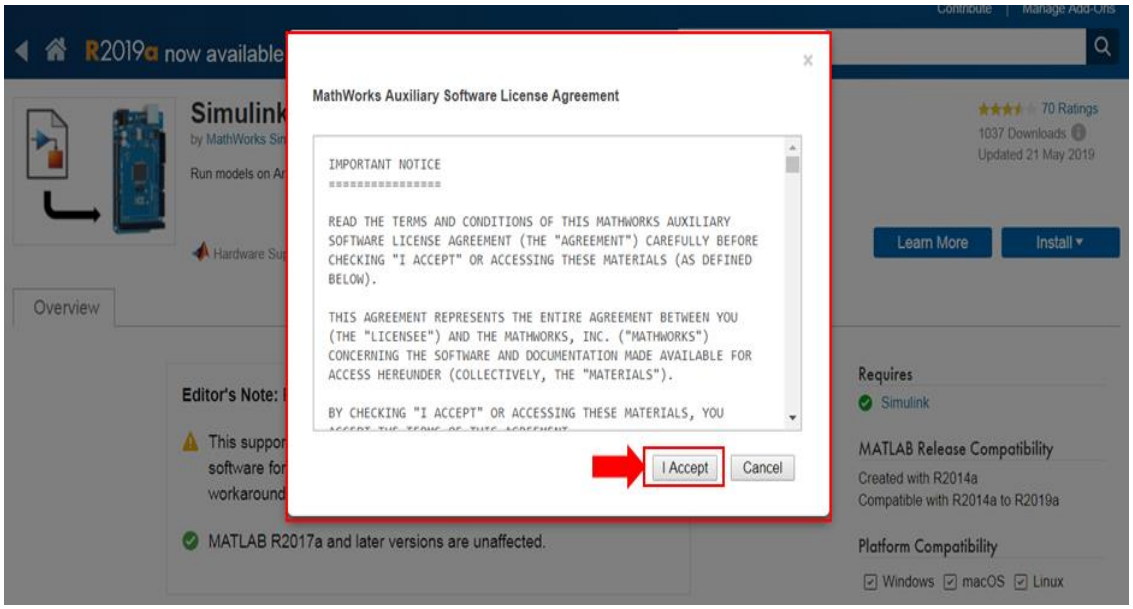

Figura 2.46. Ventana de licencia "Simulink Support Package".

Se mostrará la ventana de instalación como se muestra en la figura 2.47 y solo bastará con pinchar en "Next" para seguir con la instalación, después solo aparecerá una ventana avisando que la instalación se completó, como se muestra en la figura 2.48 y solo seleccionaremos "Setup Later", podemos observar en la figura 2.48 que todos los paquetes de Arduino para Matlab ya están instalados.

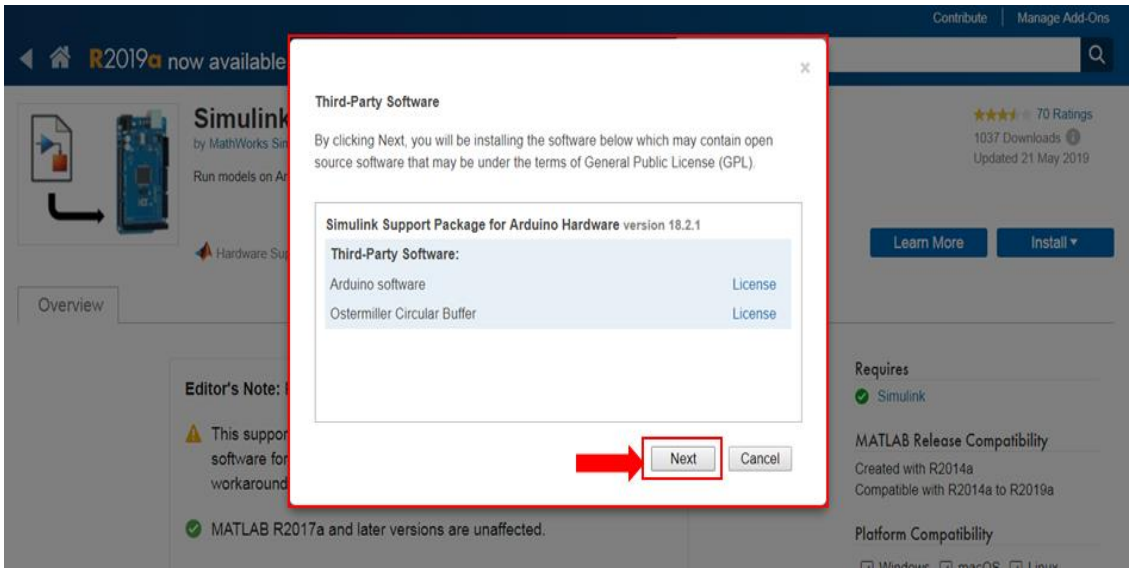

Figura 2.47. Ventana de instalación.

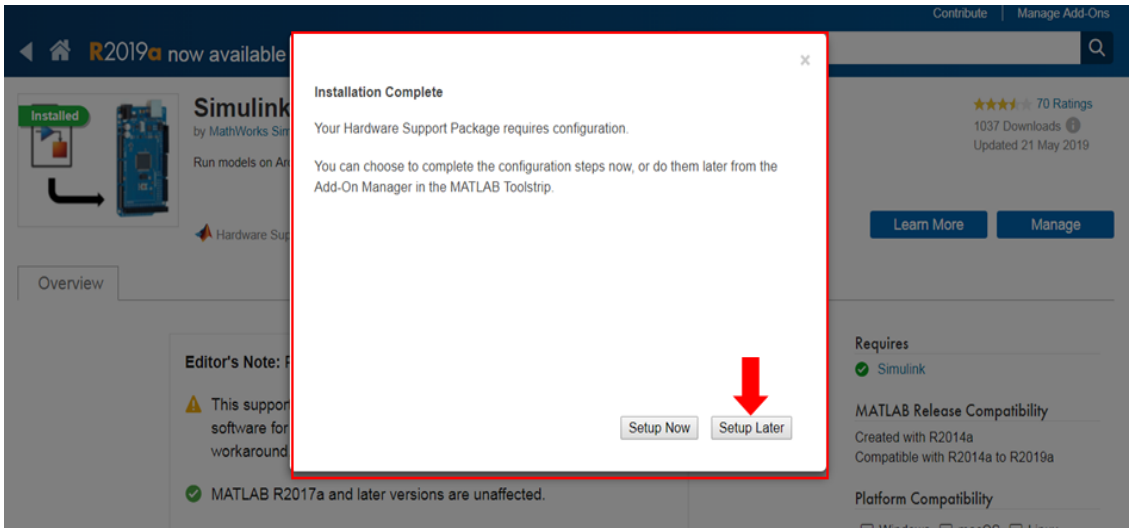

Figura 2.48. Ventana de instalación finalizada.

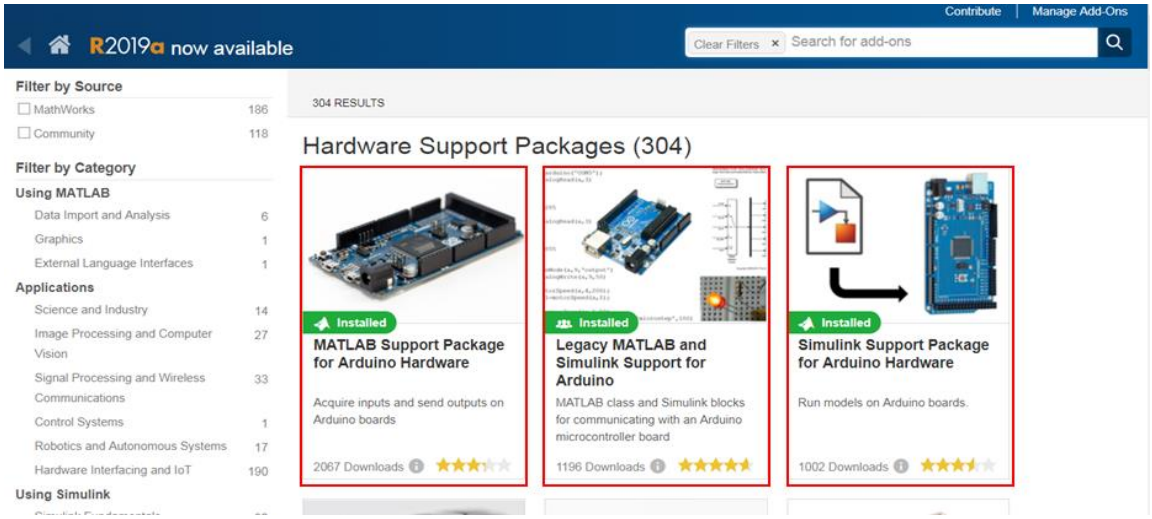

Figura 2.49. Ventana principal de "Add Ons Explorer" 4.

CONFIGURACIÓN DE ARDUINO EN MATLAB.

Cabe mencionar que no solo basta con la instalación de Arduino en Matlab para poder trabajar, también se tienen que configurar ciertos parámetros en Matlab, para ello nos colocaremos en la ventana principal de Matlab y seleccionaremos el recuadro que lleva por nombre "Simulink" como se muestra en la figura 2.50.

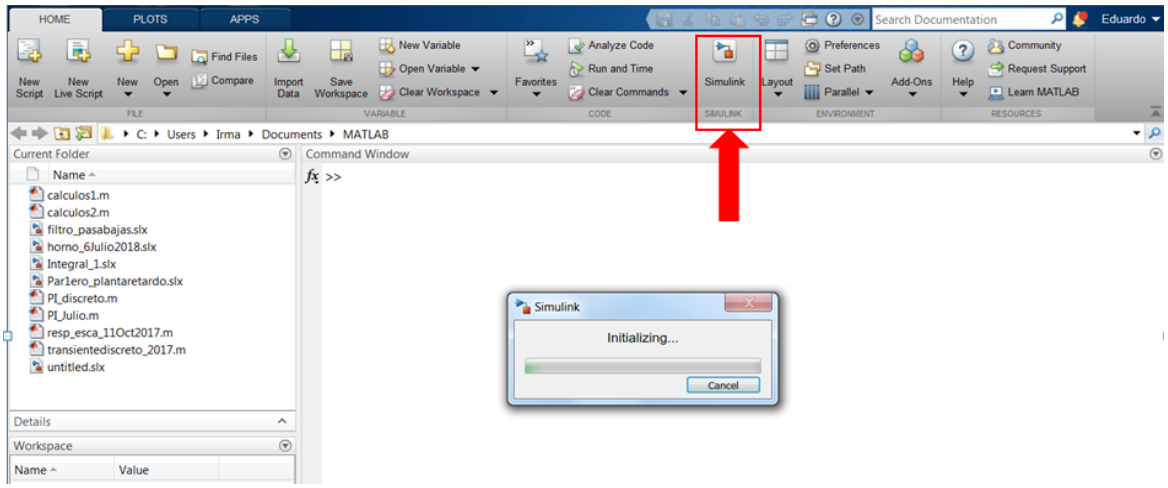

Figura 2.50. Venta principal de Matlab (Simulink).

Se desplegará la ventana "Simulink Start Page" y solo bastará con seleccionar "Blank Model" como se muestra en la figura 2.51 y se nos abrirá un modelo nuevo de Simulink como se muestra en la figura 2.52.

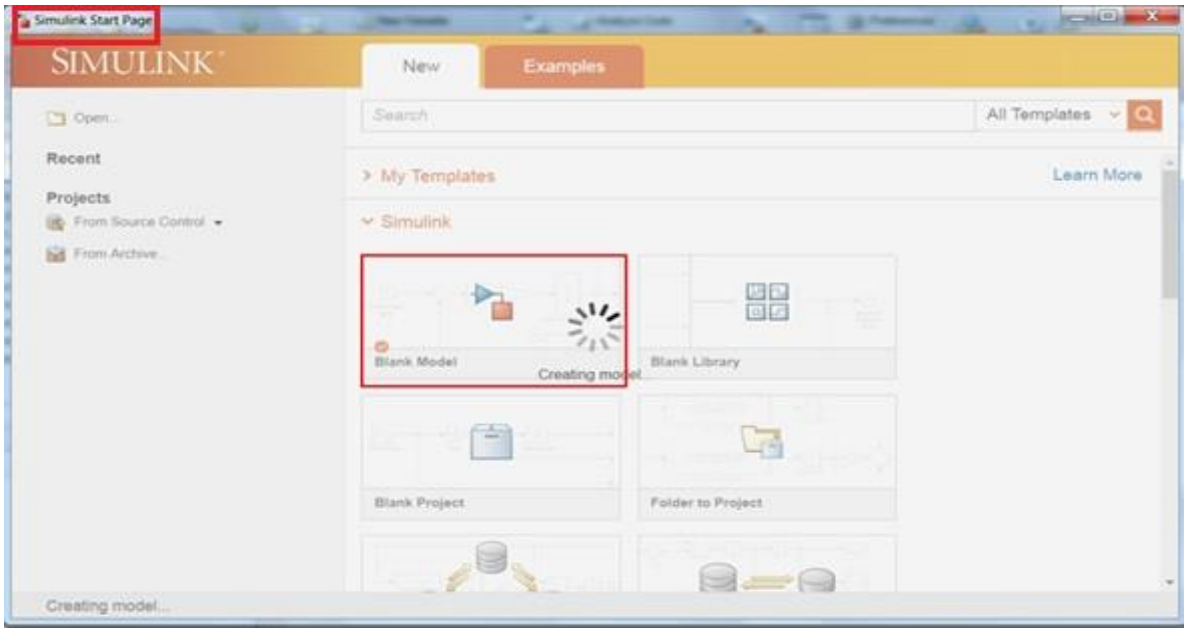

Figura 2.51. Ventana de "Simulink Start Page".

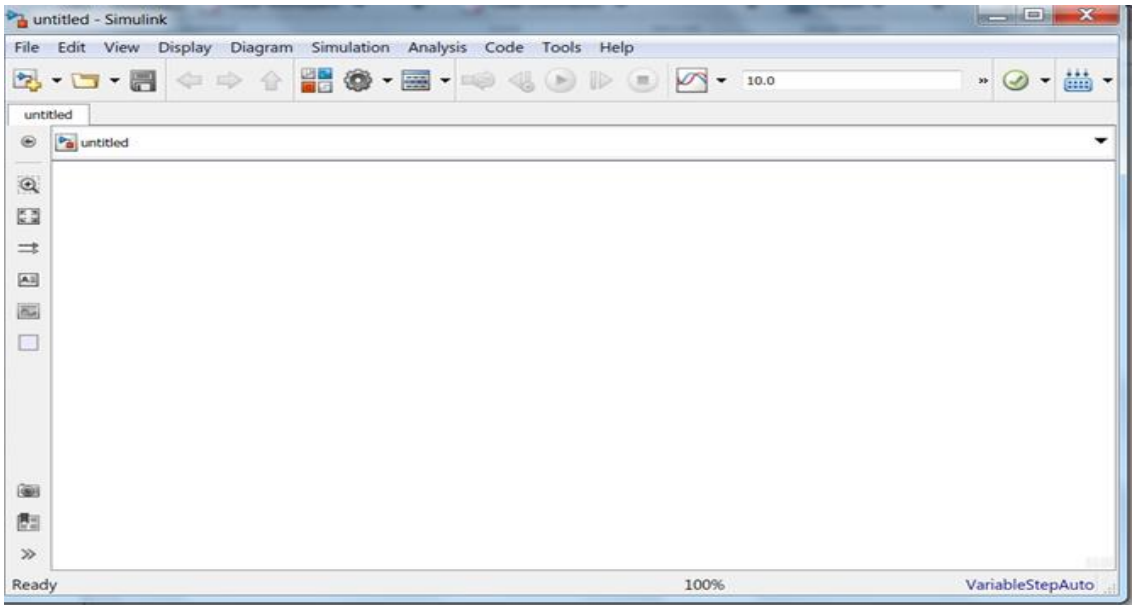

Figura 2.52. Ventana de Simulink.

A continuación, seleccionaremos la pestaña "View" y después "Library Browser" como se muestra en la figura 2.53, esto con la finalidad de poder verificar que los paquetes de Arduino estén dentro de nuestra librería, buscaremos una librería con el nombre de "Simulink Support Package for Arduino Hardware", como se muestra en la figura 2.54 y con esto comprobamos que contamos con la librería de Arduino en nuestro Matlab.

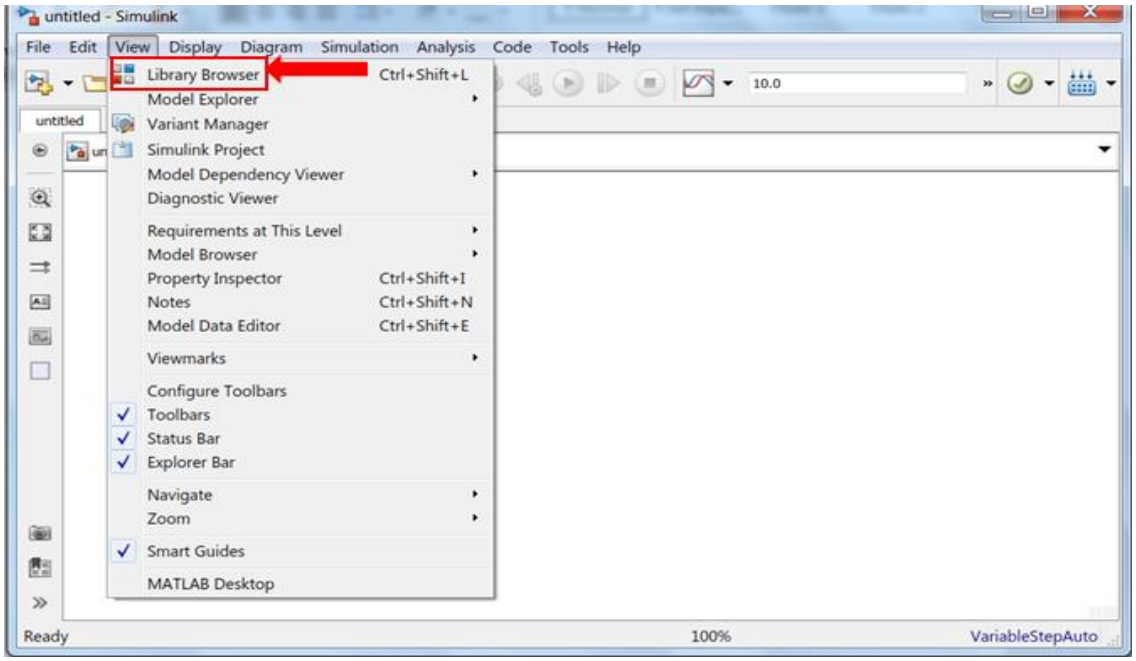

Figura 2.53. Sub Ventana "View (Library Browser)".

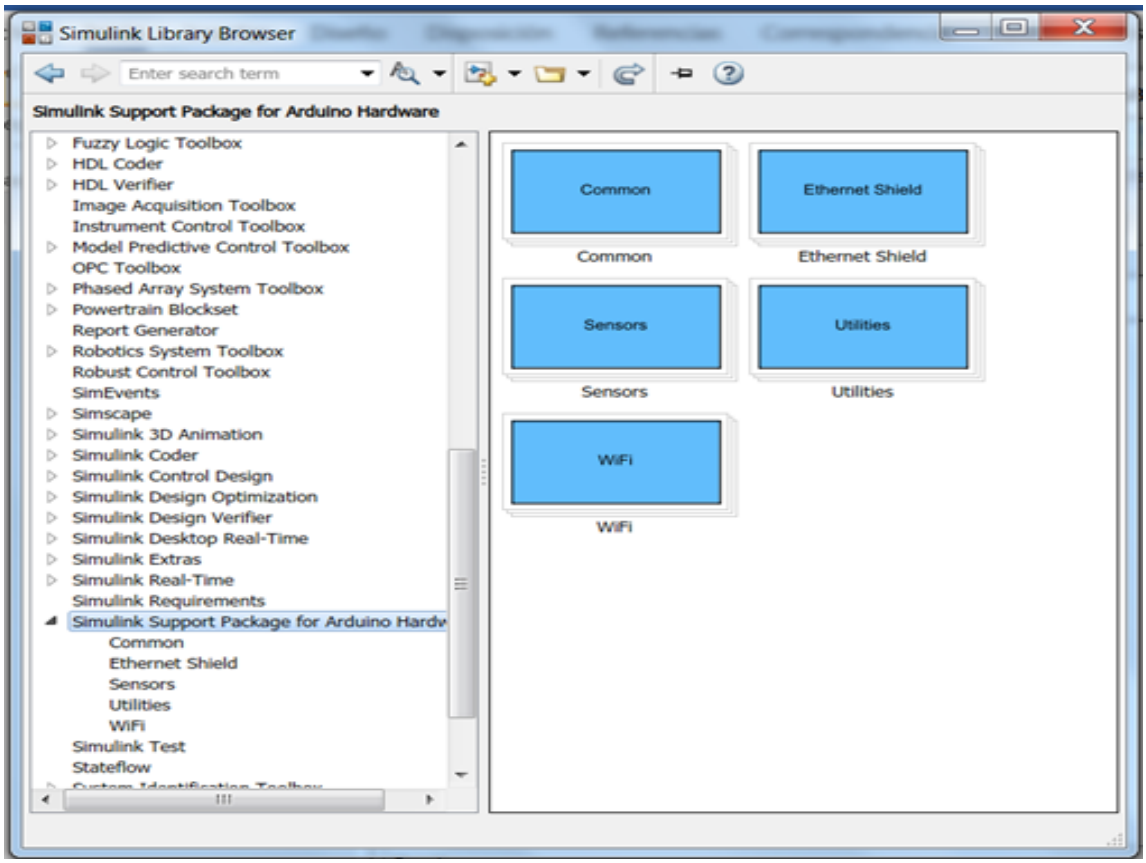

Figura 2.54. Ventana de "Simulink Library Browser.

En la figura 2.55 se pueden observar tanto salidas digitales como analógicas, de igual forma entradas digitales y entradas analógicas entre otras, como es el caso de una señal PWM. En la figura 2.56 se muestran los sensores con los que se puede trabajar, pero para este caso solo será suficiente con el sensor ultrasónico.

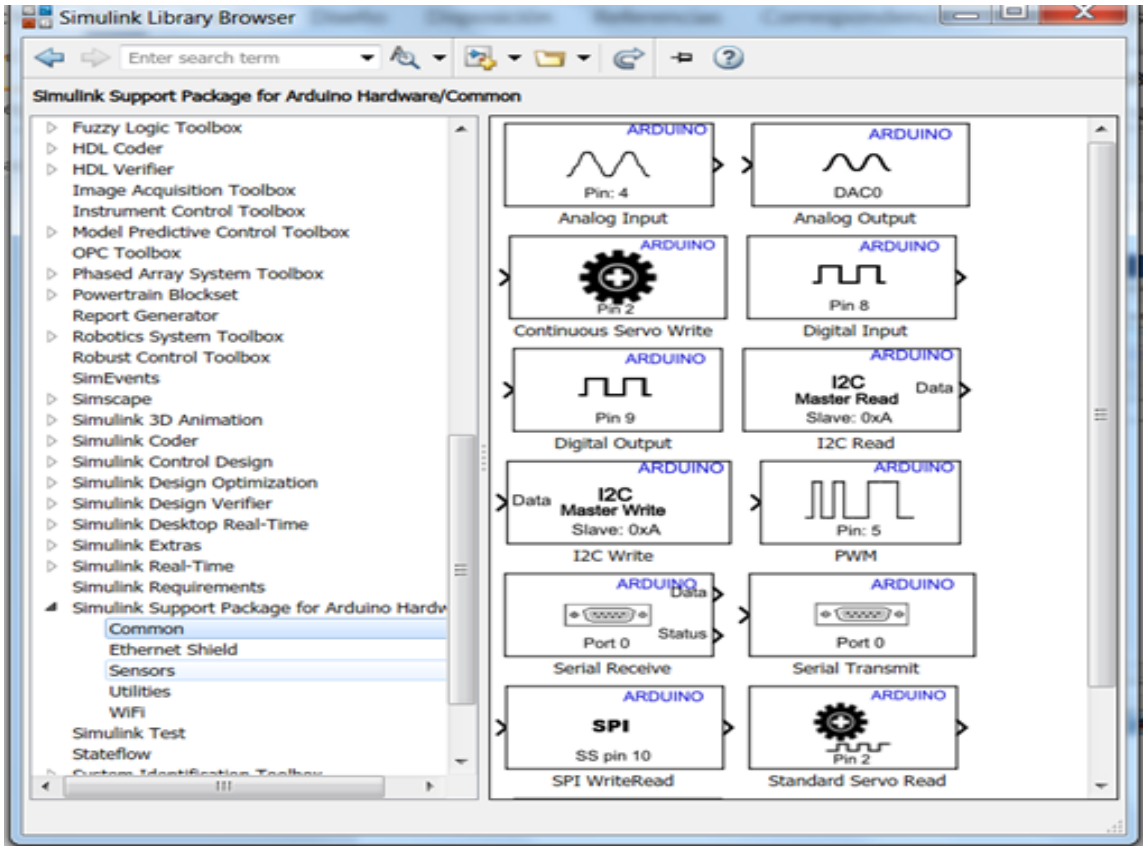

Figura 2.55. Bloques de Arduino "Common".

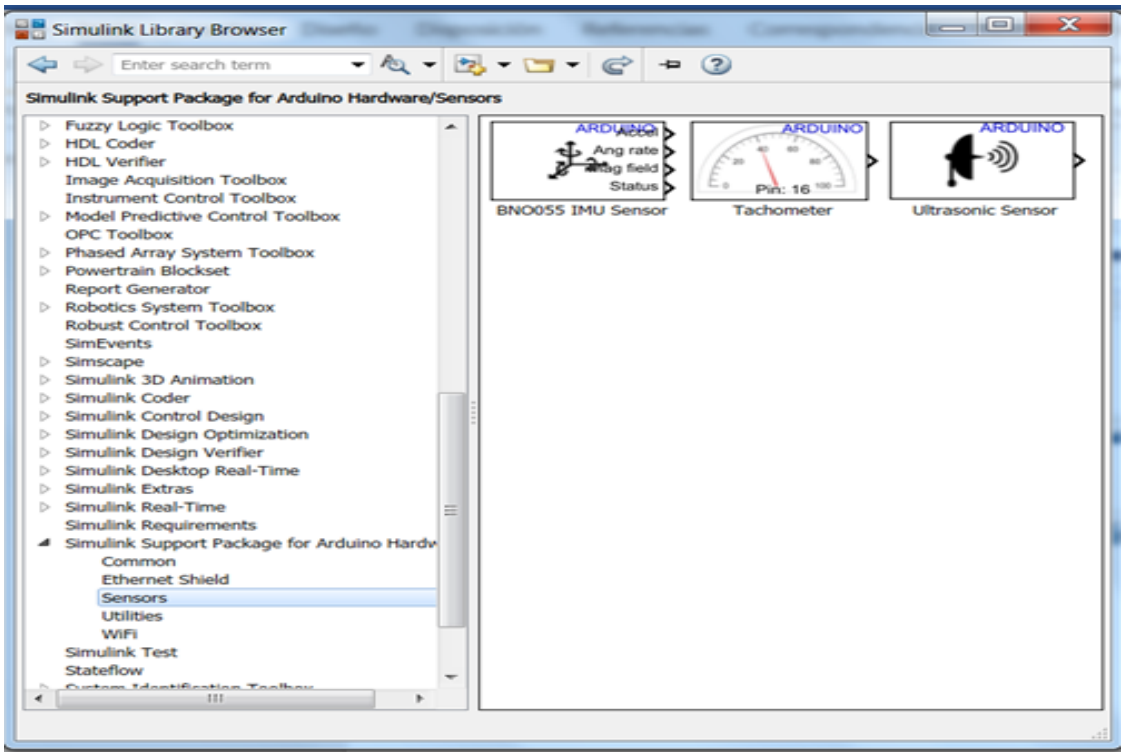

Figura 2.56. Bloques de Arduino "Sensors".

Regresando a la ventana de Simulink como se muestra en la figura 2.57 escogeremos la opción "External" ya que se trabajará con un dispositivo externo

en este caso el Arduino Uno. A continuación, se seleccionará la pestaña "Tools" después "Run on Target Hardware" y "Options" como se muestra en la figura 2.58.

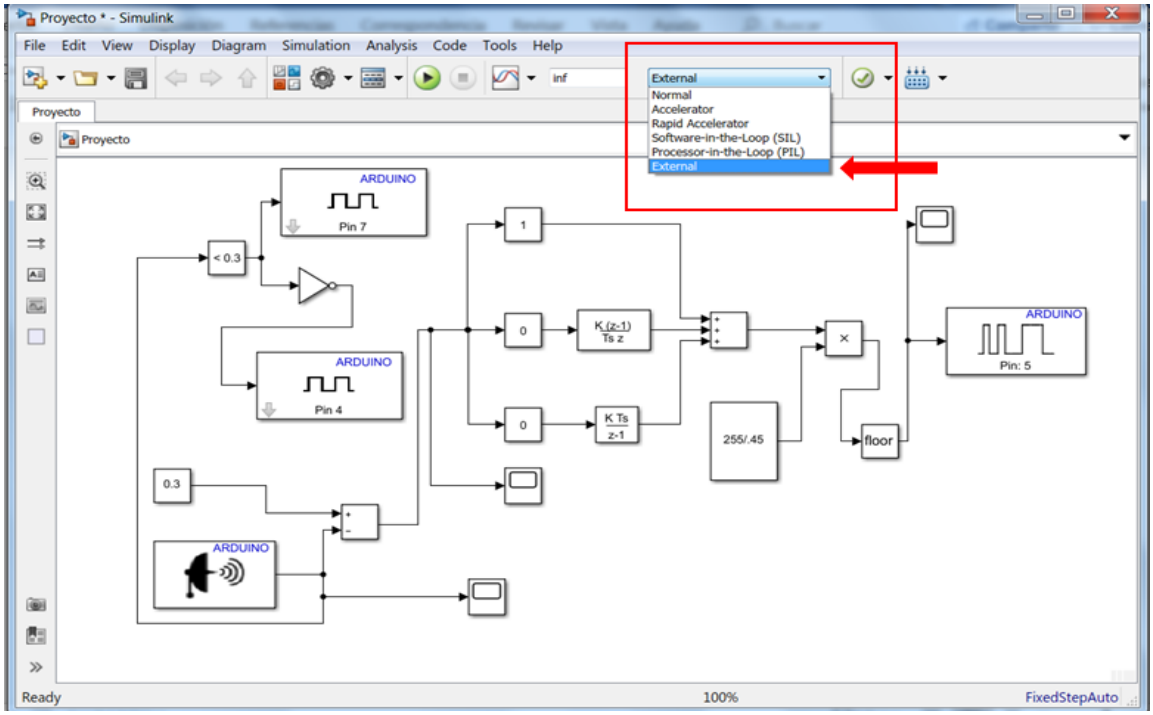

Figura 2.57. Ventana de Simulink (External).

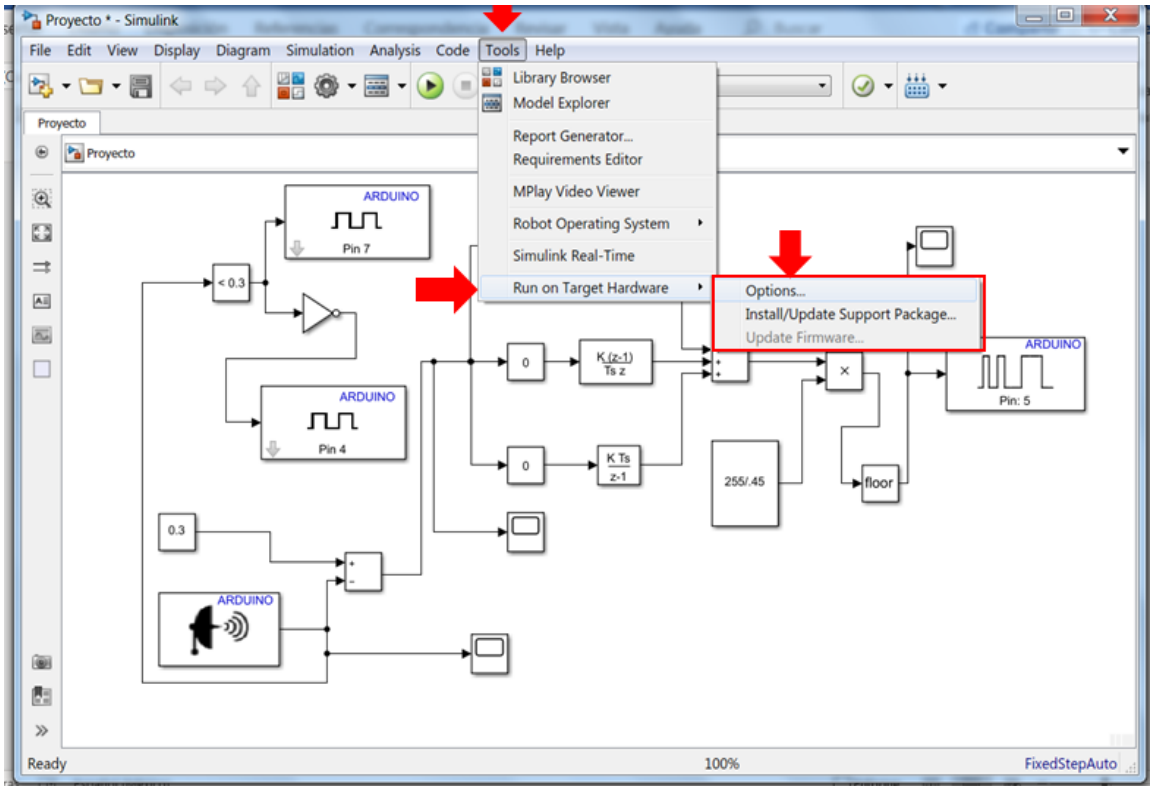

Figura 2.58. Ventana de Simulink (Tools-Run on Target Hardware-Options).

Una vez en la ventana de "Configuration Parameters", seleccionaremos la pestaña "Hardware Implementation" y en el apartado de Hardware board seleccionaremos el nombre de nuestra placa a trabajar, en este caso seleccionamos Arduino Uno como se muestra en la figura 2.59

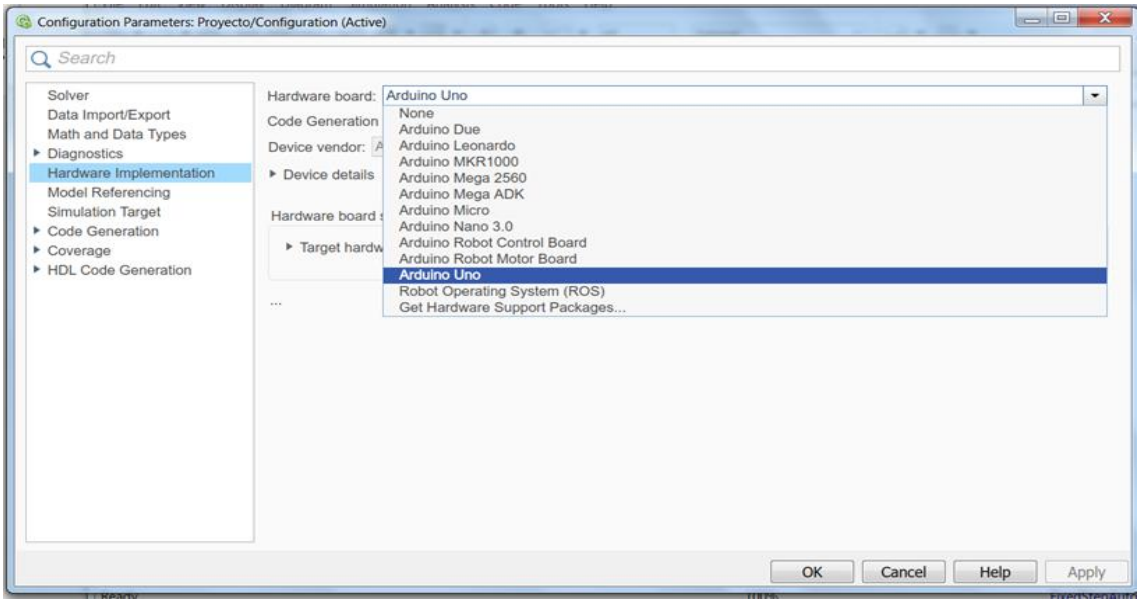

Figura 2.59. Ventana de "Configuration Parameters".

Ahora en "Groups" seleccionaremos el apartado de nombre "Host-board connection" como se muestra en la figura 2.60 y seleccionaremos en "Host COM Port" el modo "Manually" y colocaremos en "COM Port Number" el número de puerto que registra nuestra computadora para el Arduino Uno (para saber que numero de puerto asignó la computadora accederemos al icono de Windows, seleccionaremos panel de control, en el cual se encontraran muchas opciones, seleccionaremos la que lleva el nombre de administrador de dispositivos y se desplegará una lista de todos los dispositivos de nuestra computadora, seleccionaremos Puertos "COM y LPT" y en ella estará nuestra tarjeta Arduino Uno, con la leyenda COM y un cierto número de puerto en este caso el 3), se colocó el puerto número 3 como se muestra en la figura 2.61.

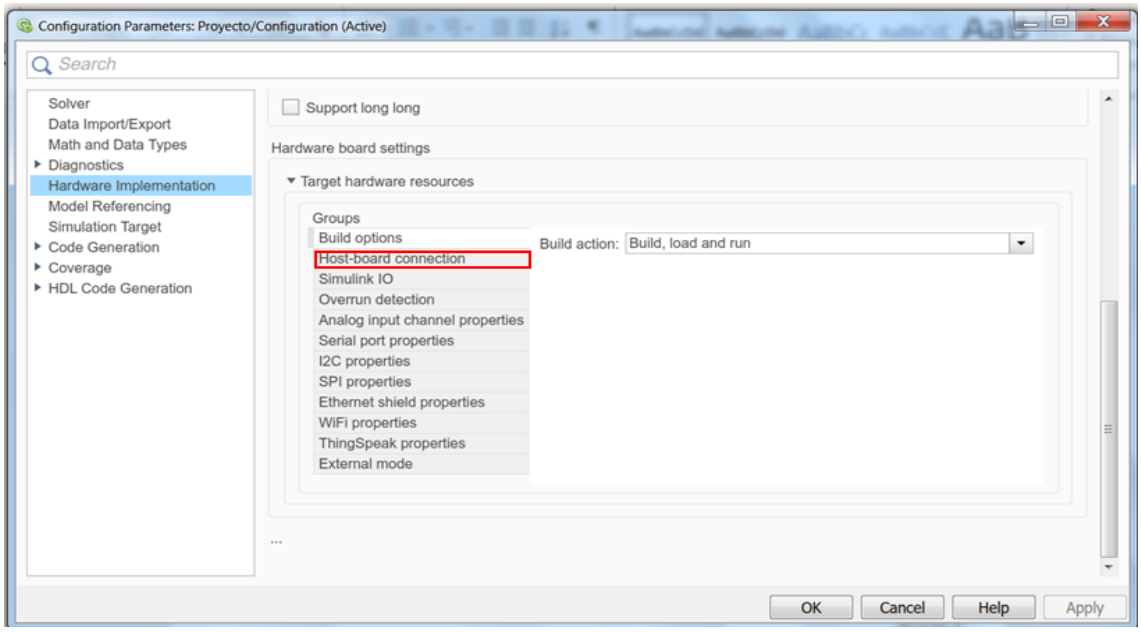

Figura 2.60. Menú de "Groups".

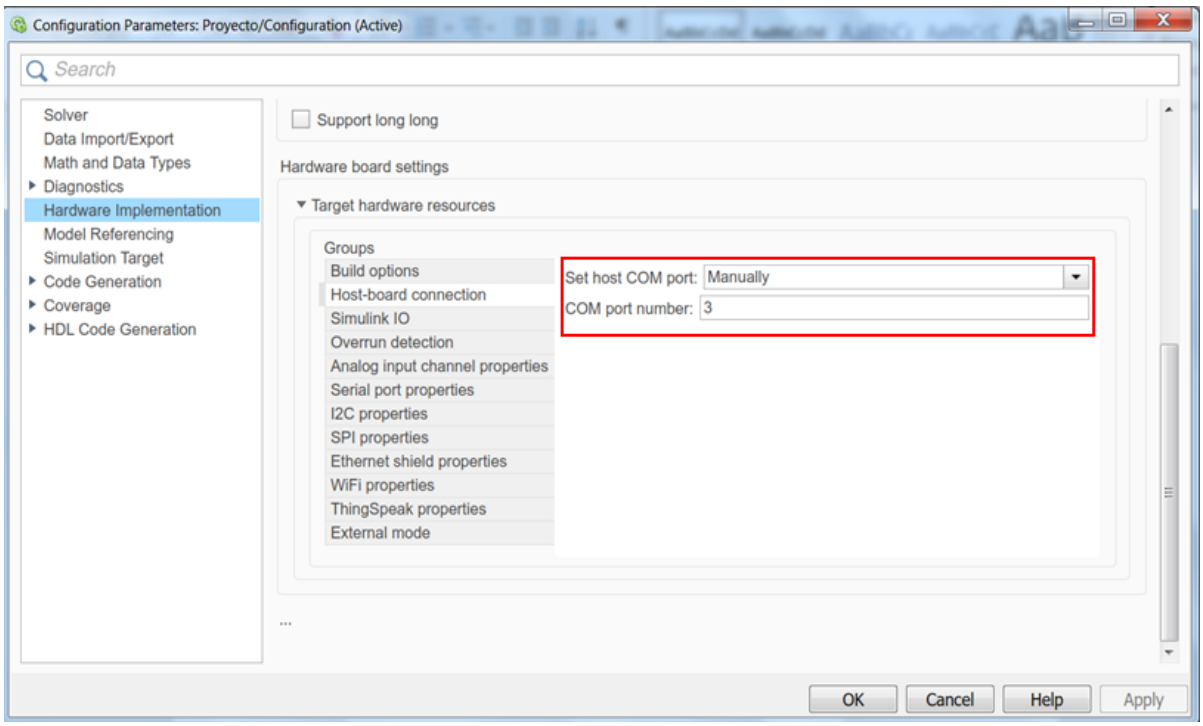

Figura 2.61. Menú de "Host-board connection".

CAPÍTULO 3. RESULTADOS

Antes de iniciar a graficar las distintas pruebas llevadas a cabo hay que configurar unos parámetros de Simulink tal como se muestra en la figura 3.1.

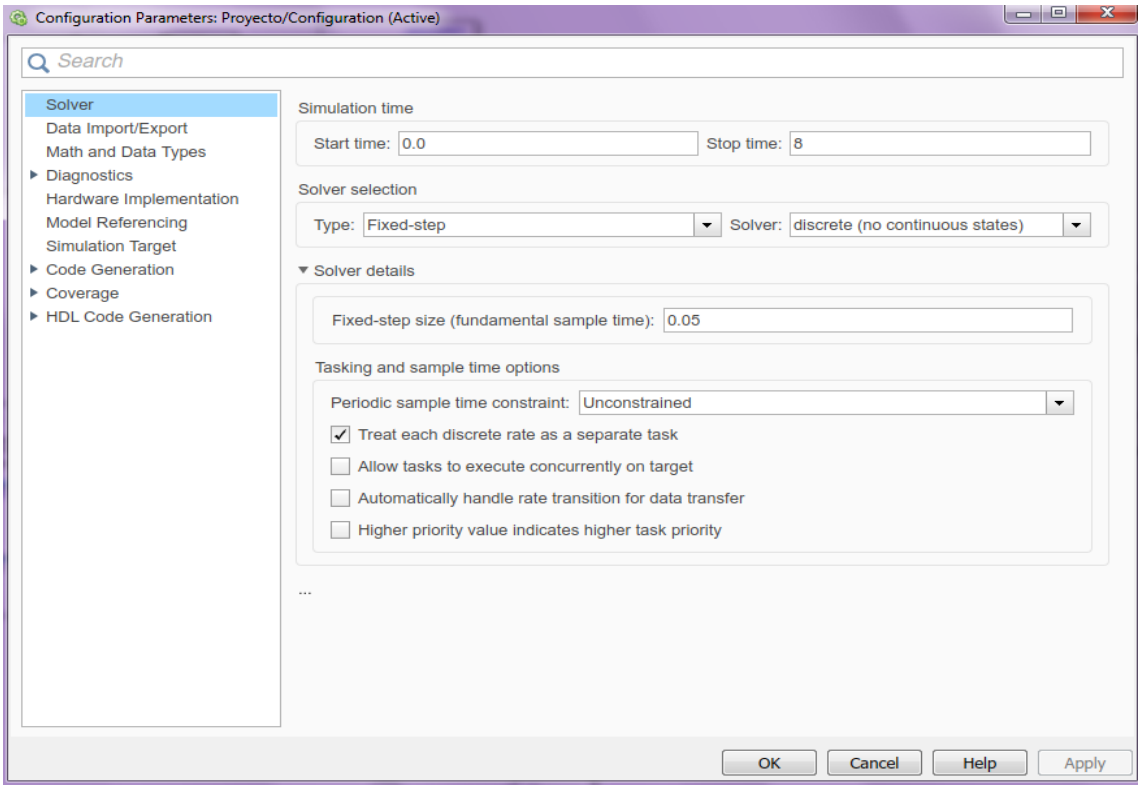

Figura 3.1. Ventana de "Configuration Parameters (Solver)".

Seleccionamos el sensor ultrasónico para configurar a que pines del Arduino lo queremos conectar tal cual como se muestra en la figura 3.2, los pines pueden ser otros de las salidas digitales, ya que el pin 8 y 9 no son exclusivos del sensor.

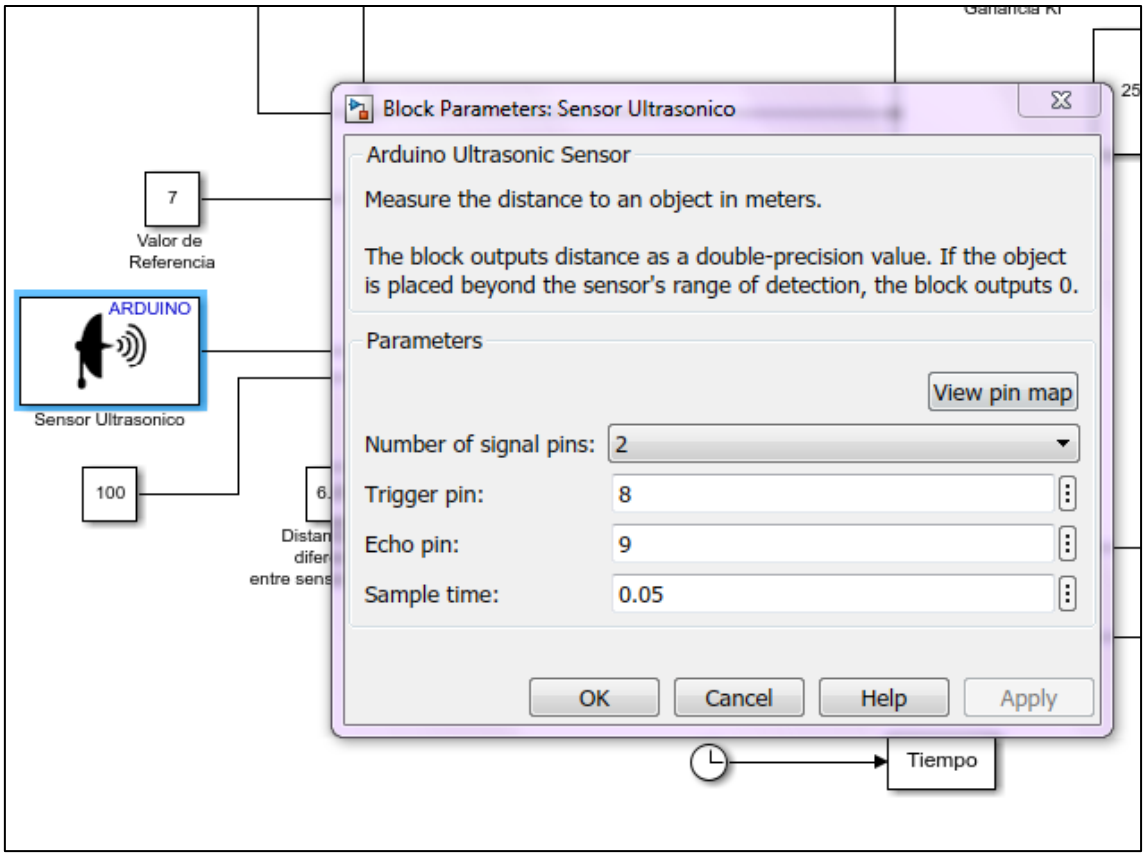

Figura 3.2. Ventana de configuración de "Sensor Ultrasónico".

También seleccionamos el bloque de señal PWM, para seleccionar a que pin del Arduino Uno lo deseamos conectar, necesitamos que sea uno de los puertos que están destinado a PWM de las salidas digitales vea la figura 3.3.

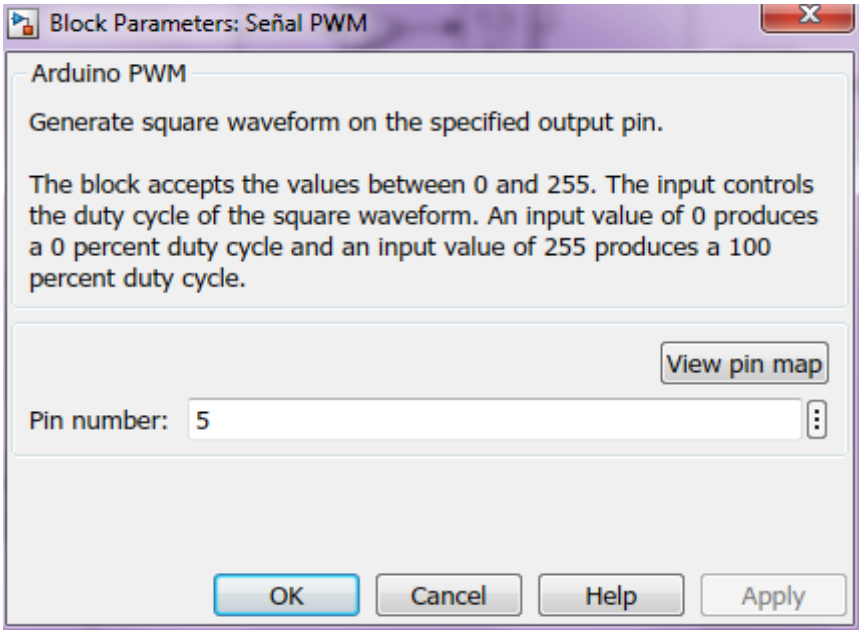

Figura 3.3. Ventana de configuración de "Señal PWM".

A continuación, en la figura 3.4 se muestran los bloques encargados de mover el motor hacia la izquierda o derecha, o bien dejarlo sin movimiento, dependiendo del valor de error registrado.

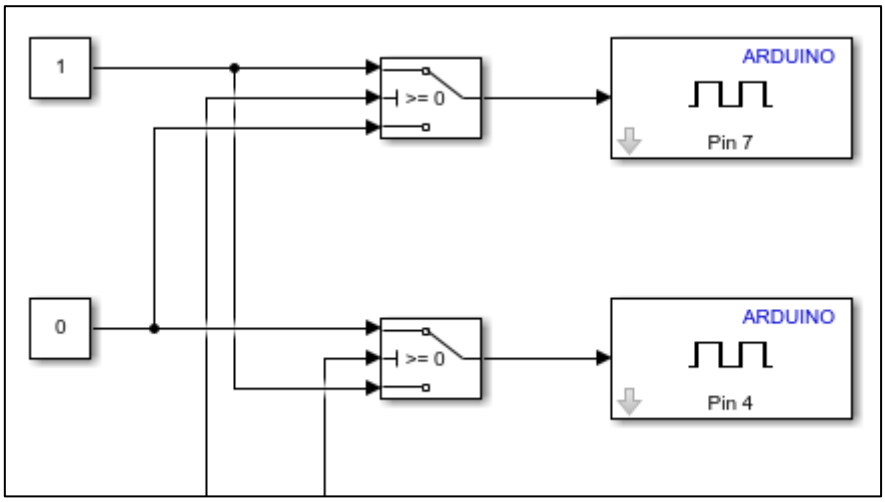

Figura 3.4. Bloques en Simulink de selección para accionar pin 7 o 4.

El diagrama completo del control PI se muestra en la figura 3.5. Podemos observar que el diagrama abarca diferentes secciones, las cuales se explican a continuación.

Sección Amarilla

En esta sección, como ya se había mencionado anteriormente, se muestra un comparador, el cual al registrar un valor de error mayor de cero, mueve el motor hacia la derecha y si es menor, lo mueve hacia la izquierda y en caso de ser cero no se mueve para ningún sentido así mismo como sucedió en la programación de Arduino.

Sección Azul

Esta sección es una de las más importantes de todo el diagrama, ya que es la que contiene al control PI, contiene dos bloques de ganancia: una para la ganancia del control Proporcional y otro para la ganancia del control Integral, siendo esta multiplicada por el error de la sección negra, para que al final de ese bloque se sumen ambos controles.

Sección Café

Esta sección a pesar de solo tener un bloque, tiene un valor muy importante, ya que será el valor de referencia, a donde queremos mover nuestro móvil, la posición estará dada en centímetros.

Sección Verde

Esta parte nos muestra el valor que genera el sensor ultrasónico, pero al visualizarlo con un scope nos mostraba valores muy pequeños, es decir, si estábamos a una distancia de 2 centímetros, solo mostraba un valor de 0.02 y para cuestiones gráficas el tener valores muy pequeños no son buenos para su visualización, por lo que se multiplicó por 100, para que en lugar de obtener 0.02, se tendría un valor de 2 centímetros. Pero también es importante saber, que se tenía que restar una pequeña distancia de 6.25 cm por cuestiones de la base del riel y de la posición del sensor ultrasónico, finalmente ya se tenía un valor más exacto de la distancia pero al llevar acabo diferentes pruebas, los valores registrados tenían mucha variación, por lo que se colocó un bloque de redondeo para obtener solo un valor de posición.

Sección Negra

Esta sección tiene el bloque del error, ya que en él se guardará el valor de la operación "valor de referencia menos el valor de la posición inicial".

Sección Morada

En esta sección redondeamos el valor generado por la sección azul, enseguida se colocó un bloque de valor absoluto para cuando hubiese valores negativos, para después pasar al bloque de la señal PWM que como dato curioso solo funciona con valores de 0 a 255.

Secciones Rosas

Para guardar y poder graficar a PWM, Error, Referencia y Posición se usó el bloque con el nombre "To Workspace".

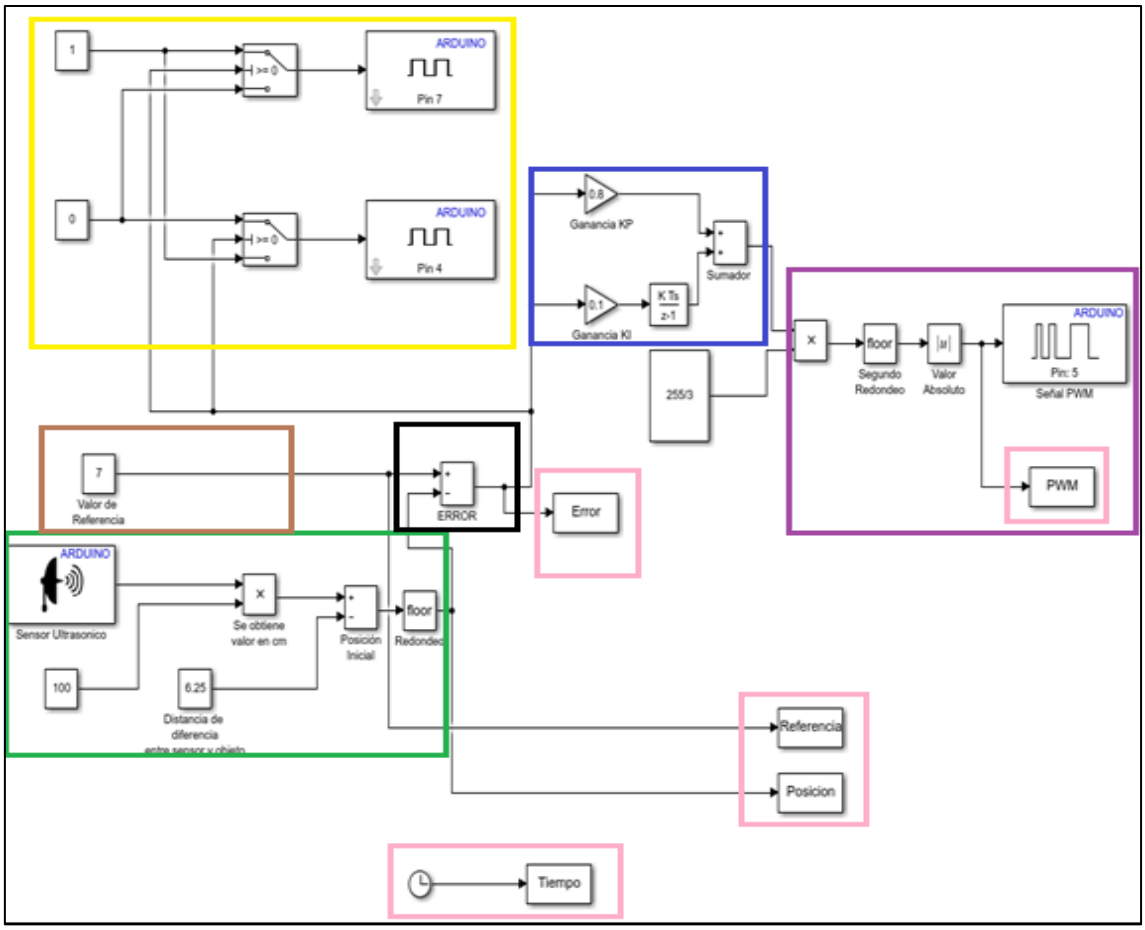

Figura 3.5. Diagrama completo del Control PI en Matlab-Simulink.

Como primera prueba se colocó una referencia de 7 cm con una posición inicial 0 y una ganancia de kp de 0.8 y ki de 0.2 como se muestra en la figura 3.6.

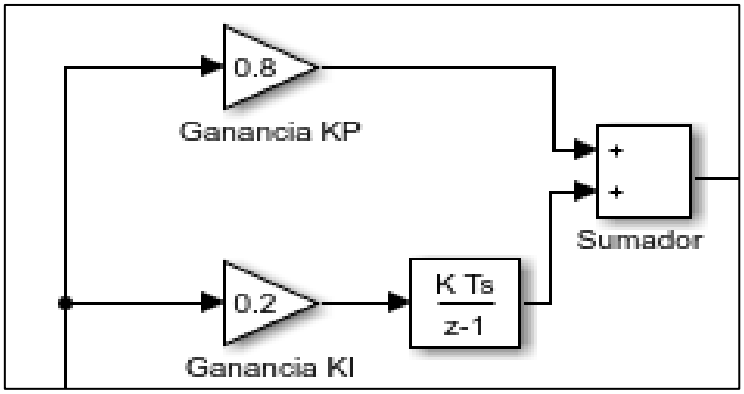

Figura 3.6. Ganancia de Kp=0.8 y Ganancia de Ki=0.2.

A continuación, se muestran unas líneas de comandos para la gráfica de referencia que se muestra en la figura 3.7, las cuales se utilizan para todas las gráficas solo cambiando los datos.

Plot: se encarga de generar una gráfica, en este caso primero va el valor que se encontrará en el eje x (para este ejemplo "Tiempo") y el segundo valor en el eje y (para este ejemplo "Referencia").

Xlabel: coloca una leyenda en el eje x para este ejemplo se colocó "Tiempo".

Ylabel: coloca una leyenda en el eje y para este ejemplo se colocó "Distancia en CM".

Title: coloca un título en la parte superior de la gráfica para este ejemplo se colocó "Grafica de Referencia".

Grid on: coloca una cuadrícula en la gráfica para poder observar los distintos cambios de esta.

```
>> plot (Tiempo, Referencia)
>> xlabel ('Tiempo')
>> ylabel ('Distancia en CM')
>> title ('Grafica de Referencia')
>> grid on
```

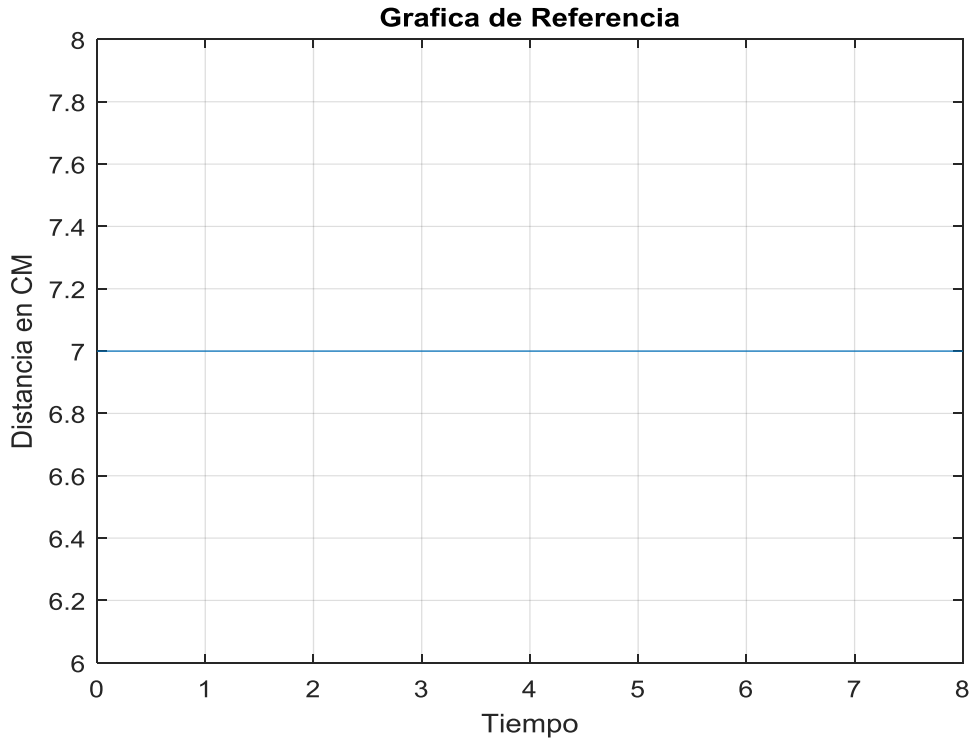

Figura 3.7. Gráfica de Referencia con Kp=0.8 y Ki=0.2.

A continuación, se generó la gráfica de Posición con kp=0.8 y ki=0.2 como se muestra en la figura 3.8.

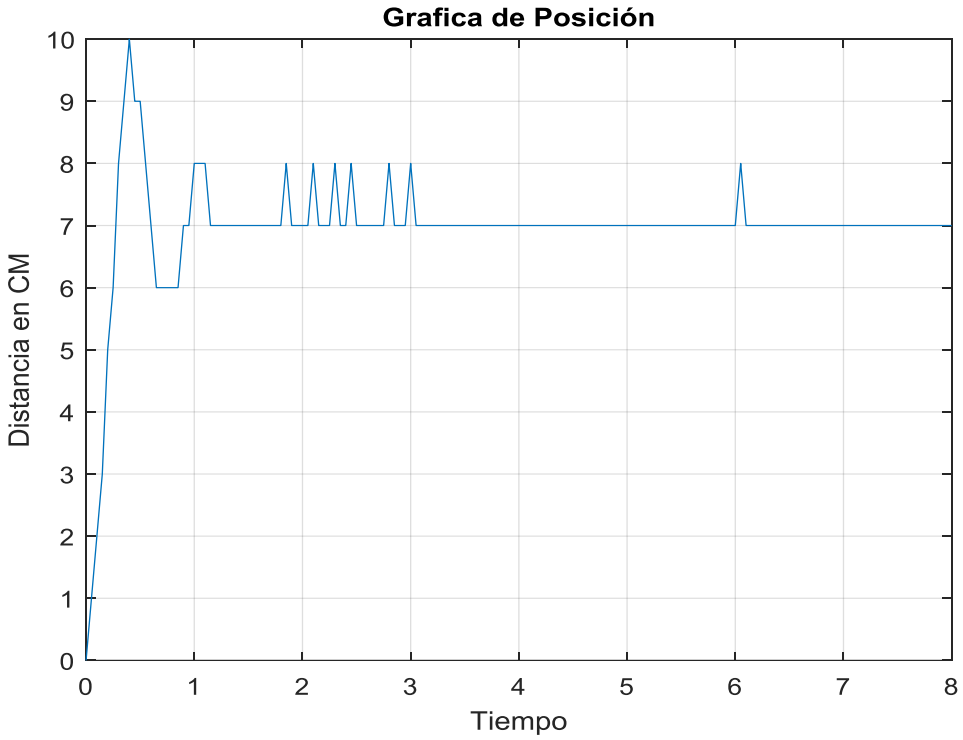

Figura 3.8. Gráfica de Posición con Kp=0.8 y Ki=0.2.

Después se generó la gráfica de Error con kp=0.8 y ki=0.2 como se muestra en la figura 3.9.

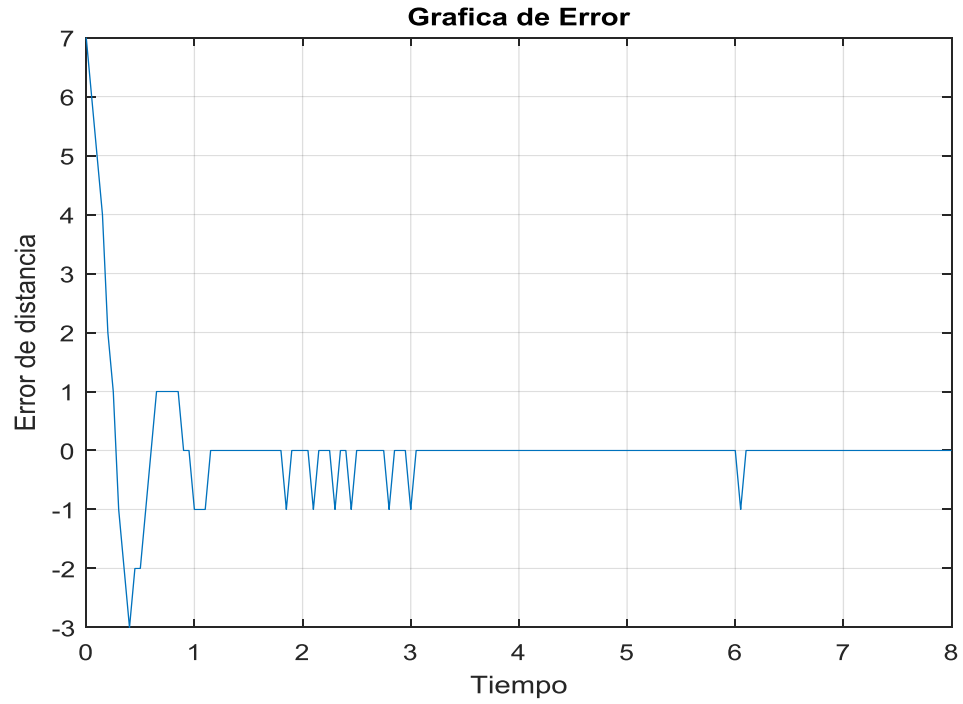

Figura 3.9. Gráfica de Error con Kp=0.8 y Ki=0.2.

Y finalmente se generó la gráfica de PWM con kp=0.8 y ki=0.2 como se muestra en la figura 3.10.

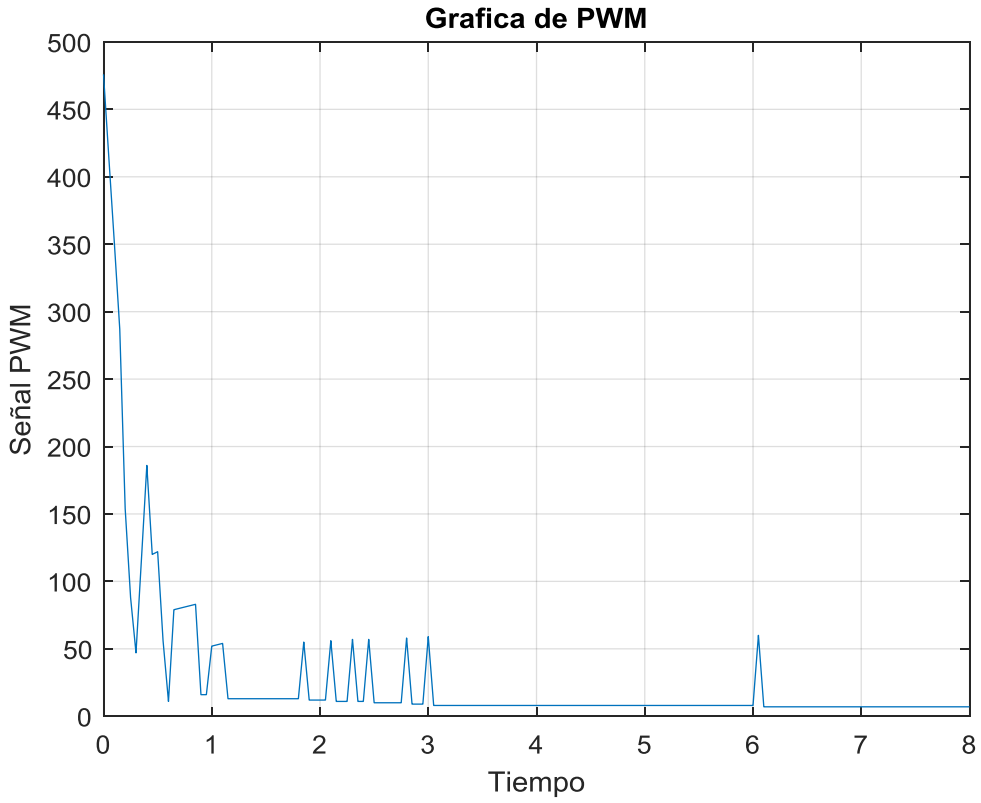

Figura 3.10. Gráfica de PWM con Kp=0.8 y Ki=0.2.

En la segunda prueba se colocó una referencia de 7 cm con posición inicial 0 y una ganancia de kp de 1 y ki de 0.6 como se muestra en la figura 3.11.

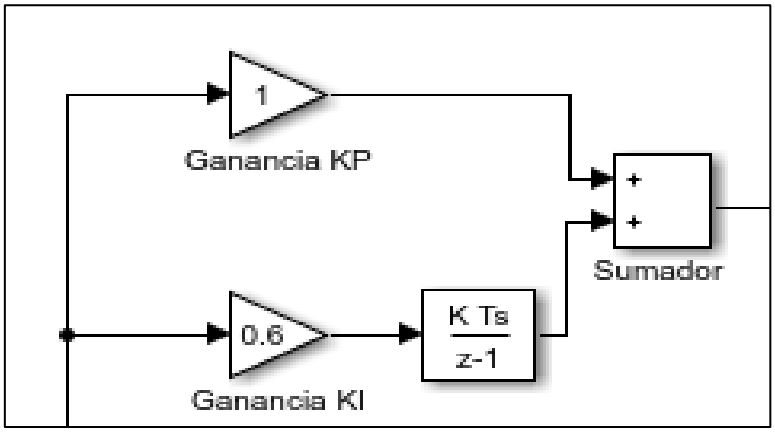

Figura 3.11. Ganancia de Kp=1 y Ganancia de Ki=0.6.

Ahora se generó la gráfica de referencia con ganancia de kp=1 y ganancia de ki=0.6 como se muestra en la figura 3.12.

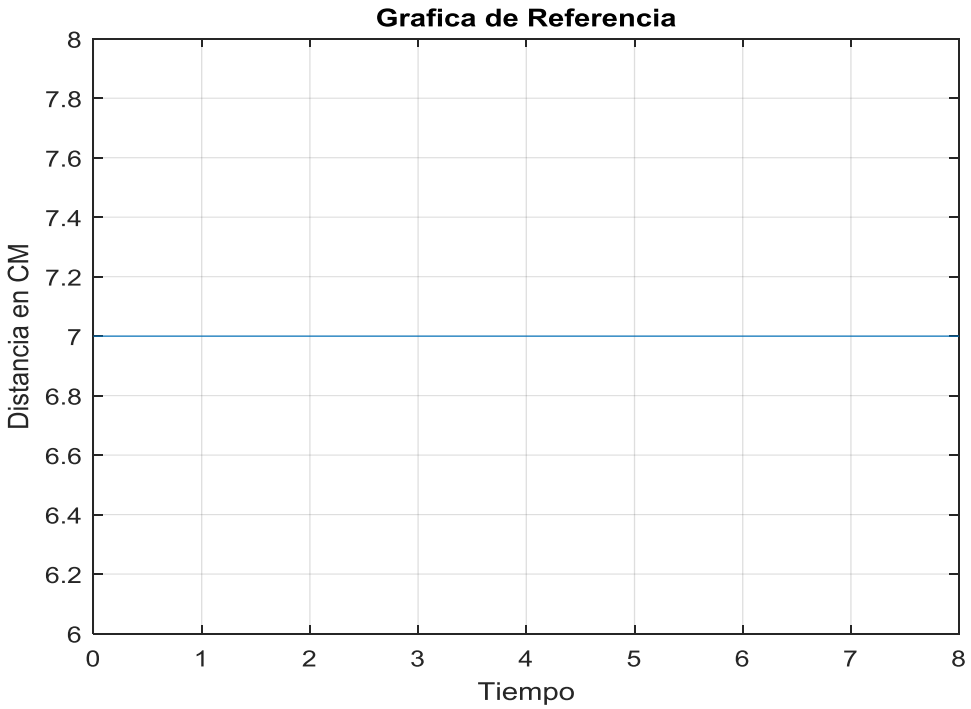

Figura 3.12. Gráfica de Referencia con Kp=1 y Ki=0.6.

A continuación, se generó la gráfica de Posición con kp=1 y ki=0.6 como se muestra en la figura 3.13.

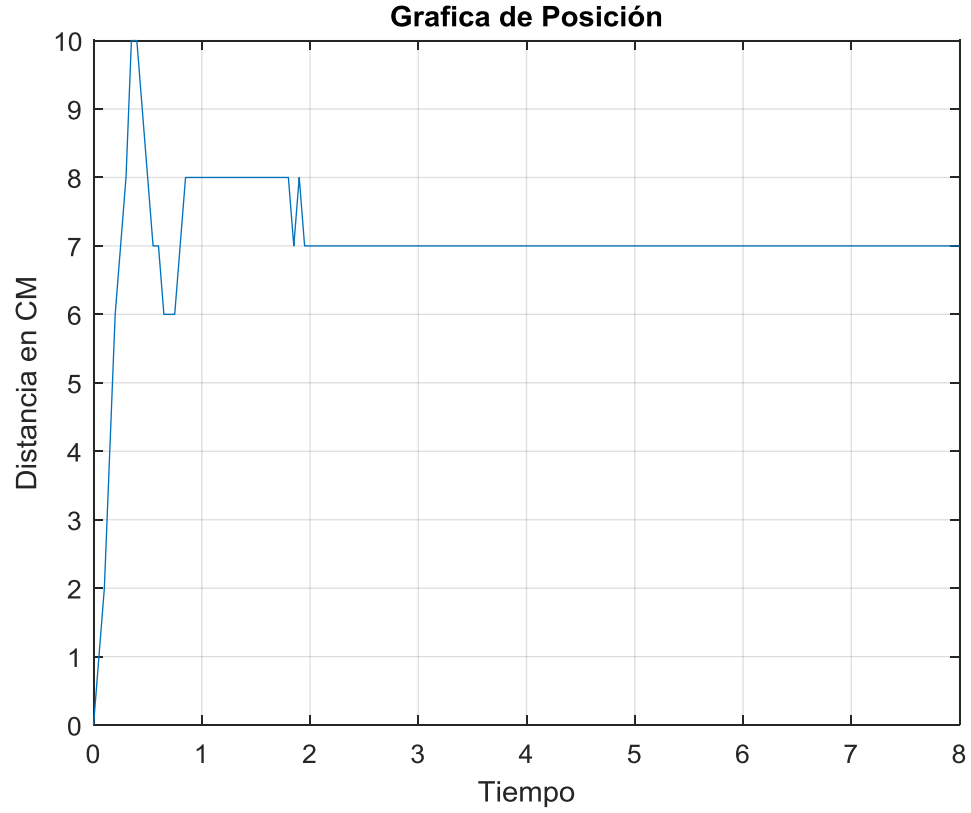

Figura 3.13. Gráfica de Posición con Kp=1 y Ki=0.6.

Después se generó la gráfica de Error con kp=1 y ki=0.6 como se muestra en la figura 3.14.

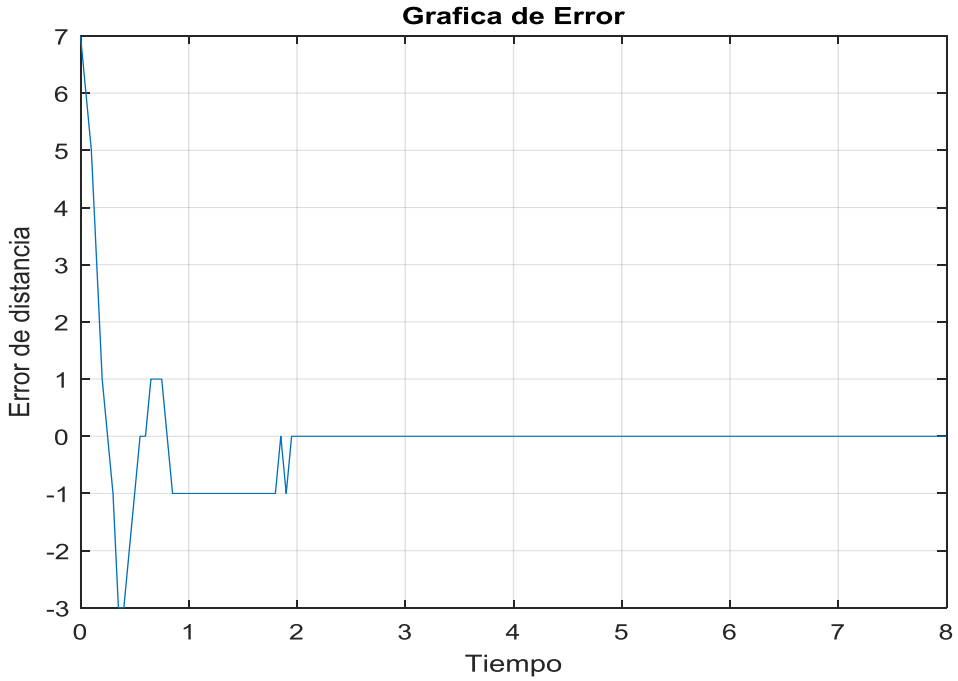

Figura 3.14. Gráfica de Error con Kp=1 y Ki=0.6.

Y finalmente se generó la gráfica de PWM con kp=1 y ki=0.6 como se muestra en la figura 3.15.

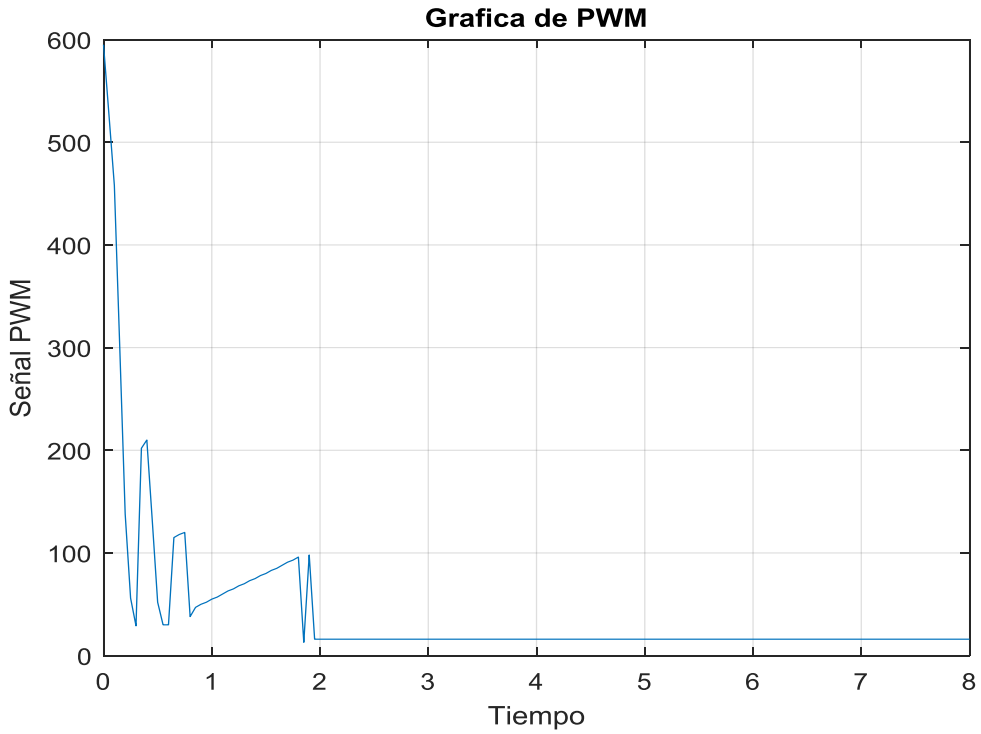

Figura 3.15. Gráfica de PWM con Kp=1 y Ki=0.6.

Para la tercera prueba se colocó una referencia de 10 cm con posición inicial de 2 cm y una ganancia de kp de 1.2 y ki de 0.5 como se muestra en la figura 3.16.

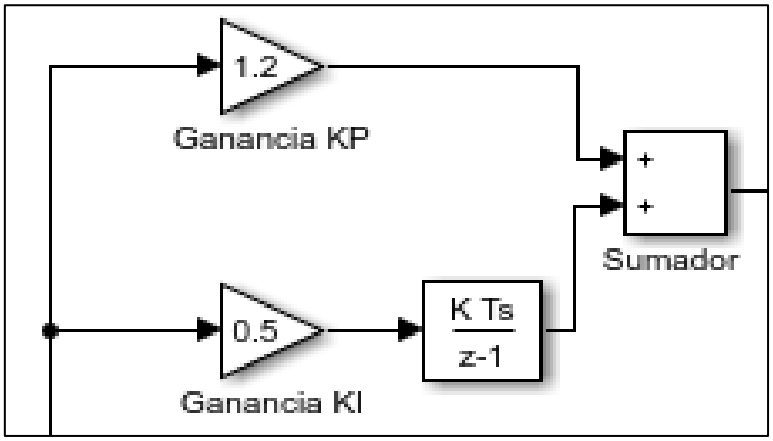

Figura 3.16. Ganancia de Kp=1.2 y Ganancia de Ki=0.5.

Ahora se generó la gráfica de referencia con ganancia de kp=1.2 y ganancia de ki=0.5 como se muestra en la figura 3.17.

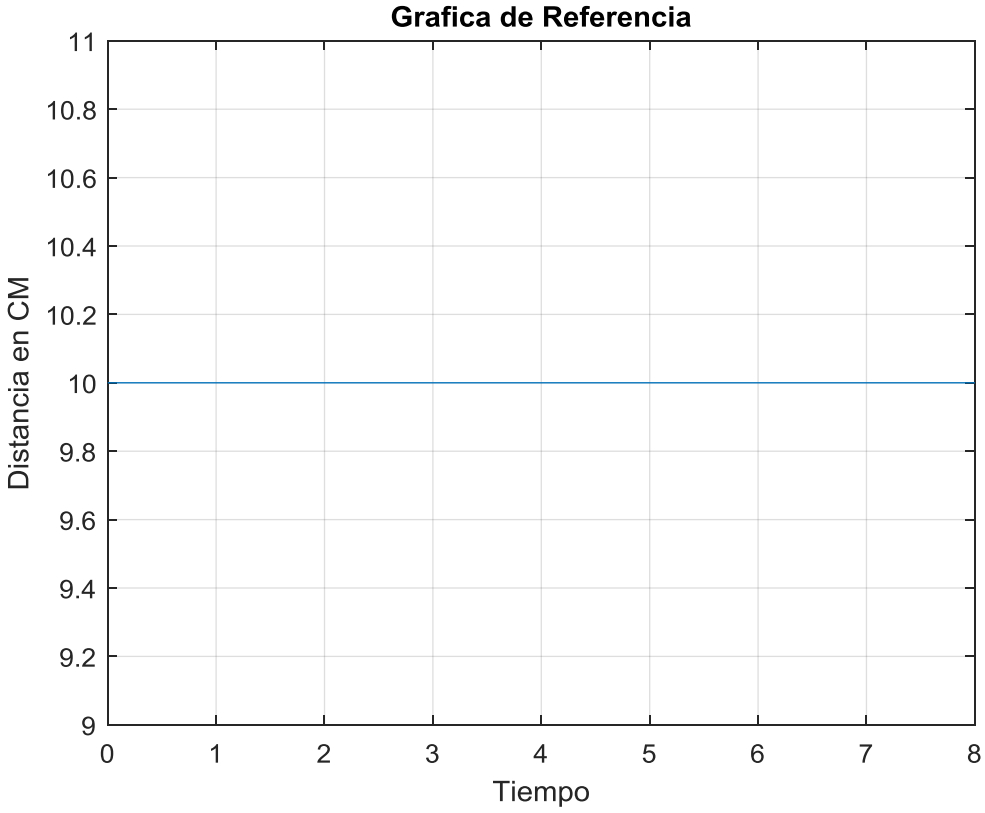

Figura 3.17. Gráfica de Referencia con Kp=1.2 y Ki=0.5.

A continuación, se generó la gráfica de Posición con kp=1.2 y ki=0.5 como se muestra en la figura 3.18.

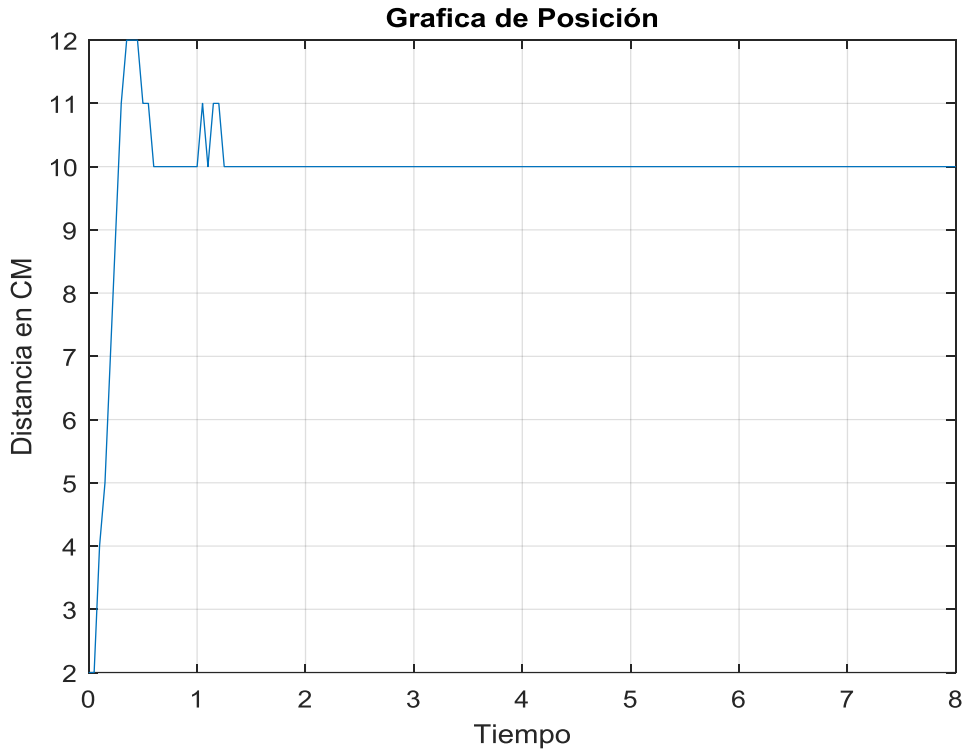

Figura 3.18. Gráfica de Posición con Kp=1.2 y Ki=0.5.

Después se generó la gráfica de Error con kp=1.2 y ki=0.5 como se muestra en la figura 3.19.

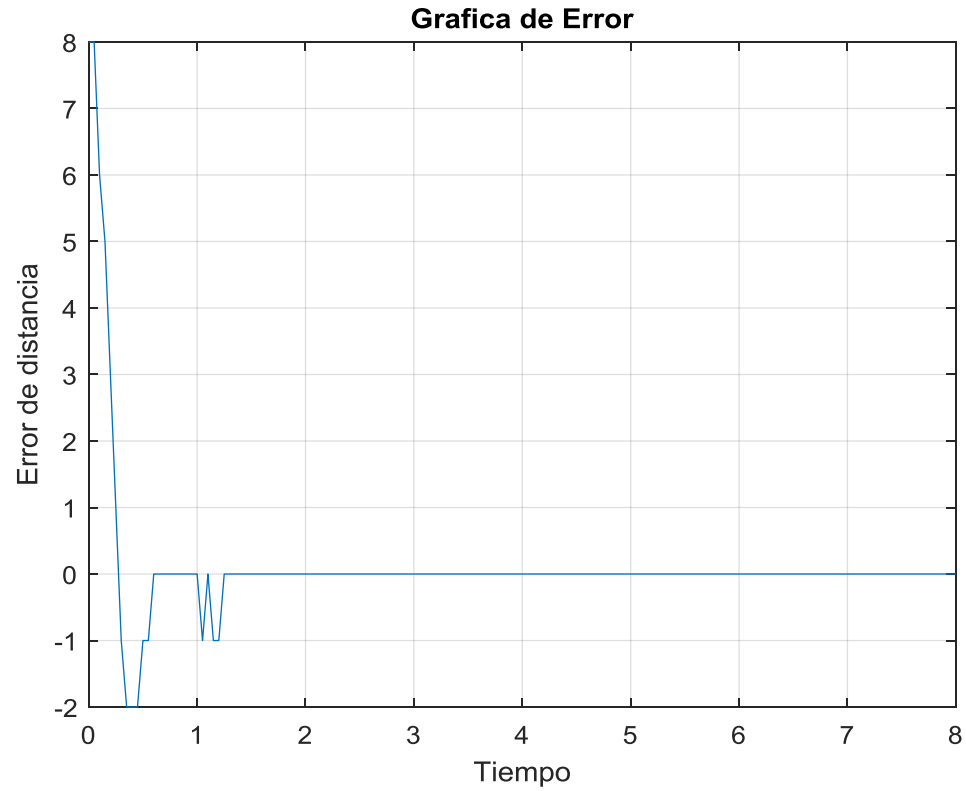

Figura 3.19. Gráfica de Error con Kp=1.2 y Ki=0.5.

Y finalmente se generó la gráfica de PWM con kp=1.2 y ki=0.5 como se muestra en la figura 3.20.

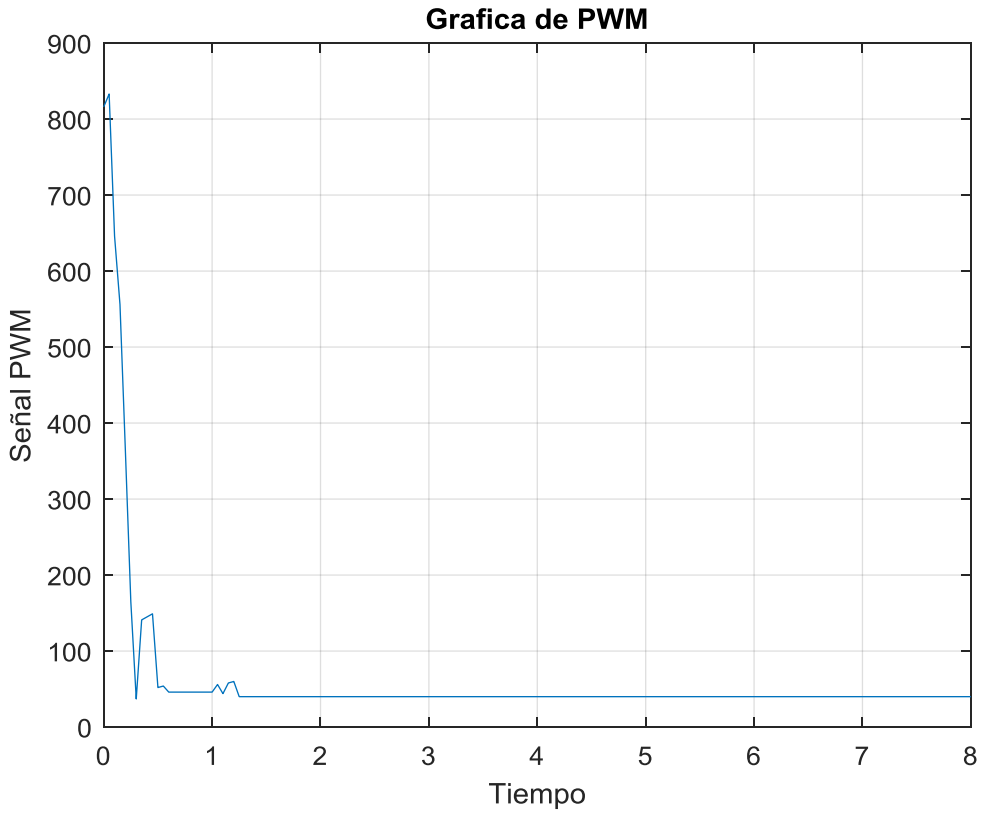

Figura 3.20. Gráfica de PWM con Kp=1.2 y Ki=0.5.

Por último, en la cuarta prueba se colocó una referencia de 10 cm con posición inicial 2 cm y una ganancia de kp de 0.5 y ki de 0.5 como se muestra en la figura 3.21.

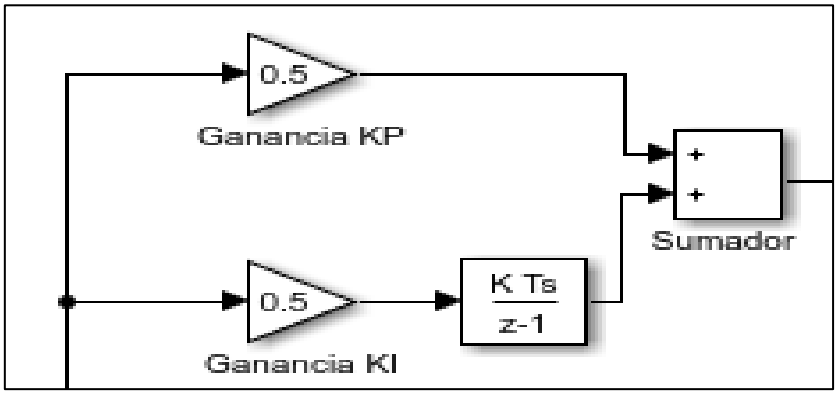

Figura 3.21. Ganancia de Kp=0.5 y Ganancia de Ki=0.5.

Ahora se generó la gráfica de referencia con ganancia de kp=0.5 y ganancia de ki=0.5 como se muestra en la figura 3.22.

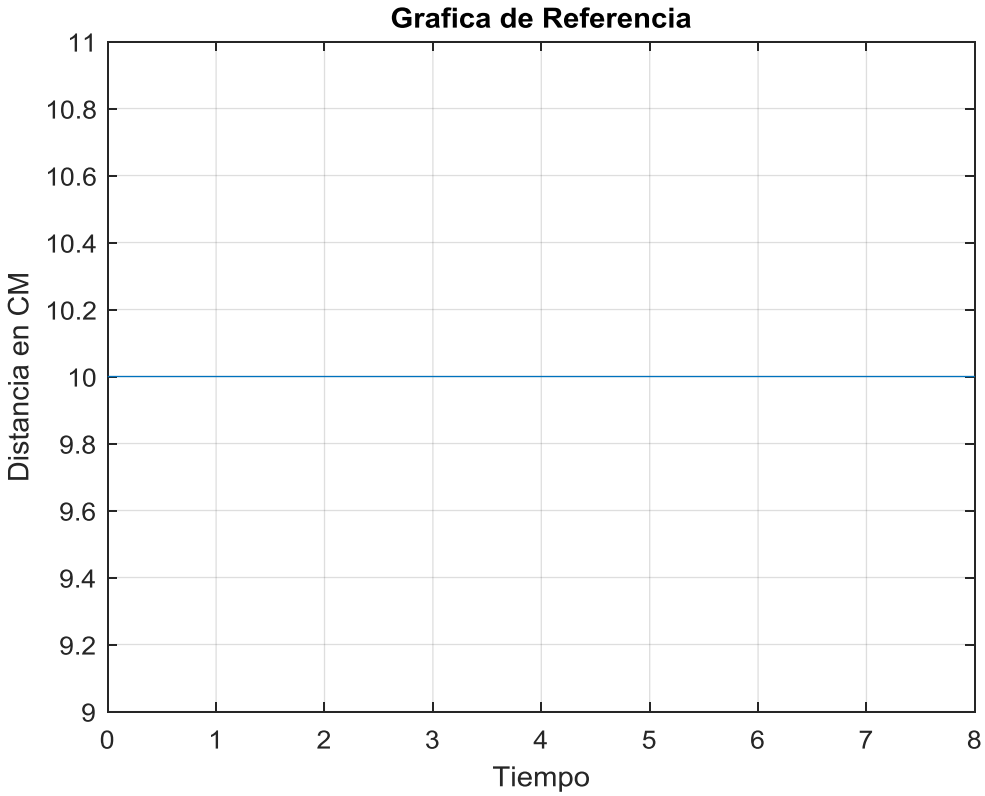

Figura 3.22. Gráfica de Referencia con Kp=0.5 y Ki=0.5.

A continuación, se generó la gráfica de Posición con kp=0.5 y ki=0.5 como se muestra en la figura 3.23.

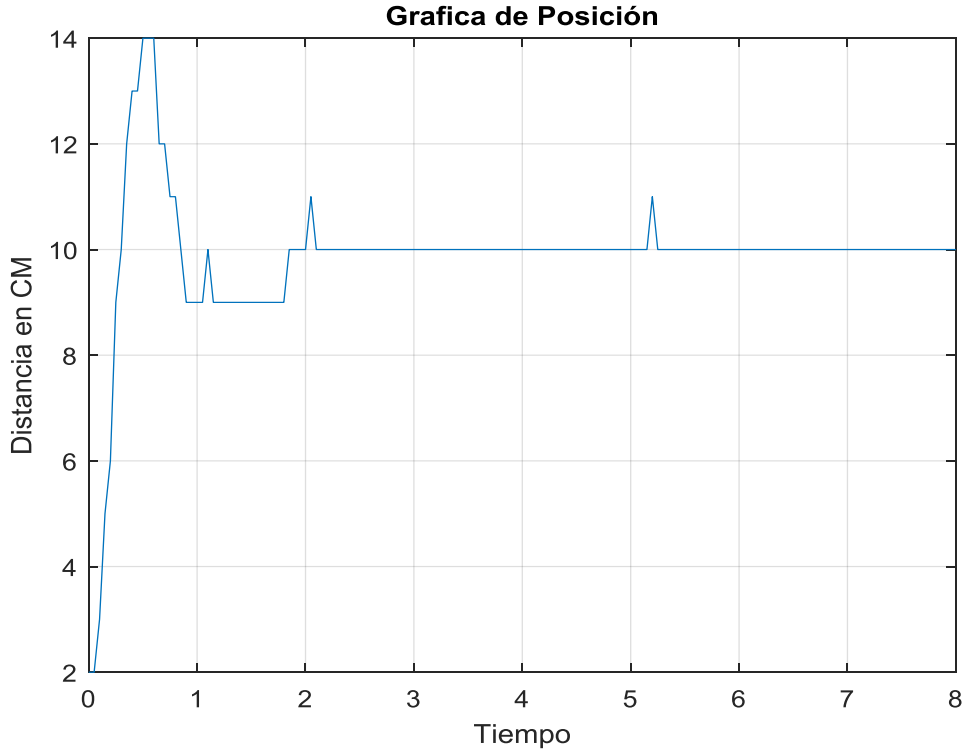

Figura 3.23. Gráfica de Posición con Kp=0.5 y Ki=0.5.

Después se generó la gráfica de Error con kp=0.5 y ki=0.5 como se muestra en la figura 3.24.

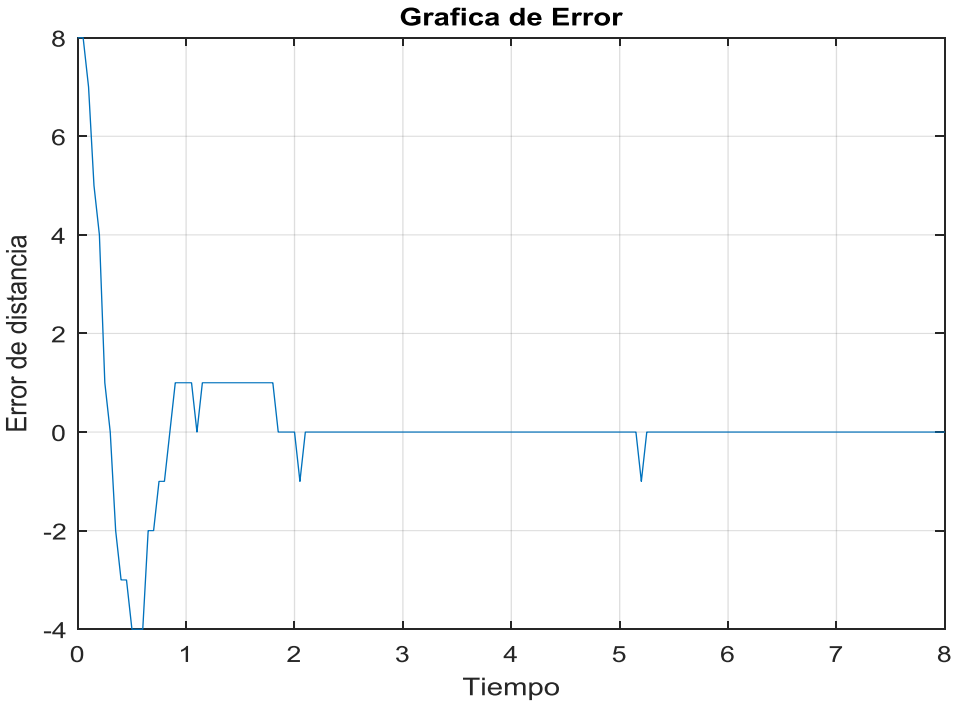

Figura 3.24. Gráfica de Error con Kp=0.5 y Ki=0.5.

Y finalmente se generó la gráfica de PWM con kp=1.2 y ki=0.5 como se muestra en la figura 3.25.

Figura 3.25. Gráfica de PWM con Kp=0.5 y Ki=0.5.

CAPÍTULO 4. ANÁLISIS Y DISCUSIÓN DE RESULTADOS

De las gráficas obtenidas del sistema de control PI se observa que el error tiende a cero a pesar de tener una referencia con distintos valores, además de tener ganancias tanto de KP como de KI distintas.

También se varió la posición inicial, y observamos que se logró llegar a la referencia.

Las gráficas de posición se mostraron muy distintas en cada uno de los casos analizados ya que como se muestra en la figura 3.8 tardo casi 6.2 segundos en llegar a la posición de referencia, mientas que en la figura 3.13 solo tardo 1.4 segundos en llegar a la posición de referencia.

En algunas pruebas a pesar de tener distintas posiciones tanto inicial como de referencia, para poder llegar a un error de 0 y una posición inicial igual a la de referencia dependió más de los valores de ganancia de KP como de KI.

El control integral fue fundamental en el sistema de control ya que de él dependía de que el vehículo llegara a su posición de referencia, ya que de lo contrario teniendo solo un control proporcional el vehículo solo se mantendría oscilando como sucedió en el código de control proporcional en la placa Arduino Uno.

CAPÍTULO 5. CONCLUSIONES

En ocasiones al llevar a cabo el diseño de un prototipo suelen ocurrir muchos problemas como sucedió en este prototipo, en el cual el sensor infrarrojo no registró valores correctos de posición, tanto de la posición inicial como de referencia, a pesar de a verlo caracterizado dos veces, por lo que no tuvo sentido registrar valores de posición con ese sensor, por lo que se optó por colocar un sensor ultrasónico el cual tuvo mediciones más precisas a diferencia del sensor infrarrojo.

Lo mismo sucedió con la etapa de potencia, en la cual se tenia un diseño de puente H para el control del giro del motor de DC, pero dicho diseño no entregaba el suficiente voltaje para poder mover un motor de DC con motorreductor, por lo que analizando diferentes posibilidades de circuitos integrados, se optó por el L293D el cual debía de ir protegido de 4 diodos, para evitar que dicho circuito se dañara por cambios bruscos al cambiar de un giro a otro el motor de DC.

La instalación de la paquetería en Matlab fue fundamental, ya que dichas instalaciones se llevaron a cabo en diferentes versiones, en las cuales la paquetería de Arduino no era compatible, por lo que se instaló la versión 2018b que a pesar de no contar con la paquetería de Arduino precargada, permitía la opción de instalación de esta.

Finalmente, al analizar las simulaciones en tiempo real, se encontró que para evitar en un tiempo menos prolongado y lograr el menor número de oscilaciones fue necesario variar los valores de ganancias de KI y KP para lograr una mejor respuesta de la posición en el sistema de lazo cerrado

CAPÍTULO 6. ENTREGABLES

1.- Puesto en operación el prototipo para el control de posición de un vehículo sobre riel como apoyo a la docencia e investigación. El cual dicho diseño se encuentra en todo el capítulo 2, donde se muestra cada uno de los componentes para llevar a cabo dicho prototipo, así como paso a paso de este, desde la caracterización del sensor, como la etapa de potencia, así como también las señales en tiempo real.

2.- Diagramas eléctricos de conexión y diagramas esquemáticos (la etapa de potencia, sensor con acondicionamiento de señal) que de igual forma se muestran en el capítulo 2.

3.- Programas en MATLAB/Simulink los cuales se muestran en el capítulo 3.

4.- Pruebas operativas en tiempo real. Estás se encuentran en el capítulo 3, las cuales son respuestas gráficas en tiempo real.

REFERENCIAS BIBLIOGRÁFICAS

[1] F. Ávila, J. I. Carbajal, y J. Mares "Control de velocidad y dirección de un robot de carreras autónomo", Tesis, ESIME, Instituto Politécnico Nacional, México D.F., 2011.

[2] I. Ramírez "Control PID de temperatura con PLC Siemens S7-300 y Allen Bradley SLC 500", Tesis, Facultad de Ingeniería, Universidad Nacional Autónoma de México, México D.F., 2017.

[3] J. P. López y J. E. Morales "Implementación de un sistema de domótica para el control de temperatura e iluminación de un hogar", Proyecto Terminal, ESIME, Instituto Politécnico Nacional, México D.F., 2017.

[4] C. Yaneth, W. Reinaldo, y E. Gómez "Instrumentación y control de nivel para un sistema de tanques acoplados en el laboratorio de control e instrumentación de la E3T/UIS", Trabajo de grado, Facultad de Ingenierías Físico-Mecánicas, Universidad Industrial de Santander, Colombia Bucaramanga, 2014.

[5] I. I. Siller-Alcalá "Notas para el curso Control Digital" UAM Azcapotzalco, Departamento de electrónica, 2018.