**Proyecto 2**

**SEGUIDOR DE LINEA CON CONTROL PID**

Pérez C. Carlos Felipe, Marmolejo V. Alejandro, Foronda M. José

***Resumen*—En este proyecto 2, se diseñará e implementará un seguidor de línea, el cual constará de una serie de partes, además su funcionamiento será controlado por la tarjeta de desarrollo ATMEGA 2560.con la cual se le hará un control PID, para mejorar la respuesta del carro.**

*Palabras claves*—PID, ATMEGA 2560, seguidor de línea, motores, llantas.

# Introducción

El controlador PID (proporcional, integrativo y derivativo), es un controlador realimentado, cuyo propósito principal es reducir el error en estado estacionario el cual está comprendido entre la diferencia de la entrada y la salida del sistema, este control por lo general usado en la industria calcula la desviación o error entre un valor medido y un valor deseado, este se compone de las constantes anteriormente mencionadas, donde la proporcional depende de la señal de error entrante multiplicado por la constante proporcional de manera que el error en estado estable sea aproximadamente cero, la acción integral pretende asegurar que la salida del proceso concuerde con la referencia en estado estacionario. Ya que, Con la acción proporcional, normalmente existirá un error en estado estacionario. De manera que, Con la acción integral, un pequeño error positivo siempre producirá un incremento en la señal de control y, un error negativo siempre dará una señal decreciente sin importar cuán pequeño sea el error, por último, la acción derivativa trata de mantener el error al mínimo corrigiéndolo proporcionalmente con la misma velocidad que se produce; de esta manera evita que el error se incremente. Para este proyecto implementaremos un seguidor de línea, el cual será direccionado con la tarjeta de desarrollo, ATMEGA 2560, de manera que esta será la encargada de controlar los movimientos del seguidor, a su vez haciendo uso de un control por realimentación de estados, PID anteriormente mencionado, de manera que logre darle mayor precisión, todo esto a partir de unas señales provenientes de un arreglo de sensores en nuestro caso digitales POLOLU, los cuales nos proporcionaran una serie de funciones incluidas en una librería, la cual fue pensada especialmente para esta serie de proyectos.

1. OBJETIVOS
   1. *GENERAL*

Implementar un vehículo seguidor de línea usando la plataforma de arduino y empleando un controlador PID que permita un movimiento acorde a las señales censadas y ejecutadas mediante la tarjeta de desarrollo ATMEGA 2560.

* 1. *ESPECÍFICOS*
* Construir o implementar la estructura física del vehículo seguidor de línea de acuerdo a parámetros de diseño tenidos en cuenta.
* Acondicionar la tarjeta de desarrollo al vehículo de forma que pueda ejercer la ejecución o intercambio de señales con el mismo, y los sensores que este tiene adaptados.
* Diseñar el código base PID que permita realizar la ejecución del movimiento de forma estable y acorde al comportamiento esperado.

# MARCO TEORICO

**Estructura del PID**

Consideremos un lazo de control de una entrada y una salida (SISO) de un grado de libertad:

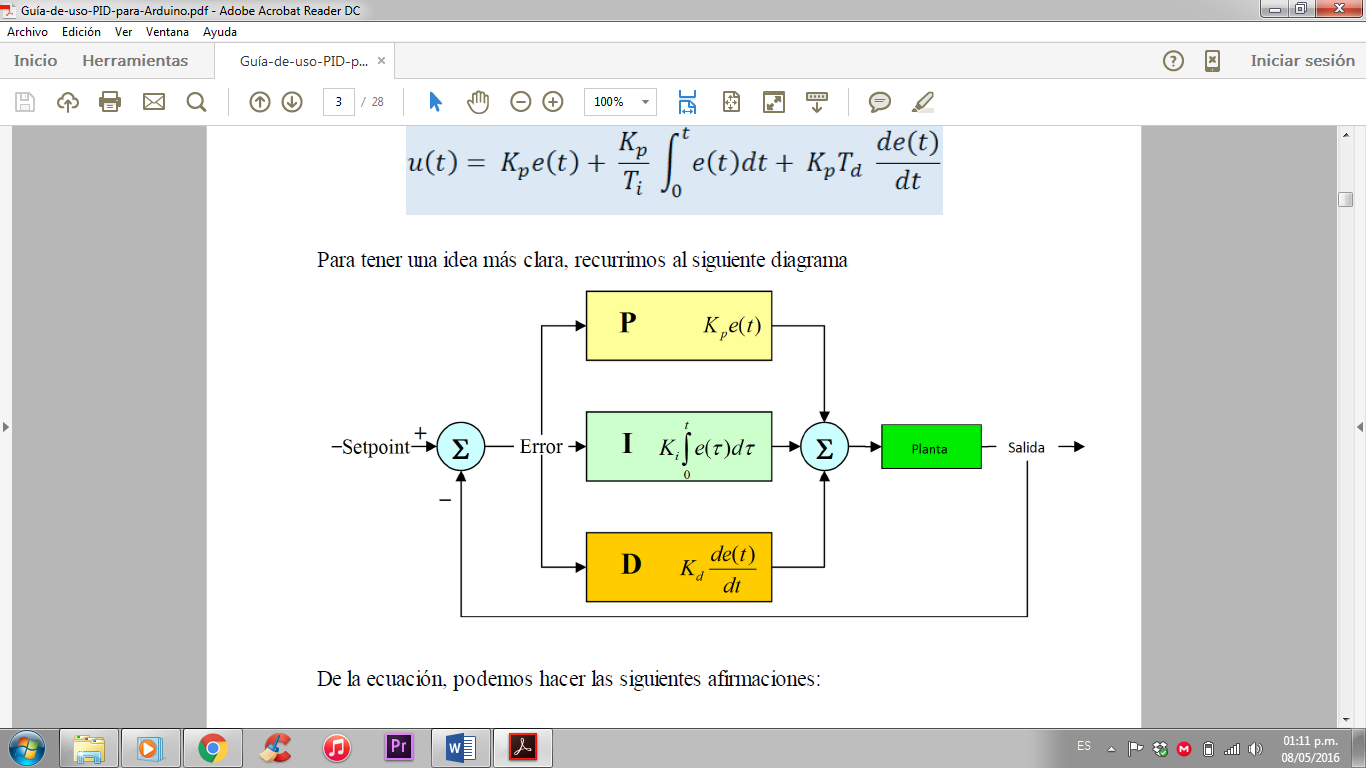


Fig. 1 Diagrama de Bloques

Los miembros de la familia de controladores PID, incluyen tres acciones: proporcional (P), integral (I) y derivativa (D). Estos controladores son los denominados P, I, PI, PD y PID

* **P: acción de control proporcional**, da una slaida del controlador que es proporcional al error, es decir u(t)=KP.e(t), que descripta desde su función transferencia queda:

(1)

Donde es una ganancia proporcional ajustable. Un controlador proporcional puede controlar cualquier planta estable, pero posee desempeño limitado y error en régimen permanente (off-set).

* **I: acción de control integral:** da una salida del controlador que es proporcional al error acumulado, lo que implica que es un modo de controlador lento.

(2)

La señal de control u(t) tiene un valor diferente de cero cuando la señal de error e(t) es cero. Por lo que se concluye que, dada una referencia constante, o perturbaciones, el error en régimen permanente es cero.

* **PI: acción de control proporcional-integral,** se define mediante

(3)

Donde se denomina tiempo integral y es quien ajusta la acción integral. La función de transferencia resulta:

(4)

Con un control proporcional, es necesario que exista error para tener una acción de control distinta de cero. Con acción integral, un error pequeño positivo siempre nos dará una acción de control creciente, y si fuera negativo la señal de control será decreciente. Este razonamiento sencillo nos muestra que el error en régimen permanente será siempre cero.

Muchos controladores industriales tienen solo acción PI. Se puede demostrar que un control PI es adecuado para todos los procesos donde la dinámica es esencialmente de primer orden. Lo que puede demostrarse en forma sencilla, por ejemplo, mediante un ensayo al escalón.

* **PD: acción de control proporcional-derivativa,** se define mediante:

(5)

Donde es una constante de, denominada, tiempo derivativo. Esta acción tiene carácter de previsión, lo que hace más rápida la acción de control, aunque tiene la desventaja importante que amplifica las señales de ruido y puede provocar saturación en el actuador. La acción de control derivativa nunca se utiliza por sí sola, debido a que sólo es eficaz durante periodos transitorios. La función de transferencia de un controlador PD resulta:

(6)

Cuando una acción de control derivativa se agrega a un controlador proporcional, permite obtener un controlador de alta sensibilidad, es decir que responde a la velocidad del cambio del error y produce corrección significativa antes de que la magnitud del error se vuelva demasiado grande. Aunque el control derivativo no afecta en forma directa al error de estado estacionario, añade amortiguamiento al sistema y, por tanto, permite un valor más grande que la ganancia K, lo cual provoca una mejora en la precisión en estado estable.

* **PID: acción de control proporcional-integral-derivativa,** esta acción combinada reúne las ventajas de cada una de las tres acciones de control individuales. La ecuación de un controlador con esta acción combinada se obtiene mediante:

(7)

Y su función transferencia resulta:

(8)“[1]

# PROCEDIMIENTO

Materiales:

* 2 ruedas de arrastre
* 1 rueda loca
* 1 circuito puente H
* 8 Sensores de referencia Array QTR-8RC Reflectance
* 2 moto reductores
* 1 chasis auto inteligente
* Tarjeta de desarrollo Arduino Mega 2560



Fig.2. ruedas de arrastre

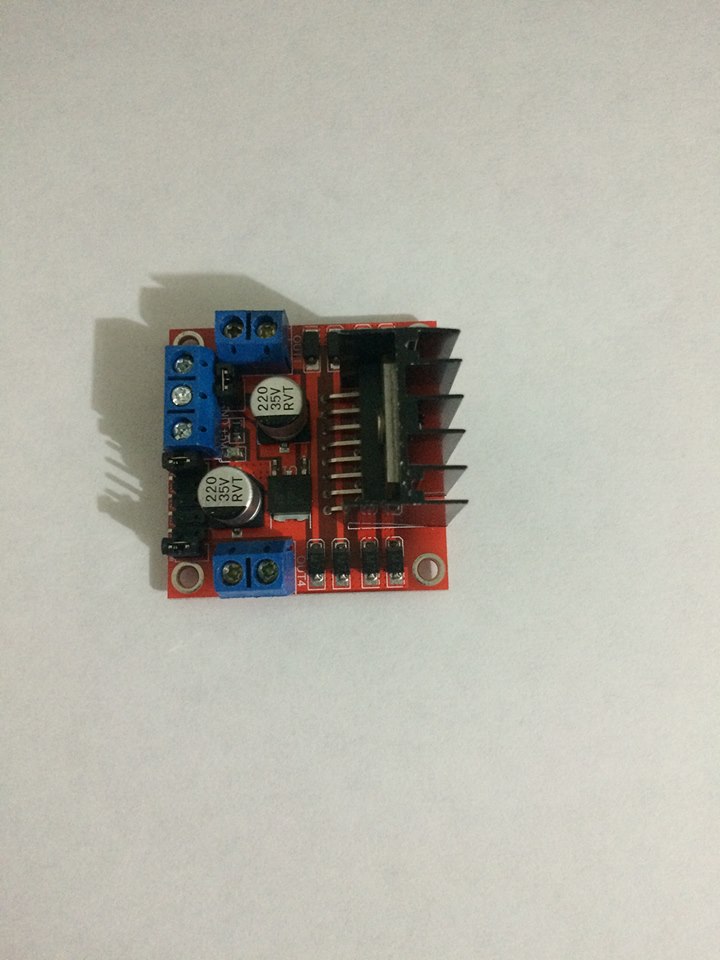


Fig.3.circuito puente H.

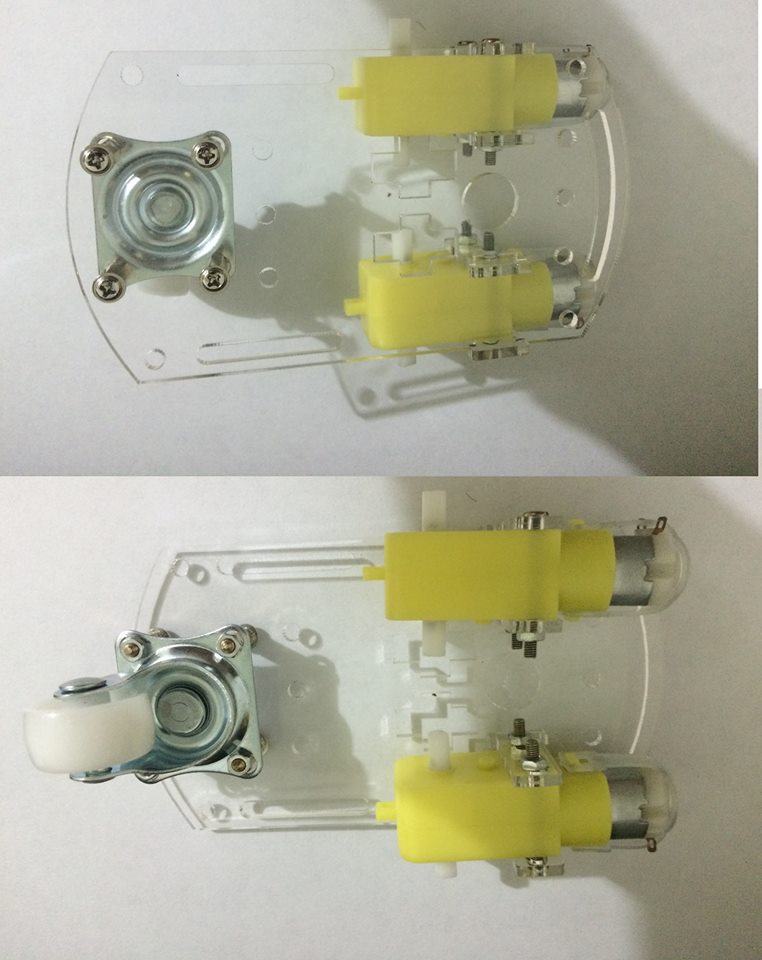


Fig.4. rueda loca, par de motores y base del chasis

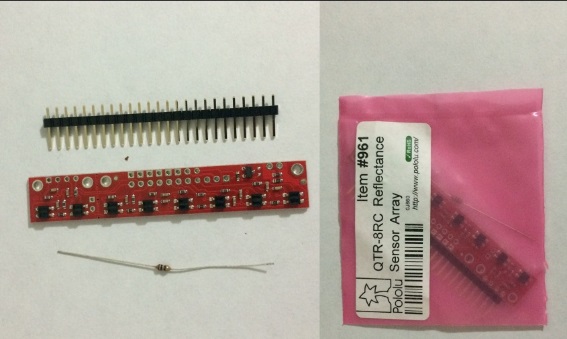


Fig.5. sensores para la detección de línea.



Fig.6. vehículo y diferentes materiales.

***Descripción del PID a emplear***

En primer lugar, definamos que es un controlador PID.

Entonces un PID es un controlador de uso genérico, el cual emplea realimentación de lazo cerrado, es un sistema al que le ingresa un error correspondiente a la diferencia entra la salida deseada (set point) y el valor de salida obtenido en la salida final del sistema. El controlador intentara en todo momento minimizar el error ajustando la entrada al sistema

El PID es un sistema al que le entra un **error** calculado a partir de la salida deseada menos la salida obtenida y su salida es utilizada como entrada en el sistema que queremos controlar. El controlador intenta minimizar el error ajustando la entrada del sistema.

Este controlador tiene implícito 3 parámetros y cada uno influye en mayor o menor medida sobre alguna característica de la salida ya sea el tiempo de establecimiento, sobre impulso, entre otros.

En este orden de ideas entonces los parámetros de los que se hablaba son

* Acción proporcional

Siendo la base de los tres parámetros, da una entrada de control proporcional con el error, usar solo este parámetro genera un error estacionario.

* Acción integral

Este da una suma de los errores previos a la entrada del sistema de control continuando hasta que el valor llegue al deseado, la mayoría de las veces el termino integral está asociado a un término proporcional, con lo cual se elimina el error estacionario.

* Acción derivativa

Este da una respuesta que es proporcional a la velocidad del cambio del error, con esta acción, y adicional a las demás se mejora la velocidad del sistema y contribuye a la disminución de oscilaciones excesivas.

Para el algoritmo de ejecución del control del PID junto con lo necesario para relacionar el sensor, los motores, el puente h y el arduino se hizo una representación gráfica de la figura 7 de los ANEXOS, donde vemos que algunas funciones hacen el llamado de otras para ejecutarse partiendo del encendido para finalmente realizar el proceso de control realizado para el seguidor de línea.

Para realizar el montaje del seguidor de línea , primero hizo una serie de pruebas las cuales fueron implementadas en el software de simulación de circuitos ISIS de PROTEUS , en ayuda de una librería de arduino , con lo cual se realizó la respectiva prueba de los motores y el puente h, de manera que se simulara el comportamiento de los sensores , y así saber si nuestros motores reaccionaban de manera correcta además de implementar un código en arduino, el cual está en prueba pues el total será mostrado al final del proyecto, para poder simular fue necesario el archivo en hexadecimal que contenía nuestro código base y con este así poder cargar al arduino simulado en ISIS , de manera que para nuestro código además se estableció de una vez el control pid.

Los logic states de la figura 10 de los ANEXOS son la representación de las lecturas que pueden ser obtenidas mediante el sensor QTR de 8 probabilidades, ya que, si este no refleja el valor será 1 y si refleja el valor es 0.

CODIGO EXPLICACION:

Para nuestro código (en ANEXOS), primero definimos una serie de macros que nos ayudarían a simplificar a la hora de implementar las variables, o si bien necesitamos definir nuestro número de sensores, el tiempo de ejecución de ellos, o simplemente definir la serie de pines tanto del puente h como de los sensores, además de definir nuestras constantes del PID, posterior a eso hicimos uso de unas funciones que la librería correspondiente a nuestros sensores QTRRC ya nos daba , primero partimos con los valores que nos llegaban de dichos sensores para así usar la función calibrate , la cual toma muestras de los sensores aproximadamente 10 veces ya dependiendo del valor de tiempo que uno establezca para la calibración en nuestro caso fueron 2,5 segundos, continuando en el void loop se creó una función llamada pid, la cual tendría como parámetros las constantes , la línea es decir la cinta negra y la velocidad, que también se estableció previamente además que esta no debería ser mayor a 255 que era el tope, para ejecutar la función pid , se hizo uso de la palabra reservada position ya que esta nos almacenaba el valor en tiempo real de los sensores, además de la función readline función que proviene de la librería de nuestros sensores QTR y esta se encarga de retornar un valor estimado, respecto a la posición en la que se encuentra el seguidor, tomando cada sensor y haciendo así un estimado; para aplicar el pid es necesario tener él cuenta el valor actual ya que si recordamos la acción proporcional depende del valor actual establecido en position, restado a nuestro set point , que es valor donde nuestros sensores por pruebas previas nos indicaron estar alineados o centrados, teniendo esto, se aplicó el PID para , dar un mejor manejo al comportamiento de nuestro carro, variando las constantes para verificar su comportamiento, hasta encontrar un valor adecuado.

seguido a esto se definió una función motores la cual era la que nos daba las direcciones de los motores es decir de acuerdo a si necesita girar hacia la izquierda o hacia la derecha, y con esto apagando un motor y prendiendo otro, por último se creó una función de frenos para que nuestro seguidor pudiera tomar las curvas, para esta función partimos del valor de la posición y se establecen unos if o condiciones, en caso que este salga por la derecha o por la izquierda , partiendo de valores que se tomaron con el sensor.

en donde si el valor es menor a 500 nos indica que salió por la parte derecha, pero si es un valor mayor a 6500 indica que salió por la parte izquierda; todo esto además con ayuda de una función propia de los sensores QTRRC la cual es read, que nos da la lectura del sensor en bruto. Y a su vez este valor se establece para nuestros dos primeros sensores es si es por la derecha, y para nuestros dos últimos si es por la izquierda, ya que nuestro arreglo de sensores comienza de mayor a menor en número es decir 7,6,5,4,3,2,1,0 completando los 8 sensores.

# conclusiones

* Se comprobó como el control PID mejoro nuestra respuesta, considerablemente, pues le daba mejor control al seguidor de línea, pero este no siempre mejoraba de acuerdo a los valores establecidos en nuestras constantes su respuesta podía llegar a ser peor también, en especial si se establecía una constante de integración muy grande.
* Se verifico el comportamiento de puente H al establecer el manejo de los motores ya que este, les daba la dirección deseada y los cambios necesarios, para que el carro girara, además de sumarle algo de potencia al giro de las llantas.
* Se implementó un vehículo seguidor de línea usando la plataforma de arduino y empleando un controlador PID permitiendo un movimiento acorde a las señales censadas y ejecutadas mediante la tarjeta de desarrollo ATMEGA 2560.
* Se construyó la estructura física del vehículo seguidor de línea de acuerdo a parámetros de diseño tenidos en cuenta.
* Se acondicionó la tarjeta de desarrollo al vehículo de forma que pueda ejercer la ejecución o intercambio de señales con el mismo, y los sensores que este tiene adaptados.
* Se diseñó el código base PID que permite realizar la ejecución del movimiento de la forma más estable y acorde posible al comportamiento esperado.

# REFERENCIAS

[1] Virginia Mazzone; Controladores PID; marzo 2002

**ANEXOS**

Algoritmo en lenguaje C utilizado para la implementación.

#include <QTRSensors.h>

#define NUM\_SENSORS 8 //numero de sensores usados

#define TIMEOUT 2500 // tiempo de espera para dar resultado en uS

#define EMITTER\_PIN 38 //pin led on

#define sensores 8

//motor izquierdo

#define IN1 5

#define IN2 6

//motor derecho

#define IN3 9

#define IN4 10

QTRSensorsRC qtrrc((unsigned char[]) {A0,A1,A2,A3,A4,A5,A6,A7} ,NUM\_SENSORS, TIMEOUT, EMITTER\_PIN);

//almacenamiento valores de sensores y posicion

unsigned int sensorValues[NUM\_SENSORS];

unsigned int position=0;

/// variables para el pid

int derivativo=0, proporcional=0, integral=0; //konstantes

int entrada\_control=0, proporcional\_pasado=0;

//parametros fisicos

int velocidad=90; //vmax es 255

float Kp=0.20, Kd=4,Ki=0.1;//4, Ki=0.001; //constantes

//variable para escoger el tipo de linea

int linea=0; // 0 para lineas negra, 1 para lineas blancas

void setup()

{

delay(1000);

pinMode(IN1, OUTPUT);//pin de direccion motor izquierdo

pinMode(IN3, OUTPUT);//pin de direccion motor derecho

for (int i = 0; i < 40; i++) //calibracion durante 2.5 segundos,

{ //poner sobre linea negra y sup blanca

delay(20);

qtrrc.calibrate(); //funcion para calibrar sensores

delay(20);

}

//apagar sensores para indicar fin

//de calibracion

delay(400);

}

void loop()

{

//pid(0, 120, 0.18, 4, 0.001);

pid(linea,velocidad,Kp,Ki,Kd); //funcion para secuencia pid

//frenos\_contorno(0,700);

frenos(600); //funcion para frenado en curvas tipo

//0 para lineas negras, tipo 1 para lineas blancas

//flanco de comparación va desde 0 hasta 1000 , esto para ver

//si esta en negro o blanco

}

////////funciones para el control del robot////

void pid(int linea, int velocidad, float Kp, float Ki, float Kd)

{

position= qtrrc.readLine(sensorValues, QTR\_EMITTERS\_ON, linea); //0 para linea

//negra, 1 para linea blanca

proporcional = (position) - 3500; // set point es 3500, valor deseado

integral=integral + proporcional\_pasado;

derivativo = (proporcional - proporcional\_pasado); //obteniedo el derivativo

if (integral>1000) integral=1000; //limites para no rebasamiento

if (integral<-1000) integral=-1000;

entrada\_control =( proporcional \* Kp ) +( derivativo \* Kd )+(integral\*Ki);

if ( entrada\_control > velocidad ) entrada\_control = velocidad; //limitamos la salida de pwm

if ( entrada\_control < -velocidad ) entrada\_control = -velocidad;

if (entrada\_control < 0)

{

motores(velocidad+entrada\_control, velocidad);//pon

}

if (entrada\_control >0)

{

motores(velocidad, velocidad-entrada\_control);//poniendo

}

proporcional\_pasado = proporcional;

}

///-------funcion motores modificada------

void motores(int motor\_izq, int motor\_der)

{

///motor izquierdo

if ( motor\_izq >= 0 ) //motor izquierdo avanza

{

//se controla de manera

//inversa para mayor control

analogWrite(IN1,255-motor\_izq);

digitalWrite(IN2,HIGH);

}

else //motor izquierdo retrocede

{

digitalWrite(IN1,LOW);

motor\_izq = motor\_izq\*(-1); //cambio de signo

analogWrite(IN2,motor\_izq);

}

///motor derecho

if ( motor\_der >= 0 ) //motor derecho avanza

{

analogWrite(IN3,255-motor\_der);

analogWrite(IN4,HIGH);

}

else //motor derecho retrocede

{

digitalWrite(IN3,LOW);

motor\_der= motor\_der\*(-1); //cambio de signo

analogWrite(IN4,motor\_der);

}

}

void frenos(int valor\_comparacion)

{

if (position<=500) //si se salio por la parte derecha de la linea

{

motores(60,-40); //debido a la inercia, el motor

//tendera a seguri girando

//por eso le damos para atras , para que frene

// lo mas rapido posible

while(1)

{

qtrrc.read(sensorValues); //lectura en bruto de sensor

if ( sensorValues[0]>valor\_comparacion || sensorValues[1]>valor\_comparacion )

//asegurar que esta en linea

{

break;

}

}

}

if (position>=6500) //si se salio por la parte izquierda de la linea

{

motores(-40,60);

while(1)

{

qtrrc.read(sensorValues);

if (sensorValues[7]>valor\_comparacion || sensorValues[6]>valor\_comparacion )

{

break;

}

}

}

}

C:\Users\Alejandro Marmolejo\Desktop\descarga.png

Fig.7. Representación en bloques del sistema a emplear

C:\Users\Alejandro Marmolejo\Desktop\descarga.png

Fig 8. Diagrama de flujo del algoritmo implementado

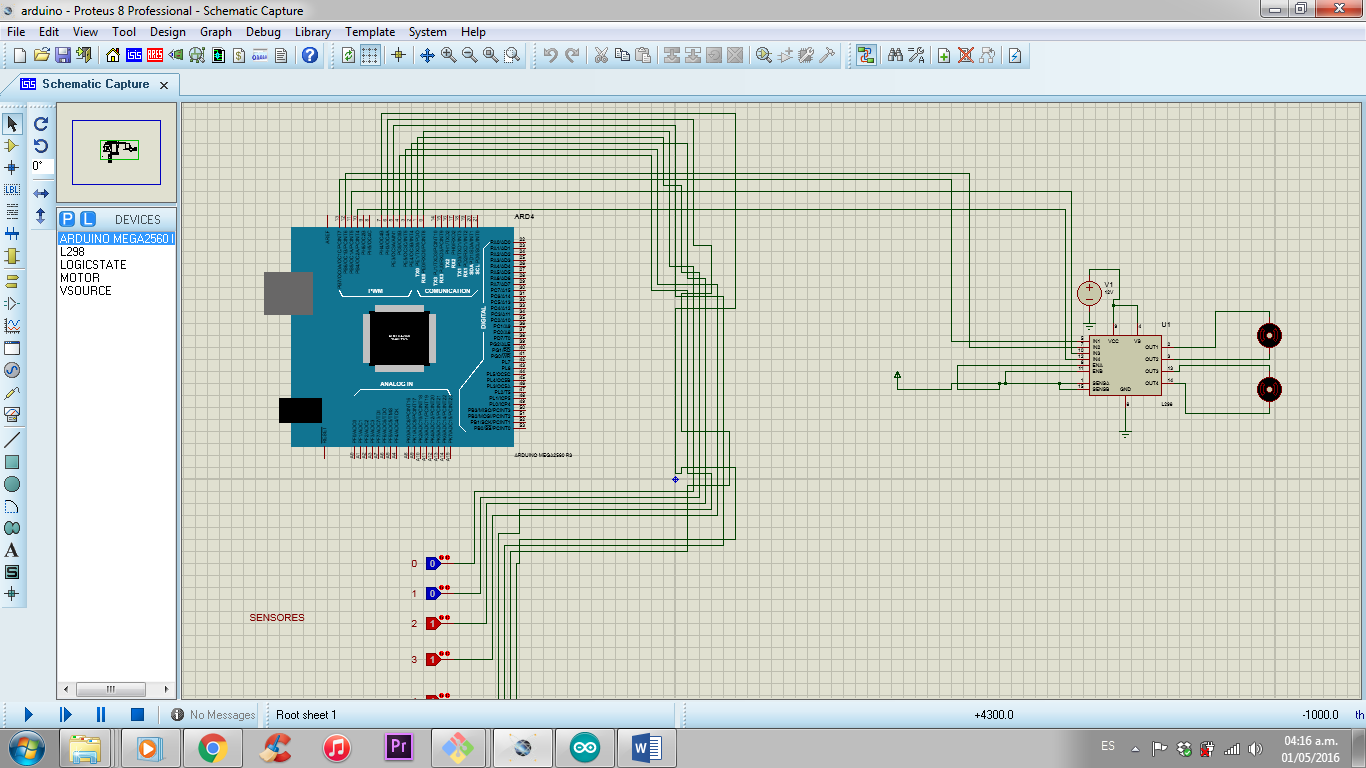


Fig.9. Implementación en proteus.

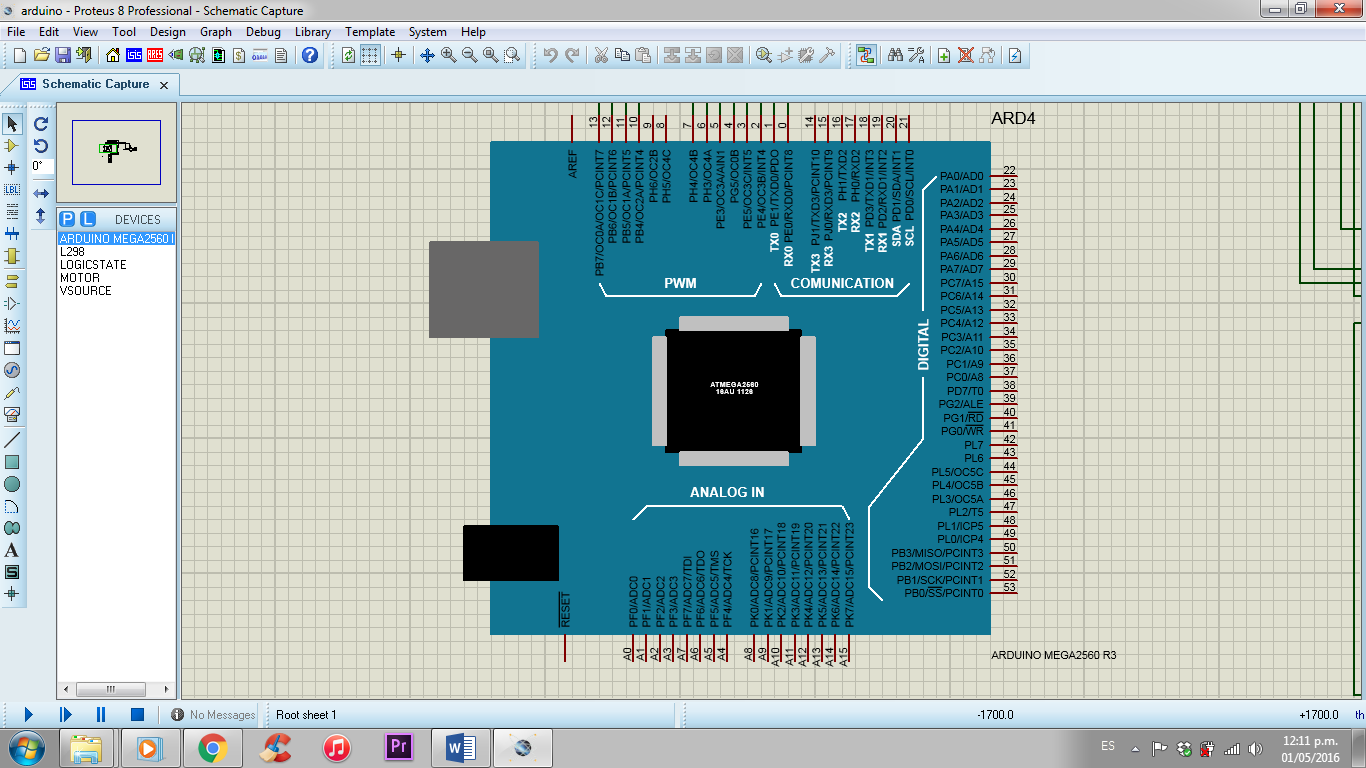


Fig. 10 Esquemático del arduino Mega2560.

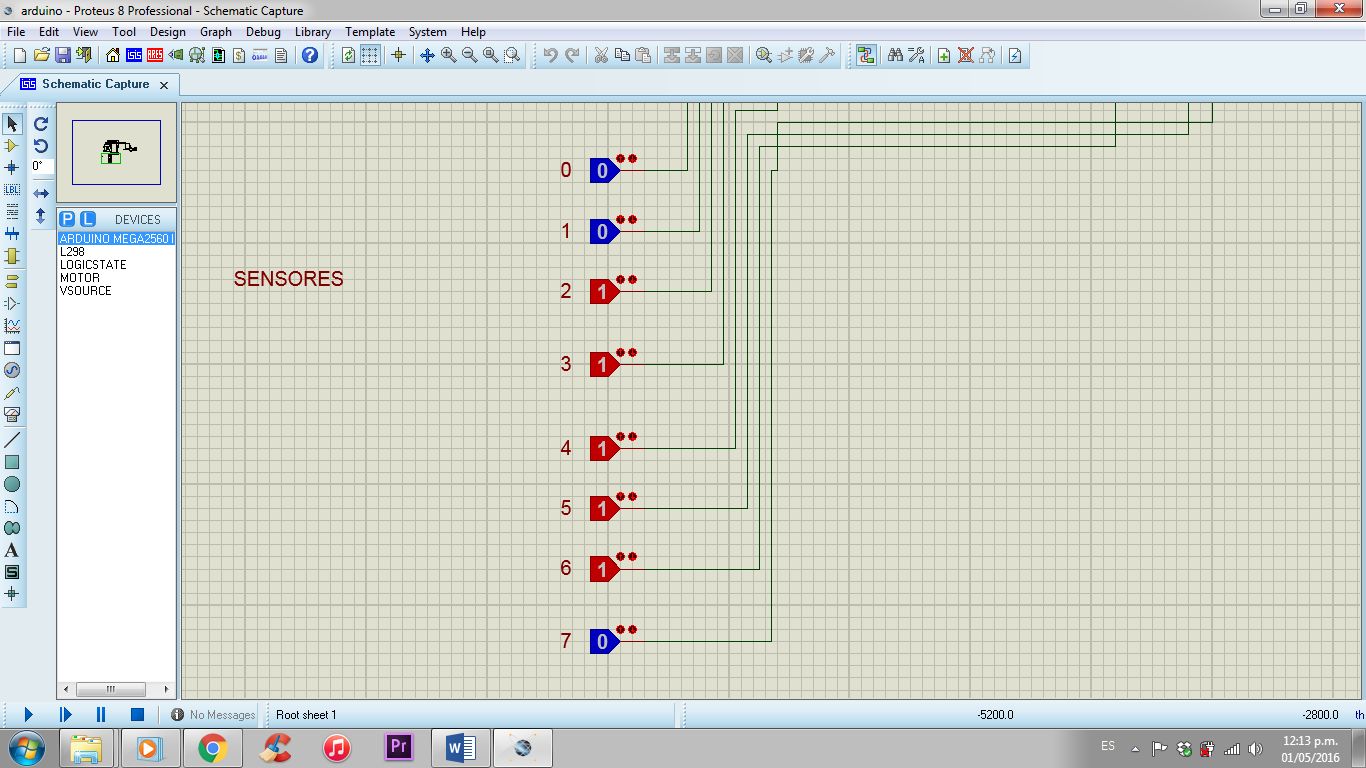


Fig. 11 Logic states

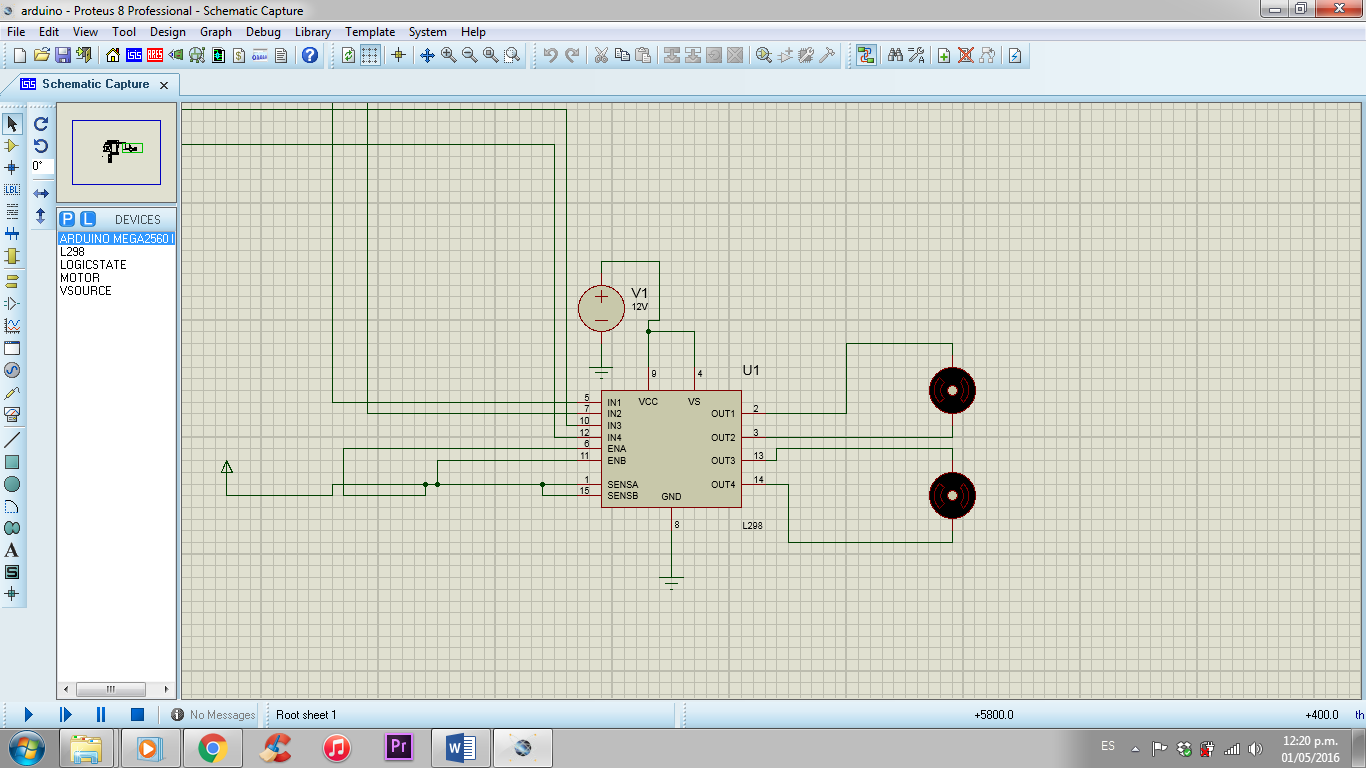


Fig. 12 Puente H y motores.