INFORME #1

SEGUIDOR DE LÍNEA

Arcos Cerón Jorge Alberto, Arcos Araujo Brayan

*Resumen*—En la presente informe se encuentra plasmado el diseño, simulación y correspondientes análisis de un carro seguidor de línea controlado por microcontrolador Atmega328P el cual se encuentra codificado mediante un control PID, para lo que se aplicara los conocimientos adquiridos en el curso en lo que al tema respecta, para así diseñar y verificar de manera correcta, la implementación, el funcionamiento y el análisis de dicho sistema.

Abstract

1. **INTRODUCCION**

Dentro de los [sistemas](http://www.monografias.com/trabajos11/teosis/teosis.shtml) mecatrónicos más interesantes se encuentran los robots, maquinas autónomas diseñadas para cumplir una tarea específica.

Para este reto construiremos un carro seguidor de línea negra con fondo blanco, controlado por un procesador Atmega328P incluido en un Arduino UNO, que trabajara junto a un controlador de motores conocido como Puente H de referencia L298N entre otros componentes descritos más adelante.

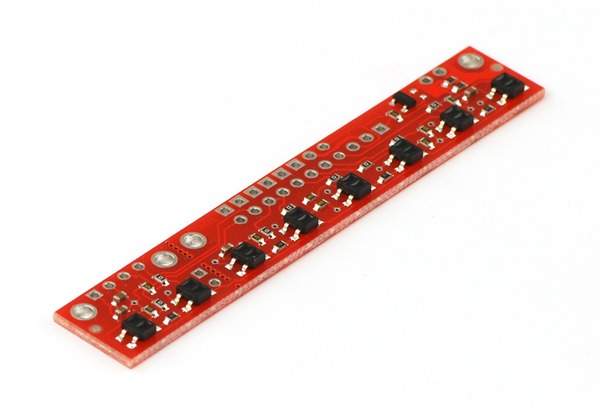
Con esto pretendemos incorporar los conceptos y bases teóricas adquiridas en el curso de Microprocesadores a la práctica.

1. **OBJETIVO**

* Diseñar e implementar un robot velocista seguidor de línea negra, controlado por un microcontrolador Atmega38P adjunto en una tarjeta Arduino UNO, programado en lenguaje C y cuyo funcionamiento este basado en un control PID.

1. **MATERIALES**

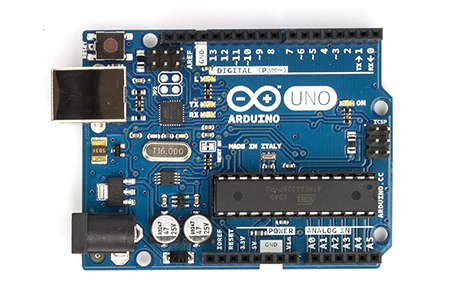
**ARREGLO DE SENSORES QTR-8RC**



Este arreglo consta de 8 sensores que utilizan pines digitales, aunque también se pueden utilizar en pines analógicos  pero previamente configurados como digitales. Es por ello que estos sensores  son muy versátiles y tienen mayor distancia de censado a comparación de los analógicos. El funcionamiento de estos sensores es a base de la carga y descarga de los condensadores que están en serie con los fototransistores, cuando este esté en una superficie blanca  el capacitor tendera a no retener carga  por lo tanto la salida será de 0 voltios, pero por el contrario, si se encuentra en una superficie oscura, el fototransistor no saturara,  de esta forma el capacitor retendrá carga, produciendo un voltaje positivo.

Para ello primero se debe configurar los pines a usar como salida, mandando un pulso de unos cuantos microsegundos, luego configurando como entrada los pines haciendo una lectura; esto continuamente en un ciclo, Es así como se puede saber si se está en blanco o negro. Pero, estos sensores ya cuentan con una librería en arduino que hace todo este procedimiento de forma automática, solo tendremos que declarar los pines que utilizaremos y haremos uso de las respectivas funciones de calibrado,  funciones de lectura de sensores y lecturas de posición, que es justamente lo que utilizaremos para nuestro robot velocista.

**PLACA ARDUINO UNO**



El Arduino UNO es una placa de desarrollo que integra un micro controlador ATmega328P, que será el cerebro de nuestro robot, pues será quien tome las decisiones para la dirección y velocidad de los motores de acuerdo a la lectura de los sensores que este mismo haga.

Esta placa posee un entorno de desarrollo (IDE) la cual usaremos para implementar el código que regirá el comportamiento de nuestro robot seguidor de línea.

Sus principales características son las siguientes:

* Microcontrolador ATmega328.
* Voltaje de entrada 7-12V.
* 14 pines digitales de I/O (6 salidas PWM).
* 6 entradas análogas.
* 32k de memoria Flash.
* Reloj de 16MHz de velocidad.

**KIT**

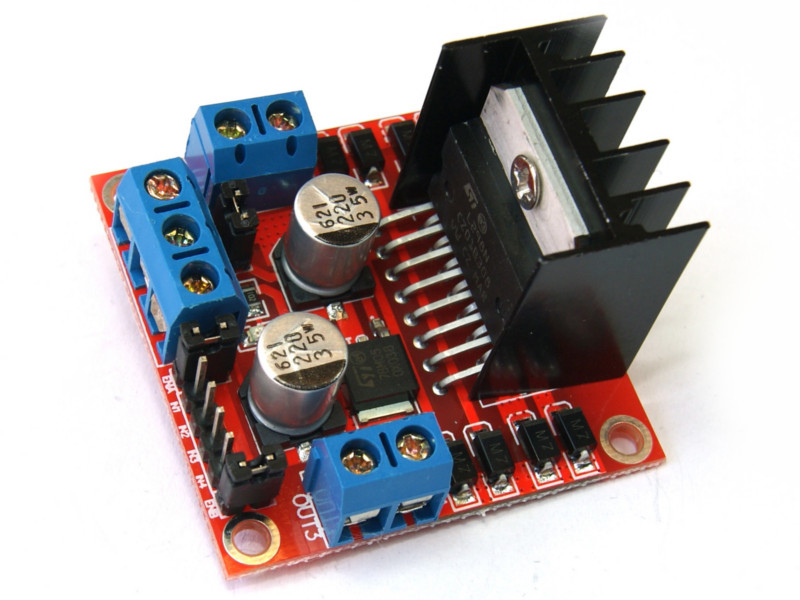


Usamos un kit que proporciona Pololu, ideal para nuestro proyecto.

Este kit consta de:

-Un chasis en acrílico  
-2 moto – reductores   
-2 Ruedas  
-4 Sujetadores   
-1 Rueda universal  
-Caja de la batería  
-Tornillos (8 largos, cortos 2)  
-10 Tuercas

**PUENTE H L298N**



Este módulo basado en un chip L298N nos permitirá controlar dos motores de corriente continua o un motor paso a paso bipolar de hasta 2 amperios, En nuestro caso serán dos moto-reductores de corriente continua.

Este módulo cuenta con todos los componentes necesarios para funcionar sin necesidad de elementos adicionales, entre ellos diodos de protección y un regulador LM7805 que suministra 5V a la parte lógica del integrado L298N. Cuenta con jumpers de selección para habilitar cada una de las salidas del módulo (A y B) estos descritos como IN1, IN2, IN3 e IN4 serán controlados por nuestro Arduino UNO. La salida A está conformada por OUT1 y OUT2 y la salida B por OUT3 y OUT4, Salidas que usaremos para nuestros dos moto-reductores.

1. **MONTAJE**

Realizando el respectivo y adecuado montaje de tal manera que se adapte a nuestra necesidad de un carro seguidor de línea obtuvimos el siguiente resultado:

// Imagen montaje

En el que organizamos el arreglo de sensores de tal manera que queden entre 3 y 5 mm del piso, para un correcto censado.

La rueda universal o rueda loca la ubicamos en la parte de trasera de nuestro carro con el fin de que este aborde las curvas de una manera más eficiente y controlada.

Ubicamos una pequeña protoboard en la parte delantera solamente para facilitar el manejo de un botón de arranque y para mejorar la visualización, aspecto y entendimiento entre la máquina y el usuario incluimos unos leds de colores que en la misma que nos indican el estado de nuestro robot seguidor de línea y la acción que este está realizando.

1. **CONEXIÓN**

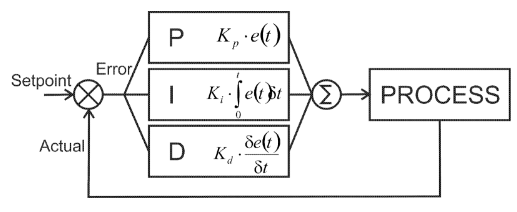
//Imagen de la coneccion y descripción de la utilización de //cada pin del arduino en cada coneccion.

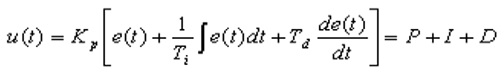
1. **MARCO TEÓRICO**

**CONTROL PID**

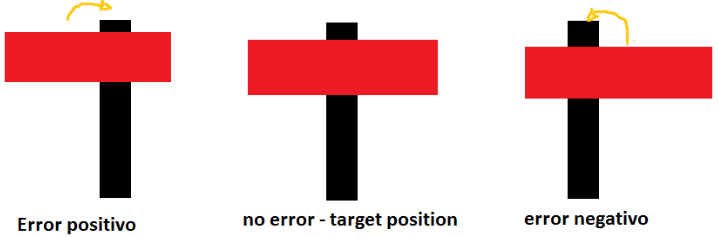
El Pid (control proporcional, integral y derivativo) es un mecanismo de control por realimentación que calcula la desviación o error entre un valor medido y el valor que se quiere obtener (set point, target position  o punto de consigna), para aplicar una acción correctora que ajuste el proceso.

En el caso nuestro robot velocista, el controlador PID ,(que es una rutina basada matemáticamente), procesara los datos del sensor, y lo utiliza para controlar la dirección (velocidad de cada motor), para de esta forma  mantenerlo  en curso.





Definimos como *Error* a la diferencia entre la posición objetivo y la posición medida del error, (Que tan lejos del punto de la línea se encuentra el sensor, en nuestro caso el objetivo es tener  los sensores centrados)



**Punto de consigna o Tarjeta Position:**Cuando el error es 0 (cero). En el caso del nuestro robot velocista, la idea es siempre mantenerlo en la línea, o en el caso de los sensores mantenerlos centrados para que así nuestro robot no se salga de la línea.

El funcionamiento del un Control PID consta principalmente de tres parámetros y tres constantes asociadas a los mismos, y cuyo objetico común es la disminución del error en la señal de salida.

Los parámetros son:

**Proporcional:** Es la respuesta  al error que se tiene que entregar de manera inmediata, es decir, si nos encontramos en el centro de la línea, los motores, tendrán en respuesta una velocidad de igual  valor, si nos alejamos del centro, uno de los motores reducirá su velocidad y el otro aumentara.

**Proporcional = (posición) - punto\_consigna**

**Integral:** La integral es la sumatoria de los errores acumulados, tiene como propósito el disminuir y eliminar el error en estado estacionario provocado por el modo proporcional, en otras palabras, si el robot velocista se encuentra mucho tiempo alejado del centro (ocurre muchas veces cuando se encuentra en curvas), la acción integral se ira acumulando e ira disminuyendo el error hasta llegar al punto de consigna.

**Integral=Integral + proporcional\_pasado**

**Derivativo:**Es la derivada del error, su función es mantener el error al mínimo, corrigiéndolo proporcionalmente con la mismo velocidad que se produce, de esta manera evita que el error se incremente, en otra palabra, anticipara la acción evitando así las oscilaciones excesivas.

**Derivativo=proporcional-proporcional\_pasado**

Y las constantes asociadas a estos parámetros conocidas como factores, son:

**Factor (Kp)**- Es un valor constante utilizado para aumentar o reducir el impacto de Proporcional. Si el valor es excesivo, el robot tendera a responder inestablemente, oscilando  excesivamente. Si el valor es muy pequeño, el robot responderá muy lentamente,  tendiendo a salirse de las curvas.

**Factor (Ki)**- Es un valor constante utilizado para aumentar o reducir el impacto de la Integral, El valor excesivo de este provocara oscilaciones excesivas, Un valor demasiado bajo no causara impacto alguno.

**Factor (Kd)**- Es un valor constante utilizado para aumentar o reducir el impacto de la Derivada. Un valor excesivo provocara  una sobre amortiguacion.  provocando inestabilidad.

**Salida\_pwm = ( proporcional \* Kp ) + ( derivativo \* Kd ) + (integral\*Ki);**

1. **FUNCIONAMIENTO**

El funcionamiento de nuestro robot seguidor de línea depende esencialmente de el algoritmo que implementemos en nuestro entorno de desarrollo que como antes mencionábamos usaremos el IDE de Arduino. Por ello es que está aquí el verdadero reto, la implementación del control pid, pues es aquí donde se tendrá que buscar las constantes que correspondan a las características físicas del robot.

Después de probar con diferentes recomendaciones propuestas en la red, concluimos que la forma más fácil de hacerlo es por el método del ensayo y el error, hasta obtener dichos valores con el resultado deseado en el comportamiento del robot.

Para esto existen algunas recomendaciones que nos resultaron muy útiles al buscar esas constantes, los pasos a seguir son:

1. Comience con Kp, Ki y Kd igualando 0 y trabajar con Kp primero. Pruebe  establecer Kp a un valor de 1 y observar el robot. El objetivo es conseguir que el robot siga la línea, incluso si es muy inestable. Si el robot llega más allá y pierde la línea, reducir el valor de Kp. Si el robot no puede navegar por una vez, o parece lenta, aumente el valor Kp.
2. Una vez que el robot es capaz de seguir un poco la línea, asignar un valor de 1 a Kd .Intente aumentar este valor hasta que vea menos oscilaciones.
3. Una vez que el robot es bastante estable en la línea siguiente, asigne un valor de 0,5 a 1,0 a Ki. Si el valor de Ki es demasiado alto, el robot se sacudirá izquierda y derecha rápidamente. Si es demasiado bajo, no se verá ninguna diferencia perceptible. El Integral es acumulativo por lo tanto  el valor Ki tiene un impacto significativo.  puede terminar  ajustando por 0,01 incrementos.
4. Una vez que el robot está siguiendo la línea con una buena precisión, se puede aumentar la velocidad y ver si todavía es capaz de seguir la línea. La velocidad afecta el controlador PID y requerirá re sintonizar como los cambios de velocidad.

Después de seguir estos pasos y lograr determinar los valores de las constantes del control PID, para los cuales el robot sigue la línea de manera ideal y a una velocidad considerable, haciendo uso de las librería de los sensores QTRsensores, logramos implementar el siguiente Algoritmo, en el cual hemos comentado la mayor parte de las líneas de código con el fin de que se logre entender el objetivo que hemos pretendido para el funcionamiento de nuestro robot seguidor de línea:

#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 A3 //pin led on

///////////////pines arduino a utilizar/////////////////////

#define led1 A0 // led rojo

#define led2 A1 // led verde

#define mot\_i 0 // -> IN2

#define mot\_d 5 // -> IN3

#define sensores A3

#define boton\_1 2 //pulsador

#define pin\_pwm\_i 10 //-> IN1

#define pin\_pwm\_d 11 // -> IN 4

QTRSensorsRC qtrrc((unsigned char[]) {6, 7, 8, 9, 1, 4, 12, 13}

,NUM\_SENSORS, TIMEOUT, EMITTER\_PIN);

//variables para almacenar 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; //parametros

int salida\_pwm=0, proporcional\_pasado=0;

//\_\_\_\_\_\_\_AQUI CAMBIEREMOS LOS PARAMETROS DE NUESTRO ROBOT\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

int velocidad=100; //variable para la velocidad, el maximo es 255

float Kp=0.18;

float Kd=2;

float Ki=0.01; //constantes

//variables para el control del sensado

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

int flanco\_color=0; // aumenta o disminuye el valor del sensado

int en\_linea=500; //valor en el cual se considerara si el sensor esta en linea o no

int ruido=50; //valor para el cual el valor del sensor es considerado como ruido

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

void setup()

{

delay(800);

pinMode(mot\_i, OUTPUT);//pin de direccion motor izquierdo

pinMode(mot\_d, OUTPUT);//pin de direccion motor derecho

pinMode(led1, OUTPUT); //led1

pinMode(led2, OUTPUT); //led2

pinMode(boton\_1, INPUT); //boton 1 como pull up

for (int i = 0; i < 50; i++) //calibracion durante 2.5 segundos,

{ //para calibrar es necesario colocar los sensores sobre la superficie negra y luego

digitalWrite(led1, HIGH); //sobre la blanca

delay(20);

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

digitalWrite(led1, LOW);

delay(20);

}

digitalWrite(led1, LOW); //apagar sensores para indicar fin

//de calibracion

delay(400);

digitalWrite(led2,HIGH); //encender led 2 para indicar la

// espera de pulsacion de boton

while(true)

{

int x=digitalRead(boton\_1); //leemos y guardamos el valor

// del boton en variable x

delay(100);

if(x==0) //si se presiona boton

{

digitalWrite(led2,LOW); //indicamos que se presiono boton

digitalWrite(led1,HIGH); //encendiendo led 1

delay(100);

break; //saltamos hacia el bucle principal

}

}

}

void loop()

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

  pid(linea,velocidad,Kp,Ki,Kd); //funcion para algoritmo pid  
                        //(tipo,flanco de comparacion)  
  
                    //frenos\_contorno(0,700);

  frenos\_contorno(linea,700); //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////

//aqui esta modificado la funcion del pid para que reciba los nuevos parametros para la libreria modificada

void pid(int linea, int velocidad, float Kp, float Ki, float Kd,int flanco\_color, int en\_linea,int ruido)

{

position = qtrrc.readLine(sensorValues, QTR\_EMITTERS\_ON, linea,flanco\_color, en\_linea, ruido ); //0 para linea

//negra, 1 para linea blanca

proporcional = (position) - 3500; // set point es 3500, asi obtenemos el error

integral=integral + proporcional\_pasado; //obteniendo integral

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

if (integral>1000) integral=1000; //limitamos la integral para no causar problemas

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

salida\_pwm =( proporcional \* Kp ) + ( derivativo \* Kd )+(integral\*Ki);

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

if ( salida\_pwm < -velocidad ) salida\_pwm = -velocidad;

if (salida\_pwm < 0)

{

motores(velocidad+salida\_pwm, velocidad);

}

if (salida\_pwm >0)

{

motores(velocidad, velocidad-salida\_pwm);

}

proporcional\_pasado = proporcional;

}

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

{

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

{

digitalWrite(mot\_i,HIGH); // con high avanza

analogWrite(pin\_pwm\_i,255-motor\_izq); //se controla de manera

//inversa para mayor control

}

else

{

digitalWrite(mot\_i,LOW); //con low retrocede

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

analogWrite(pin\_pwm\_i,motor\_izq);

}

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

{

digitalWrite(mot\_d,HIGH);

analogWrite(pin\_pwm\_d,255-motor\_der);

}

else

{

digitalWrite(mot\_d,LOW);

motor\_der= motor\_der\*(-1);

analogWrite(pin\_pwm\_d,motor\_der);

}

}

void frenos\_contorno(int tipo,int flanco\_comparacion)

{

if(tipo==0)

{

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

{

motores(-80,90); //debido a la inercia, el motor

//tendera a seguri girando

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

// lo mas rapido posible

while(true)

{

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

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

//asegurar que esta en linea

{

break;

}

}

}

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

{

motores(90,-80);

while(true)

{

qtrrc.read(sensorValues);

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

{

break;

}

}

}

}

}