IMPLEMENTACIÓN DE UN CONTROLADOR PID EN LA PLATAFORMA ARDUINO UNO.

Presentado como requisito de la asignatura micro controladores, dictada en la Universidad del Quindío durante el primer semestre del año 2016.

PRESENTADO POR:

SANTIAGO RESTREPO RUIZ

RONALD ANDRÉS MORA

NICOLÁS GUTIERREZ GONZALEZ

PRESENTADO A:

LUIS MIGUEL CAPACHO V.

GERARDO ANDRÉS LÓPEZ O.

UNIVERSIDAD DEL QUINDÍO

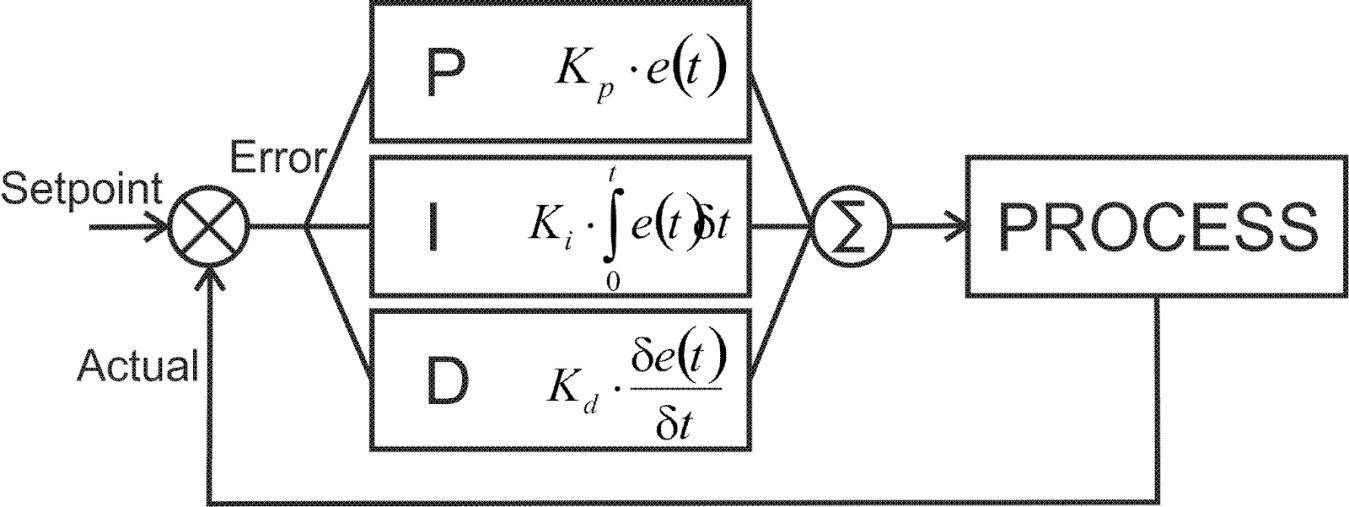
FACULTAD DE INGENIERÍA

PROGRAMA DE INGENIERÍA ELECTRÓNICA

ARMENIA, QUINDÍO EN COLOMBIA.

MES DE ABRIL DE 2016.

Un controlador PID (Proporcional Integrativo Derivativo) es un mecanismo de control genérico sobre una realimentación de bucle cerrado, ampliamente usado en la industria para el control de sistemas. En otras palabras, es un mecanismo de control por realimentación que calcula la desviación o error entre un valor medido y el valor que se quiere obtener para aplicar una acción correctora que ajuste el proceso.



En nuestro caso, el controlador PID procesa los datos del sensor, y lo utiliza para controlar la dirección (velocidad de cada motor), para de esta forma mantener el robot seguidor de línea en curso.

La ecuación general de un controlador PID está dada por:



Ahora, podemos definir cada uno de los conceptos que están involucrados en este controlador comenzando con uno de los más importantes, el error:

* ***Error*** - Llamamos a la diferencia entre la posición objetivo y la posición medida del error. (que tan lejos del punto de consigna se encuentra el sensor, en nuestro caso el objetivo es tener los sensores centrados)
* ***Target Position*** – Se presenta cuando el error es 0 (cero). Para nuestro robot seguidor de línea, la idea es siempre mantenerlo en la línea.

Como ya sabemos, el controlador PID se encuentra constituido por 3 parámetros:

* **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.
* **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 seguidor de línea se encuentra mucho tiempo alejado del centro, la acción integral se ira acumulando e ira disminuyendo el error hasta llegar al punto de consigna o target position.
* **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, osea que anticipará la acción evitando así las oscilaciones excesivas.

A partir de estos parámetros, se tendrán entonces que definir las constantes para cada uno de ellos:

* **Factor (Kp)** - Es un valor constante utilizado para aumentar o reducir el impacto de Proporcional. Si el valor es excesivo, el robot tendera 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 amortiguación. provocando inestabilidad.

La implementación del controlador PID en la placa se realizará de la siguiente manera:

* Definimos las librerías para el manejo de la placa de sensores QTR-8RC y los demás pines que vamos a usar:

|  |
| --- |
| #include <QTRSensors.h>  #define NUM\_SENSORS 8  #define TIMEOUT 2000  #define EMITTER\_PIN 6  #define led1 13  #define led2 4  #define mot\_i 7  #define mot\_d 8  #define sensores 6  #define boton\_1 2  #define pin\_pwm\_i 9  #define pin\_pwm\_d 10  QTRSensorsRC qtrrc((unsigned char[]) {19, 18, 17, 16,15,14,11,12},NUM\_SENSORS, TIMEOUT, EMITTER\_PIN); |

* Luego, definimos las variables de los sensores, posición, para el PID, las constantes y la velocidad de los motores de nuestro sistema así:

|  |
| --- |
| unsigned int sensorValues[NUM\_SENSORS];  unsigned int position=0;  int derivativo=0, proporcional=0, integral=0;  int salida\_pwm=0, proporcional\_pasado=0;  int velocidad=120;  float Kp=0.18, Kd=4, Ki=0.001;  int linea=0; // 0 para lineas negra, 1 para lineas blancas |

* Ahora, escribimos el setup, en el cual encontramos que se ejecuta la función de calibración de los sensores y una vez terminado esto se inicia el bucle principal:

|  |
| --- |
| void setup()  {  delay(800);  pinMode(mot\_i, OUTPUT);  pinMode(mot\_d, OUTPUT);  pinMode(led1, OUTPUT);  pinMode(led2, OUTPUT);  pinMode(boton\_1, INPUT);    for (int i = 0; i < 40; i++) {  digitalWrite(led1, HIGH);  delay(20);  qtrrc.calibrate();  digitalWrite(led1, LOW);  delay(20);  }  digitalWrite(led1, LOW);  delay(400);  digitalWrite(led2,HIGH);    while(true)  {  int x=digitalRead(boton\_1);  delay(100);  if(x==0)  {  digitalWrite(led2,LOW);  digitalWrite(led1,HIGH);  delay(100);  break;  }  }  } |

* Procedemos entonces a escribir el main, en donde ejecutaremos las funciones de control del seguidor de línea:

|  |
| --- |
| void loop()  {  pid(linea,velocidad,Kp,Ki,Kd);    frenos\_contorno(linea,700);  } |

* Finalmente escribimos las funciones que vamos a usar, algunas obtenidas directamente de la librería de la placa de sensores:

|  |
| --- |
| void pid(int linea, int velocidad, float Kp, float Ki, float Kd)  {  position = qtrrc.readLine(sensorValues, QTR\_EMITTERS\_ON, linea);    proporcional = (position) - 3500;  integral=integral + proporcional\_pasado;  derivativo = (proporcional - proporcional\_pasado);  if (integral>1000) integral=1000;  if (integral<-1000) integral=-1000;  salida\_pwm =( proporcional \* Kp ) + ( derivativo \* Kd )+(integral\*Ki);    if ( salida\_pwm > velocidad ) salida\_pwm = velocidad;  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 )  {  digitalWrite(mot\_i,HIGH);  analogWrite(pin\_pwm\_i,255-motor\_izq);  }  else  {  digitalWrite(mot\_i,LOW);  motor\_izq = motor\_izq\*(-1);  analogWrite(pin\_pwm\_i,motor\_izq);  }  if ( motor\_der >= 0 )  {  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) {  motores(-80,90);  while(true)  {  qtrrc.read(sensorValues);  if ( sensorValues[0]>flanco\_comparacion || sensorValues[1]>flanco\_comparacion )  {  break;  }  }  }  if (position>=6500) {  motores(90,-80);  while(true)  {  qtrrc.read(sensorValues);  if (sensorValues[7]>flanco\_comparacion || sensorValues[6]>flanco\_comparacion )  {  break;  }  }  }  }  if(tipo==1)  {  if (position<=500)  {  motores(-80,90);  while(true)  {  qtrrc.read(sensorValues);  if ( sensorValues[0]<flanco\_comparacion || sensorValues[1]<flanco\_comparacion )  {  break;  }    }  }  if (position>=6500)  {  motores(90,-80);  while(true)  {  qtrrc.read(sensorValues);  if (sensorValues[7]<flanco\_comparacion || sensorValues[6]<flanco\_comparacion)  {  break;  }  }  }  }  } |

Una vez implementado el controlador en la plataforma de Arduino, procedemos a sintonizarlo. En este proceso se tendrán que buscar las constantes adecuadas que correspondan a las características físicas del seguidor de línea. La forma más sencilla es por ensayo y error, debido a que modelar matemáticamente las constantes a partir de las características físicas del robot se hace muy difícil. Podemos determinar el valor de las constantes de la siguiente manera:

Empezamos entonces con kp y dejamos las otras constantes en cero, estableciendo kp a un valor de 1 y observamos el comportamiento del 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, se debe reducir el valor de Kp. Si el robot parece lento entonces debemos aumentar el valor de kp.

Una vez que el robot es capaz de seguir un poco la línea, debemos asignar un valor de 1 a Kd, e intentaremos reducir las oscilaciones.

Una vez que el robot es bastante estable en la línea siguiente, asignamos 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 y es recomendable ajustar a 0,01 incrementos.

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.

