AUTO VELOCISTA

PRESENTADO POR:

DAVID CAMILO MUÑOZ

PRESENTADO A:

ING. VLADIMIR TRUJILLO

UNIVERSIDAD DEL CAUCA
FACULTAD INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES
LABORATORIO II ELECTRÓNICA
POPAYÁN CAUCA
2019

I. INTRODUCCIÓN

El robot seguidor de línea es una entrada al mundo de la robótica donde se le permiten a las personas tener la libertad de diseñar, controlar, manejar y adecuar según las necesidades que se consideren, estas máquinas se encargan de facilitar o incluso realizar completamente las tareas humanas de riesgo o peligro. Nuestro proyecto presenta un robot seguidor de línea además velocista que nos ayuda acercarnos a la programación y electrónica, tratando de calcular bien los componentes y aplicando teoría de Control vista en clase. Posteriormente se explicará brevemente qué tecnologías decidimos usar, cómo se van a utilizar y qué problemas encontramos al implementarlas. Por último se analizará el funcionamiento del robot como afronta los retos, toma decisiones y que controlador usamos. Este punto nos llevará al montaje final del robot, al que hemos bautizado con el nombre de **McQueen**.

II. SELECCIÓN DEL CIRCUITO Y EXPLICACIÓN DE FUNCIONAMIENTO

SELECCIÓN DEL CIRCUITO

Figura 1. Carro Velocista

Para este proyecto se eligió diseñar en circuito impreso para no tener que lidiar con circuitería suelta y tener una mejor presentación, se escogió un diseño liviano para mejorar la aerodinámica y por lo tanto poder entregar más velocidad a los motores, los motores que se escogieron son de 10:1 HP, se decide trabajar con Arduino Nano que nos ofrece la funcionalidad de su tamaño adecuado para el proyecto y brindándonos aun así todas las características necesarias de cualquier otra referencia de Arduino. Para el buen agarre del carro en las curvas se decidió invertir en unas buenas llantas de goma que mejoren el agarre tratando de evitar que nuestro carro se salga de su trayectoria.

FUNCIONAMIENTO

El funcionamiento del robot velocista se basa en los **sensores QTR-8A** que deben ser ajustados a no más de 5 mm de la pista para que los sensores tengan un buen rastreo de la línea, para comprobar el buen funcionamiento de los sensores previamente se deben **calibrar** durante algunos segundos para que la librería de sensores procese los datos y muestre los valores correctos se realiza de la siguiente manera:

1. Conexión física de los sensores al arduino

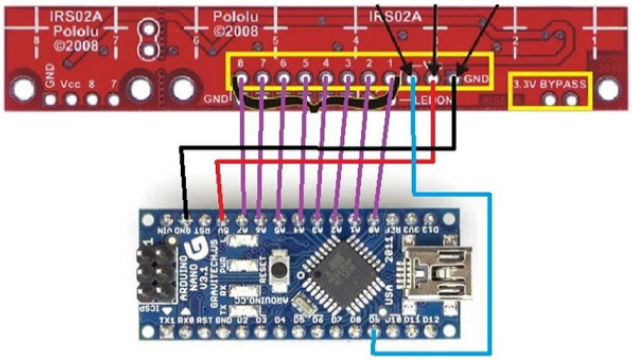

Figura 2. Conexión sensores a Arduino

2. Pasar repetitivamente los sensores sobre la línea negra y luego sobre la superficie blanca sin levantar los sensores
3. Esta calibración se debe realizar a una distancia constante de la superficie.

Se usa la librería de QTR Sensor se obtienen los datos de los sensores que son las primeras ocho columnas, la novena columna son los datos de la posición relativa del sensor que van es de 0 hasta 7000 donde el centro se encuentra en 3500 este valor es muy útil a la hora de sintonizar el controlador PD.
Podemos concluir que cuando los sensores están en la línea negra los valores del sensado aumentan alrededor de 800 a 1000 y cuando están en la superficie blanca los valores son 0. Abrimos el monitor serial de arduino y obtuvimos lo siguiente:

0	0	0	0	0	830	843	2	5503
0	0	0	0	0	827	840	0	5503
0	0	0	0	0	833	846	7	5503
0	0	0	0	0	827	840	0	5503
0	0	0	0	0	827	840	2	5503
2	0	0	3	14	830	843	5	5503
0	0	0	0	0	827	840	2	5503
0	0	0	0	0	830	840	2	5502
0	0	0	0	0	827	840	2	5503
0	0	0	0	0	830	846	2	5504
0	0	0	0	0	827	840	0	5503
0	0	0	0	0	827	840	2	5503
0	0	0	0	0	827	846	2	5505
0	0	0	0	0	827	840	2	5503
0	0	0	0	0	827	840	2	5503
5	0	0	6	28	833	846	7	5503
0	0	0	0	0	827	840	0	5503
0	0	0	0	0	827	840	2	5503

Figura 3. Monitor Serial Arduino librería QTR SENSOR

Para manejar las oscilaciones del robot y asegurarnos que sigue la línea de una manera adecuada decidimos implementar el **controlador PD** (control proporcional, integral y derivativo) este sistema lo que hace es calcular el error entre un valor medido y el valor que se quiere obtener para aplicar una acción. Este controlador procesa los datos del sensor y lo utilizara para controlar la dirección (velocidad de cada motor) para de esta forma mantenerlo en la trayectoria.

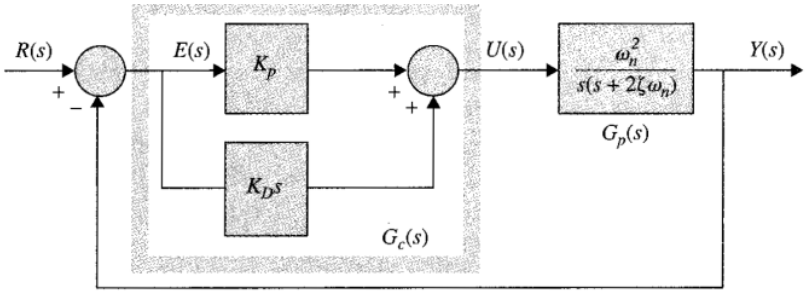

Figura 4. Control PD

- Error: Diferencia entre la posición del objeto y la posición medida del error.

Figura 5. Error según la posición.

Lo que buscamos es que el error sea cero (0) en el caso de nuestro robot esto sería cuando nuestros sensores marquen 3500. Los parámetros para nuestro controlador nos ayudan a alcanzar el equilibrio de nuestro robot por eso es importante que conozcamos cómo funcionan.

PROPORCIONAL: Es la respuesta al error que se tiene que entregar de manera inmediata, cuando los sensores da una lectura de que nos encontramos por el centro de la línea los motores tendrán una velocidad igual, si nos alejamos del centro uno de los motores reducirá su velocidad y el otro la aumenta.

DERIVATIVO: Trata de mantener el error al mínimo, corrigiendo proporcionalmente a la misma velocidad que se produce, evitando las oscilaciones excesivas.

Para garantizar un buen funcionamiento de los parámetros del controlador necesitamos asegurarnos de hallar sus constantes de forma correcta ya que esto podría afectar el funcionamiento del robot.

Factor (KP): Si el valor es excesivo el robot será muy inestable oscila excesivamente, si el valor es muy pequeño el robot responderá muy lentamente tendiendo a salirse en las curvas..
Factor (KD): Un valor excesivo en esta constante provocará una sobre amortiguación provocando la inestabilidad del sistema.

III. CÁLCULO DE COMPONENTES

Para el cálculo de nuestros elementos a usar en el proyecto necesitamos encontrar de manera experimental los valores para nuestro controlador

SINTONIZACION DEL PD
Como anteriormente lo habíamos mencionado la mejor forma para hacerlo para este caso de robot velocista es por ENSAYO Y ERROR, hasta obtener los resultados deseados.

1. Primero igualamos las constantes Kp y Kd a cero y empezamos a variar el valor de Kp empezando desde 1, el primer objetivo es que el robot siga la línea incluso si da muchas oscilaciones. La recomendación es que si el robot no sigue la línea reduce el valor de Kp y si no anda el robot aumenta ese valor.
2. Luego de esto aumentamos a 1 el valor de Kd hasta que no se viera tan inestable osea que no tuviera tantas oscilaciones al andar.
3. Por último después de tener estos valores bien calculados procedemos a variar la velocidad a nuestro velocista.

IV. JUSTIFICACIÓN DE LOS ELEMENTOS

- **Motores eléctricos:** Se encargan de proporcionar movilidad y velocidad requerida para que el robot termine un recorrido específico durante un tiempo establecido. En este caso, se emplearon dos motores marca Pololu, los cuales tienen una velocidad de seis mil rpm y un torque de 0.5 kg.cm[2]

Figura 6. Motores Pololu

❖ **Batería:** es la fuente de voltaje utilizada para energizar toda la circuitería que el robot utiliza para su funcionalidad. [2]

Figura 7. Batería Litio

❖ **Llantas de caucho (silicón):** Pieza circular mediante la cual los motores eléctricos ejercen tracción sobre la superficie en contacto, brindándole capacidad de movimiento. Tienen un mejor agarre y el rendimiento en velocidad del robot se ve incrementado en un 40 %, los aros están hechos de nylon, por lo que el peso es muy liviano y cae ideal para los robots velocistas.

Figura 8. Llantas de Caucho Silicon

❖ **Tornillos y tuercas:** son elementos mecánicos que permiten la fijación temporal de piezas entre sí.

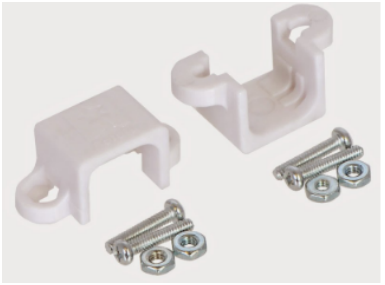

Figura 9. Tornillos y tuercas.

- **Arduino (**nano): Bueno en teoría se podria utilizar cualquier arduino, pero lo recomendable para un robot velocista de competencias (que por lo general son muy livianos) es utilizar un arduino nano o mini pro.

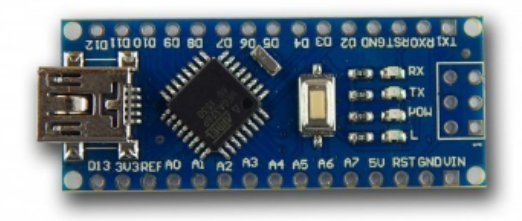

Figura 10.Arduino NANO.

- **Driver de motor o Puente H:** Es un circuito eléctrico que a través de éste, suministra la potencia necesaria para que el robot realice el recorrido a menor tiempo, El driver para motores TB6612FNG posee dos puentes H, puede controlar hasta dos motores de DC con una corriente costante de 1.2A (3.2A pico). Dos señales de entrada (IN1 y IN2) pueden ser usadas para controlar el motor en uno de cuatro modos posibles: CW(giro en sentido de las manecillas del reloj), CCW (en contra de las manecillas), short-brake y stop. Las dos salidas de motores (A y B) pueden ser controladas de manera separada, la velocidad de cada motor es controlada mediante una señal PWM con una frecuencia de hasta 100kHz. El pin STBY cuando es puesto en HIGH coloca al motor en modo de standby. El driver posee diodos internos de protección. [2]

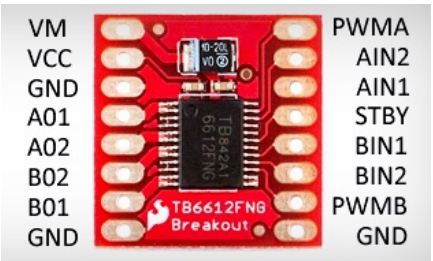

Figura 11.Puente H.

- **Conjunto de sensores de reflectancia QTR-8A:** Este módulo de sensor tiene 8 pares de IR LED / fototransistor montados en un tono de 0,375 ", lo que lo convierte en un excelente detector para un robot de seguimiento de línea. Los pares de LED están dispuestos en serie para reducir a la mitad el consumo de corriente, y un MOSFET permite que los LED se giren apagado para opciones adicionales de detección o ahorro de energía. Cada sensor proporciona una salida de voltaje analógica separada. [3]

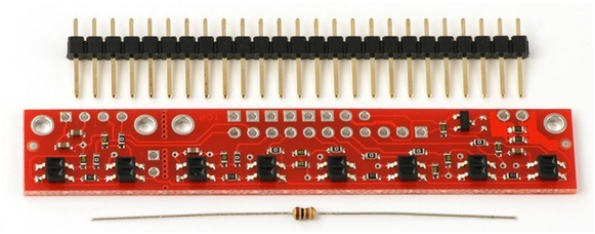

Figura 12.Sensores QTR-8A.

- **Circuito Impreso y chasis:** Se diseñó un chasis ligero y circuito impreso empleando el software de circuitos impresos PCB EAGLES, distribuyendo eficientemente los componentes, para que el centro de masa le proporcione estabilidad al robot, especialmente en las curvas. Se desarrolló el circuito impreso sobre el chasis, evitando el exceso de cableado, brindándole mayor presentación al robot.

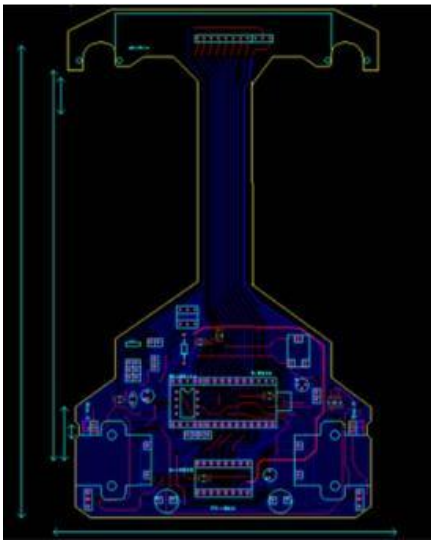

Figura 13. PCB carro velocista.

V. PLAN ELÉCTRICO PIN A PIN

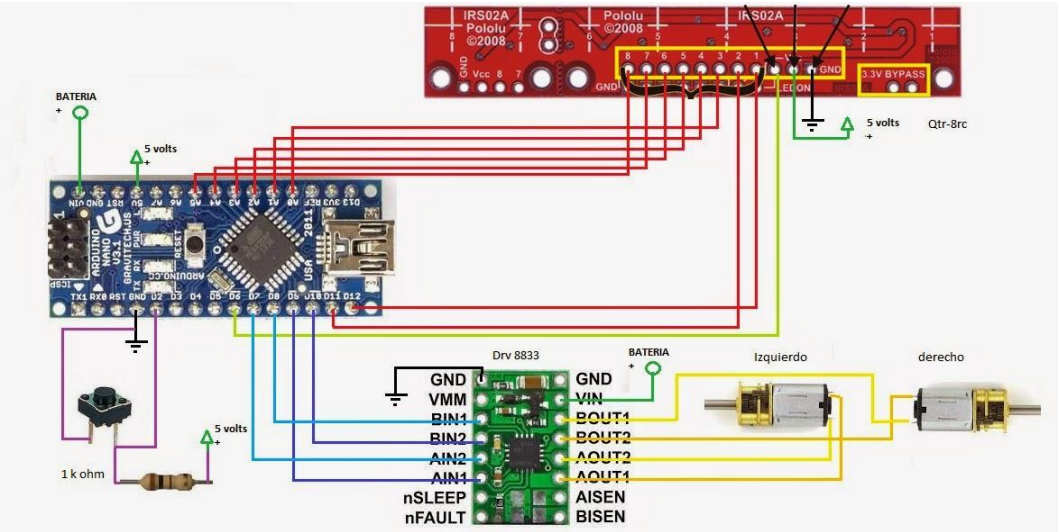

Figura 14. Diagrama circuital carro velocista

❖ ESPECIFICACIONES:

Para montar nuestro proyecto de velocista las conexiones que se hicieron se encuentran en el anterior diagrama circuital. Primero definimos las características físicas del robot:

Velocidad del robot: El robot utiliza motores de alta velocidad son de la marca pololu de 10:1 con los cuales se lograra alcanzar una buena velocidad para que el carro siga la línea y no se salga.
Peso del Robot: Es de 200g, es muy ligero para alcanzar la velocidad que deseemos.
Capacidad de respuesta a los cambios de trayectoria: El robot utiliza el arreglo lineal de ocho sensores infrarrojos QTR-8A colocados en la parte frontal con el propósito de tener mejores medidas y anticipar los cambios de trayectoria.

En la etapa de construcción y ensamblado se procedió abrir los orificios conforme al diseño para el montaje y la conexión de los componentes electrónicos:

1. En la parte frontal conectamos la regleta de sensores asegurándonos que no quedaran a más de 5mm de distancia del suelo.
2. Montamos los motores a los lados del chasis asegurándolos con tornillos y tuercas
3. Instalamos las llantas asegurándolas de forma correcta.
4. Instalamos el puente H y el arduino.

5. Finalmente conectamos la batería a todo el circuito y ya queda el montaje de nuestro robot velocista.

El siguiente paso y muy importante es la programación, se implementa un código sencillo que se explicará a continuación.

❖ PROGRAMACIÓN DEL MICROCONTROLADOR:

1. Primero que todo declaramos las librerías que utilizaremos, seguidamente las variables a utilizar con sus respectivos pines. Para el controlador PD utilizaremos 2 variables y otra adicional que es Ki pero siempre estuvo en cero (0) estas variables se podrán modificar para mejorar el desempeño del carro y una variable de velocidad que me se ajusta a la velocidad deseada.

```
#include <QTRSensors.h>
#include <SoftwareSerial.h>
//Mapeo de pines
#define STBY 6
#define AIN1 5
#define AIN2 4
#define PWMB 9
#define PWMA 3
#define BIN2 8
#define BIN1 7
#define NUM_SENSORS             6
#define NUM_SAMPLES_PER_SENSOR  5
#define EMITTER_PIN            11
#define LED                    13

#define  pulsador  12     // pin usado para el pulsador para configurar sensores

// Constantes para PID
float KP = 1.45;  // variables a cambiar  para mejorar desempeño
float KD = 1;   // variables a cambiar  para mejorar desempeño
float Ki = 0;

// Regulación de la velocidad Máxima
int Velmax = 120;   // variables a cambiar  para mejorar desempeño
```

Figura 15.Código Arduino Velocista

2. Declaramos las variables que utilizaremos para la calibración de los sensores.

```
byte estado = 0;   // variable para controlar cuando se activa pulsador calibracion
String frase; // Para almacenar varios caracteres
double valor;
float banderaVelocidad=Velmax;
 int bandera=0;
// Configuración de la librería QTR-8A
QTRSensorsAnalog qtra((unsigned char[]) {1,2,3,4,5,6}
, NUM_SENSORS, NUM_SAMPLES_PER_SENSOR, EMITTER_PIN);
```

Figura 16.Código Arduino Velocista

3. Estas dos funciones generan Movimiento a los motores dependiendo el valor que tome la variable *value* y envía el PWM a dichos motores.

```
// Función accionamiento motor izquierdo
void Motoriz(int value)
{
  if ( value >= 0 )
  {
    digitalWrite(BIN1,HIGH);
    digitalWrite(BIN2,LOW);
  }
  else
  {
    digitalWrite(BIN1,LOW);
    digitalWrite(BIN2,HIGH);
    value *= -1;
  }
  analogWrite(PWMB,value);
}

 // Función accionamiento motor derecho
 void Motorde(int value)
 {
   if ( value >= 0 )
   {
     digitalWrite(AIN1,HIGH);
     digitalWrite(AIN2,LOW);
   }
   else
   {
     digitalWrite(AIN1,LOW);
     digitalWrite(AIN2,HIGH);
     value *= -1;
   }
   analogWrite(PWMA,value);
 }
```

Figura 17.Código Arduino Velocista

4. Creamos dos funciones, la primera que accionará cada motor y la segunda que será la función que hace que me frenen dichos motores.

```
//Accionamiento de motores
void Motor(int righ, int left)
{
  digitalWrite(STBY,HIGH);
  Motoriz(left);
  Motorde(righ);
}

//función de freno
void freno(boolean righ, boolean left, int value)
{
  digitalWrite(STBY,HIGH);
  if ( left )
  {
    digitalWrite(BIN1,HIGH);
    digitalWrite(BIN2,HIGH);
    analogWrite (PWMB, value);
  }
  if ( righ )
  {
    digitalWrite(AIN1,HIGH);
    digitalWrite(AIN2,HIGH);
    analogWrite (PWMA, value);
  }
}
```

Figura 18.Código Arduino Velocista

5. Se debe presionar un pulsador para empezar nuestro proceso de calibración, cuando esto pasa el arduino apaga los leds que tenía encendidos para indicar que la calibración empezará.

```
void calibracion() {

 estado = digitalRead(pulsador);   // se lee estado pulsador

  while (estado == 1)
  {
   digitalWrite(LED, HIGH);
   estado = digitalRead(pulsador);
  };

  if (estado == 0)    // el pulsador se activa con 0 entonces apaga los led led
 {
  digitalWrite(LED, LOW);
```

Figura 19.Código Arduino Velocista

6. Cuando se presiona el pulsador empezará el proceso de calibración que aproximadamente dura 6s, cuando este proceso termina los leds que estaban apagados se encienden para indicar que el proceso de calibración ha terminado, para que el carro empiece a funcionar de manera correcta se debe volver a presionar el pulsador después de todo este proceso.

```
  ///////inicio calibracion

  for (int i=0; i<70; i++)
  {
    digitalWrite(LED, HIGH); delay(20);
    qtra.calibrate();
    digitalWrite(LED, LOW);  delay(20);
  }
  delay(3000);


/////// fin calibracion////////////


digitalWrite(LED, LOW);     // Apaga el led para indicar que se termino la calibracion.


  delay(300);
  digitalWrite(LED, HIGH);

  delay(300);
  digitalWrite(LED, LOW);     // Se encienden y apagan LEDS .

  delay(300);
  digitalWrite(LED, HIGH);

  delay(300);
  digitalWrite(LED, LOW);
```

```
  delay(300);


  estado = digitalRead(pulsador);
  while (estado == HIGH)
  {
   digitalWrite(LED, HIGH);

  estado = digitalRead(pulsador);
  };
  if (estado == LOW)     //   se esepera a que se presione pulsador para poner a funcionar robot
 {
  digitalWrite(LED, LOW);

  delay(1500);
};
```

Figura 20.Código Arduino Velocista

7. En este *void setup()* Declaramos los pines que utilizaremos como salida y llamaremos la función de *Calibración.*

```
void setup()
{

  Serial.begin(9600);

// Declaramos como salida los pines utilizados
  pinMode(LED   ,OUTPUT);
  pinMode(BIN2  ,OUTPUT);
  pinMode(STBY  ,OUTPUT);
  pinMode(BIN1  ,OUTPUT);
  pinMode(PWMB  ,OUTPUT);
  pinMode(AIN1  ,OUTPUT);
  pinMode(AIN2  ,OUTPUT);
  pinMode(PWMA  ,OUTPUT);


calibracion();

}
```

Figura 21.Código Arduino Velocista

8.Creamos las variables que utilizaremos en nuestro controlador PD, La variable *diferencial* me indica la diferencia aplicada a los motores, el *Last_prop* me indica el último valor que tomó la variable del proporcional y finalmente *Target* es nuestro setPoint que varía entre 0 y 5000 y se toma el valor de 2500 ya que este valor me indica que el sensor está sobre una línea negra.

```
unsigned int position = 0;

//declaraos variables para utilizar PID
int proporcional = 0;         // Proporcional
int integral = 0;            //Intrgral
int derivativo = 0;           // Derivativo


int diferencial = 0;
int last_prop;
int Target = 2500;
```

Figura 22.Código Arduino Velocista

9. Finalmente en el *void loop* de nuestro controlador, hacemos la parte lógica del control, que dependiendo el valor que toman los sensores, se mande una orden correspondiente a los motores para así accionar dicho motor deseado. Nuestro controlador tiene las tres constantes del control PID pero la constante *integra*l no es utilizada, por lo que utilizamos un control PD, que será eficiente si las constantes de dicho control son bien calibradas.

```
void loop()
{

  position = qtra.readLine(sensorValues);
  proporcional = ((int)position) - 2500;

  derivativo = proporcional - last_prop;
  integral = error1+error2+error3+error4+error5+error6;
  last_prop = proporcional;

  error6=error5;
  error5=error4;
  error4=error3;
  error3=error2;
  error2=error1;
  error1=proporcional;

 int diferencial = ( proporcional * KP ) + ( derivativo * KD )+ (integral*Ki) ;
```

```
  if ( diferencial > Velmax ) diferencial = Velmax;
  else if ( diferencial < -Velmax ) diferencial = -Velmax;

  ( diferencial < 0 ) ?
    Motor(Velmax+diferencial,Velmax) : Motor(Velmax, Velmax-diferencial );
  if(Velmax==1) delay(50);
  delay(20);
  serialEvent(); // Comprovar si tenemos datos entrantes

}
```

Figura 23.Código Arduino Velocista

❖ APLICACIÓN MÓVIL Y BLUETOOTH

Después de haber realizado el código, y tenerlo montado en el carro lo ponemos en funcionamiento, nos dimos cuenta que la labor de calibrar las constantes del control pd, conforme a la velocidad, no iba a ser eficiente y nos llevaría mucho tiempo, ya que para hacer un cambio en una de estas constantes se debe modificar el código, compilar, subir al controlador y realizar la prueba.
Para darle una mayor versatilidad a nuestro carro, decidimos añadirle un módulo bluetooth, con el fin de poder hacer los cambios de velocidad y de las constantes desde una aplicación móvil, este nuevo módulo nos evita el problema de estar modificando y subiendo el código.

La aplicación se realizó el el software web “APP INVENTOR” del MIT. este es un software que permite realizar diferentes aplicaciones de una manera muy sencilla y efectiva, justo lo que necesitamos.

La aplicación consta de:

- ❖ Un apartado para enlazar el bluetooth con la aplicación
- ❖ Un slider para variar la velocidad y su botón de enviar el nuevo dato
- ❖ Dos campos de texto y sus botones de enviar, para las constantes proporcional y derivativa
- ❖ Un botón para frenar el carro
- ❖ Un botón para arrancar el carro

La funcionalidad de esta aplicación es muy simple, cuando se oprime los botones de frenar o arrancar, lo que hace es enviar un numero en especifico al puerto serial del arduino, cuando se oprime el botón de enviar dato, envia al pùerto serial un numero en especifico dependiendo de que constante sea la que se va a modificar y después envía el valor de la constantes.
Estos valores que llegan al puerto serán utilizados en la lógica del carro para tomar las decisiones respectivas.

A continuación se mostraran la aplicación final y el diagrama de bloques de la construcción en App Inventor.

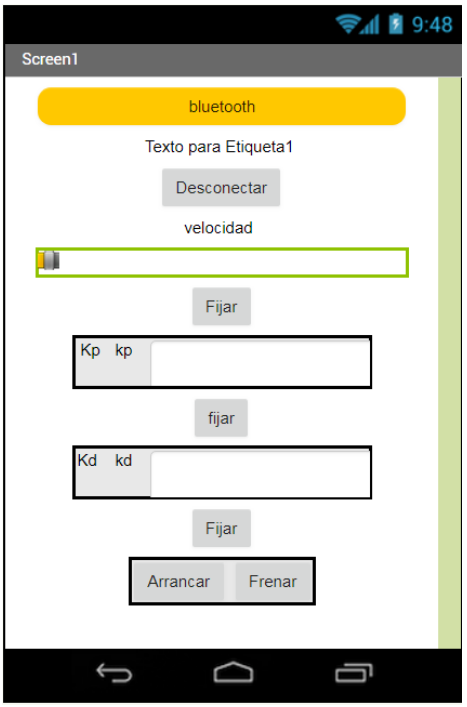

Figura 24. Aplicación móvil carro velocista.

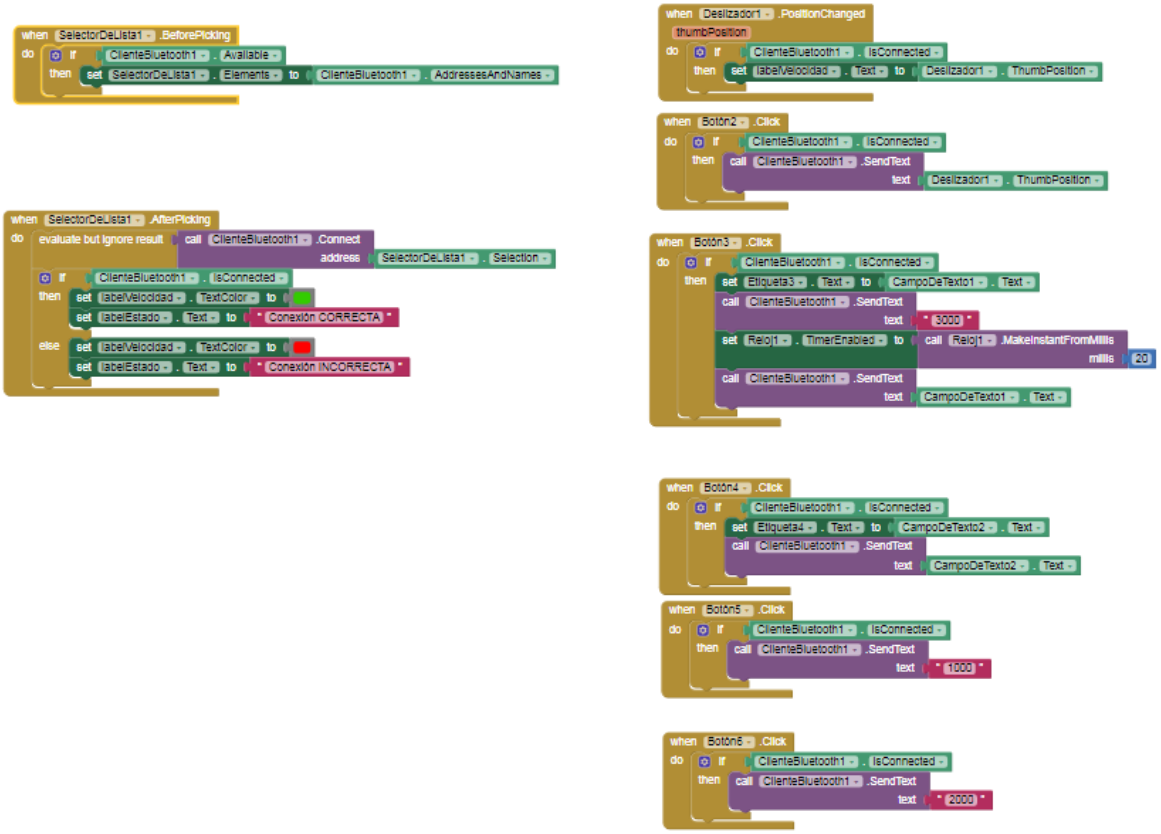

Figura 25. Diagrama de bloques aplicacion carro velocista.

Después de tener nuestra aplicación funcionando, entonces se procedió a crear un método en el código de arduino para por manipular estos datos que recibe desde el bluetooth.Este código lo que hace es leer si ha llegado un valor nuevo el puerto serial, con condicionales (IF) se evalúa las condiciones de este dato y después de haber identificado qué tipo de dato es.

```
void serialEvent() {
  while (Serial.available()) {
    if (Serial.available() > 0) { // Mirar si han entrado datos
      char letra = Serial.read();
      frase += letra; // Almacena los datos creando una frase
    }
  }
  if (frase.length() > 0) {// Si hay datos, miramos su valor
    valor = frase.toFloat(); //Convertimos a valor numerico
    Serial.println(valor);  // Muestra el valor
  }

  if(valor==1000){
    Velmax=banderaVelocidad;
    valor=6000;
  }
  if(valor==2000){
    banderaVelocidad=Velmax;
    Velmax=1;
    valor=6000;
    }
  if(valor== 3000) bandera=1;
  if(valor== 8000) bandera=2;
  if(valor== 9000) bandera=3;
  if(bandera==1 and valor!=3000){
  banderaVelocidad= valor ;
  bandera=0;

              if(bandera==2 and valor!=8000){
              KP= valor;
              bandera=0;
              }
              if(bandera==3 and valor!=9000){
              KD= valor;
              bandera=0;
              }
              frase = ""; // Borrar buffer...
            }
```

Figura 26. Logica y analisis de datos bluetooth.

VI. ANÁLISIS Y RESULTADOS DEL PLAN DE PRUEBAS

PLAN DE PRUEBAS

Una vez montado nuestro carro velocista con la programación correcta se procedió a probarlo en la pista con el objetivo que realizará el recorrido con éxito. Se colocó al robot dentro de la pista de competencia, en la primera prueba realizada en la parte de línea recta de la pista se evidencio que el robot zigzagueaba al seguir la trayectoria. Seguido a estos resultados se realizó ajustes en la programación, cambiando los valores de las constantes del controlador; luego el robot al seguir las líneas derechas en las curvas se salía de la pista a causa de la alta velocidad con la que fue programado. Se realizaron varios reajustes. Posteriormente a este periodo de pruebas y correcciones el robot seguía la trayectoria de forma correcta, hasta completar el recorrido de la pista.

Figura 27. Pista de Prueba

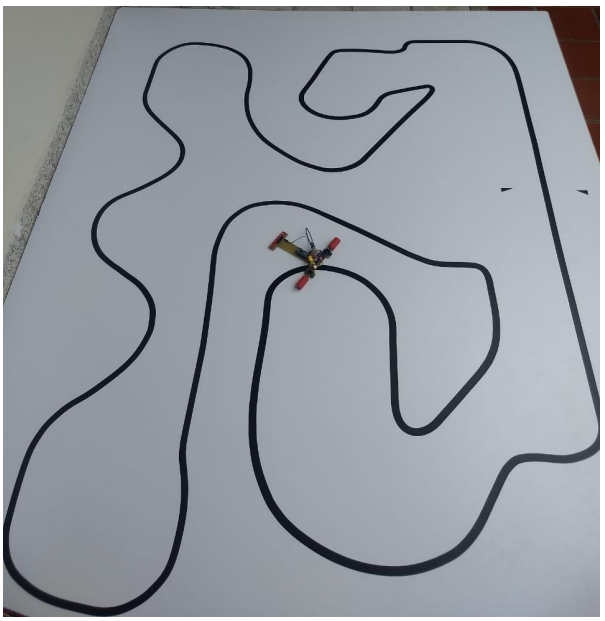

Figura 28. Robot sin seguir la linea

Figura 29. Robot siguiendo la linea

RESULTADOS

El diseño y construcción del robot seguidor de línea es una tarea que demanda la aplicación de conocimientos de las diversas áreas del saber, así como el empleo de habilidades para el

manejo de herramientas con nuestro equipo de trabajo seguimos un plan de trabajo logrando los siguientes resultados:

• Trabajo en equipo e integración de diferentes áreas del conocimiento tecnológico.

• El desarrollo de habilidades cognitivas y psicomotrices para el diseño y construcción del robot.

• Se propició el aprendizaje basado en competencias a través de estrategia de resolución de problemas, trabajo colaborativo y aprendizaje basado en proyectos.

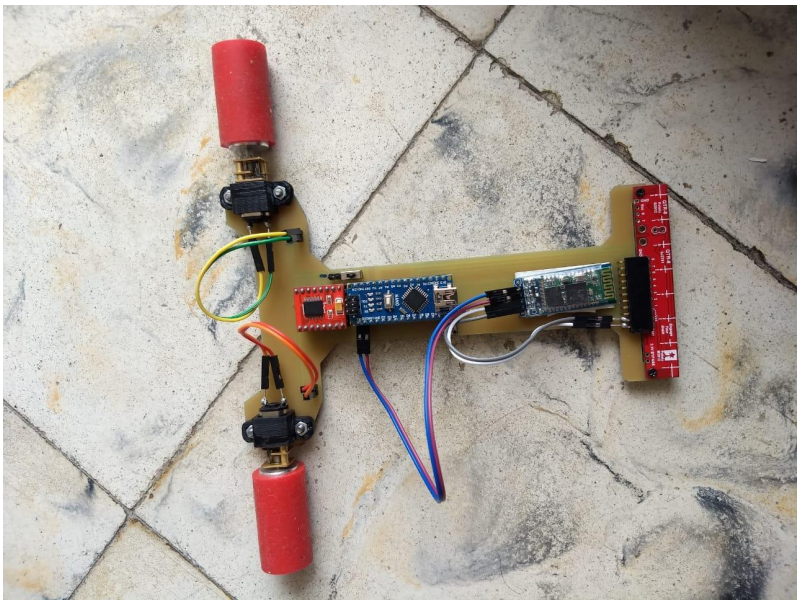

Figura 30. Vista frontal Carro Velocista

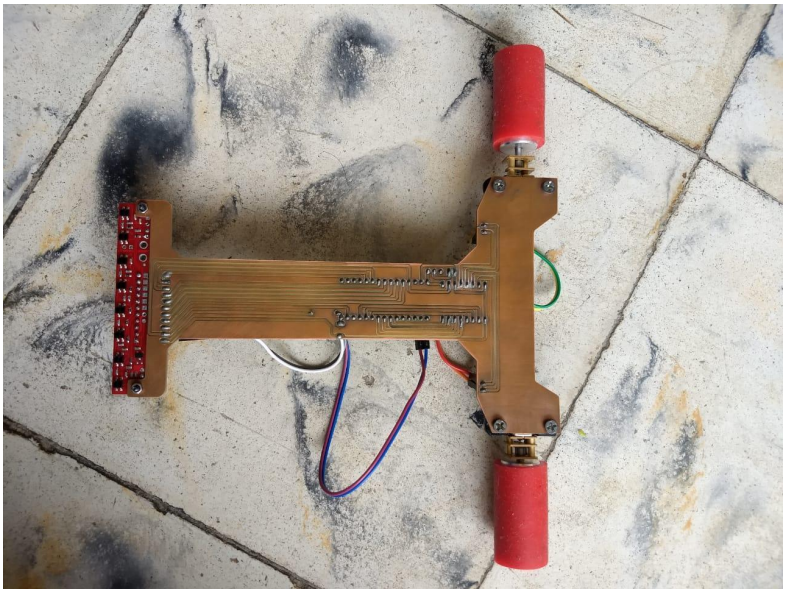

Figura 31. Vista trasera robot velocista

VII. APLICACIONES

Este tipo de robot se le llama Vehículo guiado automatizado (Automatic Guided Vehicle o en siglas AGV). Además de seguir una línea, suele añadir sensores de distancia para evitar chocar con gente u otros vehículos, radares, visión artificial, etc. Las aplicaciones de estos robots suelen estar relacionadas con el trasporte de mercancías dentro de la misma empresa o almacenaje. Se suele crear una red de robots que mueven la producción de un sitio hacia un almacén, o hacia otro punto de la cadena de fabricación.

- **Modelo Weasel® de la marca SCHAEFER** [4]

Figura 32. Modelo Weasel

Este modelo está pensado para transportar pequeñas cargas (hasta 35 Kg) por una empresa, automatizando los desplazamientos internos. Para funcionar se debe crear unos caminos utilizando unas bandas de seguimiento óptico, que no son más que líneas negras, aunque también dispone de otros medios de seguridad para evitar posibles accidentes.

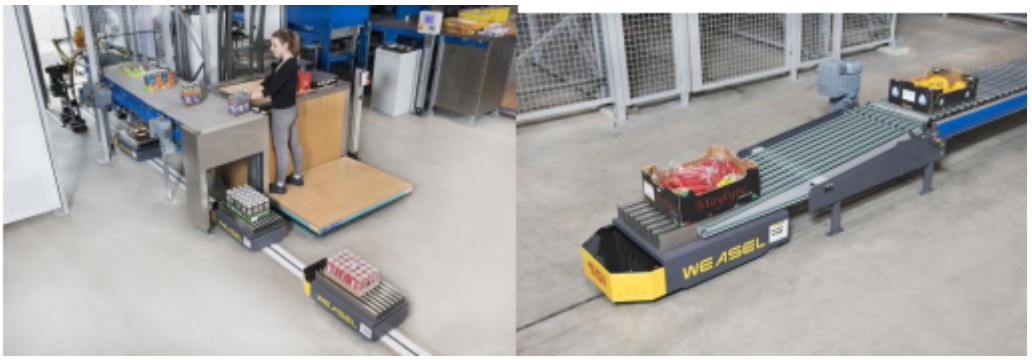

Figura 33. Carga y descarga de mercancía.

En la figura se aprecia como el carrito es utilizado en la carga y descarga de mercancía, también se aprecian los puntos de control.

- **Modelo CEITruck® de la marca CEIT Technical Innovation** [5]:

La marca CEIT dispone de un modelo parecido al Weasel, pero en vez de cargar el material encima, lo lleva con remolque.

Figura 34. Modelo CEITruck

Una diferencia notoria en cuanto al modelo anterior es que este es capaz de desplazar mucha más carga desde los 500Kg hasta los 3000Kg. De la misma manera que el otro robot este robot se integra de un sistema global de control.

Figura 35. Sistema integrado de control.

VIII. LISTADO DE ELEMENTOS

- Motores eléctricos
- Batería
- Llantas de caucho silicón
- Tornillos y tuercas
- Puente H
- Arduino Nano
- Conjunto de sensores de reflectancia QTR-8A

- PCB Circuito impreso

X. CONCLUSIONES

- Cuando el robot seguía la línea con buena precisión, se podía aumentar la velocidad, pero al aumentar la velocidad se afecta el controlador PID y requiere una Re sintonización para volver a quedar en funcionamiento.

- El código no debe ser muy complejo, ya que si tenemos un código simple que cumpla todos los requerimientos, la respuesta del sistema será mucho mas rapida.

- Para hacer una buena práctica se debe calibrar de manera adecuada los sensores y en el tiempo específico que se establece dicha calibración.

- Para usar el valor adecuado de las constantes, se debe usar el método de tanteo(*prueba y error)*, ya que no existen fórmulas sencillas para hallar dichas constantes.

- Un control se dice eficiente, cuando nuestro robot velocista sigue la línea a una velocidad adecuada, sin oscilar y con una respuesta rápida del sistema.

- En la parte del hardware, los elementos deben estar bien conectados para que la prueba sea exitosa, algo que pudimos observar es la regleta de sensores debe tener una altura mínima para que se pueda leer de manera correcta la línea negra.

IX. BIBLIOGRAFÍA

[1]Aplicaciones

https://upcommons.upc.edu/bitstream/handle/2117/97322/PFC%20David%20Ortiz%20Mart%C3%ADnez.pdf?sequence=1&isAllowed=y

[2] Materiales robot velocista

http://aprendiendofacilelectronica.blogspot.com/2014/12/robot-velocista-de-competencia.html

[3] Librería Sensores QTR-8A

http://aprendiendofacilelectronica.blogspot.com/2016/11/robot-velocista-qtr8a-parte-ii.html

[4] Modelo Weasel® de la marca SCHAEFER

http://www.ssi-schaefer.es/sistemas-logisticos/vehiculos-de-guiadoautomatico-agv/weasel-R.html

[5] Modelo CEITruck® de la marca CEIT Technical Innovation

http://www.ceittechinnovation.eu/index.php/en/agv-system