

TFG MsPackMan Físico

[Subtítulo del documento]

[Fecha]

Juan manuel salvá rossich

Contenido

[1. Objetivos 3](#_Toc93861477)

[Planificación 3](#_Toc93861478)

[2. Componentes 4](#_Toc93861479)

[3. Desarrollo del robot 5](#_Toc93861480)

[3.1 Prototipado 5](#_Toc93861481)

[10](#_Toc93861482)

[3.2 Primer planteamiento 12](#_Toc93861483)

[Planteamiento 12](#_Toc93861484)

[Representación del mapa 12](#_Toc93861485)

[Funcionamiento 13](#_Toc93861486)

[3.3 Primera iteración 14](#_Toc93861487)

[Diseño del robot 14](#_Toc93861488)

[Esquema eléctrico 15](#_Toc93861489)

[Conclusión 15](#_Toc93861490)

[3.4 Segunda iteración 15](#_Toc93861491)

[Diseño del robot 15](#_Toc93861492)

[Construcción 16](#_Toc93861493)

[Conclusión 16](#_Toc93861494)

[3.5 Tercera iteración 17](#_Toc93861495)

[Diseño del robot 17](#_Toc93861496)

[Construcción 18](#_Toc93861497)

[Pruebas 18](#_Toc93861498)

[Conclusión 19](#_Toc93861499)

[3.6 Cuarta iteración 19](#_Toc93861500)

[Diseño del robot 19](#_Toc93861501)

[Construcción 19](#_Toc93861502)

[Pruebas 19](#_Toc93861503)

[Conclusión 24](#_Toc93861504)

[3.7 Quinta iteración 24](#_Toc93861505)

[Diseño del robot 24](#_Toc93861506)

[Construcción 25](#_Toc93861507)

[Pruebas 26](#_Toc93861508)

[Conclusión 28](#_Toc93861509)

[4. Desarrollo del software 29](#_Toc93861510)

[4.1 Comunicación Java-Arduino 29](#_Toc93861511)

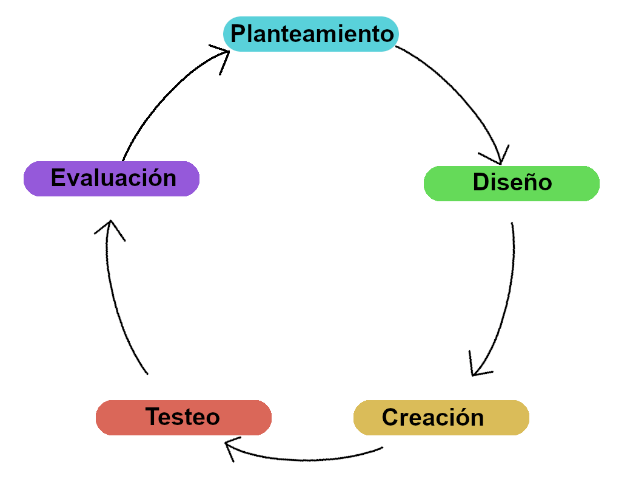
[4.2 Comunicación Java-Arduino-Arduino 30](#_Toc93861512)

[Bibliografía 33](#_Toc93861513)

1. Objetivos

La idea de este trabajo de fin de carrera (de aquí en adelante TFG), es hacer el juego MsPacMan en versión física con robots moviéndose de manera autónoma por el suelo usando una versión modificada de la inteligencia artificial ya programada en la asignatura de ingeniería de comportamiento inteligentes (ICI) el año pasado.

Mediante este trabajo se tratará de obtener conocimientos sobre el funcionamiento básico de robots, la construcción de los mismo, la manera en la que se programan y cómo conseguir una comunicación entre un programa de pc y unos robots.

Al ser un proyecto en el que hay que crear los robots desde 0, habrá diversas iteraciones del producto ya que es un proceso iterativo siguiendo el siguiente esquema:

Planificación

En un proyecto tan largo como este es necesario una buena planificación, para ello, llevaba un Excel (figura 1.1) con las tareas, plazos, hitos y progreso.

Interfaz de usuario gráfica, Escala de tiempo

Descripción generada automáticamente

Figura 1.1

Inicialmente solo había planificado hasta las fiestas navidades ya que al ser la primera vez haciendo un proyecto similar, no sabía muy bien cuanto se podían alargar las tareas. La idea principal era estar de septiembre a diciembre con el desarrollo de los robots. A partir de enero y hasta el final del curso, me centraría en la programación. Esta parte es cuando se relaciona la inteligencia artificial desarrollada en ICI con el robot para simular una partida física de MsPacMan.

2. Componentes

La idea inicial era que cada robot estuviera controlado por un Arduino Uno, que es el componente principal. El robot tendría tracción a las 4 ruedas mediante un motor dc en cada una de ellas, consiguiendo así aceleraciones más rápidas y un mejor control a la hora de girar. Para controlar los motores es necesario un controlador, para ello usaría los controladores l298N que permiten controlar dos motores cada uno, así, solo serían necesarios dos.

El coche necesitará saber por dónde va, para ello irá siguiendo unas guías que habrá por el suelo mediante cinta negra. El robot sabrá la posición de dichas líneas mediante unos sensores, en concreto 3 sensores ky-033 por coche. Cada uno de estos sensores (de aquí en adelante sensores de línea), son capaces mediante infrarrojos de detectar la intensidad de color de una superficie a unos 2 centímetros. La comunicación con el servidor será a través de bluetooth usando el módulo HC-05.

El robot estará alimentado mediante una batería recargable de 9V. Esto muy probablemente podría cambiar en futuras iteraciones ya que de momento el amperaje de la batería soporta las piezas, pero si se cambian muchas piezas, puede variar haciendo que esta batería no fuera útil. En caso de que la batería no aguantara suficiente tiempo alimentando el robot, se tendrán que usar dos baterías conectadas en serie multiplicando por dos el almacenamiento de energía. En la figura 2.1 podemos ver los componentes principales necesarios para un robot.

Imagen que contiene interior, tabla, montón, mostrador

Descripción generada automáticamente

Figura 2.1

3. Desarrollo del robot

En esta sección, se repasan todas las iteraciones desde el prototipado inicial hasta el producto final.

3.1 Prototipado

Primero de todo, una vez tengo todos los componentes, tengo que empezar a prototipar con ellos para hacer pruebas de concepto.

3.1.1 Sensor de línea

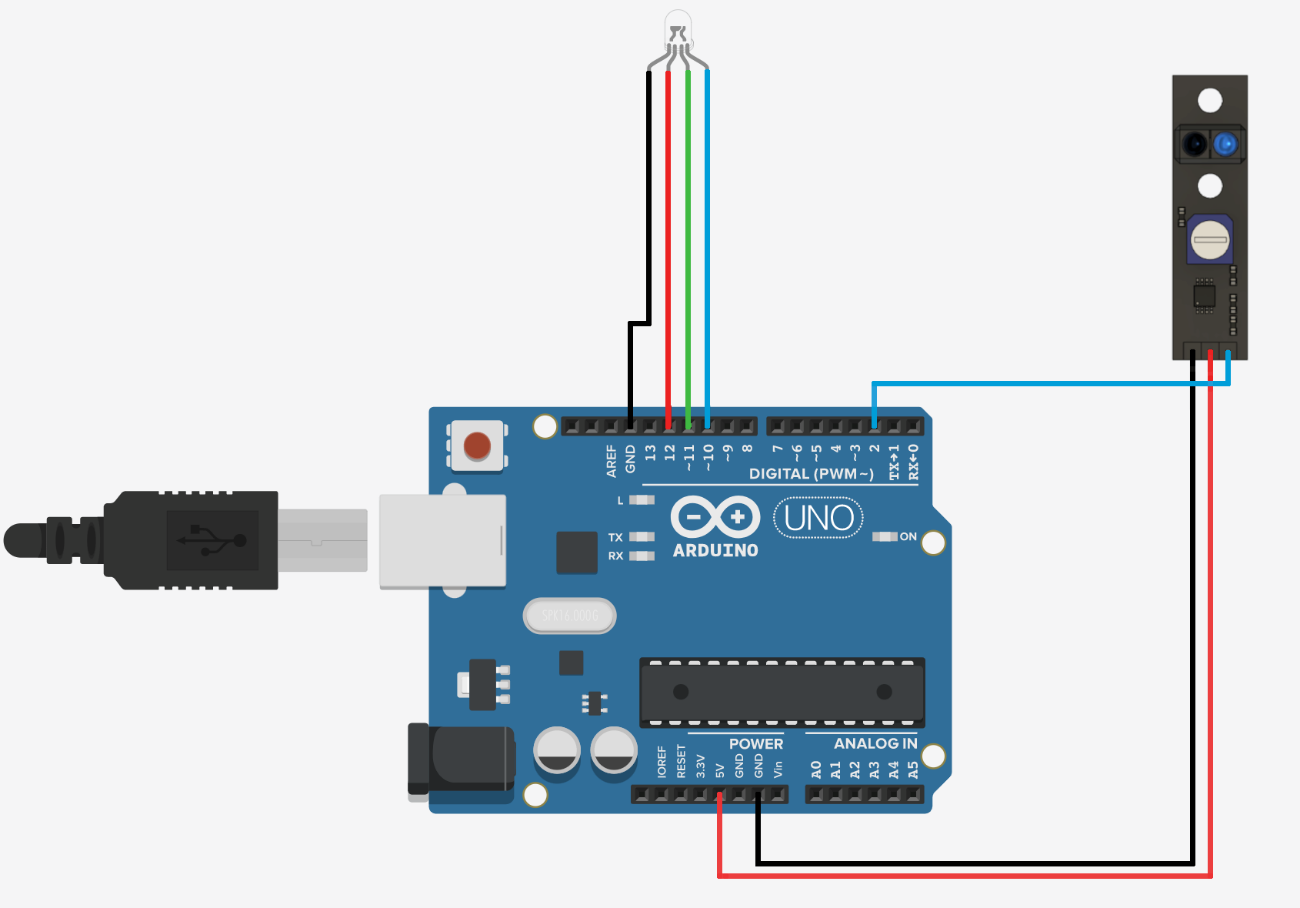
Lo primero y más importante es averiguar el funcionamiento del sensor de línea ya que esto va a ser lo que determine la forma del robot final al ser la única pieza que tiene que ir en un lugar fijo, cerca del suelo. Para esta primera prueba, hay una luz led rgb de debug para saber lo que está detectando el sensor. Podemos ver el esquema eléctrico en la figura 3.1.1 y el código para este prototipo justo a continuación en la figura 3.1.2.

Figura 3.1.1

Código 3.1.2

int SENSOR = 2;

int VALOR;

int ANTERIOR = 1;

int LEDROJO = 9;

int LEDVERDE = 10;

int LEDAZUL = 11;

void setup() {

  Serial.begin(9600);

  pinMode(SENSOR, INPUT);

  pinMode(LEDROJO, OUTPUT);

  pinMode(LEDVERDE, OUTPUT);

  pinMode(LEDAZUL, OUTPUT);

}

void loop() {

  VALOR = digitalRead(SENSOR);

  if (VALOR == HIGH){

    analogWrite(LEDROJO, 0);

    analogWrite(LEDVERDE, 255);

    analogWrite(LEDAZUL, 0);

  }

  else{

    analogWrite(LEDROJO, 255);

    analogWrite(LEDVERDE, 0);

    analogWrite(LEDAZUL, 0);

  }

}

En el código podemos ver como se inicializan los pines del Arduino para el sensor como datos de input y los pines del led como output.

En el loop principal tan solo tenemos que leer en cada bucle el input del sensor, si el valor leído es un uno (HIGH) entonces ponemos el led verde y en el caso de leer un cero (LOW) ponemos el led rojo.

Cuando el sensor de línea saca uno de output (que el Arduino usa de input), entonces significa que estamos sobre una línea.

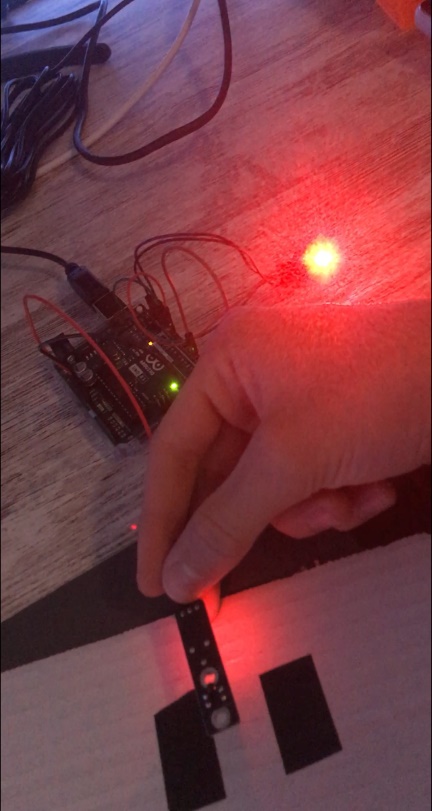
En las dos siguientes figuras vemos el resultado físico:

Figura 3.1.3

Figura 3.1.2

3.1.2 Motor

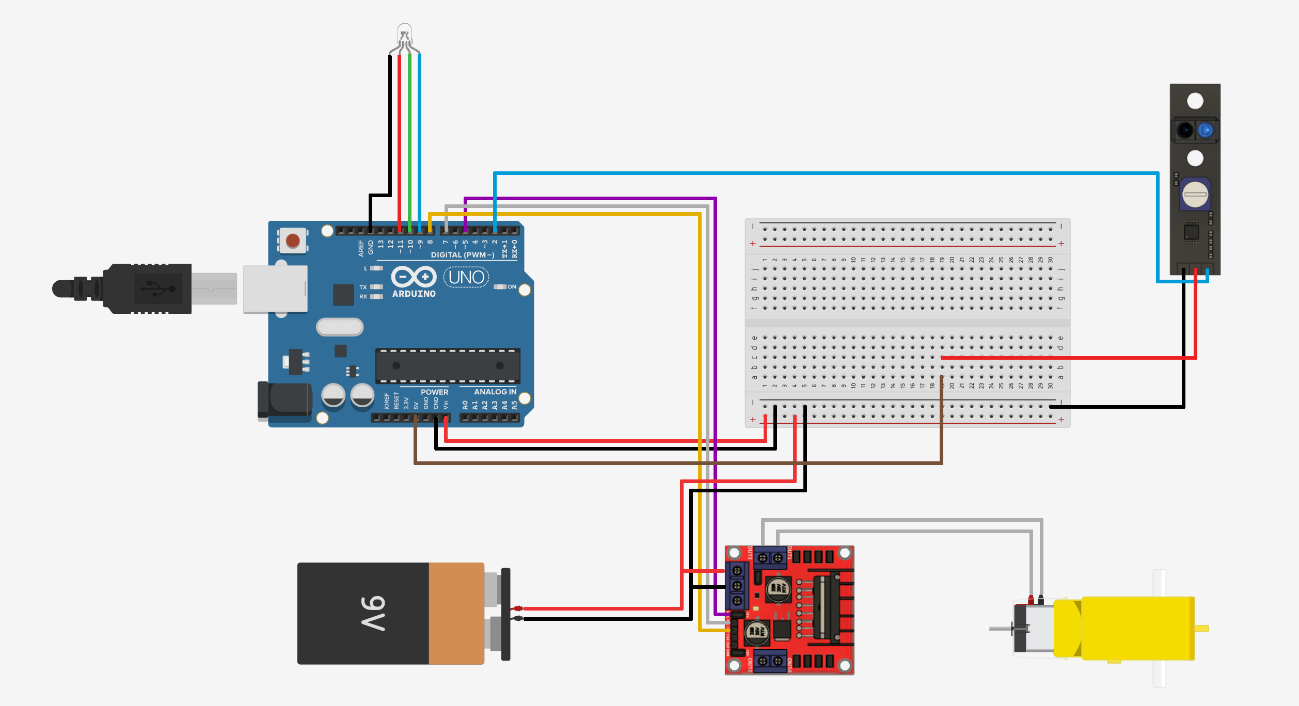
Una vez controlado el sensor de línea, es necesario probar los motores. El motor requiere más potencia eléctrica que un sensor, por tanto, la alimentación del propio Arduino no vale, hay que introducir la batería de 9V. También, el motor no se puede controlar mediante los pines normales de un Arduino, es necesario conectarlo al controlador l298N que será el encargado de controlar el motor dc. A continuación, podemos ver el esquema eléctrico de esta prueba:

Figura 3.1.4

Al ser una prueba con bastantes elementos, usar una *bread board* es útil para facilitar las conexiones. Vemos como la batería va conectada directamente al controlador que lleva el motor ya que, como mencionado antes, los motores necesitan bastante potencia. La batería también va conectada a las líneas de corriente de la *bread board* para poderlo conectar con el pin VIM del Arduino (único pin que acepta 9V de corriente) y conectar ambas tomas de tierra ya que compartir la misma toma de tierra por todos los elementos es importante.

Por limpieza de cables, el led y los pones del controlador van directamente al Arduino, los únicos cables que pasan por la *bread board* son los del sensor de línea ya que este se encuentra bastante alejado del resto de componentes.

A partir de este momento, el sistema es totalmente autónomo y no necesita estar conectado al PC, solo es necesario conectarlo para subir el programa.

int sensor = 2;

int LEDROJO = 11;

int LEDAZUL = 10;

int LEDVERDE = 9;

int valor;

int IN1 = 7;

int IN2 = 8;

int EMA = 5;

void setup() {

  pinMode(in1, OUTPUT);

  pinMode(in2, OUTPUT);

  pinMode(ema, OUTPUT);

  pinMode(sensor, INPUT);

  pinMode(LEDROJO, OUTPUT);

  pinMode(LEDAZUL, OUTPUT);

  pinMode(LEDVERDE, OUTPUT);

}

El código para esta prueba se basa en el de la prueba anterior, solo que esta vez giramos el motor en diferentes direcciones según el output del sensor de línea.

EMA es el pin que permite el flujo de corriente, el los otros dos marcan (IN1 y IN2) el sentido de la corriente que permite controlar el sentido de giro de los motores

Código 3.1.2

void loop() {

  valor = digitalRead(sensor);

  if(valor == 1){

    analogWrite(LEDROJO, 0);

    analogWrite(LEDVERDE, 255);

    analogWrite(LEDAZUL, 0);

    digitalWrite(EMA, HIGH);

    digitalWrite(IN1, LOW);

    digitalWrite(IN2, HIGH);

  }

  else {

    analogWrite(LEDROJO, 255);

    analogWrite(LEDVERDE, 0);

    analogWrite(LEDAZUL, 0);

    digitalWrite(EMA, HIGH);

    digitalWrite(IN1, HIGH);

    digitalWrite(IN2, LOW);

  }

}

En la siguiente figura vemos el resultado real de esta prueba.

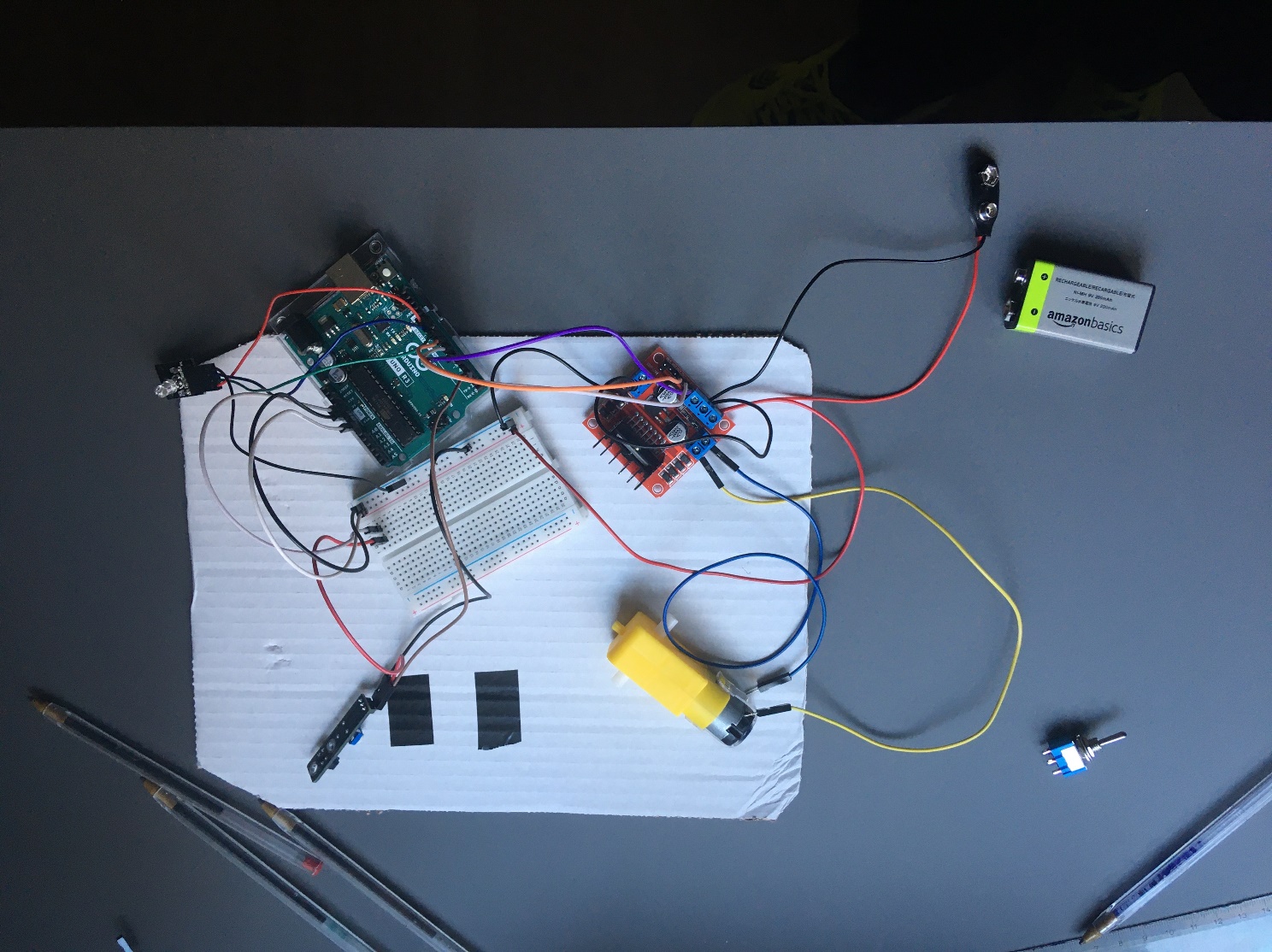


Figura 3.1.5

3.1.3 Bluetooth

La comunicación entre los robots y el servidor en el PC será mediante bluetooth, pero los módulos HC-05 solo permiten conectarse a un solo dispositivo bluetooth externo. Por tanto, el servido del PC tiene que comunicarse mediante cable o bluetooth con un Arduino que tendrá 5 módulos bluetooth, cada uno emparejado con un robot.

De momento, la primera prueba de concepto se basará en dos Arduinos enlazados entre sí. Uno será el maestro y el otro el esclavo. El maestro cada 200 milisegundos mandará un valor rotando entre 0,1,2 constantemente. El esclavo recibirá este dato e irá cambiando el color de una luz led según el dato recibido.

A continuación, vemos el esquema eléctrico de esta prueba, primero el esquema del maestro (figura 3.1.7) y después el del esclavo (figura 3.1.8).

Diagrama

Descripción generada automáticamente

Figura 3.1.6

Diagrama

Descripción generada automáticamente

Figura 3.1.7

Código 3.1.4

#include <SoftwareSerial.h>

SoftwareSerial miBT(10,11);

int LEDROJO = A2;

int LEDVERDE = A1;

int LEDAZUL = A0;

char dato;

void setup() {

  Serial.begin(9600);

  miBT.begin(38400);

  pinMode(LEDROJO, OUTPUT);

  pinMode(LEDVERDE, OUTPUT);

  pinMode(LEDAZUL, OUTPUT);

  analogWrite(LEDROJO, 255);

  analogWrite(LEDVERDE, 255);

  analogWrite(LEDAZUL, 255);

}

void loop(){

  if(miBT.available()){

    dato = miBT.read();

    if(dato == '0'){

      analogWrite(LEDROJO, 0);

      analogWrite(LEDVERDE, 0);

      analogWrite(LEDAZUL, 255);

    }

    else if(dato == '1'){

      analogWrite(LEDROJO, 0);

      analogWrite(LEDVERDE, 255);

      analogWrite(LEDAZUL, 0);

    }

    else if(dato == '2'){

      analogWrite(LEDROJO, 255);

      analogWrite(LEDVERDE, 0);

      analogWrite(LEDAZUL, 0);

    }

  }

}

Esclavo

En este caso tenemos dos códigos, uno para cada Arduino. Como se puede ver son bastantes simples.

Esta es la primera vez en la que se necesita una libraría, en concreto la de *SoftwareSerial*, pero no es la librería que se usa en el robot final ya que se usa *EasyTransfer*, porque permite pasar estructuras de datos sin tener que serializar y deserializar manualmente.

Código 3.1.3

#include <SoftwareSerial.h>

SoftwareSerial miBT(10,11);

void setup() {

  Serial.begin(9600);

  Serial.println("Listo maestro");

  miBT.begin(38400);

}

void configuracion(){

  if(miBT.available())

    Serial.write(miBT.read());

  if(Serial.available())

    miBT.write(Serial.read());

}

void loop(){

  miBT.print(0);

  delay(200);

  miBT.print(1);

  delay(200);

  miBT.print(2);

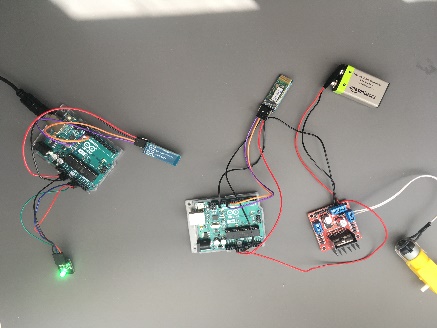
  delay(200);

}

Maestro

Texto

Descripción generada automáticamente con confianza media

Imagen que contiene cuarto

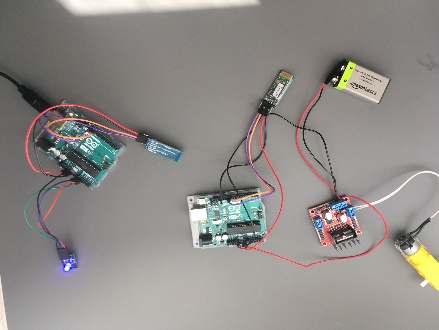
Descripción generada automáticamenteEn la siguiente cadena de imágenes se ve como el led va cambiando de color con los datos que se están mandando desde el otro Arduino.

Figura 3.1.8

La siguiente prueba era poder pasar estructuras de datos. Hasta ahora, solo se estaban mandando bytes sueltos, pero para el robot final esto no es suficiente, habrá que mandar estructuras de datos.

Para ello hay dos formas de hacerlo, la primera es completamente manual, mandas una cadena muy larga de bytes y al recibirla, la troceas en las distintas variables que necesites. La segunda manera, y la elegida para este proyecto, es usar la librería *EasyTransfer*, en concreto la versión *SoftEasyTransfer* la cual es capaz de enviar y recibir estructuras de datos.

Texto, Carta

Descripción generada automáticamenteEl código es muy similar al del apartado anterior, solo que, en vez de tener una LED para representar los datos, se hace por consola, así, podemos mostrar todos los campos de la estructura de datos.

Salida:

Código 3.1.6

[…]

struct RECIEVE\_DATA\_STRUCTURE{

  int8\_t id;

  int8\_t number;

};

RECIEVE\_DATA\_STRUCTURE myData;

[…]

void loop(){

  if(ET.receiveData()){

    Serial.print("ID: ");

    Serial.print(myData.id);

    Serial.print(", Data read: ");

    Serial.println(myData.number);

  }

Esclavo

Código 3.1.5

[…]

struct RECIEVE\_DATA\_STRUCTURE{

  int8\_t id;

  int8\_t number;

};

RECIEVE\_DATA\_STRUCTURE myData;

[…]

void loop(){

  myData.number = 10;

  ET.sendData();

  delay(200);

  myData.number = 20;

  ET.sendData();

  delay(200);

  myData.number = 30;

  ET.sendData();

  delay(200);

}

Maestro

En la salida se ve como el ciclo de datos ha empezado en 20, esto es porque aún no hay sincronización entre los Arduinos ya que para esta prueba no hace falta, esto se incorporará a la hora de montar el primer prototipo.

3.2 Primer planteamiento

Planteamiento

El proyecto entero estará controlado por una versión modificada del programa que hicimos en ICI. En el programa desarrollado en ICI, la simulación estaba controlada por el motor el cual llevaba el juego como tal y utilizaba una inteligencia artificial desarrollada por los alumnos tanto para MsPacMan como para los fantasmas. Ahora, el motor después de cada bucle de simulación tendrá un paso extra que será el encargado de comunicarse con los robots físicos para que vayan acorde con la simulación interna. Es decir, el juego como tal se corre en una máquina central que actúa como servidor, y los robots vienen siendo clientes “tontos” que van siguiendo las instrucciones del servidor.

Representación del mapa

Código QR

Descripción generada automáticamente con confianza bajaForma, Flecha

Descripción generada automáticamenteEl mapa del juego estará representado en el suelo mediante líneas negras, concretamente dos líneas marcando las paredes de cada pasillo (figura 1.2.1). Las intersecciones estarán representades mediante un punto negro en el medio (figura 1.3).

Figura 3.2.2

Figura 3.2.1

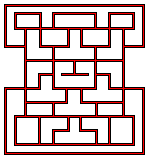
Con estas dos normas, el mapa completo quedará algo así:

Figura 3.2.3

Funcionamiento

El robot siempre tendrá un mínimo de un sensor detectando una línea, en el caso de una intersección, los tres sensores detectarán una línea, y en el caso de ir en línea recta, solo dos.

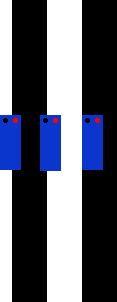
Internamente, el robot puede estar en dos estados, recto y girando.

Imagen que contiene Icono

Descripción generada automáticamente**Recto:** estará en este estado cuando esté en un pasillo. Para asegurarse de que no se desvía, los sensores 1 y 3 tendrán que estar detectando una línea (figura 1.2.4)

Figura 3.2.4

Mientras los dos sensores laterales estén viendo la línea y el central no, sabemos que estamos bien.

****

En cambio, si un sensor lateral ya no detecta la línea, pero el otro si, significa que el robot se ha movido de su línea, por tanto, tenemos que corregir un poco la dirección, en este caso, girando a la derecha levemente.

Figura 3.2.5

**Giro:** estará en este estado cada vez que se llegue a una intersección. En este momento, ya sabe si tiene seguir en línea recto o realizar un giro de 90 grados porque se lo ha dicho el servidor. En el caso de tener que girar, tendrá que parar en seco y mover las ruedas de un lado en dirección apuestas a las ruedas del otro lado, obteniendo así un giro en forma de “tanque”.

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Figura 3.2.6

Cuando queremos girar, le mandamos la orden al robot, pero este no girará hasta que todos sus sensores detecten una línea como se ve en el ejemplo asegurándonos así que estamos girando donde queremos.

3.3 Primera iteración

Diseño del robot

La idea principal, es conseguir un robot lo más compacto posible para poder representar el mapa lo más pequeño posible. Lo primero de todo es conseguir modelos 3D de todos los componentes del proyecto para poder diseñar en 3D ya que permite una mayor precisión luego a la hora de diseñar e imprimir el chasis, ya que es mucho más fácil iterar.

Por suerte, hay miles de modelos online ya disponibles para uso libre que me puedo descargar. El Arduino, los motores, las ruedas, y los controladores me los he podido descargar. El resto de los elementos al no ser tan comunes, no están disponibles online, por tanto, me tuve que hacer un modelo 3D yo mismo.

En la figura del apartado de componentes (figura 2.1) podemos ver todas las piezas necesarias para el robot.

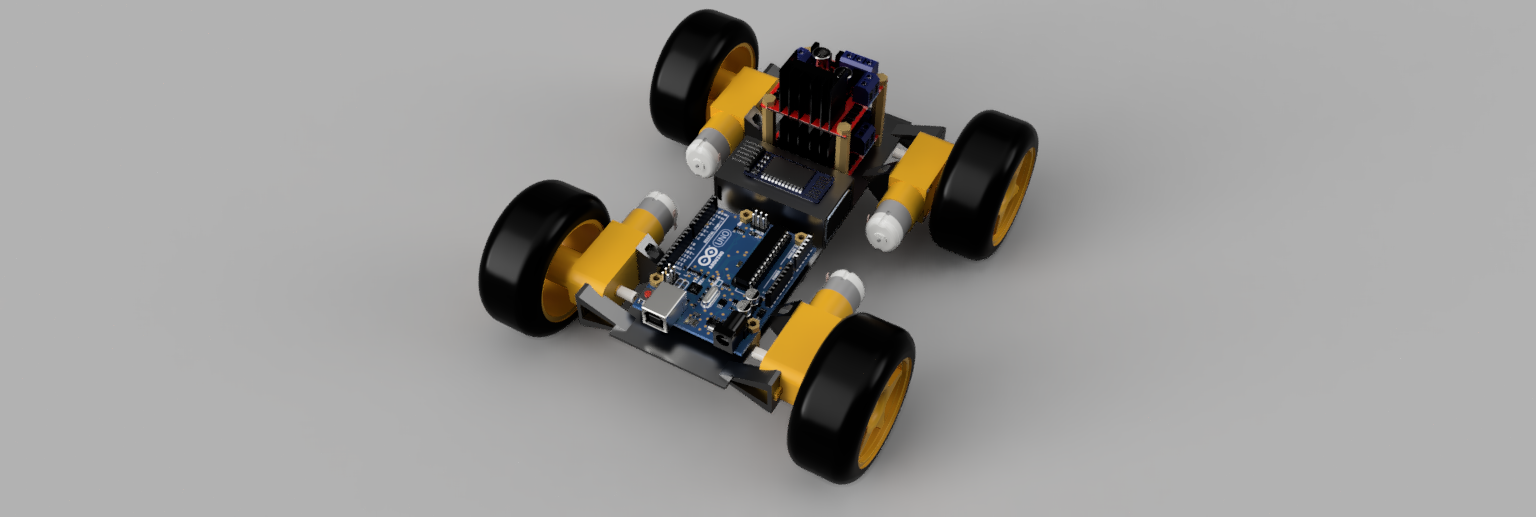
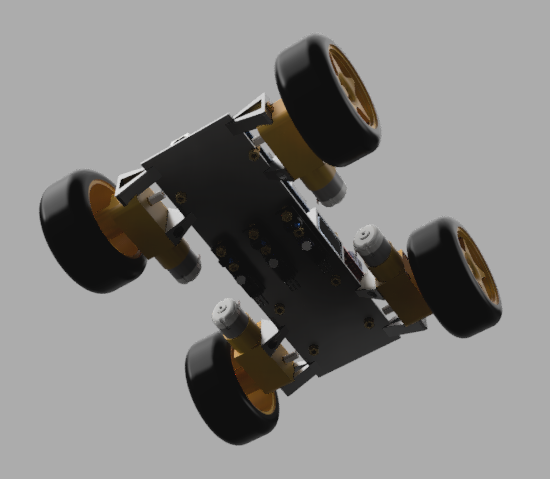
Una vez construido un chasis y montando todas las piezas en él, el robot queda con este aspecto:

Figura 3.3.1

Figura 3.3.2

Vemos en la parte inferior de la segunda figura que los sensores de línea están exactamente en el centro de los ejes del robot. Esto es para poder realizar giros sobre sí mismo precisamente.

El resto de los componentes están en la parte superior del chasis ya que no necesitan una posición exacta. Los dos controladores de los motores están montados uno encima del otro separados por unos espaciadores de nylon. Por el mismo motivo, la batería y el módulo bluetooth también están uno encima del otro, el módulo bluetooth siendo el de arriba para conseguir una mejor conectividad. Finalmente tenemos el Arduino con los puertos visibles desde el exterior para poder conectarlo al PC y realizar todos los cambios oportunos al software.

Esquema eléctrico

Conclusión

Esta primera iteración del robot sirvió como prueba de concepto sobre cómo se podrían organizar las piezas. Tras la primera presentación de ese prototipado frente al tutor, llegamos a la conclusión de que como la distancia entre ejes de ese prototipo era de aproximadamente 13cm, la representación del mapa al final acabaría siendo demasiado grande, más de 2 metros.

Por tanto, la primera iteración no se llegó ni siquiera a imprimir en 3D. Había que intentar compactar los componentes aún más.

3.4 Segunda iteración

Diseño del robot

Tras volver a Fusion 360 a repensar el prototipo, decidí colocar varias piezas de forma vertical, de esta manera, se reduce la distancia entre el eje delantero y el eje trasero, permitiéndonos reducir esta distancia de 13cm del prototipo anterior a aproximadamente 8cm.

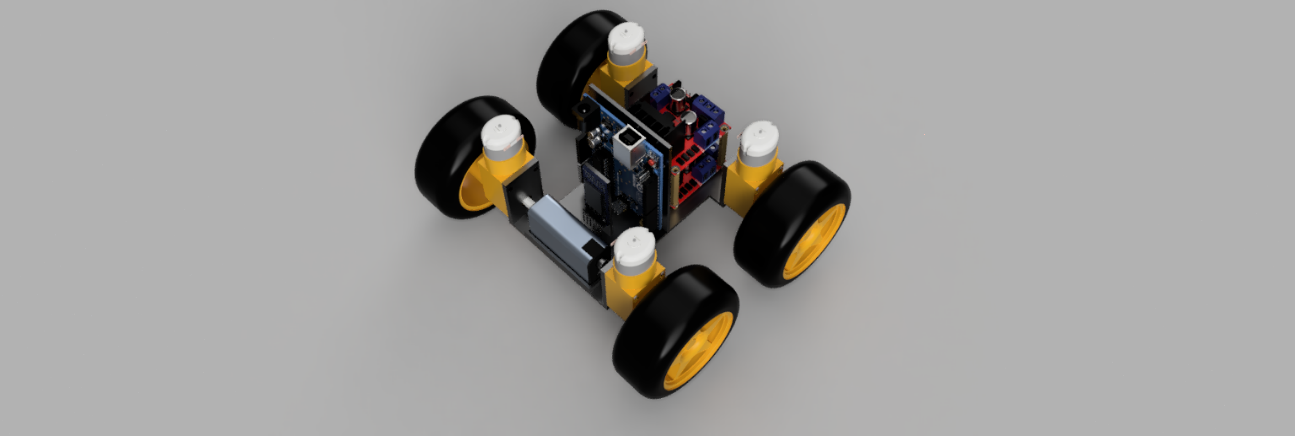
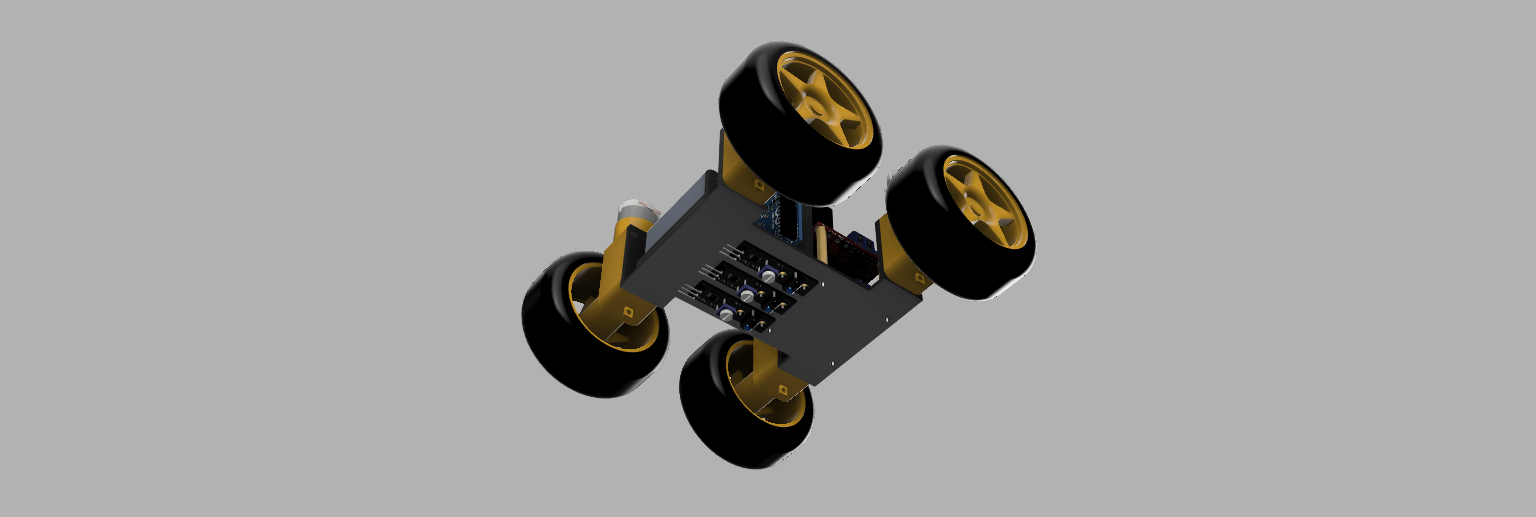


Figura 3.4.1

Figura 3.4.2

Como podemos ver en las figuras, el robot se ha compactado substancialmente en un eje principalmente. Esto, no debería ser problema ya que el mapa está compuesto por pasillos, y estos pasillos entre si tienen bastante separación, así que el eje horizontal sea más ancho que el vertical no importa tanto. Lo importante es que el eje vertical sea el corto, ya que es el que va a delimitar el tamaño del tablero.

Construcción

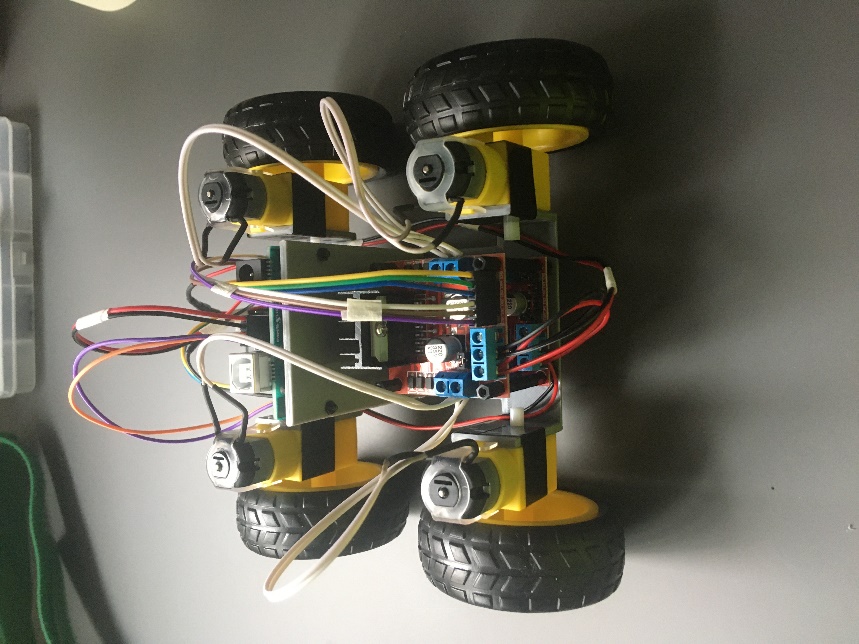
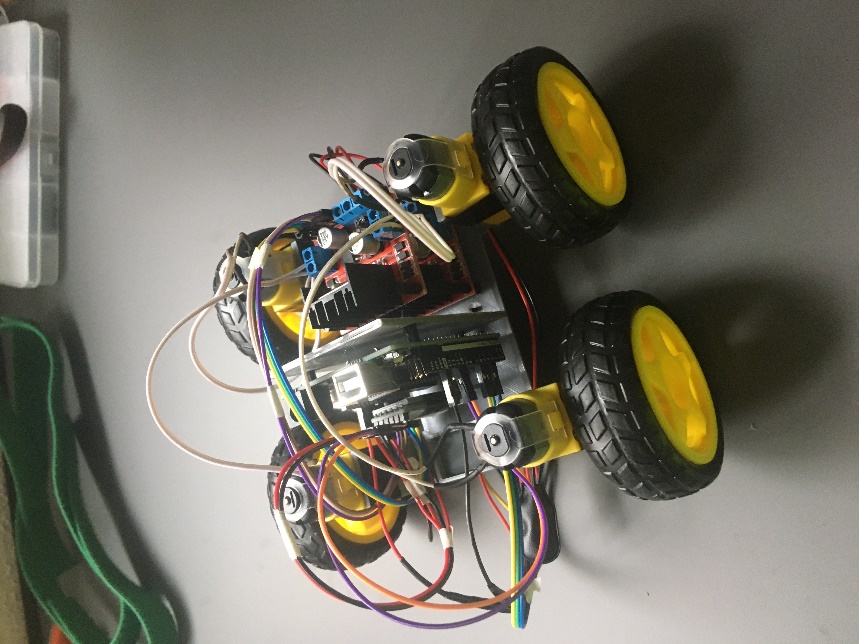
Tras imprimirlo en la impresora 3D, ya se podía empezar a montar los componentes dando el siguiente resultado:

Figura 3.4.3

Figura 3.4.5

Conclusión

Este prototipo llegó más lejos que el anterior, pero aún así tiene grandes fallos. Para empezar, el hueco para la batería no es lo suficientemente profundo como para sostener de manera estable la batería con los movimientos del robot. Por suerte esto es fácil de arreglar.

Otro problema que hay, es que cada controlador l298N es capaz de controlar dos salidas, por tanto, como se explica en el apartado de componentes del principio de la memoria, se necesitarían dos controladores para los 4 motores. Lo que no se me había pasado por la cabeza, es que los motores del mismo lado del robot siempre van a tener el mismo comportamiento, o los dos van para delante, o los dos están parados, o los dos hacia atrás. Por tanto, solo se necesita un solo controlador usando cada salida para dos motores. Son este tipo de cosas sencillas que no se te pasan por la cabeza hasta que lo estás construyendo de verdad y es precisamente por esto que esta primera parte del proyecto es un proceso altamente iterativo.

Para el siguiente prototipo, se reducirá el numero de controladores l298N a tan solo 1, se hará un compartimente para la batería más estables, un chasis más estable y ya que reducimos el número de componentes, se mirará de reducir el tamaño del robot aún más.

3.5 Tercera iteración

Diseño del robot

Para esta nueva iteración, con menos piezas, había que repensar la manera de organizar los componentes. Decidí usar el espacio liberado por uno de los controladores I298N para colocar la batería rodeada por cuatro paredes para impedir que se moviera. De esta manera, en la parte delantera del robot quedaba un gran hueco, para recolocar el Arduino y el módulo bluetooth. La anchura del robot hasta ahora estaba delimitada por el tamaño del Arduino ya que estaba colocado de manera perpendicular al robot. Pero con el hueco que había dejado la batería, se podía colocar el Arduino paralelamente permitiendo así mover los motores 30mm hacia el centro del eje compactando aún más el robot.

Imagen que contiene tabla, aire, calle, hombre

Descripción generada automáticamente

Figura 3.5.2

Figura 3.5.1

El eje vertical no se ha reducido prácticamente nada, pero el eje horizontal si, en concreto 3 cm. Este eje no era el eje crítico que delimita el tamaño del tablero, pero, aun así, cuanto más parecidos son los dos ejes, mejor girará y maniobrará además de ser más versátil.

Imagen que contiene tabla, coche, escritorio, lego

Descripción generada automáticamenteUn problema de la iteración anterior era que el chasis no era lo suficientemente rígido y al moverse se tambaleaba un poco. Por suerte, esto tenía una solución sencilla, el grosor de la base del chasis se aumentó en un 50%, y para los soportes de los motores, se les añadió refuerzos en la parte superior ya que anteriormente solo estaban soportados por la parte inferior como podemos ver en las siguientes figuras:

Después

Antes

Imagen que contiene tabla, lego, computadora, escritorio

Descripción generada automáticamente

Figura 3.5.4

Figura 3.5.3

Construcción

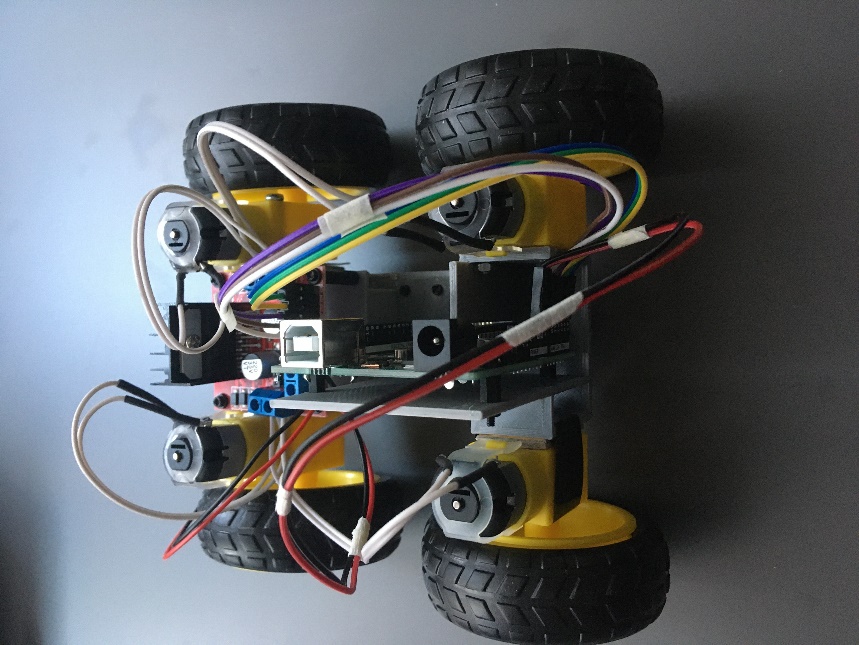
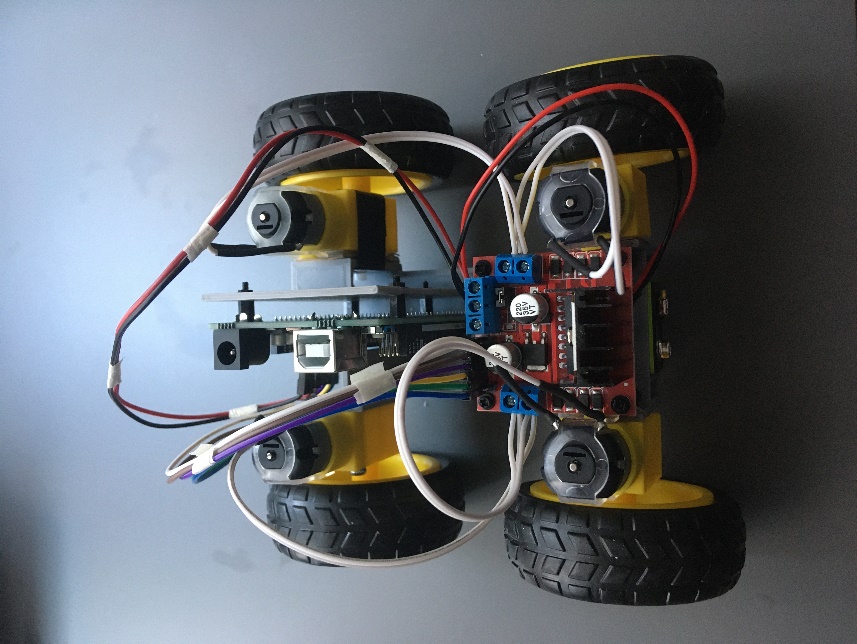
Para este prototipo, los motores ya iban enganchados con tornillos m3 de 25mm haciéndolo mucho más estable e independiente de las cintas que había en la iteración anterior. Otra mejora fue las conexiones de los cables de los motores, hasta ahora iban conectados por contacto y termo retráctil por encima, pero a partir de ahora irían soldados con estaño.

Figura 3.5.6

Figura 3.5.5

Pruebas

Este fue el primer prototipo el cual tenía algo de funcionalidad. No iba a ser el prototipo final ya que a la hora de construirlo ya me di cuenta de ciertos errores que había que solucionar cara al futuro, pero aún así se podía empezar a hacer pruebas.

La primera prueba de todas era saber la velocidad real del robot. Este dato ya se podría haber sacado tan solo con las revoluciones del motor, pero al añadirle el peso del robot, estas se reducirían ligeramente. En línea recta, el robot iba a unos 40cm/s. Si escalásemos el mapa del juego al tamaño del robot, obtendríamos un mapa de unos 2.4m de anchura, lo cual hace que la velocidad de 40cm/s coincida casi a la perfección con la velocidad de MsPacMan en el juego.

La segunda prueba era saber la velocidad de giro. Para ello, tan solo hay que hacer que los motores de un lado giren en dirección apuestas al del otro lado. De esta manera, un giro de 180 grados se realizaba en uno 450 milisegundos, y a consecuencia los giros de 90 grados en unos 230-235 milisegundos (no es exactamente la mitad por la aceleración inicial y final).

Ambas pruebas se realizaron en bucle, y al cabo de unos cuantos giros, el robot empezaba a desviarse, lo cual es totalmente normal. Por esto, están los sensores de línea, aun que para estas pruebas no se les ha dado implementación ya que había que diseñar un nuevo chasis.

Conclusión

Al igual que con el prototipo anterior, había pequeños errores de diseño, pero cada vez eran errores menos importantes. Aun así, este chasis nos ha permitido sacar datos importantes como la velocidad en línea recta, las velocidades de giro, y el tamaño final que debería tener el mapa.

3.6 Cuarta iteración

Diseño del robot

Esta fue la primera iteración de la cual realmente se pudo partir del chasis de la iteración anterior. Hasta ahora, los cambios a realizar eran tan drásticos que se tardaba menos empezando un diseño desde 0 que modificar el diseño de la iteración anterior.

Para esta iteración, tan solo se aumentaron distancias entre componentes para tener más hueco para los tornillos y cables. A consecuencia, el chasis aumentó 7mm horizontalmente (el eje vertical, se quedó intacto). Los márgenes añadidos fueron entre los motores delanteros y el Arduino, la distancia de ambos motores traseros con el controlador, y el hueco de la batería para que no entre tan forzadamente.

Estéticamente quedó prácticamente igual a la figura 3.5.1 y 3.5.2 ya que solo se modificaron pequeños márgenes.

Construcción

Imagen que contiene motor

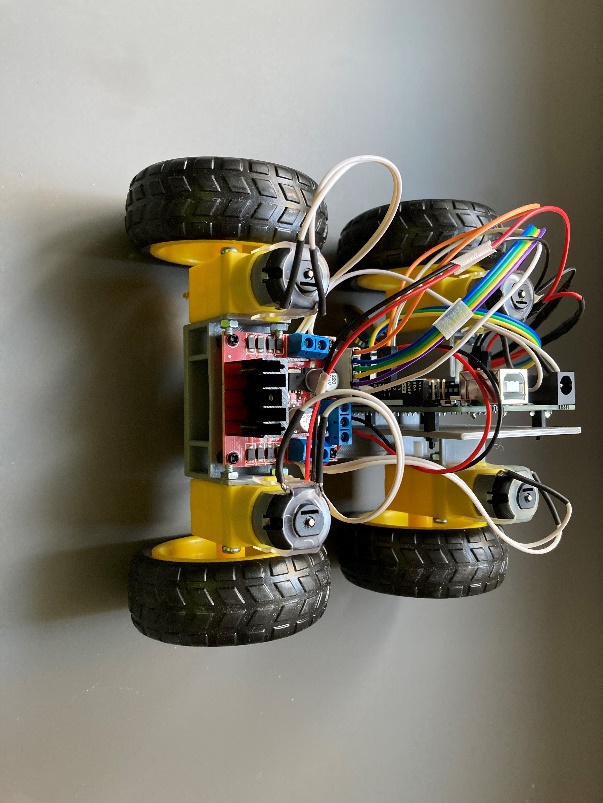
Descripción generada automáticamenteEsta iteración prometía ser la definitiva y por ese motivo ya se conectaron todos los componentes con intención de hacerlos todos funcionar dejando el robot con el siguiente aspecto:

Figura 3.6.2

Figura 3.6.1

Pruebas

**Primera prueba:** en la iteración anterior ya se hicieron pruebas a los motores, de esas pruebas se sacó la velocidad en línea recta y la velocidad de giro. Ahora se tenía que incorporar los sensores de línea. Para estas pruebas, se utilizó varios folios seguidos con líneas negras impreses simulando él una recta del mapa.

El resultado obtenido fue bastante bueno, el código se puede ver en la figura 3.6.1, el robot reaccionaba bastante rápido al salirse de las líneas, pero, aun así, tardaba en frenar por la inercia que llevaba. Para este problema hay tres soluciones posibles, la primera es invertir el sentido de los motores para reducir la distancia de frenado, la segunda es mover los sensores ligeramente hacia la parte delantera del chasis para detectar antes el cambio de línea, y la última es tener en cuenta esta distancia de frenado para empezar a realizar la acción justo antes de llegar al lugar en el que tiene que se tiene que llevar acabo.

De estas dos soluciones, la segunda es la más prometedora, no tan solo porque es la más simple (tan solo hay que mover 6 agujeros en el diseño), pero también será la más consistente al no añadir funcionalidad extra al código que potencialmente podría aumentar el error acumulativo de la trayectoria de cada robot.

[…]

void loop() {

  int izquierda = digitalRead(SENSOR\_I);

  int centro = digitalRead(SENSOR\_C);

  int derecha = digitalRead(SENSOR\_D);

  if(izquierda == LINEA && centro == PASILLO && derecha == LINEA){ //recto

    recta(true);

    analogWrite(velocidadIzquierda, VELOCIDAD\_NORMAL);

    analogWrite(velocidadDerecha, VELOCIDAD\_NORMAL);

  }

  else{ //parada

      digitalWrite(delanteDerecha, LOW);

      digitalWrite(atrasDerecha, LOW);

      digitalWrite(delanteIzquierda, LOW);

      digitalWrite(atrasIzquierda, LOW);

  }

Código 3.6.1

**Segunda prueba:** en esta prueba había que mantener al robot en línea recta durante al menos 1 metro, en la prueba anterior el robot no giraba, en el momento en el que se desviaba un poco y perdía de vista a las líneas de seguimiento, se paraba en seco. Durante este metro, el robot se desviará, pero con la información de los sensores, tendría que ser capaz de enderezar la dirección.

Esta prueba fue un poco más complicada al ya tener que incorporar comportamiento dependiendo del estado cambiante de los sensores. La idea inicial era que para este comportamiento se usaran tan solo los sensores laterales. Si tanto el sensor derecho como el izquierdo estaban viendo una línea, es que íbamos en buena dirección. En cambio, si el sensor derecho dejaba de ver la línea y el izquierdo la seguía viendo, significaba que nos habíamos desviado un poco hacia la izquierda, por tanto, hay corregir la dirección ligeramente hacia la derecha.

En la siguiente imagen vemos como quedaba una vez montado. El mapa está representado por varios folios con líneas impresas ya que las dimensiones de las líneas aun no eran definitivas y de esta manera era más fácil de prototipar.

Captura de pantalla de un celular

Descripción generada automáticamente

Figura 3.6.3

El robot era capaz de recorrer 4 folios manteniéndose por encima de las líneas negras, pero no era fiable, no todas las veces era capaz de recorrer la distancia entera. Este problema había que solucionarlo, el robot tendría que ser capaz de recorrer los metros que sean y siempre por encima de las líneas.

Tras unas pruebas más para investigar el origen del problema, observé que las líneas negras no eran lo suficientemente gordas, había veces que dos sensores estaban sobre la franja blanca que divide a las líneas. Aumentar el grosor de las líneas negras reduciría el grosor de la franja blanca central y aumentaría el margen a cada lado.

Con este cambio y algún otro en el lado del software, el robot ya era capaz de recorrer 4 folios consistentemente y sin la necesidad de empezar perfectamente en paralelo con las líneas.

El método de giro es relativamente simple, una vez se detecta una anomalía (hay que corregir la dirección), se paran los motores del lado hacia el que hay que corregir la dirección durante un instante (45 milisegundos), y a continuación se encienden todos los motores para avanzar en línea recta durante otro instante (75 milisegundos) metiendo al robot de nuevo en el rango de las dos líneas.

Se utiliza esté método de girar un poco y avanzar para salir de la zona de “riesgo” ya que si tan solo girásemos el robot, como el giro realizado ha sido muy suave, los sensores seguirían detectando que estamos fuera de posición. Por tanto, se ha añadido una etapa de avance en línea recta para volver a entrar en el rango de las dos líneas.

En la siguiente figura se puede ver una representación simplificada del sistema de giro.

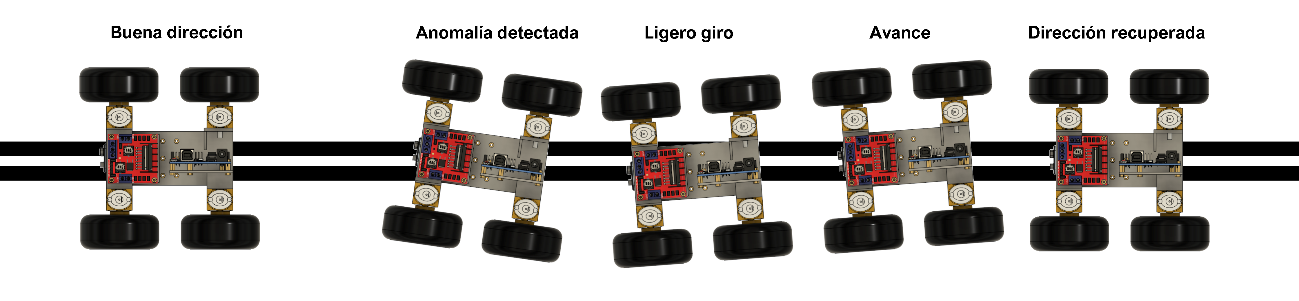


Figura 3.6.4

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamenteEl código se puede ver a continuación (se ha omitido la declaración de variables globales y la inicialización de los pines del Arduino).

Figura 3.6.6

Este método daba resultados fiables en cuanto al seguimiento de línea, pero me topé con un problema a la hora de hacer mas pruebas. En el momento en el que el software se complicaba un poco más y se le pedía más funcionalidades, la batería no era suficiente para todo. Con los cuatro motores al 100%, los tres sensores, el Arduino y el controlador, la pila se quedaba corta.

Explorando soluciones para esto, me di cuenta de que estaba controlando la velocidad de los motores de manera binaria, o encendidos al 100% o apagados. Este era el factor que causaba estos problemas. Cambié el control a un control analógico para poder bajare la potencia a los motores. De esa manera, cada motor iba al 70% (aproximadamente) de su capacidad con lo que la batería ya era capaz de alimentar todo. No solo esto, sino que, gracias a este sistema, el mapa se podría hacer más pequeño ya que podemos ajustar la velocidad del robot para recorrer menos distancia. También, podemos refactorizar el código de giro para hacerlo más simple. Ahora, en vez de hacer lo mostrado en la figura 3.6.3, se puede tan solo bajar la velocidad de los motores del lado hacia el que hay que corregir la dirección y queda un movimiento mucho más suave y un código mucho más limpio.

**Tercera prueba:** en esta prueba había que conseguir que el robot fuera capaz de realizar giros de 90 grados hacia una dirección.

Inicialmente, tras la primera prueba, la idea era mover los sensores hacia la parte delantera del robot para detectar con un poco de antelación cuando hay que girar ya que el robot tarda unos centímetros en frenar, pero esto supondría un nuevo diseño y una nueva iteración. Por eso, antes de dedicarle un par horas en rediseñar, imprimir y reconstruir el robot decidí ver como de fiable era marcar el mapa con antelación.

Tras unas cuentas pruebas conseguí que el robot girara al llegar a una intersección. El giro se conseguía al invertir el sentido de los motores durante un tiempo determinado para girar sobre si mismo. Una vez el giro se había realizado, el robot avanzaría en línea recta unos 200 milisegundos para salir de la intersección, y volver al modo seguimiento de línea.

Código 3.6.3

  [...]

  if(izquierda == LINEA && centro == PASILLO && derecha == LINEA){ //recto

    [...]

  }

  else if(izquierda == LINEA && centro == LINEA && derecha == LINEA){ //Giro 90 grados

    recta(false);

    delay(100);

    noventagrados(false);

    parada();

    delay(100);

    recta(true);

    delay(300);

  }

  else if (izquierda == PASILLO && derecha == LINEA) { //desvio derecha

    [...]

  }

  else if(izquierda == LINEA && derecha == PASILLO){ //desvio izquierda

    [...]

  }

  else  [...]

Esta prueba dejó un mal sabor de boca. Pese a que el robot era capaz de realizar giros de 90 grados, no siempre los podía hacer. Había ocasiones en las que el robot no entraba perfectamente recto a la intersección y por tanto al salir, tampoco salía entre las dos líneas haciendo que no fuera capaz de seguir el recorrido.

Otro problema, y este más importante, era que los motores no tenían la misma potencia a lo largo de la vida de la batería. Cuando la batería estaba recién cargada, el robot iba ligeramente más rápido, no mucho, pero lo suficiente como para girar más de 90 grados en el tiempo que debería haber girado solo 90. Tras unas cuentas pruebas más cambiando el sistema de representación del mapa, decidí que había que explorar otras soluciones. Tener preestablecido el tiempo de rotación no iba a ser una solución fiable. La solución más llamativa y prometedora era la incorporación de un giroscopio y utilizar los datos de este para saber cuánto tenemos que girar exactamente independientemente de la velocidad de los motores.

Conclusión

Sin lugar a duda, esta iteración ha servido para mucho, con ella hemos conseguido un prototipo capaz de seguir líneas el 100% de las veces y capaz de realizar giros, pero no lo suficientemente bien como para quedarse en el proyecto.

Para la siguiente iteración nos llevamos una representación de líneas definitiva, un seguimiento lineal sólido capaz de autocorregirse con movimientos suaves, y un sistema de detección de intersecciones. Em la próxima iteración, se tendrá que incorporar un giroscopio y conseguir unos giros de 90 grados precisos.

3.7 Quinta iteración

Diseño del robot

Para esta iteración tan solo había que añadir un componente nuevo por tanto se podía partir relativamente fácilmente del chasis de la iteración anterior. El problema con el nuevo componente a añadir, el giroscopio, es que tiene que estar orientado y posicionado de una forma precisa para su correcto funcionamiento por tanto no se podía poner de forma vertical aprovechando algún hueco disponible.

Se tuvo que mover el controlador de los motores un poco hacia arriba (20mm) para dejar hueco al giroscopio entre el controlador y el chasis ya que este componente es bastante más pequeño que el controlador.

Al volver a rediseñar el chasis, se aprovechó para hacerle ligeras modificaciones como por ejemplo ensanchar el diámetro de los agujeros para los motores y añadir un nervio en el soporte de los motores traseros ya que podían llegar a flectar un poco al aplicarle peso.

Diagrama

Descripción generada automáticamenteDiagrama, Dibujo de ingeniería

Descripción generada automáticamenteEn las siguientes imágenes podemos ver un plano cercano de los cambios realizados.

Figura 3.7.2

Figura 3.7.1

Construcción

Por primera vez desde el inicio del proyecto, durante la construcción de esta iteración no se identificaron posibles mejoras, márgenes erróneos, lugares débiles… Dejando un robot con un aspecto similar a la iteración anterior.

Imagen que contiene motor

Descripción generada automáticamente

Figura 3.7.4

Figura 3.7.3

Figura 3.7.4

Pruebas

**Primera prueba:** lo primero de todo es realizar una prueba con el giroscopio propia del apartado 3.1 de prototipado pero que no se hizo en su día ya que entonces pensaba que este componente no era necesario.

La intención de la prueba era ser capaz de leer el *pitch*, *yaw* y *roll* del componente aun que realmente solo será necesario usar el *yaw*.

Para este componente es necesario el uso de las librerías externas *I2cdev* y *Simple\_MPU6050* ya que el giroscopio es capaz de enviar mucha información y se necesita deserializarla correctamente.



Código 3.7.1

Tabla

Descripción generada automáticamenteEste código (se ha omitido la inicialización del componente) saca por el monitor seria los 3 ejes del robot con bastante precisión como se puede ver en la siguiente figura.

Figura 3.7.5

**Segunda prueba:** esta era la prueba importante ya que había pasar del método anterior de giro (definido por un tiempo fijo) al nuevo sistema (definido por ángulos de rotación) para evitar el problema de la carga de batería (cuanto más cargada estaba, más giraba el robot).

En cada bucle da la lógica se ejecutan 3 acciones principales, primero de todo se hace la lectura de los sensores de línea, a continuación, se hace la lectura de la rotación, y por último se controlan los motores. Es importante mantener este orden ya que cada acción depende de la anterior.

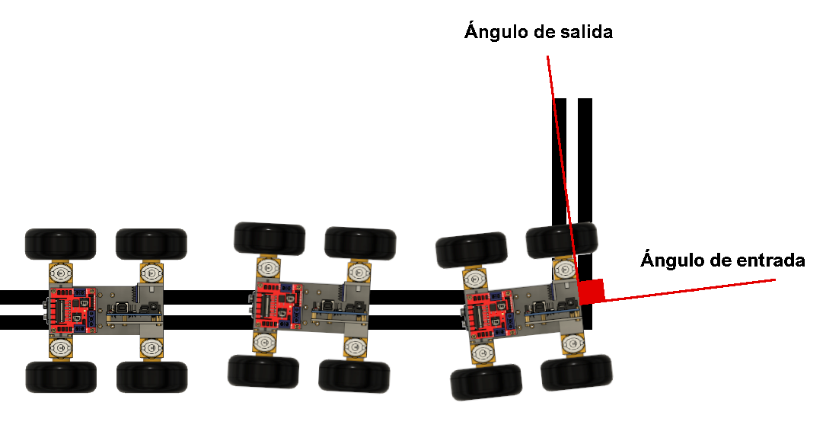
Durante todo el tiempo en el que el robot está yendo en línea recta, el giroscopio está leyendo datos y calculando su ángulo de rotación medio. Cuando hablo de ángulo medio, me refiero a la dirección media que lleva el robot desde la última intersección. Esto es necesario ya que, al llegar a la próxima intersección, el robot podría estar justo corrigiendo su trayectoria y no haber entrado a la intersección perfectamente en perpendicular a ella, dando como resultado un giro incorrecto (figura 3.7.6).

Figura 3.7.6

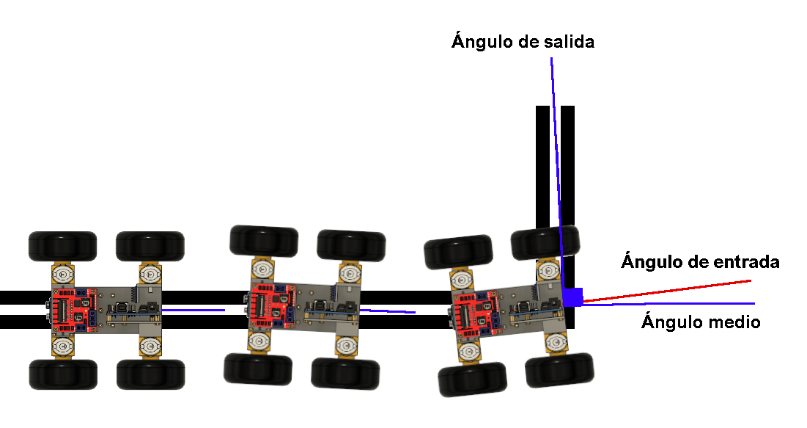
En cambio, si para calcular el ángulo de salida utilizamos el ángulo medio que se ha ido calculando a lo largo de la recta obtendremos un resultado como el siguiente:

Figura 3.7.7

Sigue sin ser un resultado 100% preciso ya que el ángulo medio no será perfecto, pero es suficiente para que luego el propio robot con los sensores de línea sea capaz de auto corregirse en la dirección ideal dando unos resultados mucho más fiables que los de la figura 3.7.6.

Cuando el robot sale de una intersección entra en un estado “de peligro” ya que corre el riesgo de perderse. Por ello, aunque el giroscopio es bastante precios, una vez se ha efectuado el giro, se vuelve a revisar si estamos en la orientación correcta. Es necesaria esta comprobación ya que el robot corta la corriente a los motores al llegar a 90 grados de giro, pero este aun lleva inercia y puede girar un poco más. Este ángulo de giro extra es extremadamente complicado de prever ya que no solo depende de los motores, puede depender también de si hay viento, de si justo esa parte del mapa tiene más fricción o menos… Por este motivo, justo al terminar un giro, durante los siguientes 250 milisegundos (como máximo), el robot se dirige con el giroscopio hasta alcanzar el rumbo óptimo, en ese momento, se vuelve a los sensores de línea.

Con este nuevo método, el robot ya era capaz de dar vueltas constantemente a un circuito cuadrado de líneas en el suelo siempre y cuando sean giros de 90 grados.

Conclusión

Muy posiblemente esta es la iteración definitiva. El robot ya es capaz de realizar las tareas básicas que le corresponden de manera fiable. La única mejora para el siguiente chasis es un bordillo en el compartimento de la batería para impedir que esta se mueva en las aceleraciones y declaraciones.

4. Desarrollo del software

Con el robot ya desarrollado, era hora de pasar al apartado del software. En este apartado se conectará el robot a un Arduino externo vía bluetooth y este estará controlado por el ordenador.

Como en la sección de prototipado ya se habían hecho pruebas de conectividad entre dos Arduinos vía bluetooth, lo importante ahora era averiguar la comunicación entre un Arduino y un proyecto de java.

4.1 Comunicación Java-Arduino

Para la comunicación entre un proyecto java y un Arduino usaré una librería llamada jSerialComm que permite mandar y recibir datos por puerto serial del Arduino.

Lo primero de todo tras haber inicializado la librería (código 4.1.1) y haberla enlazado al Arduino que estuviera en ese momento conectado era sincronizar ambos programas (código 4.1.2). La parte de java es bastante más rápida que la de Arduino, y pues esta tiene que esperar a recibir una llamada del Arduino avisándole de que ya está inicializado y listo para ejecutar instrucciones. A continuación, para la primera prueba, desde java se manda un mensaje para cambiar el color de una luz led conectada al Arduino (código 4.1.3). Finalmente, es importante cerrar el puerto serie para poderlo abrir cuando se vuelva a ejecutar el programa (código 4.1.4).

Código 4.1.1

System.***out***.println("Vamos a inicializar el arduino");

SerialPort sp = SerialPort.*getCommPort*("/dev/ttyACM0");

sp.setComPortParameters(9600, 8, 1, 0);

sp.setComPortTimeouts(SerialPort.***TIMEOUT\_READ\_BLOCKING*** | SerialPort.***TIMEOUT\_WRITE\_BLOCKING***, 0, 0);

if(sp.openPort()) {

System.***out***.println("Puerto abierto");

sp.getInputStream().close(); //limpiamos la cadena de entrada por si ha quedado algo pendiente de otra iteración

}

else {

System.***out***.println("ERROR: No se pudo abrir el puerto");

return;

}

try {

Integer i = 1;

sp.getOutputStream().write(i.byteValue());

sp.getOutputStream().flush();

System.***out***.println("DatoMandado");

} catch (IOException e) {

e.printStackTrace();

System.***out***.println("ERROR: No se ha podido mandar el dato");

}

Código 4.1.2

boolean start = false;

while(!start) {

int byteRead = 0;

try {

byteRead = sp.getInputStream().read();

} catch(IOException e) {

System.***out***.println("ERROR: Ha fallado la lectura de datos");

}

if(byteRead == 10) {

System.***out***.println("Dato correcto");

start = true;

}

else

System.***out***.println("Dato incorrecto");

}

Código 4.1.4

if(sp.closePort()) {

System.***out***.println("Puerto cerrado");

}

else {

System.***out***.println("ERROR: no se ha podido cerrar el puerto");

}

Código 4.1.3

4.2 Comunicación Java-Arduino-Arduino

El siguiente paso es conectar vía bluetooth al Arduino “servidor” con el Arduino “cliente” (el robot).

Recordemos que es necesario tener un Arduino entre los robots y el simulador de java ya que un pc tan solo es capaz de controlar un solo dispositivo bluetooth de este tipo. De este modo, tendremos conectado al PC un Arduino con cinco módulos bluetooth para controlar los cinco robots.

Al principio empecé haciendo pruebas implementando el código del apartado 3.3.3. En estas pruebas era capaz de mandar estructuras de datos, pero tan solo en una dirección. De este modo, la comunicación era java→ arduino → arduino. Este método no era la suficientemente viable ya que para el proyecto final se necesitará una comunicación bidireccional.

Tras muchas pruebas, averigüé que la librería que estaba utilizando (SoftEasyTransfer) no servía para comunicación bidireccional, por tanto, creé mi propio sistema de comunicación.

Este nuevo sistema propio tan solo es capaz de mandar *enums* entre dispositivos. Tan solo con esto, ya era suficiente. Por organización de código dividí los mensajes en dos *enums*, tenemos el *enum* con los mensajes de comunicación entre Arduinos y otro *enum* con los mensajes de comunicación entre arduino y el PC.

[insertar fotos de los enums]

El siguiente paso era sincronizar los 3 dispositivos. Esta tarea no fue sencilla ya que tenia que ser lo suficientemente robusta para funcionar independientemente del orden de encendido de los dispositivos. Durante esta fase de sincronización, el arduino servidor se queda en espera hasta recibir el mensaje *SYNC\_ATTEMP* del PC, una vez lo ha recibido, devuelve el mensaje *SYNC* indicando al PC que ambos dispositivos tienen una comunicación establecida y van a la par.

Una vez el arduino servidor y el PC están sincronizados, es hora de sincronizar el robot como tal. Siguiendo el mismo modelo que antes, el robot está esperando recibir el mensaje *SYNC\_ATTEMP* del arduino servidor y una vez recibido, devuelve el mensaje *SYNC* al servidor. El siguiente paso, es inicializar los componentes del robot. Como ahora se consta de un giroscopio, hay que esperar a que este se calibre, y no siempre tarda lo mismo. Por tanto, cada vez que el robot ha inicializado un componente, manda un mensaje al servidor avisando de qué componente se ha inicializad. A continuación, el robot no pasará a inicializar el siguiente componente hasta no recibir el mensaje *OK* del servidor indicando que se ha recibido el mensaje correctamente.

Para facilitar la depuración, el arduino servidor tiene una luz led rgb la cual indica el estado en el que se encuentra el sistema en ese instante.

* **Luz roja:** ningún dispositivo está sincronizado
* **Luz azul:** el arduino servidor y el PC están sincronizados y pueden intercambiar mensajes
* **Luz verde:** el robot y el arduino servidor están sincronizados. Solo es posible llegar a este estado habiendo pasado por el estado azul previamente, por tanto, si la luz es verde, se pueden mandar mensajes desde el PC al robot pasando por el servidor y viceversa.

[INSERTAS FOTOS DE LOS 3 ESTADOS DE LAS LUCES]

Bibliografía

Documentación de Arduino: <https://www.arduino.cc/en/main/docs>

Tutoriales básicos de componentes individuales de Arduino: <https://www.youtube.com/channel/UC4unPLtykzwO7MB3IvaQZaA>

Librería EasyTransfer/SoftTransfer: <https://github.com/madsci1016/Arduino-EasyTransfer>

Librería I2cdev: [https://github.com/jrowberg/i2cdevlib](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbV9oSU43RDZTR3lob1ZFTU9MQUJxbkJTdTlpd3xBQ3Jtc0tsWm1xc2RGdlplalRST2RFeE44d0tPbV9NRDJ3NkZQLVZSaDYwSU5vaWp0V3JvcVA1Y2VaSUF1Sk1EcXF5bm1kcnNteEp4a1dXSi1Cb3c4aDRQN0o0dVA4eTNxcjFPZVBQOElhVWdiNHl3YXZQUXZNYw&q=https%3A%2F%2Fgithub.com%2Fjrowberg%2Fi2cdevlib)

Librería Simple\_MPY6050: <https://github.com/ZHomeSlice/Simple_MPU6050>

Librería jSerialComm: <https://github.com/Fazecast/jSerialComm>