UNIVERSIDAD
REY JUAN CARLOS

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

Curso Académico 2019/2020

Trabajo Fin de Grado

RED DE CARRETERAS UTILIZANDO MICROBIT Y GIGGLEBOT

Autor : Tomás Galindo Encinas

Tutor : Pedro de las Heras Quirós

Trabajo Fin de Grado/Máster

Red de Carreteras con utilizando Microbit y Gigglebot

Autor : Tomás Galindo Encinas

Tutor : Pedro de las Heras Quirós

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 2020, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2020

Resumen

Este trabajo de fin de grado está diseñado para la posible adaptación de las tecnologías utilizadas al ámbito de la docencia. El objetivo es utilizarlo en asignaturas que sirvan para el inicio de los alumnos en la programación.

Para lograr este objetivo, se ha elaborado una red de carreteras como práctica para enseñar a programar a alumnos del primer curso de ingeniería de robótica software. Por un lado se ha planeado el diseño de un circuito de carreteras y los elementos que lo componen. Por otro lado se ha programado el comportamiento de los robots para que puedan circular por dicho circuito sin salirse. Para ello, utilizan unos sensores que captan la intensidad de color (en escala de grises) y le permite identificar en qué parte de la carretera se encuentra para seguir la dirección correcta. Los elementos de tráfico que se han diseñado en este trabajo son: salida de la carretera, entrada de carretera, semáforos, señal de cambio de velocidad, señal de stop y rotonda. Estos elementos serán representados por un código de barras, gracias a los cuales el coche (una vez leídos) reconocerá en que elemento se encuentra y actuará en consecuencia. Por otro lado, también existe una interacción entre los diferentes robots que circulan por el circuito, adaptando su velocidad para no chocar unos con otros.

Para el desarrollo de este cometido se han utilizado las placas Microbit y robots Gigglebot que son los elementos que serán programados y se han utilizado dos entornos de programación, MuEditor [7] y MakeCode [4], en los que se utilizaron los lenguajes MicroPython y Static TypeScript respectivamente para realizar el código.

Tras la realización de diferentes pruebas y analizando los resultados podemos concluir que, aunque no se han podido realizar todos los objetivos propuestos, los resultados obtenidos son buenos y prometedores en cuanto a la utilización de estas tecnologías para la docencia, ya que se puede observar el gran potencial y la capacidad de adaptación que tienen estas tecnologías para poder aplicarlas en clases.

Índice general

Índice de figuras

Índice de cuadros

ÍNDICE DE CUADROS

Capítulo 1

Introducción

En este proyecto se ha elaborado una red de carreteras como práctica para enseñar a programar a alumnos del primer curso de ingeniería de robótica software. Por lo tanto, los alumnos se centrarán exclusivamente en la creación del código para lograr el funcionamiento del robot.

No obstante, para la utilización del código y a su vez, para la comprobación del funcionamiento de los robots, ha sido necesario la elaboración de componentes externos ajenos al código de funcionamiento para el robot. Este proyecto se puede dividir en dos partes:

- Programación: En esta parte se puede englobar todo lo relacionado con el código: investigación de las tecnologías, estudio de lenguaje óptimo para la programación del código, elaboración del código.

- El diseño: En esta parte se agrupa todo lo relacionado con el diseño de los aspectos externos: diseño base de las carreteras, diseños de las curvas, los elementos de la carretera (salida de la carretera, entrada de carretera, semáforos, señal de cambio de velocidad, señal de stop y rotonda), elaboración de un circuito en el que se puedan realizar pruebas y de otros circuitos como el de calibración.

En este apartado se presentan la motivación de la realización del proyecto, los objetivos propuestos y también se explican las tecnologías utilizadas.

1.1. Motivación

La iniciación en la programación en la robótica puede ser una tarea complicada debido a que principalmente los microcontroladores y sensores están desarrollados para ser utilizadas con el lenguaje de programación C.

C es un lenguaje de bajo nivel y como consecuencia hay que tener en cuenta los procesos de gestión y de manejo de la memoria porque no resulta sencillo para personas que se están iniciando en el mundo de la programación.

Por ello, en este trabajo se pretenden utilizar las tecnologías de Microbit y Gigglebot, que utilizan lenguajes de más alto nivel como Static TypeScript o Python. Con estos lenguajes evitamos este tipo de problemas ya que lo gestionan de una manera interna, sin que el programador participe activamente en este proceso. Por lo tanto, la motivación de este trabajo es adaptar las tecnologías mencionadas anteriormente a un proyecto, en este caso una red de carreteras, para que en un futuro pueda utilizarse para la realización de clases de fundamentos de programación.

1.2. Objetivos

Este trabajo consiste en crear una red de carreteras funcional con las tecnologías de Gigglebot y Microbit. En este proyecto, todos los Gigglebot que permanezcan en el circuito deben mantenerse en el mismo y moverse de una forma independiente.

Para lograr el objetivo principal, se decidió dividirlo en objetivos más pequeños:

- Elegir las mejores características posibles para diseñar el circuito.
- Elaborar el código para que el coche permaneciera circulando en una carretera sin salirse.
- Implementar varias velocidades para el coche.
- Diseño de elementos reconocibles de la red de carreteras.
- Programación de una forma de avisar al Gigglebot en qué elemento se encuentra.
- Calibración de los Gigglebot para evitar los posibles problemas que se pueden producir debido a aspectos externos, como por ejemplo la energía de la batería o la intensidad de luz ambiental.

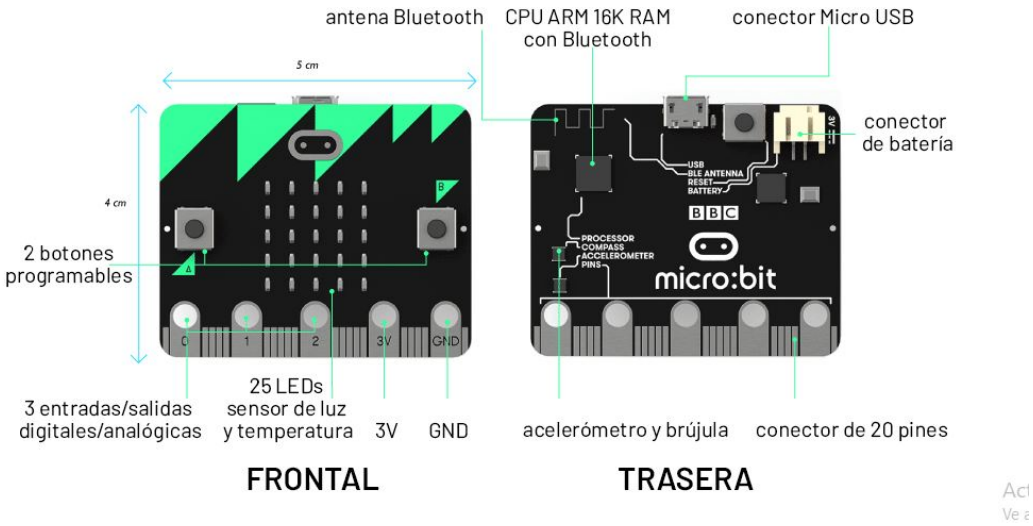

Figura 1.1: Tarjeta Microbit con sus características.

- Comunicación vía radio de los Gigglebot para lograr una interacción entre los diferentes elementos del circuito, como por ejemplo entre el semáforo un coche.
- Elaborar alguna función para que el coche pueda volver a la carretera si éste la ha perdido.
- Interacciones entre los diferentes Gigglebot que circulen por el recorrido.

1.3. Tarjeta Microbit

Microbit (también conocido como BBC Micro Bit o como micro:bit) es una tarjeta programable de pequeño tamaño pero con una gran versatilidad. Es un sistema embebido basado en un arquitectura de hardware ARM y de código abierto diseñado por la BBC para su uso en educación informática en Reino Unido.

Las características técnicas y elementos que tiene la tarjeta son las siguientes (Véase la figura 1.1):

- Microcontrolador ARM Cortex-M0 cuyas características más importantes son:
  - Memoria Flash de programa de 256Kb de capacidad.
  - Memoria RAM de datos de 16Kb.

  - Velocidad del oscilador de 16Mhz.

- Leds: 25 leds colocados en una matriz de 5x5, que pueden ser programados individualmente. Se pueden utilizar para visualizar números, palabras, o incluso figuras.

- Pulsadores programables: 2 botones, que actúan como dispositivos de entrada (nombrados como A y B) con los cuales se puede detectar si están presionados o cuántas veces se han pulsado y ejecutar acciones en función de ello.

- Pines de conexión física: Conector de 20 pines a través de los cuales puedes acceder a diferentes señales de entrada/salida del controlador y conectar otros tipos de periféricos con los que así poder extender la funcionalidad y hacer circuitos más grandes y complejos.

- Sensor de luz: Es un sensor con el cual podemos representar en una escala numérica el nivel de intensidad de luz que es capaz de detectar el sensor.

- Sistema de Radio frecuencia (RF): Sistema que permite enviar información entre Microbits y así poder establecer una comunicación entre ellas.

- Sensor de temperatura: Es un sensor gracias al cual podemos detectar la temperatura en grados Celsius.

- Acelerómetro: Es un sensor que sirve para medir la aceleración de la tarjeta, detecta el movimiento de la tarjeta.

- Brújula: Es un sensor magnetómetro que detecta el campo magnético de la tierra y se puede por tanto conocer la orientación de la tarjeta respecto al Norte. Para utilizar esta función es necesaria, primero, un fase de calibración.

- Bluetooth, BLE (Bluetooh Low Energy): La radio del Microbit incorpora la pila Bluetooth que permite al Microbit comunicarse y transferir archivos con móviles, ordenadores y viceversa.

- Puerto USB: Puerto de entrada a través de la cual podemos alimentar la tarjeta, y realizar la transferencia de los archivos.

- Puerto específico para alimentación de 2 pilas AAA o una batería.

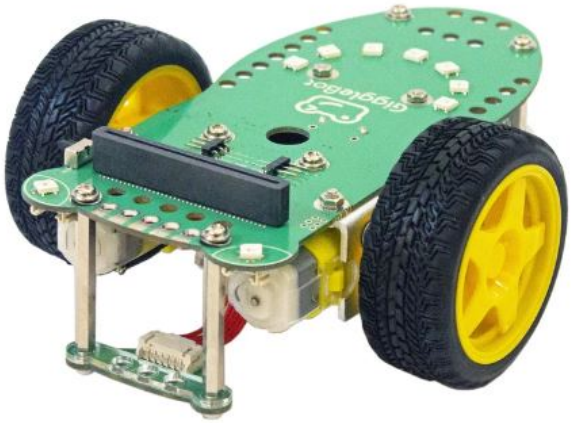

Figura 1.2: Robot Gigglebot.

- Botón de reset.

La comunidad que rodea a Microbit es muy grande y colaborativa. Además, en la página web [5] se pueden encontrar bastantes ejemplos y tutoriales con los que iniciarse en el mundo de Microbit y aprender a utilizar los diferentes sensores y elementos que forman parte de la tarjeta y esto es una ayuda muy útil para aprender a programar con este sistema de manera rápida y sencilla.

En este trabajo se han utilizado los siguientes dispositivos del Microbit: pines (para conectar al Gigglebot), leds (para mostrar mensajes por la pantalla), botones (para establecer el inicio de diferentes procesos), la radio (para establecer una comunicación entre Microbits), además del puerto USB (para hacer la transferencia de los códigos). Como se puede observar, la tarjeta tiene más opciones para aumentar la funcionalidad del trabajo y poder hacer innumerables cosas.

1.4. Robot Gigglebot

El robot Gigglebot es un periférico diseñado para la tarjeta Microbit gracias al cual se obtiene un robot completamente funcional dotado de movimiento, y con el que es posible iniciarse en la robótica.

Las funcionalidades propias del Gigglebot son las siguientes:

- Siete leds programables multicolores.

- Aberturas para colocar otras piezas como por ejemplo (Lego) y así incluirle más funcionalidades.
- Dos ruedas controladas por dos motores independientes para permitir el movimiento del robot
- Dos sensores de luz con los cuales se puede detectar la intensidad de la misma.
- Dos sensores de intensidad lumínica con los cuales es posible detectar los cambios de intensidad del suelo (escala de grises) y poder seguir líneas.
- Una abertura diseñada para introducir un rotulador y hacer que el robot dibuje cosas.
- El robot está alimentado por 3 pilas AA.
- Conexiones para dos servo-motores.
- Fila de Pines igual que el Microbit.
- Dos puertos I2C para conectarle otros sensores como:
    - Sensor de Tiempo atmosférico: mide temperatura y presión y humedad del ambiente.
    - Sensor de distancia: Medir la distancia con objetos frente al sensor

La Tarjeta Microbit se conecta al Gigglebot, por lo tanto, además de la funcionalidades propias del robot, se le suman las funcionalidades de la tarjeta. En este proyecto se han utilizado las siguiente funcionalidades: leds, motores, un puerto I2C. Para obtener información más detallada de este periférico véase la pagina [2]

1.5. Sensor de distancia

Es un sensor láser que nos permite obtener la distancia de cualquier objeto que se encuentre delante de él. Se conecta al Gigglebot a través de un puerto I2C. El funcionamiento del sensor se basa en el método de tiempo de vuelo: consiste en mandar un pequeño pulso de un láser invisible y recibir la luz de vuelta y basándose en el tiempo que tarda en llegar obtenemos la medida de la distancia.

Así, este sensor se utilizará en el proyecto para obtener información de los objetos que se encuentren enfrente de nuestro robot y actuar dependiendo de la distancia. Se utilizará la función que tiene el sensor de leer la distancia de forma continua, así los cambios que se produzcan se observarán en tiempo real y el robot podrá actuar en consecuencia.

1.6. Entornos de programación

Como se mostró anteriormente, tanto Microbit como Gigglebot son herramientas utilizadas principalmente para la docencia y hechas principalmente para el inicio en la programación. Hay que tener en cuenta que la mayoría de los microcontroladores se programan en lenguaje C, que no es un lenguaje muy óptimo para el aprendizaje en niveles muy bajos. Actualmente los lenguajes de *scripting* (JavaScript y Python), son lenguajes más sencillos de aprender y cómodos que están adquiriendo importancia en lo que se refiere a los microcontroladores.

Por lo tanto los entornos utilizados en este trabajo fueron dos. En primer lugar se utilizó MuCode, que utiliza MicroPython y en segundo lugar Makecode que utiliza Satic TypeScript.

1.6.1. MuCode

Es un entorno que utiliza MicroPython, es muy versátil ya que puede ser instalado en cualquier sistema operativo. Utiliza como lenguaje MicroPython, que es un pequeño pero eficiente intérprete del lenguaje Python. Incluye un subconjunto mínimo de librerías y que además está optimizado para que pueda correr en microcontroladores. Desde este entorno podemos utilizar funciones propias de las librerías de Python y también funciones propias de Microbit para controlar los sensores de la tarjeta. Además se nos facilita una página [1] donde se encuentran dichas funciones.

Para poder utilizar el Gigglebot y el sensor de distancia es necesaria la carga de dos bibliotecas en la memoria de la tarjeta. Esto empeoró el funcionamiento de la tarjeta Microbit.

Los microcontroladores suelen usar lenguajes a bajo nivel como C o C++, sin embargo para utilizar lenguajes de alto nivel, como Python, es necesario incluir un intérprete y para ello hay que guardarlo en la memoria de la tarjeta. El problema es que los interpretes ocupan mucho y no dejan mucho espacio para realizar programas demasiado complejos.

Una de las soluciones para reducir el uso de la memoria fue utilizar un programa para reducir

la memoria del archivo, reduciendo los espacios en blanco, comentarios y aspectos que no son útiles de cara a la ejecución del programa (Python Minifier [6]). Por otro lado, se intentó reducir el número de variables y el número de líneas del código. Sin embargo, únicamente se pospuso el problema, ya que al tener que implementar más funcionalidades del robot, volvieron a surgir los problemas de la memoria. Finalmente se decidió cambiar a Static TypeScript, ya que es un lenguaje compilado que genera código máquina que ocupa menos.

1.6.2. Makecode

Es un plataforma que se ejecuta desde un navegador web que tiene varios entornos de programación destinados a diferentes fines. (Adafruit, Microbit, Minecraft, Lego Mindstorm...). En este trabajo se utiliza un entorno destinado para Microbit.

Cada una de estos frameworks está compuesto de :

- Un simulador: en el que puedes ver en directo el resultado de lo que vayas programando.
- Editor con bloques: Entorno de programación visual en el que el código viene estructurado en bloques y vas arrastrándolos haciendo las diferentes funciones del robot. Es un entorno muy ameno y hace posible que sin tener apenas nociones de programación, se puedan ejecutar programas sencillamente.
- Editor Javascript: Esta es la parte de código para los estudiantes que tienen conocimiento en programación, con lo cual se pueden hacer programas más avanzados. Tiene herramientas de autocorrección y de autocompletado para facilitar la tarea.

Este entorno utiliza el lenguaje Static TypeScript (STS): Es un subconjunto de TypeScript. Está diseñado para la práctica (educación), siendo también utilizable para la compilación estática para dispositivos pequeños. Un programa de STS se compila a código máquina produciendo un ejecutable que es más eficiente, consiguiendo que las baterías duren más y haciendo posible hacer correr programas en dispositivos que tienen una capacidad limitada de memoria RAM (como las Microbit).

Para poder usar las funciones de Gigglebot y del sensor de distancia es necesario cargar la biblioteca giggle.

Capítulo 2

Diseño e implementación

En este capítulo se explica detalladamente el funcionamiento completo del robot así como el diseño del circuito y de los elementos que lo forman. En la primera sección se aporta una visión general del proyecto entero y en las sucesivas se explican en profundidad los diferentes aspectos en los que se divide nuestro trabajo.

2.1. Visión general

El objetivo de este trabajo es la programación del funcionamiento de robots (Microbit + Gigglebot). Deben poder circular e interactuar con los diferentes elementos de una red de carreteras y además deben poder interactuar entre sí, para que puedan circular varios Gigglebot por el mismo circuito sin problemas. Para lograr ese objetivo es necesario el diseño de un circuito simulando una red de carreteras incluyendo varios elementos reconocibles de dicha red.

Los robots están formados por microprocesadores y sensores, por lo tanto para facilitar la programación de dichos elementos, el código partirá de la idea de una máquina de estados [8].

Una máquina de estados es un modelo con el cual se puede describir con total detalle el comportamiento de los robots. Para entender este concepto debemos tener en cuenta estos tres conceptos:

- Estado: Las situaciones en las que se encuentra el robot.
- Entradas: Datos recibidos por los sensores y otras variables, que provocan las transiciones.

- Transición: es el cambio de un estado a otro que pueden tener efectos sobre los actuadores.

Si la máquina de estados está bien implementada, el robot debe encontrarse en un estado en todo momento. En cada estado, el robot se encarga de realizar las acciones propias de dicho estado y de obtener las entradas que le llegan. Una vez con todos los datos necesarios (estado actual + entradas) realizará la transición a otro estado. Finalmente, se iniciará este mismo proceso, sólo que en un estado diferente.

Los elementos de tráfico que se han diseñado en este trabajo son: salida de la carretera, entrada de carretera, semáforos, señal de cambio de velocidad, señal de stop y rotonda. Estos elementos serán representados por un código de barras, gracias a los cuales el coche (una vez leídos) reconocerá en que elemento se encuentra y actuará en consecuencia.

Por otro lado, también existe una interacción entre los diferentes Gigglebot que recorran el circuito, los coches deberán adecuar la velocidad dependiendo de los obstáculos que encuentren en la vía.

En un principio, el objetivo era realizar un único circuito en el que se pudieran observar todas las interacciones. Sin embargo, las dimensiones del circuito obtenido eran demasiado grandes (12mx10m) como para poder colocarlo en algún sitio, y además económicamente no era rentable (Véase la figura 2.1).

Debido a esos motivos, se decide hacer el circuito más pequeño. Sin embargo para poder incluir todos los elementos realizados en este trabajo es necesario la elaboración de dos circuitos. En la figura 2.2 y en la figura 2.3 se muestran los dos circuitos diseñados en los cuales se realizaron las pruebas de funcionamiento de los Gigglebot.

2.2. Diseño de carreteras

El diseño de las carreteras se basó en el trabajo de Loic Debois [9] y Benjamin Kern [10]. En estos trabajos se realiza una comparación con diferentes tipos de carreteras y explican las ventajas y desventajas de cada uno de ellos. Se llega a la conclusión de que el diseño óptimo para este tipo de carreteras es de estilo degradado. Las ventajas que se consiguen utilizando este diseño son:

- Un movimiento del Gigglebot fluido debido al cambio lineal en el grado de intensidad de blanco a negro.

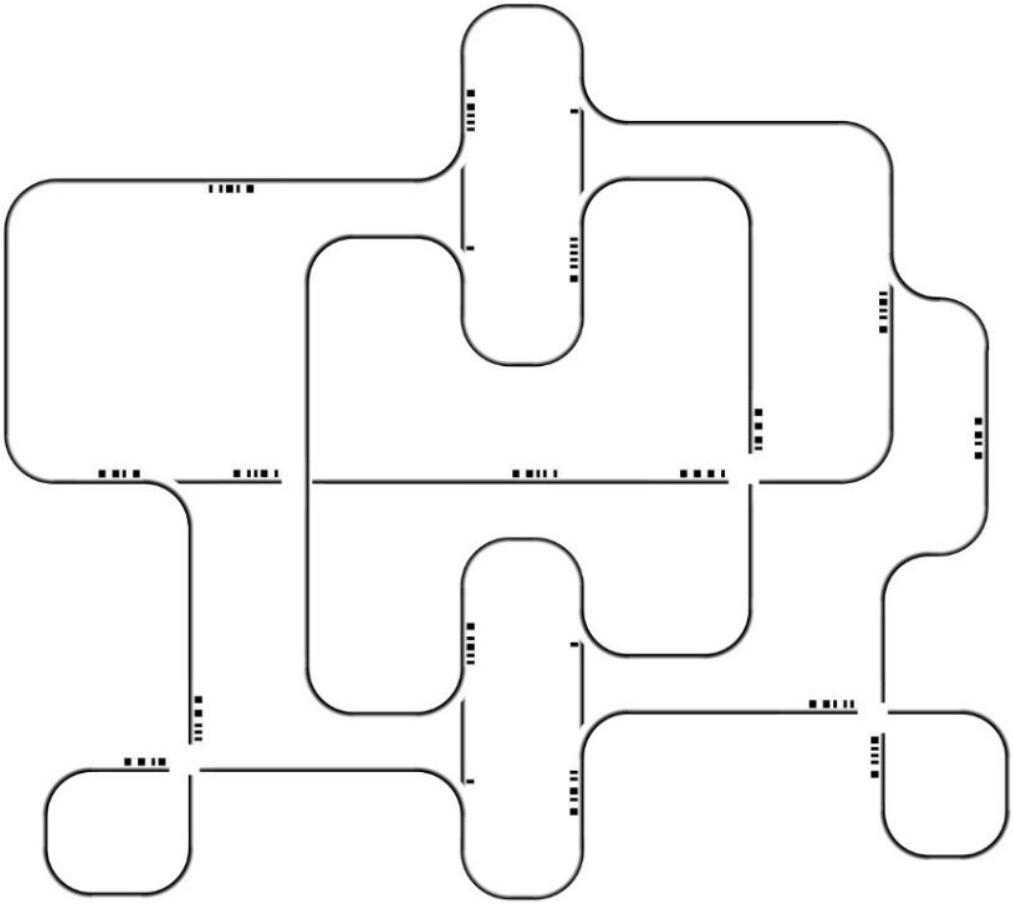

Figura 2.1: Primer circuito realizado.

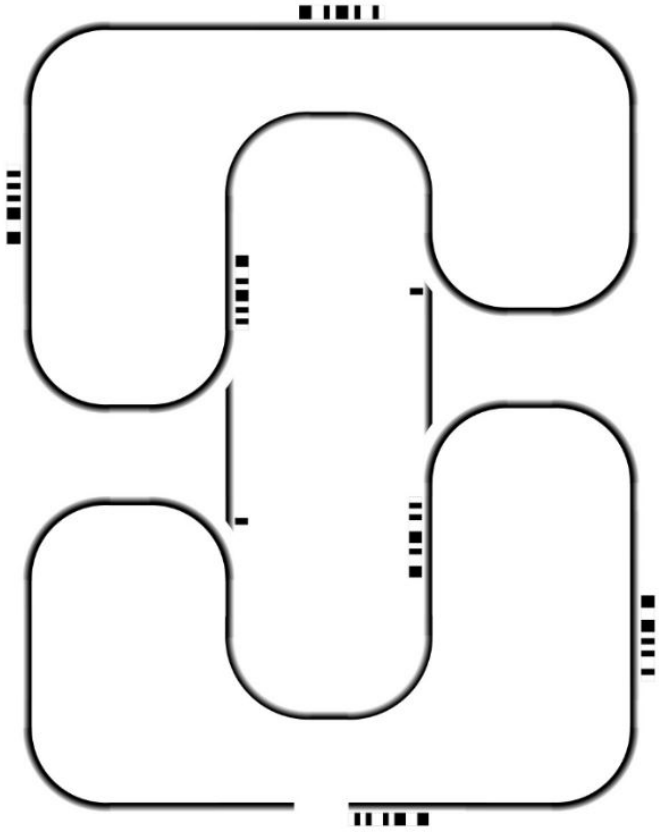

Figura 2.2: Circuito 1.

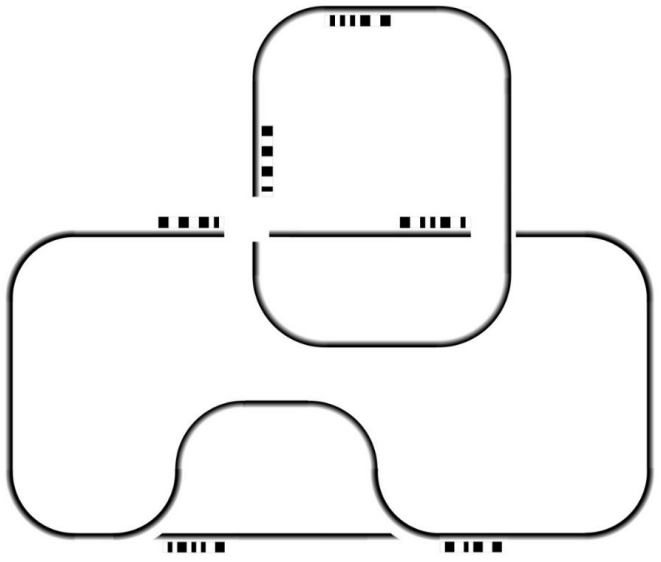

Figura 2.3: Circuito2.

- Dependiendo del nivel de intensidad que detecte el sensor de línea el coche va a estar en uno u otro lado de la carretera, por lo tanto el Gigglebot va a saber su colocación en el circuito y puede actuar en consecuencia.

Sin embargo, este modelo también tiene desventajas entre las que destacan la dificultad para poder diseñarla y los aspectos externos que pueden producir errores en el momento de la lectura.

Dado que este proyecto se basaba principalmente en la programación de los robots y de las interacciones que existen entre ellos, se decide utilizar un programa con el cual se pueden conseguir los distintos elementos del circuito (rectas, las curvas, intersecciones y los códigos) de una manera sencilla. (Véase la figura 2.4). Sin embargo, debido a la sencillez y a la no especialización de este programa para realizar los diseños, los resultados obtenidos no son óptimos y posteriormente seguimos teniendo problemas que se traducen en la lectura errónea de las carreteras.

Otro problema que surge en el diseño de los elementos ocurre en el desarrollo de las curvas, ya que no se pueden realizar curvas demasiado cerradas, porque un cambio exagerado de la intensidad provocaba que el coche se saliera de la carretera y se perdiera.

Atendiendo a los aspectos externos con los que se han producido problemas, destacan dos: la luz y el brillo. Aunque estos aspectos no se pueden controlar, se han propuesto soluciones para disminuir los efectos que provocan.

- Luz: Dependiendo de la cantidad de la luz y en zonas de excesiva diferencia de luz, es posible que el robot pueda leer incorrectamente. Para solucionar este tipo de problema, se intenta hacer que la luz empleada durante las pruebas sea homogénea durante todo el circuito.

- Brillo: La calidad del papel y de la tinta de la impresora utilizada es un aspecto fundamental. Añadiendo estos factores a la incidencia de la luz, es posible que provoque reflejos en la carretera y produzca errores en la lectura. Utilizando fotocopias con una calidad media se obtuvieron los mejores resultados que, aunque se lograba una calidad menor en cuanto a niveles en la escala de grises, se obtenía una calidad mate con las que se evitaban los brillos.

En la figura 2.4 se puede observar el diseño final de las carreteras básicas. Es necesario comentar que el ancho de la carretera es de 3 cm, así el coche puede llegar a los dos extremos

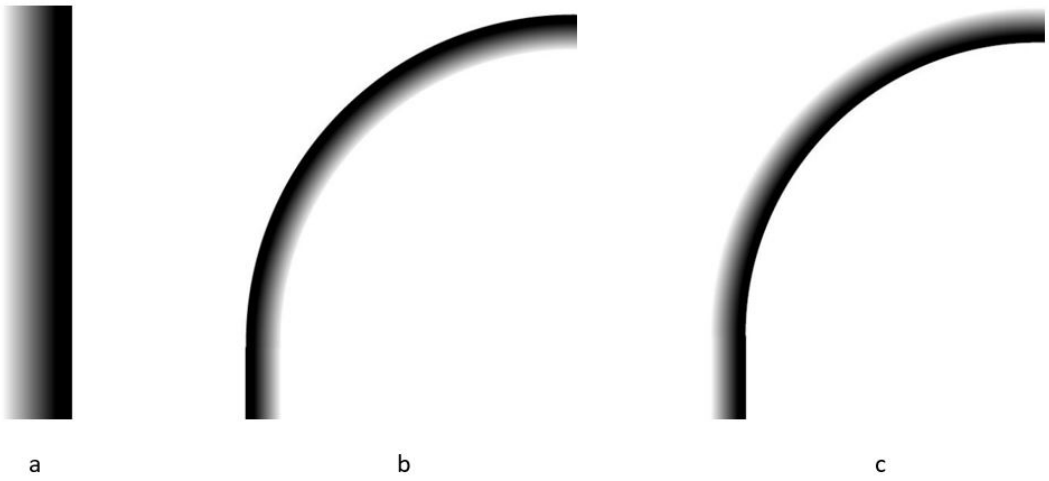

Figura 2.4: Elementos Básicos: a: Recta b: Curva a la izquierda c: Curva a la derecha.

de la carretera sin que el sensor de la izquierda pueda obtener una lectura de la carretera y afectar al comportamiento del Gigglebot. El degradado que se realizó fue teniendo la parte más oscura hacia la derecha y reduciendo el nivel de intensidad hacia la izquierda. También se puede observar que se opta por realizar curvas abiertas, que aunque ocupan más en cuanto a dimensión, se adquieren unos mejores resultados y se solventa el problema mencionado anteriormente.

2.3. Velocidad

El movimiento del Gigglebot se consigue a través de la aplicación de fuerza por parte de los motores a las ruedas. Para referirse a la velocidad del Gigglebot se utiliza el porcentaje de uso del motor. Es decir, cuando se dice que se está utilizando una velocidad de 30, en realidad se está diciendo que los motores están utilizando una potencia del 30 %.

Estas velocidades se utilizan durante el estado ROAD (apartado 2.7), ya que en los momentos críticos (el proceso de lectura y en acciones) se recurre siempre la velocidad mínima.

Sin embargo, durante la realización de las pruebas, se observa que el coche se mueve de una manera inestable (bambolea). Este movimiento genera que la probabilidad de que el coche se salga de la carretera aumente. Se concluyó que para obtener mejores resultados y evitar que el coche se saliera con otras velocidades, es necesario que tanto los diseños de las carreteras como el proceso de calibración (como se explicará más adelante en el apartado 2.6) deberían ser mejores. En consecuencia, para poder ver el impacto que genera el aumento de velocidad

en los robots se mantuvo un único cambio de velocidad, pero en vez de aplicar cambios tan bruscos como se pensó inicialmente (30-40-50) se utiliza uno más ligero (30-33).

Aunque parezca un cambio mínimo en cuanto al número, atendiendo a las pruebas, se puede observar que el robot oscila mucho más durante el estado ROAD y esto provoca que en los momentos críticos los errores se incrementen significativamente. Por lo tanto, en esos momentos, también se decide que el robot cambie a la velocidad mínima.

Estas velocidades se ajustarán al principio, antes de colocar al robot en el circuito, para cada Gigglebot dependiendo de los factores que se comentarán en el apartado de calibración.

2.4. Saltos

En ciertos momentos del circuito es necesario que el Gigglebot deje de obtener datos de los sensores, ya que se producirán cortes en las carreteras, por ejemplo en bifurcaciones y semáforos. En estos momentos, es necesario que el robot realice saltos. El coche recorrerá durante un tiempo determinado una distancia en línea recta, sin tener que tomar datos de los sensores.

En un principio, se decide unificar la distancia de salto para todos los elementos dejando siempre una distancia de salto de 13 cm para utilizar con todos la misma medida de tiempo para el salto. Sin embargo, tras la realización de pruebas y la observación del comportamiento del coche en los diferentes elementos se llega a la conclusión de que dependiendo del elemento, el coche deberá realizar un salto durante un tiempo diferente. En nuestro caso tenemos 3 tiempos diferentes (entrada_carretera, salida_carretera y semáforo).

A continuación se muestra el código utilizado para obligar al coche a que pare de leer de los sensores y únicamente se dirija de forma recta durante un tiempo determinado:

```
Function contTime(time: number, speedLeft: number, speedRight: number) {
   let init = input.runningTime()
   let speedLeftFinal = fixGoStraight(speedLeft, MOTOR_LEFT_FIX)
   let speedRightFinal = fixGoStraight(speedRight, MOTOR_RIGHT_FIX)
   let contTime = init

   while (contTime<init + time) {
      gigglebot.motorPowerAssignBoth(speedLeftFinal, speedRightFinal)
      contTime = input.runningTime()
```

```
    }
}
```

2.5. Diagrama de estados

Un diagrama de estados es una representación fiel de la máquina de estados. En la figura 2.5 se puede ver el diagrama de estados en el que se representa el funcionamiento completo de los Gigglebot que actuarán como los coches. Los diferentes estados en los que se divide el programa están marcados en azul y las líneas representan las transiciones y la circunstancia necesarias para que se produzca el cambio de estado.

Se han introducido dos elementos, representados por dos cuadros naranjas para hacer que el dibujo sea más claro. Esos recuadros representan una función que se realiza en los estados desde donde se inicia la flecha. Dependiendo del valor que se obtenga, se cambiará a un estado o a otro. En la figura 2.5 se aprecia que tenemos un total de 10 estados:

-INIT: Este es el estado inicial. En este estado no se realiza ninguna acción, el Gigglebot espera a que se pulse el botón A. Nunca más se volverá a este estado.

-CALIBRATION: En este estado el coche ajusta las variables para el sensor de línea y se inicia el proceso para la calibración de la velocidad y la dirección. Tras estos procesos, el Gigglebot espera a que se pulse el botón B para cambiar de estado.

-ROAD: Este estado ocurre cuando el coche está recorriendo el circuito, siguiendo la línea (el circuito) con el sensor de la derecha.

-BIT_SINCRO: Este estado ocurre cuando el coche detecta el inicio de la lectura del código. El proceso de lectura se explica durante el apartado 2.8.2.

-READING: El robot está leyendo el código

-PROCESS_MEMO: El robot ha terminado de leer y dependiendo del código obtenido decide a que estado de acción debe cambiar. Los estados de acción son los siguientes: RADIO, ROAD_ENTRANCE y ROAD_EXIT.

-ROAD_ENTRANCE: Ha detectado una entrada de carretera

-ROAD_EXIT: Ha detectado una salida de carretera

-RADIO: Debe empezar a escuchar en un canal de radio para iniciar las acciones propias en una comunicación con un semáforo.

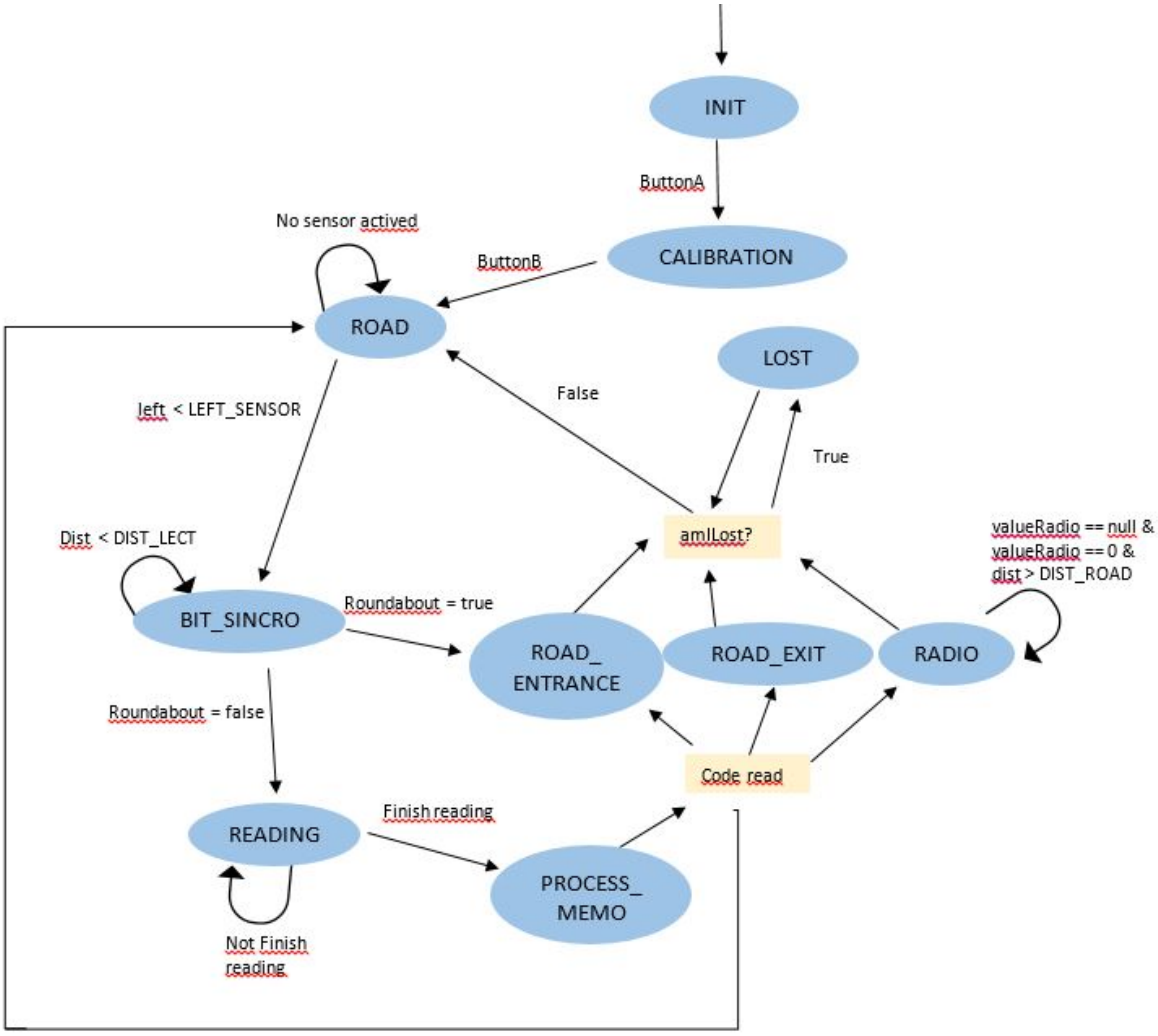

Figura 2.5: Diagrama de estados del Gigglebot que actúan como los coches.

En cada uno de los estados de acción, al terminar todas las acciones propias de cada estado, deben realizar una función para comprobar si han podido detectar la carretera (amILost())

-LOST: Tras ejecutar la función amILost(), el coche no consigue detectar la carretera.

2.6. Calibración de los Gigglebot

Para el correcto funcionamiento del Gigglebot, debido a los aspectos externos que pueden afectar al funcionamiento del robot es necesario realizar una calibración antes de colocarlo en el circuito. Este proceso se realiza durante el estado de CALIBRATION. Debido al influyente aspecto del nivel de la batería y a que los Gigglebot no tienen un movimiento unificado, es

necesario realizar este proceso siempre antes de utilizar el robot, es decir, no se pueden guardar datos obtenidos en anteriores calibraciones, ni datos que se han obtenido de la calibración de otros Gigglebot.

En este caso, se tienen que calibrar tres aspectos: Sensor de línea, la velocidad y la dirección del robot.

2.6.1. Calibración de la dirección

Consiste en hacer que el robot vaya lo más recto posible aplicando una fuerza mayor o menor en los diferentes motores. En esta calibración se utilizan funciones predefinidas que facilitan la programación y solucionan muchos aspectos de este problema. Se mide el grado de giro haciendo que el coche recorra una distancia de medio metro. Si se observa que se tuerce hay que cambiar los datos de las velocidades para cada motor y volver a probar.

No obstante, no es un proceso crítico. El único inconveniente es que cuanto peores sean los datos (si se tuerce demasiado) la probabilidad de errores en los saltos aumentará.

2.6.2. Calibración de la velocidad

La velocidad del Gigglebot es afectada por el nivel de energía de las baterías. Por lo tanto es necesario hacer un ajuste en la velocidad para que siempre mantenga la misma y que no dependa de la batería. Para ello, el coche recorrerá una distancia de medio metro para calcular el tiempo que tarda, (se utiliza el mismo circuito en la calibración de dirección) y realizaremos cálculos básicos para obtener los datos necesarios.

En primer lugar, utilizando una velocidad de 50, se calcula una medida ($\mathrm{t_{ref}}$) que tomaremos como nuestra medida ideal. Este valor representa el tiempo mínimo en el que un Gigglebot puede recorrer una distancia de medio metro. Este valor se toma asumiendo condiciones perfectas: Baterías totalmente cargadas y calibración de dirección perfecta.

En la tabla 2.6.2 se pueden observar las diferentes pruebas que se realizaron. Los datos que se obtuvieron reflejan que los coches no tienen un movimiento homogéneo y por eso necesitan diferentes calibraciones de dirección y se logran diferentes tiempos en recorrer la misma distancia. La última columna representa un promedio de los valores de las medidas separando por coches.

	Motor Izq	Motor Der	Med 1	Med 2	Med 3	Med 4	Promedio
Coche 1	50	52	2452	2399	2353	2386	2397
Coche 2	51	50	2350	2324	2290	2301	2316
Coche 3	50	50	2455	2485	2470	2429	2459
Valor Final							2200

Cuadro 2.1: Medidas tomadas para obtener la medida de tiempo de referencia. Se usan diferentes coches y se indican los valores obtenidos en 4 muestras en milisegundos y con los valores empleados de los motores.

El valor final que se utiliza es un valor que se encuentre por debajo de todas las medidas y dejamos un cierto rango de guarda por si un coche obtuviera un valor menor. En este caso se utiliza 2200 ms.

Una vez calculado este valor, se realizan cálculos para lograr las equivalencias a otras velocidades. Primero se calcula la velocidad equivalente ($\mathrm{V_X}$): Este dato representa la velocidad eliminando la influencia de las baterías, es decir, la velocidad del coche necesaria para recorrer el circuito de calibración en el tiempo de referencia. Para calcularlo, se utiliza el tiempo que se obtiene al hacer la calibración y el tiempo de referencia.

$$V_X = tiempoMed * 50/t_{ref} \tag{2.1}$$

Con esta fórmula se obtiene la equivalencia, en términos de velocidad, entre la velocidad teórica (50) y la velocidad real ($\mathrm{V_X}$), teniendo en cuenta los efectos de la disminución de la energía de la batería.

Por último se calcula la velocidad real ($\mathrm{V_{Real}}$) de la velocidad teórica objetivo ($\mathrm{V_Y}$).

$$V_{Real} = V_Y * V_X/50 \tag{2.2}$$

Por ejemplo, si la velocidad teórica objetivo es 30. La fórmula a aplicar será: ($\mathrm{V_{Real}} = 30 * \mathrm{V_X}/50$).

2.6.3. Calibración del sensor de línea

En este proyecto se utilizan variables estáticas. Se han obtenido experimentalmente los valores para que el coche circule lo más centrado posible por la carretera, procurando dejar lo

máximo posible de distancia a los dos lados para que el coche tenga posibilidad de reacción si se acerca al borde de la carretera.

Para realizar esta calibración se utilizan las medidas de la media y de la varianza. Se calculan de forma experimental los valores de máxima (208) y mínima (108) para colocar el coche en la carretera y a partir de estos valores se calculan la media y la varianza.

2.7. Estado ROAD

El Gigglebot se encuentra en este estado cuando está siguiendo la carretera. Se utiliza el sensor de la derecha para seguir el nivel de intensidad de negro (escala blanco-negro). Este sensor es lineal, es decir, detecta el nivel de intensidad y convierte esa escala a un número. El rango de valores va desde 0 hasta 1024.

Por otro lado, durante este estado, se va a controlar siempre la velocidad dependiendo de si existe un obstáculo en frente. El robot va a modificar la velocidad para que no choque con el de delante y frene de forma progresiva. Se realizan modificaciones en el código para que no frene de forma brusca, es decir, no va a pasar de una velocidad de 30 a 0 directamente, sino que va a ir reduciendo paulatinamente según se acerque a cualquier objeto detectado.

2.7.1. Fundamentos de la lectura de la carretera

El sensor de la derecha toma una medida que indica la posición actual del coche respecto a la carretera. Con esta medida, se halla el valor de la desviación ($\mathrm{medida} - \mathrm{media}$). El valor absoluto de la desviación se compara con la varianza. Si este valor es menor quiere decir que el coche se encuentra cerca de la media y actúa sobre los motores de una manera suave. Si está por encima, quiere decir que se encuentra lejos de la media, por lo tanto actúa de una manera más agresiva, proporcionando más potencia a alguno de los motores para que se recoloque mejor y vuelva a una zona cercana respecto a la media. Dependiendo de si el valor de la desviación es negativo o positivo, el coche reconoce hacia qué lado de media está, por lo que esto provocará que el coche gire hacia uno u otro lado.

2.7.2. Ejemplo

Para obtener una visión más clara del funcionamiento, se introduce un ejemplo de un caso extremo, en el que el coche está hacia la derecha de la carretera, en la zona más oscura.

En primer lugar, tendremos los siguientes valores que corresponden a la calibración del sensor de línea y se incluyen otros datos necesarios:

- Media = 158.
- Varianza = 40.

  Estos valores fueron obtenidos en el proceso de calibración del sensor de línea.

- Valor del sensor: Valor detectado muy bajo 20. (La intensidad es negro).
- Velocidad de 30.

Con estos valores obtenemos la desviación con respecto a la media ($20 - 158 = -138$). Finalmente se realiza el valor absoluto y se compara con la varianza. En este caso se puede observar que la desviación es mayor que la varianza. Tras esta comprobación se aplican las diferentes potencias a los motores:

- Motor derecho: $30 - (-138)/6 = 53$
- Motor Izquierdo: $30 + (-138)/6 = 7$

Como conclusión a este ejemplo: Los resultados obtenidos son lógicos ya que el coche se encuentra en el extremo derecho del circuito, muy alejado de la media. Por lo tanto hay que introducir mucha potencia al motor de la derecha para que gire hacia la izquierda y así vuelva a valores más cercanos a la media.

2.8. Proceso de lectura

El proceso de lectura consiste en indicar al Gigglebot el elemento de la carretera que se va a encontrar a continuación para que pueda adaptarse y actuar de una determinada manera. Para reducir la probabilidad de error en este proceso se han tomado dos medidas:

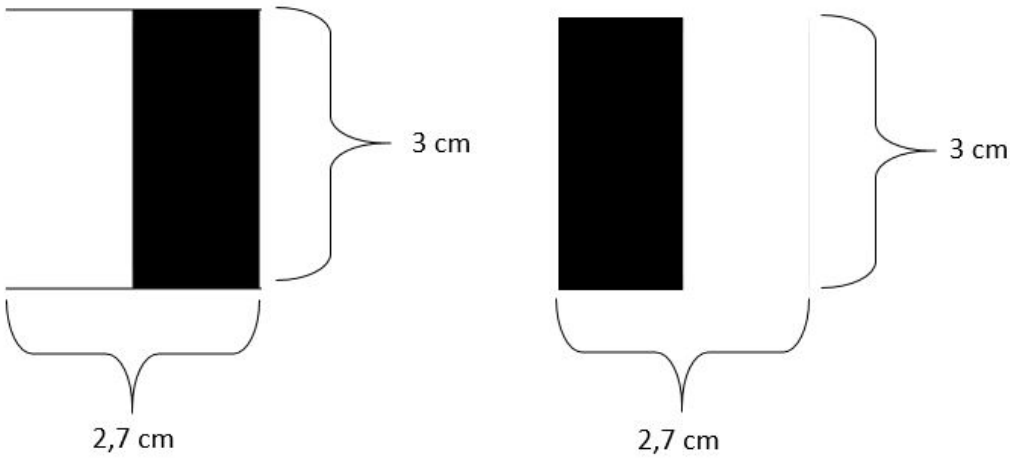

Figura 2.6: Los tipos de transiciones que existen en la lectura. La primera representa el bit 1 y la segunda el 0.

- Debido a los problemas mencionados anteriormente con las velocidades, el robot va a realizar este proceso utilizando la velocidad mínima.

- Es necesario dejar un tramo recto antes del código para lograr que el coche se adapte a la carretera y oscile lo menos posible.

2.8.1. Características de los códigos

La unidad mínima por la que está formado el código es el bit y cada bit representará un cambio de intensidad. En nuestro caso el bit 1 es la transición blanco-negro y el 0 es la transición negro-blanco. En la figura 2.6 se pueden observar los dos tipos de transiciones con sus respectivas medidas.

Un código completo de lectura se realiza con 6 bits y medio (2,5 bits de sincronización y 4 de información). Además, si es de tipo B es necesario incluir un medio bit adicional. En la figura 2.7 se observa una representación de un código de tipo A y en la figura 2.8 de tipo B.

El medio bit 'Fin' (perteneciente al código de tipo B), sirve para finalizar la lectura de los códigos que acaben en un 1. Es necesario hacer que siga leyendo hasta que vuelva a una intensidad blanca. En caso contrario, después de procesar el código, el coche volvería a leer una intensidad de color negro y volvería a intentar leer el código y esto provocaría el mal funcionamiento del coche.

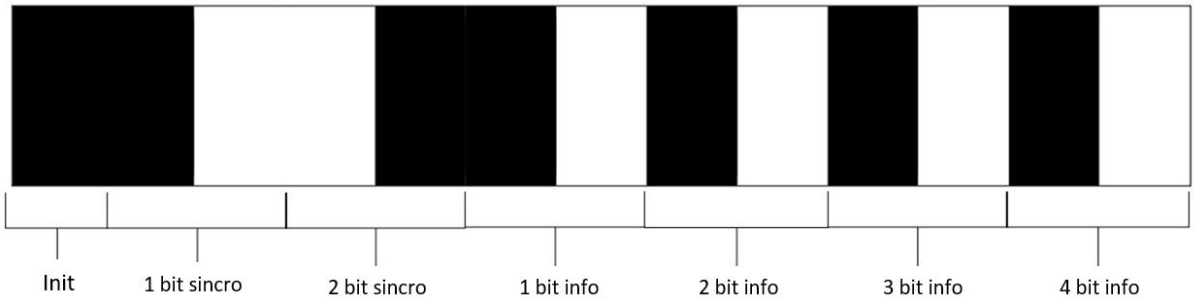

Figura 2.7: Código de tipo A.

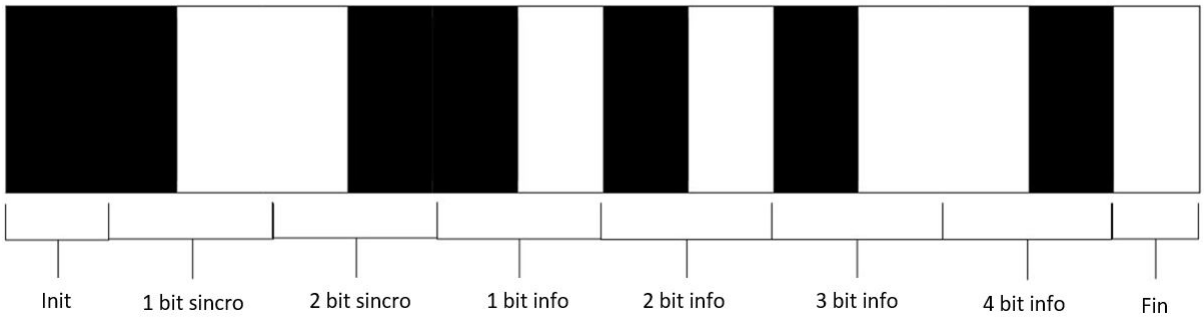

Figura 2.8: Código de tipo B.

El número total de códigos se calculan teniendo en cuenta el número de bits de información (en nuestro caso 4). Teniendo en cuenta esto, se pueden obtener un total de 16 códigos(2^{NBitInfo}).

2.8.2. Funcionamiento del proceso de lectura

El proceso de lectura se divide en tres estados: BIT_SINCRO, READING, PROCESS_MEMO.

En el estado BIT_SINCRO: el coche ha detectado el primer medio bit, al que hemos denominado como Init. Este recuadro puede indicar dos cosas:

1. El Gigglebot se encuentra dentro de una rotonda y hay una entrada de carretera. Esta opción se explicará en el apartado 2.9.

2. Hay un código de lectura.

Este apartado se centra en la segunda opción (hay un código de lectura).

Durante este estado, el coche también adapta la velocidad a la mínima, ya que estamos en un proceso crítico, además obtendrá valores del sensor de distancia para detectar la ubicación de algún obstáculo. El coche esperará a que no haya ninguno a una distancia menor al doble

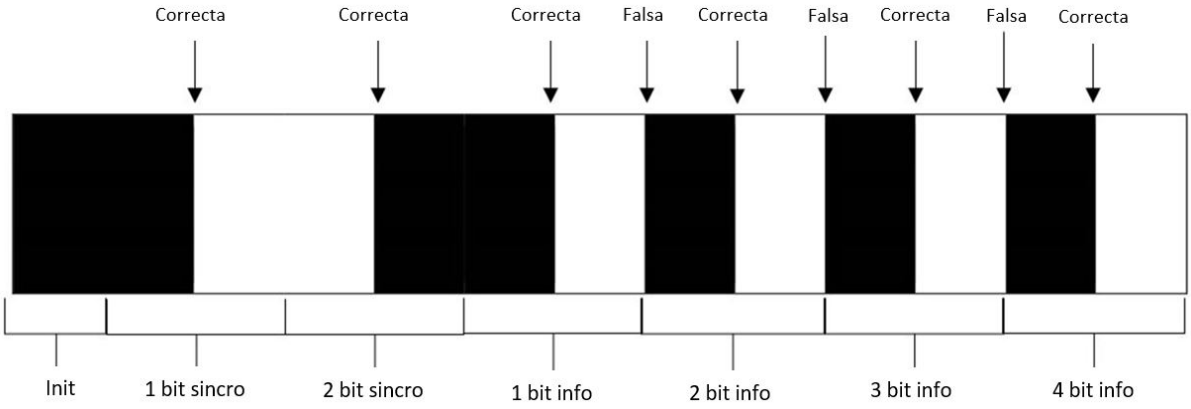

Figura 2.9: Código con trasnsiciones verdaderas y falsas.

de la longitud del código. Así nos aseguramos que el coche va a leer el código de lectura sin problemas y sin obstáculos. Por lo tanto, si existiera algún obstáculo, el coche permanecerá en su sitio sin moverse. Sin embargo, si no detecta ningún obstáculo, el coche cambiará al estado READING.

Durante el estado READING, el coche va a obtener los bits de información del código. Las funciones de este estado se realizan a través de la detección de tiempos.

En primer lugar los dos bits de sincronización se utilizan como código de entrenamiento, gracias al cual se saca el tiempo que hay entre detección de dos bits. Con el tiempo obtenido (diffGlobTime), se comparan el resto de las transiciones para comprobar si es correcta o falsa.

Hay que tener en cuenta los diferentes tipos de transiciones que nos encontramos en el código:

- La Transición verdadera se produce en el centro del bit. Es la que tenemos que buscar, la que informa sobre el bit de información.
- La Transición falsa se produce en los bordes de los bits, cuando finaliza el primero y se inicia el siguiente.

En la figura 2.9 se puede observar un código dónde se muestra cuáles son las transiciones verdaderas y las falsas.

Con el tiempo anterior se entiende que todos los cambios de intensidad verdaderos se deben dar en tiempos similares a diffGlobTime y los falsos se pueden dar en $\mathrm{diffGlobTime}/2$. Sin embargo de forma experimental se comprueba que no es así debido a aspectos externos del

robot, por lo tanto se decide colocar un rango de $3 * \mathrm{diffGlobTime}/4$ por debajo del cual no guardará el valor de la transición.

De esta manera obtenemos los cambios de transición verdaderos, y por consiguiente guardamos los bits correspondientes en la memoria. No obstante este proceso no es infalible, por lo tanto en determinados momentos puede llegar a fallar.

Tras la lectura de los 5 bits de información termina de leer el código, si es de tipo B, realiza el último proceso mencionado anteriormente y el coche cambia de estado a PROCESS_MEMO que será el encargado de comparar el código leído con sus códigos guardados y de realizar las determinadas acciones dependiendo del código.

Finalmente se realiza un borrado de las variables utilizadas durante el proceso de lectura para cuando se vuelva a detectar otro bit de sincronización, que se vuelva a realizar el proceso desde el principio.

2.9. Bifurcaciones

Para realizar una red de carreteras fiel a una red real es necesario la inclusión de elementos en los que los coches puedan interaccionar entre ellos. Además, deben existir diferentes direcciones hacia las cuales los coches puedan dirigirse. Teniendo en cuenta esto, es necesario que haya coches que se puedan incorporar a una carretera y que hayan salidas de las carreteras en las que se encuentran. Por ello se han desarrollado varios tipos de bifurcaciones.

2.9.1. Entrada de una carretera

Este elemento se compone de dos carreteras. Una de ellas se incorpora en la primera. Tras leer el código de barras, el coche cambia de estado a ENTRANCE_ROAD. En este estado el coche realizará un salto y empezará a leer de nuevo para localizar la carretera. Un problema que existe con esta bifurcación es que debido a la ubicación de los sensores de distancia en los Gigglebot, es posible que los dos coches que circulen puedan chocar. Para solucionar este problema sería necesario colocar el sensor en algún otro lado o colocar un segundo sensor. En la figura 2.10 se observa el diseño de este elemento.

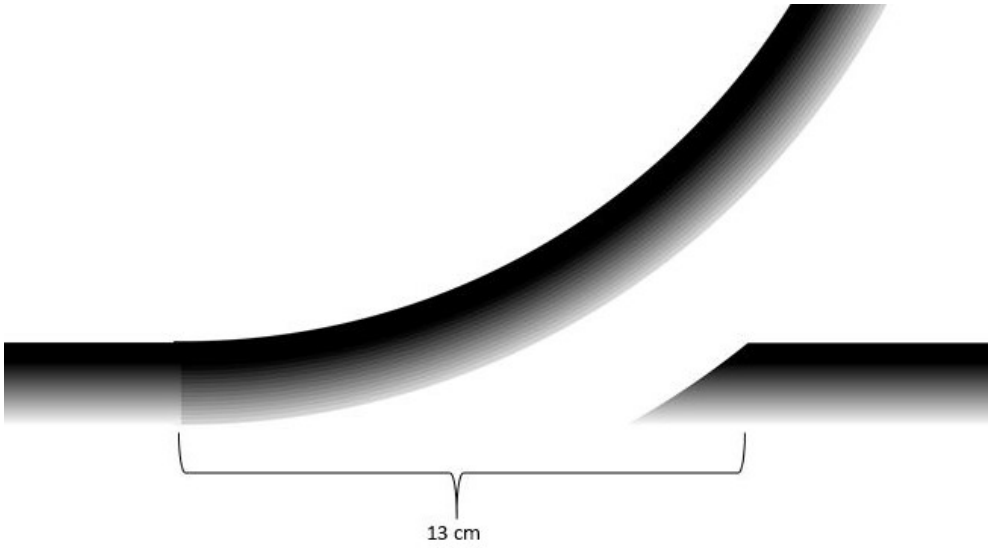

Figura 2.10: Diseño de la Entrada de la carretera.

2.9.2. Salida de una carretera

Este elemento se compone de dos carreteras en la que el coche podrá seguir diferentes caminos. En la figura 2.11 se puede observar el diseño de este elemento. Tras leer el código de barras, el coche cambia al estado EXIT_ROAD. En este estado el coche decide de manera aleatoria si realiza el salto para continuar recto o si continua leyendo de los sensores y continua circulando hacia la derecha. Esta decisión se consigue utilizando una función predefinida (Math.randomBoolean()). Con esta función se pueden obtener dos valores (Verdadero o Falso). Cada valor tiene un 50 % de probabilidades de salir. En este caso, se decidió que si el valor obtenido es verdadero, el coche realizaría el salto. Por consiguiente si sale falso, el coche seguiría leyendo de los sensores y continuaría circulando por la carretera por la que se encuentra.

2.9.3. Rotonda

La rotonda (Véase figura 2.12) es una mezcla de los dos elementos anteriores. No obstante, se realiza un cambio para evitar la utilización de dos códigos: Una vez leído el código y en el caso de que el coche realice el salto, guardará una variable para diferenciar si el elemento en el que se encuentra es una bifurcación simple (salida de carretera) o si es una rotonda. Véase la tabla 2.9.3.

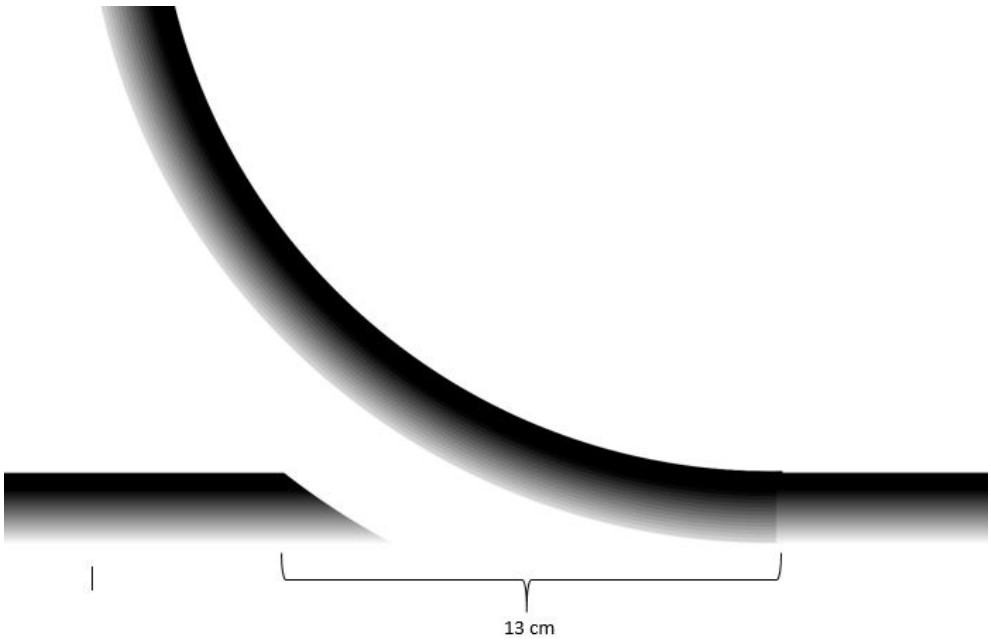

Figura 2.11: Diseño de la Salida de la carretera.

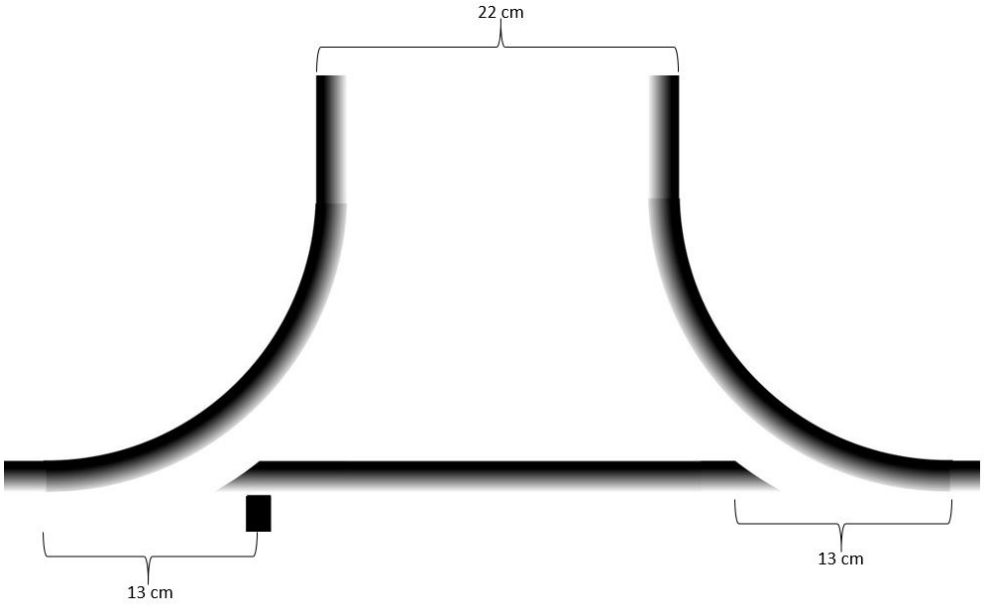

Figura 2.12: Diseño de las bifurcaciones de una rotonda.

	Bifurcacion Simple	Rotonda
Paso 1	El coche lee el código y detecta una salida de carretera	El coche lee el código y detecta una rotonda
Paso 2:	El coche decide realizar el salto	El coche decide realizar el salto
Paso 3:	En la variable ‘roundabout‘ se guarda un 0	En la variable ‘roundabout‘ se guarda un 1
Paso 4:	Se mantiene en el estado ROAD y sigue circulando por el circuito	Se mantiene en el estado ROAD y sigue circulado por el circuito
Paso 5:	Detecta un bit de sincronización y cambia al estado BIT_SINCRO	Detecta un bit de sincronización y cambia al estado BIT_SINCRO
Paso 6:	El coche cambia al estado READING y realiza el proceso de lectura explicado en el apartado 2.8	El coche cambia al estado ENTRANCE_ROAD y realiza el salto. El coche cambia la variable ‘roundabout‘ a 0 para indicar que la rotonda ha acabado.

Cuadro 2.2: Diferencias en el funcionamiento de los coches si se ha detectado una bifurcación simple o una rotonda.

2.10. Semáforos

Uno de los elementos más importantes para el control de intersecciones y la regulación en la preferencia de paso son los semáforos. Por lo tanto, se consideró indispensable la implementación de este elemento en la red básica de carreteras.

Para implementar los semáforos se utiliza un Gigglebot. Las funciones del Gigglebot son la alimentación de la tarjeta Microbit y la visualización de las luces para representar los diferentes estados en los que se encuentre el semáforo (verde, ámbar, rojo). La selección de las luces no influye en el funcionamiento de la red, sin embargo, aporta un componente estético que le otorga veracidad al proyecto. El aspecto funcional lo realiza la tarjeta por completo, ya que será la encargada de ejecutar el código, es decir, será la encargada de indicar cuál de los coches tendrá la preferencia de paso.

2.10.1. Módulo de radio

Se utiliza el módulo de radio para establecer una comunicación simple de forma inalámbrica. En el entorno de programación Makecode, se utilizan las bibliotecas predefinidas en las que vienen ya implementadas las funciones necesarias para la comunicación vía radio entre los Microbits. Entre las funciones destacadas se utilizarán las de envío y recepción números y la de asignación de canales. Sin embargo, hay otras funciones útiles con las cuales se puede elevar la complejidad de las comunicaciones entre las tarjetas.

Para cada par de semáforo-coche, se utiliza un canal que es asignado a través de un número. Gracias al canal, se consigue que no interfieran las diferentes comunicaciones entre sí. Así, cada coche reaccionará a la señal que le envíe el semáforo correspondiente.

Un semáforo podrá enviar secuencialmente a través de varios canales dependiendo del número de elementos que controle. Normalmente, estos elementos se refieren a los Gigglebot que circulan por las carreteras. Sin embargo, existe un caso especial, que se explicará posteriormente: la comunicación entre semáforos maestro-esclavo.

2.10.2. Reacción de los coches a una comunicación

Los Gigglebot que actúen como coches siempre actuarán de la misma manera. La lectura del código obliga a que empiecen a escuchar en un canal. Una vez establecido el canal, los

Gigglebot comenzarán a escuchar: Si reciben un 1, avanzarán y realizarán el salto y si reciben 0 o no reciben nada se pararán y esperarán hasta que reciban un 1.

Una vez el coche haya escuchado por el canal un 1, cambiará de canal, es decir, empezará a escuchar en un canal controlado por el que sabemos que no se va a transmitir nada. En este caso, se estableció que fuera el canal 99. Por ejemplo: una vez el coche este realizando el salto (ha recibido un 1), y el semáforo cambie y envíe un 0, el coche seguirá su camino y terminará el cruce, ya que está escuchando en otro canal.

2.10.3. Funcionalidad común de los semáforos

El código de los semáforos ha sido implementado intentando que se parezca lo máximo posible a sus homólogos en la vida real. Para ello, como en el programa principal, también se utilizó una máquina de estados.

En este apartado se explican los aspectos comunes que tienen todos los semáforos diseñados, mientras que las peculiaridades de cada uno de ellos serán explicadas en el apartado 2.10.4.

Todos los semáforos tienen dos estados comunes:

- Green: El semáforo permite el paso del Gigglebot que controlan. En aspectos funcionales, el semáforo está enviando un 1 a través del canal asignado.

- Red: El semáforo no permite el paso del Gigglebot que controlan. En aspectos funcionales, el semáforo está enviando un 0 a través del canal asignado.

No obstante, existe un tercer estado que no se ha implementado en todos los semáforos: el Yellow. En este estado se envía también un cero, sin embargo tiene otra utilidad: Se utiliza para evitar posibles colisiones y dejar un tiempo de espera entre el cambio de estado Red a Green y viceversa. Con este tiempo se consigue que todos los saltos hayan finalizado durante este estado y no podrán producirse colisiones.

2.10.4. Tipos

Atendiendo a diferentes ejemplos en carreteras reales, se han conseguido varias formas de utilización de semáforos.

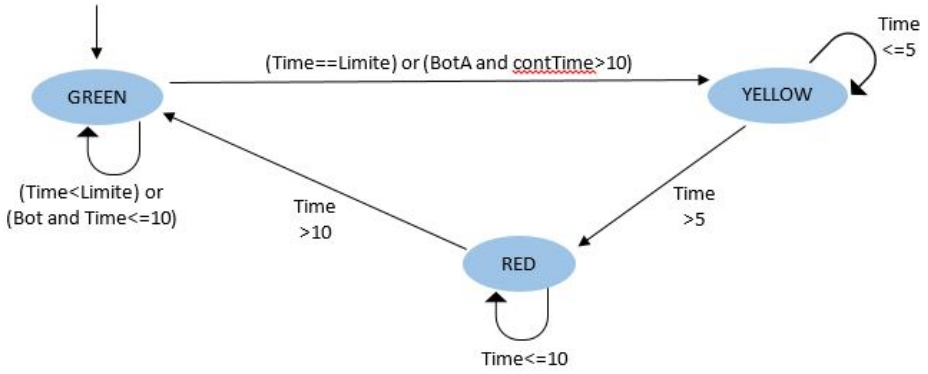

Figura 2.13: Diagrama de estados del semáforo para peatones.

Semáforo para peatones

Este semáforo simulará una intersección con un paso de peatones. El semáforo deberá ponerse en Red si un botón es pulsado por un peatón, o si pasa un tiempo determinado (que es elegido de forma aleatoria entre un rango que abarca entre 15 y 30 segundos) y nadie ha pulsado el botón.

En la figura 2.13 se representa el diagrama de estados, en el que se explican los estados de control sobre el coche, por lo tanto cuando hablemos de los colores se refieren a los que controlan el funcionamiento del coche.

El semáforo se inicia en estado *Green*, se calcula un valor aleatorio dado un rango (15-30 seg.) y permanece en este estado durante ese tiempo. Si no ocurre ninguna interacción pasado ese tiempo, el semáforo cambia a estado *Yellow* y posteriormente a *Red*.

En estado *Green*, en el caso de que ocurra una interacción con el semáforo (los peatones pulsan el botón), el semáforo cambia de estado a *Yellow* y posteriormente cambiará a *Red*. Sin embargo, no se puede mantener en estado *Red* para los coches continuamente porque se ha colocado un valor fijo (10 seg.) gracias al cual el estado *Green* se mantendrá como mínimo durante ese tiempo, es decir, si un peatón pulsa el botón entre el segundo 1 y el 10, el semáforo no cambia a *Yellow* instantáneamente, esperará hasta que hayan pasado los 10 segundos y se pondrá en *Red* para los coches.

En la figura 2.14 se muestra el diseño de la carretera para implantar la funcionalidad del semáforo; como se puede observar, es simple. Como se explicó en el apartado 2.4, hay un salto de la carretera que tiene una distancia de 13 cm de largo entre los diferentes tramos de la

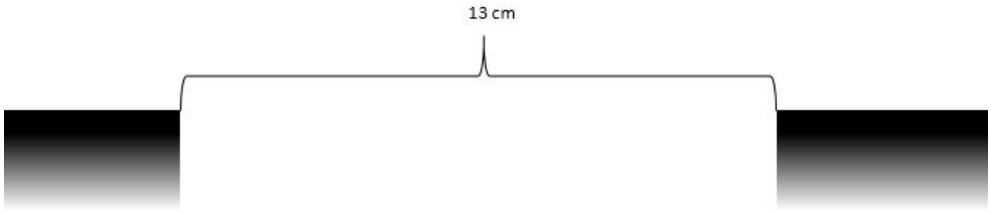

Figura 2.14: Diseño de la carretera para el semáforo de peatones.

carretera.

Semáforo de distancia

Para entender correctamente el funcionamiento de este tipo de semáforo, es necesario fijarse en la figura 2.15 ya que este elemento se divide entre carretera A y carretera B.

Este semáforo controla uno de los carriles (B) observando si hay algún Gigglebot circulando por el otro carril (A). Toma valores con el sensor de distancia verificando si hay algún Gigglebot en el carril A y dependiendo de este valor, permite el paso o no de los otros Gigglebot que se encuentran en el otro carril (B).

Fijándonos en el diseño podemos observar que mientras que la carretera A no sufre ninguna transformación, la carretera B se corta para indicar que se debe hacer un salto. La distancia que separa las dos partes de la carretera B es de 13 cm. Se deja 1 cm entre ambas carreteras para que el coche que circula por la carreta A, en caso de error de lectura, no se pueda salir de la carretera y continúe por la carretera B. Además, es necesario dejar un espacio suficiente a la izquierda para que el Gigglebot que pase por la carretera A y el que esté parado en la carretera B no puedan chocarse.

El semáforo se debe situar a 9 cm de la carretera B. Con esto se consigue que si detecta algo a una distancia menor de 600 mm aproximadamente, obliga a los coches que circulen por la carretera B a pararse. Esta distancia fue calculada de forma experimental.

En este semáforo, como se puede observar en la figura 2.16, únicamente habrá dos estados: Red y Green. El estado Yellow fue eliminado porque era innecesario.

Estado Green: Durante este estado, el semáforo está mandando un 1 al coche que se encuentra en la carretera B. Se mantiene en este estado siempre y cuando la distancia obtenida

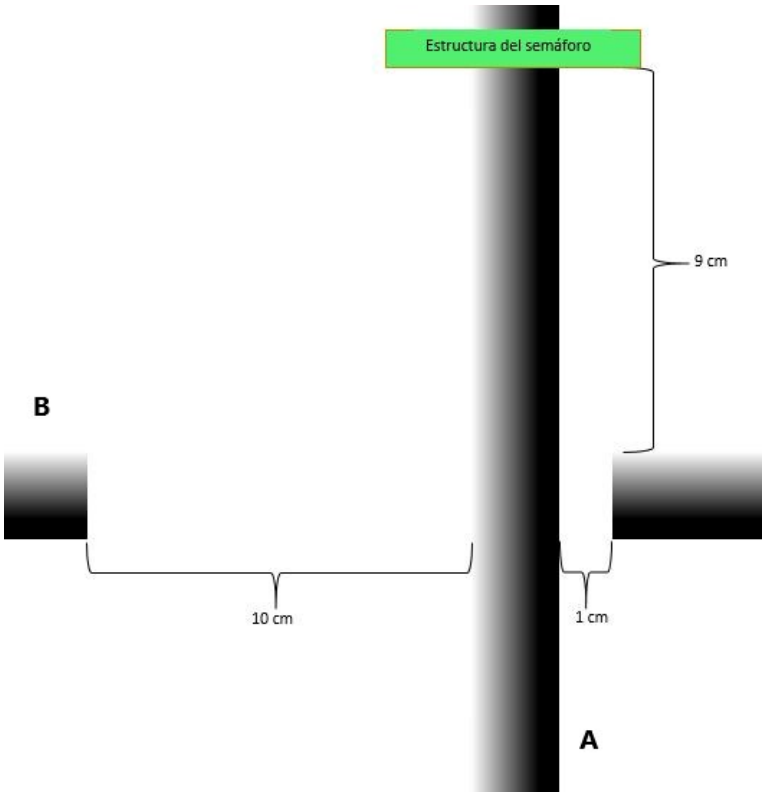

Figura 2.15: Diseño de la carretera para el semáforo de distancia.

Figura 2.16: Diagrama de estados para el semáforo de distancia.

en el sensor sea mayor que la distancia limite (600 mm). En el momento en el que el semáforo detecte algún obstáculo por debajo de la distancia límite, el semáforo cambiará a estado Red.

Estado Red: El semáforo envía un 0 al Gigglebot que se encuentra en la carretera B porque está detectando algún obstáculo por debajo de la distancia límite en la carretera A. Se mantiene en este estado si la distancia obtenida por el sensor es menor que la distancia límite. Cuando la distancia detectada sea mayor que la distancia limite, el estado volverá a cambiar a Green.

El coche que circula por la carretera A se mantiene independiente al semáforo en todo momento, éste solo se encarga de circular por la carretera porque no va a escuchar en ningún canal. Es totalmente ajeno a la comunicación vía radio.

Para utilizar este tipo de semáforo, es necesario diseñar una estructura ya que el sensor de distancia debe quedar por encima de la carretera porque es la posición óptima para poder detectar los Gigglebot. Para ello, se creó una estructura de cartón y se incluyeron piezas de lego para facilitar la sujeción del sensor a la estructura. Por otra parte, estas piezas facilitan el ángulo correcto para colocar el sensor mirando a la carretera. Véase la figura 2.17.

Con estas medidas se construyó el semáforo de distancia: alto 16,5 cm, largo 19,5 cm, ancho 7cm. El sensor de distancia forma un ángulo de 80º con respecto a la parte de arriba del semáforo. (Figura 2.18).

El semáforo se coloca, como se indicó anteriormente, a 9 cm con respecto a la carretera que tiene el código que controla (B). Con esto se consigue que el coche de la carretera A pase completamente el cruce antes de que se ponga en verde para los Gigglebot de la carretera B, por lo tanto no habrá ninguna colisión.

Semáforo doble

Este semáforo controlará dos carreteras al mismo tiempo. Envía un mensaje secuencialmente en 3 canales (Un canal para cada carril y otro para el semáforo esclavo).

En esta intersección hay 2 carriles (A y B) que se cruzan. La distancia de salto se mantiene en 13 cm. Como el sensor que lee la carretera está orientado hacia la derecha hay que dejar más distancia en el lado de la izquierda que en el de la derecha, para evitar que los coches puedan chocarse. En la parte de la derecha, como durante el salto el coche deja de leer, se puede juntar la carretera todo lo posible, porque el coche puede pisar la carretera y pasar el sensor por encima ya que no va a leer nada durante los 13 cm. Ver figura 2.19.

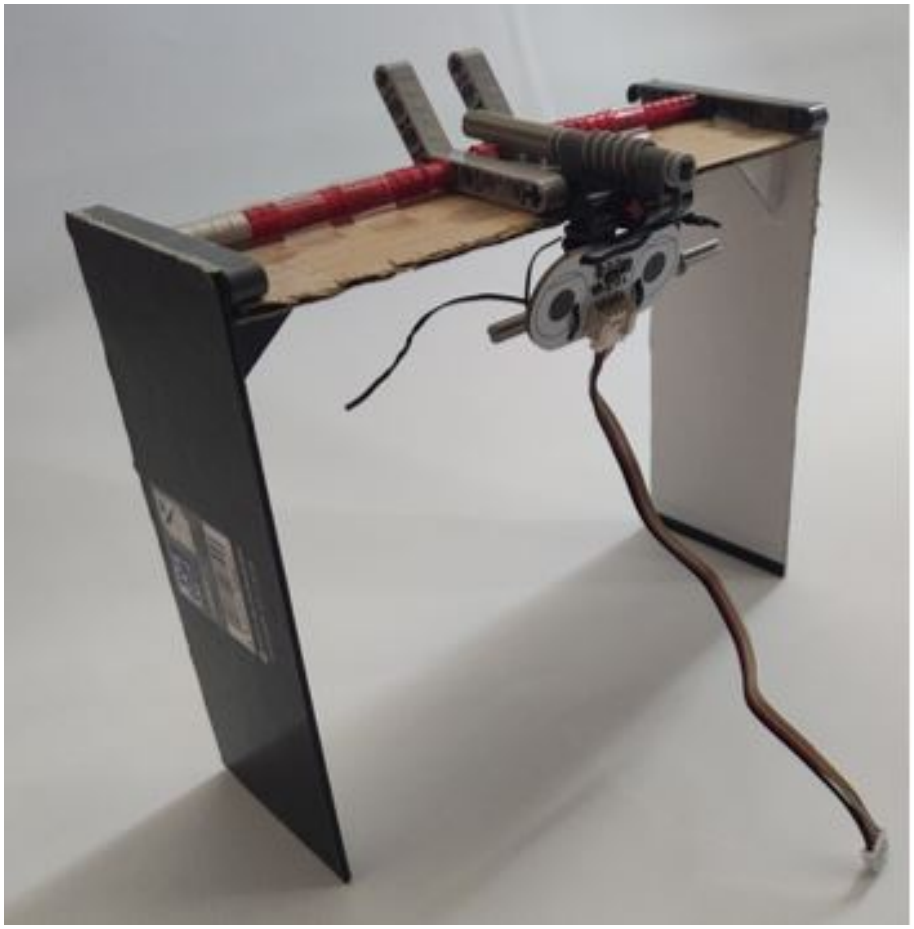

Figura 2.17: Imagen completa de la estructura del semáforo.

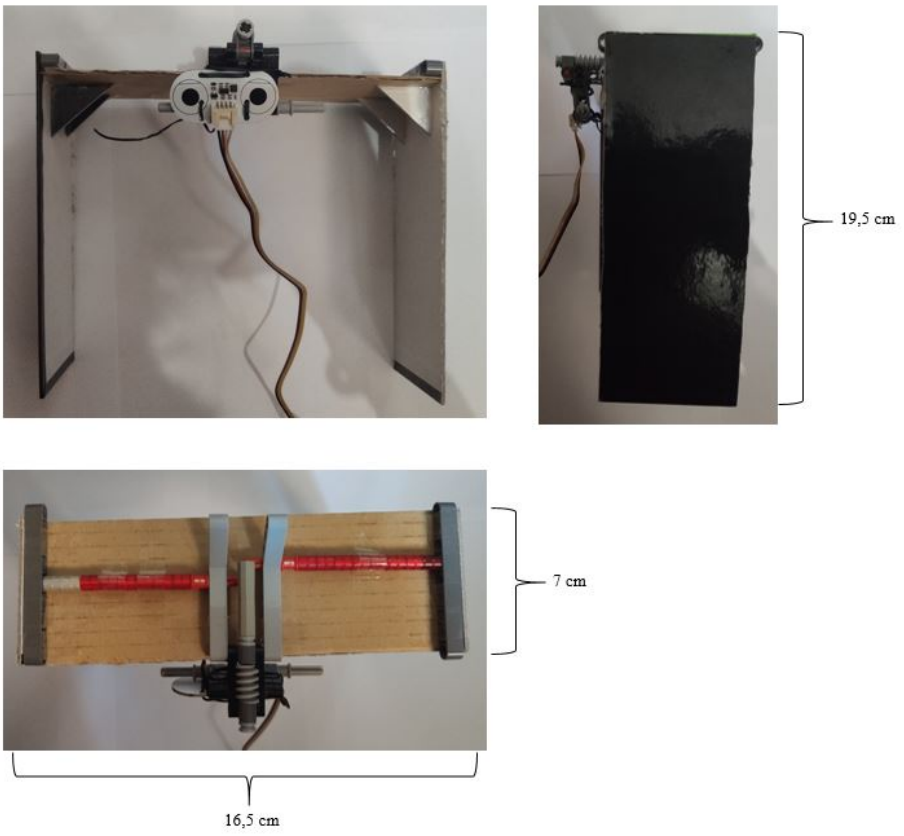

Figura 2.18: Estructura del semáforo con las medidas.

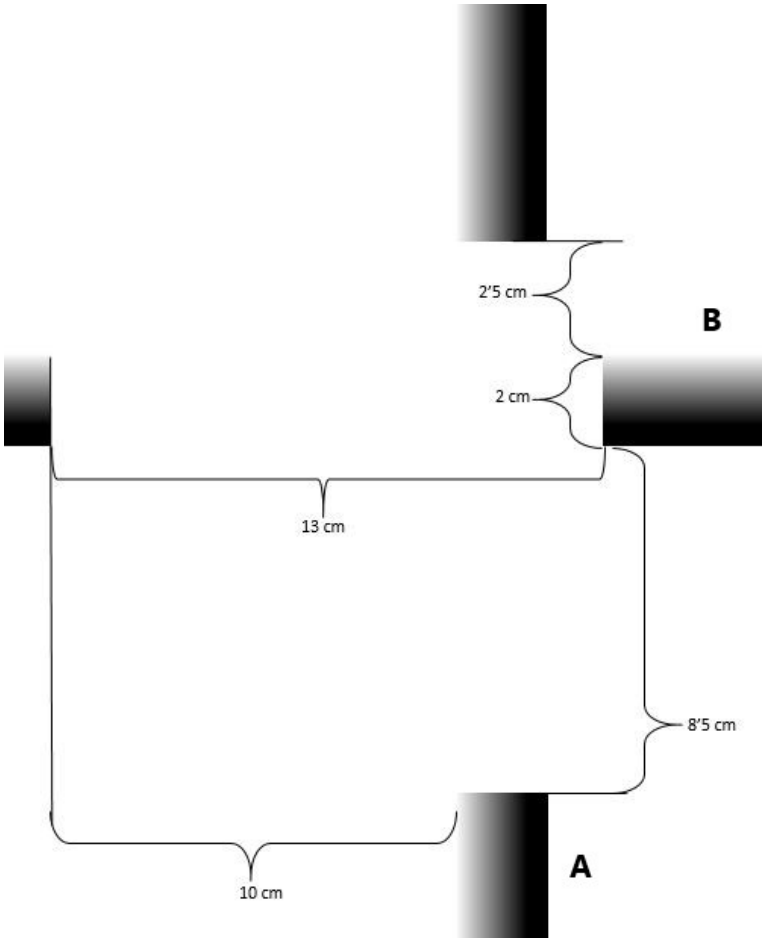

Figura 2.19: Diseño de las carreteras para el semáforo doble.

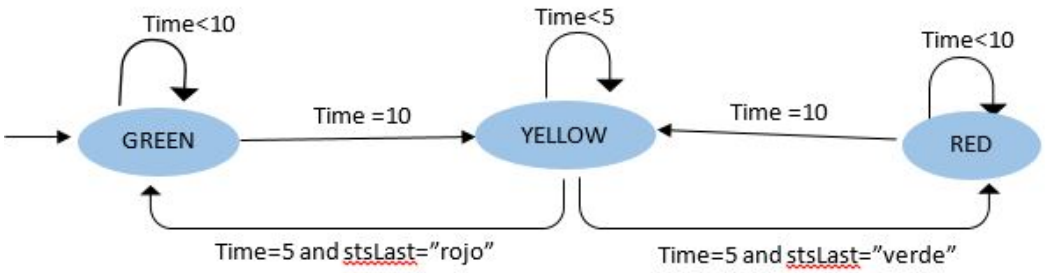

Figura 2.20: Diagrama de estados para el semáforo doble.

Estado Master	Canal 1 (x1)	Canal 2 (x2)	Canal 3 (x2-x1)	Estado Esclavo
Green	1	0	-1	Red
Yellow	0	0	0	Yellow
Red	0	1	1	Green

Cuadro 2.3: Relación que existe entre los semáforos Maestro y esclavo.

Si se observa la figura 2.20. El estado inicial del semáforo es el Green. En este estado el semáforo envía por uno de los canales un 1 al coche que circula por el carril A y secuencialmente envía por otro canal un 0 al coche que circula por el carril B. Durante 10 segundos, se mantiene en este estado, enviando los mensajes de la misma forma y posteriormente pasa al estado Yellow, en el cual el semáforo manda un 0 por ambos canales. Tras 5 segundos en este estado, el semáforo pasa a estado Red. Envía un 0 a los coches del carril A y un 1 a los que circulan por el carril B. Así se mantiene durante 10 segundos. Tras los 10 segundos el semáforo pasa a Yellow y finalmente pasará a Green. El semáforo mantendrá este comportamiento durante todo su periodo de funcionamiento.

Como se puede observar, utilizando únicamente un Gigglebot se pueden mostrar las luces de uno de los carriles. Por lo tanto, se añade un segundo Gigglebot como semáforo para indicar con luces el estado del otro carril. Se utiliza un sistema de Maestro-esclavo, con el que se da uso a un tercer canal.

El máster envía un número a través del tercer canal. Ese número es obtenido de la resta de los valores que le envía a los otros canales ($\mathrm{canal2} - \mathrm{canal1}$). Se asume que el máster controla el canal 1. En la siguiente tabla 2.10.4 se muestra la relación y el funcionamiento de este sistema.

El esclavo recibirá un número dependiendo del estado en el que se encuentra el master.

Cuando se encuentra en Green el esclavo recibirá un -1 y el esclavo pondrá luz roja; cuando recibe 0 pondrá amarillo y cuando recibe 1 pondrá luces verdes.

2.11. Señales

Para introducir variedad en el circuito, se han adaptado dos señales típicas en las redes de carreteras. La señal de cambio de velocidad y la de Stop.

El diseño de estos elementos conllevó únicamente a introducir el código de barras para la lectura en una recta. No fue necesario diseñar más.

2.11.1. Señal obligatoria de cambio de velocidad

El robot cambia su velocidad inicial a la indicada por el código. En esta versión el robot no vuelve a su velocidad inicial, se mantiene en la velocidad para siempre o hasta que encuentra otro código indicando de nuevo un cambio de velocidad. No obstante, como quedó explicado anteriormente, siempre va a cambiar a la velocidad mínima en las situaciones críticas.

2.11.2. Señal de STOP

El robot se quedará esperando durante un tiempo determinado. El Gigglebot elegirá de forma aleatoria el valor del tiempo de espera dentro de un rango. En este caso se decidió utilizar un rango entre 10 y 15 segundos. Así, se aprecia el momento de parada y no queda bloqueado un tiempo excesivo. No obstante, se podrá cambiar los límites del rango rápidamente modificando las variables en el código.

2.12. Estado LOST

Este estado indica que el Gigglebot no es capaz de encontrar la carretera, por lo tanto no sabe por dónde tiene que avanzar. Si se encuentra en este estado el Gigglebot se quedará parado y mostrará por la pantalla de Microbit la palabra ‘LOST‘. Posteriormente volverá a obtener valores de los sensores. En el caso de no obtener unos valores favorables por parte de los sensores

Sensor Izq. (s1)	Sensor Der. (s2)	Diferencia ($y = s2 - s1$)	Estado Final
Negro	Blanco	$y < 0$	LOST
Blanco	Blanco	$y >= 0$ && $y <$ Limite	LOST
Blanco	Blanco	$y >$ Limite	ROAD

Cuadro 2.4: Posibles casos en los que el coche acababa después de un salto.

y de no encontrar la carretera, es necesario recolocar el Gigglebot de forma manual porque si no permanecería en esta situación de manera infinita.

El Gigglebot llega a este estado después de los estados de acción: ENTRANCE_ROAD, EXIT_ROAD y RADIO. Por ello es necesario, que según termine el salto, el coche pueda detectar la carretera para poder avanzar. Tras ejecutar el salto, el Gigglebot toma medidas de los dos sensores de línea y dependiendo de los valores obtenidos se decidirá si ha encontrado la carretera o no. Esta decisión se calcula utilizando la diferencia entre los valores obtenidos por los dos sensores. Sin embargo, hay que tener en cuenta varios aspectos:

El primero es por la carretera. Como ésta es muy estrecha, ambos sensores no podrán obtener a la vez valores cercanos a cero (no podrán detectar a la vez la misma intensidad de negro). Por lo tanto si la diferencia es cercana a 0, asumimos que los sensores están obteniendo valores blancos (la intensidad de blanco es alta).

El segundo aspecto es que se introduce un valor límite, gracias al cual se entenderá que la diferencia entre los sensores es suficiente como para que la intensidad detectada sea diferente. En nuestro caso hemos tomado como límite un valor de 50

Asumiendo esto, tenemos tres casos de estudio. Véase la tabla 2.12.

1. La diferencia es menor que 0: En este caso, el sensor de la derecha detecta blanco y el de la izquierda detecta negro, es decir, el valor del sensor de la izquierda será menor que el de la derecha. El Gigglebot ha perdido la carretera, por lo tanto el estado final será ‘LOST‘.

2. La diferencia se encuentra entre 0 y el límite: En este supuesto se entiende que la diferencia es insignificante y se asume que los dos están en blanco, por lo tanto, el coche ha perdido la localización de la carretera y como consecuencia se quedará parado.

3. La diferencia es mayor que el límite: Los sensores están detectando una intensidad de color diferente, por lo tanto el sensor de la izquierda ha detectado blanco y el de la derecha negro y el coche podrá continuar con su camino ya que conoce donde está la carretera.

No obstante estos tres estados se pueden reducir a dos, agrupando los dos primeros casos de estudio, y como consecuencia obtenemos un código más corto y más limpio. El código final atendiendo a lo anteriormente explicado es el siguiente:

```
functionamILost() {
   let amILost
   let valueLimit = 50
   let right = gigglebot.lineReadSensor(gigglebotWhichTurnDirection.Right)
   let left = gigglebot.lineReadSensor(gigglebotWhichTurnDirection.Left)
   let diff = left - right

   if (diff<0) {
      amILost = "LOST"
   } elseif (diff>= 0&&diff<= valueLimit) {
      amILost = "LOST"
   } else {
      amILost = "ROAD"
   }
   return amILost
}
```

Para evitar el bloqueo del Gigglebot en el caso de que no encontrara la carretera, se desarrolló una función más compleja. Esa función consiste en dar indicaciones al Gigglebot para que, automáticamente inicie un proceso de búsqueda de la carretera. Lamentablemente, después de muchos intentos y pruebas no se llegó a obtener una solución factible. Finalmente, se decidió mantener el código inicial ya que, aunque sea más simple y por lo tanto menos óptimo, se obtuvieron menos problemas y errores.

El código de la función de búsqueda de la carretera se puede encontrar en el Anexo A.

Capítulo 3

Resultados y Conclusiones

En este apartado se estudian todos los aspectos relacionados con los resultados y con los objetivos conseguidos. Además, también se tiene en cuenta aquellos problemas que han aparecido durante la elaboración del proyecto, así como las posibles soluciones que se han intentado diseñar para solventar dichos problemas y las soluciones que finalmente se incorporaron al proyecto.

3.1. Consecución de objetivos

Si nos fijamos en el objetivo principal de este trabajo, se puede afirmar que se ha conseguido obteniendo unos resultados muy buenos. Se ha desarrollado una red de carreteras en la que los Gigglebot se mueven de una forma independiente. No obstante, se van a explicar cada subobjetivo por separado para explicar independientemente cada resultado obtenido.

SubObjetivo 1: Para la elaboración de los diseños nos basamos en el trabajo de Loic [9] y de Kern [10] por lo tanto se concluyó en hacer los diseños de las carreteras degradadas. Aunque los diseños realizados no son los mejores y se han producido bastantes problemas, los diseños han cumplido su función y permiten que los coches puedan utilizar los sensores para poder circular por ellos. Inicialmente, se iba a elaborar un único circuito pero debido a problemas con el espacio ya que ocupaba $12\mathrm{m}^2$ se decidió realizar dos circuitos más pequeños en los que se probarían elementos diferentes.

SubObjetivo 2: Para que los coches leyeran la carretera, se utilizan los sensores que miden la intensidad de color (escala de grises). Obtenemos que los coches son capaces de leer la carretera

con una probabilidad de error muy baja.

SubObjetivo 3: Con el movimiento más rápido del coche se consigue que oscile más dentro del circuito y puede ocurrir que pierda la carretera. Debido a esto, en vez de utilizar unos cambios de velocidad extremos (30-40-50), se utiliza un único cambio más ligero (30-33).

SubObjetivo 4: En cuanto a los elementos diseñados, se han implementado una gran variedad de elementos y han funcionado todos correctamente.

SubObjetivo 5: Respecto a la lectura de códigos se ha obtenido un modelo de lectura eficiente, pero tiene la necesidad de que se realice a una velocidad baja. Utilizando estos códigos la probabilidad de que el coche lea de una forma incorrecta es muy baja. Hemos obtenido un total de 16 códigos que se podrían reutilizar si usamos los mismos elementos.

SubObjetivo 6: Se ha diseñado un recorrido de calibración para poder hacer que los componentes externos afecten lo menos posible al coche y al recorrido que éste realiza por el circuito.

SubObjetivo 7: Para la realización de los semáforos se ha utilizado la conexión de dos Gigglebot vía radio, con esto se ha conseguido el envío de números para indicar la acción que se podía hacer.

SubObjetivo 8: En los momentos en los que el robot pierde la carretera debido a un salto, el objetivo era que el coche fuera capaz de volver a la carretera por sí sólo, sin embargo no se logró. En vez de eso se decide que si la ha perdido, el coche no se mueva y permanezca quieto hasta que alguien lo recoloque manualmente.

SubObjetivo 9: Se logró que los coches pudiesen interactuar entre ellos en cuanto a que no se choquen. Si se detecta un objeto demasiado cerca, se ajusta la velocidad reduciéndola progresivamente hasta que en un determinado momento queda parado completamente.

Como se puede observar no ha sido posible completar la totalidad de los objetivos, sin embargo se han planteado alternativas, que aunque no conducen a soluciones óptimas se logró paliar los efectos negativos.

Por lo tanto, teniendo en cuenta estos resultados, es una buena idea poder utilizar estas tecnologías en la docencia y aplicarla a las clases.

Tarea	Inicio	Fin
Red de carreteras con Microbit y Gigglebot	11/02/2019	03/07/2019
Estudio de tecnologías con Python	11/02/2019	29/03/2019
Establecimiento Inicial de máquina de estados	01/04/2019	24/04/2019
Diseños de las carreteras bases	24/04/2019	03/05/2019
Primeros programas para la circulación del robot	06/05/2019	20/05/2019
Pruebas de la circulación del coche y ajustes	20/05/2019	10/07/2019
Inclusión de varias velocidades	11/07/2019	12/07/2019
Proceso de Lectura de códigos + pruebas	12/07/2019	18/10/2019
Realización de elementos salida de carreteras	21/10/2019	28/10/2019
Cambios en método de lectura respecto a velocidades	28/10/2019	11/11/2019
Problema con la memoria y Python	11/11/2019	04/12/2019
Estudio de las tecnologías con Static TypeScript	05/12/2019	12/12/2019
Traducir el código a Static TypeScript	13/12/2019	16/01/2019
Interacciones entre los coches (freno progresivo)	16/01/2020	27/01/2020
Estudio de formas para la calibración	28/01/2020	24/02/2020
Elementos de circuitos	25/02/2019	24/04/2020
Rotonda (diseño y modificaciones del código)	25/02/2020	28/02/2020
Señal de velocidad (cambios en las velocidades)	02/03/2020	06/03/2020
Señal de Stop	09/03/2020	10/03/2020
Estudio de módulo de radio	10/03/2020	13/03/2020
Semáforos	13/03/2020	24/04/2020
Semáforo peatón	13/03/2020	19/03/2020
Semáforo doble	19/03/2020	10/04/2020
Semáforo de distancia	10/04/2020	24/04/2020
Elaboración de circuitos	13/03/2020	03/04/2020
Pruebas con los circuitos y ajustes	03/04/2020	30/06/2020
Memoria	28/04/2020	03/07/2020

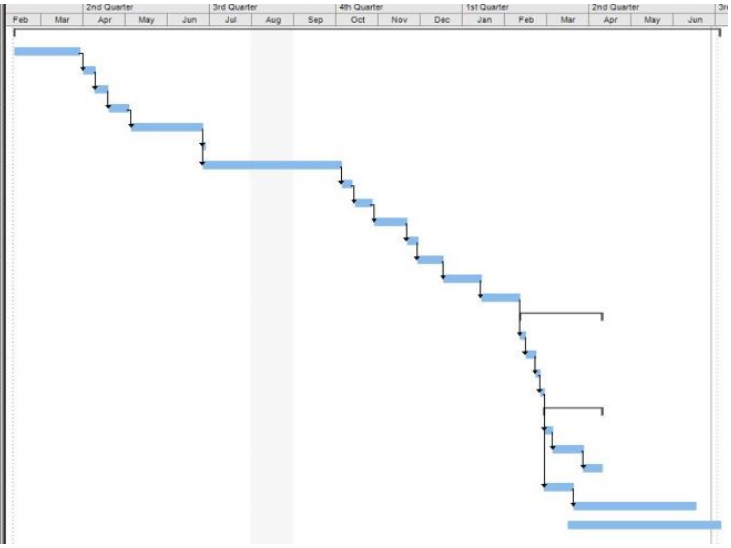

Figura 3.1: Diagrama de Gantt.

3.2. Planificación

En este apartado se muestra la planificación que se siguió para realizar el trabajo. En la figura 3.1 se puede ver un diagrama de Gantt indicando el tiempo que se utilizó para realizar el proyecto. Como se puede observar este trabajo se organizó en diferentes subtareas diferenciadas que se podían realiza por separado.

En la tabla 3.2 se puede observar de una forma más detallada la forma en la que se realizó el trabajo. En esta tabla se indican, además de los intervalos de tiempos utilizados para completar las tareas, las horas realizadas durante cada intervalo y una representación de las horas expresadas en esfuerzo (persona/mes trabajado).

3.3. Trabajos futuros

Todos los aspectos de mejora que aparecen indicados se han obtenido observando los posibles errores y problemas y tratando de buscar una solución, no obstante por diversas razones no se han podido implementar:

- Mejora en los procesos de calibración de velocidad y desvíos: conseguir que sean independientes los dos calibraciones y que puedan realizarse por separado, para que un aspecto no afecte al otro.

Tareas	Inicio	Fin	Dias	Horas	Pers/mes
Red de carreteras con Microbit y Gigglebot	11/02/2019	03/07/2020	508	795	5.679
Estudio de tecnologías con Python	11/02/2019	29/03/2019	46	70	0.5
Establecimiento Inicial de máquina de estados	01/04/2019	24/04/2019	23	35	0.25
Diseños de las carreteras bases	24/04/2019	03/05/2019	9	15	0.107
Primeros programas para la circulación del robot	06/05/2019	20/05/2019	14	21	0.15
Pruebas de la circulación del coche y ajustes	20/05/2019	10/07/2019	50	75	0.536
Inclusión de varias velocidades	11/07/2019	12/07/2019	1	4	0.0286
Proceso de Lectura de códigos + pruebas	12/07/2019	18/10/2019	98	100	0.714
Realización de elementos salida de carreteras	21/10/2019	28/10/2019	7	11	0.079
Cambios en método de lectura respecto a velocidades	28/10/2019	11/11/2019	14	20	0.143
Problema con la memoria y Python	11/11/2019	04/12/2019	23	35	0.25
Estudio de las tecnologías con Static TypeScript	05/12/2019	12/12/2019	7	12	0.089
Traducir el código a Static TypeScript	13/12/2019	16/01/2019	34	25	0.179
Interacciones entre los coches (freno progresivo)	16/01/2020	27/01/2020	11	15	0.107
Estudio de formas para la calibración	28/01/2020	24/02/2020	27	40	0.286
Elementos de circuitos	25/02/2019	24/04/2020	59	87	0.622
Rotonda (diseño y modificaciones del código)	25/02/2020	28/02/2020	3	8	0.057
Señal de velocidad (cambios en las velocidades)	02/03/2020	06/03/2020	4	10	0.071
Señal de Stop	09/03/2020	10/03/2020	1	3	0.021
Estudio de módulo de radio	10/03/2020	13/03/2020	36	6	0.043
Semáforos	13/03/2020	24/04/2020	42	60	0.429
Semáforo peatón	13/03/2020	19/03/2020	6	8	0.057
Semáforo doble	19/03/2020	10/04/2020	22	32	0.229
Semáforo de distancia	10/04/2020	24/04/2020	14	20	0.143
Elaboración de circuitos	13/03/2020	03/04/2020	21	30	0.214
Pruebas con los circuitos y ajustes	03/04/2020	30/06/2020	88	125	0.893
Memoria	28/04/2020	03/07/2020	66	75	0.536

Cuadro 3.1: Tabla descriptiva de la planificación. Vienen incluidas las tareas, los intervalos de tiempos y las horas en las que se realizaron las tareas.

- Calibración del sensor de línea: para obtener los datos de forma dinámica, en vez de establecer los datos de forma estática; esto provocará que el funcionamiento del robot sea mejor.
- Lograr el diseño de curvas cerradas para que se puedan hacer giros de 90º.
- Lograr la utilización de velocidades más altas en el circuito evitando que no se salgan del circuito.
- Realización de los procesos críticos utilizando las diferentes velocidades.
- Terminar el proceso de Perdida de carretera, para conseguir que vuelva al recorrido.
- Reducir el tamaño de los códigos de lectura para que ocupen un menor espacio y poder introducir un mayor numero de bits y en consecuencia obtener más códigos.

3.4. Demostraciones

Para poder ver los resultados de este proyecto se ha creado un repositorio en Github [3] con todos los elementos creados en este trabajo. En este se muestran los vídeos de las demostraciones, el código de todos los elementos (Coches y semáforos) y un documento con otras URLs que puede servir de utilidad.

Apéndice A

Código Recuperar la carretera

```
function estadoLost(right: number, left: number) {

    if (!sensorActivated("lostLine", left) &&
        !sensorActivated("lostLine", right)) {
        //Tengo que ir a la derecha
        //Contador de tiempo para meter menos potencia a un motor que otro
        contTime(100, speed, 0)
        gigglebot.stop()
        contTime(100, speed, speed)
    } else if (sensorActivated("lostLine", left) &&
              !sensorActivated("lostLine", right)) { //izq Neg derecha blan
        //Tengo que ir a la izquierda
        contTime(100, 0, speed)
        gigglebot.stop()
        contTime(100, speed, speed)
    } else {
        //Caso no contemplado
    }
}
```

Bibliografía

[1] Api micropython. https://microbit-micropython.readthedocs.io/en/latest/microbit_micropython_api.html.

[2] Gigglebot. https://gigglebot.io/.

[3] Github. https://github.com/TomasGalindo/Trabajo_Fin_De_Grado.

[4] Makecode. https://makecode.microbit.org/#.

[5] Microbit. https://makecode.microbit.org/.

[6] Minifier. https://pypi.org/project/python-minifier/.

[7] Mueditor. https://codewith.mu/.

[8] M. Ben-Ari and F. Mondada. *Elements of Robotics*. Springer Open, 2018.

[9] L. Dubois, F. Mondada, C. Barraud, and S. Witwicki. Thymio Road Network Local Infraestructure. https://aseba.wdfiles.com/local--files/fr%3Aroadnetwork/Report_Loic.pdf, 2015.

[10] B. Kern, F. Mondada, C. Barraud, and S. Witwicki. Thymio Road Network Global Infraestructure. https://aseba.wdfiles.com/local--files/fr%3Aroadnetwork/ReportBenFr.pdf, 2015.