Informe de Proyecto

Sistema de Vehículo Autoguiado (AGV)
con Control de Tráfico

Bernardo Aceituno-C, Alejandro Meilan,
Luis Pérez Bustos, Carlos Luis.

Índice

1. Introducción

El proyecto asociado a la asignatura EC3883 "Laboratorio de Proyectos III" corresponde a un trabajo grupal que engloba diversas áreas de la electrónica como circuitos analógicos, sistemas embebidos, control y comunicaciones.

El proyecto consiste en diseñar un vehículo autónoma con capacidades de seguimientos de una línea eléctrica y con componentes adicionales de seguridad como detección de líneas de paradas, detección de obstáculos y un controlador de alto nivel que le indica una ruta para llegar a un punto deseado.

En este documento se incluye información concisa sobre la primera propuesta inicial así como detalles de la implementación final del proyecto, con diagramas e imágenes para facilitar su entendimiento al igual que evidencia videográfica de su funcionamiento.

2. Estructura del Proyecto

Esta sección presenta una visión general del funcionamiento y la implementación del proyecto.

2.1. Objetivo General

- Realizar vehículo autoguíado (AGV) por detección de señal electromagnética con algortimos de planificación de trayectoria y detección de obstáculos a través de una cámara central.

2.2. Objetivos Específicos

1. Configurar funcionamiento de los motores con control PWM a través del driver L293D.
2. Desarrollar circuito de adquisición y acondicionamiento de señal electromagnética para mantener el vehículo en línea recta.
3. Detectar e implementar lógica de toma de decisiones frente a señales de cruce.
4. Detectar obstáculos que se encuentren a cierto rango de distancia e implementar lógica de toma de decisiones.
5. Diseñar algoritmo que controle los movimientos del vehículo a través de una camara web para evitar colisiones con obstáculos u otros vehículos que se encuentren simultáneamente en la cuadrilla.
6. Desarrollar protocolo de comunicación infraroja entre el computador y el vehículo.

2.3. Etapas

Para detallar el funcionamiento esperado del proyecto a desarollar se dividió a este en las siguientes etapas.

1. **adquisición y acondicionamiento de señal electromagnética**
2. **control PWM de los motores**
3. **detección de señal de cruce**
4. **mapeado de la cuadrilla**
5. **algortimo de planificación**
6. **protocolo de comunicación**

2.4. Representación General

El diagrama de la Figura 1 ilustra el esquema general del proyecto. Se observa el funcionamiento en conjunto de las etapas descritas con anterioridad.

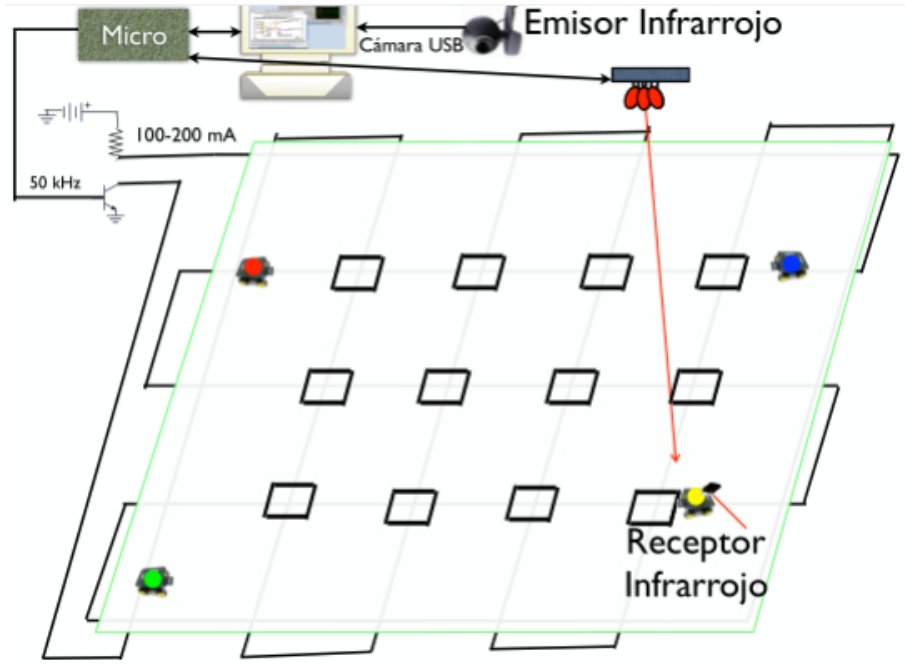

Figura 1: Esquema general del Proyecto

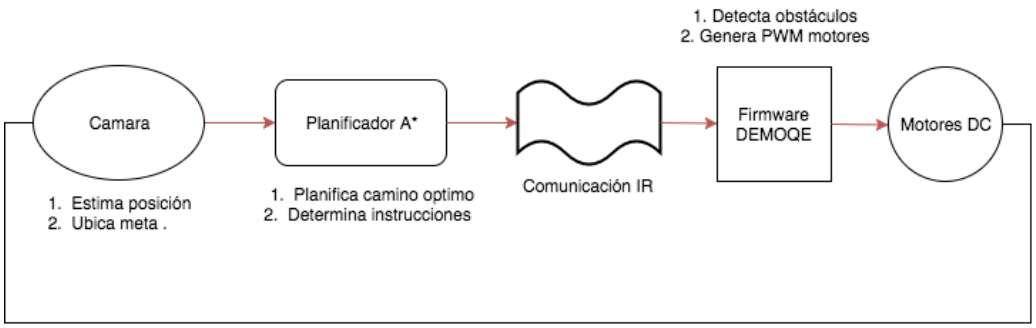

Figura 2: Diagrama de fucionamiento del sistema

3. Hardware

A continuación se presentan los circuitos utilizados para el acondimiento de señales y alimentación de los sensores y actuadores utilizados.

3.1. Fuentes de Voltaje

Se utilizaron dos fuentes de voltaje para el proyecto: una para alimentar los motores DC del vehículo y una para alimentar el resto de los sensores. Ambas fuentes se presentan en la Figura 3

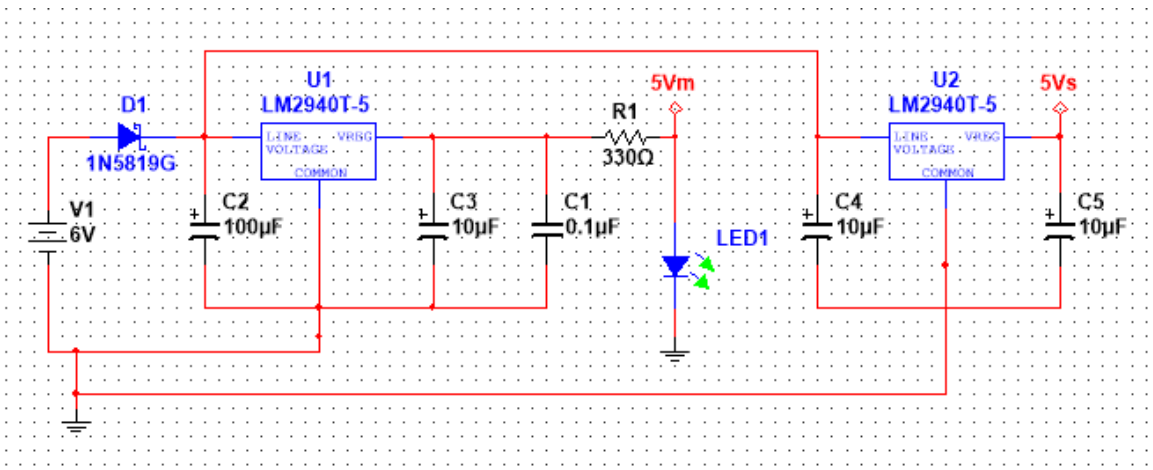

Figura 3: Fuentes de voltaje DC 5V

La señal V5m corresponde a la alimentación de los motores y la señal V5s a la alimentación de los sensores. En ambos casos se utiliza el integrado LM2940-5 que es un regulador de 5V DC. La entrada de 12V proviene del conector DC del mesón, y se utiliza un diodo rectificador Schottky de protección. En ambas fuentes se utilizan capacitores electrolíticos de bypass, y para la fuente del motor se añade tambien un capacitor cerámico extra para asegurar una alimentación libre de ruido. El LED sirve para verificar en todo momento la correcta alimentación de los motores.

3.2. Acondicionamiento de las bobinas

Para este circuito se presentan dos propuestas similares, y a través de simulaciones se determinó cual de ellas era preferible dada la naturaleza del proyecto.

- **Propuesta 1:** el circuito de la Figura 4 es la primera propuesta para acondicionar la señal proveniente de las bobinas.

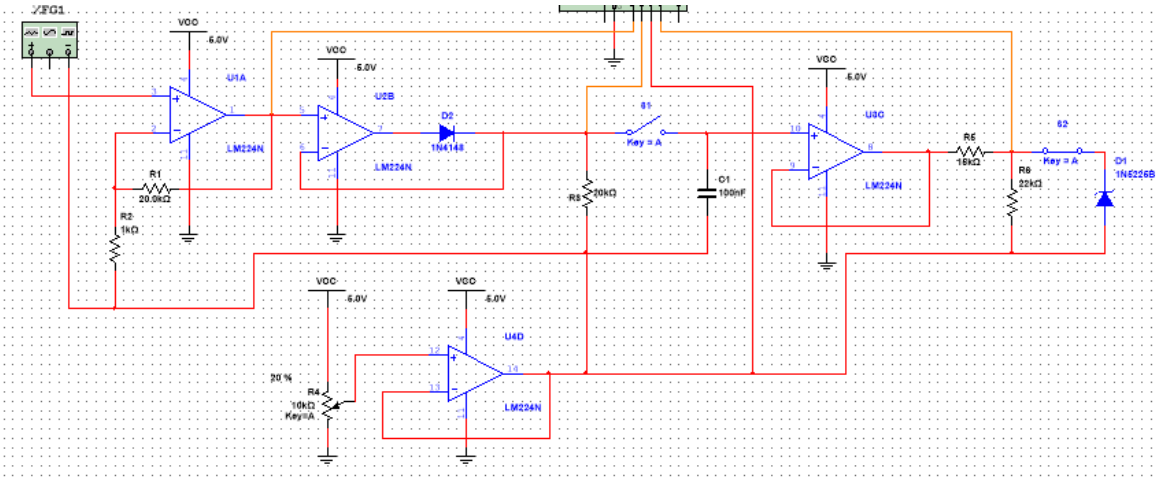

Figura 4: Acondicionamiento de bobinas - Propuesta 1

Del lado izquierdo hay una primera etapa de amplificación de la señal, con una ganancia de 20 y con una tierra virtual implementada con el seguidor de la parte de abajo, cuya salida de voltaje se maneja con el potenciómetro R4. Seguido de esta etapa de amplificación hay un rectificador de media onda utilizando el diodo 1N4148. La tercera etapa de la adquisición es un detector de pico, para el cual se escogió una combinación de resistencia y capacitor en simulación que otorgara una respuesta adecuada. El buffer de la última etapa se coloca para aislar la salida de las impedancias del resto del circuito, ya que esta salida entrará al microcontrolador. Se coloca un divisor de tensión y un diodo Zener de 3V para proteger al microcontrolador. Los valores del divisor de tensión deberán ser ajustados en práctica dependiendo de las salidas máximas y mínimas de las bobinas.

Se simuló el canal de adquisión con una señal sinusoidal de 50KHz y 50mVp que hiciera las veces de la señal de la bobina. Primero se levantó el switch A para observar la señal del rectificador de media onda (CH2), como se presenta en la Figura 5

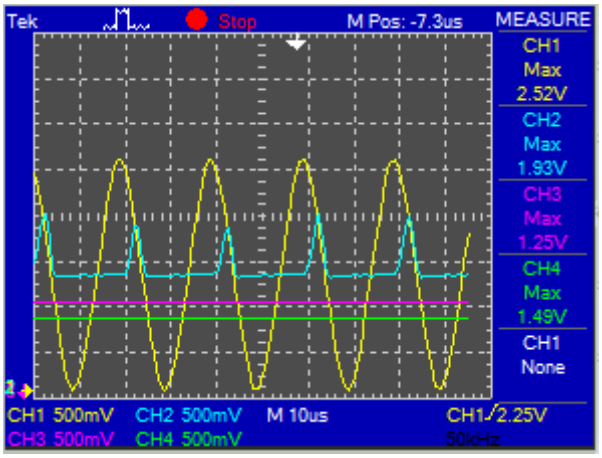

Figura 5: Amplificador (CH1), Rectificador sin detector de pico (CH2), tierra virtual (CH3), salida final (CH4)

Se observa que la rectificación con un solo diodo no es perfecta, principalmente por las frecuencias a las que se está trabajando. Internamente el diodo presenta retardos respecto a la frecuencia de trabajo de las bobinas que impiden que la rectificación sea mejor. Al bajar el switch se observa en la Figura 6 la respuesta del rectificador con detector de picos en el canal 2.

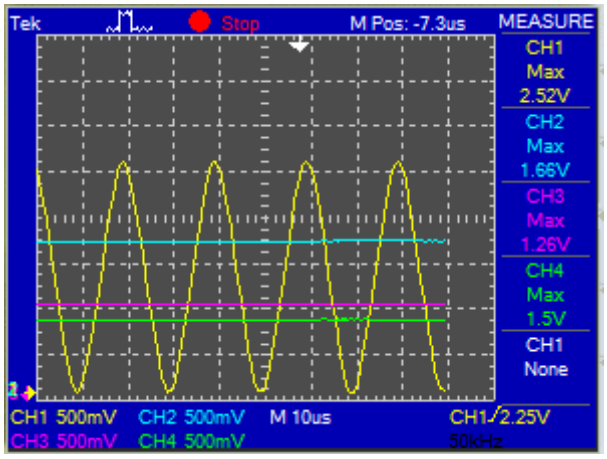

Figura 6: Amplificador (CH1), Rectificador con detector de pico (CH2), tierra virtual (CH3), salida final (CH4)

Utilizando una tierra virtual de 1.25V, se observa que se levanta el nivel DC de la señal amplificada de la bobina y se elimina la componente negativa de la misma. En el canal 2 se muestra la salida del rectificador con detector de pico, que logra un nivel DC de 1.64V. Finalmente, la señal que entrara al microcontrolador es una DC de 1.48V que es proporcional a la intensidad de campo sobre la bobina.

- **Propuesta 2:** la segunda propuesta para el acondicionamiento de las bobinas se muestra en la Figura 7:

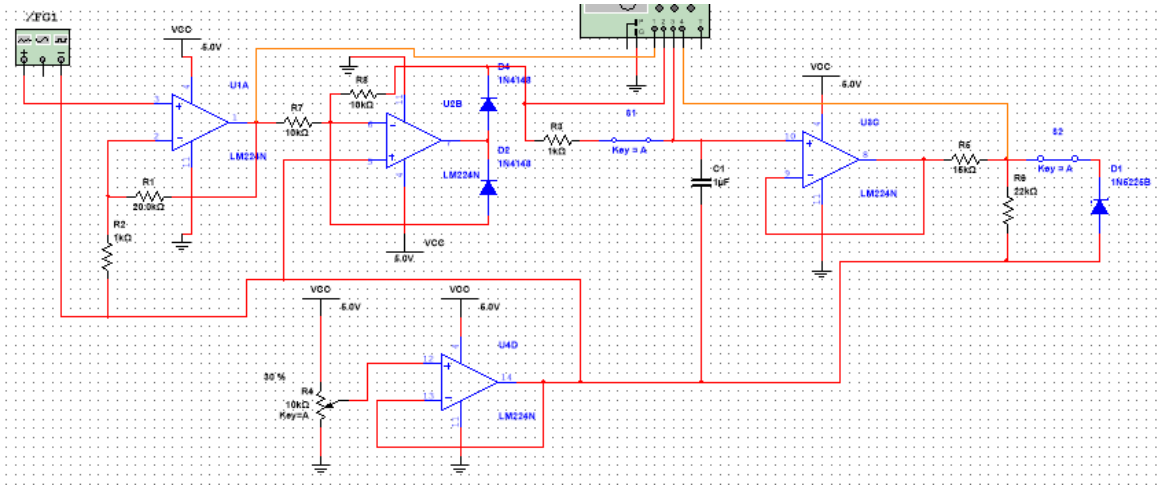

Figura 7: Acondicionamiento de bobinas - Propuesta 2

La diferencia respecto a la primera propuesta es el circuito rectificador, que en este caso se hace con dos diodos y se trata de un rectificador de media onda negativa. Después del rectificador se sustituyó el circuito de detección de picos por un filtro pasabajo con frencuencia de corte $1/RC = 1000rad/s = 159{,}15Hz$. Este nuevo circuito de rectificación otorgó mejores resultados que en el caso anterior, como muestra la Figura 8:

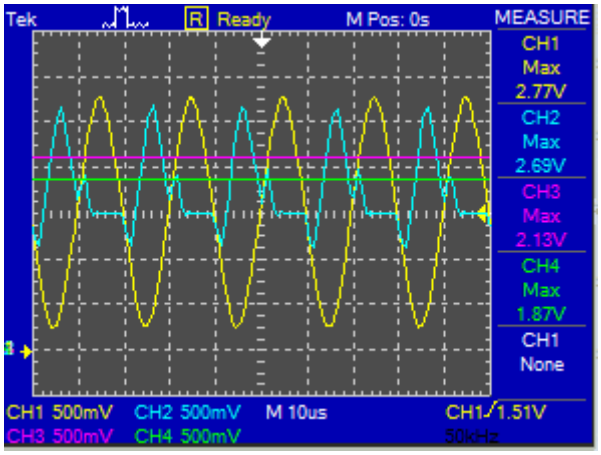

Figura 8: Amplificador (CH1), Rectificador (CH2), Filtro pasabajo (CH3), Salida final (CH4)

La señal del canal 2 muestra la salida del rectificador, que al compararla con la señal obtenida con la primera propuesta se nota que presenta una respuesta más adecuada, con menos retardos en la señal rectificada. El valor de voltaje de la salida del canal de adquisición es mayor gracias a la rectificación mejorada, por lo que esta propuesta es la que se implementará en el proyecto.

El circuito final implementado es el mostrado en la Figura 9

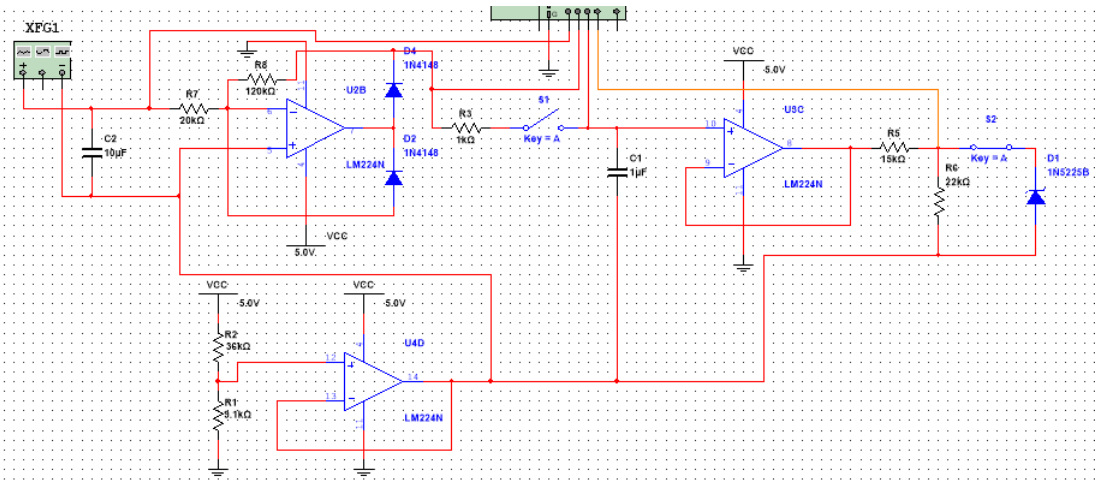

Figura 9: Circuito final para adquisición de señal de bobinas

Los principales cambios introducidos en la versión final es la eliminación de uno de los operacionales para realizar la etapa de rectificación y amplificación con un sólo opamp, además de reducir la ganancia de 20 a 6 para obtener una máxima excursión en la señal DC obtenida a través del circuito de acondicionamiento. Se sustituyó el potenciómetro que fijaba la tierra virtual por una combinación de resistencias que dio un offset DC de 1V (usando 9.1k y 36k).

La primera acción al montar el circuito fue determinar la frecuencia de resonancia de las bobinas utilizadas, es decir, la frecuencia en la que la amplitud de la onda sinusoidal era máxima, que resultó ser de 56KHz y fue la frecuencia utilizada para excitar estos sensores (utilizando el cable de la pista). Con este circuito se hicieron mediciones con dos casos especiales: la bobina está lejos de la pista, y la bobina está justo encima de la línea eléctrica. En el primero de los casos la salida del circuito de acondicionamiento era directamente el nivel DC de la tierra virtual del sistema (1V), mientras que al estar encima de la línea el voltaje arrojado por la bobina era el máximo posible que resultó ser de 2.88V. Estos datos fueron importantes a la hora de codificar el firmware de control de los motores pues permitía saber a groso modo en qué punto el carro estaba pasando sobre una de las líneas al realizar un giro.

3.3. Driver de los motores

Para controlar los motores se utilizan señales PWM, utilizando el driver de motores L293D, tal como se muestra en la Figura 10:

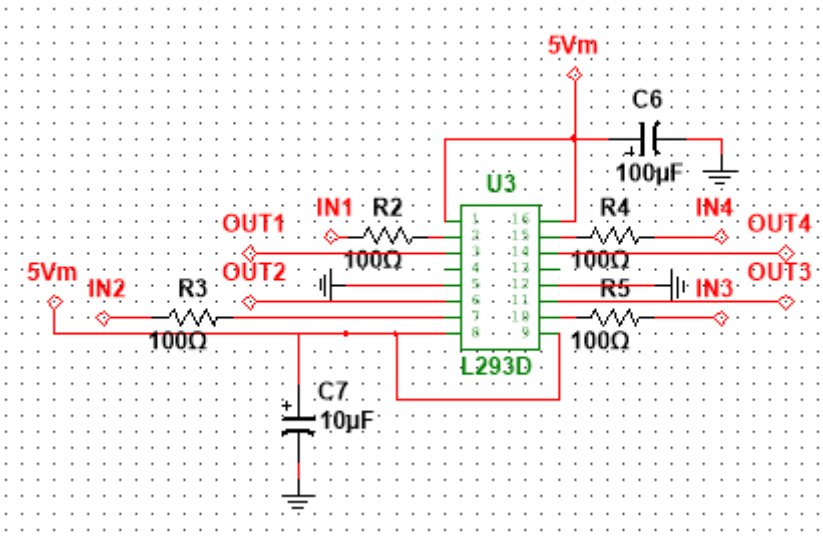

Figura 10: Driver de motores

Las entradas INx van al microcontrolador, a partir del cual se envian 1s y 0s lógicos para crear las señales cuadradas del PWM. Las salidas OUTx irán conectadas a los motores. El control del ancho de pulso se hace

también desde el microcontrolador y será la salida del controlador a implementar para que el vehículo siga la línea, según las lecturas de las bobinas.

3.4. Detección de Obstáculos

Con el objetivo de detectar objetos que obstruyan el camino del vehículo autoguíado, se utilizó el sensor de distancia por emisión infraroja *Sharp GP2Y0A21YK*. El objetivo fue incorporar una nueva primititiva que permitiera al vehículo tomar decisiones adecuadas de acuerdo a su entorno. La Figura 11 muestra el modelo comercial del cual se dispone en el laboratorio C. Las características eléctricas del dispositivo se listan a continuación:

- Vcc: 4,5V a +5,5V
- Rango de distancias: 10cm a 80cm
- Vo @ L=80cm: 0,25V a 0,55V
- Icc @ L=80cm: 30mA a 40mA
- Vo(@L=10cm) - Vo(@L=80cm): 1,65V a 2,15V

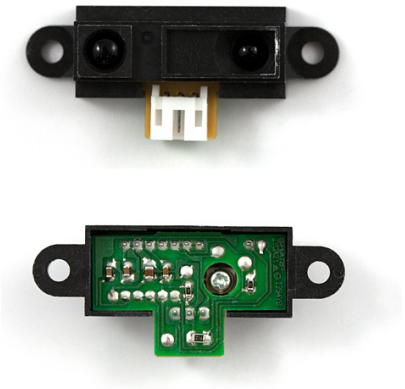

Figura 11: Sensor de proximidad *Sharp GP2Y0A21YK*

La Figura 12 ilustra la respuesta del sensor en su rango de funcionamiento correspondiente. Se observa un máximo entre 5cm y 10cm con una respuesta aproximadamente lineal hasta 30cm.

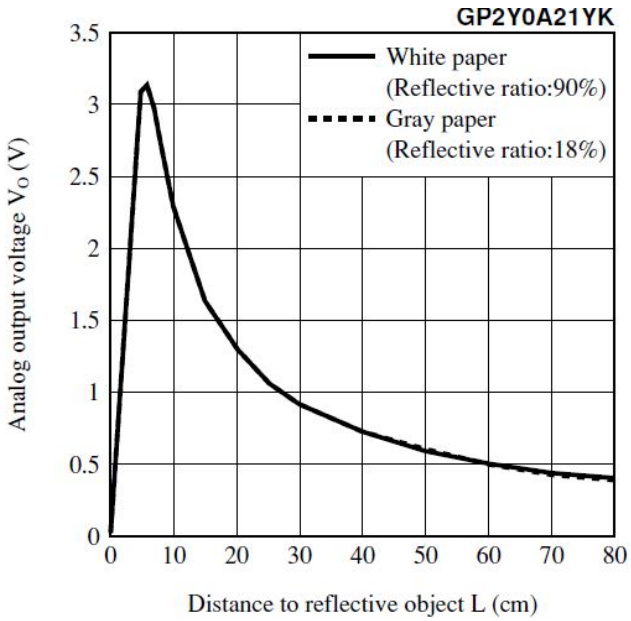

Figura 12: Relación Vo (V) vs. distancia (cm)

El funcionamiento del sensor GP2Y0A21YK se observa en la Figura 13. El detector infrarojo situado a la derecha del sensor recibe la señal reflejada desde una cierta superficie y enfocada a través del lente. El circuito interno del sensor permite arrojar una señal que sigue el patrón de la Figura 12.

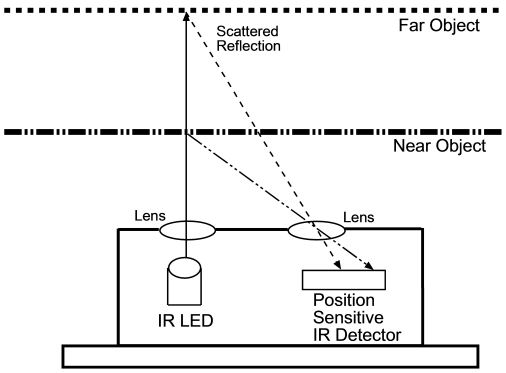

Figura 13: Funcionamiento del sensor de proximidad

Para adquirir la señal deseada desde el demoQE128, se acondicionó la señal utilizando el circuito de la Figura 14. Se utilizó la segunda etapa del circuito para reducir el nivel de voltaje en un 60 % y evitar máximos que sobrepasen los límites del micro-controlador.

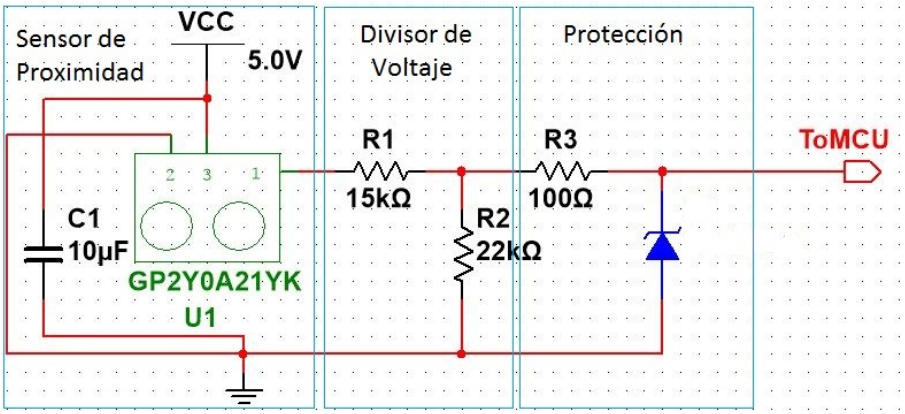

Figura 14: Circuito de adquisición y acondicionamiento

Experimentalmente se midieron los voltajes a la salida del circuito de adquisición para tener una idea de qué limite utilizar dentro del firmware para determinar que un obstáculo estaba efectivamente presente frente al vehículo. De este modo se podía ajustar con un sólo parámetro qué tan lejos se podían detectar los obstáculos, respetando los límites del sensor.

3.5. Detección de Señal de Cruce

A la lógica del vehículo autoguíado se añadió la detección de señales de cruce colocadas en las intersecciones de la cuadrilla. En la pista de pruebas existían marcas oscuras en dichos lugares. Para su adecuada detección se utilizó el sensor óptico TCRT1000 de la Figura 15. Es un sensor reflectivo que incluye un emisor infrarojo y un fototransistor en un mismo empaquetado sellado que bloquea la luz visible.

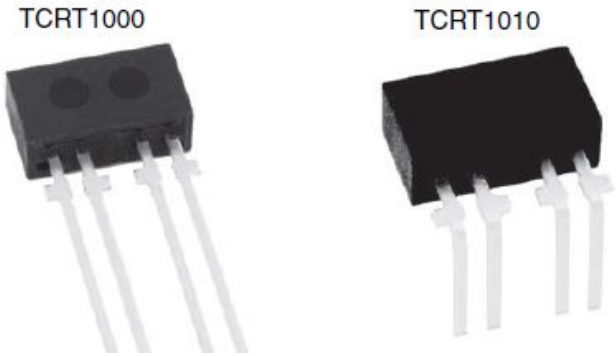

Figura 15: Sensor óptico *TCRT1000*

Las dimensiones del módulo se observan en la Figura 16.

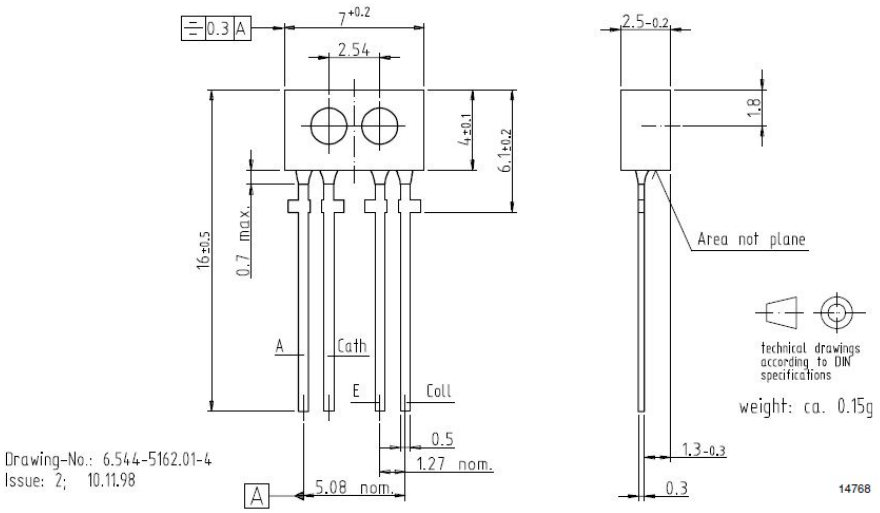

Figura 16: Dimensiones del sensor óptico *TCRT1000*

El emisor genera luz infraroja de hasta 940nm que es reflejada (o no) por la superficie correspondiente. La salida detecta esta luz y se habilita (o no) el paso de corriente de colector a emisor. La Figura 17 ilustra el funcionamiento.

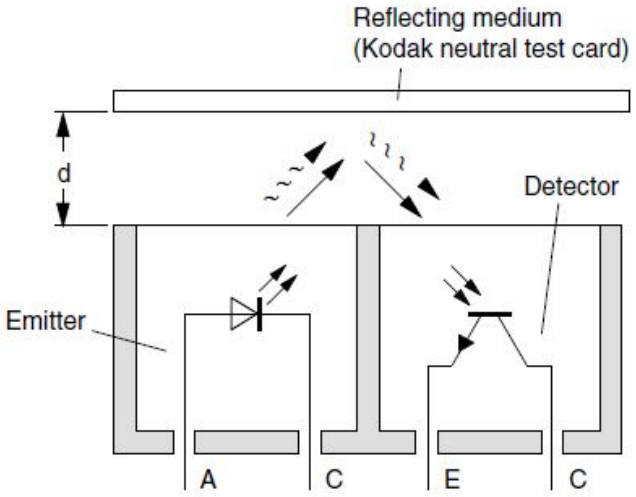

Figura 17: Funcionamiento del sensor óptico

Se añadió el circuito de acondicionamiento de la Figura 18. La salida del *TCRT1000* es a través de una resistencia *pull-up* que se conecta directamente al comparador *LM311*. La salida de este módulo depende de la diferencia de voltajes entre los pines 2 y 3. En la entrada inversora se fija un cierto voltaje de *threshold* con el cual se comparará la señal de la entrada no-inversora. Los niveles de tensión posibles estarán entre el %60 de Vcc (3V) y el voltaje fijado en el pin 1 (GND). Al final se agregó una etapa de protección como parte final del circuito.

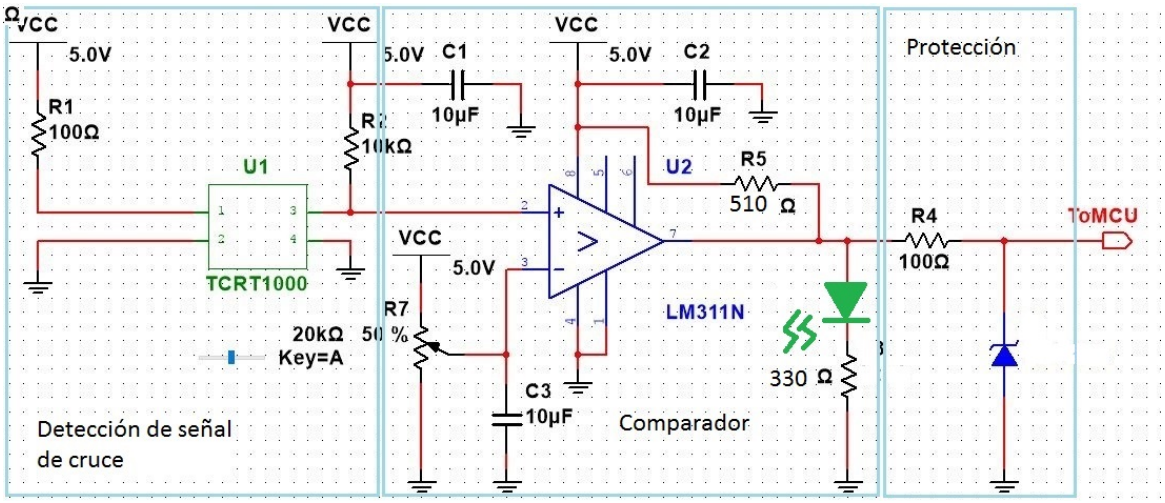

Figura 18: Circuito de adquisición y acondicionamiento

Experimentalmente lo más complicado de este circuito fue calibrar el potenciómetro para lograr, en primer lugar, detectar correctamente las líneas de parada en la pista, y en segundo lugar, evitar que el circuito arrojara falsos positivos dadas las vibraciones del carro o irregularidades en la pista. Esta parte fue esencial para el correcto funcionamiento del código embebido, puesto que éste dependía fuertemente de la detección de líneas para la realización oportuna de las rutas pedidas (cruzar hacia la derecha, izquierda, o seguir recto)

3.6. Circuito Seguidor de Línea

Esta configuración fue utilizada para acondicionar la línea de pista por la cual se desplazará el vehiculo. Consiste básicamente en obtener una señal desde el microcontrolador (onda cuadrada de 3V @56kHz) y pasarlo por un circuito emisor común. De esta forma la señal que debe seguir el carro a través de la bobinas estará correctamente acondicionada. Un esquematico del circuito es mostrado en la Figura 19.

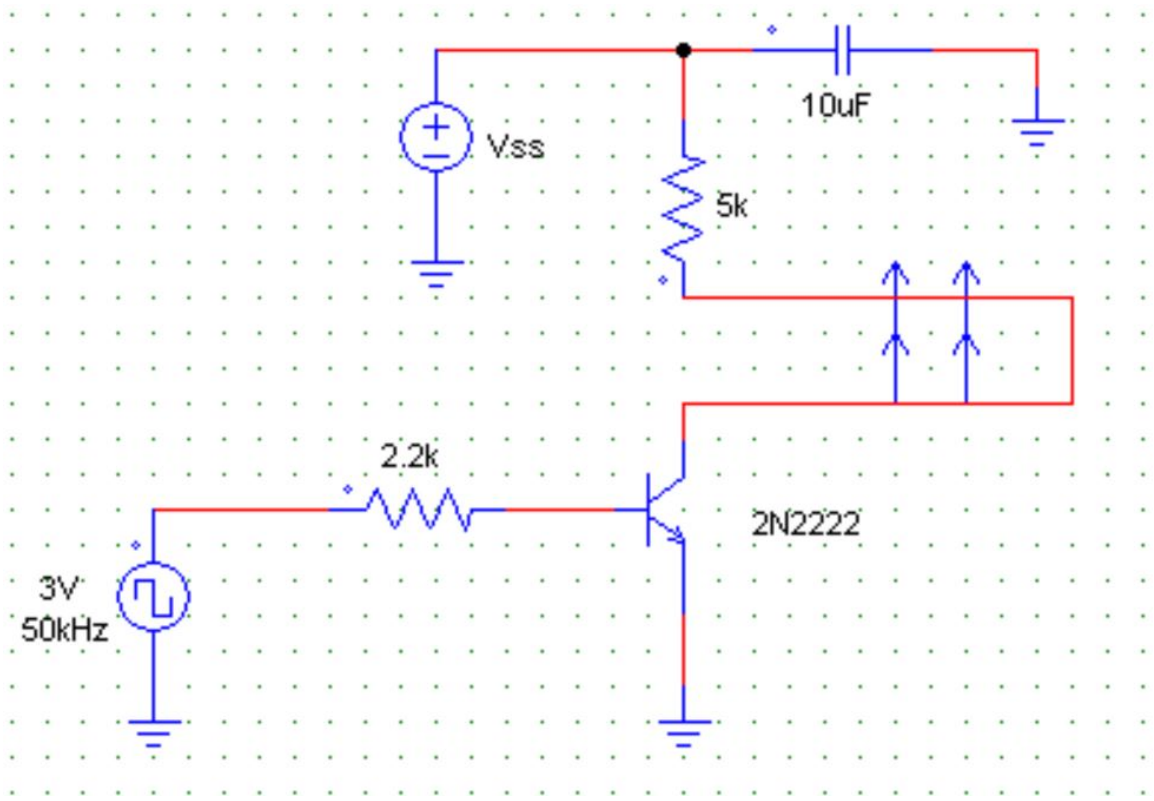

Figura 19: circuito Seguidor de Línea

3.7. Circuito Transmisión Infrarroja – Vehiculo / Mesón

Estas dos configuraciones son necesarias para acondicionar la transmisión infrarroja entre el carrito y el mesón. La primera consiste en un TSOP1136 protegido con un diodo zener de 3 voltios y una resistencia. El segundo es un circuito emisor común con un capacitor y un diodo en inverso. Ambas configuraciones son necesarias para la transmisión infrarroja entre los dispositivos. Estos circuitos mostrados en las figuras Figura 20 y Figura 21.

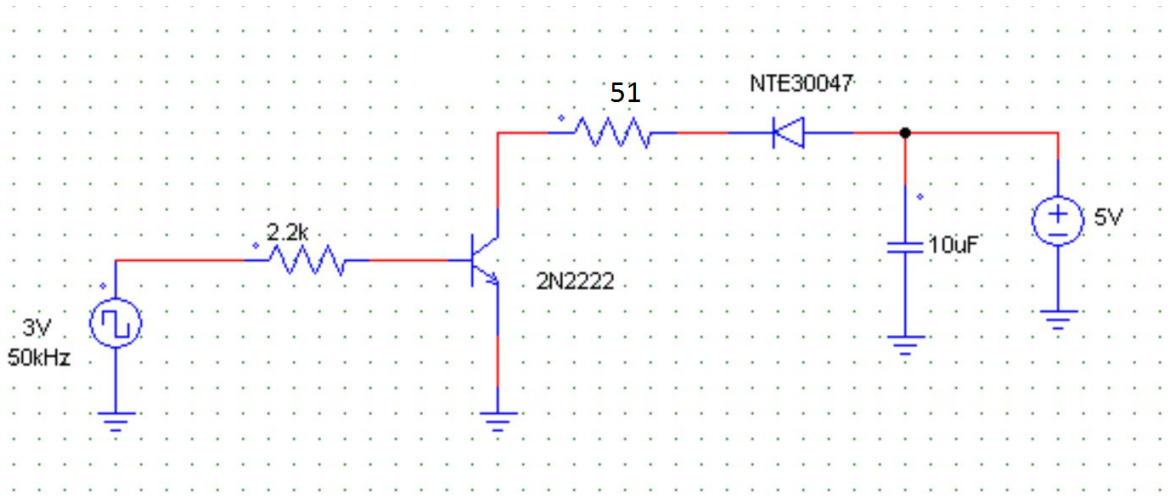

Figura 20: Circuito Trasmisión en mesón

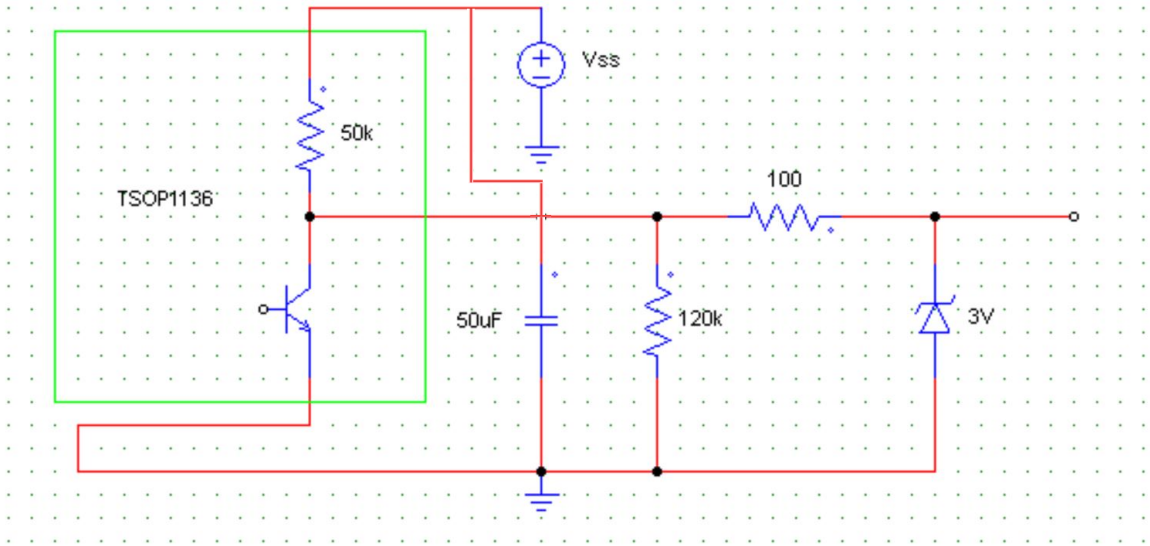

Figura 21: Circuito recepción vehículo

3.8. Montaje final

Una vez probados todos los circuitos en Protoboard, se procedió a soldarlos en baquelita para la construcción final del vehículo. Se utilizaron dos baquelitas:

- **Baquelita de bobinas:** esta baquelita iba rozando el piso y en ella se incluyeron los circuitos de ambas bobinas al igual que el circuito de detección de línea de parada, como muestra en la Figura 22. Fue de suma importancia fijar con fuerza esta baquelita al cuerpo del vehículo para minimizar las vibraciones y movimientos, que podían conllevar a falsos positivos del detector de línea de parada, lo que finalmente incurría en una mala ejecución de las rutas propuestas. La distancia entre el suelo y la baquelita fue tal que no rozara con la pista, pero que fuera lo suficientemente baja para que detectara correctamente las líneas de parada.

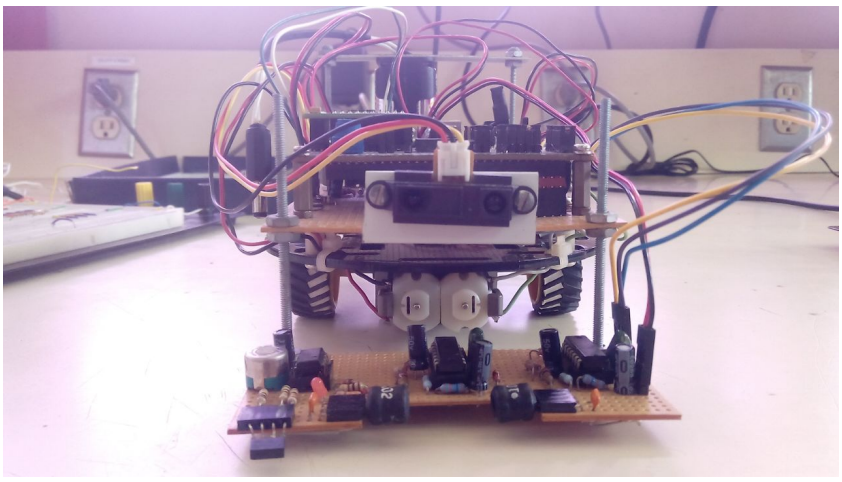

Figura 22: Baquelita de bobinas y detección de línea de parada.

- **Baquelita del microcontrolador:** esta baquelita constituyó el cuerpo del vehículo autónomo. Sobre ella reposó el demoQE128, al igual que los circuitos correspondientes a los drivers de motores, detector Sharp, detector de infrarrojo TSOP, fuentes de alimentación y todo el cableado hacia el microcontrolador. Una vista de esta baquelita se muestra en la Figura 23

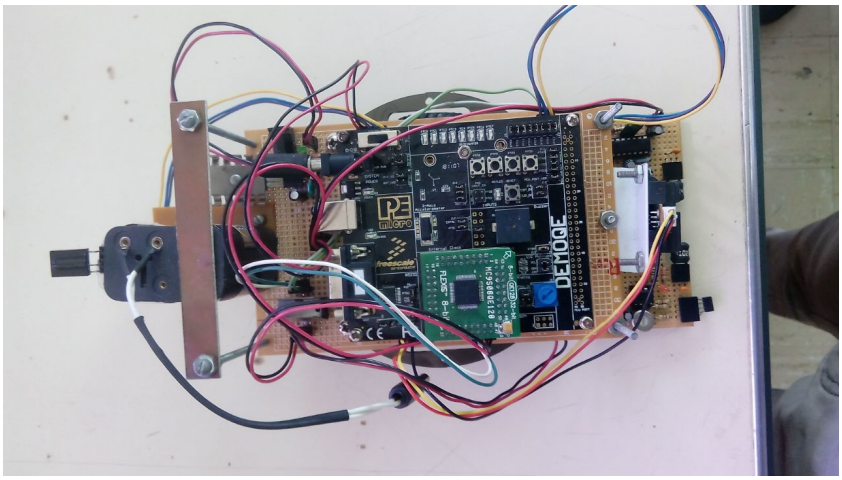

Figura 23: Baquelita del microcontrolador.

El enfoque del diseño en baquelita fue del tipo "Plug&Play" en el que fuera fácilmente accesible cada módulo por separado, lo que permitió hacer depuración tanto del hardware como del software de forma segura y cómoda.

4. Software

Esta sección presenta la arquitectura de software del sistema autónomo. Esta arquitectura es implementada con tres objetivos específicos:

- Estimar posición del vehículo.
- Planificar camino hasta la meta.
- Comunicar el computador con el vehículo.

La funcionalidad de esta arquitectura implementada es detallada en la Section 4.

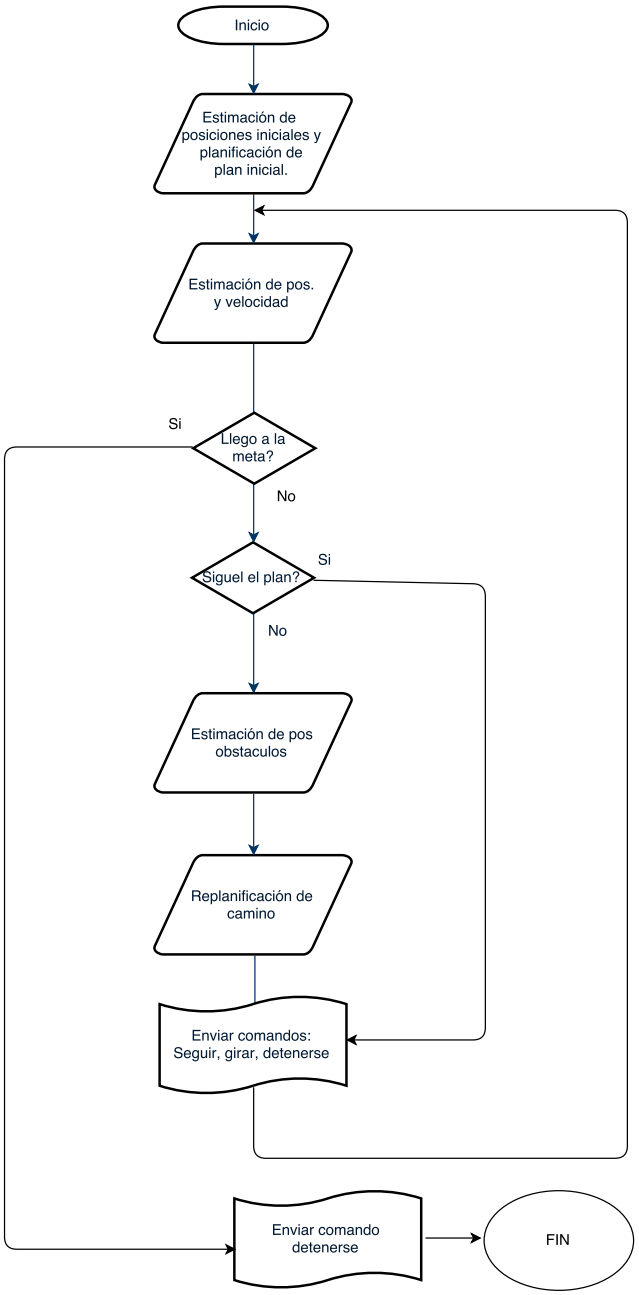

Figura 24: Arquitectura de Software

Una vez discretizado el espacio de trabajo del robot (en base a los cruces de la pista) se mapea cada intersección como nodo de un grafo, implementado en Python. Esta arquitectura de planificación es implementada en dos clases principales:

1. Robot: incluye las definiciones basicas del robot, su posición, su orientación y la ruta a seguir. Incluye métodos para estimar la posición y orientación del robot y definir la acción a tomar, en el archivo`robot.py`.

2. World: Establece el ambiente de trabajo como un grafo, en el archivo `world.py`.

3. Grafo: Define el elemento de nodo y las interconexiones e la pista, en el archivo `graph.py`.

A continuación, se presentan las especificaciones de la implementación de cada una de estas etapas.

4.1. Visión

La etapa de visión poro computadora se desarrolló en *Python* 3.5.0 bajo la librería OpenCV 3, dicha librería ofrece las siguientes funcionalidades de interés:

1. Interfaz con cámara de video.

2. Extracción de fondo de imagen.

3. Estimación de posición de objetos.

En particular, resultan de interés las funciones presentadas en la tabla Tabla 1

Tabla 1: Funciones de interés para la etapa de visión

`inRange`	Extrae los pixeles en un color de la imagen dada
`erode`	Elimina residuos de la extraccion de mascara del objeto
`dilate`	Dilata la mascara obtenida para reducción de ruido
`findContours`	calcula el contorno de las formas presentes en la máscara
`minEnclosingCircle`	Obtiene el radio y el centro del circulo presente en la máscara

Para estimar la posición del robot se comienza estimando mediante un sistema de visión por computadora, basado en OpenCV 3.0.0 para Python. El algoritmo principal realiza la estimación de la siguiente forma:

- Hacer captura de cámara.
- Obtener máscara de vehículo filtrando en el rango de colores del vehículo, aplicando erosión y dilatando la imagen.
- Hallar circunferencia de mayor radio en la captura.
- Tomar centro de circunferencia como ubicación del vehículo.
- Ubicar el vehículo en un nodo del grafo.

Una vez estimado el nodo en que se ubica el vehículo se estima su orientación, debido a que solo se usa un punto como referencia de posición, no es posible estimar la orientación real del vehículo. Por lo tanto se estima la orientación mediante los cruces realizados entre cada par de grafos, asumiendo la orientación inicial del vehículo como constante.

4.2. Planificación

La planificación implementada se basó en teoria de grafos, sobre una versión discretizada del espacio de trabajo. Para ello se implementó una librería de grafos sobre Python 3.0 con los siguientes objetivos:

1. Representación de caminos unidireccionales y bidireccionales.

2. Representación de heurísticas admisibles basadas en distancia.

3. Inhabilitación de caminos.

4. Planificación de caminos basada en búsquedas discretas con heurística.

Esta última etapa resulta de mayor interés para este proyecto, pues debe ser capaz de calcular el cámino mas corto entre dos puntos dados. Para esto se utilizó el algoritmo de busqueda $A*$, pues resulta en planes óptimos con poca exigencia computacional, como se muestra en la Figura 25

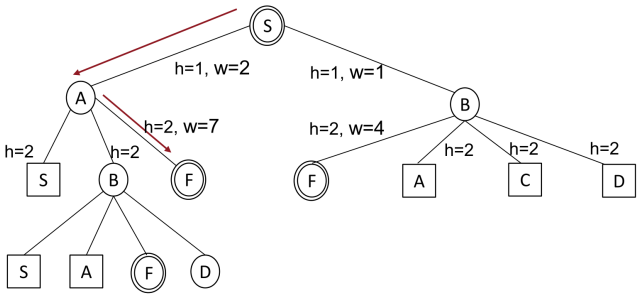

Figura 25: Interpretación gráfica del algoritmo A*

Una vez conocida la posición del vehículo es necesario planificar la trayectoria a seguir, operación que debe ser realizada a una tasa de tiempo real, tal que sea posible re-planificar cuando sea necesario. Debido a este requerimiento de velocidad y a la representación discreta del ambiente de decide hacer uso de algoritmo de busqueda de grafos como método de planificación de camino.

El algoritmo A* es implementado en base a una heuristica de distancia cartesiana entre la ubicación del robot y la meta, y se integra como parte de la clase `robot.py`.

4.3. Comunicación

La comunicación con el microcontrolador se realizó mediante protocolo serial, para ello se usó la librería `serial` incluida en Python 3.0 con un protocolo de comunicación binario de instrucciones simples. Como se muestra en la Figura 26, se utilizaron 3 elementos principales para realizar las comunicaciones dentro del proyecto. Se puede separar en dos etapas:

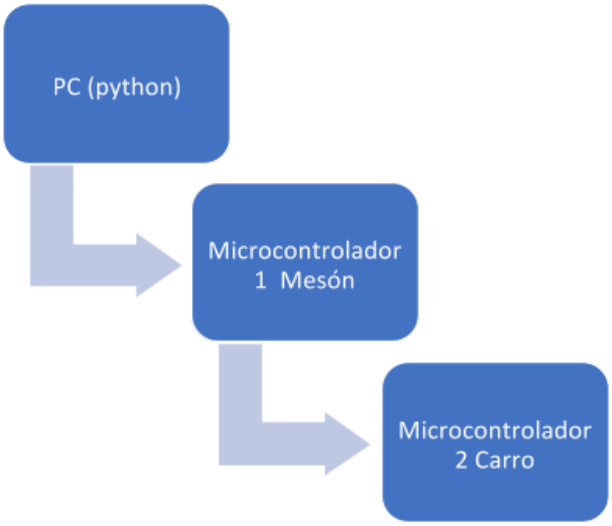

Figura 26: Diagrama de la comunicación

- **Comunicación PC-Microcontroldor1:** Esta se refiere al intercambio de información entre el programa en Python (etapa anterior) sobre la acción que debe realizar el carro de acuerdo a la pista y los obstáculos presentes y el microcontrolador del mesón. Esta comunicación es unidireccional desde el Pc al microcontrolador. Sucede a través del protocolo de comunicación serial. Esta etapa es la encargada de encapsular un paquete de datos generados por el programa en Python que contiene la dirección que debe seguir el carro y el identificador (opcional) y se envía utilizando un carácter de la siguiente forma:
  - "G" = acción adelante.
  - "S" = acción detenerse.
  - "I" = acción cruce izquierda.
  - "D" = acción cruce derecha.
  - "A" = acción retroceder.

Para evitar errores en el establecimiento de la comunicación, se decidió implementar un protocolo de 3 vías o conocido como "handshake". Consiste en enviar desde el PC un carácter (en nuestro caso C), donde se indica que el dispositivo está listo para enviar. Del otro lado, el microcontrolador está esperando recibir un C. Una vez que es recibida, se entera que puede iniciarse la comunicación y envía otro carácter, en nuestro caso C. Y el computador, tras enviar dicho carácter, queda a la espera del carácter de confirmación, conocido como ACK. Una vez la PC recibe la última C desde el microcontrolador se inicia la comunicación regular.

- **Comunicación Microcontrolador1-Microcontrolador2:** Este intercambio de datos se establece entre el microcontrolador del mesón y el microcontrolador ubicado en el carrito. La comunicación se establece a través de un protocolo propio utilizando la tecnología infrarroja (TSOP1136). Este dispositivo tiene características particulares de uso disponible en su hoja de datos, como recibir trenes de pulsos cuadrados de 36KHz, los cuales fueron aplicados al algoritmo de envío y recepción respectivamente. Para establecer la comunicación unidireccional con el sentido Microcontrolador 1 – Microcontrolador 2, se utilizan dos circuitos externos expuestos previamente. Los mensajes utilizados fueron enviados a través de 5 bits distribuidos de la siguiente forma:

  - "100000" = adelante.
  - "010000" = detenerse.
  - "001000" = cruce izquierda.
  - "000100" = cruce derecha.
  - "000001" = retroceder.

  Se utilizó esta codificación para poder detectar fácilmente los errores en la transmisión de los datos. El algoritmo implementado en el microcontrolador 2 es capaz de saber si efectivamente el mensaje recibido es un mensaje válido y en el caso contrario, descartarlo.

Por otro lado, se obtuvieron errores en cuanto a la sincronización del envío y recepción de datos a través de este medio, por lo que se decidió implementar un preámbulo a cada mensaje con el fin de brindar sincronización a la comunicación y obtener mensajes coherentes. El preámbulo utilizado fue "011". El microcontrolador al detectar esta secuencia de bits en un mensaje, interpreta que está en pleno proceso de recepción de un mensaje valido y almacena los próximos 5 bits. Posteriormente valida el mensaje recibido y toma la acción correspondiente.

La Figura 27 presenta las lecturas en osciloscopio del mensaje enviado y recibido.

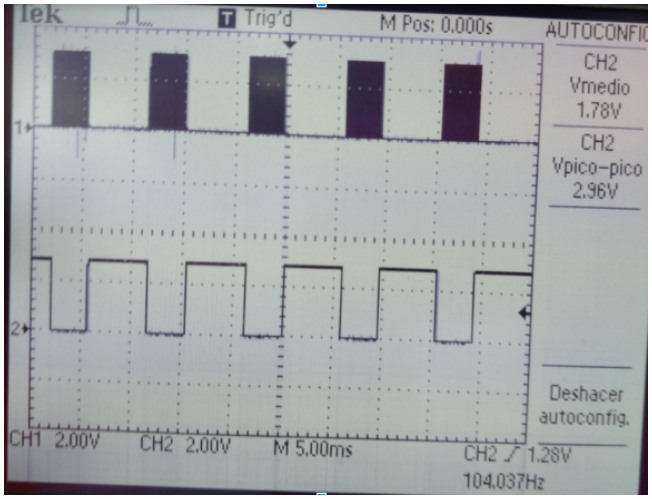

Figura 27: Señales de envío (parte superior) y recepción (parte inferior) infrarroja

Se aprecia en el canal 1, la salida del microcontrolador 1, donde se observan trenes de pulsos al enviar un 1, y bajos al enviar 0. Por otro lado, en el canal 2, se muestra la recepción de dichos pulsos en altos de 2.96 V en el caso de haber enviado un 0 lógico o un bajo de 0 V en caso de haber enviado un 1 lógico.

Toda comunicación tarda un tiempo en transmitir el mensaje, lo cual se traduce en un retraso entre la transmisión y la recepción. A continuación, en la Figura 28 se muestra la medición de dicho retraso que, para la aplicación utilizada, no representa un inconveniente. El retraso es de 102 microsegundos.

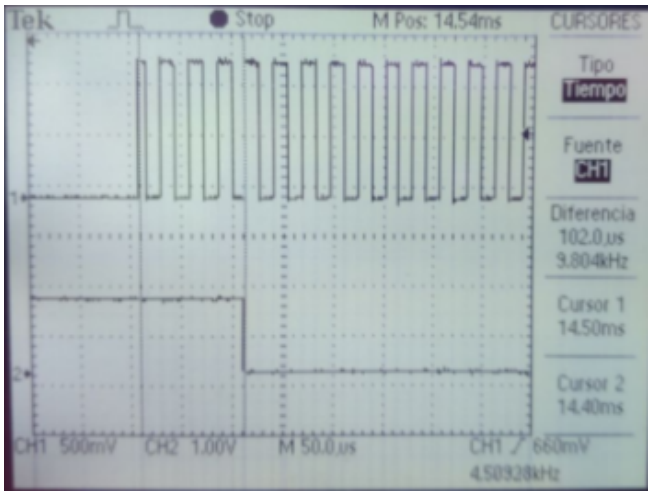

Figura 28: Retardo del mensaje recibido respecto al mensaje enviado

4.4. *Firmware* del Vehículo Autoguíado

Se utilizó el microcontrolador *HCS08QE128* de *NXP Semiconductors* y el módulo de desarrollo *DemoQE128* del mismo fabricante. Se programó utilizando el IDE *CodeWarrior 10.6*.

El *firmware* del vehículo debe integrar los distintos módulos ya detallados y generar una solución eficiente para proveer de tiempos de ejecución aceptables. Se utilizó una máquina de estados e interrupciones con temporizadores para iniciar su ejecución tras cada nuevo ciclo y manejar la comunicación con el sistema de visión por computadora.

Se implementaron 6 estados: ESTADO0, LEER_BOBINAS, CONTROL_PWM, LEER_DISTANCIA, SENAL_CRUCE y TRANSMISION_SERIAL. El primero de ellos es necesario para no permitir que la ejecución continúe. Se fija, utilizando la interrupción respectiva, la velocidad de la máquina en 50 ciclos por segundo. EL segundo adquiere el valor de las bobinas utilizando los módulos de conversión ADC de los cuales dispone el μC. El tercero se encarga de establecer las variables de control y asignar los anchos de pulsos respectivos de la señal PWM que va hacia los motores, dependiendo del modo en el cual se encuentre el AGV. Se disponen de 5 primitivas o *modos*: ADELANTE, REVERSA, TURN_RIGHT, TURN_LEFT y DETENER. La Figura 29 describe el cambio entre cada una de estas.

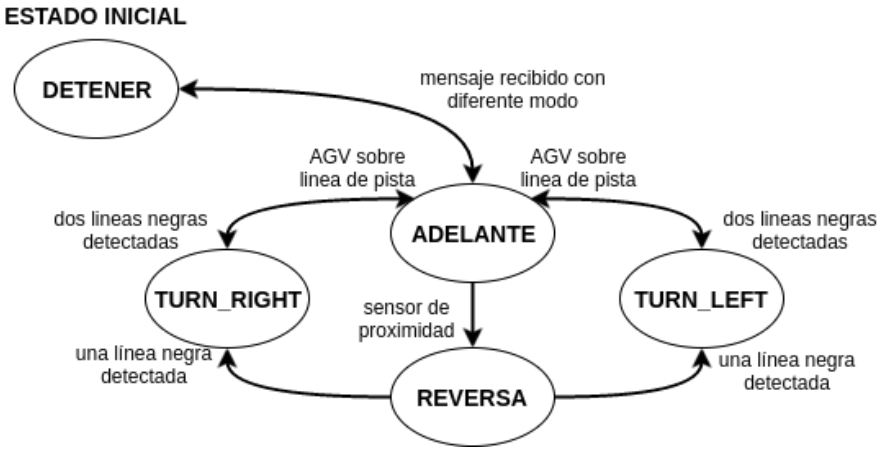

Figura 29: Modos de ejecución del AGV

El cuarto estado se encarga de adquirir el valor del sensor de proximidad por infrarojo. Nuevamente se utiliza el módulo ADC y se implementa esta vez un filtro *Moving Average* de 4 muestras para promediar los valores

obtenidos. Dependiendo del valor registrado, en el rango 0 -255, se fija un valor de *threshold* (100) a partir del cual el objeto en frente del vehículo se encuentra a una distancia lo suficientemente cercana como para activar una acción en el AGV.

El quinto registra las líneas negras sobre cada cruce. Se genera una acción, si el AGV va hacia adelante, al detectar dos líneas consecutivas. Para el vehículo en modo reversa, se requiere solamente detectar la primera de ellas. Se implementa además otro filtro *Moving Average* para evitar falsos positivos durante el recorrido sobre la pista.

El último estado, con propósitos de control de ejecución, transmite el valor de los anchos de pulsos PWM respectivos a través del terminal serial del módulo de desarrollo. La Figura 30 describe el comportamiento del *firmware* a través de diagramas de flujo.

Finalmente, la Figura 31 describe el proceso de recepción de mensajes. Se establece un preámbulo válido, en el receptor, de 3 bits (100). A partir de la recepción de estos valores, se continua con los siguiente 5 bits que componen el mensaje respectivo. Se definen 5 tipos de mensajes, uno para cada primitiva del AGV.

Con propósitos de verificación, el apéndice A contiene los códigos de programación utilizados en el vehículo. Adicionalmente, en los siguientes enlaces se logran observar las distintas pruebas realizadas:

- Recorrido libre:
  - `https://www.youtube.com/watch?v=pCBhxK1X9sM`
  - `https://www.youtube.com/watch?v=Kj2bGL5unXg`
- Recorrido con obstáculos:
  - `https://www.youtube.com/watch?v=-YINWiGF6Xo`
  - `https://www.youtube.com/watch?v=RY47U0YADUw`

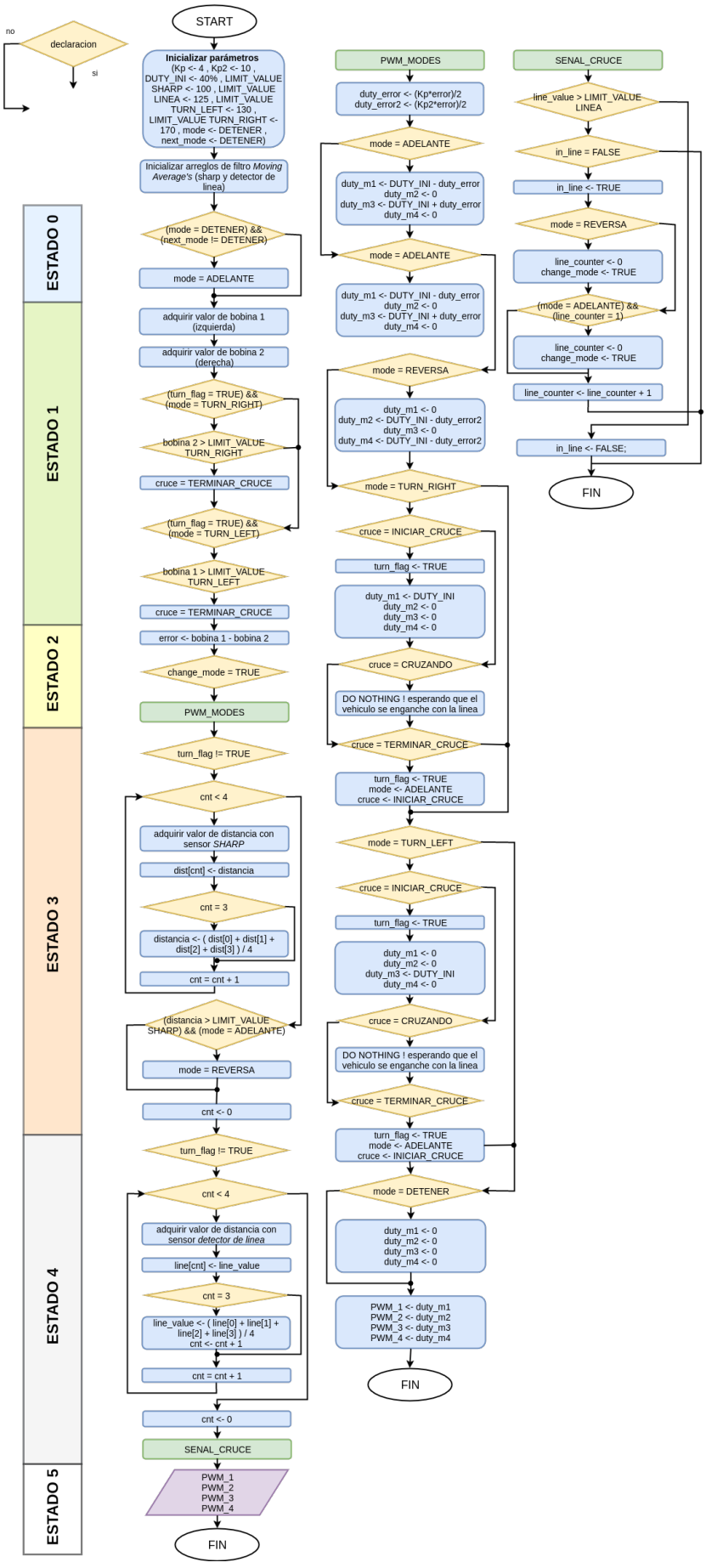

Figura 30: Diagrama de flujo que explica el *Firmware* del Vehículo Autoguiado

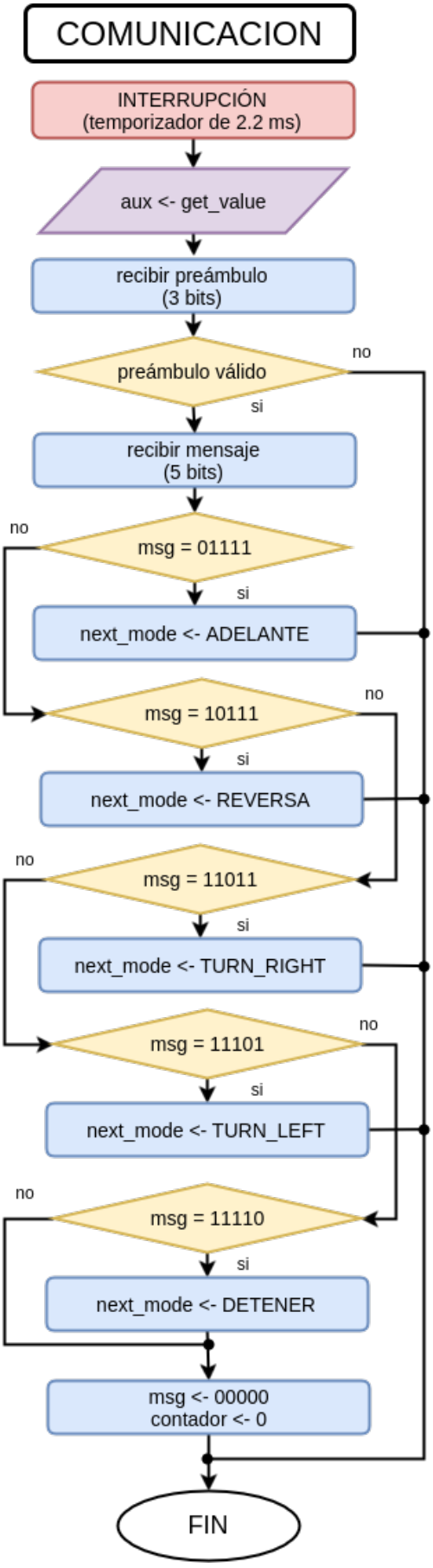

Figura 31: Diagrama de flujo que explica el *Firmware* del Vehículo Autoguiado

4.5. Consideraciones finales

El enfoque de diseño de software se basabas en programación de objetos, generando clases especificas para el robot, el espacio de trabajo y el sistema de comunicación, resultando en un funcionamiento satisfactorio en todos los escenarios de diseño requeridos. Sin embargo, se pueden plantear una serie de consideraciones para implementaciones futuras, especificamente en las siguientes etapas:

- **Visión por computador**: en un futuro, podria ser de mayor utilidad detectar tanto la orientación como la posición del vehiculo. Tal avance requeriría la asignación de un color extra al robot o el uso de una forma conocida para la mascara de robot, de forma que se pueda estimar la orientación del vehiculo, permitiendo el inicio del movimiento desde posiciones y orientaciones arbitrarias.
- **Planificación de camino**: La planificación de trayectorias se realizo de forma eficiente mediante al algoritmo A*. Sin embargo, es posible que en un futuro tambien se empleen métodos de planificación en espacios continuos, como lo son los campos de potencial y la optimización de trayectorias, lo que permitiría considerar restricciones adicionales en el problema de planificación, como los obstaculos y el movimiento no restringido a una linea.
- **Comunicación**: El sistema de comunicación basado en infrarrojo resulto satisfactorio para el entorno de trabaja empleado. Sin embargo, este tipo de comunicación cuenta con la desventaja de ser muy susceptible a las interferencias externas, generando errores cuando multiples fuentes se encuentran presentes en el ambiente de trabajo. Por lo tanto en un futuro podría resultar de interés el hacer uso de protocolos alternos de comunciación, como Wi-Fi ó Bluetooth, pues son de mayor confianza a la hora de trasmitir paquetes de datos y garantizan mayor robustez ante interferencias en el ambiente.

Apéndice A

Robot.py

```
from collections import deque
import numpy as np
import cv2
import serial

class robot(object):

    def __init__(self):
        self.id = 0xFF
        self.pos = [0,0]
        self.vel = [0,0]
        self.node = '1'
        self.id = 'L'
        # estados = "STOP", "ATRAS","DER", "IZQ", "GO"
        self.cond = 0
        self.state = "STOP"
        self.dt = 0.1
      self.lb = (10, 100, 100)
      self.ub = (30, 255, 255)

    def getpos(self,camera):
        # obtiene la imagen del espacio de trabajo
        (grabbed, frame) = camera.read()
        frame = cv2.resize(frame, (500, 500))
        ancho = 500
        largo = 500
        #frame = imutils.resize(frame, width=600)
        blurred = cv2.GaussianBlur(frame, (11, 11), 0)
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

        # calcula la mascara del vehiculo
        mask = cv2.inRange(hsv, self.lb, self.ub)
        mask = cv2.erode(mask, None, iterations=2)
        mask = cv2.dilate(mask, None, iterations=2)

        res = cv2.bitwise_and(frame,frame,mask = mask)

        # obtiene los contornos de la mascara
        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[-2]
        x = 0
        y = 0
        # calcula el centro del circulo
        if len(cnts) > 0:
            # halla el maximo circulo en la mascara
            c = max(cnts, key=cv2.contourArea)
            ((x, y), radius) = cv2.minEnclosingCircle(c)

        # calcula la velocidad del vehiculo
        dx = x - self.pos[0]
        dy = y - self.pos[1]
        self.vel = [dx/self.dt, dy/self.dt]

        # actualiza la posicion del vehiculo
        self.pos = [x, y]

        self.node = 0

        #NODO 1
        if (self.pos[0] >= ((0*ancho)/4)) and (self.pos[0] <= ((1*ancho)/4)) and (self.pos[1]
    >= ((0*largo)/3)) and (self.pos[1] <= ((1*largo)/3)) :
            self.node = "1"

         #NODO 8
        if (self.pos[0] >= ((0*ancho)/4)) and (self.pos[0] <= ((1*ancho)/4)) and (self.pos[1]
    >= ((1*largo)/3)) and (self.pos[1] <= ((2*largo)/3)) :
            self.node = "8"

         #NODO 9
        if (self.pos[0] >= ((0*ancho)/4)) and (self.pos[0] <= ((1*ancho)/4)) and (self.pos[1]
    >= ((2*largo)/3)) and (self.pos[1] <= ((3*largo)/3)) :
```

```
68              self.node = "9"
69
70          #NODO 2
71          if (self.pos[0] >= ((1*ancho)/4)) and (self.pos[0] <= ((2*ancho)/4)) and (self.pos[1]
    >= ((0*largo)/3)) and (self.pos[1] <= ((1*largo)/3)) :
72              self.node = "2"
73
74          #NODO 7
75          if (self.pos[0] >= ((1*ancho)/4)) and (self.pos[0] <= ((2*ancho)/4)) and (self.pos[1]
    >= ((1*largo)/3)) and (self.pos[1] <= ((2*largo)/3)) :
76              self.node = "7"
77
78          #NODO 10
79          if (self.pos[0] >= ((1*ancho)/4)) and (self.pos[0] <= ((2*ancho)/4)) and (self.pos[1]
    >= ((2*largo)/3)) and (self.pos[1] <= ((3*largo)/3)) :
80              self.node = "10"
81
82          #NODO 3
83          if (self.pos[0] >= ((2*ancho)/4)) and (self.pos[0] <= ((3*ancho)/4)) and (self.pos[1]
    >= ((0*largo)/3)) and (self.pos[1] <= ((1*largo)/3)) :
84              self.node = "3"
85
86          #NODO 6
87          if (self.pos[0] >= ((2*ancho)/4)) and (self.pos[0] <= ((3*ancho)/4)) and (self.pos[1]
    >= ((1*largo)/3)) and (self.pos[1] <= ((2*largo)/3)) :
88              self.node = "6"
89
90          #NODO 11
91          if (self.pos[0] >= ((2*ancho)/4)) and (self.pos[0] <= ((3*ancho)/4)) and (self.pos[1]
    >= ((2*largo)/3)) and (self.pos[1] <= ((3*largo)/3)) :
92              self.node = "11"
93
94          #NODO 4
95          if (self.pos[0] >= ((3*ancho)/4)) and (self.pos[0] <= ((4*ancho)/4)) and (self.pos[1]
    >= ((0*largo)/3)) and (self.pos[1] <= ((1*largo)/3)) :
96              self.node = "4"
97
98          #NODO 5
99          if (self.pos[0] >= ((3*ancho)/4)) and (self.pos[0] <= ((4*ancho)/4)) and (self.pos[1]
    >= ((1*largo)/3)) and (self.pos[1] <= ((2*largo)/3)) :
100             self.node = "5"
101
102         #NODO 12
103         if (self.pos[0] >= ((3*ancho)/4)) and (self.pos[0] <= ((4*ancho)/4)) and (self.pos[1]
    >= ((2*largo)/3)) and (self.pos[1] <= ((3*largo)/3)) :
104             self.node = "12"
105
106         #Show frame
107         cv2.rectangle(res,(((0*ancho)/4),((0*largo)/3)),(((1*ancho)/4),((1*largo)/3))
    ,(0,255,0),2)
108         cv2.rectangle(res,(((1*ancho)/4),((0*largo)/3)),(((2*ancho)/4),((1*largo)/3))
    ,(0,255,0),2)
109         cv2.rectangle(res,(((2*ancho)/4),((0*largo)/3)),(((3*ancho)/4),((1*largo)/3))
    ,(0,255,0),2)
110         cv2.rectangle(res,(((3*ancho)/4),((0*largo)/3)),(((4*ancho)/4),((1*largo)/3))
    ,(0,255,0),2)
111         cv2.rectangle(res,(((0*ancho)/4),((1*largo)/3)),(((1*ancho)/4),((2*largo)/3))
    ,(0,255,0),2)
112         cv2.rectangle(res,(((1*ancho)/4),((1*largo)/3)),(((2*ancho)/4),((2*largo)/3))
    ,(0,255,0),2)
113         cv2.rectangle(res,(((2*ancho)/4),((1*largo)/3)),(((3*ancho)/4),((2*largo)/3))
    ,(0,255,0),2)
114         cv2.rectangle(res,(((3*ancho)/4),((1*largo)/3)),(((4*ancho)/4),((2*largo)/3))
    ,(0,255,0),2)
115         cv2.rectangle(res,(((0*ancho)/4),((2*largo)/3)),(((1*ancho)/4),((3*largo)/3))
    ,(0,255,0),2)
116         cv2.rectangle(res,(((1*ancho)/4),((2*largo)/3)),(((2*ancho)/4),((3*largo)/3))
    ,(0,255,0),2)
117         cv2.rectangle(res,(((2*ancho)/4),((2*largo)/3)),(((3*ancho)/4),((3*largo)/3))
    ,(0,255,0),2)
118         cv2.rectangle(res,(((3*ancho)/4),((2*largo)/3)),(((4*ancho)/4),((3*largo)/3))
    ,(0,255,0),2)
119
120         #Show frame
121         cv2.imshow('res',res)
122
```

```
123     def checkstate(self, next_node, goal):
124         # detecta la accion a tomar
125         if self.pos == [0,0]:
126             self.state = "STOP"
127         elif self.node == goal:
128             self.state = "STOP"
129         elif self.node == "1":
130             if next_node == "2":
131                 self.state = "GO"
132             elif next_node == "8":
133                 self.state = "DER"
134         elif self.node == "2":
135             if next_node == "1":
136                 self.state = "ATRAS"
137             elif next_node == "7":
138                 self.state = "DER"
139             elif next_node == "3":
140                 self.state = "GO"
141         elif self.node == "3":
142             if next_node == "2":
143                 self.state = "ATRAS"
144             elif next_node == "6":
145                 self.state = "DER"
146             elif next_node == "4":
147                 self.state = "GO"
148         elif self.node == "4":
149             if next_node == "5":
150                 self.state = "DER"
151             elif next_node == "3":
152                 self.state = "ATRAS"
153         elif self.node == "5":
154             if next_node == "4":
155                 self.state = "IZQ"
156             elif next_node == "6":
157                 self.state = "ATRAS"
158             elif next_node == "12":
159                 self.state = "DER"
160         elif self.node == "6":
161             if next_node == "5":
162                 self.state = "GO"
163             elif next_node == "3":
164                 self.state = "IZQ"
165             elif next_node == "11":
166                 self.state = "DER"
167             elif next_node == "7":
168                 self.state = "ATRAS"
169         elif self.node == "7":
170             if next_node == "6":
171                 self.state = "GO"
172             elif next_node == "2":
173                 self.state = "IZQ"
174             elif next_node == "10":
175                 self.state = "DER"
176             elif next_node == "8":
177                 self.state = "ATRAS"
178         elif self.node == "8":
179             if next_node == "7":
180                 self.state = "GO"
181             elif next_node == "1":
182                 self.state = "IZQ"
183             elif next_node == "9":
184                 self.state = "DER"
185         elif self.node == "9":
186             if next_node == "10":
187                 self.state = "GO"
188             elif next_node == "8":
189                 self.state = "IZQ"
190         elif self.node == "10":
191             if next_node == "11":
192                 self.state = "GO"
193             elif next_node == "7":
194                 self.state = "IZQ"
195             elif next_node == "9":
196                 self.state = "ATRAS"
197         elif self.node == "11":
198             if next_node == "12":
```

```
                self.state = "GO"
            elif next_node == "6":
                self.state = "IZQ"
            elif next_node == "10":
                self.state = "ATRAS"
        elif self.node == "12":
            if next_node == "5":
                self.state = "IZQ"
            elif next_node == "11":
                self.state = "ATRAS"

    def send_msg(self, port):
        # chequea la tasa de baudios

        if self.state == "STOP":
            msg = "S"
        elif self.state == "ATRAS":
            msg = "A"
        elif self.state == "DER":
            msg = "D"
        elif self.state == "IZQ":
            msg = "I"
        elif self.state == "GO":
            msg = "G"
        else:
            msg = "S"

        # incializa la conexion
        if self.cond == 0:
            # envia la se\nal de sincronizacion
            port.write("A")
            if port.in_waiting > 0:
                # espera el acknowledge
                ack = port.read();
                print "Recibi respuesta " + ack
                if ack == "C":
                    port.write(self.id)
                    port.write(msg)
                    print "ACK"
                    print msg
                    self.cond = 1
            else:
                # no se detecta el acknowledge
                print "no conectado"
        else:
            # envia el mensaje por la conexion inicializada
            port.write(self.id)
            port.write(msg)
            port.flush()

            msg_rec = port.read()
            msg_rec2 = port.read()
            print "conectado"

            print msg_rec + msg_rec2
```

World.py

```
from robot import robot
from SimpleGraph import SimpleGraph
from PriorityQueue import PriorityQueue
from Grid import Grid

class world(object):
  def __init__(self):
    # inicializa la clase de mundo
    self.graph = SimpleGraph()
    self.graph.edges = {'1' : ['2','8'],
              '2' : ['1','3','7'],
              '3' : ['2','4','6'],
              '4' : ['3','5'],
              '5' : ['4','6','12'],
              '6' : ['3','5','7','11'],
```

```
            '7' : ['2','6', '8', '10'],
            '8' : ['1','7', '9'],
            '9' : ['8','10'],
            '10' : ['7','9', '11'],
            '11' : ['6','10', '12'],
            '12' : ['5','11']}
    self.posiciones = {'1' : (1,1),
            '2' : (1,2),
            '3' : (1,3),
            '4' : (1,4),
            '5' : (2,1),
            '6' : (2,2),
            '7' : (2,3),
            '8' : (2,4),
            '9' : (3,1),
            '10' : (3,2),
            '11' : (3,3),
            '12' : (3,4)}
    self.goal = '5'
    self.start = '1'
    self.plan = []

  def getNode(self, pt):
    # retorna el indice del nodo donde se ubica el vehiculo
    pass

  def cost(self, a, b):
    # define una heuristica para la busqueda informada
    (x1, y1) = a
    (x2, y2) = b
    return abs(x1 - x2) + abs(y1 - y2)

  def a_star(self):
    # Genera camino usando el algoritmo A*
    frontier = PriorityQueue()
    frontier.put(self.start, 0)
    came_from = {}
    cost_so_far = {}
    came_from[self.start] = None
    cost_so_far[self.start] = 0

    # comienza la busqueda
    while not frontier.empty():
      current = frontier.get()
      # termina si llega a la meta
      if current == self.goal:
        break
      # busca en sus vecinos
      for next in self.graph.neighbors(current):
        new_cost = cost_so_far[current] + self.graph.cost(current, next)
        if next not in cost_so_far or new_cost < cost_so_far[next]:
          cost_so_far[next] = new_cost
          priority = new_cost + self.cost(self.posiciones[self.goal], self.posiciones[next])
          frontier.put(next, priority)
          came_from[next] = current
    # ordena el plan
    current = self.goal
    path = [current]
    while current != self.start:
      current = came_from[current]
      path.append(current)
    # invierte el camino para empezar al inicio
    path.reverse()
    self.plan = path
```

Graph.py

```
class SimpleGraph:
  def __init__(self):
    self.edges = {}

  def neighbors(self, id):
    return self.edges[id]
```

```
7
8   def cost(self, a, b):
9     return 1
```

main.c

```
1  #include "Cpu.h"
2  #include "Events.h"
3  #include "TI1.h"
4  #include "AD1.h"
5  #include "AS1.h"
6  #include "PWM1.h"
7  #include "PWM2.h"
8  #include "PWM3.h"
9  #include "PWM4.h"
10 #include "TI2.h"
11 #include "Bit1.h"
12 #include "Bit2.h"
13 #include "Bit3.h"
14
15 #include "PE_Types.h"
16 #include "PE_Error.h"
17 #include "PE_Const.h"
18 #include "IO_Map.h"
19
20 // CONTROLADOR
21 #define KP 4                    // ganancia proporcional
22 #define KI 0                    // ganancia integral
23 #define INTEGRAL_MAX  1         // limite superior de la integral del error
24 #define INTEGRAL_MIN -1         // limite inferior de la integral del error
25
26 // SOME DEFINITIONS
27 #define DUTY_INI_M1             400
28 #define DUTY_INI_M2             400
29 #define LIMIT_VALUE_DIST        100
30 #define LIMIT_VALUE_LINE        125
31 #define LIMIT_VALUE_TURN_RIGHT  130
32 #define LIMIT_VALUE_TURN_LEFT   170
33
34 unsigned char estado = 0;
35
36 // BOBINAS
37 unsigned int bobina_1, bobina_2;
38
39 // CONTROL PWM
40 signed int error = 0, DT = 0.03, integral_error = 0; // <-- ** TIENEN QUE SER FLOAT **
41 unsigned int duty_m1 = 0, duty_m2 = 0, duty_m3 = 0, duty_m4 = 0;
42 unsigned char mode = DETENER;
43 unsigned char next_mode = DETENER;
44 unsigned char cruce = INICIAR_CRUCE;
45
46 // SENSOR DE DISTANCIA
47 unsigned char distance = 0, dist_fir[5], cnt = 0;
48
49 // COMUNICACION IR
50 bool ir_flag = FALSE; // TRUE == non-zero value ; FALSE == 0
51
52 // SENAL DE CRUCE
53 unsigned char line_value = 0, line_value_fir[5], line_counter = 0;
54 bool in_line = FALSE; // TRUE == non-zero value ; FALSE == 0
55 bool turn_flag = FALSE;
56 bool change_mode = FALSE;
57
58 // TRANSMISION SERIAL
59 word snd = 0;
60 unsigned char j = 0;
61
62 void main(void){
63   for(cnt = 0; cnt < 4; cnt++){
64     dist_fir[cnt] = 0;
65   }
66   cnt = 0;
67   for(cnt = 0; cnt < 4; cnt++){
```

```
68      line_value_fir[cnt] = 0;
69    }
70    cnt = 0;
71    /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
72    PE_low_level_init();
73    /*** End of Processor Expert internal initialization.                    ***/
74    for(;;)
75    {
76
77      /*
78       *  CICLO DE LA MAQUINA DE ESTADOS
79       *  (0) DO NOTHING
80       *  (1) LEE SENAL ELECTROMAGNETICA DE LAS BOBINAS
81       *  (2) CONTROL PWM DE LOS MOTORES
82       *  (3) LEE DISTANCIA DEL SENSOR SHARP
83       *  (4) EN CASO DE DETECTAR SENAL DE CRUCE
84       *  (5) TRANSMISION SERIAL PARA DEBUGGING
85       */
86
87        switch(estado){
88            case ESTADO0:
89            if((mode == DETENER) && (next_mode != DETENER)){
90              mode = ADELANTE;
91            }
92            // Hello World !
93            break;
94            case LEER_BOBINAS:
95            (void)AD1_MeasureChan(TRUE, 0x00); // REALIZA MEDICION CANAL_0
96            (void)AD1_GetChanValue16(0x00,&bobina_1); // OBTIENE VALOR BOBINA_1
97            bobina_1 = bobina_1 >> 8; // BIT SHIFTING
98            (void)AD1_MeasureChan(TRUE, 0x01); // REALIZA MEDICION CANAL_1
99            (void)AD1_GetChanValue8(0x01,&bobina_2); // OBTIENE VALOR BOBINA_2
100           bobina_2 = bobina_2 >> 8;  // BIT SHIFTING
101           if(turn_flag){
102             if(mode == TURN_RIGHT){
103               if(bobina_2 > LIMIT_VALUE_TURN_RIGHT){
104                 cruce = TERMINAR_CRUCE;
105               }
106             }
107             else if(mode == TURN_LEFT){
108               if(bobina_1 > LIMIT_VALUE_TURN_LEFT){
109                 cruce = TERMINAR_CRUCE;
110               }
111             }
112           }
113           estado = CONTROL_PWM; // CHANGE STATE
114           break;
115           case CONTROL_PWM:
116             error = bobina_1 - bobina_2; // RESTA DE SENAL EN BOBINAS
117
118 PWM_MODES:     if(change_mode){
119           mode = next_mode;
120           next_mode = DETENER;
121           change_mode = FALSE;
122           }
123             if(mode == ADELANTE){
124               duty_m1 = DUTY_INI_M1 - (KP*error + KI*integral_error)/2;
125               duty_m2 = 0;
126               duty_m3 = DUTY_INI_M2 + (KP*error + KI*integral_error)/2;
127               duty_m4 = 0;
128             }
129
130             if (mode == REVERSA) {
131               duty_m1 = 0;
132               duty_m2 = DUTY_INI_M1 + (KP*error + KI*integral_error)/2;
133               duty_m3 = 0;
134               duty_m4 = DUTY_INI_M2 - (KP*error + KI*integral_error)/2;
135             }
136
137             if (mode == TURN_RIGHT) {
138               switch(cruce){
139                 case INICIAR_CRUCE:
140                   turn_flag = TRUE;
141                   duty_m1 = DUTY_INI_M1;
142                   duty_m2 = 0;
143                   duty_m3 = 0;
```

```
144                     duty_m4 = 0;
145                     cruce = CRUZANDO;
146                     break;
147                   case CRUZANDO:
148                     // do nothing
149                     break;
150                   case TERMINAR_CRUCE:
151                     turn_flag = FALSE;
152                     mode = ADELANTE;
153                     cruce = INICIAR_CRUCE;
154                     goto PWM_MODES;
155                     break;
156                 }
157               }
158
159               if (mode == TURN_LEFT) {
160                 switch(cruce){
161                   case INICIAR_CRUCE:
162                     turn_flag = TRUE;
163                     duty_m1 = 0;
164                     duty_m2 = 0;
165                     duty_m3 = DUTY_INI_M2;
166                     duty_m4 = 0;
167                     cruce = CRUZANDO;
168                     break;
169                   case CRUZANDO:
170                     // do nothing
171                     break;
172                   case TERMINAR_CRUCE:
173                     turn_flag = FALSE;
174                     mode = ADELANTE;
175                     cruce = INICIAR_CRUCE;
176                     goto PWM_MODES;
177                     break;
178                 }
179               }
180
181               if (mode == DETENER) {
182                 duty_m1 = 0;
183                 duty_m2 = 0;
184                 duty_m3 = 0;
185                 duty_m4 = 0;
186               }
187
188             (void)PWM1_SetDutyUS(duty_m1);
189             (void)PWM2_SetDutyUS(duty_m2);
190             (void)PWM3_SetDutyUS(duty_m3);
191             (void)PWM4_SetDutyUS(duty_m4);
192             estado = LEER_DISTANCIA; // CHANGE STATE
193             break;
194             case LEER_DISTANCIA:
195             if(!turn_flag){
196               while(cnt < 4){ // MOVING AVERAGE FILTER
197                 (void)AD1_MeasureChan(TRUE, 0x02);
198                 (void)AD1_GetChanValue8(0x02,&distance); // DISTANCE ADQUISITION
199                 dist_fir[cnt] = distance;
200                 if(cnt == 3){ // AVERAGE VALUE OF 4 MEASURES
201                   distance = (dist_fir[0]+dist_fir[1]+dist_fir[2]+dist_fir[3])/4;
202                 }
203                 cnt++; // SE AUMENTA EL CONTADOR
204               }
205               if((distance > LIMIT_VALUE_DIST) && (mode == ADELANTE)){
206                 mode = REVERSA; // OBJECT DETECTED ! STOP VEHICLE AND WAIT
207               }
208               cnt = 0; // SE REINICIA EL CONTADOR
209             }
210             estado = SENAL_CRUCE; // CHANGE STATE
211             break;
212             case SENAL_CRUCE:
213             if(!turn_flag){
214               while(cnt < 4){ // MOVING AVERAGE FILTER
215                 (void)AD1_MeasureChan(TRUE, 0x03);
216                 (void)AD1_GetChanValue8(0x03, &line_value); // DISTANCE ADQUISITION
217                 line_value_fir[cnt] = line_value;
218                 if(cnt == 3){ // AVERAGE VALUE OF 4 MEASURES
219                   line_value = (line_value_fir[0]+line_value_fir[1]+line_value_fir[2]+
```

```
    line_value_fir[3])/4;
220             }
221             cnt++; // SE AUMENTA EL CONTADOR
222           }
223           cnt = 0; // SE REINICIA EL CONTADOR
224           if(line_value > LIMIT_VALUE_LINE){
225             if(!in_line){
226               in_line = TRUE;
227               if(mode == REVERSA){
228                line_counter = 0;
229                change_mode = TRUE;
230                //goto PWM_MODES;
231               }
232               else if((mode == ADELANTE) && (line_counter == 1)){
233             line_counter = 0;
234             change_mode = TRUE;
235             //goto PWM_MODES;
236               }
237               else{
238               line_counter++;
239               }
240             }
241           } else{
242             in_line = FALSE;
243           }
244         }
245         estado = TRANSMISION_SERIAL; // CHANGE STATE
246         break;
247         case TRANSMISION_SERIAL:
248         snd = 0; // VARIABLE AUXILIAR
249       /*(void)AS1_SendBlock(&bobina_1,2,&snd); // VALOR BOBINA #1
250         snd = 0;
251         (void)AS1_SendBlock(&bobina_2,2,&snd); // VALOR BOBINA #2
252         snd = 0;
253         (void)AS1_SendBlock(&error,2,&snd); // VALOR DUTY CYCLE (us)
254         snd = 0;*/
255         (void)AS1_SendBlock(&duty_m1,2, &snd); // VALOR DUTY CYCLE (us)
256         snd = 0;
257         (void)AS1_SendBlock(&duty_m2,2, &snd); // VALOR DUTY CYCLE (us)
258         snd = 0;
259         (void)AS1_SendBlock(&duty_m3,2, &snd); // VALOR DUTY CYCLE (us)
260       snd = 0;
261         (void)AS1_SendBlock(&duty_m4,2, &snd); // VALOR DUTY CYCLE (us)
262         estado = ESTADO0; // CHANGE STATE
263         break;
264         default :
265         break;
266     }
267   }
268  #ifdef PEX_RTOS_START
269    PEX_RTOS_START();                  /* Startup of the selected RTOS. Macro is defined by
    the RTOS component. */
270  #endif
271 }
```

events.c

```
1  #include "Cpu.h"
2  #include "Events.h"
3
4  extern unsigned char estado; // variable de la maquina de estados
5  extern unsigned char next_mode;
6  unsigned char contador = 0;
7  bool msg[5];
8  bool aux;
9  bool aux1;
10
11 /*
12  ** ===================================================================
13  **     Event       :  TI1_OnInterrupt (module Events)
14  **
15  **     Component   :  TI1 [TimerInt]
16  **     Description :
```

```
17  **         When a timer interrupt occurs this event is called (only
18  **         when the component is enabled - <Enable> and the events are
19  **         enabled - <EnableEvent>). This event is enabled only if a
20  **         <interrupt service/event> is enabled.
21  **     Parameters  : None
22  **     Returns     : Nothing
23  ** ===================================================================
24  */
25 void TI1_OnInterrupt(void) {
26   if (estado == ESTADO0) {
27     estado = estado++;
28   }
29 }
30
31 /*
32  ** ===================================================================
33  **     Event       :  TI2_OnInterrupt (module Events)
34  **
35  **     Component   :  TI2 [TimerInt]
36  **     Description :
37  **         When a timer interrupt occurs this event is called (only
38  **         when the component is enabled - <Enable> and the events are
39  **         enabled - <EnableEvent>). This event is enabled only if a
40  **         <interrupt service/event> is enabled.
41  **     Parameters  : None
42  **     Returns     : Nothing
43  ** ===================================================================
44  */
45 void TI2_OnInterrupt(void) {
46   aux1 = Bit1_GetVal();
47   aux = aux1;
48   aux1=0;
49   //
50   if ((contador == 0) && (aux)){
51     contador = 1;
52   } else {
53     if ((contador == 1) && (!aux)){
54       contador = 2;
55     } else {
56       if ((contador == 2) && (!aux)){
57         contador = 3;
58       }else {
59         if (contador < 3){
60           contador = 0;
61         }else {
62           if ((contador > 2) && (contador < 8)) {
63               msg[contador - 3] = aux;
64               contador++;
65           }
66         }
67       }
68     }
69   }
70
71   if (contador == 8) {
72       Bit3_NegVal();
73       if ((!msg[0]) && (msg[1]) && (msg[2]) && (msg[3]) && (msg[4])) {
74         next_mode = ADELANTE;
75         Bit2_NegVal();
76       }
77       if ((msg[0]) && (!msg[1]) && (msg[2]) && (msg[3]) && (msg[4])) {
78         next_mode = DETENER;
79         Bit2_NegVal();
80       }
81       if ((msg[0]) && (msg[1]) && (!msg[2]) && (msg[3]) && (msg[4])) {
82         next_mode = TURN_LEFT;
83         Bit2_NegVal();
84       }
85       if ((msg[0]) && (msg[1]) && (msg[2]) && (!msg[3]) && (msg[4])) {
86         next_mode = TURN_RIGHT;
87         Bit2_NegVal();
88       }
89       if ((msg[0]) && (msg[1]) && (msg[2]) && (msg[3]) && (!msg[4])) {
90         next_mode = REVERSA;
91         Bit2_NegVal();
92       }
```

```
93
94          msg[0] = FALSE;
95          msg[1] = FALSE;
96          msg[2] = FALSE;
97          msg[3] = FALSE;
98          msg[4] = FALSE;
99          contador = 0;
100     }
101 }
```

events.h

```
1  #ifndef __Events_H
2  #define __Events_H
3  /* MODULE Events */
4
5  #include "PE_Types.h"
6  #include "PE_Error.h"
7  #include "PE_Const.h"
8  #include "IO_Map.h"
9  #include "PE_Timer.h"
10 #include "TI1.h"
11 #include "AD1.h"
12 #include "AS1.h"
13 #include "PWM1.h"
14 #include "PWM2.h"
15 #include "PWM3.h"
16 #include "PWM4.h"
17 #include "TI2.h"
18 #include "Bit1.h"
19 #include "Bit2.h"
20 #include "Bit3.h"
21
22 // ESTADOS
23 #define ESTADO0 0
24 #define LEER_BOBINAS 1
25 #define CONTROL_PWM 2
26 #define LEER_DISTANCIA 3
27 #define SENAL_CRUCE 4
28 #define TRANSMISION_SERIAL 5
29
30 // MODOS DEL VEHICULO
31 #define ADELANTE  1
32 #define REVERSA   2
33 #define TURN_RIGHT  3
34 #define TURN_LEFT 4
35 #define DETENER   5
36
37 // ETAPAS DEL CRUCE
38 #define INICIAR_CRUCE 1
39 #define CRUZANDO    2
40 #define TERMINAR_CRUCE  3
41
42 //unsigned char next_mode = TURN_RIGHT; // configuracion inicial
43
44 void TI1_OnInterrupt(void);
45 /*
46 ** ===================================================================
47 **     Event       :  TI1_OnInterrupt (module Events)
48 **
49 **     Component   :  TI1 [TimerInt]
50 **     Description :
51 **         When a timer interrupt occurs this event is called (only
52 **         when the component is enabled - <Enable> and the events are
53 **         enabled - <EnableEvent>). This event is enabled only if a
54 **         <interrupt service/event> is enabled.
55 **     Parameters  : None
56 **     Returns     : Nothing
57 ** ===================================================================
58 */
59
60 void TI2_OnInterrupt(void);
61 /*
```

```
62 ** ===================================================================
63 **     Event       :  TI2_OnInterrupt (module Events)
64 **
65 **     Component   :  TI2 [TimerInt]
66 **     Description :
67 **         When a timer interrupt occurs this event is called (only
68 **         when the component is enabled - <Enable> and the events are
69 **         enabled - <EnableEvent>). This event is enabled only if a
70 **         <interrupt service/event> is enabled.
71 **     Parameters  : None
72 **     Returns     : Nothing
73 ** ===================================================================
74 */
75
76 /* END Events */
77 #endif /* __Events_H*/
```