UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática

Universitat Politecnica de Valencia

Desarrollo de un prototipo de vehículo autónomo semi-inteligente basado en Arduino

PROYECTO FINAL DE CARRERA

Ingeniería Informática

Autor: Tomás Navarro Cosme

Director: Joaquín Gracia Morán

Codirector: Juan Carlos Ruiz García

Septiembre de 2016

.

Abstract

This project presents the design and implementation on the Arduino platform of several basic algorythms related with some functionalities needed in the automotive autonomous navigation, like line tracking, light seeking and obstacle avoidance. The implementations have been done in two different prototipes, with dissimilar physical and movement characteristics, and with different kind of sensors, so each prototipe has need its own implementation of each functionality, requiring some times that the algorythm itself to be distinct.

Besides, these basic algorythms has been combined to obtain a working prototipe of a car with Adaptive Cruise Control, which interact in real-time with the other car in the same circuit.

Keywords: Arduino, ADAS, ACC, Adaptive Cruise Control, line tracking, obstacle avoidance, security, algorithm, automotive, automation

Resumen

En este PFC se presenta el diseño e implementación en la plataforma de Arduino de varios algoritmos básicos relacionados con algunas funcionalidades necesarias para la navegación autónoma de vehículos, tales como el seguimiento de trayecto, el guiado por luz (detección de luminosidad), y la evitación de obstáculos. Las implementaciones se han realizado en dos prototipos diferentes, con distintas características físicas y de movimiento, y con diferentes sensores, de tal forma que cada prototipo ha necesitado su propia implementación de cada funcionalidad, incluso a veces con algoritmo diferente.

Además, esos algoritmos básicos se han combinado para obtener un prototipo funcional de un coche con Control de velocidad adaptativo, el cual interactua en tiempo real con el otro vehículo en el mismo circuito.

Palabras clave: Arduino, ADAS, ACC, Control de velocidad adaptativo, seguimiento de trayecto, evitación de obstáculos, seguridad, algoritmo, automoción, automatización

Índice abreviado

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE LISTADOS DE CÓDIGO

CAPÍTULO 1

INTRODUCCIÓN

La industria de la automoción es un sector líder en el avance tecnológico, aunque eso sí, la aplicación de sus avances en la producción masiva de vehículos de consumo está muy limitada por el factor económico, dado que existe una fuerte competitividad global entre las grandes multinacionales. Esta competitividad, que esencialmente es económica, puede desglosarse en varios aspectos. Por una parte está el coste final del producto en sí mismo, pero también deben valorarse aspectos económicos tales como los derivados de la utilización del mismo, de la fiabilidad de los elementos comunes y del propio motor, así como los derivados del incremento de valor por las expectativas ofrecidas.

Por la tendencia actual de las preferencias del automovilista, se observa un considerable aumento por el interés por la facilidad del uso del automóvil, incluso el deseo de que éste sea lo más autónomo posible para descargar de fatiga y atención durante la conducción. Pero en la actualidad se está muy lejos de llegar a una independencia total del factor humano, tanto por el propio estado de la tecnología actual como por el grado de fiabilidad que éstos últimos avances proporcionen real y legalmente.

Así, el grado de fiabilidad es muy importante, por cuanto afecta al sobrecoste que supone las reparaciones durante el periodo de garantía, periodo que cada vez se extiende temporalmente más debido precisamente a la competitividad existente. Pero también lo es la parte correspondiente a la responsabilidad civil subsidiaria derivada al fabricante cuando se demuestra que un determinado accidente, o defecto, ha sido ocasionado por un fallo en el diseño, fabricación o funcionamiento normal o anormal del vehículo. Este último es el factor que cada vez está cobrando una mayor importancia, dada la tendencia existente a lograr vehículos cada vez más autónomos y que integren más componentes electrónicos internos de seguridad, a la vez que de intercomunicación con el exterior. Como ejemplo, véanse las (a veces sensacionalistas) noticias referentes a accidentes relacionados con vehículos con funciones medianamente autónomas, como el Autopilot de Tesla, y su impacto en la imagen de la empresa, en las consecuencias físicas que tales fallos pueden ocasionar, y en la discusión moral y legal sobre quién es el responsable último de esos percances.

Tan así es que, recientemente, en julio de 2016, el consorcio Auto-ISAC (*Automotive Information Sharing and Analysis Center*), que reúne a las principales industrias de automoción del mundo, publicó un importante informe con recomendaciones y reflexiones sobre la ciberseguridad en la automoción, denominado "*Automotive Cybersecurity Best Practices*" [8]. En él se establece que la ciberseguridad debe alcanzar una prioridad prominente en el sector tanto para los fabricantes como para los inversores, por la reper-

cusión social y económica de noticias y sucesos relacionados con fallos de seguridad o de mecanismos autónomos de navegación e intercomunicación.

Las "mejores prácticas" recomendadas abarcan aspectos tanto técnicos como organizacionales, tales como dirección estratégica, gestión de riesgos, seguridad por diseño, detección de amenazas (y vulnerabilidades), respuesta a los incidentes, formación interna, y colaboración con agentes externos relevantes. De todos ellos, a los efectos de este proyecto, resaltamos la importancia de la seguridad por diseño (basada en la "tolerancia a fallos" de los sistemas electrónicos), en la detección de amenazas y vulnerabilidades ("seguridad" en el sector informático), y la colaboración con agentes externos, donde explícitamente en el informe se destaca la importancia de la colaboración con el sector de investigación externa, principalmente, la realizada en ámbitos universitarios.

No es casualidad, sino fruto del seguimiento y reflexión sobre la tendencia del sector por parte de la comunidad investigadora de la UPV, que esos tres puntos esenciales formaran parte del proyecto nacional propuesto recientemente por la línea de investigación de Sistemas Tolerantes a Fallos (STF) del instituto ITACA de la UPV. Ese proyecto propuesto, denominado DINAMOS, plantea la investigación en sistemas tolerantes a fallos en el sector de la automoción, con el fin de influir y posicionarse activamente dentro de las actividades y especificaciones a desarrollar en el estándar Adaptive-AUTOSAR.

El presente Proyecto Fin de Carrera es un estudio previo con el que evaluar la viabilidad y líneas de investigación del proyecto DINAMOS durante sus primeras etapas, centrándose en el conocimiento básico inicial del funcionamiento del tipo de microcontroladores que con más frecuencia se utilizan en automoción. Para ello se propone partir de un *kit* basado en la plataforma Arduino, que contiene las piezas básicas para la creación de un vehículo autodirigido siguiendo un determinado objetivo, y actuando en respuesta a diferentes sensores, como de detección de luz, de obstáculos por infrarrojos y ultrasonidos, y detección de líneas. El equipamiento inicial viene con unas funciones básicas, y en el presente proyecto se desarrollan incrementando su complejidad e integración, para ofrecer una mayor y más real funcionalidad. Para ello se combinan las diferentes funciones básicas para que el movimiento del vehículo sea realmente autónomo siguiendo unas reglas básicas preprogramadas en el microcontrolador Arduino.

Así, basándose en la funcionalidad del seguimiento de línea, que sería el equivalente al seguimiento de carril, en la funcionalidad de evitación de obstáculos y en otra de control básico de velocidad, se ha implementado un control de velocidad adaptativo, o *Adaptive Cruise Control*. El ACC controla en todo momento la velocidad del vehículo para que éste circule a una velocidad prefijada por un circuito dado, evitando en todo momento colisionar con otro prototipo que circula delante de él a velocidad variable, y adaptándose a ella si éste le precede siguiendo su misma trayectoria.

Con la experiencia obtenida se ha comprobado que la plataforma Arduino es una buena herramienta para la realización de pruebas experimentales orientadas a conocer y profundizar en las funcionalidades avanzadas y novedosas que serán comunes en los próximos años dentro del sector de la automoción, estudiando en ellos tanto aspectos tecnológicos, como algorítmicos, como de seguridad.

CAPÍTULO 2

CONOCIMIENTOS PREVIOS

El sector de la automoción es un sector muy dinámico, que no cesa en su ímpetu de innovación tecnológica debido a la elevada competencia y a disponer de un mercado bastante estable, por la continua necesidad de renovar los vehículos cada cierto tiempo, ya sea por una necesidad objetiva, o por una autoconvicción subjetiva producida generalmente por la moda, e incentivada por esa sensación de "boom" tecnológico que continuamente proporciona el sector.

No obstante, en la actualidad sí que podemos estar ante el inicio de un verdadero cambio en el sector automovilístico. Una parte muy importante de esta sensación es el surgimiento de la innovación tecnológica por el cambio de combustible, pasando de combustibles fósiles a la energía eléctrica. Pero éste no es el aspecto que nos interesa en este trabajo, sino la otra parte del "boom", que siendo ortogonal a la primera, consiste en la progresiva automatización de la propia conducción, con su tendencia a largo plazo de eliminar el factor humano, en mayor o menor medida, en la continua atención a la carretera.

Ese incremento de la automatización de la conducción se está llevando a cabo desde hace tiempo, con pasos muy pequeños, por las implicaciones en la seguridad y por los altos requisitos tecnológicos que precisa. Así, se han ido introduciendo pequeñas pero cualitativamente importantes funcionalidades en los automóviles. La primera propuesta industrial consistió en el control de velocidad, y a mediados de los 50 aparecen las primeras patentes que tuvieron aplicación industrial.

En la presente década las innovaciones se están acelerando. Tanto es así, que ya empiezan a parecer alcanzables a medio y largo plazo los niveles más elevados de la clasificación de navegación autónoma, o autonavegación.

En el presente capítulo se iniciará repasando el estado de la automatización en el sector del automóvil, a través de su evolución histórica, destacando la fuerte asociatividad de este sector tan competitivo como un factor dinamizador del mismo, para después revisar el estado tecnológico actual, así como las tendencias que se prevén en el futuro.

A continuación se presentará el concepto de navegación autónoma, y los diferentes niveles definidos en función de su menor o mayor autonomía. Una de los funcionalidades previas para logar niveles de elevada navegación autónoma, son los sistemas de asistencia a la conducción (ADAS), con especial atención a los denominados "inteligentes", caracterizados ya no solo por advertir al conductor, sino por ser capaces de

tomar la iniciativa según los casos. Se clasificarán los diferentes ADAS existentes o propuestos en la actualidad, y se analizará con especial detalle el control de velocidad adaptativo (ACC), señalando las principales fuentes de información y los diferentes términos que diferencian las soluciones propuestas por diferentes investigadores o constructores.

Posteriormente se realizará un repaso al estado del arte en las funcionalidades concretas que se van a estudiar e implementar en este proyecto: el seguimiento de trayecto o carril, la evitación de obstáculos, o frenado y evasión automática, la detección de cambios en la visibilidad y en el entorno ambiental durante la circulación, y por último una combinación de ellas, que proporciona el mencionado control de velocidad adaptativo.

Finalmente, se incidirá sobre aspectos específicos a tener en cuenta en todo proyecto de investigación en el sector de la automoción: la seguridad y la fiabilidad, y la importancia de la etapa del desarrollo experimental dentro del proceso de investigación y desarrollo. En el desarrollo experimental se valorará el prototipado, la importancia de las bases teóricas a aplicar, como la teoría de control, y la formalización de los algoritmos como procedimiento para asegurar una correcta implementación de las funcionalidades según su especificación de requisitos.

2.1. La automatización en el sector del automóvil

El sector del automóvil está altamente automatizado, no sólo en su fabricación, sino recientemente también en las funcionalidades que incluyen en sus vehículos, y que tienen una gran aceptación por parte de los conductores, convirtiéndose en uno de los aspectos más importantes a la hora de valorar la elección del modelo a adquirir.

2.1.1. Breve historia

La automatización del automóvil nace con él mismo. Al principio, la automatización es mecánica, como el sistema del regulador centrífugo para el mantenimiento de la velocidad utilizado a principios de siglo XX en los vehículos Wilson-Pilcher y Peerless. Posteriormente esa automatización pasa a ser electromecánica, como la utilizada para el primer sistema de control de la velocidad en el Chrysler Imperial en 1958. Ese tipo de control estático se le conoce actualmente como "control convencional de velocidad" (CCC en inglés). La automatización posteriormente pasó a ser eminentemente electrónica, enfocándose hacia aspectos relacionados con el divertimento, la información o la comodidad, tal como elevalunas eléctricos, sistemas de información integrados en los aparatos de radio, evolucionando hasta los actuales medios de entretenimientos digitales. No obstante, a finales de los 60 Radio Corporation of America patentó el primer sistema de control de velocidad electrónico, aunque la primera implementación del control de velocidad basada en sistemas integrados tuvo que esperar a los años 80, con la aparición del *Auto Speed Control Processor* de Motorola.

En esa década de los 80 empieza el desarrollo y comercialización de sistemas electrónicos y mecánicos que entran de lleno en el tema de la seguridad, como el sistema anti-bloqueo de ruedas (ABS) y el control electrónico de estabilidad (ESP), que han sido ampliamente aceptados por los conductores, dado que son sistemas transparentes para la conducción y que resuelven problemas en momentos concretos irresolubles ya por los conductores.

No es hasta inicios de este siglo XXI, con la generalización de sistemas distribuidos basados en centralitas electrónicas (en inglés, ECU), cuando se plantean sistemas ADAS más avanzados. Mediante la integración de diferentes sistemas electrónicos ya existentes se obtienen unas funcionalidades ya dirigidas

directamente a asumir, en todo o en parte, áreas hasta ahora reservadas al conductor humano. Las primeras han sido ayudas tales como el AEB (*Autonomous Emergency Braking*) y el LKA (*Lane Keeping Assist*), que previenen o incluso en casos extremos actúan, cuando se detecta alguna situación peligrosa en la conducción, como una inminente colisión o un súbito cambio de carril. El siguiente paso ha sido integrar esas nuevas funcionalidades, dando lugar a otras más avanzadas como el ACC, que servirá de caso de estudio para este proyecto de fin de carrera. Estos sistemas equiparan la relevancia de los sistemas automáticos con el conductor humano, puesto que algunas de las tareas de la conducción son asumidas parcialmente por los automatismos, pero bajo la obligada supervisión humana.

2.1.2. Organizaciones del sector de la automoción

El sector de la automoción, aunque está regido por unas fuertes normativas legales y de estandarización, es muy dinámico debido a la confluencia de muchos factores (competitividad, internacionalización, gran valor añadido del diseño y la innovación, excelente recepción de novedades por parte de los clientes y la gran permeabilidad al avance de la tecnología en otros sectores), pero uno de los más importantes es su capacidad de organizarse a diferentes niveles, y entre diferentes actores.

Esas organizaciones son tan importantes que deben ser consideradas en todo proyecto o línea de investigación enfocada al sector de la automoción, como es el proyecto DINAMOS.

Las asociaciones profesionales son importantes, dado que toda nueva tecnología debe ser aceptada y, sobre todo, reconocida por sus ingenieros asociados para que la incluyan en sus proyectos, y previamente estandaricen sus características. También es muy importante estudiar la interoperatividad con los otros sistemas existentes, a fin de evitar redundancias y favorecer la sinergia con esas otras funcionalidades.

Las organizaciones de estandarización y normalización también juegan una importante baza, puesto que todo elemento que se pretenda incluir en un vehículo debe ser capaz de homologarse, y por lo tanto, es importante comprender que características tales como fiabilidad, seguridad, reproducibilidad y sujeción a los estándares previamente existentes, son factores vitales que deben tenerse en cuenta durante todo el proyecto de investigación.

Las organizaciones y administraciones nacionales e internacionales son vitales, dado que son las que aprueban las normativas legales que deberá cumplir cualquier nuevo sistema de ayuda a la conducción, y además, establecerá su alcance, límites y responsabilidades. Por lo tanto, los proyectos deberán proponer soluciones robustas y autocontenidas, en el sentido de que puedan adaptarse por ellas mismas a la legislación, sin dependencia de otros sistemas.

Finalmente, también están las asociaciones y consorcios empresariales, así como organizaciones internacionales conformadas por los grandes grupos de fabricantes de automóviles. Organizan numerosos foros donde se discuten y planean los grandes objetivos a largo plazo, así como la hoja de ruta para conseguirlo, realizarlo de una forma organizada, y que no planteen rupturas tecnológicas que puedan provocar la pérdida del control de sus actuales asociados.

Aunque en mucha menor medida, también podría hablarse del asociacionismo de los usuarios, de los transportistas y de los periodistas especializados en automoción y transporte, pero su impacto en la evolución tecnológica es mucho menor que en otros aspectos del mundo del automóvil.

En la Tabla 2.1 se listan las principales organizaciones de cada tipo acabado de identificar. Será conveniente consultar sus resúmenes anuales, técnicos o de previsión del futuro, para identificar nuevas tendencias, así como para conocer el estado actual y potencial futuro de las tecnologías relacionadas con nuestra área de investigación, y con cada proyecto concreto que se desarrollo dentro de ella.

Principales organizaciones del sector automovilístico			
Standard	ASME	American Society of Mechanical Engineers	https://www.astm.org/
	ISO	International Organization for Standardization	http://www.iso.org/
	JASO	Japanese Automotive Standards Organization	http://www.jsae.or.jp/
Oficiales	EUCAR	European Council for Automotive R&D	http://www.eucar.be/
	NHSTA	National Highway Traffic Safety Administration	http://www.nhtsa.gov/
	TRAN	European Parlament Committee on Transport and Tourism	http://www.europarl.europa.eu/
	WP.29	World Forum for Harmonization of Vehicle Regulations	http://www.unece.org/
Técnicas	ASME	American Society of Mechanical Engineers	https://www.asme.org/
	FISITA	International Federation of Automotive Engineering Societies	http://www.fisita.com/
	JSAE	Japan Society of Automotive Engineers	http://www.jsae.or.jp/
	SAE	Society of Automotive Engineers	http://www.sae.org/
Empresariales	AAM	Alliance of Automobile Manufacturers (Auto Alliance)	http://www.autoalliance.org/
	ACEA	Association des Constructeurs Européens d'Automobiles	http://www.acea.be
	CAAM	China Association of Automobile Manufacturers	http://www.caam.org.cn/english/
	IATF	International Automotive Task Force	http://www.iatfglobaloversight.org/
	JAMA	Japan Automobile Manufacturers Association	http://www.jama-english.jp/
	KAMA	Korea Automobile Manufacturers Association	http://www.kama.or.kr/eng/
	SIAM	Society of Indian Automobile Manufacturers	http://www.siamindia.com/
Otras	Auto-ISAC	Automotive Information Sharing and Analysis Center	https://www.automotiveisac.com/

Tabla 2.1: Principales organizaciones del sector automovilístico

2.2. Tendencia actual: hacia la navegación autónoma

Tras repasar la evolución de la automatización en el sector, a continuación se presenta uno de los principales objetivos de investigación, e incluso de implantación a pequeña escala: la conducción autónoma, entendiendo como tal, la menor, mayor o absoluta participación de los sistemas electrónicos en la conducción y gobierno del vehículo.

2.2.1. Niveles de automatización de la conducción autónoma

Existen dos principales clasificaciones, la publicada en 2014 por la SAE (*Society of Automotive Engineers*), y la presentada en 2013 por la NHTSA (*National Highway Traffic Safety Administration*). En las Tablas 2.2 y 2.3 comparan estos niveles.

Es muy importante reseñar que durante la redacción del presente proyecto, a mediados de septiembre de 2016, la NHTSA abandonó su anterior clasificación propia para adoptar la más reciente e internacionalmente reconocida propuesta de la SAE [37]:

> *There are multiple definitions for various levels of automation and for some time there has been need for standardization to aid clarity and consistency.* ***Therefore, this Policy adopts the SAE International (SAE) definitions for levels of automation****.*
>
> ...
>
> *Using the SAE levels, DOT draws a distinction between Levels 0–2 and 3–5 based on whether the human operator or the automated system is primarily responsible for monitoring the driving environment. Throughout this Policy the term "highly automated vehicle" (HAV) represents SAE Levels 3–5 vehicles with automated systems that are responsible for monitoring the driving environment.*

Nivel	Descripción
0	Los sistemas automáticos no tienen ningún control sobre el vehículo, pero pueden lanzar avisos.
1	El conductor debe estar preparado para tomar el control en cualquier momento. El sistema automatizado puede incluir funcionalidades tales como el control de velocidad adaptativo (ACC), la asistencia al aparcamiento con dirección automática, y asistencia al seguimiento de carril del tipo II en cualquier combinación.
2	El conductor está obligado a detectar objetos y sucesos, y responder ante ellos en caso de que el sistema automático no responda adecuadamente. El sistema automático acelera, frena y gira. El sistema automático puede desactivarse inmediatamente si el conductor toma el control del vehículo.
3	Dentro de entornos conocidos y limitados, tales como autopistas, el conductor puede distraer su atención de la conducción con seguridad.
4	El sistema automático puede controlar el vehículo en todos los entornos excepto en algunos casos concretos, tales como condiciones metereológicas adversas. El conductor debe activar el sistema automático únicamente cuando sea seguro hacerlo. Mientras esté activo, no se requiere de la atención del conductor.
5	Aparte de establecer el destino, y de iniciar el sistema, no es necesaria ninguna otra intervención humana. El sistema automático puede conducir por cualquier lugar por el que esté permitido conducir.

Tabla 2.2: Niveles de grado de autonomía en la conducción de la SAE

Nivel	Descripción
0	El conductor controla completamente el vehículo en todo momento.
1	Los controles individuales del vehículo están automatizados, tales como el control electrónico de estabilidad, o el frenado automático.
2	Por lo menos dos controles pueden estar automatizados conjuntamente, tal como sucede con el control ingeligente de velocidad y el seguimiento del carril.
3	El conductor puede ceder completamente el control de todas las funciones críticas de seguridad en determinadas situaciones. El coche detecta cuándo las condiciones requieren que el conductor retome el control, y proporciona para ello un “tiempo de transición suficientemente confortable” al conductor.
4	El vehículo realiza todas las funciones críticas de seguridad durante todo el trayecto, sin que se espere que el conductor deba tomar el control del vehículo en ningún momento. Dado que este vehículo podría controlar todas las funciones desde el inicio hasta el final, incluyendo las funciones de aparcamiento, podría incluir a los vehículos autónomos no tripulados.

Tabla 2.3: Niveles de grado de autonomía en la conducción de la NHTSA

Por lo tanto, dentro de los seis niveles definidos por la SAE, se distinguen dos clases muy diferentes de automatizaciones, en función de quién es el responsable último de la seguridad del vehículo y su entorno: el conductor (niveles 0 a 2) o el control automático (niveles 3 a 5).

Además, a la vista de estas redacciones, queda claro que, desde la perspectiva de la SAE, las funcionalidades en estudio en este proyecto tienen un nivel de automatización de 1, dado que expresamente se

clasifica al ACC como de ese tipo. En la antigua catalogación de la NHTSA, el control de velocidad adaptativo propuesto en este proyecto sería de tipo 2, pues reúne al menos tres características básicas de tipo 1: el control de velocidad, la evitación de obstáculos y el seguimiento de trayectoria.

2.2.2. Clasificación de los sistemas avanzados de asistencia al conductor (ADAS)

ADAS es un término muy general que incluye multitud de funcionalidades de diferente grado de complejidad y prestaciones, y que además no se trata de un conjunto cerrado, pues cada fabricante pretende distinguirse de los demás, y las soluciones que se aportan, aún pareciendo similares, varían mucho en cuanto a su alcance. Esto es así porque para cada una de las funcionalidades que podrían distinguirse, ésta se puede implementar con un grado diferente de automatización, y además, existe la integración de varias de estas funcionalidades básicas en una nueva propuesta más avanzada.

Con el objeto de simplificar y clarificar, en la Tabla 2.4 se resume la clasificación propuesta por Thalen [67]. En ella se observa que existen tres categorías diferentes: las longitudinales, las laterales y las demás. En esa tabla, el autor de este PFC ha incorporando y clasificado, siguiendo el mismo criterio, nuevas tecnologías ADAS que no fueron consideradas en su momento por Thalen.

2.2.3. Fuentes de información sobre ADAS

A la hora de buscar información sobre algún sistema ADAS en concreto, o sobre ellos en general, hay que distinguir el plano investigador universitario y gubernamental, el plano investigador privado y empresarial y el plano de producción.

La información sobre los sistemas actualmente implementados en coches de consumo puede consultarse en la página web de los fabricantes, así como en las numerosas revistas del mundo del motor cuando analizan e incluso comparan estos sistemas. La información técnica en estos casos es muy limitada, puesto que los detalles forman parte de la propiedad intelectual de las compañías automovilísticas, y por lo tanto, guardado en alto secreto. Otro tanto, e incluso todavía más acrecentado, sucede con las investigaciones privadas, pues es un arma competitiva de primer orden, que solo se revelan ante un inminente lanzamiento al mercado.

Por lo tanto, lo mejor es acudir a los trabajos académicos y gubernamentales para conocer las tendencias del sector. En este sentido, son muy útiles los *surveys*, o estudios resumen, que se centran en determinados aspectos de la tecnología.

A continuación se presenta un listado de *surveys* interesantes que es conveniente revisar para iniciarse en el área de la investigación sobre sistemas ADAS, ordenado por temática:

Algoritmos Okuda [51] presenta un resumen y análisis de los algoritmos que se están implementando actualmente, o están en investigación, especialmente los basados en el análisis de sensores de visión tridimensional. Otro trabajo interesante es el de Goerzen [23]. Aunque está centrado en la navegación autónoma aérea, los algoritmos que se describen de evitación de obstáculos y de alcance de objetivos predefinidos pueden ser de interesante estudio para su aplicación a la navegación del automóvil.

Control de velocidad adaptativo (ACC) Xiao [74] hace un muy buen resumen sobre qué han sido, cómo son, y cómo se espera que evolucionen los ACC en las diferentes áreas del mundo. Además, detalla los estudios que se han realizado sobre aspectos tales como el impacto y aceptación por parte del conductor, los efectos sobre la seguridad vial general, y otros aspectos socioambientales (medioambiente, negocio y responsabilidad legal).

ADAS			Descripción
Longitudinal	ACC	*Adaptive Cruise Control* Control de velocidad adaptativo	El sistema asegura que, en caso de que exista un coche circulando delante en el mismo sentido, se mantenga una distancia de seguridad con el mismo
	ISA	*Intelligent Speed Assistance* Asistente inteligente de velocidad	Actúa sobre la velocidad del vehículo. La velocidad puede ser fijada por el conductor, o suministrada externamente (GPS o por la infraestructura viaria)
	FCW	*Forward Collision Warning* Aviso de colisión frontal	Avisa cuando aparece un vehículo enfrente del nuestro con peligro inminente de colisión frontal
	AEBS	*Automatic Emergency Breaking System* Sistema automático de frenado de emergencia	Como el FCW pero con actuación automática en el caso extremo e inminente
	PPP	*Predictive Pedestrian Protection* Protección predictiva de peatones	Detiene el vehículo cuando detecta la presencia cercana de peatones frente al vehículo para evitar su atropello
		Wrong-Way Driver Warning Aviso al conducto de contra-dirección	Detecta la circulación en sentido contrario al permitido, avisando al conductor para que corrija la marcha
Lateral	LDW	*Lane Departure Warning* Aviso de pérdida de carril	El sistema monitoriza el carril que sigue el conductor, y avisa si el vehículo se sale de él
	LKS	*Lane Keeping System* Sistema de seguimiento de carril	Extensión del LDW: el sistema no avisa, sino que actúa, ante un cambio de carril, reposicionando el vehículo en el carril
	LCA	*Lane Change Assistance* Ayuda al cambio de carril	El sistema comprueba los ángulos muertos de visión, y advierte al conductor si éste intenta hacer un cambio de carril en condiciones peligrosas
		Turning assistant Asistente durante el giro	Se activa al cambiar de dirección en una ciudad y detecta el tráfico frontal y lateral que puede ser peligroso, avisando e incluso frenando en caso de inminente peligro
		Intersection assistant Asistente para los cruces	Se activa en las intersecciones, y avisa al conductor de situaciones peligrosas inminentes
Otras	PA	*Parking Assistance* Ayuda al aparcamiento	El vehículo ayuda en las maniobras a realizar para el aparcamiento, generalmente en paralelo, o incluso son capaces de realizar toda la maniobra completa autónomamente
	EDA	*Emergency Driver Assistant* Asistente de emergencia al conductor	Monitoriza el estado del conductor, y avisa o incluso detiene la marcha de forma segura si concluye que el conductor no está en condiciones adecuadas de salud para continuar
	TSR	*Traffic Sign Recognition* Reconocimiento de señales de tráfico	Sistema automático que reconoce las actuales señales de tráfico, y avisa sobre su existencia, así como sobre si se está violando sus indicaciones (por ejemplo, circular a mayor velocidad, o en contra-dirección)
	V2x	*Vehicular communication systems* Sistemas Sistemas de comunicación vehicular	Incluye muchos tipos de comunicaciones, tanto intra-vehiculares como entre el vehículo y la infraestructura, o comunicación por satélite o redes inalámbricas. Las más avanzadas proponen la cooperación entre vehículos como una forma de agilizar la circulación e incrementar su seguridad
		Night Vision Systems Sistemas de visión nocturna	El sistema proporciona información sobre el mundo exterior en situaciones de baja visibilidad, como conducción nocturna o tiempo adverso (lluvia intensa, nieve o niebla)
		Surround View System Sistema de visualización envolvente	Sistema de cámaras que permite mostrar conjuntamente al conductor imágenes de todo alrededor del vehículo, sin existencia de ángulos muertos. De por sí, se trata de un sistema de información, no de actuación ni aviso

Tabla 2.4: Clasificación de los ADAS

Navegación autónoma En un trabajo de 2014, Anderson et al. [3] analizan no sólo la evolución histórica y el estado actual de la navegación autónoma, sino que resumen las legislaciones, normativas, estándares y demás elementos legales y técnicos que deben tenerse en cuenta a la hora real de lanzar un vehículo autónomo, enumerando los riesgos existentes, incluyendo la ciberseguridad, y la protección de datos y comunicaciones.

Señalización vial La utilización de información extra-vehicular, basada en sistemas que modernicen o capturen la actual señalización vial son analizados con detalle por Møgelmose et al. [35].

Posicionamiento y geolocalización El trabajo de Skog y Handel [64] estudia la aplicación de la tecnología de posicionamiento, principalmente a través de la geolocalización, a la conducción autónoma y resto de ADAS, evaluando la integración de los diferentes sistemas disponibles, especialmente GPS.

Normativa americana La NHTSA acaba de publicar su política para vehículos autónomos [37]. Es muy importante porque, además de aceptar los niveles de navegación autónoma de la ESA, entra en detalles

sobre cómo deben los fabricantes homologar sus vehículos y cómo la responsabilidad debe repartirse entre ellos y el conductor según los casos.

Planificación gubernamental europea Desde las instituciones de la Unión Europea también se trabaja intensamente en la planificación, apoyo y seguimiento de la investigación en navegación autónoma.

Como ejemplo de funcionamiento y previsión política para inversiones en investigación e infraestructuras, tenemos la hoja de ruta 2016–2020 [24] que plantea el centro investigador HTSM de la Universidad de Delft (Dinamarca), en base a la agenda estratégica emanada del Consejo europeo asesor de investigación del transporte por carretera (ERTRAC), de la Unión Europea.

Por su parte, el Parlamento Europeo, a través de la Dirección general de políticas internas, del Comité TRAN, de transporte y turismo, ha publicado en 2016 un estudio [21] sobre el desarrollo de la navegación autónoma y de la conexión intra-vehicular tanto en Europa como fuera, y hace un resumen del estado actual de la tecnología, y un análisis de sus potenciales impactos, realizando recomendaciones sobre los proyectos de investigación en esa área.

2.3. ACC: el control de velocidad adaptativo

Como ya se ha visto, el ACC (*Adaptive Cruise Control*), o control de velocidad adaptativo, es un ADAS avanzado que integra varios ADAS más simples dentro de él, aunque según la clasificación de la SAE, siendo un sistema de navegación autónoma de nivel 1.

A continuación se repasará la evolución reciente que ha tenido el concepto del ACC, el cual no sólo ha evolucionado, sino que según el autor o fabricante puede tener significados ligeramente diferentes, según el alcance real de su implementación y los subsistemas ADAS básicos que integre. Finalmente se analizarán las funcionalidades básicas que se utilizarán para implementar el ACC tal cual se presenta en este PFC.

2.3.1. Evolución del ACC

El control de velocidad adaptativo, en inglés *Adaptive Cruise Control* (ACC), es una funcionalidad que empezó hace diez años a implementarse en los coches de lujo y alta gama, pero que con el tiempo ha ido popularizándose, y actualmente es muy frecuente incluso en la media gama.

Ya en 2005, en una reunión del U.S. Software System Safety Working Group [2], se definió lo que se entendía por ACC, y esa definición ha variado muy poco hasta la fecha, añadiendo variaciones en función de nuevas funcionalidades o características que cada fabricante ha ido añadiendo en sus implementaciones. En ese trabajo se define claramente el concepto, así como se exploran sus posibilidades de implementación, incluyendo los diferentes estados por los que un sistema así debería transitar.

Posteriormente, Thalen [67] estudió también su relación con otros sistemas inteligentes que empezaban a surgir, o que su desarrollo era inminente, aunque de implementación no trivial algunas de ellas, llegándose a analizar su interrelación, centrándose también en los aspectos psicológicos para su aceptación, lo cual apareció como algo muy importante a tener en cuenta para el éxito y propagación de estas tecnologías.

Es interesante reseñar que, en este PFC, el ACC que se implementará parte del ACC tal cual es definido por Thalen, pero incorpora además el sistema automático de frenado de emergencia (AEBS), que es una simplificación o particularización de nuestra funcionalidad de evitación de obstáculos, de real actuación ante la posibilidad de choque, y el seguimiento de carril (LKS), esto es, combina tres funciones ADAS, dos longitudinales y una lateral.

Para conseguir las avanzadas funcionalidades del tipo ADAS, varios investigadores han propuesto que la arquitectura hardware y software que dé soporte a esta red sea distribuida [27]. En ese trabajo, no sólo se diseña e implementan varios ADAS, incluido un ACC, sino que también se realiza una comprobación experimental en circuito cerrado controlado [28]. El trabajo con componentes distribuidos está generalizado en el sector de la automoción, dado que las centralitas electrónicas actuales agrupan varios sensores y sistemas distribuidos por todo el coche, mediante la utilización de buses industriales estandarizados. La arquitectura distribuida deberá contemplarse en una fase posterior del proyecto DINAMOS, para integrar las diferentes funcionalidades avanzadas que vayan desarrollándose.

Además de todas estas tecnologías, que se centran especialmente en funcionalidades a implementar individualmente en cada vehículo, sin necesidad de interacción con los demás ni en general tampoco con la infraestructura vial, existen otras áreas de investigación que propugnan una eficiente cooperación intravehicular como método más eficiente para resolver algunos de los problemas más complejos. Así aparece el concepto del control cooperativo y adaptativo de velocidad, *Cooperative Adaptive Cruise Control* (CACC), del que incluso se han realizado pruebas experimentales en trazados largos reales, tanto en turismos [54] como en transporte pesado [50].

2.3.2. Términos directamente relacionados con el ACC

Aunque el término ACC (*Adaptive cruise control*) está fuertemente adoptado, su alcance y límites a la hora de la implementación es algo difuso, puesto los fabricantes ofrecen características ligeramente diferentes entre sí. Xiao y Gao [74] indicaron otros términos o conceptos que están íntimamente ligados con el ACC y que pueden o no estar integrados en una implementación real, los cuales se resumen en la Tabla 2.5.

Autónomos	No autónomos
Auto-Adaptive Cruise Control	
Automatic Cruise Control	Semi-Autonomous Adaptive Cruise Control
Active Cruise Control	Coordinated Adaptive Cruise Control
Advanced Cruise Control	Cooperative Adaptive Cruise Control
Intelligent Cruise Control	

Tabla 2.5: Términos relacionados con el ACC

Para ver las sutiles diferencias entre estos términos, acúdase a las referencias del trabajo de Xiao y Gao.

2.3.3. Integración de las funcionalidades básicas del ACC

El ACC, en este PFC, se va a implementar a partir de tres funcionalidades básicas: la evitación de obstáculos, el seguimiento de trayectoria y el control de velocidad.

Para la integración de funcionalidades, es interesante el trabajo de Yoshioka [75], por cuanto propone un *framework* para el diseño de sistemas incrustados para la automoción, pero desde una perspectiva de la reutilización de componentes hardware y software existentes. Para ello se insiste en la detección de módulos o componentes tanto software como hardware que puedan ser comunes a los diferentes sistemas, y así ahorrar en el desarrollo y en las pruebas de estandarización y de testeo.

Dado que en este proyecto se aboga por la implementación de las funcionalidades complejas en base a funcionalidades más simples, y éstas, en las más básicas, la reutilización de componentes hardware y software es muy interesante. Así, en el presente trabajo, se han diseñado e implementado primero las

funcionalidades básicas (evitación de obstáculos y seguimiento de carril), para después dar paso a otra más compleja, el control de velocidad adaptativo, que lo conforma una parte de la evitación de obstáculos con el seguimiento de carril más una tercera funcionalidad nueva, el control de la velocidad. El hardware se reutiliza, así como gran parte del código de las funcionalidades básicas, que se integran para obtener la implementación final.

2.3.3.1. Seguimiento de trayecto

El seguimiento de trayecto basado en el seguimiento de una línea es una variante del LKS, un ADAS consistente en la reacción por parte del propio vehículo a una pérdida del carril, de tal forma que éste recupere el carril por el que circulaba. La variación consiste en la forma de señalizar el carril: en lugar de ser líneas laterales continuas o discontinuas como en las carreteras convencionales, se sigue una única línea negra pintada en el suelo que representa la trayectoria a seguir.

Esta funcionalidad está bastante presente en los automóviles de alta gama, pero aún así se continúa en su investigación. Los algoritmos para mantener el vehículo en el carril son variados, por ejemplo basándose en un control de tipo PID, como en [61], en donde se estudia teóricamente el bucle cerrado de control y se realizan simulaciones, suponiendo un sistema de sensores y actuadores múltiples. Un trabajo similar, también basado en la teoría de control, analiza el caso de las curvas cerradas, y de cómo el sistema podría emular el comportamiento humano en cuanto a la forma de trazar esas curvas de la forma más natural posible [77]. Un trabajo más fundado teóricamente y mucho más parametrizado fue presentado por Wu et al. [73].

Un estudio pionero muy interesante fue el realizado por Pilluti y Galip [53] en donde el sistema de control propuesto analizaba en línea el comportamiento dinámico del conductor durante sus tareas de conducción fijándose en cómo éste mantenía el carril. El sistema se adaptaba al estilo de conducción y con ello llegaba a detectar situaciones anómalas provocadas por falta de atención o fallo en la salud del conductor, pudiendo servir como base a un futuro sistema de monitorización del estado del conductor.

2.3.3.2. Detección de variación de la iluminación

La variación de la iluminación es un factor que puede ser interesante considerar en varias situaciones de la conducción. Por ejemplo, durante la entrada en un túnel, o atravesar una carretera con mucha sombra, o la excesiva iluminación frontal en los amaneceres y ocasos. Estas situaciones pueden poner en riesgo la capacidad de percepción del conductor.

Las acciones de control básicas consisten en la variación de la intensidad de la propia luz del vehículo, para ajustarse a las condiciones de penumbra, o incluso, la activación automática de filtros en el parabrisas para limitar el deslumbramiento. Otras técnicas basadas en la detección de la luz consisten en proporcionar una ayuda a la conducción nocturna, no tan sólo a los conductores, sino también como un elemento más a considerar para un conducción autónoma nocturna o en situaciones de iluminación cambiantes [15].

2.3.3.3. Evitación de obstáculos

La evitación de obstáculos, y cómo reaccionar o avisar ante ellos, es una de las principales funcionalidades que debe integrar un sistema de navegación autónoma, dado que éste debe evitar en todo momento el choque, pero también debe ser capaz de cambiar de ruta y seleccionar la más adecuada para no colisionar, y a la vez, permitir recuperar la ruta inicial sin más retraso.

Esta funcionalidad también es útil para el ACC, aunque con una variante más simple: en el ACC la trayectoria está prefijada, puesto que estamos siguiendo el carril, de tal forma que sólo se requiere el no

colisionar, esto es, frenar o disminuir la velocidad lo que sea necesario, pero no se le requiere al sistema que abandone su actual dirección de marcha para después tener que recuperarla.

En las aplicaciones actuales hay dos principales tipos de sensores que se utilizan a estos efectos: el radar de larga distancia (LRR, *Long Range Radar*) y el sensor Lidar, basado en la detección de reflexión de un haz de luz polarizada. El trabajo de Sun, Bebis y Miller [65] es un clásico del análisis del problema de la evitación de obstáculos, discutiendo las dos tecnologías de sensores comentadas, los problemas de la detección y del seguimiento de los obstáculos, distingue entre sensores activos y pasivos, y analiza comparativamente los diferentes métodos y tecnologías propuestas hasta entonces.

2.4. Seguridad y fiabilidad

En esta sección se repasará brevemente los aspectos de seguridad y fiabilidad más relevantes del software para automoción.

En primer lugar se comentará sobre la seguridad. El término "seguridad" tiene múltiples acepciones, y como tal, resulta ambiguo si no se indica con antelación a su uso el significado que le queremos dar. En este apartado trataremos varios aspectos de la seguridad: la seguridad vial, la seguridad software y la ciberseguridad.

Finalmente se hablará sobre la importancia de la fiabilidad.

2.4.1. Seguridad vial

La seguridad vial es un aspecto vital siempre a considerar cuando se trata de asuntos relacionados con la automoción, pero es mucho mayor incluso cuando se pretende automatizar la propia conducción, o parcelas de ella, por sus importantes repercusiones en la vida de los conductores, pasajeros y viandantes.

Está ampliamente extendido el convencimiento de que la automatización completa de la conducción, si ésta es generalizada, debería conllevar una importante reducción de la accidentalidad, así como de las repercusiones de esos accidentes. Así, el sistema que se diseñe deberá ser seguro en el sentido de que deberá resolver todas las circunstancias que pueda pleantearle la conducción real, con un alto grado de competencia y sin fallos, o al menos, no tan severos como los que cometería un conductor humano en las mismas circunstancias.

2.4.2. Seguridad del software

Para lograr la seguridad vial es vital un correcto funcionamiento del software. En ese sentido es muy importante la labor de ingeniería del software que ha de realizarse, combinada con una adecuada elección del hardware sobre el que se ejecute. En sendos artículos, Pretschner, Broy et al. [57][14] realizan un desmenuzado análisis del problema, e identifican los principales problemas del software en el sector de la automoción, no incidiendo en aquellos que son comunes a todos los campos del software. Así, identifica como propios dos problemas básicos: la integración y la evolución. El problema de la integración consiste en que normalmente cada sistema se desarrolla independientemente de los demás, y su interacción con el resto es descubierta en la etapa del despliegue en los prototipos. El problema de la evolución es debido a la incesante evolución tecnológica del sector, coexistiendo muchas veces diversas versiones de la misma funcionalidad con diferentes comportamientos según cuándo, cómo y dónde se implementaron. Como solución proponen el uso de nuevas tecnologías de ingeniería del software, principalmente las dirigidas por modelos.

Además de estos trabajos, cabe destacar también la necesidad de asumir los estándares principales en el sector de la automoción para los sistemas electrónicos y sus sistemas de comunicación. A continuación se indican las tres principales normas que han de considerarse en un proyecto relacionado con la automoción:

- **OSEK/VDX**: *Open Systems and the Corresponding Interfaces for Automotive Electronics/Vehicle Distributed eXecutive* [52]: Se trata de un estándar abierto, consistente en un sistema operativo para entornos incrustados para la automoción, que incorpora los principales sistemas de intercomunicación usado en el sector.
- **Autosar v4.2.2**: *Automotive Open System Architecture* [9]: Es una evolución del estándar OSEK/VDX, retrocompatible con su sistema operativo y sus módulos de comunicación.
- **CiA**: *CAN (Controller Area Network) in Automation*. Es también el nombre de la organización de fabricantes y de proveedores de servicios que se encargan de la estandarización del bus CAN para su uso en el campo de la automatización, bus que tiene amplio predicamento en el sector del automóvil, a pesar de que no está avalado directamente por una norma estándar internacional, como la ISO. Tiene especial relevancia porque Arduino es compatible con el bus y controladores CAN, y por lo tanto, un punto a favor para su uso en el desarrollo de prototipos para la automoción. Es un ejemplo de cómo el sector a veces se auto-regula para continuar con una tecnología muy extendida que simplemente funciona, aunque formalmente no es un estándar internacional [60].

2.4.3. Ciberseguridad

En este caso utilizamos el término de seguridad con su acepción relacionada con la integridad del sistema, esto es, abarca aspectos tales como la impenetrabilidad, la inalterabilidad de su funcionamiento y imposibilidad de interferencia en las acciones que acomete.

Por las noticias aparecidas estos últimos años, la seguridad vehicular no parece ser el fuerte de las actuales implementaciones, dado que se han publicado muchos ataques exitosos a los mismos, tanto en el entorno investigador, como en el del delito directamente. La inclusión de sistemas inteligentes capaces de gestionar autónomamente un vehículo, hace que éstos puedan ser objeto de ataques no solo individuales, sino incluso a escala general, sobre todo con las recientes amenazas de ataques terroristas.

Es por ello que en estos últimos años han aparecido muchas iniciativas del sector para estudiar este problema y establecer un marco de actuación común, en busca de detectar y sellar los posibles vectores de ataque y amenazas existentes:

Entre ellas caben destacar las siguientes:

- El resumen ejecutivo de 2016 de Auto-ISAC sobre las mejores prácticas en ciberseguridad [8]
- Alliance of Automobile Manufacturers (“Auto Alliance”) y la Association of Global Automakers elaboraron un *framework* sobre las mejores prácticas en ciberseguridad [1].
- El trabajo técnico de análisis de la ciberseguridad en la automoción, especialmente en las comunicaciones, realizado por Asaduzzaman [7]

Finalmente, otra amenaza de reciente y fulgurante aparición, es el *ransomware*. El problema con esta amenaza es que puede afectar a todo un parque de vehículos completo en un mismo momento, provocando atascos de gran tamaño al bloquear el funcionamiento de los vehículos [16].

2.4.4. Fiabilidad

La fiabilidad en el sector de la automoción es un factor crítico, sobre todo en componentes ligados a la seguridad humana y vial. En la última década han sido muy frecuentes las llamadas generales a reparación efectuadas por los fabricantes de coches ante la detección de defectos graves de fabricación o de diseño. Estas llamadas generales son muy dañinas para la reputación de la marca afectada, sobre todo si afecta a componentes tales como el motor o la seguridad. Estas *car recalls* están incluso reguladas legalmente, por ejemplo, en EE.UU. la propia NHTSA informa sobre qué marcas, modelos y número de serie están afectados por estas llamadas a reparación, e incluso permite a los propios usuarios denunciar posibles defectos generalizados de fabricación [48].

Además del desgaste en la reputación, el coste económico es muy elevado. Se estima que junto con el coste de la garantía, las llamadas a reparación representan unos 40.000 millones de dólares en todo el sector, lo que representa entre un 3–5 % de las ventas. General Motors, sólo en el año 2014 reconoció un coste de 2.900 millones de dólares por causas de las llamadas a reparación [22].

En la parte que afecta a este proyecto, los componentes electrónicos también deben tener en consideración estos problemas de fiabilidad. Es cierto que al final, como en todo negocio, el análisis se realiza en base a la relación coste/beneficio, tal como se analiza en [29], pero por ello mismo hay que evaluar muy bien si una reducción en el coste de componentes electrónicos está justificada si pone en riesgo la seguridad y la fiabilidad. En [36] se analizan las principales características físicas y del entorno que afectan a la fiabilidad de los sensores y actuadores electrónicos, mientras que Watson y Castro [68] analizan el problema concreto de las altas temperaturas (más de 150°C) a las que pueden estar sometidos algunos componentes electrónicos críticos.

Esa importancia debe extenderse también a la etapa de prototipado, por cuanto unos componentes electrónicos defectuosos, poco reproducibles o poco fiables pueden dar lugar a comportamientos anómalos que se atribuyan al algoritmo o a la implementación, cuando el problema puede estar en el hardware. Otra fuente de problemas puede estar en la corrección del propio firmware, por lo que una adecuada elección del suministrador y modelo debe valorar también la frecuencia de actualización y disponibilidad del código fuente de este componente intermedio entre el hardware y los *drivers*.

2.5. Desarrollo experimental con prototipos

La evolución tecnológica en el área de la automoción, como ya hemos visto, depende mucho de los desarrollos teóricos, de la investigación básica, del desarrollo normativo y de seguridad, así como de la sinergia de los movimientos asociativos en el sector. Pero todo ello al final debe materializarse en una investigación experimental para comprobar la viabilidad de los proyectos.

2.5.1. Escalas de prototipado

Son muchos los proyectos de investigación que se han realizado con prototipos de tamaño real, como por ejemplo los clásicos ejemplos del CyCab [56], utilizado para estudiar el movimiento autónomo rodeado de viandantes, o el CLMR [31], prototipo para estudiar el aparcamiento autónomo.

Pero estos prototipos a escala real son caros, por lo que el primer paso de esta investigación experimental es probar los algoritmos, los componentes hardware y su integración en prototipos de bajo coste.

Es sorprendente observar que una de las plataformas hardware que es activamente utilizada tanto en el sector industrial como en el investigador es la plataforma *open hardware* Arduino [4], toda vez que ésta nace básicamente como un proyecto educativo.

Así, se observa su utilización como ejemplo base de estudio para el modelado de la implementación de un sistema complejo de *Cooperative Adaptive Cruise Control* (CACC), en donde se utilizan múltiples placas Arduino conjuntamente sobre un bus CAN para implementar ese avanzado sistema [55]. Ése trabajo puede ser un buen referente para el futuro trabajo a realizar tras este proyecto.

2.5.2. Aplicación de la teoría de control

Además de estos trabajos de investigación amparados en realizaciones experimentales para la comprobación del funcionamiento, existen otros en los que se analiza el problema desde el punto de vista del control automatizado. Esto es relevante, toda vez que en este proyecto se implementa un controlador PID para la funcionalidad del control de la velocidad en el ACC estudiado, tipo de control que es bastante básico. Resulta interesante el estudio es los siguientes trabajos:

- Análisis teórico y comprobación experimental del funcionamiento en convoy de varios prototipos Arduino [76], con comunicación intra-vehicular incluida.
- Estudio teórico del control de un convoy de vehículos cooperativos mediante comunicación intra-vehicular, incluida una realización experimental en carretera real [54].
- Control mediante lógica difusa aplicada al problema del aparcamiento automático [31], también con realización experimental incluida, a tamaño real pero en entorno controlado.
- Estudio de la integración de varios sistemas básicos de ADAS para configurar un ACC, desde el punto de vista de la teoría de control, con realización de simulaciones por ordenador [63].

A la vista de estos trabajos, se considera que la formalización de los algoritmos que se utilicen debe ser fuerte, así como su debida contrastación experimental, no sólo porque una buena base teórica ayuda a entender mejor el comportamiento real del prototipo, sino porque en caso de resultar exitoso, es un requisito básico para que pueda evaluarse su implementación en producción, sobre todo si pretenden estandarizarse algunos o todos sus componentes, lo que debe hacerse por organismos externos.

2.5.3. Prototipado a escala pequeña

La utilización de prototipos básicos para simular entornos de navegación de vehículos está muy extendido. Así se puede destacar el reciente prototipo desarrollado para su uso a nivel educativo realizando una combinación de plataformas, Arduino y Android [32].

Además, disminuyendo en complejidad, existe multitud de prototipos económicos que pueden ser utilizados para estas primeras etapas de investigación, y que incluyen tutoriales básicos de montaje y programas de ejemplo, como son el DFRobot [18] y el TINKBOX [25], que comparten muchas características con uno de los usados en este proyecto, el Robotale [62], de la familia 4WD, y también el SunFounder™ [66], que tiene una configuración diferente, con una maniobrabilidad mucho más parecida a los vehículos reales, y por ello más interesante para este proyecto.

CAPÍTULO 3

ESPECIFICACIÓN DE REQUISITOS

A continuación se va a proceder a definir los límites prácticos del alcance del presente proyecto mediante la especificación de requisitos de cada una de las funcionalidades que van a estudiarse con detalle, así como las características que debe aportar la parte hardware para poder dar soporte a dichas funcionalidades de una forma adecuada a las exigencias del sector de la automoción.

La especificación aquí presentada no es la inicial, sino que ha sufrido modificaciones durante la propia realización del proyecto, puesto que, tratándose de un tema de investigación aplicada sobre componentes no habituales y variados, ha debido adaptarse en función del conocimiento y experiencia que se iba adquiriendo. Esta adaptación no implica una variación sustancial de los requisitos iniciales, sino simplemente la corrección de aquellos aspectos tecnológicos que han permitido alcanzar los objetivos inicialmente fijados.

A continuación se especifica, por secciones, cada una de las funcionalidades estudiadas, iniciando con la especificación hardware genérica sobre la que se realizarán los demás apartados.

La especificación hardware se realizará en base a microcontroladores de uso genérico y de fuente abierta, puesto que permite iniciar la investigación a partir de componentes de amplia, variada y rápida disponibilidad. A continuación se especificarán las características básicas que debe tener un sistema de seguimiento autónomo de trayecto, para pasar a continuación al del seguimiento por señalización luminosa y de detección de la luz, por tratarse la variación de condiciones lumínicas un aspecto importante en la conducción. Posteriormente se detallarán los requisitos para la evitación de obstáculos, aspecto muy importante para la seguridad de la navegación. Tras ello se analizará qué es necesario para obtener un adecuado sistema de control de velocidad inteligente, que integra el seguimiento del trayecto con la detección de obstáculos.

3.1. Especificación técnica de la plataforma hardware

En los proyectos destinados a discutir, probar, investigar y experimentar funcionalidades destinadas a su implementación en producción masiva a gran escala en el mundo real, sobre todo en áreas tan altamente competitivas y avanzadas tecnológicamente, la especificación de los requisitos hardware es una parte esencial.

La especificación que aquí se presenta se refiere ante todo, a la de los componentes hardware con los que realizar la parte experimental del proyecto DINAMOS, especialmente para este proyecto fin de carrera

y los trabajos que deriven de él. Pero no por ello se debe perder de vista que, aunque no en detalle, sí que deben coincidir básicamente con los que deberá exigirse al hardware industrial utilizado en producción masiva.

Las características básicas de la plataforma hardware, así como de los componentes individuales utilizados con ella, se detallan a continuación, justificando la necesidad y conveniencia de cada una de ellas:

Plataforma genérica Debe evitarse caer en plataformas específicas y propietarias, que no tengan un amplio reconocimiento y aceptación, y con facilidad de uso. Dentro del sector de la automoción es muy importante la estandarización, no sólo normativa, sino también de producción, al efecto de poder gestionar eficientemente la subcontratación de la fabricación y asegurarse una gran variedad de suministradores con calidades de fabricación comparables.

Normalización Por exigencia legal, y también de diseño, todo componente de un vehículo debe cumplir las correspondientes normativas, certificaciones y estándares legales y técnicos, tales como de compatibilidad electromagnética, seguridad, de calidad, medioambientales, etc. Sobre todo, se debe asegurar la compatibilidad entre los diferentes componentes, y que no aparezcan ruidos parásitos, o que comportamientos secundarios provoquen efectos inesperados en otros sistemas.

Especificaciones abiertas, no privativas Los componentes deben poder ser caracterizados perfectamente, con unas condiciones de control de calidad y de capacidad de comprobación de funcionamiento correcto según unas especificaciones bien establecidas y conocidas. Esto permite una integración eficiente en las etapas de diseño, y además, asegurar una amplia disponibilidad de suministradores.

Alta disponibilidad Los componentes deben ser *common commodities*, esto es, estar ya presentes en el mercado, con una gran implantación, y facilidad y rapidez de obtención. La implementación de técnicas agresivas de producción tales como el *just in time* necesitan que la gestión de suministros sea ágil, eficaz, eficiente y segura.

Elevada fiabilidad Los componentes deben ser fiables tanto en su operación (baja tasa de fallos por unidad de tiempo), como en sus resultados, esto es, que si son sensores, proporcionen resultados correctos dentro de un intervalo de error aceptable predefinido. Esto es muy relevante en el sector de automoción, puesto que no es aceptable que los dispositivos relacionados intrínsecamente con la seguridad del vehículo y de su entorno puedan fallar o proporcionar datos incorrectos durante su vida útil. Esa vida útil ser muy elevada para evitar continuas revisiones o incluso reparaciones en el taller, que siempre son muy costosas, especialmente por la mano de obra y maquinaria usada.

Componentes económicos, baratos Dado que la automoción requiere de la integración de infinidad de pequeños componentes, y que la producción de unidades de coches es muy elevada, unido a ser un sector altamente competitivo en precios, es importante encontrar un ajustado compromiso entre su coste y sus prestaciones, que siempre deben cubrir unas especificaciones mínimas muy exigentes.

Componentes ligeros Deben integrarse fácilmente con el resto, y en relación con su exigencia computacional o de intercomunicación, requerir de bajo consumo eléctrico, bajos tiempos de respuesta y de lectura de las medidas o de actuación, y utilizar eficientemente protocolos simples, ligeros, no pesados, que no saturen las redes (generalmente buses) de comunicación intra- (y en el futuro, inter-) vehicular.

Componentes livianos Referido al peso, que es un factor cada vez más importante en los vehículos, por las exigencias medioambientales sobre su consumo energético. Es por ello que deben ser lo más

pequeños y con menos peso posible, con materiales de baja densidad, pero alta resistencia a las condiciones específicas que deban soportar.

A los efectos de este proyecto, tal como se justificará en el Subsección 4.1.2, la elección se basa en la plataforma Arduino, seleccionándose dos prototipos diferentes, con distintos componentes, para realizar las pruebas.

3.2. Seguimiento de trayectos basado en una línea

El seguimiento de trayectos se implementará mediante el seguimiento de una línea negra pintada en el suelo. En el caso real de conducción en las vías públicas, el seguimiento puede plantearse gracias a las marcas del carril, como en el caso del ProPILOT™ [49][19] propuesto por Nissan, o ya usado por otras marcas como Mercedes-Benz con su DISTRONIC PLUS™ [33] que integra el Lane Keeping Assist™ [17], BMW con su Active Cruise Control™ [13], u otros que prometen ser más avanzados como el Autopilot™ [30] de Tesla.

Esta línea del suelo será de unos 30–40 mm de grosor, y de un color suficientemente distinto en contraste con el resto, preferiblemente blanco.

La línea no se cruzará entre sí (no se ha experimentado cruzándolas), y deberá tener tanto rectas como curvas con distinto radio, radio siempre más o menos adecuado al tipo de vehículo que debe seguir la trayectoria definida. En caso de un radio excesivamente reducido para las capacidades de maniobrabilidad intrínsecas del robot, éste podrá tomar acciones tales como retroceder y volver a avanzar, para lograr seguir la trayectoria fijada.

El pavimento estará lo suficientemente plano como para que todas las ruedas de los vehículos puedan tener agarre simultáneamente en todo momento, para evitar que se pierda tracción, que es un problema que requeriría su propia solución.

El vehículo podrá adaptar su velocidad al trazado, aunque intentará ir a la máxima velocidad dentro de sus posibilidades. Esto es, en curvas pronunciadas es razonable disminuir la velocidad.

En el eventual caso de que el coche salga de la trayectoria (por un error de sensores, por discontinuidad de la línea, por curva excesivamente cerrada, u otros casos), éste deberá tratar de recuperar dicha trayectoria en el menor tiempo posible, y preferiblemente volviendo a entrar por el mismo lugar por donde la extravió.

3.3. Seguimiento de luz y detección de nivel de luminosidad

El objetivo de la prueba de seguimiento de luz consiste en determinar las condiciones bajo las cuales los sensores son capaces de distinguir la diferencia entre diferentes luminosidades, al objeto de poder detectar, en la vida real, situaciones tales como entrada en un túnel, o atardecer y el amanecer, que obligue al vehículo a avisar o realizar acciones tales como encender las luces de cruce o de posición.

Además, disponer de esta funcionalidad permite poder integrarla en cualquier momento con otras funcionalidades que estén dando algún tipo de problemas, al objeto de estudiar si la variación de luminosidad ambiental o dirigida, puede ser la causa de dicho comportamiento.

El seguimiento de luz, tal como se va a implementar en este proyecto, tiene la siguiente especificación:

- El vehículo dispondrá de varios sensores de luminosidad, preferiblemente con salida analógica (esto es, que proporcione información sobre el grado de luminosidad detectado por cada uno de los sensores).
- Los sensores se dispondrán de tal forma que permitan detectar variaciones de luminosidad en los alrededores de los puntos críticos de la marcha del vehículo: parte frontal y parte trasera.
- El vehículo deberá auto-adaptarse a la luminosidad existente durante toda la prueba, tanto si es buena luminosidad general, como oscuridad total.
- Se requerirá una determinada mínima diferencia de luminosidad que permitirá decidir si ha habido cambio de luminosidad, o si hay un diferencial suficiente de luminosidad entre los diferentes puntos alrededor del vehículo.
- El vehículo, como prueba de que detecta esas variaciones, y de cuán rápido lo hace, deberá reaccionar ante esos cambios inmediatamente, dirigiéndose siempre hacia el punto máximo de luminosidad, siempre que la maniobrabilidad intrínseca del vehículo lo permita. Concretamente, no se requiere que tenga inteligencia para detectar un incremento de luminosidad lateral central, y por el que debería realizar maniobras atrás/delante para encararse con ese foco lumínico.
- Cuando se retorne a una luminosidad homogénea, el vehículo deberá detenerse.
- Cuando haya un cambio repentino de luminosidad global, el vehículo deberá detectarlo, pero no deberá realizar movimiento alguno si el cambio de luminosidad es homogéneo en sus aledaños.
- Es interesante que los propios sensores informen de la luminosidad detectada en cada momento, al objeto de determinar la correcta o anómala reacción de la implementación.
- Las acciones en búsqueda de la mayor luminosidad deberán ser también posibles cuando la maniobra requiera dar marcha atrás. Esto es, el vehículo deberá girar de forma acorde si la luminosidad mayor se detecta en un flanco trasero del vehículo, no sólo echar marcha atrás y después marcha adelante en busca del foco.
- En las pruebas, la mayor luminosidad se provocará mediante el foco adecuado de una linterna de mano.

3.4. Evitación de obstáculos

La evitación de obstáculos es un problema muy estudiado en robótica. Pero es preciso definirlo exactamente, puesto que según el criterio que se siga, tiene una funcionalidad diferente.

En la literatura se suele vincular la evitación de obstáculos con el seguimiento de una ruta, o incluso la búsqueda de una ruta desde un punto origen a otro destino sorteando las dificultades existentes, materializadas como objetos que deben evitarse. Tratada en este sentido, no se trata de una funcionalidad propia e individual, sino como una accesoria a otra funcionalidad más importante, pero que le dota de ciertas características que permiten mejorar el comportamiento de la principal, que suele ser la de autonavegación para generar una ruta con la que alcanzar de forma segura (sin choques ni bloqueos) un objetivo, partiendo de un origen. Para ello generalmente se requiere del conocimiento previo de un plano del entorno, o de la existencia de una señal exterior que de alguna forma guíe al robot.

Sin embargo, en este proyecto, la evitación de obstáculos se va a definir de una forma independiente, siguiendo el criterio general de este proyecto de crear primero algoritmos básicos que solucionen problemas

básicos, para después combinarlos para obtener una funcionalidad mayor combinada. En ese momento será preciso adaptar los algoritmos originales en pos de la obtención de un algoritmo más global que dote al prototipo de una funcionalidad de nivel superior. Se trata pues, de generar soluciones del estilo *buttom-up*, esto es, primero las funcionalidades y algoritmos básicos, para posteriormente ir combinándolos para generar algoritmos de mayor nivel. La razón de seguir esta aproximación al problema es porque se parte de la base de querer implementar los algoritmos en una plataforma hardware limitada computacionalmente, por lo que la escasez de recursos de memoria y de cálculo hacen que deba asegurarse que las implementaciones sean lo más eficientes posibles, y para ello se requiere que sus algoritmos también sea escuetos, aunque funcionalmente eficaces.

A nuestros efectos la evitación de obstáculos se va a definir como la capacidad del robot de evitar a toda costa el choque físico con otros elementos que le rodean, sean éstos fijos o móviles, manteniendo en lo posible su capacidad de continuar en movimiento.

Consecuentemente, la especificación de la funcionalidad general de evitación de obstáculos consiste en los siguientes puntos básicos comunes:

- El coche, o robot, dispone de sensores de detección de obstáculos. Éstos podrán ser uno o varios, y de diferentes características en cuanto a la sensibilidad y precisión de sus datos, pero deberán ser predeterminadas puesto que caracterizarán el algoritmo que resuelva cada problema concreto.
- El coche, o robot, tiene capacidad para moverse de una forma no-holonómica, pero conocida.
- El coche, o robot, simplemente se mueve mientras haya espacio libre para hacerlo, pero se detiene o varía su dirección ante un obstáculo, evitando chocar con él.
- El coche, o robot, en caso de no poder avanzar, buscará una ruta alternativa que le permita su capacidad de direccionalidad, para tratar de estar siempre en movimiento.
- Se asume que no hay mala fe por parte del entorno, y que el vehículo podrá retroceder un poco durante sus maniobras para buscar una nueva orientación que le permita eventualmente seguir avanzado. Esto es, se supone que no hay un obstáculo móvil que inadvertidamente se coloque ex-profeso en su parte trasera cuando está maniobrando para evitar una esquina, o conjunto de elementos que impida continuar el avance, debiendo realizar giros de más de 90°.

Finalmente, a pesar de estas especificaciones básicas comunes, en el presente proyecto se realizarán dos implementaciones muy distintas debido a que se ejecutarán sobre dos prototipos que incorporan dos tipos de sensores diferentes: binarios (todo/nada) y analógicos. Así se verá cómo un cambio en la especificación individual de los sensores físicos a utilizar influye en gran manera en la selección o diseño del algoritmo que implemente esta funcionalidad, y que dará lugar a dos programas muy diferentes, con comportamiento muy particular sobre cada uno de los prototipos.

3.5. Control de velocidad adaptativo (ACC)

La funcionalidad básica del *Adaptive Cruise Control* ya fue especificada en 2005 [2], donde además de definirla como la mejora del convencional sistema de control de velocidad de crucero que permite a un vehículo seguir al que le antecede manteniendo una determinada distancia mínima, también se detectaron sus principales características. De entre ellas destaca el hecho de que precisa de la correcta combinación de varias unidades funcionales básicas, denominados módulos, tales como el módulo de control de motores, el de control de frenado y el módulo de instrumentación (medición de distancias) e información al conductor.

Como ya se descrito, el ACC básicamente consiste en un control de la velocidad del automóvil, con el cual se fija una velocidad máxima de circulación, pero al que se le dota de información adicional, mediante diversos sensores (generalmente radares o por láser), sobre la distancia a la cual se encuentra el coche que eventualmente circula justo delante del nuestro. Aunque no fue definido en su momento, en la actualidad se entiende que para que sea realmente válido, interesa que se sume otra funcionalidad, la de detección del carril por el que se circula, toda vez que esto es muy importante en el caso de curvas, para no confundir como obstáculo a un vehículo que circula por el carril aledaño.

A los objetos de este proyecto, se especifica la funcionalidad de la siguiente manera:

- El vehículo debe ser capaz de mantener una velocidad máxima predeterminada en el caso de que no existan vehículos ni obstáculos que impidan su marcha natural.
- El vehículo debe seguir un trayecto predefinido, para lo cual se le dotará de sensores y señalizaciones adecuadas al efecto. En las pruebas que se realizarán esa trayectoria se materializará en una línea negra pintada en el pavimento.
- El vehículo debe ser capaz de modificar su velocidad máxima de tal forma que evite alcanzar un obstáculo (móvil o fijo) que se interponga en su trayectoria.
- El vehículo debe seguir su trayectoria en caso de que sea posible, incluso en el caso de que un vehículo que le anteceda vaya a una velocidad menor o se cruce durante unos momentos, reduciendo su marcha si es necesario, para no alcanzarlo, manteniendo una distancia mínima de seguridad en dicho caso, y recuperando su velocidad máxima si el vehículo antecedente se aparta, o se aleja por acelerar su marcha.
- No se exige que el vehículo busque una ruta alternativa en el caso de que el obstáculo esté fijo en medio de la trayectoria prefijada.

CAPÍTULO 4

IMPLEMENTACIÓN

En el presente capítulo se procederá a detallar la implementación realizada de las funcionalidades especificadas en el anterior Capítulo 3 “Especificación de requisitos”.

Dado que se está ante un proyecto que consiste en la implementación de las funcionalidades en prototipos hardware reales, no sólo se debe discutir las particularidades de las implementaciones en su parte software, sino que también es muy importante describir el hardware sobre el que se ejecutarán tales implementaciones, especialmente por realizarse sobre dos prototipos con características diferentes.

Tras justificar la elección de la plataforma hardware (Arduino) para este proyecto, y describirlo, se pasará a describir las características de los vehículos autónomos utilizados, puesto que es la base sobre la que las implementaciones de la misma funcionalidad divergen, y posteriormente se detallará la implementación de cada una de las funcionalidades, a saber, el seguimiento de trayecto, el seguimiento de luz, la evitación de obstáculos y el control de velocidad adaptativo.

Así, para cada una de estas funcionalidades, excepto para el seguimiento de luz, existirán dos implementaciones diferentes según el prototipo utilizado, por lo que se justificará por qué razón los algoritmos diseñados en cada implementación son diferentes entre sí.

4.1. Plataforma hardware: Arduino

La plataforma seleccionada para las pruebas del presente proyecto ha sido Arduino [4][6], por haberse considerado la más idónea para iniciar la investigación experimental, por las razones que se explicarán más adelante.

A continuación se presentará dicha plataforma, se justificará su selección, y finalmente se presentarán los dos prototipos que se han utilizado en el presente proyecto para programar en ellos los algoritmos seleccionados para implementar las funcionalidades básicas que permiten dotar a estos prototipos de un básico sistema de control de velocidad adaptativo (ACC).

4.1.1. Arduino: la plataforma *open hardware* más extendida

Arduino es una plataforma de prototipado electrónico de código abierto (*open hardware*) basado en la flexibilidad y facilidad de uso de su hardware y software.

Al ser *open hardware* los diseños de las placas y de los componentes Arduino están a libre disposición, sin necesidad de pagar *royalties*. Por eso, cualquier manufacturador puede fabricar una copia de un diseño existente, y cualquier diseñador de prototipos y demás hardware puede basarse en ellos para realizar sus propios diseños. Esto ha permitido que sea una plataforma muy popular, dado que le ha permitido convertirse en una plataforma muy atractiva, accesible, eficiente, estandarizada y económica para iniciarse en el prototipado electrónico.

La plataforma hardware Arduino consiste en unas placas base (*motherboards*), unas placas de extensión (*shields*) y unos módulos con sensores y actuadores compatibles con ellos. Las placas bases y las de extensión permiten la interacción del microcontrolador con el mundo externo a través de un determinado número de conexiones (*pins*) que pueden ser analógicas (con diferente número de bits de resolución según modelos) o digitales (1 ó 0).

Históricamente, los microcontroladores pertenecen a la familia Atmel™, con alguna edición especial como el Arduino Yún que además tiene un coprocesador MIPS, pero últimamente se han diseñado nuevas placas basándose en otros "cerebros", incluso usándose microprocesadores, como los fabricados por ARM™. Se caracterizan por su baja capacidad computacional, pero con gran capacidad de conexionado mediante un número variable de pines digitales y analógicos, y que proporcionan comunicaciones variadas, tales como puertos asíncronos UART, síncronos USART, y buses comunes tales como IIC y SPI.

La placa base seleccionada ha sido una de las más genéricas, extendidas y económicas de la plataforma, la Arduino UNO R3, que integra el microcontrolador ATmega328 de Atmel y posee 6 entradas/salidas analógicas, y 14 entradas/salidas digitales, aunque se ha probado exitosamente también la Arduino MEGA para comprobar su compatibilidad. En cuanto a placas de extensión, principalmente se han usado las que extienden la capacidad de conexionado de sensores, *sensor shields*, concretamente la popular Sensor shield v5.0, y la diseñada especialmente por SunFounder™, la Sensor shield v5.1.

Pero Arduino no es sólo una plataforma hardware. De hecho está formada por cuatro componentes muy interrelacionados y que han potenciado su adopción general por su compromiso con el software libre (*free and open software*), a saber:

Plataforma hardware abierta Lo novedoso del proyecto Arduino es que incluso el hardware es abierto, usando componentes de gran disponibilidad y bien documentados, con facilidad de reprogramar su firmware (sobre todo los microcontroladores Atmel). Nace de un proyecto anterior, Wiring [11][12], aunque curiosamente, siendo ambos proyectos abiertos, existe cierta polémica entre ellos sobre la atribución de la originalidad de Arduino[10].

Lenguaje de programación propio El lenguaje también se denomina Arduino, y está basado en el lenguaje libre Processing [58]. Tiene la misma sintaxis que C y con algunas características de C++, a todo lo cual se le ha añadido la funcionalidad de fácil interacción con los componentes hardware de entrada/salida y de comunicaciones.

Software abierto y libre Se utilizan principalmente microcontroladores y componentes de los cuales se dispone la especificación, y que permiten reprogramar su firmware para poder extraer lo máximo de ellos. Además, exige una gran variedad de bibliotecas y de software de ejemplos libres escritos en Arduino para su uso directo o modificación.

Entorno de desarrollo abierto y libre Existe un IDE oficial, denominado Arduino también, basado en otro IDE libre preexistente, Processing [59], que utiliza la tecnología multiplataforma Java™.

4.1.2. Razones de la selección de la plataforma Arduino

Las razones de la elección de la plataforma Arduino como plataforma para la implementación experimental, han sido las siguientes:

- Alta disponibilidad en el mercado de diferentes prototipos ya directamente utilizables para el proyecto, lo que ha permitido seleccionar dos de ellos con diferentes características que permitieran diseñar y aplicar diferentes algoritmos para cada una de las funcionalidades a implementar. Concretamente, se han seleccionado dos prototipos (SunFounder™ y 4WD) muy diferentes entre sí, con código ya existente, pero el cual se ha demostrado insuficiente para cumplir las especificaciones de cada funcionalidad ya detalladas en el capítulo anterior.
- Es una plataforma de desarrollo con gran variedad de componentes, tanto a nivel de placas base con diferentes capacidades de cómputo y de gestión de diferente cantidad de entradas y salidas analógicas y digitales, como a nivel de módulos y sensores, lo que permite seleccionar para cada funcionalidad aquél que en principio parece más adecuado según las especificaciones de cada funcionalidad.
- Es una plataforma que incluye la posibilidad de incorporar componentes interconectados mediante buses industriales con gran adopción en el sector de la automoción, como es el caso del puerto serie (comunicaciones serie síncronas y asíncronas vía UART y USART) y buses serie industriales (CAN, LIN y FlexRay), además de los más genéricos (IIC y SPI).
- Es una plataforma *open hardware* continuamente en desarrollo, por lo que no sólo hay gran disponibilidad de componentes y módulos, sino que éstos pueden ser modificados y adaptados al caso concreto sin necesidad de pagar *royalties*.
- Cumple todos los requisitos que se enumeraron en Sección 3.1 “Especificación técnica de la plataforma hardware” para el hardware de producción, aunque probablemente en un rango de calidad algo limitado, a no ser que se busquen diseños especialmente pensados para su uso industrial:

  Plataforma genérica La plataforma cuenta con una gran aceptación en el campo del prototivado y también a nivel educativo. Al ser *open hardware* las especificaciones son abiertas, y por lo tanto, en continua evolución pero siempre respetando la plena compatibilidad entre módulos, que es lo que la hace especialmente atractiva.

  Normalización La gran variedad de componentes permite elegir aquellos que cumplan las normas concretas de cada aplicación. Al ser open hardware, incluso si hay algún problema concreto o variación/actualización de la norma a cumplir, siempre existe la posibilidad de modificar el diseño anterior (disponible por ser abierto y libre, generalmente libre de *royalties*) y distribuirlo entre diferentes suministradores sin problemas, en busca del precio más económico, la mejor calidad y el suministro más fiable.

  Especificaciones abiertas, no privativas Lo cumple intrínsecamente, al tratarse de una plataforma *open hardware*.

  Alta disponibilidad Plataforma muy extendida, y con diseño abierto, de tal forma que existen muchos fabricantes diferentes, que permite no estar atado a un único suministrador.

Elevada fiabilidad Es quizá uno de los puntos críticos a la hora de elegir (o no) Arduino. Para la realización experimental es suficiente, aunque se han encontrado problemas concretos en algunos módulos y componentes. No obstante, esta fiabilidad puede incrementarse seleccionando adecuadamente el suministrador, sobre todo si el cliente es una empresa de automoción, por el gran volumen de unidades que necesita. Así, puede exigir la utilización de subcomponentes de mayor calidad, y métodos de producción más profesionales que aseguren productos que sean fiables. No obstante, no se recomienda basarse en modelos baratos de uso educativo, sino que debería contratarse específicamente el rediseño con una buena garantía de calidad tanto en el diseño como en la producción.

Componentes económicos, baratos De nuevo, al ser diseño y especificaciones abiertas y haber tanta disponibilidad de componentes y variedad de fabricantes y suministradores, el precio de los componentes de Arduino son muy económicos, incluso a escala individual, no digamos ya adquiridos en gran volumen. Precisamente ése es otro de las claves del éxito de la plataforma en el sector educativo, recreativo y de prototipado.

Componentes ligeros En general, los componentes son sencillos, basados en la combinación de pocas partes simples, pero pueden combinarse para lograr diseños más complejos, incluso para la fabricación de unidades ya totalmente integradas. Téngase en cuenta que el diseño es abierto, y por lo tanto, con una alta capacidad de integración con alta densidad de componentes, a la vez que seleccionar sólo aquello que sea imprescindible, evitando que existan partes del todo que no se usen.

Componentes livianos Por la misma razón que la anterior, la posibilidad de integrar todo en una nueva placa o componente Arduino permite ajustar mucho el peso de la solución final.

Aparte de estas razones que justifican la adopción de la plataforma Arduino en este proyecto de investigación como en una posible futura implantación industrial real, existen las siguientes especificidades por las que Arduino se adapta muy bien al presente proyecto:

- Disponibilidad y variedad de prototipos ya conformados con forma y maniobrabilidad similar a los vehículos reales.
- Existencia de módulos y sensores directamente utilizables para implementar las funcionalidades deseadas: detectores de líneas, de iluminación, de detección de obstáculos, de gestión de motores y servomotores, sensores de diferentes tipo (sonidos, ultrasonidos, luz visible, luz infrarroja, etc.), y gran variedad de posibilidad de componentes de comunicación inalámbrica (infrarrojos, radio frecuencia, WiFi, Bluetooth, ...).
- Existencia de código en Arduino que implementa, al menos básicamente, algunas de estas funcionalidades, aunque de calidad muy variada y cuestionable.
- Amplia base de fuentes de documentación, conocimiento y tutoriales sobre la plataforma Arduino e incluso relacionados con la implementación de coches autónomos como los aquí presentados.

Aparte de ellos, también es interesante tener en cuenta para el seguimiento futuro del proyecto, que Arduino es multiplataforma (se puede desarrollar en Linux, en macOS™, Windows™ y en *BSD como *port*) y hay una amplia y muy dinámica comunidad de usuarios, destacando principalmente en el sector de la educación.

Una vez justificada la elección de la plataforma hardware Arduino, a continuación se describen los prototipos utilizados para el apartado experimental que permite probar los algoritmos que se presentarán en cada una de las secciones de cada funcionalidad.

4.1.3. Características de los prototipos utilizados

Se utilizan dos prototipos diferentes:

- SunFounder™: prototipo con estética y maniobrabilidad similar a un bólido de Fórmula 1.
- 4WD: prototipo con aspecto de todo terreno, auto-rotante sobre su centro.

La utilización de dos prototipos diferentes, basados en la misma plataforma, pero con diferentes componentes hardware (placas, módulos y sensores) y diferente configuración de maniobrabilidad, permitirá una mayor comprensión de la problemática a la hora de implementar cada una de las diferentes funcionalidades, dado que nos obligará a diseñar diferentes algoritmos para adaptarse a cada prototipo y aún así obtener un comportamiento funcionalmente equivalente entre ellos.

Además, esta diversidad de prototipos ha permitido lograr un mayor conocimiento y compresión tanto sobre la plataforma hardware utilizada (Arduino) como sobre los componentes y sensores utilizados, como también sobre las propias funcionalidades, al enfrentarnos con retos diferentes según el prototipo para conseguir un funcionamiento similar.

A continuación se describen las características básicas de cada uno de estos prototipos, haciendo especial hincapié en las importantes diferentes existentes entre ellos y así entender en los siguientes capítulos, al estudiar la implementación de cada funcionalidad en ellos, el porqué de las decisiones de diseño del algoritmo tomadas en cada caso.

4.1.3.1. Vehículo de tracción de dos ruedas y un único eje de giro (SunFounder™)

Este vehículo no tiene sólo una estética similar al de un bólido de Fórmula 1 (ver Figura 4.1), sino que también comparte su capacidad de maniobrabilidad, lo cual influye de gran manera en la implementación que debe realizarse de cada funcionalidad. En la Figura A.1 se muestra la configuración utilizada para la prueba del seguimiento de líneas.

Estas características básicas son:

- La maniobrabilidad del coche robot está restringida de la siguiente manera: dispone de dos ejes perpendiculares al sentido de marcha, con dos ruedas en cada eje. El eje trasero es fijo, y por lo tanto, siempre perpendicular al sentido de la marcha. El eje delantero puede rotar en torno a un punto fijo perteneciente a la mediatriz del eje fijo trasero. Esa rotación tiene un ángulo máximo (en ambos sentidos de izquierda/derecha) de rotación, y que es previamente conocido. Esto implica que existe un radio mínimo de giro que el coche puede girar.
- La maniobrabilidad del coche robot no es del tipo de Ackermann [34]. Esto es, existe una cierta tensión en la goma exterior de las ruedas respecto al firme por el que desliza, dado que el ángulo entre la rueda y la línea que une los dos centros de ambos ejes de ruedas es el mismo para la rueda delantera interior que para la rueda exterior. Esto hace que teóricamente el movimiento sea imposible, dado que la tendencia de ambas ruedas es de recorrer una circunferencia con el mismo radio, lo cual es físicamente imposible porque la distancia entre ambas ruedas está prefijada por su fijación al

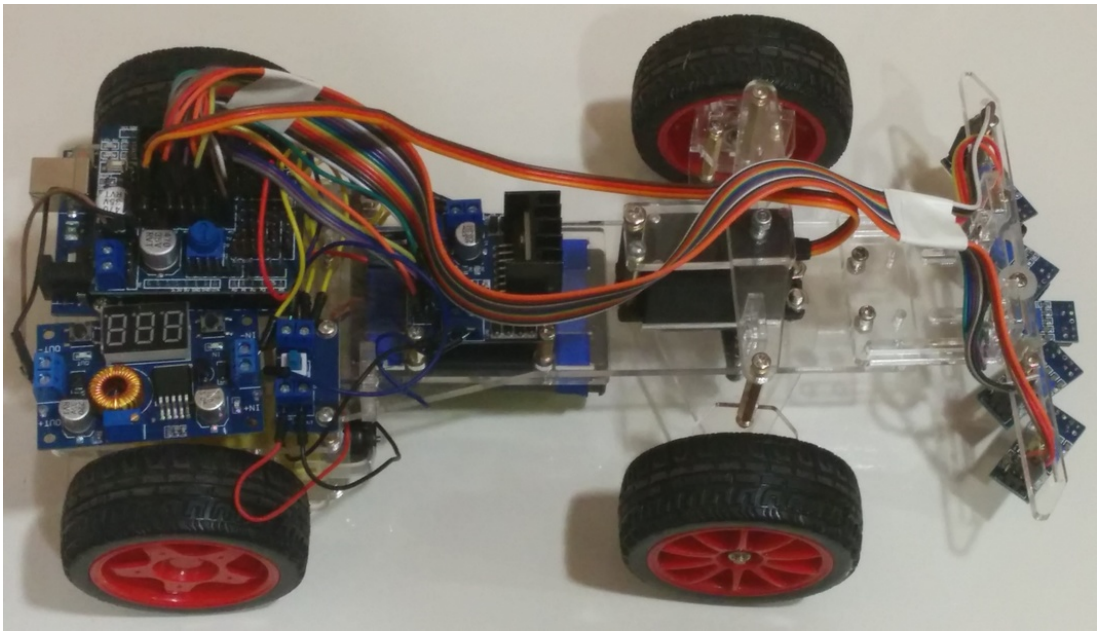

Figura 4.1: Prototipo SunFounder™ configurado para el seguimiento de líneas y el ACC

mismo eje delantero, y es evidente que dos circunferencias con el mismo radio con dos centros de giro diferentes nunca pueden ser paralelas, sino cortarse en dos puntos predefinidos por los anteriores parámetros dados (distancia entre ejes delantero y trasero, distancia entre ruedas exterior e interior y ángulo de giro, o del eje delantero). Esta tensión es resuelta físicamente mediante pequeñas fricciones superficiales en el contacto de las ruedas sobre el pavimento, y suponiendo ruedas iguales, es repartida de similar forma entre ambas ruedas, y por lo tanto, a cada ángulo de giro del eje delantero se le puede presuponer (e incluso calcular o estimar) un radio de giro global del vehículo correspondiente. Además tiene un comportamiento no-holonómico [71][70], esto es, para conseguir una posición determinada es necesario conocer no sólo su posición actual sino también la posición de los elementos que influyen en su movimiento (posición de las cuatro ruedas y de los ejes).

- Sólo las dos ruedas traseras, las del eje fijo, son automotrices.
- Sólo las dos ruedas delanteras, las del eje móvil, son las causantes del giro del vehículo.
- El vehículo puede hacer marcha atrás, siguiendo las mismas restricciones de movimiento.
- El vehículo dispone de dos tipos de sensores de detección de obstáculos, con las siguientes características:
  - Sensor móvil de luz visible, de larga distancia, con salida todo/nada, montando sobre un servomotor para dotarle de capacidad giratoria.
  - Dos o tres sensores fijos de luz de infrarrojos, de corta distancia, también de todo/nada.
- Los sensores se ubican en la parte frontal. No hay sensores laterales ni traseros.

Se trata de un prototipo comercializado por la empresa SunFounder™ y está formado por los siguientes componentes comunes básicos de Arduino:

- el chasis, parecido al de un bólido de Fórmula 1,

- la placa base Arduino UNO R3,
- la placa de expansión para sensores, versión 5.1, especial de SunFounder™,
- dos baterías 18650 en serie, de 3,7V cada una,
- un interruptor (*switch*) de alimentación,
- un regulador de voltaje de alimentación a 5V,
- un módulo para gestionar conjuntamente los dos motores DC que controlan las ruedas traseras,
- y el servomotor que gobierna la dirección del eje delantero.

A estos componentes, en función de la funcionalidad a implementar, se le añadirán otros, generalmente sensores, los cuales se indicarán en cada apartado.

4.1.3.2. Vehículo de tracción a las cuatro ruedas, fijas, con capacidad rotatoria (4WD)

La Figura 4.2 muestra el aspecto del 4WD con la configuración real usada para el control de velocidad adaptativo en las pruebas realizadas.

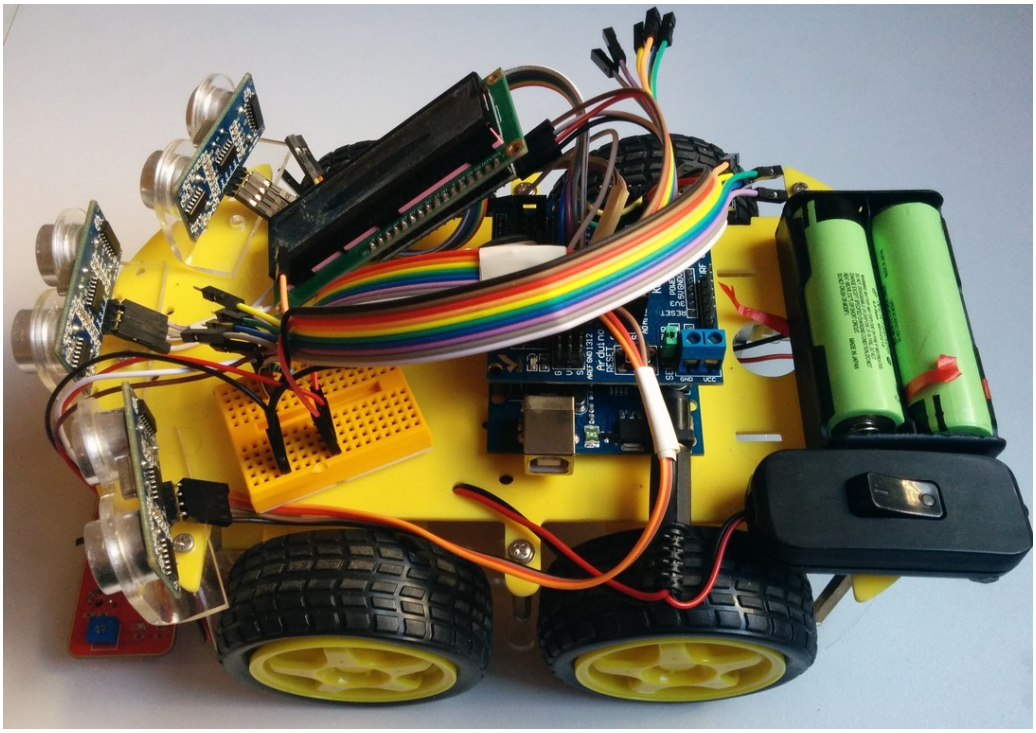

Figura 4.2: Prototipo 4WD configurado para el ACC, con bus IIC y sensores de ultrasonidos

La especificación general se especializa en los siguiente puntos:

- El vehículo dispone de cuatro rueda fijas automotrices, que son operadas simultáneamente de dos en dos, por una parte las del lado izquierdo, y por otra, las del lado derecho.
- A resultas de ello, el vehículo es capaz de rotar sobre su eje central, haciendo funcionar unas ruedas de un lado hacia delante, y las otras, hacia detrás, a la misma velocidad.

- También puede girar con radios mayores, pero poco definidos, variando la velocidad relativa entre las ruedas del flanco derecho y del izquierdo.
- Consta de un único sensor de distancia que proporciona un valor analógico cuantizable por un conversor digital a analógico (CDA) interno.
- El sensor está colocado en la parte delantera, y acoplado a un servomotor para dotarle de rotación en un arco de 0° a 180°, siendo 90° la dirección frontal de avance.

Estas características hacen que el vehículo pueda tomar decisiones más bruscas si son necesarias, al poder rotar sobre sí mismo, sin moverse del sitio, buscando una ruta alternativa libre de obstáculos sin riesgo de choque lateral ni trasero durante el escaneo.

Además, al ser el sensor analógico, se puede estimar la distancia a la que está el obstáculo, y por lo tanto, dar más tiempo de respuesta, y con mayor información de entrada, para seleccionar la nueva ruta.

Esto permite no tener necesidad de los detectores de obstáculos de proximidad, dado que el mayor tiempo de reacción teórico, debería permitirle actuar antes de llegar a necesitarlos. No obstante, sería una mejora en cuanto a dar mayor seguridad y robustez al algoritmo.

Las implementaciones realizadas sobre el 4WD tiene los siguientes elementos comunes básicos:

- el chasis, caracterizado porque las cuatro ruedas son fijas, sin que puedan moverse para cambiar la dirección del vehículo, y por tener una muy poca distancia entre los ejes que unen por pares las ruedas,
- la placa base Fundino UNO R3, que es prácticamente una copia de la placa Arduino UNO R3 original (recordemos que es *open hardware*, y por lo tanto, legal realizar copias de los diseños hardware),
- la placa de expansión para sensores, versión 5.0, estándar.
- dos baterías 18650 en serie, de 3,7V cada una,
- un módulo de control de dos motores, con el que se gestionan en realidad los cuatro motores DC que controlan cada una de las cuatro ruedas. Eso es posible porque cada salida de control se conecta a un par de motores: por una parte los de las ruedas derechas, y por otra las de las ruedas izquierdas. Por lo tanto, ambas ruedas derechas realizan las mismas acciones entre sí, de una forma independiente a las dos ruedas izquierdas, las cuales también están sincronizadas entre sí,

De nuevo, en función de la funcionalidad a implementar, a estos componentes básicos se le acoplarán otros, tanto sensores como algún servomotor para dotarlos de capacidad de giro si es conveniente.

4.2. Seguimiento de un trayecto definido por una línea

El hecho de que el seguimiento de un trayecto se realice a través de una línea pintada en el suelo es por conveniencia en estas pruebas iniciales del proyecto DINAMOS. No obstante, en las actuales tecnologías sí que se suele incluir un seguimiento similar, consistente en la detección de los carriles viales pintados en la carretera.

El seguimiento de líneas se implementará en los dos prototipos, por tratarse de una funcionalidad fundamental para cualquier sistema de navegación, sea lo simple o avanzado que sea. De hecho, para la implementación del ACC se verá que es interesante contar con dos prototipos para realizar las pruebas de forma más real.

Las implementaciones, además, tienen características diferentes, no sólo porque se deben adaptar a las características propias de cada prototipo, tales como su maniobrabilidad, sino también porque se implementan basándose en funcionamientos diferentes de los sensores: aunque los sensores son analógicos, en un algoritmo (el implementado en el SunFounder™) se tratarán como tales, pero en el otro (el implementado en el 4WD) se tratan como si fueran binarios, todo/nada.

4.2.1. Coche SunFounder™

El diagrama de conexionado para el coche SunFounder™, para la código versión v7, se muestra en la Figura A.1.

4.2.1.1. Especificaciones hardware

A los elementos base del SunFounder™ se le incorporan cinco módulos detectores de línea. Estos módulos funcionan por infrarrojos, utilizando el sensor TCRT5000, que dispone de un LED que emite una luz en el espectro de los infrarrojos y también de un receptor también de infrarrojos, que detecta la reflexión de la luz del LED. Dispone de un potenciómetro que permite ajustar la sensibilidad del sensor, comparando mediante un chip comparador LM393 ambas señales (la emitida y la recibida). De esta forma se puede ajustar el contraste existente entre la línea negra que debe seguirse y el resto del fondo sobre el que se desplaza el prototipo.

Los cinco sensores se ubican en la parte delantera, y la distancia entre ellos puede ajustarse según el grosor de la línea a seguir. Para seguir una línea de unos 3 cm se ha encontrado conveniente reducir la distancia original de estos sensores.

En la implementación realizada, para seguir líneas de entre 2 y 4 cm, los sensores se han ubicado simétricamente, con el tercero en el centro. El segundo y el cuarto (dispuestos entre el central y los dos extremos), están cada uno a 13 mm del central, y los dos extremos a 17 mm de éstos intermedios, o lo que es lo mismo, a 30 mm del central. De esta forma, la distancia entre los dos extremos más alejados es de 60 mm. Recordemos que 90° representa la marcha perfectamente frontal.

Cada uno de estos sensores se conecta a un pin digital de la placa Arduino UNO R3. Ese pin digital no hace falta que sea del tipo PWM, los cuales conviene guardar para otros usos como control del ángulo de giro de los servomotores, o la velocidad de giro de los motores DC.

4.2.1.2. Algoritmo con sensores analógicos

El algoritmo que se presenta en este trabajo es una modificación muy elaborada del código original proporcionado por SunFounder™, variando cosas sustanciales como la forma de asignar el valor a cada uno de los sensores, la variación de velocidad en función del estado, la detección de pérdida de línea, y realización de maniobra parcialmente aleatoria para la recuperación de la línea perdida, que incluye retroceder si es necesario para intentar recuperar la trayectoria en el mismo punto en que se perdió.

4.2.1.2.1. Descripción general El algoritmo va en todo momento monitorizando los valores leídos por cada uno de los sensores de seguimiento de línea. Los valores reales actuales se utilizan para actualizar el valor asignado a cada sensor a través de la "media móvil exponencial" [69] lo que implica tener una acumulación de sus variaciones históricas. Según la elección del parámetro `beta` (β) se controlará la importancia relativa entre los valores históricos y los valores más recientes, esto es, podrá ser más reactivo a los cambios o más opuesto al cambio de dirección.

Una vez se dispone de los valores asignados a cada sensor, éstos se ponderan, asignando valores negativos a los sensores de la izquierda y valores positivos a los derecha. Además, cuanto más al extremo estén, mayor será su valor absoluto, y por lo tanto, más afectarán al comportamiento del vehículo. El valor ponderado así obtenido permitirá seleccionar el ángulo de giro del eje delantero. Cuanto más negativo sea, es indicación de que la línea tiene más tendencia a detectarse por la izquierda, y por lo tanto, más hacia la izquierda deberá girar el eje. Por contra, cuanto más positivo sea el valor ponderado, el eje deberá girar más hacia la derecha, puesto que la línea se está detectando preferentemente en esa dirección.

Además, se añade un control de velocidad en función del ángulo de giro del eje. Si la marcha es frontal, la velocidad asignada es la máxima predeterminada, pero cuanto más deba girar el eje hacia la derecha o hacia la izquierda, más disminuirá dicha velocidad, hasta llegar a una velocidad mínima cuando el eje esté girado lo máximo posible hacia un lado. Esto se hace así porque en las curvas interesa tener un mayor control sobre la evolución de la trayectoria de la línea seguida, y conviene disminuir la velocidad no solo para que haya mayor agarre al pavimento en curvas e impedir posibles derrapes, sino sobre todo, para que el espacio recorrido por el coche entre dos lecturas consecutivas de los sensores (que incluye también el tiempo de procesamiento, cálculo y decisión utilizado por el microprocesador) sea el mínimo posible, y exista menos probabilidad de pérdida de la línea por tratarse de una curva excesivamente cerrada, por ejemplo.

Por otra parte, también se incluye una detección de pérdida de línea. Se considera que la línea se pierde cuando ninguno de los sensores detecta zona negra. Para evitar microcortes en la línea se tienen tres mecanismos simultáneos: se requiere un mínimo de iteraciones en las que se ha dejado de percibir la línea; hay un valor umbral mínimo que el valor asignado a cada sensor debe rebasar para considerar que se ha perdido la línea; y la velocidad se disminuye para que durante la exploración en busca de la continuidad de la línea se realicen más comprobaciones por unidad de distancia recorrida.

Una vez se toma la decisión de que la línea se ha perdido, se inicia una maniobra especial consistente en retroceder durante una determinado tiempo a baja velocidad (que implica una distancia hacia atrás a recorrer), y a partir de ahí se continua el retroceso mientras no se detecte línea (teniendo en cuenta que esa detección depende de un umbral mínimo del valor de la media móvil exponencial de cada sensor, y por lo tanto, una detección de varias veces seguidas para lograr superar ese umbral). Para evitar ciclos de avance y retroceso, cada cierto número de iteraciones en retroceso se cambia aleatoriamente el ángulo de giro, para generar ruido que rompa ciclos de detección/pérdida de línea, e incrementar la probabilidad de encontrar una línea (esto es, se genera una especie de movimiento browniano durante esa búsqueda).

4.2.1.2.2. Adaptación específica al prototipo En la Figura 4.3 se muestra el diagrama de flujo de este algoritmo, donde también aparecen los parámetros de la implementación realizada.

Es importante la elección de los parámetros que influyen en el comportamiento específico del algoritmo a cada situación: no es lo mismo seguir líneas estrechas o amplias, o que el prototipo tenga más o menos sensores, o más o menos maniobrabilidad.

A continuación se describen dichos parámetros, así como se analizan los criterios básicos para la elección de su valor en función de comportamiento deseado, para poder ajustarlo al entorno real existente.

Umbral para binarización de las lecturas reales El algoritmo parte de la base de que el valor real de la lectura actual es del tipo todo/nada, esto es, 1/0. Si los sensores son analógicos, como es el caso, éstos proporcionan valores comprendidos en un intervalo, con un número dado de bits de resolución. Por ello deben binarizarse, lo cual se realiza definiendo un umbral (*threshold*) y según si el valor cuantizado es mayor o menor que éste, se asigna un 1 ó un 0.

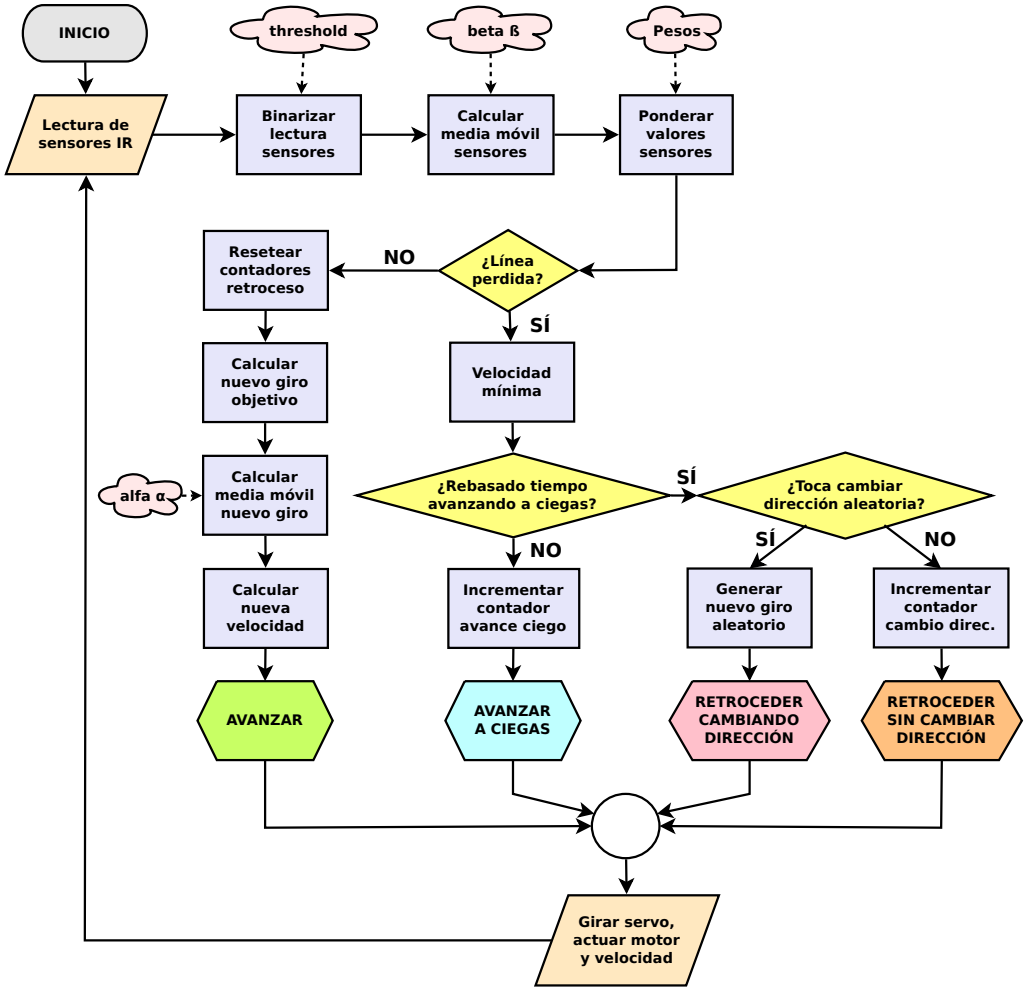

Figura 4.3: Diagrama de flujo del algoritmo de seguimiento de línea para el SunFounder™

El requisito de que el valor asignado sea 1/0 (todo/nada) tiene relación con el hecho de que se está detectando, o no, la presencia de una línea negra. Para ello existe un potenciómetro que regula físicamente la sensibilidad ante la luminosidad infrarroja. Esta sensibilidad se tiene que ajustar previamente según el contraste existente entre la línea negra y el resto de pavimento en torno a ella.

Parámetro β Es el valor por el que se pondera relativamente el valor previo asignado (acumulado histórico) con el nuevo valor real (leído por el sensor). A mayor β, mayor importancia se le dan a los valores recientes (y por lo tanto es más reactivo a los cambios), mientras que a menor β, es más importante la historia (y por lo tanto es más resilente al cambio de dirección). Este parámetro depende de la distancia entre sensores, la amplitud de la línea a seguir y el tiempo medio transcurrido entre lecturas consecutivas de los sensores.

Pesos de ponderación Son los valores (pesos) que sirven para ponderar los valores históricos de cada sensor. Existen varias estrategias, pero la que se considera más apropiada, por las pruebas realizadas, es la de asignar mayor peso a los extremos, y decrementarlos conforme se aproximen al centro, cuyo peso suele ser cero. Para poder determinar la deriva de la línea hacia uno u otro flanco, se utilizan pesos negativos para un lado, y positivos para el contrario. Si los sensores están ubicados simétricamente respecto al eje longitudinal del vehículo, los pesos se asignan simétricamente también. Sus valores

relativos dependerán del número de sensores, de cómo éstos estén distribuidos en la parte frontal del vehículo, teniendo en consideración sobre todo la distancia entre ellos y la ubicación relativa más avanzada o atrasada de sus LED, y de la amplitud de la línea a seguir. Su valor absoluto conviene seleccionarlo de forma que permita utilizar operaciones computacionales más livianas (trabajar con operaciones enteras de registros con pocos bits, pero suficiente resolución, `signed char` (8 bits) o `short int` (16 bits), por ejemplo.

Relación entre valor ponderado de los sensores y el ángulo de giro del eje delantero El valor ponderado de los valores de los sensores calculados mediante la "media móvil exponencial" proporciona un valor negativo si la mayor parte de la línea detectada está desplazada hacia la izquierda del frontal del vehículo, mientras que el valor positivo indica que la línea está hacia la derecha. Por lo tanto, cuando el valor es negativo el coche debería girar hacia la izquierda para seguir la línea, y si es positivo hacia la derecha. Cuanto mayor sea en valor absoluto, mayor tiene que ser el ángulo de giro del eje delantero hacia el costado adecuado. En esta implementación se propone una relación lineal: el valor mínimo (máximo absoluto con signo negativo) que pueda proporcionar la ponderación hará girar al máximo hacia la izquierda el eje ($90° - 30° = 60°$), mientras que el valor máximo positivo lo girará al tope hacia la derecha ($(90° + 30° = 120°$). No obstante, la variación no será radical, sino también amortiguada por el mecanismo de la media móvil exponencial.

Parámetro α Se utiliza de forma similar al parámetro β, pues es el parámetro de otra "media móvil exponencial". En este caso sirve para que el servomotor no esté variando muy bruscamente el ángulo de giro del eje delantero. Así, dicho ángulo también está sujeto a una evolución en función de la media móvil exponencial entre el valor acumulado anterior y el valor del ángulo γ calculado según la relación definida justo en el punto anterior.

Variación de la velocidad de marcha La velocidad de marcha también es dependiente de la ubicación del coche sobre la línea, o más concretamente, del ángulo del eje delantero. Si es próximo a $90°$ (dirección frontal) la velocidad es más próxima a la máxima preasignada, pero si el ángulo de giro respecto a la marcha frontal es mayor, la velocidad decrece proporcionalmente hasta una velocidad mínima predefinida.

Número de iteraciones para considerar que se ha perdido la línea Existe un parámetro que permite indicar cuántas veces deben estar todos los lectores por debajo del umbral de detección de línea para considerar que se ha perdido la línea que se estaba siguiendo, y que por lo tanto hay que pasar al procedimiento de recuperar la línea. Si ese número es excesivamente bajo, pequeñas discontinuidades en la línea, o algún reflejo extraño en ella que haga variar su oscuridad, hará que el algoritmo pase a la etapa de recuperar la línea. Si es excesivamente alto, el vehículo recorrerá mucha distancia antes de que lo advierta, aunque vaya a su mínima velocidad, por lo que será más difícil recuperar la línea perdida. Además, especialmente si hay varias líneas, se corre el problema de que detecte otra línea diferente a la que se estaba siguiendo, cambiando por lo tanto de trayectoria, lo cual debe evitarse. Otro peligro es que aunque la trayectoria sea la misma, no se recupere en el mismo punto, sino en otro diferente e incluso en el peor de los casos, que lo haga de una forma tal que cambie el sentido de marcha dentro de la misma trayectoria.

Parámetro de aleatoriedad Se utiliza en la tarea de recuperación de la línea perdida. Define el número de iteraciones seguidas sin haberse detectado la línea mientras va buscándola marcha atrás. Por lo tanto, está relacionado directamente con el tiempo transcurrido y el espacio recorrido durante esa maniobra. Alcanzado ese número de iteraciones, se cambia el ángulo de giro del eje delantero de

forma aleatoria, para continuar retrocediendo en esa nueva dirección durante otras tantas iteraciones mientras no se detecte una nueva línea.

4.2.1.2.3. Formalización computacional En este apartado se procede a formalizar las ecuaciones que se implementarán en el programa en Arduino.

El valor de cada sensor es un número real que se sitúa siempre en el intervalo $[0,1]$, el cual va modificándose utilizando la ponderación de la "media móvil exponencial", por el que se pondera el valor anterior con el nuevo valor leído (con un valor binario de 1 si detecta línea negra, ó 0 si no). El factor de ponderación es `beta` (β) según la siguiente fórmula:

$$x_i^{new} = x_i^{old} \cdot (1-\beta) + x_i^{read} \cdot \beta, \quad \text{siendo} \begin{cases} \beta \in]0,1] \wedge \beta \in \mathbb{R} \\ x_i^{new} \in [0,1] \wedge x_i^{new} \in \mathbb{R} \\ x_i^{old} \in [0,1] \wedge x_i^{old} \in \mathbb{R} \\ x_i^{read} = 0 \oplus x_i^{read} = 1 \end{cases}$$

Cada sensor tiene un peso, dependiendo de su ubicación, que es parametrizable igualmente. Así, los sensores extremos (x_1 y x_5) tienen un peso mayor (w_1 y w_5, por ejemplo de valor ± 30), los intermedios (x_2 y x_4) un valor menor (pesos w_2 y w_4, por ejemplo de ± 15), y el central (x_3) no cuenta, pues se le asigna un peso de cero ($w_3 = 0$). El flanco derecho pondera sumando, y el izquierdo, restando. La fórmula que pondera el resultado de los sensores ($\bar{x}$), nombrando a los sensores (x_1, x_2, x_3, x_4, x_5) de izquierda a derecha, es la siguiente:

$$\bar{x} = \sum_{i=1}^{n=5} (w_i \cdot x_i) = w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + w_4 \cdot x_4 + w_5 \cdot x_5$$

Si asignamos $w_1 = -30$, $w_2 = -15$, $w_3 = 0$, $w_4 = +15$ y $w_5 = +30$, entonces:

$$\bar{x} = -30 \cdot x_1 - 15 \cdot x_2 + 15 \cdot x_4 + 30 \cdot x_5$$

Según el valor ponderado obtenido, se selecciona el ángulo del servomotor que deberá girar el eje delantero. Así, cuanto mayor sea el valor, más girará el eje hacia la derecha, hasta llegar al máximo posible (en torno a $90° + 30° = 120°$), mientras que cuanto menor sea (esto es, valores más negativos) se girará el eje hacia la izquierda, hasta llegar al máximo posible en ese otro sentido (en torno a $90° - 30° = 60°$). Para ello se utiliza la siguiente fórmula, en donde se observa que el valor ponderado $\bar{x}$ se divide por n, que es el número de sensores que en ese momento han detectado la línea negra, lo que permite un incremento gradual de los pesos conforme la línea se aleja más de la marcha frontal.

$$\gamma^{calc} = \gamma^{old} - \frac{\bar{x}}{n}$$

Hay que tener en cuenta que existe una relación entre el ángulo del eje (γ) y el valor histórico ponderado de los sensores ($\bar{x}$). Es por ello que los valores de los pesos se seleccionaron como ± 30 y ± 15: si sólo se detecta la línea en el sensor del extremo derecho, $w_5 = 30$, y si esa situación se prolonga varias iteraciones, entonces $x_5 \to 1$, por lo que $\bar{x} \to 30$. Como en ese caso sólo hay un sensor detectando la línea, $n = 1$, por lo que $\frac{\bar{x}}{n} \to 30$, y consiguientemente se tenderá a obtener el giro máximo posible hacia la derecha, de $120°$.

El ángulo de giro también se actualiza en cada iteración también mediante la técnica de la media móvil exponencial, tanto si proviene del cálculo ponderado mientras se avanza, como si procede de un valor aleatorio cuando retrocede:

$$\gamma^{\texttt{new}} = \gamma^{\texttt{old}} \cdot \left(1-\beta\right) + \gamma^{\texttt{calc}} \cdot \beta, \quad \texttt{siendo} \begin{cases} \beta \in]0,1] \wedge \beta \in \mathbb{R} \\ \gamma^{\texttt{new}} \in [60^\circ, 120^\circ] \wedge \gamma^{\texttt{new}} \in \mathbb{R} \\ \gamma^{\texttt{old}} \in [60^\circ, 120^\circ] \wedge \gamma^{\texttt{old}} \in \mathbb{R} \\ \gamma^{\texttt{calc}} \in [60^\circ, 120^\circ] \wedge \gamma^{\texttt{calc}} \in \mathbb{R} \end{cases}$$

4.2.1.2.4. Comentarios sobre el algoritmo Con las fórmulas anteriores se logra tener siempre en cuenta todas las lecturas de los cinco sensores, previamente ponderadas temporalmente mediante la técnica de la "media móvil exponencial". Se da más importancia a que los sensores extremos detecten la línea a que lo hagan los intermedios, y sobre todo el central, que no aporta peso alguno a la ponderación.

El valor del sensor central (línea o no línea) no influye en la dirección de avance, pero sí que es considerado para el caso de que ningún sensor detecte ninguna línea, esto es, se ha perdido el seguimiento de la línea. En ese momento, el coche en lugar de seguir avanzando, retrocederá hasta que recupero alguno de sus sensores la visión de una línea negra.

Durante el periodo de retroceso en búsqueda de la trayectoria perdida, el eje delantero gira un número aleatorio de grados entre el máximo a la derecha y el máximo a la izquierda (típicamente entre 60° y 120°). La razón para que la dirección de retroceso sea aleatoria es por la comprobación experimental de que así se consigue evitar ciclos cerrados de avance/retroceso por estar siempre buscando en la misma zona, y por lo tanto, detectando y perdiendo siempre el mismo trozo de línea. Con un movimiento aleatorio durante la marcha atrás, se permite incrementar la probabilidad de que en algún momento el coche se alinee de una forma correcta tal que permita que al siguiente avance se detecte una porción de línea significante y con un radio suficiente para que aún la baja maniobrabilidad de este tipo de prototipo pueda retomar la trayectoria salvando una eventual curva cerrada, que es donde más problemas se han detectado durante las pruebas experimentales.

El otro caso interesante es el de cruce de líneas. En ese momento, hasta los cinco sensores pueden estar detectando zona negra. Pero como los pesos de la parte izquierda y la derecha tienen signo opuesto, algebraicamente se compensan, de tal forma que en el caso de detectarse una intersección a la que se incida ortogonalmente, la tendencia es la de continuar con la dirección original de la marcha, y no girar a uno u otro lado. Además, el valor de `n` incrementa, y por lo tanto, aunque haya una pequeña variación, ésta se ve reducida al dividirla por `n`. Como se aplica la "media móvil exponencial", en caso de que el cruce no sea totalmente ortogonal es cierto que puede generarse cierta tendencia hacia el lateral que antes se detecte, pero su mayor peso no entra en efecto hasta pasadas ciertas iteraciones, puesto que debe incrementar desde un valor próximo a cero hasta 0,5 para que tenga el mismo peso que el que proporciona el sensor intermedio del lado opuesto, lo que ralentiza la primera tendencia a dar mayor importancia a la recién descubierta línea lateral del cruce oblicuo.

4.2.2. Coche 4WD, con cuatro ruedas motrices y capacidad auto-rotante

El código de la versión v4 se ejecuta en el 4WD con la configuración y conexionado que se muestra en la Figura A.2.

4.2.2.1. Especificaciones hardware

A los elementos base del prototipo 4WD se le añade un módulo con tres sensores detectores de línea por infrarrojos. El módulo se comporta en la práctica como tres sensores individuales, sólo que en lugar de haber sido físicamente separadas sus placas de circuito impreso (PCB, *Printed Board Circuit*), éstas se han mantenido juntas de tres en tres. Por lo tanto, son funcionalmente equivalentes a los sensores utilizados en el SunFounder™, y ya descritos en el Punto 4.2.1.1.

Este módulo de tres sensores se ubica en la parte delantera central, y su distancia es 14 mm. Cada uno de los sensores se conecta a un pin de la placa base Fundino UNO R3 configurado como de entrada.

4.2.2.2. Algoritmo con sensores todo/nada

El programa localizado por Internet era simple y funcional. No obstante se realizaron dos modificaciones para mejorar el comportamiento en las pruebas realizadas. La primera, mínima, fue la de incorporar variación de velocidad según la localización de las líneas. La segunda, más relevante, fue la de dotarlo de un mecanismo similar al caso anterior con el SunFounder™ para que fuera capaz de recuperar la línea perdida, caso que no tenía implementado el algoritmo original localizado del 4WD.

La sencillez del algoritmo es debida a la alta maniobrabilidad que presenta este prototipo, puesto que puede responder fácilmente ante una curva muy cerrada utilizando su posibilidad de rotar en torno a su propio centro.

Una importante diferencia con el algoritmo del SunFounder™ es que en este caso los sensores, a pesar de tener salida analógica (10 bits en Arduino), se tratan como digitales, puesto que su valor se lee con `digitalRead()`, lo que redondea a 0 ó 1 con el umbral en 512 (esto es, toma el valor del bit más significativo de los 10 que proporciona).

4.2.2.2.1. Descripción general En la Figura 4.4 se presenta el diagrama de flujo de este algoritmo, que se utilizará en el prototipo 4WD.

El algoritmo se basa en la monitorización continua de los valores leídos por los detectores de línea. Mientras estas lecturas sean constantes, el vehículo sigue moviéndose con los mismos parámetros de acción de las ruedas y su velocidad.

Si los sensores detectan alguna variación, se analiza su estado conjunto. Hay dos casos básicos, a saber, que el sensor central detecte línea o que no la detecte. Si la detecta, se seguirá con el avance, pero se deberá girar en el caso de que sólo uno de los dos sensores laterales no detecte la línea negra. El giro será hacia el lado opuesto del sensor que no la detecte. El otro caso es que el sensor central no detecte la línea negra, y se subdivide en tres posibilidades: si ninguno de los otros dos detecta línea, entonces la línea se ha perdido y debe procederse a recuperarla; si los otros dos sí que la detectan, entonces es un caso extraño, pero posible en la realidad, y el coche deberá seguir avanzando, pero precavidamente, a menor velocidad; finalmente si sólo hay un sensor que detecte la línea, el coche deberá girar más bruscamente hacia ese lado, para no perderla rápidamente.

Para recuperar la línea perdida se utiliza un sistema parecido al del algoritmo anterior del SunFounder™, pero simplificado: simplemente se retrocederá hasta que algún sensor recupere la línea, y a partir de ahí se seguirá retrocediendo un poco más hasta alcanzar un determinado mínimo de detecciones consecutivas. Esto se hace así para evitar señales espurias, y asegurarse de que realmente hay una alta probabilidad de que se trate de una línea negra a seguir. Cuando esto se alcanza, el vehículo vuelve a avanzar, y su dirección será en función de qué sensores estén detectando la línea negra en ese momento, según el procedimiento descrito en el párrafo anterior.

No se considerado necesario incorporar el movimiento aleatorio mientras se retrocede, dado que la pérdida de la línea es un caso más extraño en este algoritmo debido a que se usa en un prototipo con gran maniobrabilidad, y por lo tanto, con menores restricciones a la hora de poder seguir las líneas trazadas.

4.2.2.2.2. Adaptación específica al prototipo El algoritmo anterior es básico y simple gracias a que se utilizará en un prototipo de gran maniobrabilidad. Las adaptaciones al coche concreto serán en función de

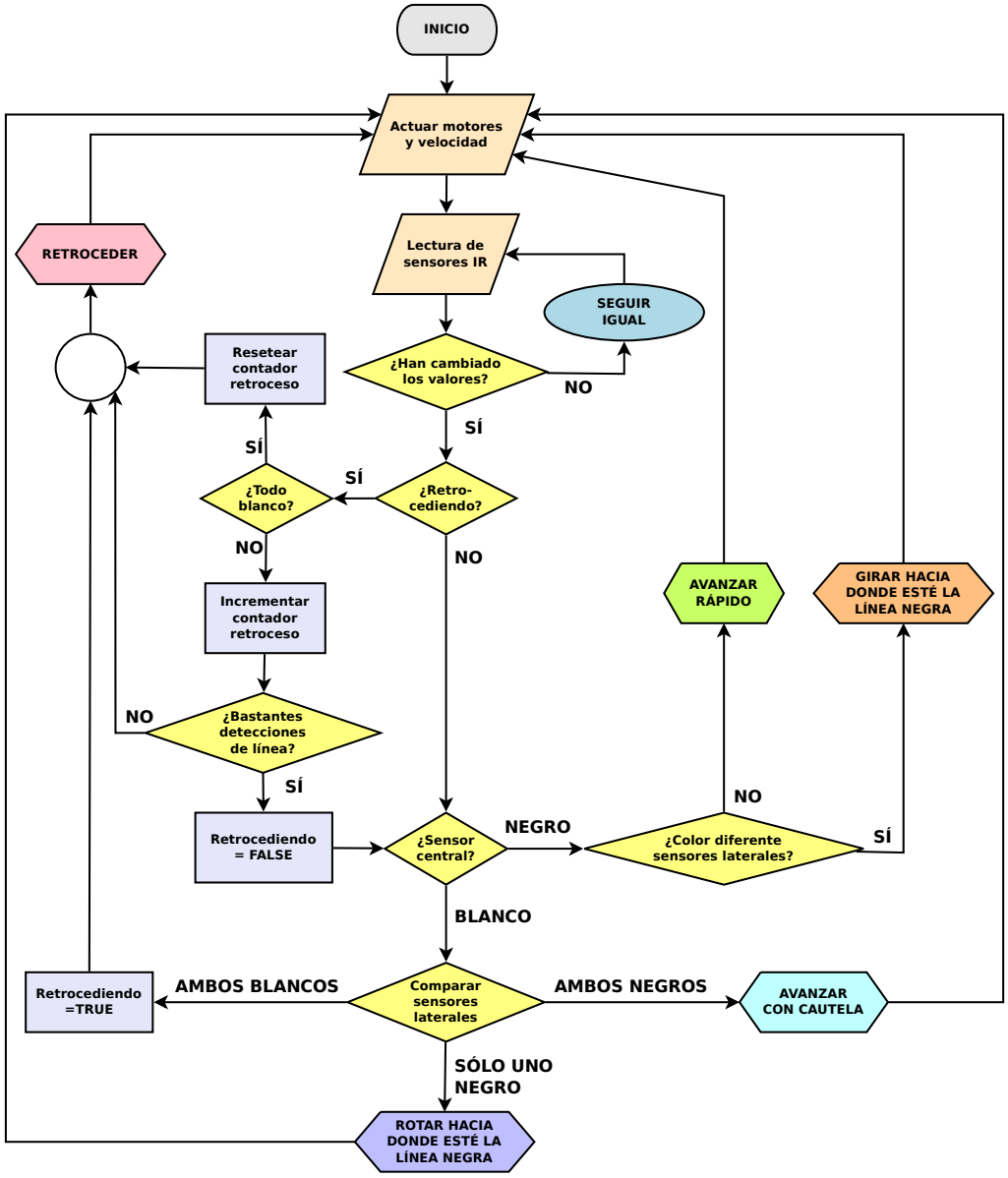

Figura 4.4: Diagrama de flujo del algoritmo de seguimiento de línea para el 4WD

la velocidad que pueda alcanzar las ruedas, la separación entre los ejes fijos de las ruedas, y el ancho de las líneas negras a seguir, básicamente.

Para el caso concreto del prototipo 4WD son las siguientes:

Velocidades de los motores Se asigna una velocidad alta para la marcha adelante, y una velocidad baja para la marcha hacia atrás. En el caso de los giros, cuando estos son suaves (parando las ruedas de un lado y girando sólo las del otro), su velocidad es alta, para permitir generar el suficiente momento angular para girar razonablemente el vehículo. En el caso de las rotaciones, aunque podría hacerse que la velocidad de los motores fueran las mismas, pero en sentidos contrarios, ello haría que el vehículo rotara exactamente sobre su centro geométrico. Esto implica que el ángulo girado por unidad de tiempo puede ser muy grande, además de no avanzar el vehículo, por lo que se puede perder rápidamente la línea, aunque ésta esté escorada. Por eso se ha considerado más oportuno que las ruedas que

marchan hacia atrás, que es hacia el lado que rotará el vehículo, lo hagan a una velocidad inferior a la de avance de las ruedas del lado opuesto. Esto genera una rotación, pero con un ligero movimiento de traslación hacia delante, que permite que el vehículo siga teniendo cierta tendencia a seguir su marcha frontal, aunque girando mucho. Para otros casos, la velocidad relativa entre ambas ruedas, es un parámetro adicional que permite adaptarse a la naturaleza del terreno, lo cerrado que sean las curvas y la anchura de las líneas.

Iteraciones mínimas retrocediendo antes de volver a buscar la línea perdida Cuanto mayor sea el número de iteraciones del ciclo `loop()` de Arduino que esté retrocediendo antes de volver a comprobar el estado de sus sensores tras haber perdido la línea, mayor será el espacio recorrido hacia atrás. Este valor deberá ajustarse según las características del circuito.

Iteraciones mínimas seguidas con detección de línea antes de considerar que la línea se ha recuperado Al igual que en el caso anterior, depende del caso particular. Es interesante incrementarlo cuando no hay mucho contraste entre la línea y el fondo, o cuando existen motas negras en el pavimento que pueden confundir al sensor. También es interesante cuando hay cruces de línea, puesto que permite que el algoritmo disponga de más información futura cuando deba volver a avanzar.

4.3. Seguimiento de luz

El diagrama de conexionado para el coche SunFounder™, para la código versión v5, se muestra en la Figura B.1.

Esta funcionalidad sólo se implementa en el SunFounder™, por ser previsiblemente la menos interesante para las futuras líneas de trabajo y de investigación. Además, la traslación al 4WD parece, en un principio, trivial, simplemente necesitando cambiar la forma de actuar sobre los motores para conseguir los giros deseados.

4.3.1. Coche SunFounder™

SunFounder™ proporciona un código pero está pensado para ubicar tres sensores delanteros y sólo uno trasero, de tal forma que cuando retrocede, lo hace sin poder saber si está más iluminado su flanco trasero izquierdo o el derecho. Es por ello que se modifica esa disposición, y tal como se muestra en el diagrama de conexionado para la versión v7 de la Figura B.1, se ubican dos sensores de luz delanteros y dos traseros. Este cambio permitirá maniobrar el coche también al realizar marcha atrás, en función de dónde se localice la máxima luminosidad.

El código de SunFounder™ no funcionaba correctamente, y los indicadores de luz tampoco mostraban la información correcta. Se encontró un error en el conexionado de los módulos, que también impedía el funcionamiento correcto de alguno de éstos. El esquema de la Figura B.1 ya corrige esas deficiencias.

4.3.1.1. Especificaciones hardware

Al conjunto básico del prototipo SunFounder™ se le añaden cuatro sensores analógicos de luz basados en termorresistores (detectores de temperatura resistivos), que además disponen de un indicador de la luminosidad leída basada en una barra de LED que se iluminan conforme el sensor detecta más luz. Además, su sensibilidad puede regularse mediante un potenciómetro físico, pero no mediante programación.

4.3.1.2. Algoritmo

El algoritmo que se presenta es muy simple, pero funciona correctamente en las pruebas realizadas, independientemente del grado de luminosidad u oscuridad del entorno.

4.3.1.2.1. Descripción general Se mide continuamente la luminosidad existente en torno al vehículo gracias a los sensores de luz, realizando varias medidas para obtener un valor medio significativo para cada sensor. A continuación se comparan entre sí. Primero se determina qué sensor recibe más luminosidad, en función de eso se seleccionará el sentido del movimiento (hacia atrás o hacia delante), siempre que la diferencia de luminosidad sea significativa, para lo que se predefinirá un umbral mínimo de diferencia para ello. Posteriormente se compara el valor del sensor más iluminado con su pareja delantera o trasera, según el caso, y dependiendo de la diferencia relativa entre ambos, se selecciona un ángulo de giro para el eje móvil delantero.

El algoritmo requiere que el vehículo al inicio esté en una zona con luminosidad homogénea, dado que se realiza un proceso de auto-calibración de los sensores.

4.3.1.2.2. Adaptación específica al prototipo Los sensores utilizados disponen de un potenciómetro para regular su sensibilidad, pero es evidente que es muy difícil calibrarlos para que den valores significativamente iguales.

Es por ello que se idea una forma de auto-calibración que permite que independientemente de las condiciones lumínicas del entorno, de su evolución, y de la diferente sensibilidad de los sensores (siempre dentro de un mínimo de ajuste básico manual), en todo momento se puede realizar una comparación cuantitativa significativa entre ellos.

Esa auto-calibración consiste en normalizar el valor de cada sensor con el valor promedio propio al inicio del programa (que se promedia realizando un elevado número de mediciones iniciales). De esta forma es posible comparar entre sí las lecturas analógicas de los diferentes sensores en un momento determinado, aunque no se hayan calibrado exactamente sus sensibilidades a través de sus potenciómetros.

Cada vez que se mide una luminosidad en un sensor, la medida se realiza un número de veces (50 por ejemplo), y se promedian los valores obtenidos. Con ello se obtiene un valor promedio adecuado, y además, no penaliza en exceso al algoritmo, dado que la lectura de estos módulos es muy rápida.

Para determinar si se mueve adelante o atrás, se obtiene el valor medio de los sensores frontales y traseros. Si ese valor es mayor que un umbral mínimo (`THRESHOLD`) prefijado, entonces se considera significativa la diferencia entre delante y detrás. Si no se supera, el vehículo se detiene.

Si hay diferencia entre delante y detrás, se compara entre los dos sensores que dieron la media más brillante, y se realiza una especie de ponderación entre las luminosidades leídas para seleccionar un ángulo de giro entre el máximo a la derecha y el máximo a la izquierda, seleccionándolo de forma que el vehículo gire hacia el punto de mayor luminosidad.

4.3.1.2.3. Formalización computacional En el inicio se hace el promedio de `N` valores leídos de cada sensor, siendo `N` un número elevado (500 por ejemplo). Después, en cada iteración, para cada sensor se repite también la medición de la luminosidad un número `n` de veces, grande, pero no tanto como `N`, y también se promedian. Finalmente, cada valor actual $\bar{x}'_i$ se normaliza respecto al valor promedio inicial $\bar{x}^0_i$, para obtener el valor normalizado $\hat{x}_i$:

$$\bar{x}_i^0 = \frac{\sum_{j=1}^{N} x_{i_j}^0}{N} \quad ; \quad \bar{x}_i' = \frac{\sum_{k=1}^{n} x'_{i_k}}{n} \quad \Longrightarrow \quad \hat{x}_i = \frac{\bar{x}_i'}{\bar{x}_i^0}$$

Una vez se tienen los valores actuales normalizados de los cuatro sensores se compara la media de los traseros ($\hat{x}_B$) con la de los delanteros ($\hat{x}_F$), y si según qué zona tenga más iluminación, siempre que se supere un umbral (`THRESHOLD` = T), se determina el nuevo ángulo de giro del eje delantero del SunFounder™ a través de las siguientes relaciones:

$$\left.\begin{array}{ll} \text{Si} & |\hat{x}_B - \hat{x}_F| \le T \implies \text{Permanecer parado} \\ \text{Si} & \hat{x}_F > \hat{x}_B + T \implies \text{Mover al frente} \\ \text{Si} & \hat{x}_B > \hat{x}_F + T \implies \text{Mover hacia atrás} \end{array}\right\} \Longrightarrow \begin{cases} \hat{x}_r < \hat{x}_l \implies L = \text{mín}\Big(\text{máx}\big[+64 + (\hat{x}_r - \hat{x}_l) \times 200, +1\big], +127\Big) \\ \hat{x}_l < \hat{x}_r \implies L = \text{máx}\Big(\text{mín}\big[-64 + (\hat{x}_r - \hat{x}_l) \times 200, -1\big], -127\Big) \end{cases}$$

Si hay movimiento, tras determinar L (recordemos que $L \in [-127, 127]$), se obtiene el ángulo α a girar el eje con una simple relación lineal, que en Arduino puede implementarse con la función `map`. Sean α_{min} y α_{max} los ángulos mínimo y máximo que puede girar el eje del vehículo, entonces el nuevo ángulo α será:

$$\alpha_{\text{new}} = \alpha_{\text{min}} + \frac{|L|}{127} \times (\alpha_{\text{max}} - \alpha_{\text{min}})$$

4.3.1.2.4. Comentarios sobre el algoritmo Como comprobación, se observa que si $\hat{x}_r = \hat{x}_l$, entonces $L = 64$, y por lo tanto, $\alpha_{\text{new}} = 90°$, esto es, en caso de igual iluminación por la derecha que por la izquierda, el eje se pone perpendicular al sentido de la marcha. El parámetro 200 es un valor experimental que variándolo hace que la reacción del vehículo a un cambio de luminosidad lateral sea más o menos brusca, eso es, que se gire más o menos el vehículo. Su ajuste adecuado depende de las características de maniobrabilidad del vehículo.

4.4. Evitación de obstáculos

En la evitación de obstáculos sucede algo similar a lo que ya se comentó en la Sección 4.2 "Seguimiento de un trayecto definido por una línea": cada prototipo dispone de un tipo de sensores diferentes, y por lo tanto, eso ya implica la necesidad de disponer de dos algoritmos diferentes.

En el prototipo de SunFounder™ se utilizan sensores del tipo todo/nada. Además, se ha comprobado que el código suministrado ni funciona directamente, ni ha sido posible "repararlo" puesto que hay varios casos que no maneja bien. Es por eso que se ha hecho necesario idear un algoritmo partiendo de cero, toda vez que no se ha localizado ningún código similar para este tipo de sensores, y además prácticamente no hay referencias bibliográficas de artículos científicos en los que se plantee solucionar este problema con sensores todo/nada, sino siempre se supone que es posible la detección de la distancia del objeto o de otro dato escalar o incluso vectorial, como la magnitud y dirección de un vector de fuerza.

Por su parte, el 4WD dispone de un sensor analógico por ultrasonidos, y su algoritmo sí que puede ser reutilizado, aunque requiere de mucha modificación para obtener un funcionamiento aceptable.

4.4.1. Algoritmo para sensores todo/nada

La evitación de obstáculos en un problema muy analizado en la historia de la computación, incluso desde el inicio de los mismos, sobre todo en conjunción con el campo de la robótica. Básicamente podemos destacar dos campos diferentes, por una parte las soluciones analógicas, y por otra parte las soluciones digitales. Dentro de las digitales, puede distinguirse a la vez soluciones en las cuales los sensores son digitales

(entendiendo como tales, de todo/nada) o los analógicos (que evidentemente son cuantizados, pero que proporcionan un rango de valores considerablemente superior). Otra forma de clasificación depende del conocimiento previo del medio circundante, por ejemplo, conociendo el mapa de obstáculos fijos, o plenamente agnósticos, con desconocimiento total del ambiente. Finalmente, y no menos importante para el objeto del presente proyecto, otra forma de catalogar los algoritmos es la capacidad de cálculo necesaria, o disponible, para la ejecución del algoritmo, sobre todo cuando está ligado a problemas que tienen que resolver en tiempo real con hardware con capacidades limitadas de recursos computacionales.

En nuestro caso vamos a enfrentarnos a la evitación de obstáculos con las siguientes características y restricciones:

- **Muy baja capacidad de cálculo**: utilización de microcontroladores del tipo ATmega, con velocidades de reloj en torno a los 16 MHz.
- **Respuesta en tiempo real**: el móvil a gestionar está en movimiento real en línea, con una velocidad máxima en torno a 0,9 m/s.
- **Número de sensores limitados**: sólo tres sensores, y todos ellos en la parte frontal, ninguno en la parte trasera.
- **Sensores digitales**: los tres son de todo/nada, y sólo pueden ajustar su sensibilidad de forma muy limitada y siempre fuera de línea (potenciómetros físicos), nunca en funcionamiento. Dos de ellos son de corta distancia y fijos, mientras que el tercero será de largo alcance y además es móvil, teniendo la posibilidad de abarcar un ángulo de hasta 180°.
- **Baja maniobrabilidad del vehículo**: aspecto de F1, el ángulo de rotación está comprendido entre 60° y 120° (siendo 90° marcha frontal), gran distancia relativa entre ejes. Esto hace que el radio interno de giro mínimo del vehículo es de unos 185 mm.

Aunque se ha estado buscando algoritmos previos en la bibliografía, no se ha localizado ningún algoritmo que fuera diseñado bajo estos estrictos requisitos. Sí que se ha localizado un código fuente para evitación de obstáculos para coches basados en Arduino similares, el proporcionado por SunFounder™.

En ese algoritmo, el coche tiene una tendencia a quedar en un estado de retroceso continuo, generalmente con las ruedas giradas a tope en un sentido, esto es, dando todo el rato vueltas sobre un mismo centro. Incluso aunque manualmente se provoque la colisión (activación de los módulos de sensores IR) o la visión de un objeto (activación del radar de "larga" distancia), el algoritmo no interrumpía ese estado y continuaba todo el rato retrocediendo en círculo.

Es por ello que se ha considerado pertinente elaborar otro algoritmo diferente que permite adaptarse a las características concretas del prototipo usado en este proyecto. A continuación se procede a desarrollarlo.

4.4.1.1. Especificaciones hardware: prototipo SunFounder™

Para esta funcionalidad, a los elementos base del SunFounder™ se le añaden dos tipos diferentes de sensores:

- Un sensor de larga distancia (que denominaremos "radar") con salida de todo/nada y basado en la reflexión de rayos infrarrojos. Este sensor se haya ubicado sobre un plataforma rotatoria accionada por un servomotor que gira entre 0° y 180°. El intervalo de detección del "radar" es de entre 10 y 80 cm aproximadamente.

- Dos sensores de corta distancia (que denominaremos, "detectores de choque"), también basados en eco de infrarrojos, pero en este caso proporcionan valores analógicos. Su sensibilidad puede ajustarse mediante un potenciómetro de accionamiento manual. Su rango de detección es de 0 a 8 cm. Están separados 15 mm entre sí.

Esos sensores se ubican en la parte delantera del vehículo.

El diagrama de conexionado para el SunFounder™, código versión v2, se muestra en la Figura C.1.

4.4.1.1.1. Objetivo Elaboración de un algoritmo de evitación de obstáculos basado en sensores todo/-nada a ejecutarse en un entorno *on-line* móvil, mediante un vehículo gestionado por una placa Arduino UNO R3, y con un chasis limitado por tener un gran radio de giro mínimo, debido a tener un bajo ángulo de rotación de las ruedas, y una elevada separación entre ejes.

4.4.1.1.2. Condiciones y restricciones Un sensor de "larga" distancia, denominado en adelante "radar", móvil, de todo/nada. Alcance de unos 20–50 cm (depende de condiciones de iluminación, del color del obstáculo y del ajuste de un potenciómetro físico del sensor).

Dos módulos sensores de muy corta distancia, fijos, denominados en adelante "sensores" o "de choque". Alcance de 0–8 cm (depende del ajuste de dos potenciómetros físicos del sensor, y de las condiciones de iluminación y color del obstáculo).

Baja capacidad de cómputo (Arduino UNO R3, con microprocesador ATmega328P a 16 MHz).

Vehículo con poca maniobrabilidad: radio central de giro mínimo de 260 mm (radio interior 185 cm y radio exterior 335 mm), distancia entre ejes de 160 mm y ángulo de rotación de las ruedas máximo de 30° hacia cada flanco.

4.4.1.1.3. Principios del algoritmo El radar será el detector primario de proximidad, así como la guía fuente de información para resolver situaciones al detectar obstáculos. Los dos sensores "de choque" estáticos se tratarán como elementos de seguridad. Si falla la detección del radar (que está en un plano horizontal bastante separado del suelo), existirá cierta posibilidad de que los obstáculos sean detectados cuando ya estén muy próximos al coche por los detectores de corto alcance, o "de choque".

4.4.1.1.4. Descripción del algoritmo El vehículo irá guiado por el radar, el cual irá moviéndose continuamente, barriendo la ventana angular comprendida entre 60° y 120°, siendo 90° la marcha frontal. Mientras el radar no detecte nada, el vehículo seguirá avanzando en la misma dirección. Cuando el radar detecte algún obstáculo, se disminuirá la velocidad, o incluso se detendrá si está excesivamente cerca, realizándose un barrido de todo el frente (desde 0° a 180°). Una función específica evaluará, a partir de los datos de ese escaneo, el sentido de giro y ángulo de las ruedas para evitar el obstáculo. Si se encuentra una ventana libre adecuada, se adopta el nuevo rumbo a velocidad de crucero de nuevo, volviendo a la detección normal del radar (ángulo de 60° a 120°). Si no se encuentra un hueco apropiado, el vehículo retrocederá un determinado tiempo, pasado el cual, se volverá a efectuar un nuevo escaneo total por parte del radar (0° a 180°), volviendo a evaluar si existe un hueco apropiado para avanzar, actuando de igual forma que lo descrito con anterioridad. Finalmente, los dos sensores estáticos "de choque" siempre estarán en estado de detección. Si estos sensores detectan un obstáculo, en cualquiera de los estados anteriores descritos, el vehículo retrocederá durante un determinado tiempo, y como en el anterior caso, se volverán a realizar escaneos periódicos completos (0° a 180°) para determinar si hay posibilidad de avance, o debe seguir retrocediendo.

4.4.1.1.5. Diagrama BPMN del algoritmo A continuación se presenta el diagrama BPMN (*Business Process Model and Notation* del algoritmo propuesto:

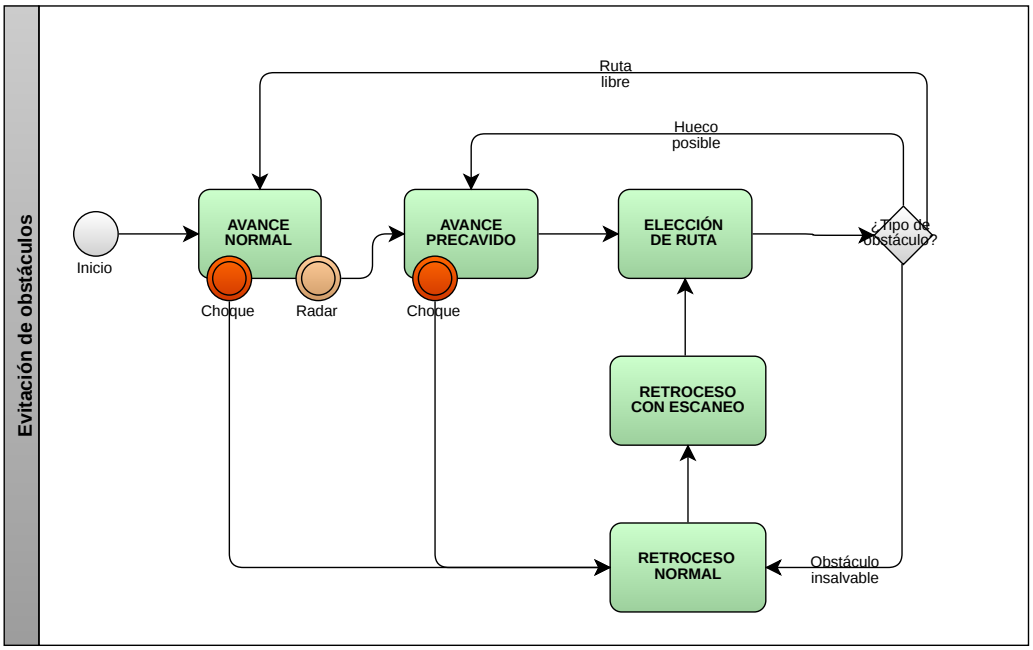

Figura 4.5: Diagrama BPMN del algoritmo propuesto de evitación de obstáculos

Las tareas, que como se verá en la siguiente sección se implementan como diferentes estados, consisten en lo siguiente:

AVANCE_NORMAL Avance normal a velocidad máxima, con el radar barriendo un arco de ±30° sobre la frontal (entre 60° y 120°) para detectar algún obstáculo en el sentido de marcha. Si se detecta algún obstáculo se pasa al AVANCE_PRECAVIDO. Además, continuamente se comprueban los sensores de choque (de corta distancia), y se detecta un choque, se pasa inmediatamente al RETROCESO_NORMAL.

AVANCE_PRECAVIDO Avance a velocidad baja, con el radar barriendo todo lo que permite el servomotor (±90° sobre la frontal, esto es, entre 0° y 180°). Guarda los datos de cada escaneo realizado durante el barrido, para su posterior uso en el estado de ELECCIÓN_RUTA, al que pasa cuando se acaba normalmente el barrido. Además, continuamente se comprueban los sensores de choque (de corta distancia), y se detecta un choque, se pasa inmediatamente al RETROCESO_NORMAL.

ELECCIÓN_RUTA Se detiene la marcha, y se computa la ruta a seguir a partir de los datos obtenidos del escaneo realizado durante la etapa previa de AVANCE_PRECAVIDO. Para elegir la nueva ruta (que será el nuevo ángulo de giro de las ruedas delanteras del coche SunFounder™), se analizan los datos del escaneo, y se calcula cuál es la ventana más ancha sin haberse detectado ningún obstáculo:

- Si esa ventana es mayor a un predeterminado ángulo mínimo óptimo, entonces se cambia a esa ruta, y se pasa al AVANCE_NORMAL, puesto que se entiende que hay suficiente espacio para pasar sin ningún problema.
- En el caso de que la ventana libre mayor no sea tan grande, pero aún así es mayor que otro ángulo mínimo de seguridad, entonces se sigue avanzando, pero con precaución, puesto que hay dudas razonables sobre si habrá bastante espacio para que quepa el vehículo, esto es, se pasa al AVANCE_PRECAVIDO.

- Si no se encuentra ninguna ventana suficiente, entonces se retrocede, puesto que se entiende que el obstáculo es insalvable marchando hacia delante, pasando al `RETROCESO_NORMAL`.

En este caso no hace falta comprobar los sensores de choque, porque se está en estado parado, y en todo caso, si se avanzara, se detectaría en ese momento.

`RETROCESO_NORMAL` En esta tarea el coche simplemente retrocede a velocidad media durante un tiempo predeterminado. No se comprueba ningún sensor, puesto que no hay sensores traseros. Cuando pasa ese tiempo, directamente se pasa al `RETROCESO_ESCANEANDO`.

`RETROCESO_ESCANEANDO` Actúa de forma similar al `AVANCE_PRECAVIDO`, pero circulando a velocidad baja marcha atrás. Se escanea todo el frente, y se comprueban también los sensores de choque. Tras acabar, se pasa a la `ELECCIÓN_RUTA` con los datos recabados.

4.4.1.1.6. Diseño de la implementación La implementación se realizará en C de Arduino, implementando una máquina de estados con los siguientes estados básicos:

- A – AVANCE_NORMAL
- B – AVANCE_PRECAVIDO
- C – ELECCIÓN_RUTA
- D – RETROCESO_NORMAL
- E – RETROCESO_ESCANEANDO

El proceso irá transitando entre estos estados según el diagrama de estados de la Figura 4.6, que se irán sucediendo conforme a las transiciones indicadas.

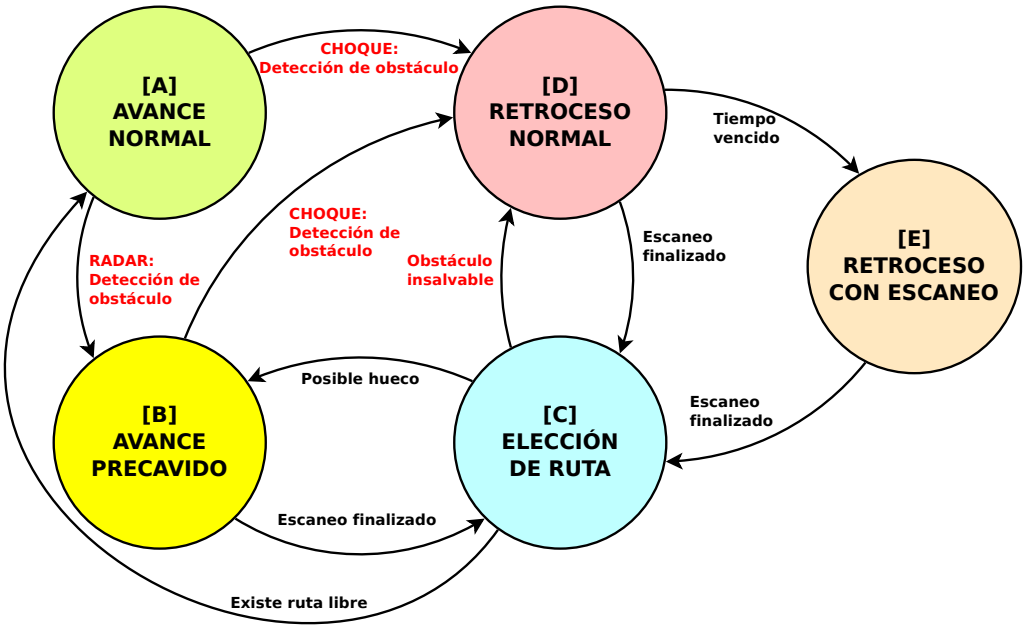

Figura 4.6: Diagrama de estados del algoritmo de evitación de obstáculos con sensores todo/nada

Nótese que en el estado E, teóricamente y de una forma algo rocambolesca, también podría detectarse una "colisión frontal" por los sensores de choque estáticos, puesto que el coche puede estar retrocediendo

con las ruedas giradas. Si en ese momento se aproxima a una pared, el sensor estático del lado exterior al giro puede llegar a detectar esa pared. En ese caso podría plantearse el cambiar el ángulo de giro de las ruedas al complementario (en lugar de girar a derechas, girar a izquierdas, y viceversa). Este problema se visualiza en la Figura 4.7. Este aspecto no está indicado, ni implementado, en el algoritmo, pero pudiera ser interesante si se observara este fenómeno durante las pruebas experimentales. Dicha implementación es bastante sencilla a partir del código presentado en este proyecto.

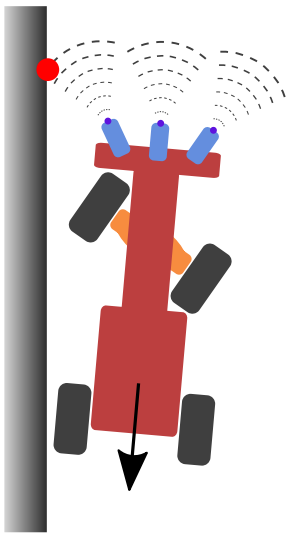

Figura 4.7: Problema de la detección de choque yendo marcha atrás

4.4.1.1.7. Descripción de la implementación La implementación se hace en C de Arduino. Mediante un enumerador se controla la evolución de su estado. Según dicho estado, otras propiedades indicarán los parámetros del movimiento (sentido, ángulo y velocidad), así como se almacenarán las lecturas necesarias para poder informar sobre su estado actual e histórico, así como el entorno detectado. Las funciones servirán para, en función del estado, determinar las características del movimiento puntual en ese ciclo del `loop()` principal del programa Arduino.

4.4.1.1.8. Adaptación específica al prototipo Como ajustes especiales relacionados con las características del propio vehículo se destaca lo siguiente:

1. La velocidad mínima del vehículo para moverse está siendo cada vez mayor, quizá por desgaste de los motores, de las ruedas, o del estado de las baterías. Esto ha hecho que durante el estado AVANCE_PRECAVIDO no solo se disminuya al mínimo la velocidad del coche, sino que se detenga su marcha a la mitad del escaneo. Así se evitan impactos por exceso de velocidad cuando ya se ha detectado un obstáculo con el radar. Podría "mejorarse" como se hace en otros algoritmos: deteniendo el coche totalmente durante ese escaneo.

2. Se ha considerado conveniente, a la vista de las pruebas, que pasado cierto tiempo, en el estado AVANCE_NORMAL, las ruedas se vuelvan a poner rectas, dirigiendo el coche al frente, para evitar exceso de círculos en el recorrido.

3. Se han creado varios parámetros, que según las características de los obstáculos y sobre todo, del vehículo robot, se pueden ajustar para un mejor comportamiento particular. Algunos de estos parámetros son:

Tiempo de retroceso mínimo (`BACK_TIME`) Establece el tiempo mínimo, en milisegundos, que el robot va a retroceder cuando encuentre un obstáculo insalvable. Cuanto más grande sea, más retrocederá, y por lo tanto, mayores probabilidades de encontrar un hueco enfrente. No conviene retroceder en exceso, puesto que puede chocar con objetos que se hayan ubicado detrás durante ese tiempo, dado que en este prototipo no se ha incluido un detector de obstáculos trasero, lo cual podría ser una buena idea, necesaria si se desea probar además otras funcionalidades como la del auto-aparcamiento.

Iteraciones de avance hacia delante (`TURNS_FORWARD`) Establece un número de iteraciones del bucle `loop`. Cuando se llega a ese límite, las ruedas vuelven a la posición de marcha frontal. Esto es interesante porque así se evita que el coche avance en círculos: cuando detecte un obstáculo, girará para evitarlo, y pasado un tiempo, recuperará la marcha frontal, siempre que no encuentre un obstáculo justo delante en ese momento. Según el tiempo de ciclo medio de cada bucle, esas iteraciones serán de más o menos tiempo.

Tiempo de avance mínimo (`FORWARD_TIME`) No es un parámetro propio del algoritmo en sí. Este tiempo se añade para dar tiempo al servomotor que controla el giro del radar a que éste se posicione antes de realizar la siguiente medida. Debe calcularse en función de la velocidad de giro del servomotor, dada por sus especificaciones técnicas, y añadir un poco más de tiempo, por precaución, para que se estabilice.

Ángulo de ventana mínimo (`MINIMUM_WINDOW`) Tras un barrido completo del radar, es la ventana mínima libre, en donde el radar no ha detectado obstáculo, que debe observarse para que el vehículo opte por intentar pasar a través de ese hueco yendo a una velocidad moderada, dado que se supone que es un hueco angosto, justo para el paso del vehículo. Si es demasiado pequeño, el prototipo no podrá pasar a través de él y deberá retroceder, con el peligro de que se caiga en un bucle de intentos y retrocesos. No obstante, para evitar ese peligro, se ha ideado un mecanismo por el cual el prototipo, en caso de demasiados obstáculos enfrente de él, tiene tendencia a girar en el mismo sentido, al efecto de evitar quedar bloqueado en las esquinas, por ejemplo.

Ángulo de ventana óptimo (`OPTIMUM_WINDOW`) Similar al anterior, pero en esta ocasión define un ángulo mínimo libre de obstáculos lo suficientemente grande como para estar seguros de que el prototipo podrá pasar por ese hueco. La diferencia con el caso anterior es que en este caso la velocidad de marcha girando y hacia delante es mucho más rápida, pues hay garantía de que durante los próximos momentos no hay peligro de choque a pesar de estar girando y haciendo una maniobra de evitación de un obstáculo detectado. Un valor demasiado grande hará que el prototipo no sea eficiente y retroceda en exceso antes de atreverse a avanzar de nuevo, pero un valor demasiado pequeño provocaría el mismo efecto bucle que el descrito para el caso del ángulo de ventana mínimo.

Velocidades máxima, mínima y media En todos estos tipos de implementación siempre es interesante controlar el intervalo de las velocidades de los prototipos para cada etapa del algoritmo, según qué se esté realizando. El ajuste dependerá de las capacidades dinámicas del prototipo, de la capacidad de detección de los obstáculos, y de los otros parámetros, como el tamaño de las ventanas mínima y óptima. Conviene que presente el intervalo más amplio posible, al menos para visualizar el comportamiento de la implementación durante las pruebas.

El diagrama de conexionado, para el SunFounder™, que es el prototipo con los sensores digitales todo/-nada, se muestra en la Figura C.1.

4.4.2. Algoritmo para sensores analógicos

Los sensores analógicos utilizados son sensores de emisión y recepción de su eco de ondas de ultrasonidos. Responden con un valor que indica el tiempo de retardo con el que se ha recibido el eco. Ese tiempo, dividiendo por dos (ida y vuelta), y asumiendo unas condiciones ambientales normales (25°C a una presión a nivel del mar, de 1 atmósfera), se convierte en la distancia a la cual está el obstáculo detectado, que en este caso será en centímetros, por ser suficiente esa precisión para esta funcionalidad.

El diagrama de conexionado para el coche 4WD se muestra en la Figura C.2.

4.4.2.1. Especificaciones hardware: prototipo 4WD

La implementación de esta funcionalidad de evitación de obstáculos con sensores analógicos se realiza sobre el prototipo 4WD. Sobre su base común Arduino se le añade únicamente los siguientes componentes:

- Un servomotor pequeño, que es capaz de girar 180° y que se instala en una plataforma que permite rotar, y sobre la cual se instala también el siguiente componente.
- Un sensor de detector de distancias por ultrasonidos, denominado HC-SR04, que puede determinar la distancia de un obstáculo mediante la emisión de un ultrasonido durante unos 2μs, y después esperar su eco. El sensor mide el tiempo entre la emisión del ultrasonido y la recepción de su eco, el cual se produce si existe un obstáculo enfrente del sensor. El tiempo de ida y vuelta del eco está ligado con la distancia a la cual se encuentra el obstáculo, dado que la velocidad del sonido depende fundamentalmente de la densidad del aire, y ésta, de la presión y sobre todo, de la temperatura ambiente. Para que funcione, el obstáculo debe reflejar el sonido, no absorberlo completamente, pues en ese caso, no se puede percibir el eco. Es el caso de sustancias esponjosas y porosas, absorbentes del ruido.

El conjunto de plataforma, con el servomotor y el sensor de distancia, se instala en la parte frontal del prototipo, de tal forma que cuando el servomotor está oriental a la mitad de su recorrido (ángulo de 90°) el sensor de distancia por ultrasonidos apunta al frente, en la dirección perpendicular a los dos ejes fijos de las ruedas del prototipo.

4.4.2.1.1. Descripción general del algoritmo El algoritmo se divide en dos partes principales: la detección de obstáculos y la toma de decisión sobre si se debe modificar la marcha del vehículo en función de lo que se ha detectado.

La detección de obstáculos se realiza a través del sensor analógico de distancias. En marcha normal, hacia delante, este sensor se mueve constantemente a izquierda y derecha en un ángulo pequeño, pero que cubre el hueco necesario para que el prototipo no tropiece con ningún obstáculo si durante el barrido del sensor, éste no detecta ningún obstáculo.

Mientras el sensor no detecta nada, esto es, la distancia del eco es superior a una distancia mínima de seguridad, el vehículo sigue su marcha frontal sin ninguna modificación en su velocidad (típicamente irá a su velocidad máxima, o próxima a ella).

Cuando el sensor detecta algo, determina la distancia del obstáculo. Si esa distancia es demasiado pequeña significa que hay riesgo de colisión, y por lo tanto el vehículo se detiene inmediatamente y empieza a realizar un pequeño movimiento hacia atrás durante una cierta cantidad de tiempo predefinida. Cuando acaba ese retroceso, en estado parado, realiza un escaneo del frente, que más adelante describiremos.

Por otra parte, si la distancia medida por el sensor es superior a la mínima, pero inferior a una distancia de seguridad, entonces significa que se ha detectado un obstáculo cercano al vehículo y que posiblemente puede impedir su paso. En ese caso, el vehículo se detiene, y al igual que en el caso anterior, realiza un escaneo del frente.

El escaneo del frente consiste en que el sensor se sitúa al máximo hacia su izquierda o su derecha (según qué extremo esté más cerca en ese momento), y recorre todo el frente desde ese extremo hasta el otro, realizando medidas cada cierta cantidad de grados. Cuantas más medidas, mejor será para estimar la nueva dirección, pero más tardará en realizarse el escaneo. Durante este escaneo, el vehículo se mantiene parado. Con las mediciones ya realizadas, se evalúa cuál es la mayor ventana libre de obstáculos, si existe alguna, y cuál es su centro, para poder dirigirse a él. No obstante, su tamaño debe ser suficiente. Si es mayor que una ventana mínima de seguridad, entonces se gira hacia ella, y se sigue la marcha normal a la máxima velocidad, pues se presupone que hay espacio suficiente para pasar. Si la ventana no es suficientemente grande, se actúa de la misma forma que cuando se detectaba un obstáculo a una distancia inferior a la mínima de colisión: se retrocede un poco, y se procede a un nuevo escaneo.

4.4.2.1.2. Descripción de los parámetros principales Para la implementación de este algoritmo se utilizan los siguientes parámetros, que deben ajustarse al caso concreto de entorno y prototipo utilizado:

Distancia mínima de seguridad (`DIST_MINIMUM`) Si el detector de distancia devuelve una distancia inferior a ésta, significa que hay riesgo inminente de colisión. Cuando esto ocurre, el vehículo no solo se detiene sino que además retrocede para buscar un hueco por el que seguir moviéndose. Debe seleccionarse de forma que no sea demasiado grande como para que esté continuamente retrocediendo, ni demasiado corta como para que no le dé tiempo al vehículo a detenerse por su propia inercia, y por el tiempo que tarda en volver a comprobar esa dirección exacta.

Distancia de alarma (`DIST_WARNING`) Es la distancia a partir de la cual se considera detectado un obstáculo y hay que hacer una maniobra de evasión con un barrido previo de todo el frente en búsqueda del hueco mayor, si existe. Al igual que con la distancia mínima de seguridad, la distancia de alarma no debe ser excesivamente grande para evitar paradas innecesarias. Pero tampoco debe ser demasiado pequeña, pues el vehículo debe ser capaz de encontrar un hueco suficiente que salve el obstáculo detectado desde la posición en que se detenga, teniendo en cuenta que siempre habrá cierta inercia de movimiento.

Ventana mínima (`WINDOW_MINIMUM`) Ángulo mínimo que debe formar una secuencia continua de medidas del sensor de obstáculos por ultrasonidos en las que el eco recibido se corresponda con una distancia superior a la distancia mínima de detección de obstáculos. Se corresponde con la detección de un hueco libre de obstáculos, y se caracteriza además por su bisectriz, que permite dirigir el vehículo hacia él. Tras un barrido de todo el frente, se elige siempre la ventana mayor, siempre que supere este ángulo de ventana mínima.

Paso de barrido (`ANGLE_STEP`) Es el ángulo que se mueve el servomotor cada vez (y con él, el sensor de distancia) durante el barrido de detección del mayor hueco disponible. Si es excesivamente grande, se realizan pocas mediciones, y puede ser que no se detecten obstáculos finos, estrechos, y se dé por buena una ventana por la que no hay hueco físico real para avanzar. Si es demasiado pequeña, se realizan muchas mediciones, y el proceso se retrasa considerablemente, perdiendo reactividad el prototipo. Depende en gran manera de la capacidad del servomotor en desplazarse y estabilizarse, y también del tiempo de medida mínimo del sensor, porque el tratamiento computacional de cada resultado no es demasiado pesado (sólo almacenar y comparar con los resultados previos).

Ángulo de barrido durante la marcha normal (`ANGLE_LEFT_VISION` y `ANGLE_RIGHT_VISION`) Son los límites del barrido del sensor en marcha normal. Este ángulo no precisa abarcar los 180° que permite el servomotor, sino sólo aquella ventana que asegura que el prototipo no encontrará ningún obstáculo marchando de frente. Si es demasiado estrecho se corre el riesgo de que el prototipo quede encallado al no haber detectado obstáculos laterales que dificulten su marcha, mientras que si es excesivamente grande, tardará mucho en realizar un barrido, y por lo tanto, avanzará mucho entre el reconocimiento consecutivo de la distancia a un punto, con el peligro de que llegue a chocar con él por no dar tiempo a detectarlo en función de la velocidad del prototipo.

Umbrales de selección de nueva dirección (`ANGLE_WINDOW_RIGHT` y `ANGLE_WINDOW_LEFT`) Se definen dos ángulos, típicamente simétricos respecto a la marcha frontal de 180°, de tal forma que se determinan tres zonas (la más a la derecha, la frontal, y la de la izquierda). Según el valor central determinado durante el barrido total tras detectarse un obstáculo, se selecciona una de estas zonas, y con ello, se hace girar/rotar el prototipo hacia la derecha, izquierda o seguir al frente. Nótese de nuevo que en este prototipo no hay forma segura de determinar cuántos grados ha cambiado la marcha tras una rotación sobre su centro.

Tiempos de espera Se definen varios tiempos de espera durante los cuales el prototipo mantiene sus características cinemáticas (velocidad y sentido de giro) sin que se realice comprobación de existencia de obstáculo, ni se busquen nuevas direcciones. Estos tiempos de espera son debidos especialmente al problema que presenta el prototipo para determinar en qué posición definitiva queda tras un determinado giro a una determinada velocidad. Estableciendo unos tiempos de espera fijos, permite predeterminar (aunque sea experimentalmente) el alcance de estos giros, y con ello, afinar el comportamiento del algoritmo al prototipo concreto que lo implementa. Se proponen los siguientes:

Avance mínimo al frente (`DELAY_FRONT`) Es el tiempo mínimo que está el prototipo avanzando al frente cuando no se ha detectado ningún obstáculo. No conviene que sea alto porque le resta reactividad, pero si es muy pequeño se hace mucho uso del servomotor, lo cual puede dañarlo. Conviene que sea pequeño.

Tiempo de espera mínimo (`DELAY_STOP`) Tiempo que está completamente detenido el prototipo tras la detección de un obstáculo. Sirve para frenar y estabilizar el vehículo, así como el servomotor y también preparar el sensor de distancia.

Tiempo mínimo de giro (`DELAY_TURN`) Tiempo mínimo que se le da a los motores para que actúen las ruedas realizando un giro rotacional en torno al centro del prototipo. Este tiempo es el que determina, junto con la velocidad de giro de cada par de ruedas, el ángulo final resultante de la maniobra de rotación. Es por lo tanto, un parámetro importante y debe ajustarse comprobando que el prototipo realiza un giro adecuado en cada maniobra, ni demasiado grande que haga que apunte hacia atrás, o demasiado al lado, ni demasiado pequeño que apenas haya cambiado su dirección, apuntando muy probablemente al mismo obstáculo que fue previamente detectado y que forzó la maniobra de evitación actual.

Tiempo de retroceso previo a un cambio de dirección (`DELAY_BACK_TURN`) Tiempo mínimo del retroceso realizado antes de iniciar el movimiento de giro rotacional para evitar un obstáculo inminente. Cuando se ha decidido girar hacia un determinado flanco, primero se realiza una pequeña marcha atrás, definida por este parámetro, y a continuación ya se rota. Esto es para permitir una mayor maniobrabilidad, especialmente útil si el obstáculo está muy cercano.

Tiempo de retroceso ante obstáculos infranqueables (`DELAY_BACK_BACK`) Es el tiempo mínimo que debe retrocederse cuando el escaneo total demuestra que no hay huecos libres posibles. Por ello

se retrocede una mayor distancia, para volver a realizar un nuevo escaneo en busca de algún nuevo hueco libre.

Tiempo de giro en maniobra de evasión (`DELAY_BACK_RND`) Este parámetro permite al prototipo evadirse de una esquina, o incluso en la mayoría de las situaciones de encallamiento dentro de un callejón sin salida, siempre que éste no sea excesivamente profundo. La idea es que, tras retroceder `DELAY_BACK_BACK` milisegundos, se realiza una rotación (siempre hacia el mismo lado) dada por este parámetro `DELAY_BACK_RND`. Esto permite que el prototipo tenga tendencia a rotar en el mismo sentido cuando está en una situación de bucle de avance/retroceso. Al girar hacia el mismo lado, el prototipo eventualmente girará en torno a sí mismo hasta `360°`, lo que permitirá encontrar algún hueco libre por el que continuar la marcha, si existe. Debe ser lo suficientemente grande como para asegurar un cierto cambio de orientación mínimo, para incrementar las posibilidades de detección de ventana de escape, pero no tanto como para que recorra los `360°` en torno a sí mismo habiendo realizado pocos escaneos, obviando reconocer algún hueco pequeño existente pero susceptible de ser la única vía de escape.

Tiempo mínimo de retroceso en situación de pre-colisión (`DELAY_BACK_NEAR`) Cuando se detecta un peligro inminente de colisión (distancia detectada inferior a `DIST_MINIMUM`), el prototipo retrocede inmediatamente durante este periodo (`DELAY_BACK_NEAR`, al fin de alejarse de ese obstáculo tan cercano. Eso se hace por dos razones: por si es móvil y para evitar que se siga aproximando, y para alejarse lo suficiente como para que no entorpezca la detección de un hueco libre suficiente para continuar la marcha. Si es muy grande, se retrocede demasiado, pudiendo incluso colisionar por la parte trasera; si es muy pequeño, no se aleja lo suficiente del obstáculo cercano y esto impide encontrar un hueco suficiente para avanzar.

4.4.2.1.3. Adaptación específica al prototipo Evidentemente, en el anterior apartado se han detallado los parámetros que permiten adaptar el algoritmo básico a la situación real conjunta formada por el entorno y el prototipo usado. Además de esos parámetros, hay otras dos características que, no siendo exactamente parámetros del algoritmo, sí que deben ajustarse al caso real.

Nótese que según la descripción general del algoritmo, sólo se define una ventana mínima de seguridad, a la hora de determinar si existe un hueco suficiente para pasar el vehículo. Sin embargo, en el anterior algoritmo, el basado en sensores todo/nada montados sobre el SunFounder™, había dos tamaños de ventana mínimo. Esto es así por la diferente maniobrabilidad de los prototipos. En el SunFounder™ se puede determinar exactamente el ángulo de giro del coche, dado que depende del giro con el que el servomotor actúe sobre el eje direccional del coche. Sin embargo, en el 4WD, las cuatro ruedas son fijas, y tanto si se intenta girar suavemente parando las ruedas de un lado, como si se rota sobre el centro del vehículo accionando en sentido opuesto ambos lados, el ángulo de giro del vehículo depende de muchas variables, y por lo tanto es incontrolable el ángulo final que realizará.

Para solucionar ese problema se pensó en acoplar un codificador de velocidad en las ruedas, un sensor acelerómetro o incluso un sensor brújula al prototipo, pero finalmente no se realizó esa prueba.

En la adaptación al hardware utilizado, es de reseñar el parámetro `SERVO_SPEED`, que es la inversa de la velocidad de rotación. Es el tiempo en milisegundos que necesita el servomotor para girar un determinado número de grados sexagesimales y estabilizarse, antes de poder realizar la siguiente medida con el sensor de ultrasonidos que mueve. Si el servo es rápido en su movimiento y estable en su posicionamiento final, este tiempo será menor, y por lo tanto, el prototipo podrá hacer mayor uso del sensor de detección de obstáculos, y con ello tener un comportamiento mejor. Si ese parámetro no se ajusta adecuadamente a las especificaciones del servomotor, entonces puede que éste no haya realizado el giro adecuado, y el programa

ordene al sensor realizar una medida estando en una posición inadecuada. A raíz de eso, la determinación de la nueva dirección de giro también será errónea, y por lo tanto el funcionamiento del prototipo se verá muy afectado, puesto que determinará la posibilidad de seguir un trayecto imposible, y volverá a detectar un obstáculo, por lo que su avance será muy insatisfactorio.

4.4.2.1.4. Formalización computacional Este algoritmo no tiene apenas cómputo que realizar, pues básicamente funciona por acción y reacción. La única parte computacional reside en la determinación de la ventana libre mayor durante el escaneo frontal de `0°` a `180°`, y que servirá para determinar el sentido de giro del prototipo para evitar un obstáculo.

Esa determinación se hace simplemente comparando el valor de la mayor ventana anterior con la actual que se ha encontrado y seleccionando la mayor. Los parámetros que la definen son el ángulo de inicio de la ventana ($\alpha_0^{\texttt{max}}$) y el tamaño de esa ventana ($\Delta\alpha^{\texttt{max}}$). El ángulo de giro ideal ($\alpha_{\texttt{new}}$) será la bisectriz de esa ventana angular, siempre que ésta respete la ventana mínima (`WINDOW_MINIMUM`, $\Delta\alpha'_{\texttt{min}}$):

$$\text{Si} \quad \Delta\alpha^{\texttt{max}} \geq \Delta\alpha'_{\texttt{min}} \implies \alpha_{\texttt{new}} = \alpha_0^{\texttt{max}} + \frac{\Delta\alpha^{\texttt{max}}}{2}$$

La determinación del sentido del giro se realiza comparándolo con los parámetros ya descritos anteriormente, `ANGLE_WINDOW_RIGHT` ($\alpha'_{\texttt{right}}$) y `ANGLE_WINDOW_LEFT` ($\alpha'_{\texttt{left}}$):

$$\begin{array}{lll} \text{Si} \quad \alpha_{\texttt{new}} > \alpha'_{\texttt{left}} & \implies & \text{Rotar hacia la izquierda} \\ \text{Si} \quad \alpha_{\texttt{new}} < \alpha'_{\texttt{right}} & \implies & \text{Rotar hacia la derecha} \\ \text{Si} \quad \alpha_{\texttt{new}} \in \left[\alpha'_{\texttt{right}}, \alpha'_{\texttt{left}}\right] & \implies & \text{Seguir de frente} \end{array}$$

4.4.2.2. Diseño de la implementación

La implementación se realizará en C de Arduino, en donde se tendrán en cuenta los siguiente estados:

- A – AVANCE_CON_ESCANEO_DIRIGIDO
- B – FRENADO_Y_RETROCESO_EMERGENCIA
- C – DETENCIÓN_Y_ESCANEO_TOTAL
- D – ELECCIÓN_DE_RUTA
- E – GIRAR_HACIA_LADO_LIBRE
- F – RETROCESO_Y_EVASIÓN
- G – DETECCIÓN_FRONTAL

El proceso irá transitando entre estos estados según el diagrama de estados de la Figura 4.8, que se irán sucediendo conforme a las transiciones indicadas.

A continuación se describe brevemente las tareas a realizar en cada uno de estos estados:

`AVANCE_CON_ESCANEO_DIRIGIDO` El vehículo avanza frontalmente a la máxima velocidad predefinida, escaneando continuamente un ángulo pequeño a izquierda y derecha para localizar obstáculos que directamente impidan su avance. Es el estado inicial.

`FRENADO_Y_RETROCESO_EMERGENCIA` Cuando se produce una detección de un obstáculo muy cercano, con peligro inminente de colisión, se detiene inmediatamente y se retrocede automáticamente un

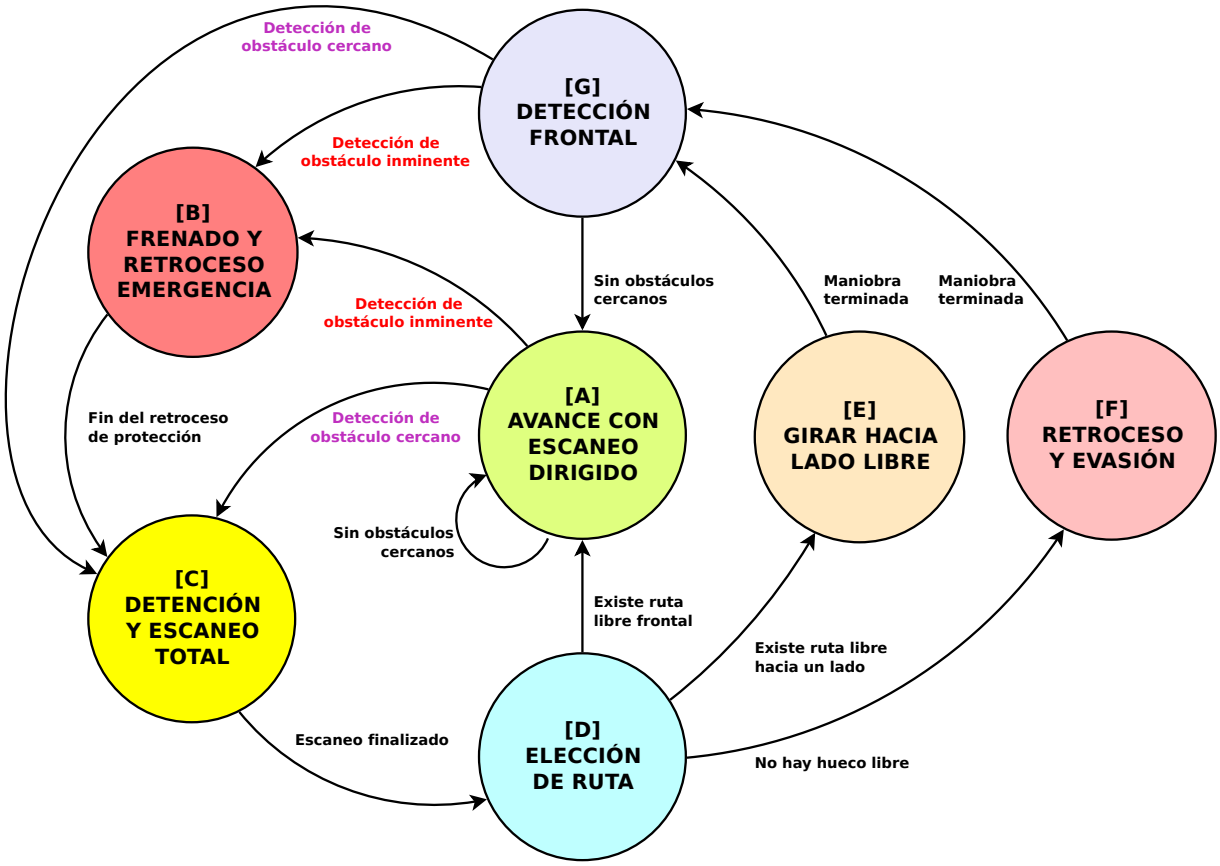

Figura 4.8: Diagrama de estados del algoritmo de evitación de obstáculos para sensor analógico

cierto espacio, para evitar el choque, y alejarse lo suficiente del obstáculo como para favorecer un exitoso escaneo total en busca de una ruta alternativa viable.

`DETENCIÓN_Y_ESCANEO_TOTAL` El vehículo se detiene y escanea todo el frente, de 0° a 180° (o en sentido inverso), realizando y guardando un mapa de obstáculos para uso posterior.

`ELECCIÓN_DE_RUTA` A partir de los datos del escaneo total, se determina si existe una ruta libre, escogiendo la más amplia de las viables. Como resultado da una dirección de movimiento: adelante, a izquierda, a derecha o atrás, si no detecta ningún hueco posible.

`GIRAR_HACIA_LADO_LIBRE` Realiza una rotación del vehículo sobre su propio centro, precedido de un movimiento hacia atrás, para lograr mejor encaminamiento. La rotación se hace en el sentido indicado por la etapa anterior de elección de ruta (izquierda o derecha).

`RETROCESO_Y_EVASIÓN` Se invoca cuando de la elección de ruta se llega a la conclusión de que no hay ningún hueco viable enfrente. Por esa razón se retrocede un poco, y a continuación se rota hacia un lado (siempre hacia el mismo) para colocar el vehículo más atrasado y tener más visión, así como para apuntar hacia otra dirección, a fin de incrementar sus probabilidades de encontrar una ruta viable. La rotación es un proceso vital para evitar que el prototipo se quede bloqueado en un bucle sinfín en una esquina o un callejón sin salida no muy profundo.

`DETECCIÓN_FRONTAL` El sensor comprueba si existe algún objeto cercano justo enfrente de él. Si no hay ninguno a una distancia próxima, procede al avance con escaneo dirigido, y si hay alguno, en función de la distancia a la que se encuentre, se frena y retrocede, o se detiene y realiza un escaneo total.

4.5. Control de velocidad adaptativo

El control de velocidad adaptativo requiere de la combinación de dos de las funcionalidades ya presentadas e implementadas, el seguimiento de trayecto y la evitación de obstáculos, pero también de una tercera que será presentada conjuntamente, que es el control de velocidad.

Veamos la función de cada una de estas funcionalidades individuales en la funcionalidad combinada del control de velocidad adaptativo (ACC):

- **el seguimiento de un trayecto, definido por una línea**: se va a necesitar que el vehículo circule por un circuito, para que su propio movimiento esté controlado y en el que podamos introducir un obstáculo móvil adecuado, que preferentemente también siga esa línea, al menos temporalmente. Esto es lógico, puesto que en las más avanzadas tecnologías reales actuales, como en el Autopilot, el seguimiento de carril (denominado *Active Lave Keeping*) está incluido, que es el equivalente en funcionalidad al seguimiento de línea en nuestro prototipo: seguir un trayecto predefinido.
- **la evitación de obstáculos**: es necesario por cuanto hay que evitar que el prototipo impacte con cualquier obstáculo que se cruce por su camino, especialmente aquellos que siguen su misma trayectoria y son más lentos, de tal forma que el vehículo debe evitar alcanzarlos, cosa que ocurriría si mantuviera su velocidad prefijada. Es por ello que el control de velocidad adaptativo necesita de un mecanismo para la detección de obstáculos, para implementar su evitación. En este caso, la funcionalidad de la evitación de obstáculos es algo diferente a la presentada en secciones anteriores, dado que aquí el prototipo no puede evitar el obstáculo cambiando de dirección, y mucho menos retrocediendo, ya que es una premisa que debe preservar la trayectoria (línea) a seguir. Por ello, en caso de que el vehículo anterior vaya más lento o incluso se detenga (caso de un atasco), la única solución es moderar la velocidad o incluso detener la marcha. Otras posibilidades, como el cambio de carril, se corresponden con otras funcionalidades todavía más avanzadas que el ACC, dado que ya requieren de criterios propios de autonavegación limitada.
- **el control de velocidad**: se trata de una funcionalidad nueva. Es imprescindible en el control de velocidad adaptativo, como se desprende de su propio nombre, dado que debe ser el propio vehículo el que ajuste su velocidad a la del obstáculo (vehículo) que le precede, en el caso de que éste circule a una velocidad inferior a la deseada por el vehículo que tiene activo el ACC.

A la hora de implementar, también es importante tener en cuenta que ésta debe realizarse utilizando dos prototipos, para que las pruebas sean más reales y concluyentes.

Es por ello que se utilizarán los dos prototipos presentados y que cada uno cumplirá una función, y por ello, cada uno implementará una funcionalidad diferente a saber:

- SunFounder™: Será el obstáculo. Irá circulando por el circuito a una velocidad variable, para generar distancia variable, que fuerce al perseguidor a controlar inteligentemente su velocidad para no colisionar con él. Por lo tanto, tendrá implementada la funcionalidad de seguimiento de trayecto, pero con variación periódica de velocidad.

- 4WD: Es el perseguidor. Circulará por el circuito, detrás del SunFounder™, evitando en todo momento colisionar con él. Además, deberá variar su velocidad de tal forma que mantenga una distancia de seguridad prefijada, y en todo caso, nunca inferior a un mínimo. Cuando esté a una distancia mayor, deberá circular a su máxima velocidad predefinida. En él, pues, se implementará la funcionalidad de control de velocidad adaptativo, ACC.

Estos dos prototipos circularán sobre un circuito cerrado definido por una línea de unos 40–50 milímetros de anchura. No conviene que tenga curvas excesivamente cerradas, para evitar que el comportamiento diferente de los dos prototipos en esos casos extremos impida comprobar el correcto funcionamiento de la funcionalidad que nos ocupa ahora, el control de velocidad adaptativo.

4.5.1. Especificaciones hardware: prototipo SunFounder™

Son las mismas que las ya detalladas en el apartado 4.2.1.1, e implementará un código muy similar al mostrado en el Listado A.1, pero al que se le añade un cambio periódico de velocidad. El código definitivo se ha incluido en el Listado D.1.

4.5.2. Especificaciones hardware: prototipo 4WD

En principio, la implementación se realizó partiendo de la configuración hardware del 4WD del seguimiento de trayecto, según se ha indicado en el apartado 4.2.2.1. No obstante, durante las pruebas se observó que uno de los tres sensores de infrarrojos, que están físicamente unidos, dejó de funcionar, concretamente el de un lateral. Esto obligó a tener que cambiar el hardware detector de líneas, y el único disponible en ese momento fue un único módulo con ocho sensores por infrarrojos integrados, y que se comunica por el bus IIC, no directamente con pines individuales.

Esto ha obligado a rehacer el código, variando incluso algo el propio algoritmo, puesto que se ha trabajado con cuatro de esos ocho sensores, y por lo tanto, la lógica sobre la detección de líneas, y hacia donde girar, es diferente a que si son sólo tres. Un cambio tan simple como que el número sea par o impar ha tenido un impacto considerable, puesto que ahora ya no hay un sensor central dominante, sino que son dos, y las combinaciones de lecturas son diferentes y deben interpretarse y gestionarse de una nueva forma.

En general el algoritmo es el mismo, pero no la detección de la línea, y por lo tanto, la forma de decidir qué hacer en cada momento.

La implementación definitiva, basándose en la configuración base del 4WD, es la siguiente:

- Se le incorpora un módulo que tiene integrados ocho sensores de detección de línea por infrarrojos. Los sensores físicos son también TRCT5000, como los anteriores, pero en lugar de consultarse de forma independiente, se consultan de forma colectiva a través del bus IIC, también conocido como I2C. De estos ocho sensores, sólo se utilizan los cuatro centrales, para que el algoritmo sea parecido al anterior. Estos sensores están separados 12,8 mm entre sí (½").
- Para la detección de obstáculos se utilizan tres sensores por ultrasonidos, modelo HC-SR04, como los usados en este mismo prototipo para la funcionalidad de evitación de obstáculos. Sin embargo, en lugar de montar un único sensor sobre una plataforma giratoria accionada por un servomotor, se colocan tres sensores iguales fijos, en la parte delantera. El central mirando al frente, y los dos laterales, un poco más retrasados por cuestiones de montaje, girados levemente hacia el exterior, aproximadamente unos 10° respecto al frontal.

De esta forma, el seguimiento de línea se realiza a través de los cuatro sensores por infrarrojos, mientras que la detección de distancia con los tres sensores de medición del tiempo de eco por ultrasonidos. Estas distancias servirán para controlar la velocidad del vehículo, así como para evitar colisiones.

El hecho de incorporar tres sensores de distancia fijos frontales es una variación importante sobre la implementación de la funcionalidad de la evitación de obstáculos que se ha presentado en el Punto 4.4.2.1. Recordemos que en la implementación de la funcionalidad básica se utilizaba un único sensor de distancias por ultrasonidos, y que éste estaba montado sobre una plataforma giratoria accionada por un servomotor.

En el apartado experimental se justificará este cambio.

4.5.3. Algoritmo del control de velocidad adaptativo

El algoritmo que se presenta a continuación se basa en las implementaciones ya analizadas en este proyecto de las funcionalidades individuales, pero la combinación es completamente original, puesto que no se ha localizado ningún código ni algoritmo similar que las implementara conjuntamente.

4.5.3.1. Descripción general

Continuamente se monitorizan los valores leídos por el módulo de sensores de seguimiento de líneas. Según estos valores se controla el movimiento hacia delante o de giro del prototipo, siempre con el objetivo de mantenerlo sobre la línea a seguir. Según la dirección a seguir, se decide qué sensores de distancia por ultrasonidos se consultan, a los efectos de controlar la distancia a la cual se encuentra un posible obstáculo que pueda interferir con la marcha del vehículo.

En caso de que la línea se pierda, se retrocede hasta recuperar la línea, hecho que consiste en detectar repetidamente la línea un determinado número de veces continua, para asegurarse de que realmente se trata de una línea a seguir. Durante el retroceso no hace falta consultar los sensores de distancia, toda vez que éstos están sólo en la parte delantera.

En el caso de que se observe que la línea se desvía, el vehículo rota para mantenerse encima de ella. Tras cada rotación se comprueban los sensores fijos distancia, para ver si aparece súbitamente un obstáculo inesperado por ese lado, para frenar o controlar la velocidad, según su distancia.

Si la detección de línea muestra que ésta sigue dirigiéndose al frente, se procede a comprobar los tres sensores de distancia. Si ninguno de los tres detectan obstáculos peligrosos, se procede a controlar la velocidad la velocidad de avance mediante un controlador PID que tiene en cuenta la historia previa, y la anterior y actual media de distancias. Este controlador será el responsable de que se mantenga en lo posible la distancia al vehículo anterior, si es que éste está dentro de la zona de control.

Por otra parte, si el sensor frontal detecta un objeto a una distancia menor que la mínima, o alguno de los otros lo hace a una distancia menor que la de alarma, el vehículo se detiene para evitar el choque.

Finalmente, si el vehículo que precede está a una distancia inferior a la distancia máxima de control, entonces entra en marcha el controlador PID. Este controlador funciona como es usual, teniendo ganancia (parte proporcional), y términos integrales y derivativos, que permiten adaptarse a la velocidad del coche que lo antecede manteniendo un intervalo seguro de distancias, dados por la distancia mínima y la máxima, intentando estabilizarla en la distancia prefijada intermedia. Gracias a los datos de la distancia frontal (promediada con la lateral hacia la que se gira si ésta es menor), se varía la velocidad de marcha. Esta velocidad, no obstante, tiene uno límites prefijados de velocidad mínima y máxima definidos por el propio prototipo, de forma que a veces no es posible gestionar la distancia prefijada, de ahí la salvaguarda de que en caso de aproximarse demasiado al coche de delante, se habilite la detención del vehículo.

4.5.3.2. Formalización del algoritmo

En la 4.9 se muestra el diagrama de flujo de la implementación del algoritmo realizada en C de Arduino.

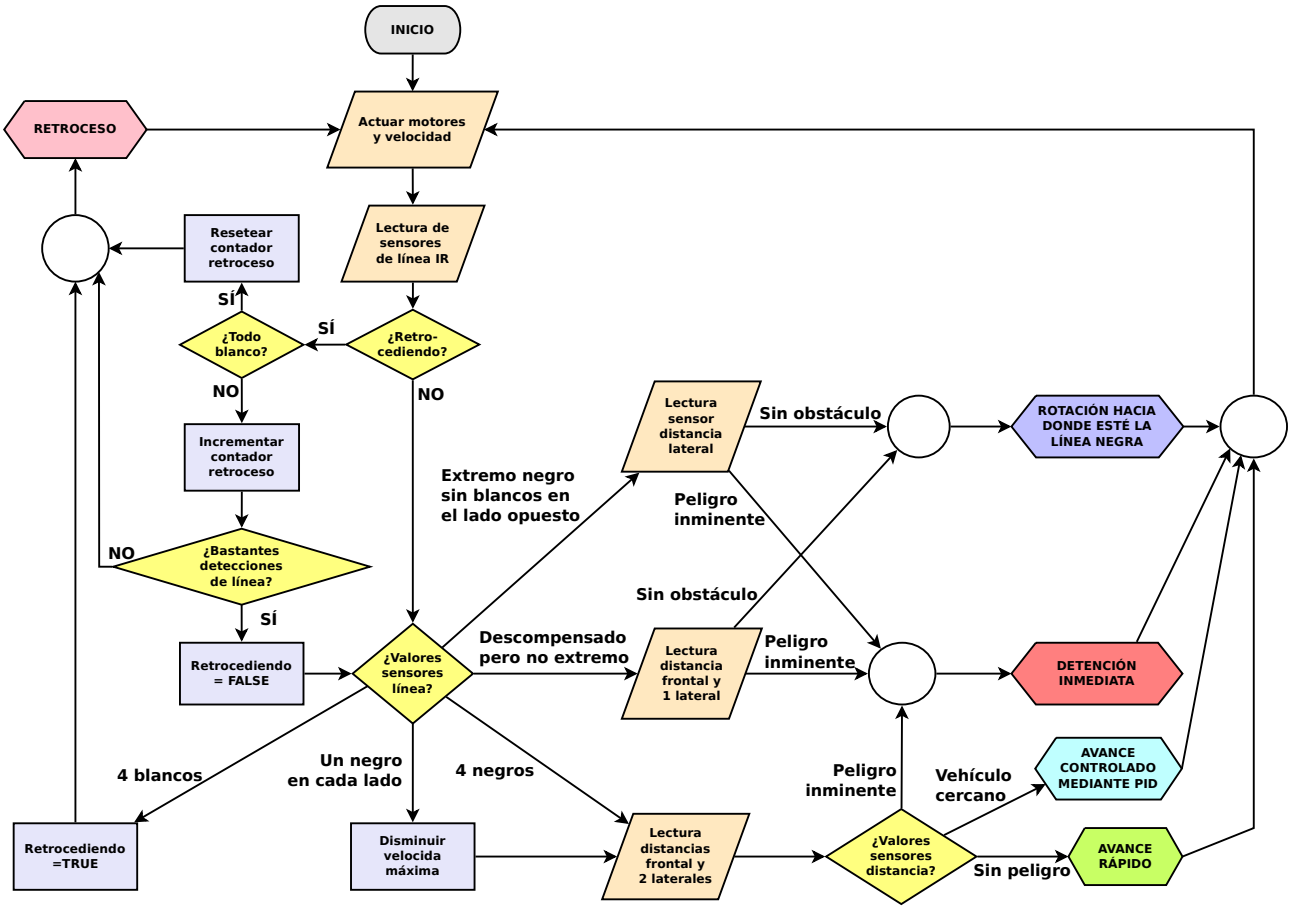

Figura 4.9: Diagrama de estados del algoritmo de evitación de obstáculos para sensor analógico

4.5.3.3. Descripción de los parámetros principales

Los parámetros que se definen en este algoritmo son similares a los introducidos ya en el seguimiento de líneas del 4WD, los cuales no se repetirán aquí. Los que se añaden son los siguientes:

Velocidad de crucero (máxima) (`SPEED_MAX`) Es la velocidad que fija el conductor, y que debe mantener el sistema ACC siempre que pueda, esto es, mientras no haya obstáculos frente al vehículo que obliguen a reducir la marcha.

Distancia deseada (`DIST_DESIRED`) Distancia prefijada de control que es la distancia que el vehículo con ACC deberá mantener en caso de que el vehículo que le precede circule a una velocidad inferior a la velocidad de crucero fijada.

Distancia de inicio de control `DIST_MAXIMUM`) Es la distancia a partir de la cual el controlador PID de velocidad no entra en juego, y el vehículo se mantiene a su velocidad de crucero fijada. Por el contrario, si el vehículo precedente está a una distancia menor, la funcionalidad de control de velocidad adaptativo, ACC, entra en juego para intentar mantener una distancia adecuada a ese vehículo, siempre sin superar la velocidad de crucero fijada.

Distancia mínima (`DIST_MINIMUM`) Es la distancia mínima medida por el sensor frontal que puede tolerarse durante el estado de control de velocidad adaptativo. El controlador PID deberá evitar que la distancia al vehículo de delante caiga por debajo de ese valor. En ese caso (por ejemplo, una situación de atasco, una velocidad anormalmente baja del vehículo precedente, o incluso un frenazo), nuestro vehículo debe también frenar, para evitar la colisión, reanudando la marcha en cuanto el obstáculo desaparezca (o incremente su velocidad por encima de esa distancia mínima).

Distancia de alarma o de seguridad (`DIST_WARNING`) Es la distancia que, una vez detectada por algún sensor lateral, invoca el procedimiento de frenazo de urgencia, para evitar una colisión inminente. El comportamiento es similar al descrito en la distancia mínima respecto al sensor frontal.

Parámetros del controlador PID En el controlador PID están los términos habituales de ganancia (`KPP`), integral (`KPI`) y derivativo (`KPD`), pero también se han considerado otros tres parámetros para mejorar el comportamiento de este control en una situación de control de velocidad inteligente:

Máximo error de la señal (`MAX_ERROR`) Por señal se entiende el resultado del cómputo normal del controlador PID, su salida. Este parámetro viene a limitar su valor, para limitar así la actuación del controlador PID sobre la velocidad del vehículo. Ese valor es en términos absolutos, esto es, limita tanto la señal máxima positiva como la mínima negativa. Así se impide que una anómala lectura de los sensores provoque un cambio desmesurado en la velocidad.

Error máximo en la distancia (`DIST_MAX_ERR`) Se establece un valor máximo (en términos absolutos, válido para errores positivos y negativos), que limita el error calculado entre dos medidas consecutivas del mismo sensor. Tiene la misma finalidad que el caso anterior: filtrar los errores de medida. Además tiene significado físico: circulando a una determina velocidad, no es lógico que una distancia en un determinado intervalo de tiempo conocido varíe en más o menos de unos determinados valores.

Error máximo del factor integral (`DIST_MAX_INT`) Limita el sumando integral en el cómputo del controlador PID. Sirve para limitar situaciones tales como que el vehículo de enfrente circule a nuestra máxima velocidad, pero a una distancia comprendida entre la deseada y la de inicio de control. Si no se limitara, este factor tendería al infinito, y una variación (disminución en este caso) brusca de la distancia no sería convenientemente gestionada por el controlador PID, puesto que la parte derivativa no sería capaz de compensar ese valor tan grande acumulado históricamente.

4.5.3.4. Adaptación específica al prototipo

Los parámetros del controlador PID se han determinado empíricamente. Aunque existen métodos para su determinación experimental, tal como el método de Ziegler–Nichols [72], esto requeriría una monitorización experimental complicada. El método seguido para su ajuste ha consistido en dos etapas:

- Realización de un pequeño ajuste inicial manteniendo el vehículo sin capacidad de accionar las ruedas (por ejemplo, haciendo que éstas giren en el vacío). En esa situación se le acerca o aleja un obstáculo, y a través de la monitorización, por el puerto serie o a través de la pantalla LCD, de la señal resultante del PID, ajustar sus parámetros.
- Posteriormente se procede a realizar unos primeros intentos en el circuito con los dos vehículos circulando, y se ajustan ligeramente los parámetros del PID mediante prueba y error hasta obtener unos valores que permitan un funcionamiento razonablemente correcto en el entorno experimental diseñado del vehículo que implementa el ACC.

CAPÍTULO 5

RESULTADOS

En este capítulo se presentan las pruebas realizadas junto con los resultados obtenidas de ellas. El procedimiento de la implementación no ha sido tan directo como podría deducirse del Capítulo 4 “Implementación”, en donde sólo se ha presentado un algoritmo con su correspondiente código listado en los anexos. Por el contrario, ha sido un continuo procedimiento de implementación, de prueba y observación del comportamiento de los prototipos ante diferentes escenarios y condiciones las que han ido perfilando los algoritmos presentados en este proyecto.

A continuación se van a describir las pruebas realizadas, funcionalidad por funcionalidad, y para cada prototipo, haciendo especial hincapié en aquellos aspectos de las pruebas que han influido en el código final, y realizando valoraciones sobre el comportamiento no sólo del software, sino también del hardware.

En aquellos casos en que se considere interesante se incluirán datos sobre los tiempos de ejecución de los algoritmos, relevante cuando la saturación computacional, o de lectura/escritura en los sensores o actuadores, es tal que ha llegado a ser un componente clave para la limitación del tamaño del código, y por lo tanto, ha podido, o no, afectar a la propia funcionalidad, o a cómo ésta se ha conseguido.

Otro aspecto valorado durante las pruebas ha sido el comportamiento del prototipo, especialmente ante variaciones del entorno. Se tratarán de valoraciones cualitativas, puesto que es difícil cuantificar el mejor o peor comportamiento observado, pero se ilustrarán indicando los momentos relevantes en que se manifiestan en los vídeos grabados durante las pruebas.

El factor de la fiabilidad de los sensores y componentes Arduino también ha jugado un factor importante en el desarrollo de las pruebas, debiendo incluso cambiar algoritmos que funcionaban por otros nuevos porque algún componente ha dejado de funcionar correctamente, y ha sido imposible encontrar un reemplazo igual a tiempo.

De prácticamente todas las pruebas importantes realizadas se han grabado vídeos. Esos vídeos se han publicado en un canal de YouTube, convenientemente referenciados durante la explicación de las pruebas.

5.1. Montaje

La primera tarea experimental fue el montaje básico de los prototipos. El montaje está bastante bien explicado en el manual original del *Smart Car Kit for Arduino* [66] y en uno de los varios manuales que explican el montaje del prototipo genérico conocido comúnmente como 4WD [62].

Basándose en ese montaje básico, para cada funcionalidad y cada prototipo se añadieron los sensores necesarios, muchas veces sin eliminar físicamente los anteriores si éstos no molestaban ni para el montaje ni para su programación y uso, al fin de evitar perder las configuraciones ya realizadas que se iban a utilizar posiblemente al realizar las pruebas de las funcionalidades combinadas.

5.1.1. Incidencias en el montaje

Como única incidencia reseñable, cabe señalar que hay un error en la Figura 4.4.1 "Conecting the First Photosensitive Module" del manual de SunFounder™ [66]: los cables naranja y azul que conectan el sensor a la placa de extensión de sensores no deben conectarse al pin `G` (masa) de las señales 3 y 4 respectivamente, sino evidentemente al pin `S` (*signal*, señal) de las mismas señales. Tal como está representado los sensores no funcionan adecuadamente, al estar en la práctica anulado el primer sensor, el del flanco izquierdo.

5.1.2. Conexionados

Los conexionados definitivos utilizados para cada prueba se muestran en los anexos se han realizado utilizando el programa de código abierto y libre Fritzing [20]. El código de colores del conexionado no es uniforme, aunque en general se ha respetado el uso del rojo como señal de voltaje positivo (`VCC` o `+5V`), y el negro el de masa (`GND`). Lo que sí que se corresponde es exactamente con el color de los cables utilizados en el prototipo real, de tal forma que al observar con detenimiento los vídeos de las pruebas se puede determinar la función de cada uno de ellos.

También se han realizado diagramas con Fritzing, en los cuales se han incorporado los componentes reales utilizados en las pruebas. En general los sensores no estaban disponibles en las bibliotecas de componentes de Fritzing, por lo que ha sido necesario crear dichos componentes como partes individuales en dicho programa, incluyendo también su esquemático y sus características. De los diagramas se puede extraer esa información, para reutilizar esas partes en nuevos diagramas de conexionado a realizar en futuros trabajos.

5.2. Seguimiento de trayecto

Para el seguimiento de trayecto se creó un circuito con líneas muy estrechas, de entre 7 milímetros en las líneas rectas a unos 20–25 milímetros en las curvas. Por ese circuito ha circulado individualmente cada uno de los prototipos, en un sentido y en el inverso. El circuito se ideó de forma que contaba con curvas bastante cerradas, tanto que el SunFounder™, girando al máximo posible su eje (unos ±30° respecto de la marcha frontal), era incapaz de seguirlas sin tener que hacer maniobras.

A continuación se comentan las principales observaciones realizadas sobre el comportamiento de cada prototipo.

5.2.1. Prototipo SunFounder™

El comportamiento es correcto. A continuación se detallan las principales observaciones, haciendo referencia también al vídeo en el que se pueden visualizar:

- Las líneas rectas muy estrechas, de 7 milímetros de anchura, son seguidas sin demasiado problema, y eso a pesar de que los sensores centrales están separados unos 13 mm entre ellos. Esto es gracias a la media móvil, que proporciona cierto tiempo de recuperación de la línea perdida en caso de una ligera desviación. Ver vídeo [47] (00:44 a 00:47).
- La utilización de la media móvil como forma de convertir una señal originariamente digital en un valor real (`float`) comprendido entre 0 y 1, permite eliminar ruidos de fondo en la lectura de los sensores, esto es, filtrar las motas más oscuras del suelo que no pertenecen a la verdadera línea negra a seguir. Ver vídeo [45] (00:15 a 00:25).
- Las líneas gruesas, de unos 20–25 mm, son seguidas sin problema, incluso en curvas cerradas, siempre que no sobrepasen el radio mínimo de giro del SunFounder™.
- El prototipo es capaz de recuperar fácilmente la línea perdida en las curvas cerradas, cuando el radio de giro es demasiado cerrado, inferior al radio mínimo de giro del SunFounder™. Ver vídeo [46] (01:50 a 02:12).
- El prototipo recupera la línea perdida o un cambio brusco de dirección de la misma gracias al procedimiento de avanzar un cierto tiempo y después retroceder cambiando aleatoriamente la dirección de las ruedas, realizando esta maniobra cuantas veces sea necesaria para recuperar, o encontrar, la línea perdida, o encontrar otra línea semejante cercana, incluso en el caso más desfavorable de líneas de sólo 7 mm de ancho con múltiples cruces entre ellas. Ver vídeo [47] (00:33 a 00:43).
- El prototipo es capaz de recorrer en ambos sentidos de rotación el circuito cerrado diseñado. Según el sentido de giro, la dificultad de las zonas varía, dado que así lo hacen los radios de las curvas, así como la anchura del inicio de las curvas. Compárense para ello los vídeos [45] y [46].

Para obtener estos resultados, la parametrización del código ha sido la indicada en el anexo.

5.2.1.1. Pruebas con otras placas hardware

Además de las pruebas realizadas con la configuración mostrada en la Figura A.1, que se utilizaron para afinar la implementación presentada, se realizaron otros conexionados para pruebas adicionales, utilizando componentes diferentes.

Al objeto de conocer desde el principio la diferencia real existente entre varias alternativas de hardware, y una vez afinada la implementación del seguimiento de líneas para el prototipo SunFounder™, se procedió a realizar una serie de pruebas con otras placas base (*motherboards*) y de extensión (*shields*), y otros microprocesadores. En esas pruebas el conexionado fue diferente al mostrado en la Figura A.1, pero adaptado a partir de él, para modificar lo menos posible el código, y evitar introducir errores de programación al cambiar el número de los pines, por ejemplo.

En la Tabla 5.1 se comparan las tres placas que se han probado al inicio del proyecto. Nótese que no se incluye en la comparación la placa Fundino UNO R3 que utiliza el prototipo 4WD, dado que ésta es prácticamente un clon de la Arduino UNO R3, y comparte las mismas especificaciones que las indicadas en esa tabla. También se ha añadido la placa Arduino Yún, que aunque no se ha utilizado en este proyecto, es una alternativa interesante porque, manteniendo un microcontrolador Atmel, incorpora un coprocesador MIPS, capaz de ejecutar un Linux dentro de él, lo que le dota de un mucho mayor computacional al poder descargar parte de la funcionalidad e incluso cómputo en él. El problema es que es mucho más caro que el Arduino UNO R3, de tal forma que tiene un precio incluso comparable con la teóricamente más potente Intel Galileo.

Característica	Arduino UNO R3	Arduino Mega 2560	Arduino Yún	Intel Galileo
Microcontrolador				
Modelo	ATmega328P	ATmega2560	ATmega32U4	
Arquitectura	AVR RISC 8-bit	AVR RISC 8-bit	AVR RISC 8-bit	—
Frecuencia trabajo	16 MHz	16 MHz	16 MHz	
Microprocesador				
Modelo	—	—	AR9331 Linux	Intel Quark X1000
Arquitectura	—	—	MIPS RISC	Intel x86
Frecuencia trabajo	—	—	400MHz	400MHz
Voltaje				
de operación	5V	5V	5V	3,3V
de alimentación	7–12V	7–12V	5V	5V
Pines analógicos				
Entrada	6	16	12	6
Salida	0	0	0	0
Resolución	10 bit	10 bit	10 bit	12 bit
Pines digitales				
Entrada/Salida	14	54	20	12
Con capacidad PWM	6	15	7	6
Memoria				
EEPROM	1 kB	4 kB	1 kB	—
SRAM microcontrolador	2 kB	8 kB	2,5 kB	—
SRAM microprocesador	—	—	16 MB	512 kB
DRAM microprocesador	—	—	—	256 MB
Flash microcontrolador	32 kB	256 kB	32 kB	—
Flash microprocesador	—	—	64 MB	8 MB
Comunicaciones				
USB	Normal	Normal	Micro	Micro
USAT	1	4	1	1
Tamaño	53×75mm	53×102mm	75×102mm	70×100mm

Tabla 5.1: Comparación de las placas base utilizadas en este proyecto

Aparte de lo arriba indicado, la placa Intel Galileo integra otras muchas funcionalidades de serie inexistentes en las placas Arduino arriba indicadas: bus de expansión mini PCI Express, puerto USB anfitrión (para teclados, ratones,...), lector de microSD, un puerto de Ethernet RJ45 y otro puerto RS232. Algunas de estas características están presentes en placas bases específicas basadas en Arduino UNO R3, o a través de módulos que se conectan a ella, pero no de una forma tan integrada, cómoda y transparente en hardware como lo hace la placa Intel Galileo.

A continuación se comentan los resultados y las impresiones obtenidas de esas pruebas para cada una de las alternativas probadas.

5.2.1.1.1. Utilización de una otra tarjeta de expansión para sensores El conexionado físico de los componentes Arduino es una labor delicada y trabajosa. Trabajosa por cuanto se trata de pines muy pequeños y a veces de difícil acceso cuando la placa de extensión empieza a estar muy poblada de cables conectados a ella. Delicada porque un error en el conexionado, sobre todo si erróneamente se conecta un pin de masa a un pin de corriente, puede provocar no sólo el incorrecto funcionamiento del sistema (con el consiguiente quebradero de cabeza para la localización del error, incluyendo la revisión del software por si es de ahí el problema, que es lo primero que se suele pensar), sino porque pone en peligro la integridad y funcionamiento del propio hardware.

El prototipo SunFounder™ viene con una tarjeta de expansión para sensores especial, la *sensor shield v5.1*, que parece que es un diseño propio de SunFounder™, puesto que no se ha localizado ningún otro fabricante ni suministrador que lo comercialice. Sí que se ha localizado alguna que otra placa de expansión con la misma nomenclatura, pero que en realidad es totalmente diferente. Con el fin de poder cambiar funcionalidades rápidamente, y dado el largo plazo de entrega de la *shield* original, se probó con la tarjeta de expansión para sensores *sensor shield* v5.0, mucho más estandarizada.

Esta tarjeta de expansión para sensores v5.0 tiene una funcionalidad relativamente parecida a la v5.1 de SunFounder™, pero con un conexionado y distribución diferente. La v5.1 tiene mejor conectividad, más surtido de pines (varios pines físicos diferentes en lugares distintos para el mismo pin lógico), e incluso tiene dos pines con resistencias de *pull-up* (jumpers 34 y 35 de la figura 2.1.0 "Schematic Diagram for Sensor Shield", del tutorial de SunFounder™ [66]).

No obstante, por si fuera necesario, se probó realizar el conexionado para la funcionalidad de seguimiento de líneas en la v5.0, y se comprobó que funcionaba correctamente todo, siendo únicamente necesario conectar los cables a los correspondientes pines de la nueva placa de extensión. A pesar de este éxito, en el trabajo se continuó utilizando la v5.1, propia de SunFounder™, al ser ésta la que se utilizaba en el tutorial.

Conclusión de la prueba La placa de extensión para sensores genérica *sensor shield v5.0* funciona de forma análoga a la específica v5.1 diseñada y fabricada por SunFounder™. No se han detectado problemas o incompatibilidades en su utilización, aunque se observa que presenta una menor conectividad y una disponibilidad más reducida de conectores.

5.2.1.1.2. Arduino Mega 2560 La placa base Arduino Mega 2560 es una mejora de la Arduino UNO R3 en términos de conectividad y manejo de señales, así como de memoria disponible para la programación y la ejecución de los *sketches*, pero en cuanto a computación pura es igual, ocupando más espacio. Además, dispone de una distribución de los pines ligeramente a la Arduino UNO R3, por lo que no es compatible con tantas *shields* como sí lo es la Arduino UNO R3, y además puede provocar problemas de cortocircuito si no se tiene en cuenta este hecho cuando se cambia el desarrollo de una a otra.

En este proyecto se probó esta alternativa por su mayor capacidad de entradas/salidas (ver vídeo [47]), lo cual en algún momento se pensó que podría ser crítico conforme se fueran combinando funcionalidades en el prototipo. No obstante, la implementación final del control de velocidad adaptativo no ha requerido el uso de más pines que los que ya disponía la Arduino UNO R3, así que se ha realizando todo el desarrollo en este PFC con ella. Sí que es una placa que puede ser útil si se necesita más conectividad con buses IIC y SPI, o se van a acoplar más componentes, módulos y sensores al proyecto.

Sin embargo, no soluciona el principal problema que se ha encontrado en la implementación del control de velocidad adaptativo: la capacidad o velocidad de procesamiento, para conseguir tiempos de respuesta de la placa acordes a las funcionalidades necesarias en un entorno de ejecución real.

Conclusión de la prueba Las pruebas realizadas fueron completamente exitosas, dado que no se observó ninguna diferente apreciable de comportamiento del prototipo en el mismo circuito. Consiguientemente se confirma que es plenamente compatible hardware y software, aunque no en conexionado, y que en función de la cantidad de componentes y sensores a conectar, puede ser interesante su utilización. Si no es por esa razón, lo más práctico parece ser mantenerse con la Arduino UNO R3, pues por su menor tamaño se adapta mejor a los prototipos utilizados.

5.2.1.1.3. Intel Galileo Intel ha iniciado un movimiento para entrar en el floreciente mercado de las plataformas simples de desarrollo electrónico, como es Arduino, lanzando la plataforma Intel Galileo [26], que actualmente va ya por su segunda generación. A diferencia de Arduino que utiliza microcontroladores, la plataforma Intel Galileo se basa el microprocesador Intel Quark SoC X1000, de 32 bits, que es compatible con el conjunto de instrucciones x86. Por lo tanto, a nivel de poder computacional es mucho más potente: mejor conjunto de instrucciones, más capacidad y rapidez de acceso a memoria, y mayor frecuencia de trabajo.

El movimiento de Intel ha consistido en sumarse a la plataforma Arduino, asegurando una compatibilidad hardware y software a nivel de *pin* [5][26], y también por integrar su cadena de compilación en el propio IDE de Arduino. A nivel de hardware, Intel Galileo prefiere, por consumo, trabajar con entradas/-salidas de 3,3 voltios, pero puede trabajar a 5 voltios perfectamente simplemente mediante el cambio de posición de un *jumper*.

Por ello, en teoría, el único cambio a realizar para pasar de un montaje en funcionamiento realizado con un Arduino basado en microcontroladores Atmel (como la placa Arduino UNO R3 del SunFounder™) a usar una placa base de Intel Galileo con un microprocesador con el juego de instrucciones x86, sería simplemente intercambiar las placas bases y acoplar la tarjeta de expansión a la Galileo, manteniendo la misma distribución de cableado que se tenía ya en funcionamiento con Atmel.

A pesar de ello, las pruebas realizadas fueron un fracaso. Al igual que se ha comentado en el caso de la prueba con Arduino Mega 2560, se probó cambiar la placa base Arduino UNO R3 por la Intel Galileo, manteniendo la misma placa de extensión original de SunFounder™. No obstante, no se consiguió que el sistema funcionara. Aparentemente el programa ni se cargaba en la placa Galileo.

Posteriores averiguaciones y comprobaciones llevaron a la conclusión de que hubo durante esas pruebas dos factores que se acumularon y que se combinaron de tal forma que se impedía un correcto funcionamiento de la conexión vía USB entre el PC y la placa Galileo:

- La fuente de estabilización y conversión de voltaje de SunFounder™ sufrió algún tipo de cortocircuito o sobrecarga durante las pruebas por los repetidos desensamblajes y ensasamblajes de las placas base y de extensión. Finalmente se observó que el voltaje de salida, que debía ser de 5,0 voltios, en realidad era de 2,0 voltios, por lo que la placa no estaba convenientemente alimentada eléctricamente.
- Una actualización de Windows 10, o quizá incluso la propia instalación del software de la tarjeta Intel Galileo (se simultanearon ambas), al parecer cambió el *driver* del sistema operativo, de tal forma que el USB quedó en un estado de muy baja usabilidad: solamente era posible utilizarlo una única vez para cargar programas en las placas de desarrollo, y para reiniciar su funcionalidad se requería un total apagado y encendido en frío del PC.

El problema del *driver*, una vez detectado, se solucionó rápidamente cambiando el sistema operativo a una distribución estándar de Linux, concretamente a Fedora 24. Es más, se comprobó que la rapidez de compilación y sobre todo de carga del programa del PC a la placa base a través del USB era mucho más rápida y fiable. De esta experiencia, y viendo todos los problemas que acarrea el uso de Windows 10, se recomienda que en lo posible, el entorno de desarrollo evite este sistema operativo, recomendándose como alternativa cualquier distribución mayor de Linux, dado que el IDE Arduino es perfectamente funcional en él. Recordemos, además, que ambos son *open source*, así que su combinación es la opción más natural.

El problema de la fuente de estabilización y conversión de voltaje se solucionó adquiriendo un nuevo módulo compatible Arduino que proporcionaba el mismo voltaje permitiendo también una intensidad má-

xima nominal de 3 amperios, con picos de 4 amperios. Por ello se observa en los videos un cambio en la configuración del prototipo SunFounder™ a partir de la versión 7 de la implementación de la funcionalidad del seguimiento de trayecto y en todas las demás funcionalidades.

Conclusión de la prueba La mala experiencia sufrida con la placa Galileo (a la que no se puede culpar, salvo la sospecha de que si el problema con el *driver* del USB fue o no culpa del software de instalación de la propia placa), hace ver que los componentes de Arduino, al ser hardware comprado muy económicamente, son poco fiables, lo que es importante tener en cuenta para futuros trabajos experimentales con esta plataforma: conviene invertir un poco más y adquirir componentes y sensores Arduino de fabricantes consolidados y con buena reputación.

Dado que no se pudo realmente comprobar su funcionamiento, no se puede determinar si el incremento de prestaciones computacionales puede compensar su mayor precio, ni se ha podido tampoco comprobar si la compatibilidad hardware y software teórica es real e inmediata, o precisa ciertos ajustes a nivel de programación o de montaje más allá de la diferente disposición de los pines.

5.2.2. Prototipo 4WD

En el vídeo [44] se muestra el comportamiento del prototipo en un circuito cerrado con curvas mucho más cerradas que las que puede superar el prototipo SunFounder™.

El comportamiento es adecuado, realizándose las siguientes observaciones:

- Las líneas de 25–30 mm, al ser más anchas que las distancias entre los sensores de infrarrojos encargados de detectarlas, que es de unos 13 mm, permite prácticamente que los tres sensores estén frecuentemente visualizando la línea, por lo que una ligera desviación de ella es rápidamente detectada por los sensores, actuando en consecuencia.
- El prototipo va más rápido en las rectas que en las curvas, ya no solo porque adapta la velocidad en las curvas según su curvatura, sino también porque cuando son muy cerradas se produce la maniobra de rotación, la cual implica que las ruedas de un lateral funcionen en el sentido contrario al del otro, por lo tanto, produciendo un efecto neto de estar parado, rotando sobre el centro del prototipo.
- La velocidad general del prototipo 4WD es más rápida que la del SunFounder™, incluso aunque se prefije la misma velocidad (en realidad pulsos PWM al motor DC). Esto es así porque:
  - Cuando la línea es recta no hay apenas diferencia. Si acaso, es algo más rápido el SunFounder™ pues sus giros son más suaves que los de 4WD gracias al efecto de las medias móviles. En el 4WD cuando la línea se desvía muy ligeramente, se produce un giro que quizás es excesivo, dado que no es posible controlar el ángulo de giro o rotación del vehículo, por su tracción a cuatro ruedas fijas.
  - Cuando la curva es suave, el SunFounder™ la realiza más rápidamente, pero sin una gran diferencia, por la misma razón que la indicada en el punto anterior.
  - Cuando las curvas con cerradas, aunque sean superiores al mínimo practicable por el SunFounder™, éste decelera más para evitar salirse de la trazada, mientras que el 4WD la traza más fácilmente, sin apenas salirse.
  - Cuando las curvas son muy cerradas, el SunFounder™ debe realizar muchas maniobras de avance y retroceso para colocar adecuadamente el vehículo en la curva, mientras que el 4WD la traza más rápidamente por su capacidad de rotación sobre su propio centro.

- Aunque no se ha mostrado en el vídeo, el comportamiento ante la pérdida de la línea es semejante pero no igual. En el 4WD se ajustaron más los parámetros en el sentido de dar menos tiempo de prórroga para la confirmación de la pérdida de la línea, y necesitar menos detecciones para suponer recuperada la línea. Esto implica que reacciona más rápidamente, y suele recuperarla también más fácilmente, pero como tiene tendencia a rotar, existen los siguientes peligros:
  - Que retome la línea en sentido contrario. Este fenómeno se observa en el vídeo [43], pero también es parcialmente debido a que el sensor de la izquierda comenzaba a dar problemas pero no se había detectado todavía esa anomalía.
  - Es más probable que el prototipo 4WD se quede en una especie de bucle buscando la línea, dado que recorre menos distancia para encontrarla, sobre todo si hay mucho ruido de fondo, como en el circuito utilizado en sus pruebas. En el SunFounder™ este fenómeno es menos probable por el efecto de filtro que realiza la media móvil.
- En general, el movimiento del 4WD es más a trompicones, mientras que el SunFounder™ tiene un comportamiento más suave, debido a su sistema de dirección.

5.2.3. Conclusión de la prueba

La implementación del algoritmo se seguimiento de línea en el 4WD ha sido exitosa, pero con un comportamiento diferente al del SunFounder™, debido a sus diferentes maniobrabilidades.

El 4WD permite seguir trayectorias mucho más sinuosas en menos tiempo, aunque presenta cierto nerviosismo en líneas rectas. Es mucho más adaptable a las condiciones del circuito, pero su marcha no es tan fluida como la del SunFounder™ en los tramos en que éste puede maniobrar correctamente (cuando no hay curvas cerradas).

5.3. Seguimiento de luz

Las pruebas de seguimiento de luz sirven para comprobar la funcionalidad de la detección de luz, y conocer y ajustar el comportamiento del prototipo ante condiciones cambiantes de iluminación.

Esta funcionalidad sólo se ha implementado en el prototipo SunFounder™.

5.3.1. Montajes realizados

Se preparon dos montajes, aunque en el Anexo A sólo se detalla el segundo de ellos. En el primero se siguió la idea original proporcionada por SunFounder™ de usar los cuatros sensores de luz colocando tres de ellos en la parte frontal, para que pudieran detectar si la iluminación mayor provenía del frente, o de uno de los lados para girar hacia él. Pero sólo colocaba uno en la parte trasera. El problema está en que cuando el foco de luz se está en la parte trasera del vehículo, con un único sensor no se puede determinar hacia qué lado girar, de tal forma que sólo existe la posibilidad de dar marcha atrás, para después, cuando el foco esté delante, ya girar en su búsqueda.

Posteriormente se realizó el que se presenta en la Figura B.1, en el cual se ubican dos sensores delanteros y dos traseros, cada uno separado lo más posible del otro para que pueden obtener puntos espaciales diferentes para medir la iluminación circundante en aquellas partes que pueden ser accesibles por el prototipo con maniobras normales. Con este montaje no sólo se puede orientar el vehículo marcha adelante, sino que también, cuando la luz predominante está en la parte de atrás, el vehículo puede girar su eje marcha atrás para buscarlo directamente, sin necesidad de realizar maniobras.

5.3.2. Resultados

En el vídeo [42] se observa el comportamiento del segundo montaje ante variaciones de iluminación, y cómo el prototipo reacciona ante estas variaciones.

Las observaciones más destacables son las siguientes:

- El vehículo responde correctamente a diferencias mínimas de luminosidad.
- El vehículo maniobra correctamente tanto marcha atrás como hacia delante.
- El proceso de calibración inicial permite tener una respuesta estable de los sensores durante las pruebas.
- Esa calibración, y la capacidad de diferenciación relativa de intensidad entre los cuatro sensores, se mantiene aunque se realicen cambios bruscos de iluminación: aun quedando a oscuras, se sigue detectando correctamente una luz suave del foco, y si se recupera la iluminación intensa, la incidencia del foco en un punto cercano a ellos también es detectada.

5.3.3. Conclusiones

Se ha podido comprobar que los sensores utilizados son suficientemente sensibles como para poder utilizarlos para obtener una funcionalidad más compleja combinándolos con otro, para permitir por ejemplo, la detección de vehículos u obstáculos en la oscuridad mediante la detección de luminosidad. Además, son de muy rápida lectura y procesamiento, lo que también ayuda a esta facilidad de integración con otros sensores, al no añadir apenas sobrecoste temporal.

Los sensores utilizados proporcionan unas lecturas estables, gracias a que el método ideado de la auto-calibración se ha demostrado muy eficaz. Asímismo se ha comprobado que esa auto-calibración se mantiene durante toda la prueba. Cuando la iluminación es homogénea, incluso aunque haya cambios súbitos, el vehículo responde correctamente, permaneciendo parado.

5.4. Evitación de obstáculos

El entorno de pruebas de la evitación de obstáculos no requiere una especial preparación. Simplemente suficiente espacio con obstáculos por el medio y que lo delimiten como para comprobar si el vehículo durante su trayecto los evita.

A continuación se comentan las principales observaciones realizadas sobre el comportamiento de cada prototipo, que, recordemos, implementan algoritmos bien diferentes, dado que sus sensores son radicalmente diferentes: el SunFounder™ utiliza un sensor todo/nada por infrarrojos (reflexión de luz), mientras que el 4WD utiliza un sensor analógico, de 10 bits de resolución (valores enteros del 0 al 1023), por ultrasonidos (reflexión del sonido) .

5.4.1. Prototipo SunFounder™

Debe tenerse en cuenta que en el prototipo SunFounder™ el sensor, además de ser todo/nada, su otra importante peculiaridad es que se basa en la reflexión de la luz infrarroja. Eso quiere decir que sólo detecta objetos de color infrarrojo, esto es, que no absorba esa banda de frecuencias. Aunque nosotros no podemos saber a ciencia cierta si algo refleja el infrarrojo, sí que suele ser cierto que los objetos blancos suelen reflejarla, dado que reflejan todo el espectro visible, mientras que es de esperar que los objetos negros, no la reflejen tampoco.

Este comentario es importante, porque puede ser que el sensor no detecte algún obstáculo que realmente nosotros vemos que está ahí, pero para asegurarnos al 100 % de que es un problema achacable a la implementación o al hardware sensor, deberíamos analizar su espectro de reflexión para comprobar que si realmente refleja o no el infrarrojo.

No obstante, en general, el prototipo detectaba gran parte de los obstáculos, siendo los negros y más oscuros con los que tenía más problemas de detección.

5.4.1.1. Pruebas con el código original

El código original no proporcionaba la funcionalidad deseada ya desde el principio. Incluso después de corregir varias partes del código, el prototipo, al retroceder, entraba en un bucle indefinido dando vueltas en círculo marcha atrás, sin que ni la detección de un obstáculo lo hiciera salir de ese estado. Ese anómalo comportamiento se puede observar en el vídeo [39] se muestra esa circunstancia desde 00:35 hasta el final.

Por ello se descarta y se diseña e implementa un algoritmo original, cuyas pruebas se comentan a continuación

5.4.1.2. Pruebas con el nuevo algoritmo propuesto

El video [41] muestra el comportamiento del algoritmo de detección de obstáculos mediante sensores todo/nada implementado en el SunFounder™.

A continuación se destacan y comentan las observaciones experimentales más importantes:

- El sensor todo/nada de largo alcance por infrarrojos tiene problemas para detectar los objetos negros, dado que es muy probable que también absorban la luz infrarroja, y al no reflejarla, el sensor no lo detecta como obstáculo. (Del 00:55 al 01:05).
- El vehículo evita los choques incluso en espacios angostos, como se observa entre los tiempos de 01:28 a 02:00.
- El vehículo, aunque no posee un generador de movimientos aleatorios para expresamente intentar escaparse de esquinas, sí que las suele evitar correctamente, como se observa en el vídeo, de 02:42 a 03:20.
- Cuando no hay suficiente espacio para retroceder, y estar frente a una pared, el barrido del sensor no es capaz de determinar una salida suficientemente grande, y se queda en un bucle continuo. Esta es una situación límite, y se puede observar en el vídeo, entre 02:02 y 02:25.
- Se observa que posiblemente el tiempo que se le dio para la estabilización del servomotor sea demasiado corto. Es por ello que en la siguiente funcionalidad de control de velocidad adaptativo, ya se tiene ese factor en cuenta para que, en función del arco que deba recorrer el servomotor para ubicar el sensor (ya sea de infrarrojos o por ultrasonidos), se le dé más o menos tiempo para continuar el procesamiento del algoritmo.

5.4.1.3. Conclusiones

El algoritmo ideado ha mejorado mucho el comportamiento del prototipo, de tal forma que ahora sí que es funcional. Se podría ajustar algo más, puesto que debería tenerse en cuenta el tiempo que necesita el servomotor para alcanzar el ángulo deseado antes de ordenar al sensor que mueve que tome lectura.

El comportamiento es reactivo, y en general evita los obstáculos muy bien si éstos son de tonalidad clara. Cuanto más oscuros son, necesita estar mucho más cerca para detectarlos, pudiendo llegar a colisionar levemente por la propia inercia del prototipo antes de poder detenerse.

Sin necesidad de realizar movimientos aleatorios, es capaz de resolver el problema de las esquinas, e incluso de escapar de callejones de salida angostos. Sólo se encaja en el problema de difícil solución de estar perpendicular a una pared larga sin espacio para retroceder, pero dada la baja maniobrabilidad el prototipo es algo comprensible. No obstante, si se añadiera algún detector de situación de bucle, y tras él un generador de movimiento aleatorio, o de movimiento de giro siempre en el mismo sentido, este problema probablemente desaparecería.

5.4.2. Prototipo 4WD

El prototipo incorpora un único sensor de obstáculos, basado en la detección del eco de una señal propia de ultrasonidos, el cual está acoplado a una plataforma giratoria accionada por un servomotor con un ángulo de giro de 180°.

Las pruebas fueron exitosas, dado que en ningún momento se atascó, aunque sí que se detectaron un par de colisiones, las cuales se analizarán.

A continuación se remarcan algunos comportamientos que se observan en las pruebas. Los tiempos indicados se refieren al vídeo [40].

- Al utilizar sensor por ultrasonidos, la oscuridad no le afecta (00:27 a 00:40).
- El prototipo presenta una rápida detección y toma de decisión ante obstáculos frontales y laterales, sobre todo si son más o menos perpendiculares a su marcha (00:08 a 00:17).
- El algoritmo de resolución de esquinas es eficaz, puesto que en todo el vídeo de más de seis minutos no se ha quedado atascado en ninguna situación. Puede observarse su comportamiento en varios momentos (01:28 a 01:47, y de 04:26 a 04:48).
- Tiene cierta capacidad para detectar incluso los obstáculos laterales que son bastante paralelos a la dirección de marcha, y por lo tanto, de difícil detección por ultrasonidos, dado que detecta el eco, y una pared muy paralela a la marcha ofrece poco eco (01:49 a 01:52). Nótese que se trata de una puerta de madera maciza, así que la reflexión parece haber sido suficiente, al tratarse de un material duro, barnizado (no poroso) y poco absorbente.
- Sin embargo, hay unos otros obstáculos laterales bastante paralelos al sentido de marcha que sí que presentan importantes problemas de detección, tanto es así que repetidamente ha chocado con ellos (03:04 a 03:10). La explicación es que se trata de una valla hueca de plástico rugoso. Por lo tanto, con cierta capacidad de absorción del sonido. A ello cabe atribuírsele la razón de su no detección.
- Es sorprendente la capacidad demostrada de atravesar una zona con múltiples obstáculos estrechos, como son las patas de las sillas y las mesas que sortea en la cocina (03:45 a 04:12).
- Al igual que antes, y combinada la zona de patas con varias esquinas, el prototipo también ha sido capaz de sortear la situación exitosamente (04:47 a 06:53).
- El otro momento en que ha habido una colisión se observa en el tiempo 05:40, en donde choca con un saliente de la pata central de la mesa. No parece ser un problema del algoritmo, dado que se trata de un obstáculo de una altura de unos 9 cm, esto es, inferior a la altura del sensor. Por lo tanto, ese choque se atribuye a que el sensor no ha recibido el eco de un obstáculo más bajo que la fuente

del sonido porque el sonido deberá haber rebotado hacia el suelo, no pudiendo ser detectado por el sensor.

- Se observa que el servomotor necesita medio segundo más o menos en realizar el barrido entero de 180°, pero que en cuanto termina ese escaneo, el tiempo de decisión es corto.
- También se observa que dado que el prototipo avanza a mucha velocidad, cuando se detecta un obstáculo se recorre una cierta distancia que hace que se aproxime mucho y casi llegue a chocar. El ajuste de la distancia es vital para que no se produzca el choque, y la maniobra de retroceder es la que le permite obtener un ángulo de visión mayor por los laterales tras aproximarse tanto a una pared o obstáculo frontal.

Por todo ello, se considera que esta implementación cumple perfectamente los objetivos de la funcionalidad de evitación de obstáculos.

5.4.3. Comparación entre las diferentes pruebas

De las pruebas realizadas se extrae la conclusión de que, como ya cabía de esperar, la detección de obstáculos funciona mejor con los sensores analógicos (basados la medición del tiempo de ida y vuelta en el eco de los ultrasonidos) que con los sensores binarios de todo/nada (basados en la detección del reflejo de la luz infrarroja).

Hay varias razones para ello:

- Los sensores analógicos siempre proporcionan más información que los binarios todo/nada. De hecho, un sensor analógico puede convertirse en un sensor binario todo/nada simplemente comparando su resultado con un valor umbral dado.
- Los sensores binarios de todo/nada se deben calibrar manualmente, generalmente mediante potenciómetros, para fijar su umbral de activación. Los analógicos no dependen tanto de esa calibración manual, porque pueden auto-calibrarse por software, como se ha realizado en este proyecto con los sensores de intensidad de luz.
- La recepción del eco de los ultrasonidos son más fiable que la recepción del reflejo de la luz infrarroja. Al menos eso se ha observado en las pruebas. En general, el eco del ultrasonido se genera siempre que el obstáculo no sea un absorbente o gran difusor de las ondas sonoras. Esos materiales suelen ser menos frecuentes que los materiales de color negro dentro del espectro de los infrarrojos. Hay muchos materiales y colores que absorben casi completamente la luz infrarroja, y por lo tanto son invisibles a los detectores basados en su reflexión.
- Es mucho más frecuente la contaminación lumínica que la contaminación por ultrasonidos.
- Sin embargo, el tiempo de medición y lectura del sensor de ultrasonidos es mucho más lento que el de la reflexión de la luz infrarroja. Esto es normal, dado que la velocidad del sonido es de unos 340 m/s mientras que la de la luz es de 300.000.000 m/s.

5.5. Control de velocidad adaptativo (ACC)

Las primeras pruebas del ACC consistieron en la comprobación visual de que la implementación de la funcionalidad en el 4WD funcionaba correctamente. Para ello se probaba moviendo manualmente una caja de cartón delante del prototipo mientras éste se desplazaba en línea recta, para comprobar que el contro-

lador PID funcionaba correctamente, y además, ajustar los parámetros de control, esto es, la ganancia y los términos integrado y derivativo.

Pero esas pruebas no eran concluyentes, pues un buen control de velocidad adaptativo debe probarse no sólo con todos los elementos activos, sino también en un entorno real, con un obstáculo realmente dinámico y que recorra la misma trayectoria, siendo esta no lineal, sino también con curvas, para analizar el comportamiento del ACC durante los giros también.

En lo que sigue, se desarrolla, describe y comenta la segunda prueba.

5.5.1. Diseño experimental

El diseño de la segunda y definitiva prueba experimental presenta las siguientes características:

- Un circuito cerrado, con variedad de líneas rectas, donde los coches puedan acelerar sin problemas, y con curvas moderadas y curvas algo más cerradas, para que se observe también el mantenimiento de distancia en las curvas y sobre todo que no se colisiona en ningún momento con el coche precedente.
- El circuito está representado por una línea negra de 50 milímetros de grosor. Se opta por una línea bien ancha para evitar problemas en las curvas, pues lo que interesa en estas pruebas es que los vehículos rueden fluidos, sin realizar maniobras extrañas.
- Dos prototipos, uno delantero con velocidad variable, pero de forma predecible, para poder estimar en qué estado está, y otro perseguidor, que implementa el ACC, que debe mantener la velocidad o conservar la distancia si el precedente se acerca demasiado.
- En el coche precedente se le añade una caja en la parte trasera. Esto se hace así porque el sensor de ultrasonidos del perseguidor está en una posición algo más elevada que la parte trasera del precedente. Al añadir la caja se asegura la reflexión de las ondas de ultrasonidos de los sensores, y con ello, se asegura también la correcta medición de la distancia existente entre los dos prototipos. Esto es especialmente relevante en las curvas, pues los laterales del SunFounder™ son aún más bajos.
- Las velocidades se seleccionan de tal forma que sean parecidas y elevadas, pero el delantero debe ser, en promedio, más lento que el trasero, para que este último lo alcance de cuando en cuando y deba controlar su velocidad, gestionando adecuadamente el espacio entre ambos.
- Se hacen varias pruebas iniciales al objeto de ajustar los parámetros de los programas que se ejecutan en cada prototipo, para ajustarse al circuito y para conseguir el comportamiento deseado.
- La iluminación se intenta mantener constante en todo el circuito, evitando en lo posible la existencia de sombras que pudieran confundir a los sensores de detección de línea.

En la Figura 5.1 se muestra el circuito utilizado para la prueba. Obsérvese que se ha evitado trazar curvas muy cerradas, para que el SunFounder™ pueda tomarlas con la suficiente facilidad como para que no disminuya demasiado su velocidad o empiece a realizar maniobras que desvirtúen la prueba. El tramo que hay en la parte superior, más cercana a la pared frontal, no forma parte del circuito. En el circuito se encuentran además los dos prototipos, a la izquierda el 4WD y a la derecha el SunFounder™.

5.5.2. Montajes de los prototipos

En el SunFounder™ se implementa el seguimiento de líneas, pero incluyendo un variador de velocidad máxima que se activa por un temporizador. Así, cada cinco segundos se cambia de velocidad lenta a velo-

Figura 5.1: Circuito de prueba de la funcionalidad del ACC

cidad rápida, y viceversa. Es el vehículo que circulará por delante y que servirá de obstáculo móvil con velocidad variable para que el vehículo que implementa el ACC deba gestionar correctamente la situación, sin colisionar y manteniendo la distancia entre unos márgenes predefinidos siempre que la velocidad máxima de los dos vehículos lo permita. En el Listado D.1 se muestra el código de la implementación.

El prototipo 4WD ejecuta el algoritmo del control de velocidad adaptativo, con los parámetros ya ajustados a la situación real. Además de los sensores de línea, se le han acoplado tres sensores fijos frontales de detección de obstáculos por ultrasonidos. Según la distancia que detecten estos sensores, se controlará la velocidad del 4WD para conseguir que éste nunca colisione con el SunFounder™, se desplace al máximo de la velocidad posible, siempre sin rebasar su propia velocidad máxima de crucero, y que cuando se ponga detrás del 4WD porque éste circule más lento, se mantenga a una distancia prudencial. En el Listado D.2 se muestra el código de la implementación.

5.5.2.1. Elección de los sensores de detección de obstáculos

En el 4WD se han incorporado tres sensores de distancia por ultrasonidos, fijos frontales. Eso es una variación importante sobre la implementación de la funcionalidad de la evitación de obstáculos que se ha presentado en el Punto 4.4.2.1. Recordemos que en la implementación de la funcionalidad básica se utilizaba un único sensor de distancias por ultrasonidos, y que éste estaba montado sobre una plataforma giratoria accionada por un servomotor.

A pesar de tener que rediseñar completamente la parte de la evitación de obstáculos, esta decisión se ha tomado por las siguientes razones:

- La detección tiene que ser lo más rápida posible, evitando retrasos debidos a la necesidad de esperar a que el sensor recorra el arco necesario para posicionar el detector en la dirección deseada.
- Al poner los sensores fijos se baja unos 3 centímetros la altura del sensor de ultrasonidos, por lo que se ajusta algo más a la parte trasera del SunFounder™.
- Hay implementaciones actuales de sistemas parecidos al ACC con ambas soluciones, con varios sensores o con un sensor único: probar la utilización conjunta de los tres sensores iba a favor de la filosofía de este PFC en lo que respecta de estudiar la adaptabilidad del hardware a diferentes entornos y comprobar su comportamiento en situaciones reales.
- Se había adquirido el material y se quería probar cómo se podían combinar varios sensores iguales tanto en la programación como en la práctica: es una razón más peregrina, pero que también entró en consideración.

Como peligros de este montaje estaban los siguientes, los cuales no se han materializado, y por lo tanto, no han obstaculizado la instalación y uso de los tres sensores fijos:

- La lectura de los tres sensores a la vez puede retrasar el tiempo de ciclo: aunque el cómputo ha resultado muy pesado, ha sido posible obtener una solución final de compromiso con resultados aceptables. Además, se ha mejorado el algoritmo, y sólo se realizan las medidas necesarias, pudiéndose a veces realizar dos o incluso una única medida según las posiciones relativas de los prototipos.
- Posibles interferencias entre los sensores: riesgo de que el segundo sensor, al activarse, pudiera oír las reflexiones de los ultrasonidos generados por el primer sensor, desvirtuándose su medida. No se ha detectado ese problema.
- Falsos positivos de obstáculo por detectarse las paredes cercadas: al igual que en el primer caso, el efectuar la medida sólo de los sensores necesarios disminuye este riesgo. Además, en el diseño experimental ya se ha tratado de alejar las líneas de la pared. Pero incluso en la línea inicial de pruebas más cercana a ellas, ese fenómeno no pareció ser relevante.

5.5.2.2. Nuevas modificaciones en el montaje

Por razones imprevistas, el montaje final sufrió otra modificación: se incorporó un módulo de 8 canales con sendos sensores por ultrasonidos que se comunica con la placa Arduino UNO R3 gracias al bus IIC. Dado que las pruebas fueron costosas, se estaba utilizando también otro componente que usaba ese bus, un visualizador de caracteres tipo LED para visualizar datos de depuración durante la prueba en real.

En principio, el uso de dos componentes IIC con el Arduino es perfectamente posible: la placa base se configura como maestro del bus, y los otros dos módulos como esclavos. El problema está en que el bus IIC requiere que en esas configuraciones, las señales `SDA` y `SCL` sean del tipo triestado, quedando en alta impedancia esos puertos cuando no estén enviando o recibiendo datos. Para que ese montaje funcione es necesaria la instalación de resistencias de *pull-up* que conectan el bus (línea común) de cada señal (`SDA` y `SCL`) con la alimentación `+5V`. Lo ideal es que el valor de esas resistencias se determine en función de las características de todos los componentes que comparten el bus, pero una regla bastante habitual consisten en colocar resistencias de `10`kΩ, y comprobar si funciona correctamente.

En la Figura 5.2 se muestra el esquemático de esta configuración.

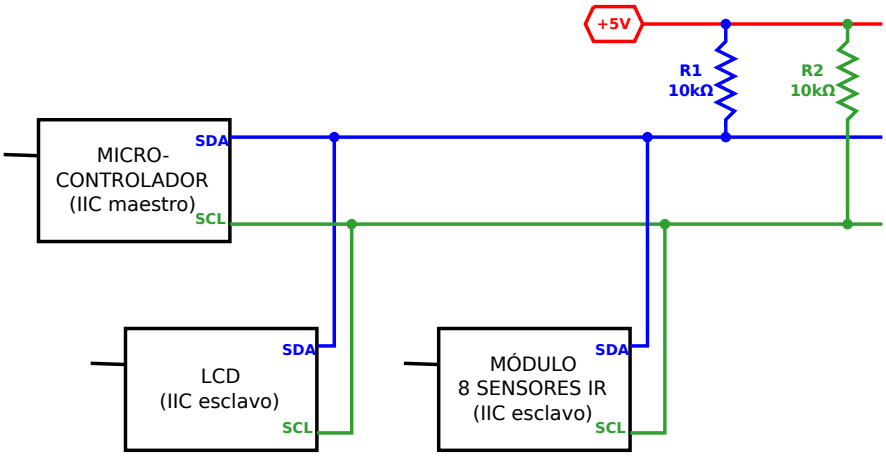

Figura 5.2: Esquema de montaje de varios nodos en el bus IIC

En nuestro caso, no se disponía en ese momento de resistencias de `10kΩ`, por lo que se probó con resistencias de `1kΩ`, y dado que todo aparentemente funcionó a la primera, se dio por buena ese montaje con resistencias de `1kΩ`.

En la Figura D.1 se detalla el conexionado especial realizado para conectar los dos componentes (módulo rojo de 8 canales de sensores por infrarrojos y el visualizador LCD) a los pines de Arduino UNO R3 correspondientes al bus IIC. Al igual que en esa figura, en la realidad también se utilizó una placa base de prototipado (*breadboard*) de tamaño mini. Esas conexiones parecen endebles, y aparatosas, pero una vez bien acomodada la placa de pruebas al chasis del prototipo 4WD no dio especiales problemas durante las pruebas. No obstante, para prototipos de más duración, habría que buscar una solución más compacta y robusta.

En el apartado software, el disponer de más o menos componentes conectados al mismo bus IIC no representa ningún problema, dado que la biblioteca estándar `Wire.h` de Arduino gestiona transparentemente esta configuración.

5.5.3. Realización experimental

A continuación se detallan las principales incidencias habidas durante la realización de las pruebas del ACC, las cuales hubo que solucionar para llegar a la solución definitiva presentada en este proyecto.

La descripción de estos problemas y de cómo se han resuelto forman parte importante del *know-how* aprendido durante el proyecto, y que se considera interesante reflejar en esta memoria para las futuras líneas de investigación del proyecto DINAMOS relacionadas con los prototipados basados en Arduino.

5.5.3.1. Conocimiento de la dinámica de los motores DC

Una actividad que fue necesaria hacer fue la comprobación experimental sobre cómo la variación del pulso PWM afecta al cambio de velocidad. Es importante tener en cuenta que en los prototipos utilizados, en la combinación del motor DC y de las ruedas y chasis utilizados, existe un pulso mínimo de PWM para que el motor realmente consiga actúe y consiga hacer girar las ruedas. El conocimiento de la relación entre el pulso PWM y la velocidad real del prototipo obtenida es importante para poder estimar los valores iniciales del controlador PID, así como para definir los parámetros de muchos de las implementaciones presentadas

PWM (pulsos)	Distancia (m)	Tiempo (s)	Velocidad (mm/s)
255	14.0	16.0s	875
240	14.0	18.0s	778
225	14.0	19.0s	737
210	14.0	20.3s	691
190	14.0	22.4s	625
170	14.0	24.7s	567
150	14.0	29.9s	468
130	14.0	37.9s	369
120	14.0	44.8s	312
110	14.0	53.7s	261
100	14.0	73.3s	191
90	14.0	108.5s	129

Tabla 5.2: Variación de la velocidad del prototipo 4WD con el pulso PWM

en este proyecto, en el que la variación de la velocidad juega un papel importante, sobre todo en este de control de velocidad adaptativo.

Esta curva se obtuvo para el prototipo 4WD, que es el que implementa el ACC.

La realización experimental consistió en hacer circular el 4WD a una velocidad fija (con un determinado valor prefijado del pulso PWM accionando sus motores) por una superficie plana y libre de obstáculos de unos 14 m.

Se realizaron tres mediciones coherentes (se rechazaron valores visiblemente anómalos) de cada una de las varias velocidades seleccionadas, que cubrían el espectro de pulso PWM viable para el prototipo seleccionado. Estas mediciones se realizaron con el estado de las baterías en el nivel medio-bajo (entre 7,68 V y 7,51 V).

Aunque la alimentación de los motores está asegurada a 5,0 V por un módulo de alimentación y gestión de motores propio, el estado de las baterías podría afectar a la velocidad efectiva del vehículo, por la intensidad de corriente que les proporciona. Simplemente se hace notar para mejor referencia.

En la Tabla 5.2 se muestran los resultados experimentales de estas pruebas, los cuales se representan gráficamente en la Figura 5.3, y que ajustan muy bien a dos rectas con el punto de intersección en `PWM=161`. En esa gráfica aparecen las dos fórmulas experimentales que permiten, según su intervalo de aplicación, estimar la velocidad real del prototipo 4WD cuando marcha en línea recta, en función de la señal de pulsos PWM que se suministre uniformemente a sus dos motores.

5.5.3.2. Problemas al incluir un nuevo nodo en el bus IIC

Las pruebas fueron costosas principalmente porque se estropeó uno de los tres sensores que conformaban el módulo que se utilizó en la implementación del seguimiento de líneas en el 4WD.

Una vez detectado que el problema era hardware, se tuvo que cambiar ese componente. El problema es que en esos momentos no se disponía de un sensor similar, y se tuvo que recurrir a un módulo de SunFounder™ que tenía integrados ocho sensores. Esto supuso un doble cambio: modificar el algoritmo, y con él, el programa para implementar el seguimiento de líneas con el nuevo módulo; y utilizar un nuevo

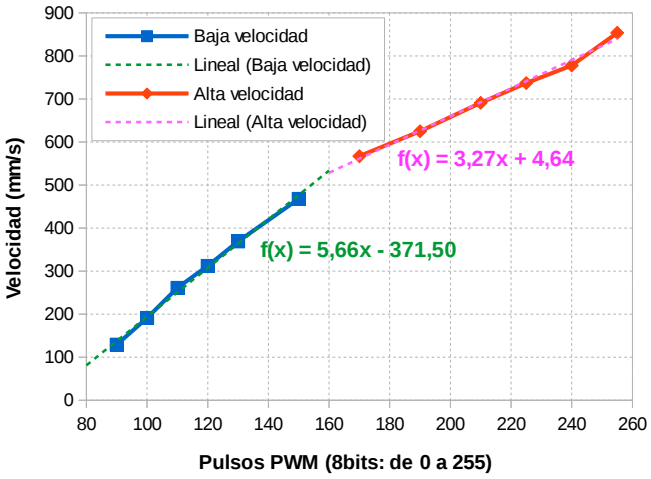

Figura 5.3: Ajuste de la velocidad respecto al pulso PWM

sistema de conexionado, pues en lugar de tener ocho entradas/salidas, una para cada sensor, dispone de sólo cuatro pines porque el módulo utiliza el bus IIC.

En el Punto 5.5.2.2 ya se ha discutido la parte teórica, así como la modificación que hubo que realizar.

5.5.3.3. Parametrización de la parte del seguimiento de líneas

Solucionados los problemas relacionados con el bus IIC, se comprobó que en el prototipo 4WD había mucha inestabilidad al recorrer el circuito, circunstancia que no sucedía cuando lo recorría el mismo prototipo con los anteriores sensores. Por eso se añadió un parámetro (`DEBUG0`) para desactivar la funcionalidad del ACC en esta implementación. Se comprobó que tras desactivar el ACC el robot funcionaba correctamente de nuevo, aunque quizá un poco más nervioso. Eso dio la pista de que el problema era debido a un excesivo tiempo de ciclo, denominando así al tiempo que le cuesta a la placa Arduino a realizar una ejecución completa de la función `loop()`.

Tras múltiples pruebas, el ajuste correcto de la parte relacionada con el seguimiento de líneas consistió en desactivar (esto es, poner a cero) todos los parámetros `DELAY_*` que se habían introducido en la implementación inicial del seguimiento de líneas (véase la Tabla 5.3). Sin embargo, si se hace lo mismo con el parámetro `DEBUG0` activado (que implica desactivar el ACC), el comportamiento del prototipo no era adecuado, puesto que se quedaba atrancado en las maniobras de rotación por falta de par motor para ejecutar el movimiento de rotación.

La explicación de este fenómeno es sencilla: el tiempo de actuación de los motores es demasiado corto para realizar esa maniobra. Ésa es precisamente la razón inicial de porqué se incluyeron: para dar tiempo a los motores a activarse y poder vencer la inercia de las ruedas ante un cambio de movimiento, sobre todo si implica cambio de sentido de rotación de las ruedas.

Una vez puesto los retrasos a cero, el tiempo de actuación de los motores depende del tiempo de ciclo del Arduino, el cual se ha elevado significativamente al incluir la funcionalidad del control de velocidad adaptativo, especialmente por dos factores: la necesidad de leer tres sensores de distancia por ultrasonidos, que son de lectura lenta (unos 10 ms cada uno), y por el tiempo de cómputo del control PID de la señal de error.

Como estos tiempos ya tienen una duración igual o superior a los que en la implementación original del seguimiento de trayecto se tuvieron que añadir artificialmente, éstos ya se podían eliminar sin efecto dañino para esa funcionalidad en el ACC.

5.5.3.4. Parametrización de la parte del control PID

Una vez todo más o menos en orden, se hace preciso ajustar los parámetros del controlador PID, que es el encargado de variar la velocidad del vehículo perseguidor para mantener una velocidad de crucero constante, y si encuentra un obstáculo, ajustarse a la velocidad de éste intentando mantener una distancia prefijada al mismo, así como asegurarse de que no se rebase la distancia mínima de seguridad para evitar colisionar con él.

Existen seis parámetros, y los seis tuvieron que ser ajustados, pues están interrelacionados, y además se encontró que, como se esperaba, tuvieran influencia en el comportamiento del vehículo en las pruebas.

A continuación se resumen las razones por las cuales se eligieron los valores presentados:

Máximo error de la señal (`MAX_ERROR`) El error de la señal es el resultado de la aplicación de los términos proporcional, integral y derivativo al error del valor leído. Se limita a 30, porque es el valor que permite definir el nuevo valor deseado de la velocidad. Si este valor es mayor, entonces las modificaciones de la velocidad son demasiado bajas, y por lo tanto, el prototipo es poco reactivo a los cambios de velocidad del vehículo precedente, lo que obliga a abusar de los frenazos, o a que al acelerar el precedente, no se responda a tiempo, quedándose rezagado sin necesidad.

Error máximo en la distancia (`DIST_MAX_ERR`) Se ha fijado en 20 cm. Ésa es la máxima diferencia tolerable entre dos medidas de distancia consecutivas. Se sabe que la velocidad máxima de nuestro vehículo es de unos 90 cm/segundo (ver Tabla 5.2). El tiempo de ciclo máximo medido en algunas pruebas llegó a ser de 150 ms. Por lo tanto, el vehículo durante esos 0,15 segundos, a máxima velocidad puede recorrer unos 14 cm. Dado que el otro vehículo también puede variar su velocidad, se consideró oportuno definir como máxima distancia esos 20 cm para estas pruebas. Un valor superior no puede fundamentarse físicamente, por lo que probablemente se trate de un error de medición del sensor.

Error máximo del factor integral (`DIST_MAX_INT`) Sirve para limitar la parte integral. La parte integral ayuda a evitar el desplazamiento (*offset*) entre la señal generada y el valor real de la variable a controlar. No conviene que sea muy grande, puesto que la dinámica de la circulación puede ser muy rápida, y ante eventos tales como frenados repentinos del auto precedente, o de que otro vehículo se interponga en nuestro carril, el sistema debe reaccionar rápidamente, por lo que debe primar siempre la parte derivativa sobre la integral. Un ligero desfase en la distancia mantenida con el coche precedente que no consiga corregir la parte integral no es importante, pero sí lo es una respuesta demasiado lenta que provoque excesivos frenazos por deficiente control sobre la distancia mantenida.

Ganancia (`KPP`) Se ha fijado en 0,5. Si el error en la distancia es digamos de 10 cm, entonces la señal generada sólo por la ganancia sería de 5, que es la sexta parte del máximo permitido (recordemos que se ha definido `MAX_ERROR=30`). Por lo tanto, ante ese cambio, la velocidad variaría una sexta parte entre el máximo y mínimo de las velocidades permitidas, lo cual, para un bucle de unos 50–150 ms se considera ya un cambio importante pero oportuno de velocidad.

Parte integral (`KPI`) Como ya se ha comentado en el análisis del valor del error máximo del factor integral (`DIST_MAX_INT`), en el caso de la circulación real de vehículos no parece interesante que sea un factor dominante en el control inteligente de vehículos. Si lo fuera, ajustaría bien la distancia al precedente en el estado estacionario, pero a costa de una menor responsividad.

Parte derivativa (KPD) Se la ha dado un valor considerablemente elevado, 40. Esto es así porque interesa que un repentino cambio brusco en la distancia respecto al vehículo anterior produzca un rápido cambio de velocidad, sobre todo si la distancia ha disminuido, al objeto de evitar un choque, que es una restricción básica de seguridad en la funcionalidad del control de velocidad adaptativo. En el control de velocidad de vehículos, la parte derivativa debe ser predominante, sobre todo si los sensores son de alta fiabilidad. Si no lo son, debería confirmarse rápidamente ese valor anómalo, para evitar frenazos innecesarios que alteren la fluidez de la conducción, y que además, pueden ser focos de accidentes, con respecto a la acción que deban tomar los vehículos que circulan detrás.

5.5.3.5. Ajuste de velocidades

El ajuste de las velocidades se ha realizado de una forma ortogonal a las anteriores, puesto que conforme se han ido solucionando los otros problemas, se ha podido incrementar la velocidad máxima de los dos prototipos, para dotar de mayor vivacidad y representatividad a las pruebas.

Al final se ha conseguido ajustar las velocidades a las que se muestran en los códigos correspondientes a la implementación de cada prototipo, y que han permitido realizar las pruebas exitosamente.

El objetivo de estos parámetros de velocidad es que el coche delantero (SunFounder™) varíe su velocidad de forma que periódicamente sea más rápido y más lento que el perseguidor (4WD), para forzar la variación de la distancia que los separa, y además, que la velocidad promedio por vuelta del delantero sea inferior a la del perseguidor, para que éste tenga la posibilidad siempre de alcanzarlo y así poder comprobar el comportamiento de la funcionalidad de ACC.

5.5.4. Parametrización definitiva del ACC

En la Tabla 5.3 se detallan los valores de los parámetros del control de velocidad finalmente utilizados en esta prueba. Sus significados fueron explicados en el punto 4.4.2.1.2.

Parámetro(s)	Valor	Descripción
SPEED_MAX	250	Velocidad máxima de crucero, grande para alcanzar al SunFounder™
SPEED_HIGH	230	Velocidad máxima si la línea se detecta sólo en dos sensores
SPEED_MED	210	Velocidad máxima si línea detectada más en un flanco, si no es extremo
SPEED_LOW	180	Velocidad máxima si línea sólo en un flaco, incluido el sensor extremo
SPEED_MIN	150	Si se pone menos, el 4WD se atranca al parar o rotar
DIST_MAXIMUM	90	Distancia máxima para utilizar el controlador de velocidad
DIST_DESIRED	60	Intentar mantener una distancia de 60 centímetros
DIST_MINIMUM	30	Distancia mínima a respetar con el vehículo de enfrente
DIST_WARNING	10	Distancia mínima de seguridad, para los sensores de los flancos
MAX_ERROR	30	Valor experimental. Si es más grande, fluctúa más
DIST_MAX_ERR	20	Diferencia entre dos distancias seguidas. No tiene sentido que sea mayor
DIST_MAX_INT	100	Limitación del error integral. Para que no se acumule mucha historia
KPP	0,5	Valor experimental. Si es muy grande, fluctúa más
KPI	0,2	Si se incrementa mucho, tiene demasiada memoria
KPD	40	Para responder rápidamente a bruscos cambios de velocidad (frenazos)
All DELAY_*	0	No perder tiempo. El cómputo ya es muy costoso temporalmente

Tabla 5.3: Valores de los parámetros del ACC en la prueba real

5.5.5. Resultados

El comporamiento de los dos vehículos durante la prueba se pueden observar en el vídeo [38].

A continuación se destacan y comentan las observaciones experimentales más importantes de la prueba del control de velocidad adaptativo:

- Se observa un funcionamiento correcto de ambos prototipos, y por lo tanto, también de la funcionalidad de control de velocidad adaptativo implementado en el 4WD.
- El 4WD es, en tiempo por vuelta, más rápido que el SunFounder™, lo cual es requerido para que eventualmente siempre alcance al SunFounder™ y deba activar su control de velocidad.
- El prototipo SunFounder™ efectivamente varía su velocidad periódicamente, lo que provoca que su distancia con el 4WD varíe.
- Si el camino está despejado, el 4WD recorre el circuito a su velocidad máxima de crucero, adaptando ésta a las curvas, como ya hacía en el seguimiento de trayecto.
- En todos los casos el perseguidor detecta la presencia del delantero, incluso en las curvas cerradas, evitando siempre la colisión.
- Cuando se ve forzado a detenerse por la excesiva baja velocidad del SunFounder™, que tiene una velocidad mínima inferior al 4WD, la distancia depende del procedimiento que esté siguiendo: si es en avance frontal, la distancia a partir de la que frena es de 30 cm (`DIST_MINIMUM`), pero si es en una curva cerrada, esa distancia es mucho más corta, porque en la parametrización la distancia de emergencia para los sensores laterales (`DIST_WARNING`) es de tan solo 10 cm.
- Cuando el 4WD alcanza al SunFounder™ modera su velocidad para intentar adaptarse a la del delantero, intentando mantener una distancia, que está prefijada en 60 cm pero se le permite variar entre 30 y 90 cm para acabar de ajustarse (parámetros de control).
- En las pruebas se puede comprobar la variación de velocidad debida a la acción del controlador no tan solo visualmente, sino también por el ruido realizado por el cambio del pulso PWM que actúa sobre los motores del prototipo perseguidor, 4WD.

5.5.6. Conclusiones

De las pruebas realizadas se concluye que la implementación del control de velocidad adaptativo en un prototipo basado en Arduino, con una placa poco potente computacionalmente, como lo es la Arduino UNO R3, ha sido completamente exitosa, dado que se han cumplido todos los objetivos que fijaba la especificación de requisitos de dicha funcionalidad (ver la Sección 3.5).

No obstante, la capacidad computacional de Arduino UNO R3 está en sus límites con esta funcionalidad combinada.

Por lo tanto, en caso de que en el futuro se desee expandir esta funcionalidad combinándola con otras y probarlas experimentalmente, se recomienda que ya desde el inicio se valore la posibilidad de cambiar esa placa base por otra con más prestaciones computacionales.

En cuanto a entradas/salidas todavía quedan algunas libres, pero quizá esa futura expansión de la funcional también requiera pasar a una placa base con mayor conectividad y más entradas/salidas.

Por lo tanto, la plataforma Arduino es perfectamente viable como medio experimental para comprobar los desarrollos que se realicen en el proyecto DINAMOS, al menos en estas y próximas etapas de investigación.

CAPÍTULO 6

CONCLUSIONES

En el presente Proyecto Fin de Carrera se ha estudiado el problema de la navegación autónoma de los vehículos desde un punto de vista práctico, mediante la implementación en dos prototipos diferentes basados en la plataforma Arduino de algoritmos básicos como son el seguimiento de trayecto, el seguimiento y detección de luz, y la evitación de obstáculos.

Para ello se ha partido de código suministrado por los vendedores de los coches Arduino o buscando mejores implementaciones en Internet. Se ha comprobado que esos algoritmos, o las implementaciones consultadas, no eran suficientes para asegurar un funcionamiento correcto de los mismos sobre los coches probados. Por ello ha sido necesario formalizar y reimplementar esos algoritmos, o incluso idear unos nuevos, como en los casos del seguimiento de trayecto con coches del tipo SunFounder™, y de la evitación de obstáculos usando sensores todo/nada.

Con ello se han obtenido implementaciones individuales para cada uno de los dos prototipos utilizados, que tienen características diferentes que obligan a adaptar cada algoritmo a su propias especificidades.

A partir de ahí se han combinado dichos algoritmos individuales para proporcionar una nueva funcionalidad al vehículo autónomo: el control de velocidad adaptativo. Se ha comprobado que es posible realizarlo incluso con los pocos recursos computacionales que proporciona la placa base Arduino UNO R3, aunque se comprueba que ya se está en los límites, puesto que ya se está ante tiempos de respuesta en el peor de los casos en torno a los 100–130 milisegundos. Este tiempo no sólo es debido al propio cómputo, sino que gran parte de él (en torno al 60 % es debida a la latencia introducida por la captura y posterior lectura de los sensores físicos de los que dependen las funcionalidades utilizadas.

Finalmente, se comprobado el correcto funcionamiento de la implementación del control de velocidad adaptativo (ACC) haciendo circular por el mismo circuito a dos coches, uno delantero variando su velocidad aleatoriamente, y el segundo, que implementa el ACC, debiendo adaptarse dinámicamente y en tiempo real a la variación de distancia entre ambos para mantener una distancia mínima de seguridad mediante la variación de su velocidad de forma automática, y sin sobrepasar la velocidad máxima establecida.

Con respecto a la plataforma Arduino es importante remarcar la baja fiabilidad y durabilidad de su hardware, al menos las de algunos de sus sensores. Por ello es recomendable no escatimar recursos económicos a la hora de elegirlos, para evitar retrasos en la fase experimental no debidos propiamente al desarrollo investigado.

Como conclusión final, se ha comprobado la validez de la plataforma Arduino para el desarrollo de prototipos que implementen funciones avanzadas novedosas en el sector de la automoción.

CAPÍTULO 7

TRABAJO FUTURO

Tras comprobar la viabilidad y acercarse a conocer los límites de la plataforma Arduino como soporte experimental para el proyecto de estudio de la seguridad y viabilidad de tecnologías tendentes a conseguir un incremento de la autonomía de los automóviles respecto al factor humano, se proponen las siguientes actuaciones como futuras etapas dentro de la línea de investigación global.

- **Estudiar el impacto de utilizar mejores sensores**: Esta línea consistiría en mantener como plataforma de desarrollo Arduino UNO R3 o incluso Arduino Mega, pero analizar qué mejora en tiempos de respuesta se obtienen con sensores más capaces y de mayor calidad (más precisión, más fiabilidad y rapidez en las lecturas y más durables). La idea de esta línea es mantener el coste del microcontrolador limitado, y ver si con mejores sensores se pueden obtener resultados consistentes. La economía de escala en la fabricación de mejores sensores debería mantenerlos a precios razonables.
- **Utilizar microprocesadores más potentes**: La utilización de plataformas más potentes tales como el Arduino Yún o, sobre todo, la plataforma Intel Galileo. Con ello se permitiría ampliar el número de sensores y sobre todo, la cantidad de lógica de los algoritmos, para aprovechar la mayor capacidad de cálculo para obtener menores tiempos de respuesta con unas reacciones más precisas.
- **Estudiar la intercomunicación vehicular mediante redes inalámbricas**: Las redes más interesantes serían las distribuidas, no centralizadas, que permitieran comunicaciones entre pares y dinámicas, esto es, redes *ad-hoc*. Las redes inicialmente más interesantes son WiFi y Bluetooth. WiFi permitiría, además, mediante puntos de accesos a lo largo de la vía, el acceso a Internet. Bluetooth tiene un buen manejo de las conexiones entre pares, pero precisa vincular los nodos. No obstante, por la literatura consultada, ambas podrían ser demasiado lentas para las redes vehiculares por el largo tiempo de establecimiento de la conexión. Existen otras redes que quizá tengan menor tiempo de gestión de descubrimiento e interconexión, como las redes basadas en radio frecuencia, pero que precisaría que se definiera un protocolo por encima de ellas para dotarlas de funcionalidad.
- **Ampliar el catálogo de funcionalidades básicas**: Otras funcionalidades próximas en el futuro son la conducción realmente autónoma bajo ciertos estrictos entornos, tales como el aparcamiento. También hay otras como la de seguimiento de rutas completas, a partir del conocimiento de mapas y de la geolocalización, generalmente mediante GPS o mediante indicadores especializados en las vías.

(Página intencionalmente en blanco)

Anexos

ANEXO A

SEGUIMIENTO DE TRAYECTO: CONEXIONADOS Y CÓDIGOS

Se presentan dos implementaciones diferentes, con conexionado y códigos diferentes para cada uno de los prototipos utilizados.

La primera se corresponde con el seguimiento de trayectoria implementado en el SunFounder™, y utiliza cinco sensores de infrarrojos de corto alcance, separados entre sí entre 13 el central de los dos que tiene al lado, y 17 mm entre éstos intermedios y los extremos.

La segunda es la implementación del seguimiento de trayectoria en el 4WD, y utiliza tres sensores de infrarrojos de corto alcance, separados 13 mm entre sí.

A.1. Prototipo SunFounder™

A.1.1. Conexionado

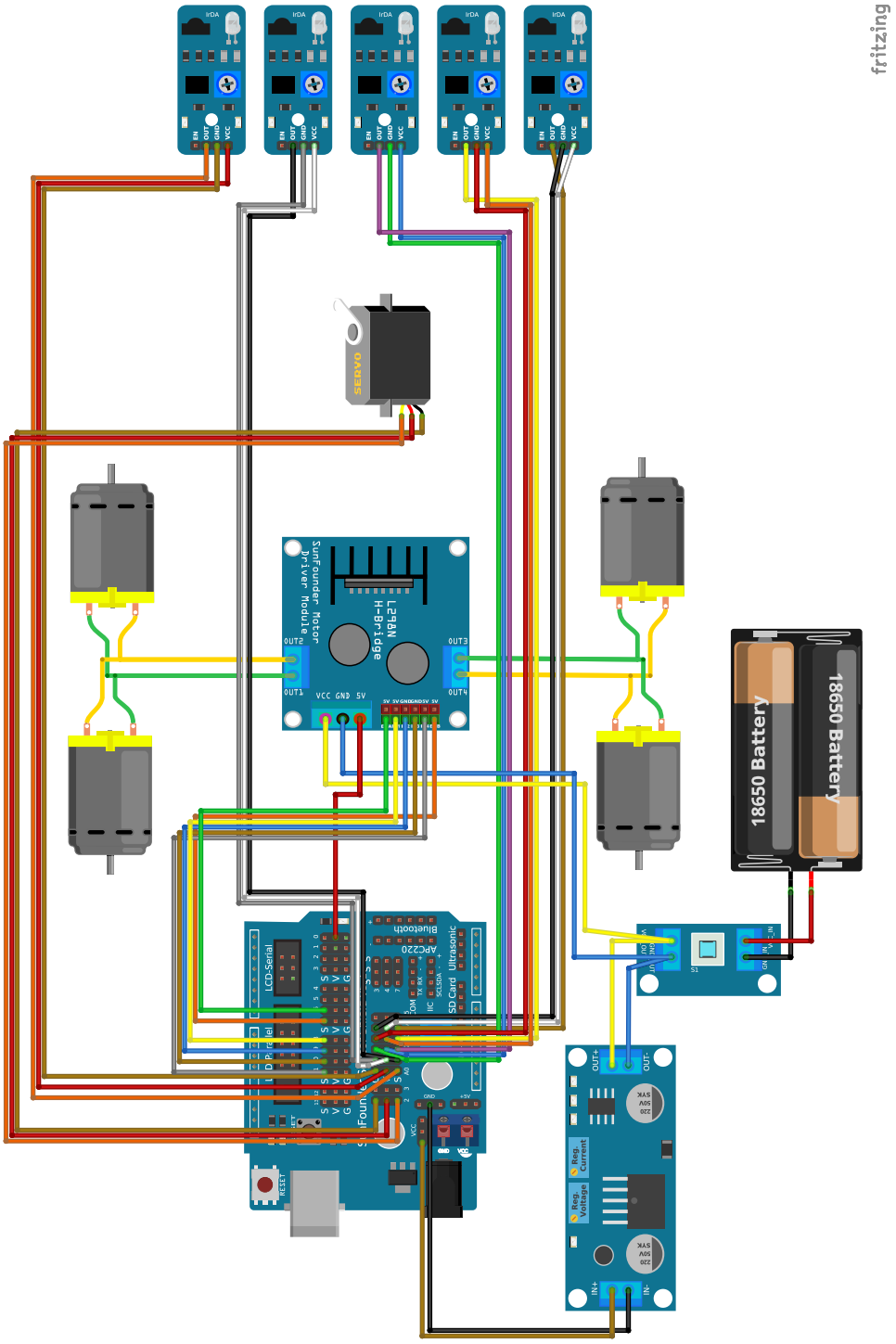

Figura A.1: Diagrama de conexionado del seguimiento de líneas del SunFounder™

A.1.2. Código en Arduino

Listado A.1: Seguimiento de trayecto para Arduino usando el SunFounder™ (v7b)

```
#include <Servo.h>

Servo myservo;

// Define Arduino pinout
// - for DC motors
#define ENA_PIN   ( 5) // ENA attached to pin5
#define ENB_PIN   ( 6) // ENB attached to pin6
#define MOTOR_L_1 ( 8) // MOTOR_L_1 attached to pin1
#define MOTOR_L_2 ( 9) // MOTOR_L_1 attached to pin9
#define MOTOR_R_1 (10) // MOTOR_R_1 attached to pin10
#define MOTOR_R_2 (11) // MOTOR_R_1 attached to pin11
#define SERVO_PIN ( 2) // Servo attached to pin2
// - for IR line tracking sensors
#define IR_LEFT_MAX_PIN   (A0) //attach the left  first  Tracking module pinA0 to A0
#define IR_LEFT_HALF_PIN  (A1) //attach the left  second Tracking module pinA0 to A1
#define IR_MIDDLE_PIN     (A2) //attach the central      Tracking module pinA0 to A2
#define IR_RIGHT_HALF_PIN (A3) //attach the right second Tracking module pinA0 to A3
#define IR_RIGHT_MAX_PIN  (A5) //attach the right first  Tracking module pinA0 to A5
// WARNING: A4 pins are not working on my original SunFounder Sensor Shield v.5.1

// Indexes for the vector of exponential mobil average
#define NUM_SENSORS   (5)
#define IR_LEFT_MAX   (0)
#define IR_LEFT_HALF  (1)
#define IR_MIDDLE     (2)
#define IR_RIGHT_HALF (3)
#define IR_RIGHT_MAX  (4)

#define FORWARD 0 // car moves forward
#define BACK    1 // car moves back

// Define physical axis' angle properties
#define ANGLE_SERVO_BIAS ( -7) // To alineate the front wheels if servo not ortogonal
#define ANGLE_LEFT_MAX   ( 60+ANGLE_SERVO_BIAS)
#define ANGLE_LEFT_HALF  ( 75+ANGLE_SERVO_BIAS)
#define ANGLE_MIDDLE     ( 90+ANGLE_SERVO_BIAS)
#define ANGLE_RIGHT_HALF (115+ANGLE_SERVO_BIAS)
#define ANGLE_RIGHT_MAX  (120+ANGLE_SERVO_BIAS)

// Define weighting parameters for updating sensor values on each iteration
#define W_LEFT_MAX   (-30.0f)
#define W_LEFT_HALF  (-15.0f)
#define W_MIDDLE     (  0   )
#define W_RIGHT_HALF (+15.0f)
#define W_RIGHT_MAX  (+30.0f)

// Algorythm main parameters
#define TOUT_MAX  (2000) // When tout exceeds it, car starts looking for black line
```

```
50  #define TOUT_INC  (   3) // tout+=TOUT_INC each (backwards) iteration
51  #define TOUT_STEP ( 100) // When steps exceeds it, car change its backward's direction
52  #define THRESHOLD (1000) // IR sensor analog value to say if black (1) or white (0)
53  #define SPEED     ( 160) // DC motor (rear wheels) maximum speed
54  #define LOWSPEED  ( 130) // DC motor (rear wheels) minimum speed
55  #define ALPHA     (0.1f) // 0..1: Higher values gives more importance to present sensor
56                           //       values. Used in history angle
57  #define BETA     (0.05f) // 0..1: Higher values gives more importance to present sensor
58                           //       values. Used in history vector
59
60  // Set DEBUGX to 0 when loading into the real car for a real test (or lower TOUT_MAXv)
61  #define DEBUG0 0      // Delays execution to allow easier debugging
62  #define DEBUG1 0      // Debugs basic state change
63  #define DEBUG2 0      // More detailed debug info
64  #define PRINT1(r)     if (DEBUG1) Serial.print(r)
65  #define PRINT2(r,s)   if (DEBUG2) Serial.print(r,s)
66  #define PRINTLN1(r)   if (DEBUG1) Serial.println(r)
67  #define PRINTLN2(r,s) if (DEBUG2) Serial.println(r,s)
68
69  // Algorythmic variables
70  signed   char mean  = 0; // Present weighted sensors value
71  signed   char line  = 0; // Number of sensors detecting line (n on algorythm descript.)
72  signed   int  tout  = 0; // Counts how many backward iterations with no black detection
73  signed   int  steps = 0; // Counts how many iterations after the last random dir change
74  unsigned char velo  = 0; // Preent DC motors speed
75  unsigned char angle = ANGLE_MIDDLE; // Present front axis' angle
76  float prevangle_f = 0.0f;// Previous angle value, in float precission
77  float mean_f      = 0.0f;// Present weighted sensors value, in float precission
78  float history[NUM_SENSORS]; // Given (weight calculated) values for each sensor
79
80  void setup() {
81    /* Software serial initializaion for debugging purposes */
82    Serial.begin (115200);
83    // Arduino Leonardo: wait for the serial port to connect
84    while (!Serial) {};
85    PRINT1("Seguimiento. v7b");
86
87    /* set below pins as OUTPUT */
88    pinMode(ENA_PIN  , OUTPUT);
89    pinMode(ENB_PIN  , OUTPUT);
90    pinMode(MOTOR_L_1, OUTPUT);
91    pinMode(MOTOR_L_2, OUTPUT);
92    pinMode(MOTOR_R_1, OUTPUT);
93    pinMode(MOTOR_R_2, OUTPUT);
94    myservo.attach(SERVO_PIN); // servo attached to SERVO_PIN
95    prevangle_f = ANGLE_MIDDLE;
96    angle = (unsigned char) prevangle_f;
97    myservo.write(angle);      // set the angle of servo
98    for (int i=0; i<NUM_SENSORS; i++) {
99      history[i] = 0.0f;
100   }
101 } // end of setup()
102
```

```
103  void loop() {
104    if (DEBUG0) delay (500);
105    PRINTLN1("");
106    PRINT1("INICIO: Angle=");
107    PRINT1(angle,DEC);
108    PRINT1(" : SENSORES=");
109
110    // statistical variables
111    mean_f = 0.0f;
112    line = 0; // checks if any sensor has detected a line (1) or none of them have (0)
113
114    // FIRST MEASURE LINES
115    //1) Checks each sensor for black line detection
116    //   (present analog value > threshold ==> black; if not, white)
117    //2) updates moving average exponential,
118    //3) and weight-averages their mean value
119    if(analogRead(IR_LEFT_MAX_PIN)  > THRESHOLD) {//if read the value of the left first
           Tracking module pinA0 is more than THRESHOLD
120        PRINT2(1,DEC);
121        line += 1;
122        history[IR_LEFT_MAX] = history[IR_LEFT_MAX]*(1.0f-BETA) + BETA;
123    } else {
124        PRINT2(0,DEC);
125        history[IR_LEFT_MAX] = history[IR_LEFT_MAX]*(1.0f-BETA);
126    }
127    mean_f += (W_LEFT_MAX * history[IR_LEFT_MAX]);
128    if(analogRead(IR_LEFT_HALF_PIN)  > THRESHOLD) {//if read the value of the left second
            Tracking module pinA0 is more than THRESHOLD
129        PRINT2(1,DEC);
130        line += 1;
131        history[IR_LEFT_HALF] = history[IR_LEFT_HALF]*(1.0f-BETA) + BETA;
132    } else {
133        PRINT2(0,DEC);
134        history[IR_LEFT_HALF] = history[IR_LEFT_HALF]*(1.0f-BETA);
135    }
136    mean_f += (W_LEFT_HALF * history[IR_LEFT_HALF]);
137    if(analogRead(IR_MIDDLE_PIN)  > THRESHOLD) {//if read the value of the module
           Tracking module pinA0 is more than THRESHOLD
138        PRINT2(1,DEC);
139        line += 1;
140        history[IR_MIDDLE] = history[IR_MIDDLE]*(1.0f-BETA) + BETA;
141    } else {
142        PRINT2(0,DEC);
143        history[IR_MIDDLE] = history[IR_MIDDLE]*(1.0f-BETA);
144    }
145    //mean_f += (W_MIDDLE * history[IR_MIDDLE]); // It's always ZERO
146    if(analogRead(IR_RIGHT_HALF_PIN) > THRESHOLD) {//if read the value of the right
           second Tracking module pinA0 is more than THRESHOLD
147        PRINT2(1,DEC);
148        line += 1;
149        history[IR_RIGHT_HALF] = history[IR_RIGHT_HALF]*(1.0f-BETA) + BETA;
150    } else {
151        PRINT2(0,DEC);
```

```
152         history[IR_RIGHT_HALF] = history[IR_RIGHT_HALF]*(1.0f-BETA);
153     }
154     mean_f += (W_RIGHT_HALF * history[IR_RIGHT_HALF]);
155     if(analogRead(IR_RIGHT_MAX_PIN) > THRESHOLD) {//if read the value of the right first
            Tracking module pinA0 is more than THRESHOLD
156         PRINT2(1,DEC);
157         line += 1;
158         history[IR_RIGHT_MAX] = history[IR_RIGHT_MAX]*(1.0f-BETA) + BETA;
159     } else {
160         PRINT2(0,DEC);
161         history[IR_RIGHT_MAX] = history[IR_RIGHT_MAX]*(1.0f-BETA);
162     }
163     mean_f += (W_RIGHT_MAX * history[IR_RIGHT_MAX]);
164     mean = (signed char) mean_f;
165 
166     // Show detailed debugging info about mean calculation
167     PRINT1(" : History=(");
168     for (int i=0; i<NUM_SENSORS; i++) {
169       PRINT2(history[i],DEC);
170       PRINT1("-");
171     }
172     PRINT1(" : mean_f=");
173     PRINT2(mean_f,DEC);
174     PRINT1(" : mean=");
175     PRINT2(mean,DEC);
176     PRINT1(" : line=");
177     PRINTLN2(line,DEC);
178 
179     // THEN TAKE ACTIONS
180     if (line == 0) {         // If none sensor has detected a black line... (going back)
181       if (tout > TOUT_MAX) { // If reached maximum iterations going back since line lost
182         steps++;
183         if (steps > TOUT_STEP) { // If reached maximum iterations to change randomly dir.
184           steps=0;               // reset counter
185           angle = random(ANGLE_LEFT_MAX,ANGLE_RIGHT_MAX); // choose a new random angle
186           prevangle_f = prevangle_f*(1.0f-ALPHA) + (float) angle*ALPHA; // float precissi
187           angle = (unsigned char) prevangle_f;
188           myservo.write(angle);          // set the angle of servo
189         }
190         velo = LOWSPEED;                 // moving backwards at low speed
191         CAR_move(BACK,LOWSPEED,LOWSPEED);// car moves BACK much slower
192                                          // trying to find again the lost line
193         PRINT1("RETROCEDIENDO : tout=");
194       } else {                  // First stage of moving back for a while since line lost
195         velo = LOWSPEED;        // Set lower speed to move back
196         CAR_move(FORWARD,LOWSPEED,LOWSPEED); // Move back at low speed
197         tout += TOUT_INC;       // Iterations counter
198         PRINT1("AVANZANDO A CIEGAS : tout=");
199       }
200     } else {                  // At least one sensor has detected a black line under it
201       angle = ANGLE_MIDDLE - mean/line; // new angle, depending on where the line is
202       prevangle_f = prevangle_f*(1.0f-ALPHA) + (float)angle*ALPHA; // update angle value
203       angle = (unsigned char) prevangle_f;
```

```
204       myservo.write(angle);          // set the angle of servo
205       tout = 0;                      // reset backwards iteration counting
206       velo = (unsigned char) (SPEED - (signed char)(SPEED-LOWSPEED)*abs((float)(angle-
              ANGLE_MIDDLE)/(float)(ANGLE_RIGHT_MAX-ANGLE_MIDDLE))); // Exponential moving
              average
207       CAR_move(FORWARD,velo,velo);   // car move forward at desired "velo" speed
208       PRINT1("AVANZANDO NORMAL : tout=");
209     }
210     PRINT2(tout,DEC);
211     PRINT1(" : angle=");
212     PRINT2(angle,DEC);
213     PRINT1(" : velo=");
214     PRINTLN2(velo,DEC);
215   } // end of loop()
216
217   void CAR_move(unsigned char directionTo, unsigned char speed_left, unsigned char
        speed_right) {
218     switch(directionTo) {
219       //car moves forward
220       case FORWARD:
221         //left  motor clockwise rotation --> forward
222         digitalWrite(MOTOR_L_1,HIGH);
223         digitalWrite(MOTOR_L_2,LOW );
224         //right motor clockwise rotation --> forward
225         digitalWrite(MOTOR_R_1,HIGH);
226         digitalWrite(MOTOR_R_2,LOW );
227         break;
228       //car moves back
229       case BACK:
230         //left  motor counterclockwise rotation --> back
231         digitalWrite(MOTOR_L_1,LOW);
232         digitalWrite(MOTOR_L_2,HIGH);
233         //right motor counterclockwise rotation --> back
234         digitalWrite(MOTOR_R_1,LOW);
235         digitalWrite(MOTOR_R_2,HIGH);
236         break;
237       default:
238         break;
239     }
240     // PWM: Sets the speed of both DC motors, writting speed_X to the ENA_PIN or ENB_PIN
241     analogWrite(ENA_PIN,speed_left );
242     analogWrite(ENB_PIN,speed_right);
243   } // end of CAR_move()
```

A.2. Prototipo 4WD con tres sensores por IR

A.2.1. Conexionado

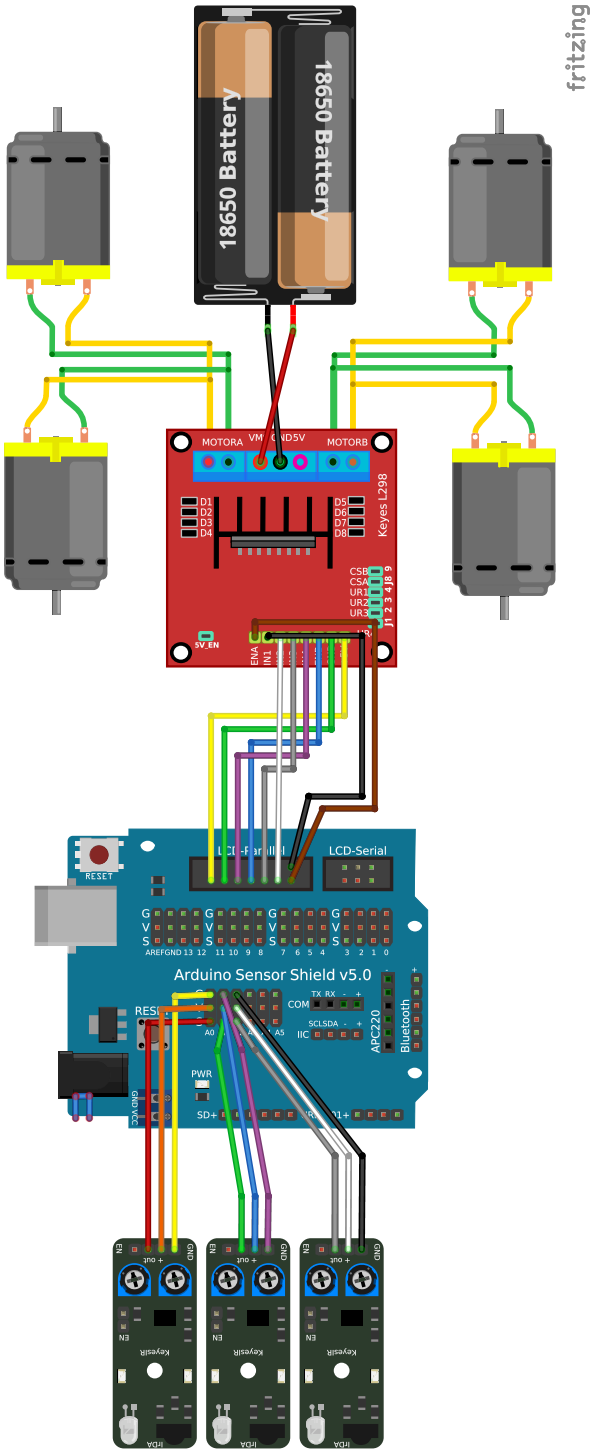

Figura A.2: Diagrama de conexionado del seguimiento de líneas del 4WD

A.2.2. Código en Arduino

Listado A.2: Seguimiento de trayecto para Arduino usando el 4WD con tres sensores IR (v4)

```
// Movement parameters
#define SPEED_HIGH     (200)
#define SPEED_MED      (160)
#define SPEED_LOW      (120)
#define SPEED_ZERO     (  0)
#define STEPS_BACK_MIN ( 40)
#define DELAY_BACK     (100)
#define DELAY_TURN     ( 80)

// For debugging purposes
#define DEBUG (0)
#define _SERIAL if (DEBUG) Serial

// Define Arduino pinout:
// - for DC motors
int MotorRight2 = 7; // Clockwise        pin of right side motors
int MotorRight1 = 5; // Counterclockwise pin of right side motors
int speedpinR   = 6; // Right side motor: ENable pin. Used for PWM -> speed
int MotorLeft2  = 4; // Clockwise        pin of left  side motors
int MotorLeft1  = 2; // Counterclockwise pin of left  side motors
int speedpinL   = 3; // Left  side motor: ENable pin. Used for PWM -> speed
// - for IR line tracking sensors
const int SensorLeft   = A0; // 7; // Left  sensor input pin
const int SensorMiddle = A1; // 4; // Front sensor input pin
const int SensorRight  = A2; // 3; // Right sensor input pin

// Present values of the IR sensors
int SL = LOW; // Left  sensor status
int SM = LOW; // The   sensor status
int SR = LOW; // Right sensor status

int previous;
int backSteps = 0;
boolean goingBack = false; // Controls the direction (forward/backward)

void setup() {
  Serial.begin (9600);
  pinMode (MotorRight1 , OUTPUT); // Signal 1 right side motor
  pinMode (MotorRight2 , OUTPUT); // Signal 2 right side motor
  pinMode (speedpinR   , OUTPUT); // PWM pin, to control speed right side motor
  pinMode (MotorLeft1  , OUTPUT); // Signal 1 left  side motor
  pinMode (MotorLeft2  , OUTPUT); // Signal 2 left  side motor
  pinMode (speedpinL   , OUTPUT); // PWM pin, to control speed left  side motor
  pinMode (SensorLeft  , INPUT ); // Left  infrared sensor
  pinMode (SensorMiddle, INPUT ); // Front infrared sensor
  pinMode (SensorRight , INPUT ); // Right infrared sensor
} // end of setup()

void loop() {
```

```
50  start:
51    previous = (SL << 2) + (SM << 1) + SR;
52    if (DEBUG) delay (500);
53    SL = digitalRead (SensorLeft  );
54    SM = digitalRead (SensorMiddle);
55    SR = digitalRead (SensorRight );
56    _SERIAL.print("Sensors (L/M/R) = ");
57    _SERIAL.print(SL, DEC);
58    _SERIAL.print(" / ");
59    _SERIAL.print(SM, DEC);
60    _SERIAL.print(" / ");
61    _SERIAL.print(SR, DEC);
62    _SERIAL.print(" = ");
63    _SERIAL.println(previous, DEC);
64    if (previous == (SL << 2) + (SM << 1) + SR) {
65      goto start; // Loop again if there is no change on the sensor's status
66    }
67
68    // Goes back if there is no line under the car, looking for the lost line
69    // Does not goes forward untill it detects STEPS_BACK_MIN times in a row a line under
70    if (goingBack) {
71      if (((SL << 2) + (SM << 1) + SR) != 0) {
72        backSteps++;
73        if (backSteps == STEPS_BACK_MIN) {
74          goingBack = false;
75          backSteps = 0;
76        }
77      } else {
78        backSteps = 0;
79      }
80    }
81
82    if (SM == HIGH) {                       // CENTRAL SENSOR IN BLACK AREA
83      digitalWrite(MotorLeft2 , HIGH);      // DC motor (right) clockwise: Forward
84      digitalWrite(MotorLeft1 , LOW );
85      digitalWrite(MotorRight1, LOW );      // DC motor (left) counterclockwise: Forward
86      digitalWrite(MotorRight2, HIGH);
87      if (SL == LOW && SR == HIGH) {        // left white and right black, turn right
88        _SERIAL.println("BLACK AREA: left white and right black, turn right");
89        analogWrite(speedpinR, SPEED_HIGH); // set the speed of the input analog value
90        analogWrite(speedpinL, SPEED_ZERO);
91        delay(DELAY_TURN);
92      } else if (SR == LOW && SL == HIGH) { // left black and right white, turn left
93        _SERIAL.println("BLACK AREA: left black and right white, turn left");
94        analogWrite(speedpinR, SPEED_ZERO); // set the speed of the input analog value
95        analogWrite(speedpinL, SPEED_HIGH);
96        delay(DELAY_TURN);
97      } else {                              // both sides white, go straight
98        _SERIAL.println("BLACK AREA: Both sides white, straight");
99        analogWrite(speedpinR, SPEED_HIGH); // set the speed of the input analog value
100       analogWrite(speedpinL, SPEED_HIGH);
101     }
102   } else {                                 // CENTRAL SENSOR IN THE WHITE AREA
```

```
103       if (SL == LOW && SR == HIGH) {         // left white and right black, turn right
104         _SERIAL.println("WHITE AREA: left white and right black, turn right");
105         digitalWrite(MotorLeft2 , LOW     ); // DC motor (right) counterclockwise: Back
106         digitalWrite(MotorLeft1 , HIGH    );
107         digitalWrite(MotorRight1, LOW     ); // DC motor (left) counterclockwise: Forward
108         digitalWrite(MotorRight2, HIGH    );
109         analogWrite (speedpinR, SPEED_HIGH); // set the speed of the input analog value
110         analogWrite (speedpinL, SPEED_MED );
111         delay(DELAY_TURN);
112       } else if (SR == LOW && SL == HIGH) {  // left black and right white, turn left
113         _SERIAL.println("WHITE AREA: left black and right white, turn left");
114         digitalWrite(MotorLeft2 , HIGH    ); // DC motor (right) clockwise: Forward
115         digitalWrite(MotorLeft1 , LOW     );
116         digitalWrite(MotorRight1, HIGH    ); // DC motor (left) clockwise: Back
117         digitalWrite(MotorRight2, LOW     );
118         analogWrite (speedpinR, SPEED_MED ); // set the speed of the input analog value
119         analogWrite (speedpinL, SPEED_HIGH);
120         delay(DELAY_TURN);
121       } else if (SR == HIGH && SL == HIGH) { // both sides black, go ahead slowly
122         _SERIAL.println("WHITE AREA: left black and right black, go ahead slowly");
123         digitalWrite(MotorLeft2 , HIGH    ); // DC motor (right) clockwise: Forward
124         digitalWrite(MotorLeft1 , LOW     );
125         digitalWrite(MotorRight1, LOW     ); // DC motor (left) counterclockwise: Forward
126         digitalWrite(MotorRight2, HIGH    );
127         analogWrite (speedpinR, SPEED_LOW ); // set the speed of the input analog value
128         analogWrite (speedpinL, SPEED_LOW );
129       } else {                     // all white, go back trying to recover the black line
130         _SERIAL.println("WHITE AREA: all three are white, go back trying to recover "
131                         "the black line");
132         digitalWrite (MotorLeft2 , LOW    ); // DC motor (right) counterclockwise: Back
133         digitalWrite (MotorLeft1 , HIGH   );
134         digitalWrite (MotorRight1, HIGH   ); // DC motor (left) clockwise: Back
135         digitalWrite (MotorRight2, LOW    );
136         analogWrite  (speedpinR, SPEED_LOW); // set the speed of the input analog value
137         analogWrite  (speedpinL, SPEED_LOW);
138         goingBack = true;
139         delay(DELAY_BACK);
140       }
141     }
142   } // end of loop()
```

(Página intencionalmente en blanco)

ANEXO B

SEGUIMIENTO DE LUZ: CONEXIONADO Y CÓDIGO

El seguimiento de luz se ha implementado únicamente en el SunFounder™.

Se realizaron dos implementaciones. La del montaje original, que disponía tres sensores delanteros y uno trasero, y la que aquí se presenta, que es la que dispone dos sensores de luz delante y dos sensores de luz detrás. De esta forma, el vehículo puede guiarse lateralmente tanto hacia delante como marcha atrás.

A continuación se muestra el conexionado y su código correspondiente.

B.1. Seguimiento de luz en el SunFounder™

B.1.1. Conexionado

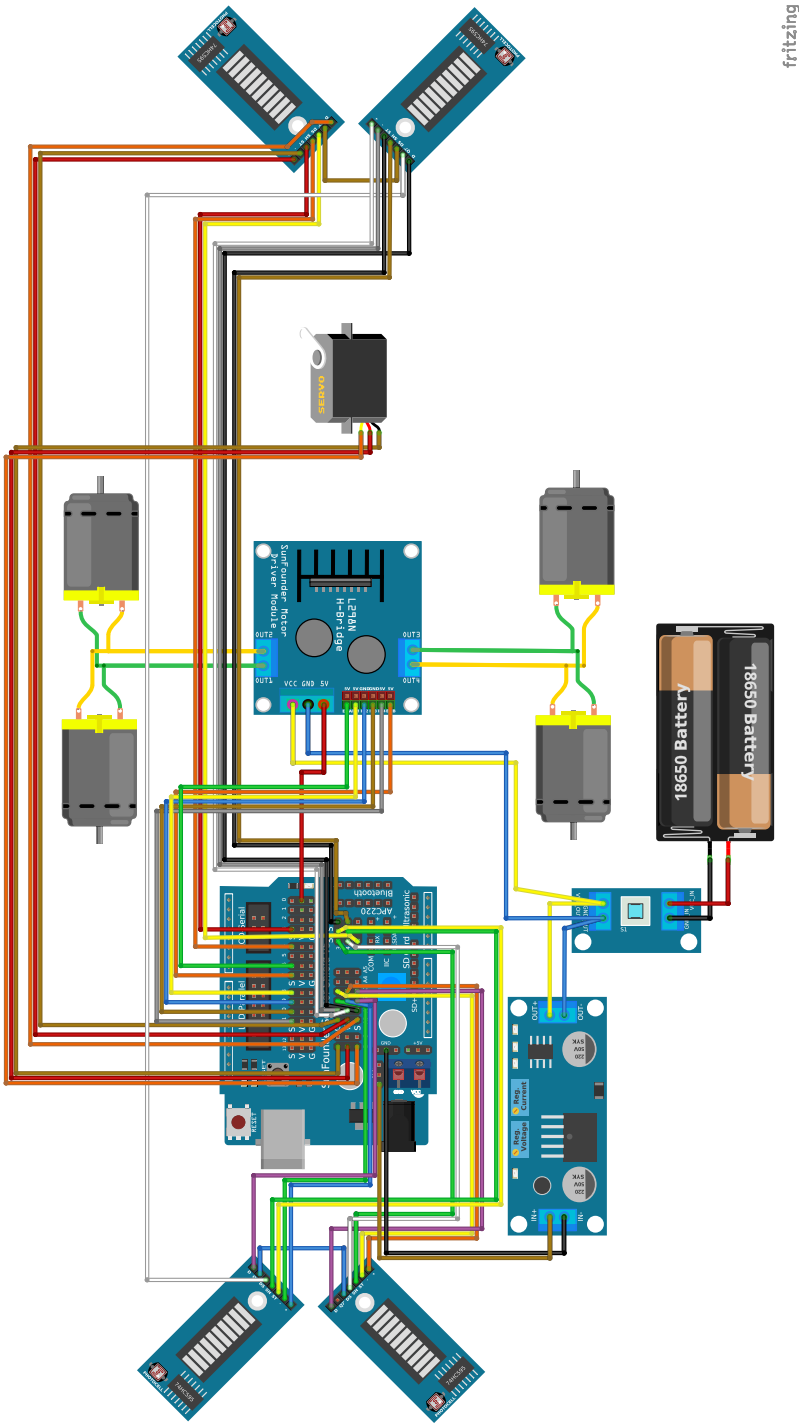

Figura B.1: Diagrama de conexionado del seguimiento de luz del SunFounder™

B.1.2. Código en Arduino

HC595 es el chip contador, basado en un registro de desplazamiento, que permite gestionar las diferentes salidas de cada una de las barras de luces de cada sensor.

Listado B.1: Seguimiento de luz para Arduino usando el SunFounder™ (v5)

```
#include <Servo.h>    //Servo library
#include "hc595.h"    //74HC595 shift register library: manages LED bars of each sensor
#include <MsTimer2.h> //Library to manage interruptions driven by own Arduino timers

unsigned char code_buf[] = {0, 0, 0, 0}; // To represent the intensity of light
unsigned char code_val[] = {0x01, 0x03, 0x07, 0x0f, 0x1f, 0x3f, 0x7f, 0xff};

Servo myservo;

// Define Arduino pinout:
// - for DC motors
#define ENA_PIN    5 // ENA attached to pin5 --> PWM to set wheel's spped
#define ENB_PIN    6 // ENB attached to pin6 --> PWM to set wheel's spped
#define MOTOR_L_1  8 // MOTOR_L_1 attached to pin8
#define MOTOR_L_2  9 // MOTOR_L_1 attached to pin9
#define MOTOR_R_1 10 // MOTOR_R_1 attached to pin10
#define MOTOR_R_2 11 // MOTOR_R_1 attached to pin11
// - for lightseeking sensors
#define PHOTOCELL_1_PIN A0 //attach the first  photocell to pinA0
#define PHOTOCELL_2_PIN A1 //attach the second photocell to pinA1
#define PHOTOCELL_3_PIN A2 //attach the third  photocell to pinA2
#define PHOTOCELL_4_PIN A3 //attach the four   photocell to pinA3

#define FORWARD 0 //define forward=0, car moves forward
#define BACK    1 //define back=1,    car moves back

#define SERVO_BIAS       (-5)             //SW hack: Allows align your servo exactly
#define ANGLE_RIGHT_MAX  (115+SERVO_BIAS) //Maximum angle rotation to right: 115
#define ANGLE_LEFT_MAX   ( 65+SERVO_BIAS) //Minimum angle rotation to left:   65
#define ANGLE_MIDDLE     ( 90+SERVO_BIAS) //Middel, central point: frontal direction

#define SPEED 200           // Speed of the motor [0..255] (wheels)
#define SENSIBILITY (0.10f) // Minimum difference to say whether which sensors
                            // (front or back located) are reciving more light

#define DEBUG (0)               // For debugging purposes
#define SERIAL if (DEBUG) Serial // Show debug info via the serial connection

int value_FL_Dir  = 0;
int value_FL_Init = 0;
int value_FL_Temp = 0;

int value_FR_Dir  = 0;
int value_FR_Init = 0;
int value_FR_Temp = 0;

int value_BL_Dir  = 0;
```

```
48  int value_BL_Init = 0;
49  int value_BL_Temp = 0;
50
51  int value_BR_Dir  = 0;
52  int value_BR_Init = 0;
53  int value_BR_Temp = 0;
54
55  int value_light_threshold = 0;
56
57  signed   char state = 0;
58  unsigned char speed = 0;
59  unsigned char directionTo = ANGLE_MIDDLE;
60  unsigned char dir = FORWARD;
61  unsigned char directionTo_lastest = 0;
62  unsigned char angle = ANGLE_MIDDLE;
63
64  void setup() {
65    HC595_Init();//initialize 74hc595,set DS,SH_CP,ST_CP as output
66    //set below pins as output
67    pinMode(ENA_PIN  , OUTPUT);
68    pinMode(ENB_PIN  , OUTPUT);
69    pinMode(MOTOR_L_1, OUTPUT);
70    pinMode(MOTOR_L_2, OUTPUT);
71    pinMode(MOTOR_R_1, OUTPUT);
72    pinMode(MOTOR_R_2, OUTPUT);
73
74    delay(500);
75    //Calibration: read the value of each photocell 512 times, then averaging
76    value_FL_Init = average(PHOTOCELL_1_PIN, 512);
77    value_FR_Init = average(PHOTOCELL_2_PIN, 512);
78    value_BL_Init = average(PHOTOCELL_3_PIN, 512);
79    value_BR_Init = average(PHOTOCELL_4_PIN, 512);
80    delay(500);
81    //Calibration: read the value of photocell another 512 times,then averaging
82    value_FL_Init = (value_FL_Init + average(PHOTOCELL_1_PIN, 512))/2;
83    value_FR_Init = (value_FR_Init + average(PHOTOCELL_2_PIN, 512))/2;
84    value_BL_Init = (value_BL_Init + average(PHOTOCELL_3_PIN, 512))/2;
85    value_BR_Init = (value_BR_Init + average(PHOTOCELL_4_PIN, 512))/2;
86
87    SERIAL.begin(115200);
88    SERIAL.println("Inicializacion");
89    SERIAL.print(value_FL_Init); SERIAL.print(",");
90    SERIAL.print(value_FR_Init); SERIAL.print(",");
91    SERIAL.print(value_BL_Init); SERIAL.print(",");
92    SERIAL.println(value_BR_Init);
93
94    myservo.attach(2);              //attach servo to pin2
95    myservo.write(ANGLE_MIDDLE);    //the initial angle of servo is 90
96    MsTimer2::set(10, SERVO_Write); //10 periods, then run SERVO_Write()
97    MsTimer2::start();              //start timing
98  } // end of setup()
99
100 void loop() {
```

```
101     if (DEBUG) delay(5000);
102     value_light_threshold = average(A4,10); //read value of PinA4 10 times, then averages
103
104     SERIAL.print ("loop(): value_light_threshold = ");
105     SERIAL.println(value_light_threshold);
106
107     //read the value of each photocell 50 times, then average
108     value_FL_Dir = average(PHOTOCELL_1_PIN, 50);
109     value_FR_Dir = average(PHOTOCELL_2_PIN, 50);
110     value_BL_Dir = average(PHOTOCELL_3_PIN, 50);
111     value_BR_Dir = average(PHOTOCELL_4_PIN, 50);
112
113     // Debug use
114     SERIAL.println("loop(): average 50 values [x,y,z,u]");
115     SERIAL.print(value_FL_Dir); SERIAL.print(",");
116     SERIAL.print(value_FR_Dir); SERIAL.print(",");
117     SERIAL.print(value_BL_Dir); SERIAL.print(",");
118     SERIAL.println(value_BR_Dir);
119
120     // Determines where is the main luminance using the value of the four sensors
121     state = Light_Max(value_FL_Dir, value_FR_Dir, value_BL_Dir, value_BR_Dir);
122
123     SERIAL.print("loop(): state = ");
124     SERIAL.println(state);
125
126     // Depending on state value, sets the action to follow
127     switch((state>0) ? 1 : ((state<0) ? -1 : 0)) {
128       case 0: // car stop
129         speed = 0;
130         directionTo = directionTo;
131         dir = FORWARD;
132         break;
133       case +1: // car moves forward with angle greater if state greater
134         speed = SPEED;
135         directionTo = map( state, +1, +127, ANGLE_LEFT_MAX, ANGLE_RIGHT_MAX);
136         dir = FORWARD;
137         break;
138       case -1: // car moves forward with angle greater if state greater
139         speed = SPEED;
140         directionTo = map(-state, +1, +127, ANGLE_LEFT_MAX, ANGLE_RIGHT_MAX);
141         dir = BACK;
142         break;
143       default: // theoretically this cannot be never reached
144         speed = 0;
145         directionTo = ANGLE_MIDDLE;
146         dir = FORWARD;
147     } // end of switch
148
149     SERIAL.print("directionTo = ");
150     SERIAL.print(directionTo, DEC);
151     SERIAL.print(" : dir = ");
152     SERIAL.println(dir, DEC);
153
```

```
154     //map value_XX to led: 0 to 0 and 1023 to 8: will show detected light via LED bars
155     value_FL_Temp = map(value_FL_Dir, 0, 1023, 0, 8);
156     value_FR_Temp = map(value_FR_Dir, 0, 1023, 0, 8);
157     value_BL_Temp = map(value_BL_Dir, 0, 1023, 0, 8);
158     value_BR_Temp = map(value_BR_Dir, 0, 1023, 0, 8);
159 
160     // Codification of the lights to be shown on each LED bar
161     code_buf[0] = code_val[value_FL_Temp];
162     code_buf[1] = code_val[value_FR_Temp];
163     code_buf[2] = code_val[value_BL_Temp];
164     code_buf[3] = code_val[value_BR_Temp];
165 
166     SERIAL.print("value_X_Temp = ");
167     SERIAL.print(value_FL_Temp, DEC); SERIAL.print(",");
168     SERIAL.print(value_FR_Temp, DEC); SERIAL.print(",");
169     SERIAL.print(value_BL_Temp, DEC); SERIAL.print(",");
170     SERIAL.println(value_BR_Temp, DEC);
171 
172     SERIAL.print("LEDBAR_V_Dis() buf[i] = ");
173     SERIAL.print(code_buf[0], BIN); SERIAL.print(",");
174     SERIAL.print(code_buf[1], BIN); SERIAL.print(",");
175     SERIAL.print(code_buf[2], BIN); SERIAL.print(",");
176     SERIAL.println(code_buf[3], BIN);
177 
178     LEDBAR_V_Dis(code_buf);//display ledbar
179     CAR_move(dir,speed,speed);
180   } // end of loop()
181 
182   // Returns a value from -127 to +128, depending on the detected light values:
183   //  0 : If all 4 values are enough similar, so car must no move
184   // <0 : Moves backward, because sensors detected more light back than forward
185   //      Lower (greater in absulute value) means light comes more the from right side
186   // >0 : Moves forward,  because sensors detected more light forward than back
187   //      Greater value means light comes more the from right side
188   char Light_Max(int x, int y, int z, int u) {
189     // Normalized values = value_X_Dir / value_X_Init;
190     float value_FL_Norm = float(x) / value_FL_Init;
191     float value_FR_Norm = float(y) / value_FR_Init;
192     float value_BL_Norm = float(z) / value_BL_Init;
193     float value_BR_Norm = float(u) / value_BR_Init;
194 
195     SERIAL.println("Light_Max(): absolute values [x,y,z,u]");
196     SERIAL.print(x); SERIAL.print(",");
197     SERIAL.print(y); SERIAL.print(",");
198     SERIAL.print(z); SERIAL.print(",");
199     SERIAL.println(u);
200     SERIAL.println("Light_Max(): normalized values [x,y,z,u]");
201     SERIAL.print(value_FL_Norm, 4); SERIAL.print(",");
202     SERIAL.print(value_FR_Norm, 4); SERIAL.print(",");
203     SERIAL.print(value_BL_Norm, 4); SERIAL.print(",");
204     SERIAL.println(value_BR_Norm, 4);
205 
206     // Not necessary to divide always by 2.0f --> faster and equal final result
```

```
207     float mean_F_Norm = (value_FL_Norm + value_FR_Norm);
208     float mean_B_Norm = (value_BL_Norm + value_BR_Norm);
209
210     SERIAL.print("Mean_Forward = "); SERIAL.print(mean_F_Norm, 4);
211     SERIAL.print(" : Mean_Back = "); SERIAL.print(mean_B_Norm, 4);
212     SERIAL.print(" : Difference(F-B) = "); SERIAL.println(mean_F_Norm - mean_B_Norm, 4);
213
214     if (abs(mean_B_Norm - mean_F_Norm) < SENSIBILITY) return 0; //4 sensors similar light
215     if (mean_B_Norm < mean_F_Norm) { // FORWARD: more light detected in front
216       // returns a value [+1,+127] : greater --> turn more to the RIGHT, moving FORWARD
217       return min(max(+64 + (value_FR_Norm - value_FL_Norm)*200, +1), +127);
218     } else { // BACKWARD: more light detected in back side
219       // returns a value [-127,-1] : lower   --> turn more to the RIGHT, moving BACKWARD
220       return max(min(-64 - (value_BR_Norm - value_BL_Norm)*200, -1), -127);
221     }
222   } // end of Light_Max()
223
224   void CAR_move(unsigned char dirTo, unsigned char speed_left, unsigned char speed_right)
225   {
226     switch(dirTo) {
227       //car moves forward
228       case 0:
229         //left  motor clockwise rotation --> forward
230         digitalWrite(MOTOR_L_1,HIGH);
231         digitalWrite(MOTOR_L_2,LOW );
232         //right motor clockwise rotation --> forward
233         digitalWrite(MOTOR_R_1,HIGH);
234         digitalWrite(MOTOR_R_2,LOW );
235         break;
236       //car moves backward
237       case 1:
238         //left  motor counterclockwise rotation --> back
239         digitalWrite(MOTOR_L_1,LOW );
240         digitalWrite(MOTOR_L_2,HIGH);
241         //right motor counterclockwise rotation
242         digitalWrite(MOTOR_R_1,LOW );
243         digitalWrite(MOTOR_R_2,HIGH);
244         break;
245       default:
246         break;
247     } // end of switch
248     // PWM: Sets the speed of both DC motors, writting speed_X to the ENA_PIN or ENB_PIN
249     analogWrite(ENA_PIN,speed_left );
250     analogWrite(ENB_PIN,speed_right);
251   } // end of CAR_move()
252
253   int average(unsigned char pin, int num)//average the value {
254     unsigned long value = 0;
255     for(int i = 0; i < num - 1; i++)
256       value = value + analogRead(pin);
257     return(value/num);
258   } // end of average()
259
```

```
260  void SERVO_Write(void) {
261    if(angle < directionTo)
262      angle++;
263    else if(angle > directionTo)
264      angle--;
265    myservo.write(angle);
266    directionTo_lastest = directionTo;
267  } // end of SERVO_Write
```

ANEXO C

EVITACIÓN DE OBSTÁCULOS: CONEXIONADO Y CÓDIGO

Se presentan dos implementaciones diferentes, con conexionado y códigos diferentes para cada uno de los prototipos utilizados.

En el SunFounder™ se implementa el algoritmo de evitación de obstáculos utilizando sensores todo/-nada de dos tipos: dos sensores fijos de detección de colisión (corta distancia), y un sensor digital, de larga distancia, montado sobre una plataforma móvil accionada por su correspondiente servomotor.

El algoritmo de evitación de obstáculos utilizando sensores analógicos se implementa en el 4WD. Se utiliza un único sensor analógico de distancia por ultrasonidos, que se monta también sobre una plataforma rotatoria movida por un servomotor.

C.1. Prototipo SunFounder™

C.1.1. Conexionado

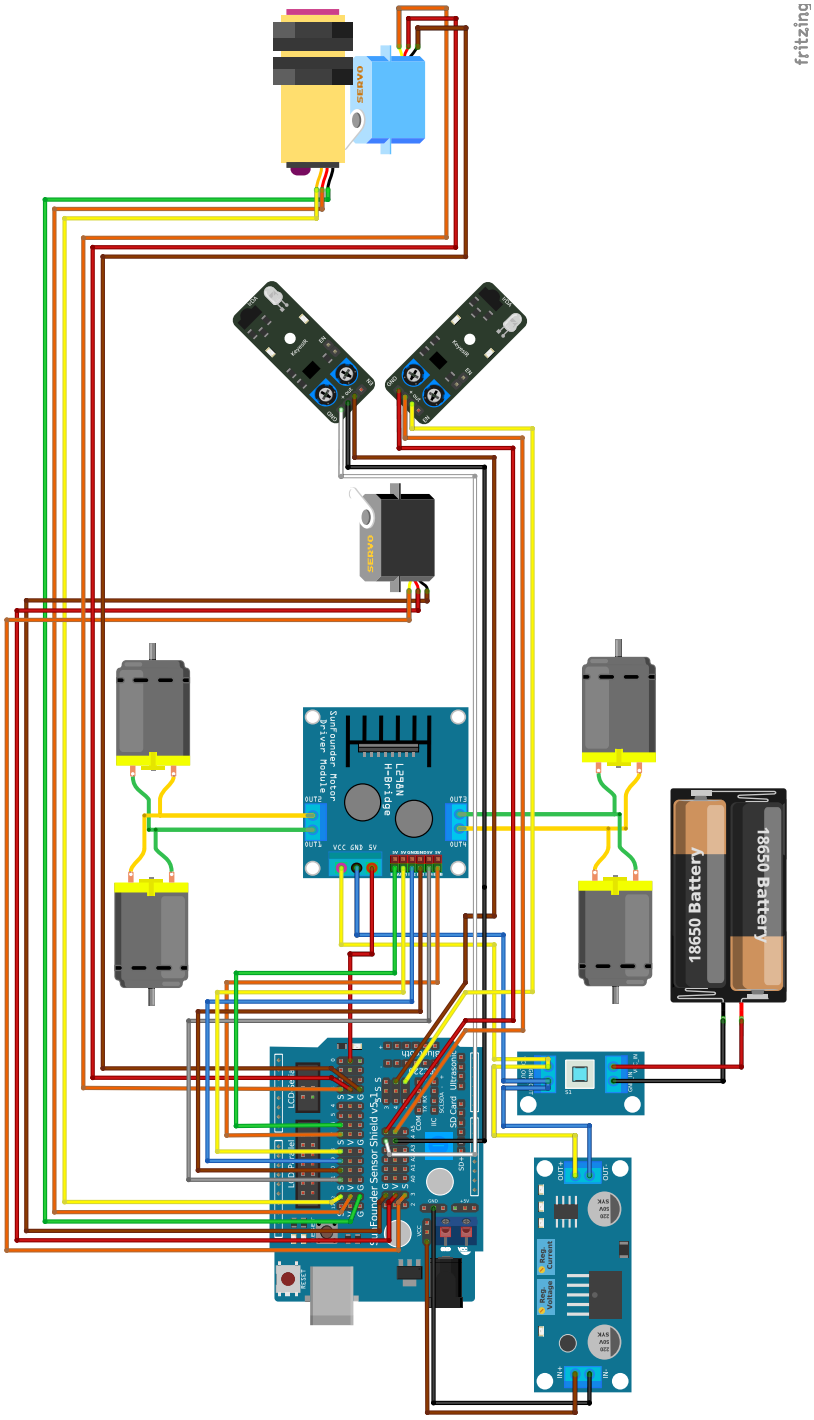

Figura C.1: Diagrama de conexionado de la evitación de obstáculos del SunFounder™ con sensores todo/nada (código v2)

C.1.2. Código en Arduino

Listado C.1: Evitación de obstáculos para Arduino usando el SunFounder™ (v2)

```
#include <Servo.h>

Servo servoW; // Wheels (before servoW)
Servo servoR; // Radar  (before servoR)

#define ENA_PIN    5 // ENA attached to pin5
#define ENB_PIN    6 // ENB attached to pin6
#define MOTOR_L_1  8 // MOTOR_L_1 attached to pin8
#define MOTOR_L_2  9 // MOTOR_L_1 attached to pin9
#define MOTOR_R_1 10 // MOTOR_R_1 attached to pin10
#define MOTOR_R_2 11 // MOTOR_R_1 attached to pin11

#define SERVO_TURN_PIN     2 // servoW attached to pin2
#define SERVO_MEASURE_PIN  3 // servoR attached to pin3
#define SENSOR_IR_L        4 // left  obstacle module attached to pin4
#define SENSOR_IR_R        7 // right obstacle module attached to pin7
#define SENSOR_IR_M       12 // radar module attached to pin12

#define FORWARD 0 //define forward=0,car move forward
#define BACK    1 //define back=1   ,car move back

// LOOKOUT, the direction of the angle is different in both servos!!!
// set the rotate angle of servoR
// Wheels' servo (MG995) has LEFT=0 and RIGHT=180 : 0 ---> 180
#define ANGLE_SERVOR_LEFT_MAX   (180)
#define ANGLE_SERVOR_LEFT_HALF  (120)
#define ANGLE_SERVOR_MIDDLE     ( 90)
#define ANGLE_SERVOR_RIGHT_HALF ( 60)
#define ANGLE_SERVOR_RIGHT_MAX  (  0)
// set the rotate angle of servoW
// Radar's servo (GS90)  has LEFT=180 and RIGHT=0 : 180 <--- 0
#define SERVOW_BIAS             ( -3)
#define ANGLE_SERVOW_RIGHT_MAX  (115 + SERVOW_BIAS)
#define ANGLE_SERVOW_MIDDLE     ( 90 + SERVOW_BIAS)
#define ANGLE_SERVOW_LEFT_MAX   ( 65 + SERVOW_BIAS)

#define N_DIS (200) // 0 to 180 --> 181 elements!

#define SPEED_MAX (230)
#define SPEED_MED (190)
#define SPEED_LOW (170)

#define BACK_TIME      (200)
#define FORWARD_TIME   (  2)
#define RADAR_DELAY    (  5)
#define MINIMUM_WINDOW ( 40)
#define OPTIMUM_WINDOW ( 60)
#define TURNS_FORWARD  (600)

```

```
50  #define DETECTION  (0)
51  #define FREEWAY    (1)
52  #define TO_RIGHT  (-1)
53  #define TO_LEFT   (-TO_RIGHT)
54
55  #define DEBUG (0)
56  #define _SERIAL if (DEBUG) Serial
57  #define DEBUG_DELAY (0)
58
59  unsigned int  arrayDetections[N_DIS] = {0};
60
61  unsigned char speedl    = SPEED_MAX;
62  unsigned char speedr    = SPEED_MAX;
63  unsigned char irValue   = 0;
64  unsigned char count_l   = 0;
65  unsigned char count_r   = 0;
66  unsigned char count_0_l = 0;
67  unsigned char count_1_l = 0;
68  unsigned char count_0_r = 0;
69  unsigned char count_1_r = 0;
70  unsigned char irValueL  = 1;
71  unsigned char irValueR  = 1;
72
73  enum state_t {
74    AVANCE_NORMAL,
75    AVANCE_PRECAVIDO,
76    ELECCION_RUTA,
77    RETROCESO,
78    RETROCESO_ESCANEANDO
79  };
80  state_t myState = AVANCE_NORMAL;
81  unsigned char angleW       = ANGLE_SERVOW_MIDDLE; // Wheels
82  unsigned char angleR       = ANGLE_SERVOR_MIDDLE; // Radar
83  unsigned char directionTo  = FORWARD;
84  unsigned char speed_left   = SPEED_MAX;
85  unsigned char speed_right  = SPEED_MAX;
86  bool          speedChanged = true;
87  bool          stateChanged = true;
88  bool          angleChangedWhenBack = false;
89  int           iRadar = 0;
90  int           iRdir  = TO_LEFT;
91  int           turnsForward = 0;
92
93  void setup() {
94    _SERIAL.begin(115200);
95    //set below pins as output
96    pinMode(ENA_PIN  , OUTPUT);
97    pinMode(ENB_PIN  , OUTPUT);
98    pinMode(MOTOR_L_1, OUTPUT);
99    pinMode(MOTOR_L_2, OUTPUT);
100   pinMode(MOTOR_R_1, OUTPUT);
101   pinMode(MOTOR_R_2, OUTPUT);
102   digitalWrite (SENSOR_IR_L,HIGH );// set left  obstacle module high
```

```
    pinMode      (SENSOR_IR_L,INPUT);// set left  obstacle module output
    digitalWrite (SENSOR_IR_R,HIGH );// set right obstacle module high
    pinMode      (SENSOR_IR_R,INPUT);// set right obstacle module output
    digitalWrite (SENSOR_IR_M,HIGH );// set radar module high
    pinMode      (SENSOR_IR_M,INPUT);// set radar module output
    servoW.attach(SERVO_TURN_PIN);   // servoW attached to pin2
    servoR.attach(SERVO_MEASURE_PIN);// servoR attached to pin3
    servoW.write(angleW);
    _SERIAL.println("\r\nSETUP DONE!");
  } // end of setup()

  void loop() {
  startLoop:
    if (DEBUG) delay (DEBUG_DELAY);
    _SERIAL.println("Starting loop().runCar()");
    _SERIAL.print("  speedChanged = ");
    _SERIAL.println(speedChanged, DEC);
    if (speedChanged) {
      moveCar();
      speedChanged = false;
    }
    if (checkSensors() == DETECTION) {
      _SERIAL.println("  checkSensors() == DETECTION");
      if (myState == RETROCESO_ESCANEANDO && angleChangedWhenBack == false) {
        _SERIAL.println("  Changed to opposite rhumb while going back");
        angleChangedWhenBack = true;  // Only one change allowed
        angleW = 180 - angleW;
        servoW.write(angleW);
      } else {
        changeStateToGoBack();
        goto startLoop;
      }
    } // end of if (checkSensors() == DETECTION)
    if (stateChanged) {
      _SERIAL.print("  State has changed... initializing new state ");
      stateChanged = false;
      switch (myState) {
        case AVANCE_NORMAL:
          _SERIAL.println("AVANCE_NORMAL");
          iRdir  = TO_LEFT;
          iRadar = ANGLE_SERVOR_MIDDLE;
          speed_left  = SPEED_MAX;
          speed_right = SPEED_MAX;
          directionTo = FORWARD;
          speedChanged = true;
          servoR.write(ANGLE_SERVOR_MIDDLE);
          CAR_move(FORWARD,SPEED_LOW,SPEED_LOW);
          delay(200); // advance a bit to give time radar to position
          moveCar();
          turnsForward = 0;
          break;
        case AVANCE_PRECAVIDO:
          _SERIAL.println("AVANCE_PRECAVIDO");
```

```
156           servoW.write(ANGLE_SERVOW_MIDDLE);
157           iRdir  = TO_LEFT;
158           iRadar = ANGLE_SERVOR_RIGHT_MAX;
159           speed_left  = SPEED_LOW;
160           speed_right = SPEED_LOW;
161           directionTo = FORWARD;
162           speedChanged = true;
163           moveCar();
164           break;
165         case ELECCION_RUTA:
166           _SERIAL.println("ELECCION_RUTA");
167           break;
168         case RETROCESO: // it should never enter here
169           _SERIAL.println("RETROCESO");
170           servoW.write(ANGLE_SERVOW_MIDDLE);
171           iRadar = ANGLE_SERVOR_RIGHT_MAX;
172           speed_left  = SPEED_MED;
173           speed_right = SPEED_MED;
174           directionTo = BACK;
175           speedChanged = true;
176           moveCar();
177           break;
178         case RETROCESO_ESCANEANDO:
179           _SERIAL.println("RETROCESO_ESCANEANDO");
180           iRdir  = TO_LEFT;
181           iRadar = ANGLE_SERVOR_RIGHT_MAX;
182           speed_left  = SPEED_LOW;
183           speed_right = SPEED_LOW;
184           directionTo = BACK;
185           speedChanged = true;
186           moveCar();
187           break;
188       }
189     } // end of if (stateChanged)
190     switch (myState) {
191       case AVANCE_NORMAL:
192         _SERIAL.println("  state: AVANCE_NORMAL");
193         _SERIAL.print("    iRadar = ");
194         _SERIAL.println(iRadar, DEC);
195         servoR.write(iRadar);
196         delay(FORWARD_TIME);
197         if (digitalRead(SENSOR_IR_M) == DETECTION) {
198           _SERIAL.println("    Radar DETECTION! changing state to cautious");
199           changeStateToGoCautiously();
200         }
201         if (iRadar == ANGLE_SERVOR_LEFT_HALF || iRadar == ANGLE_SERVOR_RIGHT_HALF){
202           _SERIAL.println("   Arrived to maximum allowed angle. Rebounding...");
203           iRdir = -iRdir;
204         }
205         iRadar += iRdir;
206         if (turnsForward++ == TURNS_FORWARD) {
207           turnsForward = 0;
208           angleW = ANGLE_SERVOW_MIDDLE;
```

```
209           servoW.write(angleW);
210         }
211         break;
212       case AVANCE_PRECAVIDO:
213         _SERIAL.println("  state: AVANCE_PRECAVIDO");
214         _SERIAL.print("    iRadar = ");
215         _SERIAL.println(iRadar, DEC);
216         servoR.write(iRadar);
217         delay(RADAR_DELAY);
218         arrayDetections[iRadar] = digitalRead(SENSOR_IR_M);
219         _SERIAL.print("    radarDetection = ");
220         _SERIAL.println(arrayDetections[iRadar], DEC);
221         if (iRadar == ANGLE_SERVOR_LEFT_MAX) {
222           _SERIAL.println("    Arrived to maximum allowed angle. "
223                     "Going to choose rhumb...");
224           changeStateToChooseRhumb();
225         }
226         iRadar += iRdir;
227         if (iRdir == ANGLE_SERVOR_MIDDLE) {
228           CAR_move(FORWARD,0,0); // The car runs too fast even in slower speed.
229         }
230         break;
231       case ELECCION_RUTA:
232         _SERIAL.println("  state: ELECCION_RUTA");
233         CAR_move(FORWARD,0,0);
234         {
235           int  _run        = 0;
236           int  run_max     = 0;
237           int  ini_run     = 0;
238           int  ini_run_max = 0;
239           for (int i=ANGLE_SERVOR_RIGHT_MAX; i<=ANGLE_SERVOR_LEFT_MAX; i++) {
240             if (arrayDetections[i] == DETECTION) {
241               if (_run>0) {
242                 if (_run > run_max) {
243                   run_max = _run;
244                   ini_run_max = ini_run;
245                 }
246               }
247               _run = 0;
248               ini_run = i;
249             } else {
250               _run++;
251             }
252             if (_run > run_max) {
253               run_max = _run;
254               ini_run_max = ini_run;
255             }
256           }
257           _SERIAL.print("  ini_run_max = ");
258           _SERIAL.print(ini_run_max,DEC);
259           _SERIAL.print(" : run_max = ");
260           _SERIAL.println(run_max,DEC);
261           if (run_max > OPTIMUM_WINDOW) {
```

```
262             _SERIAL.println("    -> Enough safe room -> Going forward");
263             angleW = 180 - (ini_run_max + run_max/2); // Opposite angle criteria!!!
264             // Ensure it is in a viable angle (60--120)
265             angleW = max(min(angleW , ANGLE_SERVOW_RIGHT_MAX) , ANGLE_SERVOW_LEFT_MAX);
266             servoW.write(angleW);
267             _SERIAL.print("        new wheels' angle = ");
268             _SERIAL.println(angleW,DEC);
269             changeStateToGoForward();
270           } else if (run_max > MINIMUM_WINDOW) {
271             _SERIAL.println("    -> Perhaps there is room -> Going cautious");
272             angleW = 180 - (ini_run_max + run_max/2); // Opposite angle criteria!!!
273             // Ensure it is in a viable angle (60--120)
274             angleW = max(min(angleW , ANGLE_SERVOW_RIGHT_MAX) , ANGLE_SERVOW_LEFT_MAX);
275             servoW.write(angleW);
276             _SERIAL.print("        new wheels' angle = ");
277             _SERIAL.println(angleW,DEC);
278             changeStateToGoCautiously();
279           } else {
280             _SERIAL.println("    -> No enough room -> Going back");
281             CAR_move(FORWARD,0,0); // stop the car
282             delay(200);
283             changeStateToGoBack();
284           }
285         } // end of block
286         break;
287       case RETROCESO: // it should never enter here
288         _SERIAL.println("  state: RETROCESO");
289         servoR.write(iRadar); // To give time to servo to stay at left when state changes
290         delay(RADAR_DELAY);
291         break;
292       case RETROCESO_ESCANEANDO:
293         _SERIAL.println("  state: RETROCESO_ESCANEANDO");
294         // TODO: it is the same than AVANCE_PRECAVIDO. Check also when stateChanged
295         _SERIAL.print("    iRadar = ");
296         _SERIAL.println(iRadar, DEC);
297         servoR.write(iRadar);
298         delay(RADAR_DELAY);
299         arrayDetections[iRadar] = digitalRead(SENSOR_IR_M);
300         _SERIAL.print("    radarDetection = ");
301         _SERIAL.println(arrayDetections[iRadar], DEC);
302         if (iRadar == ANGLE_SERVOR_LEFT_MAX) {
303           _SERIAL.println("    Arrived to maximum allowed angle. "
304                     "Going to choose rhumb...");
305           angleChangedWhenBack = false;
306           changeStateToChooseRhumb();
307         }
308         iRadar += iRdir;
309         break;
310     } // end switch(myState)
311   } // end of loop()
312
313   void CAR_move(unsigned char directionTo, unsigned char speed_left,
314           unsigned char speed_right) {
```

```
315     switch(directionTo) {
316       //car move forward with speed 180
317       case 0:
318         //left  motor clockwise rotation
319         digitalWrite(MOTOR_L_1,HIGH);
320         digitalWrite(MOTOR_L_2,LOW);
321         //right motor clockwise rotation
322         digitalWrite(MOTOR_R_1,HIGH);
323         digitalWrite(MOTOR_R_2,LOW);
324         break;
325       //car move back with speed 180
326       case 1:
327         //left  motor counterclockwise rotation
328         digitalWrite(MOTOR_L_1,LOW);
329         digitalWrite(MOTOR_L_2,HIGH);
330         //right motor counterclockwise rotation
331         digitalWrite(MOTOR_R_1,LOW);
332         digitalWrite(MOTOR_R_2,HIGH);
333         break;
334       default:
335         break;
336     }
337     // Set the speed for each motor (pair of wheels)
338     analogWrite(ENA_PIN,speed_left );
339     analogWrite(ENB_PIN,speed_right);
340   }
341
342   // changeStateToChooseRhumb()
343   void changeStateToChooseRhumb() {
344     _SERIAL.println("changeStateToChooseRhumb() --> ELECCION_RUTA");
345     myState = ELECCION_RUTA;
346     stateChanged = true;
347   } // end of changeStateToChooseRhumb()
348
349   // changeStateToGoCautiously()
350   void changeStateToGoCautiously() {
351     _SERIAL.println("changeStateToGoCautiously() --> AVANCE_PRECAVIDO");
352     myState = AVANCE_PRECAVIDO;
353     stateChanged = true;
354   } // end of changeStateToGoCautiously()
355
356   // changeStateToGoForward()
357   void changeStateToGoForward() {
358     _SERIAL.println("changeStateToGoForward() --> AVANCE_NORMAL");
359     myState = AVANCE_NORMAL;
360     stateChanged = true;
361   } // end of changeStateToGoForward()
362
363   // changeStateToGoBack(): moves the car backwards at lower speed for given time
364   void changeStateToGoBack() {
365     _SERIAL.println("changeStateToGoBack() --> RETROCESO");
366     myState = RETROCESO;
367     speedChanged = true;
```

```
368     speed_left   = SPEED_LOW;
369     speed_right  = SPEED_LOW;
370     moveCar();
371     delay(BACK_TIME);
372     _SERIAL.println("changeStateToGoBack() --> RETROCESO_ESCANEANDO");
373     myState = RETROCESO_ESCANEANDO; // OK, RETROCESO state could be avoided but
374                                     // I keep it for the shake of clarity
375     stateChanged = true;
376   } // end of changeStateToGoBack()
377
378   // checkSensors(): returns TRUE if no obstacle is detected, FALSE if detected.
379   bool checkSensors() {
380     irValueL = digitalRead(SENSOR_IR_L);//read the value of left obstacle module
381     irValueR = digitalRead(SENSOR_IR_R);//read the value of right obstacle module
382     // if obstacle module detect obstacles, it will out low level
383     return (irValueL == FREEWAY && irValueR == FREEWAY);
384   } // end of checkSensors()
385
386   // moveCar(): updates the car movement, depending on the private variables
387   void moveCar() {
388     switch(directionTo) {
389       case 0: //car move forward
390         //left  motor clockwise rotation
391         digitalWrite(MOTOR_L_1,HIGH);
392         digitalWrite(MOTOR_L_2,LOW);
393         //right motor clockwise rotation
394         digitalWrite(MOTOR_R_1,HIGH);
395         digitalWrite(MOTOR_R_2,LOW);
396         break;
397       case 1: //car move back
398         //left  motor counterclockwise rotation
399         digitalWrite(MOTOR_L_1,LOW);
400         digitalWrite(MOTOR_L_2,HIGH);
401         //right motor counterclockwise rotation
402         digitalWrite(MOTOR_R_1,LOW);
403         digitalWrite(MOTOR_R_2,HIGH);
404         break;
405       default:
406         break;
407     }
408     // Set the speed for each motor (pair of wheels)
409     analogWrite(ENA_PIN,speed_left );
410     analogWrite(ENB_PIN,speed_right);
411   } // end of moveCar()
```

C.2. Prototipo 4WD con tres sensores por IR

C.2.1. Conexionado

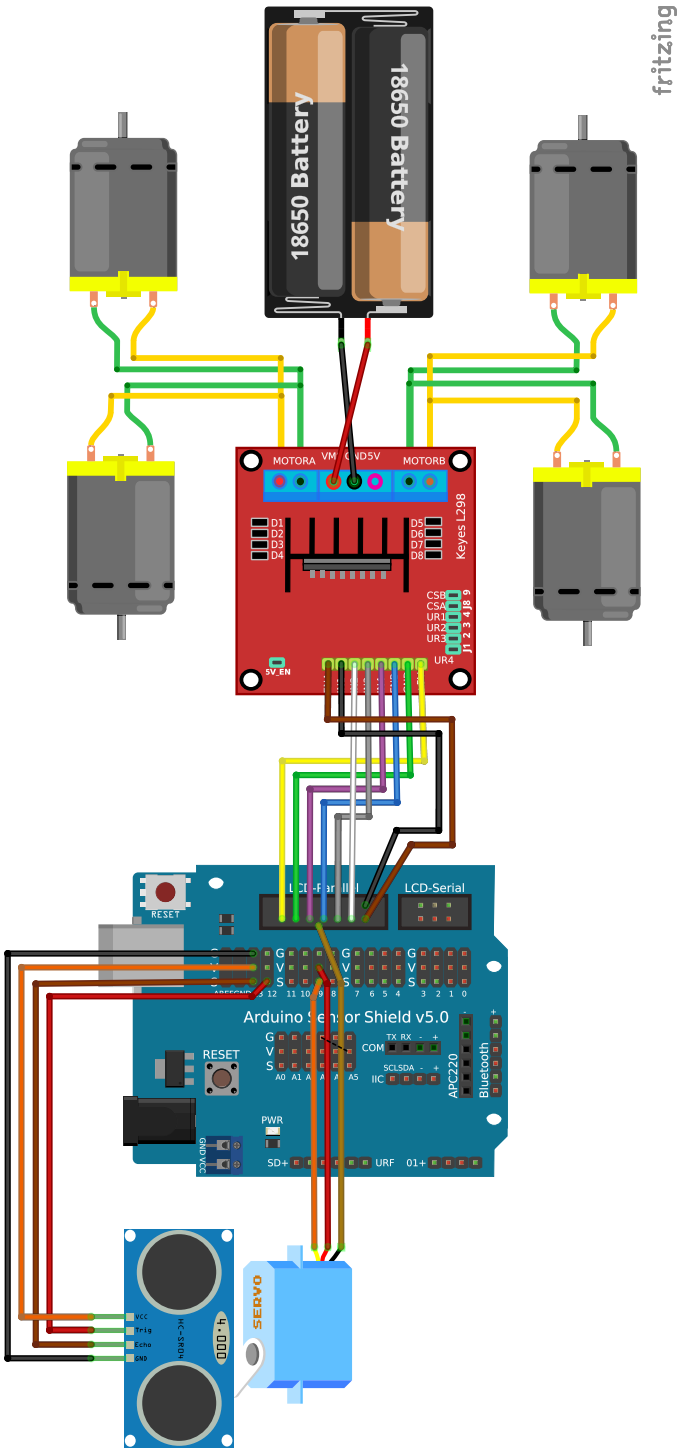

Figura C.2: Diagrama de conexionado de la evitación de obstáculos con sensores analógicos (v5) en el 4WD

C.2.2. Código en Arduino

Listado C.2: Evitación de obstáculos para Arduino usando el 4WD (v5b)

```
#include <Servo.h>

// Define movement directions
#define FRONT (8)
#define LEFT  (4)
#define BACK  (2)
#define RIGHT (6)
#define STAY  (0)

// Define servo movement capability
#define ANGLE_LEFT_MAX     (180) // Most left
#define ANGLE_LEFT_VISION  (130)
#define ANGLE_MIDDLE       ( 90) // Front
#define ANGLE_RIGHT_VISION ( 50)
#define ANGLE_RIGHT_MAX    (  0) // Most right
#define ANGLE_STEP         ( 10) // Scanning step. If too low, too many steps, too slow
#define ANGLE_WINDOW_RIGHT ( 75) // If center of free window less than this, turn right
#define ANGLE_WINDOW_LEFT  (105) // If center of free window more than this, turn left
#define SERVO_SPEED        (  2) // msÂº/. Specs' SG90 -> 100msÂº/60 -> 600msÂº/360
                                 // So define it as 2msÂº/ (conservative)

// Define parameters of the algorythm: delays in milliseconds
#define DELAY_FACTOR_F  (  8) // To calibrate according the prototipe response
#define DELAY_FACTOR_B  (  1) // To calibrate according the prototipe response
#define DELAY_BACK_RND  (200/DELAY_FACTOR_B) // Turn left X ms after going backward
#define DELAY_FRONT     ( 40/DELAY_FACTOR_F) // Advance during X ms minimum
#define DELAY_TURN      (600/DELAY_FACTOR_F) // Rotate at least during X ms
#define DELAY_STOP      (100)                // Stop at least during X ms
#define DELAY_BACK_NEAR (200/DELAY_FACTOR_B) // Goes back X ms if too obstacle too near
#define DELAY_BACK_BACK (500/DELAY_FACTOR_B) // Goes backward at least during X ms
#define DELAY_BACK_TURN (100/DELAY_FACTOR_B) // Goes back X ms before rotating
#define WINDOW_MINIMUM  ( 40) // Sexagesimal degrees. Minimum hole window to go forward
#define DIST_MINIMUM    ( 20) // Centimeters: Distance minimum to prevent a collision
#define DIST_WARNING    ( 40) // Centimeters: If obstacle near than this, scan front

// For debugging purposes via serial connection: Set to 0 on real tests
#define DEBUG (0)
#define _SERIAL if (DEBUG) Serial

// Pins 3 and 6 "reserved" for future use to set the speed of each pair of wheels
int pinLB      =  2; // Attached pin of the back  left  motor/wheel
int pinLF      =  4; // Attached pin the front left  motor/wheel
int pinRB      =  5; // Attached pin of the back  right motor/wheel
int pinRF      =  7; // Attached pin of the front right motor/wheel
int inputPin   = 13; // Attached pin for the ultrasonic signal receiver    (RX)
int outputPin  = 12; // Attached pin for the ultrasonic signal transmitter (TX)
int servoPin   =  9; // Attached pin for the servo moving the ultrasound sensor

int Fspace     = 0;    // Free forward space
```

```
50  int Rspace     = 0;    // Free fight   space
51  int Lspace     = 0;    // Free left    space
52  int directionn = STAY; // Front=8 : Left=4 : Back=2 : Right=6 : Stop=0
53                         // Lookout: direction is a reserved word in Arduino language!
54
55  Servo myservo;                // Create myservo object
56  int angle     = ANGLE_MIDDLE; // Angle to move the servo to
57  int prevAngle = ANGLE_MIDDLE; // Previous value of angle (last loop)
58  int incAngle  = 1;            // Moves +/-incAngle sexagesimals degrees when advancing
59  int distances[(ANGLE_LEFT_MAX-ANGLE_RIGHT_MAX)/ANGLE_STEP*2+2] = {0}; // Vector of dist
60
61  void setup() {
62    Serial.begin(115200);
63
64    // Set DC motor pinout
65    pinMode (pinLB    , OUTPUT); // backward action left  motor/wheels
66    pinMode (pinLF    , OUTPUT); // forward  action left  motor/wheels
67    pinMode (pinRB    , OUTPUT); // backward action right motor/wheels
68    pinMode (pinRF    , OUTPUT); // forward  action right motor/wheels
69    // Set UltraSound pinout
70    pinMode (inputPin , INPUT ); // Define ultrasound input pin
71    pinMode (outputPin, OUTPUT); // Define ultrasonic output pin
72
73    myservo.attach(servoPin);    // Define servo motor output section 5 pin (must be PWM)
74  } // end of setup()
75
76  void advance(int a) {        // Go forward
77    digitalWrite(pinRB, LOW) ; // make DC motor (left)  counterclockwise: Forward
78    digitalWrite(pinRF, HIGH);
79    digitalWrite(pinLB, LOW ); // make DC motor (right) clockwise: Forward
80    digitalWrite(pinLF, HIGH);
81    delay(a);
82  } // end of advance()
83
84  void turnR(int d) {          // Turn right (wheel)
85    digitalWrite(pinRB, LOW ); // make DC motor (left)  counterclockwise: Forward
86    digitalWrite(pinRF, HIGH);
87    digitalWrite(pinLB, HIGH); // make DC motor (right) counterclockwise: Backward
88    digitalWrite(pinLF, LOW );
89    delay(d);
90  } // end of turnR()
91
92  void turnL(int e) {          // Turn left (wheel)
93    digitalWrite(pinRB, HIGH); // make DC motor (left)  counterclockwise: Backward
94    digitalWrite(pinRF, LOW );
95    digitalWrite(pinLB, LOW ); // make DC motor (right) clockwise: Forward
96    digitalWrite(pinLF, HIGH);
97    delay(e);
98  } // end of turnL()
99
100 void stopp(int f) {          // Stop (lookout: stop is a reserved word in Arduino!)
101   digitalWrite(pinRB, HIGH); // make DC motor (left)  break: Stop
102   digitalWrite(pinRF, HIGH);
```

```
103     digitalWrite(pinLB, HIGH); // make DC motor (right) break: Stop
104     digitalWrite(pinLF, HIGH);
105     delay(f);
106   } // end of stopp()
107
108   void back(int g) {           // Go backward
109     digitalWrite(pinRB, HIGH); // make DC motor (left)  counterclockwise: Backward
110     digitalWrite(pinRF, LOW );
111     digitalWrite(pinLB, HIGH); // make DC motor (right) counterclockwise: Backward
112     digitalWrite(pinLF, LOW );
113     delay(g);
114   } // end of back()
115
116   void rotateServoTo(int alpha) {
117     _SERIAL.print("Rotating servo from ");
118     _SERIAL.print(prevAngle, DEC);
119     _SERIAL.print(" to ");
120     _SERIAL.print(alpha, DEC);
121     _SERIAL.print(" : Delay=");
122     _SERIAL.println(abs(alpha-prevAngle)*2, DEC);
123     if (prevAngle==alpha) return;  // Don't waste energy if already at desired position
124     myservo.write(alpha);          // Sets the new angle position
125     delay(abs(alpha-prevAngle)*2); // Waits untill servo reaches the new position
126     prevAngle = alpha;             // Stores this new angle position for future use
127     return;
128   } // end of rotateServoTo()
129
130   void detection() {           // Detects obstacles by measuring the distance
131     Fspace = ask_pin(angle);   // Read from front (angle)
132     _SERIAL.print("Front distance = ");
133     _SERIAL.println(Fspace, DEC);
134     if (Fspace<DIST_MINIMUM) { // If the distance is less than 10 cm in front of
135       _SERIAL.println("ALERT!, going back!");
136       stopp(DELAY_STOP);       // Clear the output data
137       back(DELAY_BACK_NEAR);   // Goes backward for DELAY_BACK_NEAR ms
138     }
139
140     if (Fspace<DIST_WARNING) { // If the front distance is less than DIST_WARNING cm
141       _SERIAL.println("Warning... look out!");
142       stopp(DELAY_STOP);       // Clear the output data
143       int i=0;
144       int _run = 0;
145       int _ini_run = 0;
146       int _run_max = 0;
147       int _ini_run_max = 0;
148       if (prevAngle < ANGLE_MIDDLE) {   // scan all front from the right to the left side
149         rotateServoTo(ANGLE_RIGHT_MAX);
150         for (int alpha=ANGLE_RIGHT_MAX;
151              alpha<=ANGLE_LEFT_MAX;
152              alpha+=ANGLE_STEP, i++) {
153           distances[i] = ask_pin(alpha);// stores each distance measured during the scan
154           _SERIAL.print("  alpha=");
155           _SERIAL.print(alpha, DEC);
```

```
156           _SERIAL.print(" : dist[]=");
157           _SERIAL.println(distances[i], DEC);
158           // Checks if a full window has been detected, if so stores data of the greatest
159           if (distances[i] > DIST_WARNING && alpha<=ANGLE_LEFT_MAX-ANGLE_STEP) {
160             _run += ANGLE_STEP;
161           } else {
162             if (_run > _run_max) {
163               _run_max     = _run;
164               _ini_run_max = _ini_run;
165             }
166             _run     = 0;
167             _ini_run = alpha;
168           }
169         }
170       } else {                          // scan all front from the left to the right side
171         rotateServoTo(ANGLE_LEFT_MAX);
172         for (int alpha=ANGLE_LEFT_MAX;
173              alpha>=ANGLE_RIGHT_MAX;
174              alpha-=ANGLE_STEP, i++) {
175           distances[i] = ask_pin(alpha);// stores each distance measured during the scan
176           _SERIAL.print("  alpha=");
177           _SERIAL.print(alpha, DEC);
178           _SERIAL.print(" : dist[]=");
179           _SERIAL.println(distances[i], DEC);
180           // Checks if a full window has been detected, if so stores data of the greatest
181           if (distances[i] > DIST_WARNING && alpha>=ANGLE_RIGHT_MAX+ANGLE_STEP) {
182             _run -= ANGLE_STEP;
183           } else {
184             if (_run < _run_max) {     // _run_max is negative
185               _run_max     = _run;
186               _ini_run_max = _ini_run;
187             }
188             _run     = 0;
189             _ini_run = alpha;
190           }
191           _SERIAL.print("  _ini_run=");
192           _SERIAL.print(_ini_run, DEC);
193           _SERIAL.print(" : _run=");
194           _SERIAL.print(_run, DEC);
195           _SERIAL.print(" : _ini_run_max=");
196           _SERIAL.print(_ini_run_max, DEC);
197           _SERIAL.print(" : _run_max=");
198           _SERIAL.println(_run_max, DEC);
199         } // end for (scanning filling distances vector and checking max free window)
200         _ini_run_max = _ini_run_max + _run_max; // Change to match the positive criteria
201         _run_max = -_run_max;
202         _SERIAL.print("  END->CHANGE: _ini_run=");
203         _SERIAL.print(_ini_run, DEC);
204         _SERIAL.print(" : _run=");
205         _SERIAL.print(_run, DEC);
206         _SERIAL.print(" : _ini_run_max=");
207         _SERIAL.print(_ini_run_max, DEC);
208         _SERIAL.print(" : _run_max=");
```

```
209         _SERIAL.println(_run_max, DEC);
210       } // end if [radar scanning 0..180 looking for obstacles and free windows]
211       _SERIAL.print("_run_max=");
212       _SERIAL.print(_run_max, DEC);
213       _SERIAL.print(" : _ini_run_max=");
214       _SERIAL.print(_ini_run_max, DEC);
215       if (_run_max >= WINDOW_MINIMUM) {
216         // TODO: IDEA: supersed directionn with angle, perhaps works better
217         angle = _ini_run_max + _run_max/2;
218         _SERIAL.print(" : alpha=");
219         _SERIAL.println(angle, DEC);
220         if (angle < ANGLE_WINDOW_RIGHT) {
221           directionn = RIGHT;  // Turn right
222           _SERIAL.print(" --> RIGHT");
223         } else if (angle > ANGLE_WINDOW_LEFT) {
224           directionn = LEFT ;  // Turn left
225           _SERIAL.print(" --> LEFT");
226         } else {
227           directionn = FRONT;  // Go straight
228           _SERIAL.print(" --> FRONT");
229         }
230         angle = ANGLE_MIDDLE;  // Put the ultrasound sensor in the center
231         _SERIAL.print(" : angle=");
232         _SERIAL.println(angle, DEC);
233         rotateServoTo(angle);
234       } else { // if (_run_max >= WINDOW_MINIMUM)
235         directionn = BACK;     // Go backwards
236         _SERIAL.println(" --> BACK");
237       }
238     } else { // if (Fspace<DIST_WARNING)
239       directionn = FRONT;      // Move forward
240       _SERIAL.println("No problem, continue --> FRONT");
241     }
242   } // end of detection()
243
244   int ask_pin(int alpha) {         // Measure the distance from the given angle
245     rotateServoTo(alpha);
246     digitalWrite(outputPin, LOW ); // Let ultrasonic transmitter low voltage 3 Î¼s
247     delayMicroseconds(3);
248     digitalWrite(outputPin, HIGH); // Let ultrasonic transmitter high voltage 15 Î¼s
249     delayMicroseconds(15);
250     digitalWrite(outputPin, LOW ); // Maintain low voltage ultrasonic transmitter
251     float Fdistance = pulseIn(inputPin, HIGH); // Read worse time difference
252     Fdistance = Fdistance/58;      // Time to turn to the distance (unit: cm)
253     _SERIAL.print("Fdistance:");   // Output distance (unit: cm)
254     _SERIAL.println(Fdistance);    // Display the distance
255     return (int) Fdistance;        // Read into the distance space
256   } // end of ask_pin()
257
258   void loop() {
259     detection();             // Scan to get the better angle and direction where to move
260     if (directionn == BACK ){// If directionn (direction) = 2 (go backwards)
261       back(DELAY_BACK_BACK); // Go backwards
```

```
262       turnL(DELAY_BACK_RND); // Move slightly to left, to get prevent stuck in dead alley
263       _SERIAL.print("Back"); // Display direction (backwards)
264     }
265     if (directionn == RIGHT){// If directionn (direction) = 6 (turn right)
266       back(DELAY_BACK_TURN);
267       turnR(DELAY_TURN);     // Turn right
268       _SERIAL.print("Right");// Display direction (turn left)
269     }
270     if (directionn == LEFT ){// If directionn (direction) = 4 (turn left)
271       back(DELAY_BACK_TURN);
272       turnL(DELAY_TURN);     // Turn left
273       _SERIAL.print("Left"); // Display direction (turn right)
274     }
275     if (directionn == FRONT){// If directionn (direction) = 8 (go forward)
276       advance(DELAY_FRONT);  // Normal, forward
277       _SERIAL.print("Front");// Display direction (forward)
278       angle += incAngle;
279       //myservo.write(angle);
280       if      (angle >= ANGLE_LEFT_VISION ) incAngle = -ANGLE_STEP;
281       else if (angle <= ANGLE_RIGHT_VISION) incAngle = +ANGLE_STEP;
282     }
283   } // end of loop()
```

(Página intencionalmente en blanco)

ANEXO D

ACC: CONEXIONADO Y CÓDIGO

A pesar de que la funcionalidad del control de velocidad adaptativo se ha implementado únicamente en un prototipo, el 4WD, en este Anexo se presenta también la modificación de la implementación del seguimiento de líneas en el SunFounder™, que modifica periódicamente la velocidad del prototipo para ayudar a visualizar en las pruebas el comportamiento de la funcionalidad del ACC implementada en el 4WD.

D.1. Prototipo SunFounder™

El conexionado del SunFounder™ utilizado en esta prueba es exactamente el mismo que el ya mostrado en la Figura A.1.

D.1.1. Código en Arduino

Listado D.1: Seguimiento de trayecto modificado usando el SunFounder™ (v7c)

```
#include <Servo.h>     // To control the servo
#include <MsTimer2.h>  // To manage interruptions

Servo myservo;

// Define Arduino pinout
// - for DC motors
#define ENA_PIN   ( 5) // ENA attached to pin5
#define ENB_PIN   ( 6) // ENB attached to pin6
#define MOTOR_L_1 ( 8) // MOTOR_L_1 attached to pin1
#define MOTOR_L_2 ( 9) // MOTOR_L_1 attached to pin9
#define MOTOR_R_1 (10) // MOTOR_R_1 attached to pin10
#define MOTOR_R_2 (11) // MOTOR_R_1 attached to pin11
#define SERVO_PIN ( 2) // Servo attached to pin2
// - for IR line tracking sensors
#define IR_LEFT_MAX_PIN   (A0) //attach the left  first  Tracking module pinA0 to A0
#define IR_LEFT_HALF_PIN  (A1) //attach the left  second Tracking module pinA0 to A1
#define IR_MIDDLE_PIN     (A2) //attach the central      Tracking module pinA0 to A2
#define IR_RIGHT_HALF_PIN (A3) //attach the right second Tracking module pinA0 to A3
#define IR_RIGHT_MAX_PIN  (A5) //attach the right first  Tracking module pinA0 to A5
// WARNING: A4 pins are not working on my original SunFounder Sensor Shield v.5.1

// Indexes for the vector of exponential mobil average
#define NUM_SENSORS   (5)
#define IR_LEFT_MAX   (0)
#define IR_LEFT_HALF  (1)
#define IR_MIDDLE     (2)
#define IR_RIGHT_HALF (3)
#define IR_RIGHT_MAX  (4)

#define FORWARD 0 // car moves forward
#define BACK    1 // car moves back

// Define physical axis' angle properties
#define ANGLE_SERVO_BIAS ( -7) // To alineate the front wheels if servo not ortogonal
#define ANGLE_LEFT_MAX   ( 60+ANGLE_SERVO_BIAS)
#define ANGLE_LEFT_HALF  ( 75+ANGLE_SERVO_BIAS)
#define ANGLE_MIDDLE     ( 90+ANGLE_SERVO_BIAS)
#define ANGLE_RIGHT_HALF (115+ANGLE_SERVO_BIAS)
#define ANGLE_RIGHT_MAX  (120+ANGLE_SERVO_BIAS)

// Define weighting parameters for updating sensor values on each iteration
#define W_LEFT_MAX   (-30.0f)
#define W_LEFT_HALF  (-15.0f)
#define W_MIDDLE     (  0   )
#define W_RIGHT_HALF (+15.0f)
#define W_RIGHT_MAX  (+30.0f)

// Algorythm main parameters
```

```
#define TOUT_MAX  (2000) // When tout exceeds it, car starts looking for black line
#define TOUT_INC  (   3) // tout+=TOUT_INC each (backwards) iteration
#define TOUT_STEP ( 100) // When steps exceeds it, car change its backward's direction
#define THRESHOLD (1000) // IR sensor analog value to say if black (1) or white (0)
#define ALPHA     (0.1f) // 0..1: Higher values gives more importance to present sensor
                         //       values. Used in history angle
#define BETA     (0.05f) // 0..1: Higher values gives more importance to present sensor
                         //       values. Used in history vector

// Defines for the dynamic speed change to test ACC together with the W4D car
#define SPEED_TIME_FREQ (5000) // in milliseconds
#define SPEED_HIGH (210) //
#define SPEED_LOW  (160) //
#define SPEED_MIN  (120) //

// Set DEBUGX to 0 when loading into the real car for a real test (or lower TOUT_MAXv)
#define DEBUG0 0      // Delays execution to allow easier debugging
#define DEBUG1 0      // Debugs basic state change
#define DEBUG2 0      // More detailed debug info
#define PRINT1(r)     if (DEBUG1) Serial.print(r)
#define PRINT2(r,s)   if (DEBUG2) Serial.print(r,s)
#define PRINTLN1(r)   if (DEBUG1) Serial.println(r)
#define PRINTLN2(r,s) if (DEBUG2) Serial.println(r,s)

// Dynamic speeds
int highSpeed = SPEED_HIGH;
int lowSpeed  = SPEED_MIN;

// Algorythmic variables
signed   char mean  = 0; // Present weighted sensors value
signed   char line  = 0; // Number of sensors detecting line (n on algorythm descript.)
signed   int  tout  = 0; // Counts how many backward iterations with no black detection
signed   int  steps = 0; // Counts how many iterations after the last random dir change
unsigned char velo  = 0; // Preent DC motors speed
unsigned char angle = ANGLE_MIDDLE; // Present front axis' angle
float prevangle_f = 0.0f;// Previous angle value, in float precission
float mean_f      = 0.0f;// Present weighted sensors value, in float precission
float history[NUM_SENSORS]; // Given (weight calculated) values for each sensor

void setup() {
  /* Software serial initializaion for debugging purposes */
  Serial.begin (115200);
  // Arduino Leonardo: wait for the serial port to connect
  while (!Serial) {};
  PRINT1("Seguimiento. v7b");

  /* set below pins as OUTPUT */
  pinMode(ENA_PIN  , OUTPUT);
  pinMode(ENB_PIN  , OUTPUT);
  pinMode(MOTOR_L_1, OUTPUT);
  pinMode(MOTOR_L_2, OUTPUT);
  pinMode(MOTOR_R_1, OUTPUT);
  pinMode(MOTOR_R_2, OUTPUT);
```

```
103     myservo.attach(SERVO_PIN); // servo attached to SERVO_PIN
104     prevangle_f = ANGLE_MIDDLE;
105     angle = (unsigned char) prevangle_f;
106     myservo.write(angle);      // set the angle of servo
107     for (int i=0; i<NUM_SENSORS; i++) {
108       history[i] = 0.0f;
109     }
110     MsTimer2::set(SPEED_TIME_FREQ, changeSpeed); // Every X ms change speed LOW<->HIGH
111     MsTimer2::start();                           // Start timing
112   } // end of setup()
113
114   void changeSpeed() { // Toggles SPEED_HIGH and SPEED_LOW as maximum current speed
115     highSpeed = (highSpeed==SPEED_HIGH) ? SPEED_LOW : SPEED_HIGH;
116   } // end of changeSpeed()
117
118   void loop() {
119     if (DEBUG0) delay (500);
120     PRINTLN1("");
121     PRINT1("INICIO: Angle=");
122     PRINT2(angle,DEC);
123     PRINT1(" : SENSORES=");
124
125     // statistical variables
126     mean_f = 0.0f;
127     line = 0; // checks if any sensor has detected a line (1) or none of them have (0)
128
129     // FIRST MEASURE LINES
130     //1) Checks each sensor for black line detection
131     //   (present analog value > threshold ==> black; if not, white)
132     //2) updates moving average exponential,
133     //3) and weight-averages their mean value
134     if(analogRead(IR_LEFT_MAX_PIN)  > THRESHOLD) {//if read the value of the left first
            Tracking module pinA0 is more than THRESHOLD
135         PRINT2(1,DEC);
136         line += 1;
137         history[IR_LEFT_MAX] = history[IR_LEFT_MAX]*(1.0f-BETA) + BETA;
138     } else {
139         PRINT2(0,DEC);
140         history[IR_LEFT_MAX] = history[IR_LEFT_MAX]*(1.0f-BETA);
141     }
142     mean_f += (W_LEFT_MAX * history[IR_LEFT_MAX]);
143     if(analogRead(IR_LEFT_HALF_PIN)  > THRESHOLD) {//if read the value of the left second
             Tracking module pinA0 is more than THRESHOLD
144         PRINT2(1,DEC);
145         line += 1;
146         history[IR_LEFT_HALF] = history[IR_LEFT_HALF]*(1.0f-BETA) + BETA;
147     } else {
148         PRINT2(0,DEC);
149         history[IR_LEFT_HALF] = history[IR_LEFT_HALF]*(1.0f-BETA);
150     }
151     mean_f += (W_LEFT_HALF * history[IR_LEFT_HALF]);
152     if(analogRead(IR_MIDDLE_PIN)  > THRESHOLD) {//if read the value of the module
            Tracking module pinA0 is more than THRESHOLD
```

```
153         PRINT2(1,DEC);
154         line += 1;
155         history[IR_MIDDLE] = history[IR_MIDDLE]*(1.0f-BETA) + BETA;
156     } else {
157         PRINT2(0,DEC);
158         history[IR_MIDDLE] = history[IR_MIDDLE]*(1.0f-BETA);
159     }
160     //mean_f += (W_MIDDLE * history[IR_MIDDLE]); // It's always ZERO
161     if(analogRead(IR_RIGHT_HALF_PIN) > THRESHOLD) {//if read the value of the right
        second Tracking module pinA0 is more than THRESHOLD
162         PRINT2(1,DEC);
163         line += 1;
164         history[IR_RIGHT_HALF] = history[IR_RIGHT_HALF]*(1.0f-BETA) + BETA;
165     } else {
166         PRINT2(0,DEC);
167         history[IR_RIGHT_HALF] = history[IR_RIGHT_HALF]*(1.0f-BETA);
168     }
169     mean_f += (W_RIGHT_HALF * history[IR_RIGHT_HALF]);
170     if(analogRead(IR_RIGHT_MAX_PIN) > THRESHOLD) {//if read the value of the right first
        Tracking module pinA0 is more than THRESHOLD
171         PRINT2(1,DEC);
172         line += 1;
173         history[IR_RIGHT_MAX] = history[IR_RIGHT_MAX]*(1.0f-BETA) + BETA;
174     } else {
175         PRINT2(0,DEC);
176         history[IR_RIGHT_MAX] = history[IR_RIGHT_MAX]*(1.0f-BETA);
177     }
178     mean_f += (W_RIGHT_MAX * history[IR_RIGHT_MAX]);
179     mean = (signed char) mean_f;
180
181     // Show detailed debugging info about mean calculation
182     PRINT1(" : History=(");
183     for (int i=0; i<NUM_SENSORS; i++) {
184       PRINT2(history[i],DEC);
185       PRINT1("-");
186     }
187     PRINT1(" : mean_f=");
188     PRINT2(mean_f,DEC);
189     PRINT1(" : mean=");
190     PRINT2(mean,DEC);
191     PRINT1(" : line=");
192     PRINTLN2(line,DEC);
193
194     // THEN TAKE ACTIONS
195     if (line == 0) {         // If none sensor has detected a black line... (going back)
196       if (tout > TOUT_MAX) { // If reached maximum iterations going back since line lost
197         steps++;
198         if (steps > TOUT_STEP) { // If reached maximum iterations to change randomly dir.
199           steps=0;               // reset counter
200           angle = random(ANGLE_LEFT_MAX,ANGLE_RIGHT_MAX); // choose a new random angle
201           prevangle_f = prevangle_f*(1.0f-ALPHA) + (float) angle*ALPHA; // float precissi
202           angle = (unsigned char) prevangle_f;
203           myservo.write(angle);          // set the angle of servo
```

```
204         }
205         velo = lowSpeed;                  // moving backwards at low speed
206         CAR_move(BACK,lowSpeed,lowSpeed);// car moves BACK much slower
207                                           // trying to find again the lost line
208         PRINT1("RETROCEDIENDO : tout=");
209       } else {                    // First stage of moving back for a while since line lost
210         velo = lowSpeed;          // Set lower speed to move back
211         CAR_move(FORWARD,lowSpeed,lowSpeed); // Move back at low speed
212         tout += TOUT_INC;         // Iterations counter
213         PRINT1("AVANZANDO A CIEGAS : tout=");
214       }
215     } else {                  // At least one sensor has detected a black line under it
216       angle = ANGLE_MIDDLE - mean/line; // new angle, depending on where the line is
217       prevangle_f = prevangle_f*(1.0f-ALPHA) + (float)angle*ALPHA; // update angle value
218       angle = (unsigned char) prevangle_f;
219       myservo.write(angle);             // set the angle of servo
220       tout = 0;                         // reset backwards iteration counting
221       velo = (unsigned char) (highSpeed -
222              (signed char)(highSpeed-lowSpeed)*abs((float)(angle-ANGLE_MIDDLE) /
223              (float)(ANGLE_RIGHT_MAX-ANGLE_MIDDLE)));  // Exponential moving average
224       CAR_move(FORWARD,velo,velo);      // car move forward at desired "velo" speed
225       PRINT1("AVANZANDO NORMAL : tout=");
226     }
227     PRINT2(tout,DEC);
228     PRINT1(" : angle=");
229     PRINT2(angle,DEC);
230     PRINT1(" : velo=");
231     PRINTLN2(velo,DEC);
232   } // end of loop()
233
234   void CAR_move(unsigned char directionTo, unsigned char speed_left, unsigned char
        speed_right) {
235     switch(directionTo) {
236       //car moves forward
237       case FORWARD:
238         //left  motor clockwise rotation --> forward
239         digitalWrite(MOTOR_L_1,HIGH);
240         digitalWrite(MOTOR_L_2,LOW );
241         //right motor clockwise rotation --> forward
242         digitalWrite(MOTOR_R_1,HIGH);
243         digitalWrite(MOTOR_R_2,LOW );
244         break;
245       //car moves back
246       case BACK:
247         //left  motor counterclockwise rotation --> back
248         digitalWrite(MOTOR_L_1,LOW);
249         digitalWrite(MOTOR_L_2,HIGH);
250         //right motor counterclockwise rotation --> back
251         digitalWrite(MOTOR_R_1,LOW);
252         digitalWrite(MOTOR_R_2,HIGH);
253         break;
254       default:
255         break;
```

```
256     }
257     // PWM: Sets the speed of both DC motors, writting speed_X to the ENA_PIN or ENB_PIN
258     analogWrite(ENA_PIN,speed_left );
259     analogWrite(ENB_PIN,speed_right);
260   } // end of CAR_move()
```

D.2. Prototipo 4WD para el ACC

D.2.1. Conexionado

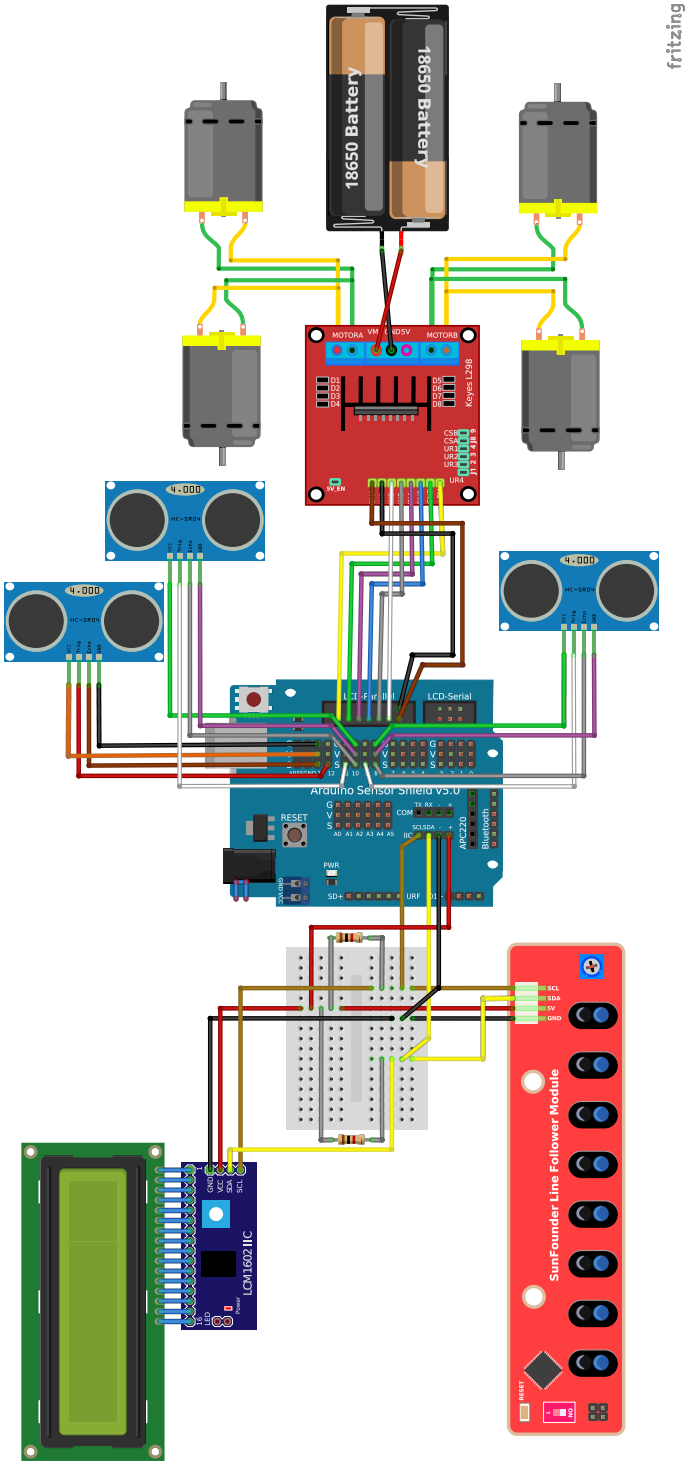

Figura D.1: Diagrama de conexionado del control de velocidad adaptativo del 4WD

D.2.2. Código en Arduino

Listado D.2: Control de velocidad adaptativo (ACC) para Arduino usando el 4WD (v5j)

```
/* Arduino AutoNavigator by Tomás Navarro Cosme, Valencia, september 2016
 * Implementation on the 4WD prototipe of an ACC (Adaptive Cruise Control)
 */

#include <Wire.h>              // for the IIC module(s): IR and LCD
#include <LiquidCrystal_I2C.h> // Using version 1.2.1
#include <stdio.h>             // for sprintf()
#include "moveDirections.h"    // enum type states to keep track of previous state

// Speed parameters for the line tracking and distance control
#define SPEED_MAX      (250)
#define SPEED_HIGH     (230)
#define SPEED_MED      (210)
#define SPEED_LOW      (180)
#define SPEED_MIN      (150)
#define SPEED_ZERO     (  0)
#define STEPS_BACK_MIN ( 10)
#define DELAY_BACK     (  0)
#define DELAY_TURN     (  0)
#define DELAY_WING     (  0)
#define DELAY_FRONT    (  0)
#define DELAY_STOP     (  0)
#define DELAY_SAME     (  0)

// Distance parameters for the cruise control (in centimeters)
#define DIST_MINIMUM ( 30) // Minimum distance setpoint PID
#define DIST_DESIRED ( 60) // Desired distance setpoint PID
#define DIST_MAXIMUM ( 90) // Maximum distance setpoint PID
#define DIST_WARNING ( 10) // Warning distance:if lower stop the car to avoid collision

// To de/active debug mode (SLOW!): Set all to 0 for real tests
// For debugging purpouses
#define DEBUGL0 (0) // LCD: Show sensors data
#define DEBUGL1 (0) // LCD: Show sensors data
#define DEBUG0  (0) // <-- DISABLE CRUISE CONTROL -->
#define DEBUG1  (0) // Show detailed steps
#define DEBUG2  (0) // Show loop time
#define DEBUG3  (0) // Show detailed times for US sensor reading rutines
#define DEBUG4  (0) // Show detailed times for IR line tracking sensor reading
#define DEBUG5  (0) // Show detailed times just for US sensor reading
#define DEBUG6  (0) // Show IR Module internals
#define DEBUG7  (0) // Show IR Module results
#define DEBUG8  (0) // Show space/obstacle detection values
#define DEBUG9  (0) // Show PID values
#define LCD0     if (DEBUGL0) lcd
#define LCD1     if (DEBUGL1) lcd
#define _SERIAL0 if (DEBUG0)  Serial
#define _SERIAL1 if (DEBUG1)  Serial
#define _SERIAL2 if (DEBUG2)  Serial
```

```
50  #define _SERIAL3 if (DEBUG3)  Serial
51  #define _SERIAL4 if (DEBUG4)  Serial
52  #define _SERIAL5 if (DEBUG5)  Serial
53  #define _SERIAL6 if (DEBUG6)  Serial
54  #define _SERIAL7 if (DEBUG7)  Serial
55  #define _SERIAL8 if (DEBUG8)  Serial
56  #define _SERIAL9 if (DEBUG9)  Serial
57
58  unsigned long startTime  =0, startTimeLoop  =0;
59  unsigned long elapsedTime=0, elapsedTimeLoop=0;
60
61  // Define Arduino pinout for DC motors
62  int pinMotorR2 = 7; // Clockwise        pin of right side motors
63  int pinMotorR1 = 5; // Counterclockwise pin of right side motors
64  int pinSpeedR  = 6; // Right side motor: ENable pin. Used for PWM -> speed
65  int pinMotorL2 = 4; // Clockwise        pin of left  side motors
66  int pinMotorL1 = 2; // Counterclockwise pin of left  side motors
67  int pinSpeedL  = 3; // Left  side motor: ENable pin. Used for PWM -> speed
68
69  // Line tracking sensors values
70  int SLM = LOW; // Line tracking left  sensor status
71  int SLH = LOW; // Line tracking front sensor status
72  int SRH = LOW; // Line tracking front sensor status
73  int SRM = LOW; // Line tracking right sensor status
74
75  //IIC SunFounder 8Channel Infrared Detection Tracking Sensor Module for Smart Car Robot
76  #define IR_MODULE_ADDRESS (0x09) // Address of the module in the IIC bus (slave)
77  unsigned char data[16];    // To store IR module values
78  #define IR_LEFT_MAX   (10) // Element 10 of the data[16] vector
79  #define IR_LEFT_HALF  ( 8) // Element 10 of the data[16] vector
80  #define IR_RIGHT_HALF ( 6) // Element 10 of the data[16] vector
81  #define IR_RIGHT_MAX  ( 4) // Element 10 of the data[16] vector
82  #define IR_THRESHOLD (200) // 255->White:0->Black; Greater if more oscure.Lower lighter
83  // IIC LCD
84  #define LCD_ADDRESS (0x27) // Address of the module in the IIC bus (slave)
85  LiquidCrystal_I2C lcd(LCD_ADDRESS, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
86  char lcdText[17] = {0};    // To store data before
87
88  // Line track recovering
89  int frontSteps = 0;
90  int backSteps  = 0;
91  boolean goingFront = true ;
92  boolean goingBack  = false;
93
94  // Ultrasonic sensors (HC-SR04)
95  int inputPinL  = 13; // Pin ultrasonic signal reception   (left)
96  int outputPinL = 12; // Pin ultrasonic signal transmitter (left)
97  int inputPinF  = 11; // Pin ultrasonic signal reception   (front)
98  int outputPinF = 10; // Pin ultrasonic signal transmitter (front)
99  int inputPinR  =  9; // Pin ultrasonic signal reception   (right)
100 int outputPinR =  8; // Pin ultrasonic signal transmitter (right)
101
102 // Measured distances
```

```
103  int Fspace     = 0;  // Front free space
104  int Rspace     = 0;  // Right free space
105  int Lspace     = 0;  // Left  free space
106  int prevDist   = 0;  // Previous front free space
107  bool turning   = false;
108
109  // PID control definitions and variables
110  #define KPP ( 0.5f) // Proportional term of the PID controller
111  #define KPI ( 0.2f) // Integral     term of the PID controller
112  #define KPD (40.0f) // Derivative   term of the PID controller
113  #define MAX_ERROR    ( 30) // Absolute maximum signal error
114  #define DIST_MAX_ERR ( 20) // Absolute maximum distance between two measurements
115  #define DIST_MAX_INT (100) // Absolute maximum integral part of the PID
116  float integral   = 0.0f;
117  float error      = 0.0f;
118  float signal     = 0.0f;
119  int   speedF     = SPEED_MED;
120  int   prevSpeedF = 0;
121  move_t _move = _STOP;
122
123  void cronometer(unsigned long start, const char *text) {
124    elapsedTime = millis() - start;
125    _SERIAL3.print(text);
126    _SERIAL3.println(elapsedTime, DEC);
127  } // end of cronometer()
128
129  void advance(int _speed, int a) {// Forward
130    _SERIAL1.print("_speed=");
131    _SERIAL1.print(_speed, DEC);
132    _SERIAL1.print(" : speedF=");
133    _SERIAL1.println(speedF, DEC);
134    if (_speed != prevSpeedF) {
135      analogWrite(pinSpeedL, _speed);
136      analogWrite(pinSpeedR, _speed);
137      prevSpeedF = speedF;
138    }
139    digitalWrite(pinMotorR1, LOW) ; // make DC motor (left) counterclockwise: Forward
140    digitalWrite(pinMotorR2, HIGH);
141    digitalWrite(pinMotorL1, LOW ); // make DC motor (right) clockwise: Forward
142    digitalWrite(pinMotorL2, HIGH);
143    goingFront = true;
144    turning = false;
145    _move = _FRONT;
146    delay(a);
147  } // end of advance()
148
149  void turnL(int _speed, int d){    // Turn right (rotational)
150    if (_speed != prevSpeedF) {
151      analogWrite(pinSpeedL, _speed);
152      analogWrite(pinSpeedR, _speed);
153      prevSpeedF = speedF;
154    }
155    digitalWrite(pinMotorR1, LOW ); // make DC motor (left) counterclockwise: Forward
```

```
156     digitalWrite(pinMotorR2, HIGH);
157     digitalWrite(pinMotorL1, HIGH); // make DC motor (right) counterclockwise: Back
158     digitalWrite(pinMotorL2, LOW );
159     goingFront = false;
160     turning = true;
161     _move = _TURNL;
162     delay(d);
163   } // end of turnR()
164
165   void turnR(int _speed, int e){    // Turn right (rotational)
166     if (_speed != prevSpeedF) {
167       analogWrite(pinSpeedL, _speed);
168       analogWrite(pinSpeedR, _speed);
169       prevSpeedF = speedF;
170     }
171     digitalWrite(pinMotorR1, HIGH); // make DC motor (left) clockwise: Back
172     digitalWrite(pinMotorR2, LOW );
173     digitalWrite(pinMotorL1, LOW ); // make DC motor (right) clockwise: Forward
174     digitalWrite(pinMotorL2, HIGH);
175     goingFront = false;
176     turning = true;
177     _move = _TURNR;
178     delay(e);
179   } // end of turnL()
180
181   void left(int _speed, int b) {   // Turn left (single wheel)
182     if (_speed != prevSpeedF) {
183       analogWrite(pinSpeedL, _speed/2);
184       analogWrite(pinSpeedR, _speed);
185       prevSpeedF = speedF;
186     }
187     digitalWrite(pinMotorR1, LOW ); // make DC motor (left) counterclockwise: Forward
188     digitalWrite(pinMotorR2, HIGH);
189     digitalWrite(pinMotorL1, LOW ); // no action on CD motor (right): Stop
190     digitalWrite(pinMotorL2, HIGH);
191     goingFront = false;
192     turning = true;
193     _move = _LEFT;
194     delay(b);
195   } // end of right()
196
197   void right(int _speed, int c) {    // Turn right (single wheel)
198     if (_speed != prevSpeedF) {
199       analogWrite(pinSpeedL, _speed);
200       analogWrite(pinSpeedR, _speed/2);
201       prevSpeedF = speedF;
202     }
203     digitalWrite(pinMotorR1, LOW ); // no action on CD motor (left): Stop
204     digitalWrite(pinMotorR2, HIGH);
205     digitalWrite(pinMotorL1, LOW ); // make DC motor (right) clockwise: Forward
206     digitalWrite(pinMotorL2, HIGH);
207     goingFront = false;
208     turning = true;
```

```
209     _move = _RIGHT;
210     delay(c);
211   } // end of left()
212
213   void stopp(int f) {               // Stop
214     digitalWrite(pinMotorR1, HIGH); // block CD motor (left): Stop
215     digitalWrite(pinMotorR2, HIGH);
216     digitalWrite(pinMotorL1, HIGH); // block CD motor (right): Stop
217     digitalWrite(pinMotorL2, HIGH);
218     goingFront = false;
219     turning = false;
220     _move = _STOP;
221     delay(f);
222   } // end of stopp()
223
224   void back(int _speed, int g) {    // Back
225     if (_speed != prevSpeedF) {
226       analogWrite(pinSpeedL, _speed);
227       analogWrite(pinSpeedR, _speed);
228       prevSpeedF = speedF;
229     }
230     digitalWrite(pinMotorR1, HIGH); // make DC motor (left) clockwise: Back
231     digitalWrite(pinMotorR2, LOW );
232     digitalWrite(pinMotorL1, HIGH); // make DC motor (right) counterclockwise: Back
233     digitalWrite(pinMotorL2, LOW );
234     goingFront = false;
235     goingBack = true;
236     turning   = false;
237     _move = _BACK;
238     delay(g);
239   } // end of back()
240
241   int ask_pin_F() {                 // Measure the distance from the front
242     delayMicroseconds(2);
243     digitalWrite(outputPinF, HIGH);// Let US TX high voltage 10 Î¼s, where at least 10
          Î¼s
244     delayMicroseconds(10);
245     digitalWrite(outputPinF, LOW );// Maintain low voltage ultrasonic transmitter
246     if (DEBUG5) startTime=millis();
247     float Fdistance = pulseIn(inputPinF, HIGH); // Read worse time difference
248     if (DEBUG5) cronometer(startTime,":F=");
249     Fdistance = Fdistance*0.01724f;// 1/5.8/10=0.01724 Time to turn to the distance (unit
          : cm)
250     _SERIAL1.print("F distance:"); // Output distance (unit: cm)
251     _SERIAL1.println(Fdistance);   // Display the distance
252     Fspace = Fdistance;            // Read into the distance Fspace  (front space)
253     return Fspace;
254   } // end of ask_pin_F()
255
256   int ask_pin_R() {                 // Measure the distance from the right
257     delayMicroseconds(2);
258     digitalWrite(outputPinR, HIGH);// Let US TX high voltage 10 Î¼s, where at least 10
          Î¼s
```

```
259     delayMicroseconds(10);
260     digitalWrite(outputPinR, LOW );// Maintain low voltage ultrasonic transmitter
261     if (DEBUG5) startTime=millis();
262     float Rdistance = pulseIn(inputPinR, HIGH); // Read worse time difference
263     if (DEBUG5) cronometer(startTime,":R=");
264     Rdistance = Rdistance*0.01724f;// 1/5.8/10=0.01724 Time to turn to the distance (unit
            : cm)
265     _SERIAL1.print("R distance:"); // Output distance (unit: cm)
266     _SERIAL1.println(Rdistance);   // Display the distance
267     Rspace = Rdistance;            // Read into the distance Fspace  (front space)
268     return Rspace;
269   } // end of ask_pin_R()
270
271   int ask_pin_L() {                // Measure the distance from the right
272     delayMicroseconds(2);
273     digitalWrite(outputPinL, HIGH);// Let US TX high voltage 10 Î¼s, where at least 10
            Î¼s
274     delayMicroseconds(10);
275     digitalWrite(outputPinL, LOW );// Maintain low voltage ultrasonic transmitter
276     if (DEBUG5) startTime=millis();
277     float Ldistance = pulseIn(inputPinL, HIGH); // Read worse time difference
278     if (DEBUG5) cronometer(startTime,":L=");
279     Ldistance = Ldistance*0.01724f;// 1/5.8/10=0.01724 Time to turn to the distance (unit
            : cm)
280     _SERIAL1.print("L distance:"); // Output distance (unit: cm)
281     _SERIAL1.println(Ldistance);   // Display the distance
282     Lspace = Ldistance;            // Read into the distance Fspace  (front space)
283     return Lspace;
284   } // end of ask_pin_L()
285
286   void frontSpaceDetection() { // Measure three angles (0.90.180) [former detection()]
287     if (goingFront == false) {
288       frontSteps = 0;
289     } else if (frontSteps < SPEED_MAX-SPEED_MED && SPEED_MAX>SPEED_MED) {
290       frontSteps++;
291     }
292     if (DEBUG4) startTime=millis();
293     //if (turning) advance(speedF, DELAY_FRONT/2);
294     Lspace = ask_pin_L();
295     delay(1); // to avoid echoes
296     Fspace = ask_pin_F();
297     delay(1); // to avoid echoes
298     Rspace = ask_pin_R();
299     if (DEBUG4) cronometer(startTime, ":a=");
300     if (DEBUG4) startTime=millis();
301     _SERIAL8.print("frontSpaceDetection: Lspace=");
302     _SERIAL8.print(Lspace, DEC);
303     _SERIAL8.print(" : Fspace=");
304     _SERIAL8.print(Fspace, DEC);
305     _SERIAL8.print(" : Rspace=");
306     _SERIAL8.println(Rspace, DEC);
307     if (DEBUG4) cronometer(startTime, ":l=");
308
```

```
309     if (DEBUG4) startTime=millis();
310     if (Fspace > DIST_MAXIMUM &&
311         Rspace > DIST_MINIMUM &&
312         Lspace > DIST_MINIMUM ) {
313       speedF = SPEED_MED + frontSteps;
314       prevDist = Fspace;
315       advance(speedF, DELAY_FRONT);
316       _SERIAL9.println("No obstacles, advancing at max speed...");
317       if (DEBUG4) cronometer(startTime, ":1=");
318       return;
319     }
320     if (Fspace  < DIST_MINIMUM) {
321       _SERIAL9.println("ALERT!, stopping!");
322       prevDist = Fspace;
323       stopp(DELAY_STOP);       // Clear the output data
324       _SERIAL9.println("Car at front too near! Stopping...");
325       if (DEBUG4) cronometer(startTime, ":2=");
326       return;
327     }
328     if (Rspace  < DIST_WARNING ||
329         Lspace  < DIST_WARNING ) {
330       _SERIAL9.println("ALERT!, stopping!");
331       prevDist = Fspace;
332       stopp(DELAY_STOP);       // Clear the output data
333       _SERIAL9.println("Obstacles too near! Stopping...");
334       if (DEBUG4) cronometer(startTime, ":3=");
335       return;
336     }
337     if (DEBUG4) cronometer(startTime, ":t=");
338     if (DEBUG4) startTime=millis();
339
340     // PID controller
341     if (_move==_TURNR && Rspace<Fspace) {
342       error = DIST_DESIRED - (Fspace+Rspace)/2;
343     } else if (_move==_TURNL && Lspace<Fspace) {
344       error = DIST_DESIRED - (Fspace+Lspace)/2;
345     } else {
346       error = DIST_DESIRED - Fspace;
347     }
348     if (abs(error) < DIST_MAX_ERR){       // prevent integral 'windup'
349        integral = integral + error;       // accumulate the error integral
350     } else {
351       if (error<0) integral -= DIST_MAX_ERR; // accumulate a maximum error
352       else         integral += DIST_MAX_ERR; // accumulate a maximum error
353     }
354     if      (integral> DIST_MAX_INT) integral = +DIST_MAX_INT;
355     else if (integral<-DIST_MAX_INT) integral = -DIST_MAX_INT;
356     signal  = error*KPP;                  // calc proportional term
357     signal += integral*KPI;               // add integral term
358     signal += (prevDist-Fspace)*KPD;      // and add derivative term
359     if      (signal >  MAX_ERROR) signal= MAX_ERROR; // set the maximum signal
360     else if (signal < -MAX_ERROR) signal=-MAX_ERROR; // set the minimum signal
361     _SERIAL9.print("  prevSpeed=");
```

```
362     _SERIAL9.print(speedF, DEC);
363     // Get the new speed
364     speedF = map(signal, +MAX_ERROR, -MAX_ERROR, SPEED_MIN, SPEED_MED + frontSteps);
365     _SERIAL9.print("Control=");
366     _SERIAL9.print(elapsedTime, DEC);
367     _SERIAL9.print("  Fspace=");
368     _SERIAL9.print(Fspace, DEC);
369     _SERIAL9.print("  prevDist=");
370     _SERIAL9.print(prevDist, DEC);
371     _SERIAL9.print("  error=");
372     _SERIAL9.println(error, DEC);
373     _SERIAL9.print("  integral=");
374     _SERIAL9.print(integral, DEC);
375     _SERIAL9.print(" : sKPP=");
376     _SERIAL9.print(error*KPP, DEC);
377     _SERIAL9.print(" : sKPI=");
378     _SERIAL9.print(integral*KPI, DEC);
379     _SERIAL9.print(" : sKPD=");
380     _SERIAL9.println((prevDist-Fspace)*KPD, DEC);
381     _SERIAL9.print("  signal=");
382     _SERIAL9.print(signal, DEC);
383     _SERIAL9.print(" : newSpeed=");
384     _SERIAL9.print(speedF, DEC);
385     _SERIAL9.print(" : elapsedTime=");
386     _SERIAL9.println(elapsedTime, DEC);
387     prevDist = Fspace;                    // save current front space value for next time
388     _SERIAL1.println("Controlled advance...");
389     if (DEBUG4) cronometer(startTime, ":c=");
390     if (DEBUG4) startTime=millis();
391     advance(speedF, DELAY_FRONT);
392     if (DEBUG4) cronometer(startTime, ":m=");
393   } // end of frontSpaceDetection()
394
395   void leftObstacleDetection() {
396     Lspace = ask_pin_L();
397     _SERIAL8.print("leftSpaceDetection : Lspace=");
398     _SERIAL8.println(Lspace, DEC);
399   } // end of leftObstacleDetection()
400
401   void frontObstacleDetection() {
402     Fspace = ask_pin_L();
403     _SERIAL8.print("frontSpaceDetection: Fspace=");
404     _SERIAL8.println(Fspace, DEC);
405   } // end of frontObstacleDetection()
406
407   void rightObstacleDetection() {
408     Rspace = ask_pin_R();
409     _SERIAL8.print("rightSpaceDetection: Rspace=");
410     _SERIAL8.println(Rspace, DEC);
411   } // end of rightObstacleDetection()
412
413   void readIRmodule() {
414     int t=0;
```

```
415     _SERIAL6.println("start of readIRmodule");
416     _SERIAL6.println("requesting data");
417     Wire.requestFrom(IR_MODULE_ADDRESS, 16);    // request 16 bytes from slave device #9
418     _SERIAL6.println("received data");
419     _SERIAL6.println("reading data");
420     while (Wire.available()) {  // slave may send less than requested
421       _SERIAL6.print("  t=");
422       _SERIAL6.println(t, DEC);
423       data[t] = Wire.read(); // receive a byte as character
424       _SERIAL6.print("  data[t]=");
425       _SERIAL6.println(data[t], HEX);
426       if (t < 15)
427         t++;
428       else
429         t = 0;
430     }
431   } // enf of readIRmodule()
432
433   void setup() {
434     Serial.begin (115200); //9600
435
436     // IIC IR tracking module setup
437     Wire.begin();          // join i2c bus (address optional for master)
438
439     // LCD setup (for the YWRobot LCM1602 IIC model)
440     if (DEBUGL0 || DEBUGL1) {
441       lcd.begin(16,2); // sixteen characters across - 2 lines
442       lcd.backlight();
443       lcd.setCursor(0,0);
444     }
445
446     // Define motor output pins
447     pinMode (pinMotorR1, OUTPUT); // Signal 1 right side motor
448     pinMode (pinMotorR2, OUTPUT); // Signal 2 right side motor
449     pinMode (pinSpeedR , OUTPUT); // PWM pin, to control speed right side motor
450     pinMode (pinMotorL1, OUTPUT); // Signal 1 left  side motor
451     pinMode (pinMotorL2, OUTPUT); // Signal 2 left  side motor
452     pinMode (pinSpeedL , OUTPUT); // PWM pin, to control speed left  side motor
453
454     // Ultrasound input/output pins
455     pinMode (inputPinL , INPUT ); // Define ultrasound input  pin (left)  (receive )
456     pinMode (outputPinL, OUTPUT); // Define ultrasonic output pin (left)  (transmit)
457     pinMode (inputPinF , INPUT ); // Define ultrasound input  pin (front) (receive )
458     pinMode (outputPinF, OUTPUT); // Define ultrasonic output pin (front) (transmit)
459     pinMode (inputPinR , INPUT ); // Define ultrasound input  pin (right) (receive )
460     pinMode (outputPinR, OUTPUT); // Define ultrasonic output pin (right) (transmit)
461
462     // Measure free front space distance: averages "MEASURES" values
463     prevDist = ask_pin_F();
464     _SERIAL1.print("prevDist=");
465     _SERIAL1.println(prevDist, DEC);
466
467     //advance(SPEED_MED, DELAY_FRONT);
```

```
468     _SERIAL1.println("Go!");
469   } // end of setup()
470
471   void loop() {
472     if (DEBUG2 || DEBUGL1) {
473       elapsedTimeLoop = millis() - startTimeLoop;
474       startTimeLoop = millis();
475       if (DEBUGL1) {
476         lcd.setCursor(0,0);
477         sprintf(lcdText, "%d-%d-%d-%d v%3dt%3d",SLM,SLH,SRH,SRM,speedF,elapsedTimeLoop);
478         lcd.println(lcdText);
479       } else {
480         _SERIAL2.print("startTimeLoop=");
481         _SERIAL2.print(startTimeLoop, DEC);
482         _SERIAL2.print(" : elapsedTimeLoop=");
483         _SERIAL2.println(elapsedTimeLoop, DEC);
484         static int i=0;
485         LCD0.setCursor(i,1);
486         LCD0.print("    ");
487         LCD0.setCursor(++i,1);
488         LCD0.print(speedF);
489         if(i>12) i=0;
490       }
491     }
492     // Check tracking sensors' status
493     readIRmodule();
494     SLM = data[IR_LEFT_MAX]   < IR_THRESHOLD;
495     SLH = data[IR_LEFT_HALF]  < IR_THRESHOLD;
496     SRH = data[IR_RIGHT_HALF] < IR_THRESHOLD;
497     SRM = data[IR_RIGHT_MAX]  < IR_THRESHOLD;
498
499     if (DEBUGL0) {
500       char text[17];
501       LCD0.setCursor(0,0);
502       sprintf(text, "%3d-%3d-%3d-%3d", data[IR_LEFT_MAX], data[IR_LEFT_HALF],
503               data[IR_RIGHT_HALF], data[IR_RIGHT_MAX]);
504       LCD0.print(text);
505       LCD0.setCursor(0,1);
506       sprintf(text, " %d - %d - %d - %d", SLM, SLH, SRH, SRM);
507       LCD0.print(text);
508     }
509     if (DEBUG7) {
510       char text[128];
511       sprintf(text, "(L/M/R) : dataSensors[] = %3d %3d %3d %3d :"
512               " result = %d / %d / %d / %d",
513               data[IR_LEFT_MAX], data[IR_LEFT_HALF], data[IR_RIGHT_HALF],
514               data[IR_RIGHT_MAX], SLM, SLH, SRH, SRM);
515       Serial.println(text);
516     }
517     // Goes back if there is no line under the car, looking for the lost line
518     // Doesn't goes forward till it detects STEPS_BACK_MIN times in a row a line under it
519     if (goingBack) {
520       if (SLM + SLH + SRH + SRM < 2) {
```

```
521         backSteps++;
522         if (backSteps == STEPS_BACK_MIN) {
523           goingBack = false;
524           backSteps = 0;
525         }
526       } else {
527         backSteps = 0;
528       }
529     }
530     switch ((SLM<<3) + (SLH<<2) + (SRH<<1) + SRM) {
531       case  0: // 0b0000: all them white, go back trying to recover the black line
532         frontSteps = 0;
533         if (DEBUG0) {
534           speedF = SPEED_LOW;
535           back(speedF, DELAY_BACK);
536         } else {
537           _SERIAL1.println("WHITE AREA: all them are white, "
538                            "go back trying to recover the black line");
539           if (DEBUG3) startTime=millis();
540           speedF = SPEED_LOW;
541           back(speedF, DELAY_BACK);
542           if (DEBUG3) {
543             cronometer(startTime, ":b=");
544           }
545         }
546         break;
547       case 15: // 0b1111: all them black: go front at max speed
548         if (DEBUG0) {
549           speedF = SPEED_MED + frontSteps;
550           advance(speedF, DELAY_FRONT);
551         } else {
552           _SERIAL1.println("BLACK AREA: All sensors black, straight");
553           if (DEBUG3) startTime=millis();
554           frontSpaceDetection(); // includes advance(), or stopp() if finds obstacles
555           if (DEBUG3) {
556             cronometer(startTime, ":f=");
557           }
558         }
559         break;
560       case 10: // 0b1010
561       case  9: // 0b1001
562       case  6: // 0b0110: In this case, no caution :)
563       case  5: // 0b0101
564         // 2 whites and 2 blacks, one on each wing: go stright by with caution
565         if (speedF>SPEED_HIGH && !SLH && !SRH) speedF = SPEED_HIGH;
566         if (DEBUG0) {
567           advance(speedF, DELAY_FRONT);
568         } else {
569           _SERIAL1.println("BLACK AREA: Two sensors black, one in each wing, straight
                with caution");
570           if (DEBUG3) startTime=millis();
571           frontSpaceDetection();     // includes advance(), or stopp() if finds obstacles
572           if (DEBUG3) {
```

```
573             cronometer(startTime, ":f=");
574           }
575         }
576         break;
577       case 14: // 0b1110
578       case 13: // 0b1101
579       case  4: // 0b0100
580         // one more black (non-extrem) in the left side than in the right one: move left
581         if (speedF>SPEED_MED) speedF = SPEED_MED;
582         if (DEBUG0) {
583           left(speedF, DELAY_WING);
584         } else {
585           if (DEBUG3) startTime=millis();
586           leftObstacleDetection();
587           frontObstacleDetection();
588           if (DEBUG3) {
589             cronometer(startTime, ":l=");
590           }
591           if (Lspace < DIST_WARNING || Fspace < DIST_MINIMUM) {
592             stopp(DELAY_STOP);       // Clear the output data
593           } else {
594             left(speedF, DELAY_WING);
595           }
596           if (DEBUG3) {
597             cronometer(startTime, "l0=");
598           }
599         }
600         break;
601       case  2: // 0b0010
602       case  7: // 0b0111
603       case 11: // 0b1011
604         // one more black (non-extrem) in the right side than in the left one: move right
605         if (speedF>SPEED_MED) speedF = SPEED_MED;
606         if (DEBUG0) {
607           right(speedF, DELAY_WING);
608         } else {
609           if (DEBUG3) startTime=millis();
610           rightObstacleDetection();
611           frontObstacleDetection();
612           if (DEBUG3) {
613             cronometer(startTime, ":l=");
614           }
615           if (Rspace < DIST_WARNING || Fspace < DIST_MINIMUM) {
616             stopp(DELAY_STOP);       // Clear the output data
617           } else {
618             right(speedF, DELAY_WING);
619           }
620           if (DEBUG3) {
621             cronometer(startTime, "l0=");
622           }
623         }
624         break;
625       case  8: // 0b1000
```

```
626       case 12: // 0b1100
627         // bias to left  with the extreme black: turn (rotate) left
628         if (speedF>SPEED_LOW) speedF = SPEED_LOW;
629         if (DEBUG0) {
630           turnL(speedF, DELAY_TURN);
631         } else {
632           _SERIAL1.println("WHITE AREA: left black and right white, turn left");
633           if (DEBUG3) startTime=millis();
634           leftObstacleDetection();
635           if (DEBUG3) {
636             cronometer(startTime, ":l=");
637           }
638           if (Lspace < DIST_WARNING) {
639             stopp(DELAY_STOP);       // Clear the output data
640           } else {
641             turnL(speedF, DELAY_TURN);
642           }
643           if (DEBUG3) {
644             cronometer(startTime, "l0=");
645           }
646         }
647         break;
648       case  1: // 0b0001
649       case  3: // 0b0011
650         // bias to right with the extreme black: turn (rotate) right
651         if (speedF>SPEED_LOW) speedF = SPEED_LOW;
652         if (DEBUG0) {
653           turnR(speedF, DELAY_TURN);
654         } else {
655           _SERIAL1.println("WHITE AREA: left black and right white, turn right");
656           if (DEBUG3) startTime=millis();
657           rightObstacleDetection();
658           if (DEBUG3) {
659             cronometer(startTime, ":l=");
660           }
661           if (Rspace < DIST_WARNING) {
662             stopp(DELAY_STOP);       // Clear the output data
663           } else {
664             turnR(speedF, DELAY_TURN);
665           }
666           if (DEBUG3) {
667             cronometer(startTime, "l0=");
668           }
669         }
670         break;
671       default:
672         // not possible, all 16 values, from 0 to 15, taken into account
673         break;
674     }
675   } // end of loop()
```

Listado D.3: Fichero de cabecera con la definición de enumeradores (moveDirections.h)

```
#ifndef _MOVEDIRECTIONS
#define _MOVEDIRECTIONS

typedef enum Move_t {_STOP =0 , _BACK=1, _TURNL=2, _LEFT=3,
                     _FRONT=4, _RIGHT=5, _TURNR=6} move_t;
char moves[7][6] = {"STOP", "BACK", "TURNL", "LEFT", "FRONT", "RIGHT", "TURNR"};

#endif
```

GLOSARIO

breadboard En inglés, placa base de prototipado. Es una placa de pruebas de uso provisional consistente en tablero con orificios conectados eléctricamente entre sí de manera interna, habitualmente siguiendo patrones de líneas, en el cual se pueden insertar componentes electrónicos y cables para el armado y prototipado de circuitos electrónicos y sistemas similares.

car recall En inglés, llamadas generales a la reparación. Son realizadas por los fabricantes, en este caso, de automóviles, para solucionar defectos de fabricación, independientemente del estado de garantía del vehículo. Existe normativa que obliga a los fabricantes a realizarlas cuando afectan a sistemas críticos del vehículo que pueden incidir en la seguridad vial, en la emisión de gases contaminantes, cumplimiento normativo, etc.

common commodity En inglés, bien de consumo comúnmente comerciado. También es frecuente el término *commonly traded commodity*. Son aquellos bienes de consumo con gran disponibilidad comercial, y cuya compraventa es común, frecuente y a gran escala.

controlador PID Mecanismo de control por realimentación de una señal obtenida computacionalmente a partir de los datos leídos del sistema real a controlar. La señal está basada generalmente en la diferencia existente entre el valor medido y su valor deseado, que se le denomina "error". En un controlador PID hay tres términos que se conjugan para realizar el cálculo: la ganancia (o parte proporcional), la parte integral y la derivativa.

driver En inglés, controlador de dispositivo, generalmente software. Programa de bajo nivel que permite al sistema operativo interactuar directamente con el hardware, generalmente a través de su firmware.

firmware Programa informático grabado en memorias especiales de un dispositivo o circuito electrónico, y que se encarga de su lógica de más bajo nivel, gestionando físicamente dicho hardware.

framework En inglés, infraestructura software. Consiste en una estructura conceptual y tecnológica, así como una especial metodología del trabajo, aplicable al dominio para el que está diseñado. Proporciona herramientas comunes básicas, tales como artefactos y módulos concretos del software, incluyendo a veces un propio lenguaje interpretado, para un desarrollo más rápido, estandarizado y seguro.

jumper En inglés, puente. Elemento que permite abrir, cerrar o cortocircuitar físicamente a voluntad un circuito eléctrico, o parte de él, al que pertenecen los dos conectores que se puentean, al objeto de hacer variar el comportamiento de dicho circuito eléctrico de una forma conocida y prefijada, permitiendo el intercambio entre diferentes configuraciones o modos de funcionamiento, por ejemplo.

just-in-time manufacturing En inglés, fabricación justo a tiempo. Es una metodología de organización de la producción por el que sistema tiende a producir justo lo que se requiere, cuando se necesita, con excelente calidad y sin desperdiciar recursos del sistema.

motor de corriente continua Máquina que convierte la energía eléctrica proporcionada por una corriente eléctrica contínua en energía mecánica, provocando un movimiento rotatorio gracias a la acción que se genera del campo magnético.

movimiento browniano Movimiento aleatorio estocástico que se observa en algunas partículas microscópicas que se hallan en un medio fluido, sin razón aparente para ello.

offset En inglés, desplazamiento. En un sistema de control, es la diferencia existente entre el valor de entrada y la señal de control generada en el estado estable, no durante el proceso dinámico de control, esto es, un error que el controlador es incapaz de corregir con el tiempo.

open hardware En inglés, dispositivo abierto, o dispositivo libre. Dispositivos de hardware cuyas especificaciones y diagramas esquemáticos son de acceso público, ya sea bajo algún tipo de pago, o de forma gratuita. Es un concepto más reciente que el software libre, pero íntimamente relacionado con él.

pin En inglés, terminal o patilla de cada uno de los contactos metálicos de un conector o de un componente fabricado de un material conductor de la electricidad. Se utilizan para conectar componentes sin necesidad de soldar nada, de esta manera se logra transferir electricidad e información. Generalmente por pin se entiende el terminal macho fijo, y es el tipo más frecuente de contacto en el hardware Arduino, en el que generalmente están separados entre sí 2,54 mm (0,10 pulgadas).

ransomware En inglés, software que reclama rescate. Es una amenaza informática consistente en la ejecución de un código malintencionado que restringe el acceso a partes o archivos del sistema infectado, exigiendo un rescate a cambio de liberar la restricción. Esa restricción puede ser, entre otras, por ocultamiento, por cifrado o por robo hacia el exterior de la información.

resistencia de pull-up Resistencia especialmente destinada a elevar la tensión de salida de un circuito lógico, a la tensión que requiere algún componente electrónico. Se utiliza mucho en los buses. En Arduino el *pull-up* se suele elevar a la tensión de trabajo de los pines, que normalemente es de 5 voltios.

royalty En inglés, regalía de patentes. Consiste en el pago de una cantidad variable ligada al volumen de producción o ventas que debe abonarse durante un tiempo al propietario de la patente que se esté explotando.

shield En inglés, placa de extensión. Dispositivo que se acopla a una placa base de Arduino, o a otra *shield* de Arduino, incluso permitiendo su apilamiento sucesivo, para añadirle funcionalidad, y generalmente facilitando el acceso al resto de conexiones de la placa base.

survey En inglés, estudio resumen. Es un trabajo académico de análisis y recopilación del estado actual del conocimiento en un área determinada.

SIGLAS

ABS *Anti-lock Braking System*, Sistema antibloqueo de ruedas.

ACC *Adaptive Cruise Control*, también *Autonomous Cruise Control*, Control de velocidad adaptativo.

ADAS *Advanced Driver-Assistant System*, Sistema avanzado de asistencia al conductor.

AEB *Autonomous Emergency Braking*, Frenado de emergencia autónomo.

Auto-ISAC *Automotive Information Sharing and Analysis Center* (consorcio internacional): Centro de intercambio y análisis de información sobre automoción.

BPMN *Business Process Model and Notation*, Modelo y notación de procesos de negocio.

CACC *Cooperative Adaptive Cruise Control*, Control cooperativo y adaptativo de velocidad (varios automóviles cercanos cooperando en la navegación).

CAD Conversor analógico a digital.

CCC *Conventional Cruise Control*, Control convencional de la velocidad (de crucero).

CiA *CAN (Controller Area Network) in Automation*, CAN (controlador de red de área) en automatización.

CLMR *Car-Like Mobile Robot*, Robot móvil semejante a un coche.

DC *Direct current*, corriente continua.

ECU *Electronic control unit*, Unidad de control electrónico, o simplemente centralita electrónica.

ERTRAC *European Road Transport Research Advisory Council*, Consejo europeo asesor de investigación del transporte por carretera.

ESP *Electronic stability control*, Control electrónico de estabilidad.

FCW *Forward Collision Warning*, Aviso de colisión frontal.

HTSM *High Tech Systems and Materials*, Sistemas y materiales de alta tecnología (centro de investigación danés).

HVA *Highly Automated Vehicle*, Vehículo altamente automatizado.

IDE *Integrated development environment*, Entorno de desarrollo integrado.

IR Infrarrojo(s).

ISA *Intelligent Speed Assistance*, Asistencia inteligente de velocidad.

ITACA Instituto de Aplicaciones de las Tecnologías de la Información y de las Comunicaciones Avanzadas.

LCA *Lane Change Assist*, Ayuda para el cambio de carril.

LDW *Lane Departure Warning*, Aviso de pérdida del carril.

LIDAR *LIght Detection And Ranging* o también *Laser Imaging Detection and Ranging*, es una técnica basada en la reflexión de un haz de luz láser que se utiliza para detectar objetos, obteniendo una nube de datos a través de la cual se puede estimar tanto su forma como su distancia.

LKA *Lane Keeping Assist*, Ayuda para el seguimiento del carril.

LRR *Long Range Radar*, Radar de largo alcance.

NHTSA *National Highway Traffic Safety Administration*, Administración nacional de seguridad del tráfico en las carreteras (Estados Unidos de América).

PCB *Printed Circuit Board*, placa de circuito impreso.

PFC Proyecto fin de carrera.

PWM *Pulse Width Modulation*, modulación por ancho de pulsos.

SAE *Society of Automotive Engineers*, Sociedad de ingenieros de la automoción.

STF Sistema tolerante a fallos.

TSR *Traffic Sign Recognition*, Reconocimiento de las señales de tráfico.

UAV *Unmanned Aerial Vehicle*, vehículo aéreo no tripulado.

US Ultrasonido(s).

BIBLIOGRAFÍA

[1] [Alliance of Automobile Manufacturers & Association of Global Automakers]: *Framework for automotive cybersecurity best practices*, Ene. 2016. http://www.autoalliance.org/download.cfm?downloadfile=1E518FB0-BEC3-11E5-9500000C296BA163&typename=dmFile&fieldname=filename, visitado el 08/09/2016.

[2] [Anaheim]: *Adaptive Cruise Control system overview.* En *5th Meeting of the U.S. Software System Safety Working Group*, Abr. 2005.

[3] J. M. Anderson, K. Nidhi, K. D. Stanley, P. Sorensen, C. Samaras y O. A. Oluwatola: *Autonomous vehicle technology: A guide for policymakers.* Rand Corporation, 2014.

[4] [Arduino LLC]: *Arduino.cc.* https://www.arduino.cc, visitado el 23/09/2016 , *Nota: tiene registrado el nombre Arduino en los EE.UU. y en el resto del mundo comercializa con el nombre Genuino, está formada por la mayoría de los fundadores del proyecto Arduino inicial.*

[5] [Arduino LLC]: *Intel Galileo.* https://www.arduino.cc/en/ArduinoCertified/IntelGalileo, visitado el 29/09/2016.

[6] [Arduino SRL]: *Arduino.org.* https://www.arduino.org, visitado el 23/09/2016, *Nota: sede en Italia, tiene registrado el nombre Arduino en todo el mundo, excepto en los EE.UU. y están especializados en fabricación.*

[7] S. Asaduzzaman: *Analysis of automotive cybersecurity. Attack surfaces and users' privacy concerns.* Trabajo Fin de Máster (*Master's Thesis*), University of Turku, Finlandia, Abr. 2016.

[8] [Auto-ISAC]: *Automotive cybersecurity best practices. Executive summary*, Jul 2016. https://automotiveisac.com/assets/img/executive-summary.pdf, visitado el 08/09/2016.

[9] [AUTOSAR]: *AUTOSAR release 4.2.2: Overview and revision history*, Jul 2015. http://www.autosar.org/specifications/release-42/, visitado el 25/09/2016.

[10] H. Barragán: *The untold history of Arduino.* http://arduinohistory.github.io/, visitado el 23/09/2016.

[11] H. Barragán: *Wiring: What will you do with the W?* http://wiring.org.co/, visitado el 23/09/2016.

[12] H. Barragán: *Wiring: Prototyping physical interaction design.* Tésis, Interaction Design Institute Ivrea, Ivrea, Italy, Jun 2004.

[13] [BMW AG]: *Active Cruise Control with Stop&Go function.* http://www.bmw.com/com/en/insights/technology/technology_guide/articles/active_cruise_control_stop_go.html, visitado el 21/09/2016.

[14] M. Broy, I. H. Kruger, A. Pretschner y C. Salzmann: *Engineering automotive software.* PROCEEDINGS-IEEE, 95(2):356, 2007.

[15] Y. L. Chen, Y. H. Chen, C. J. Chen y B. F. Wu: *Nighttime vehicle detection for driver assistance and autonomous vehicles.* En *18th International Conference on Pattern Recognition (ICPR'06)*, vol. 1, págs. 687–690. IEEE, 2006.

[16] S. Cobb: *Jackware: When connected cars meet ransomware*, Jul 2016. http://www.welivesecurity.com/2016/07/20/jackware-connected-cars-meet-ransomware/, visitado el 27/09/2016.

[17] [Daimler AG (Mercedes Benz)]: *Mercedes-Benz TechCenter: Lane keeping assist*, Nov. 2015. http://techcenter.mercedes-benz.com/en/lane_keeping_assist/detail.html, visitado el 21/09/2016.

[18] [DFRobot]: *DIY remote control robot kit: User manual v1.0*, 2016. http://7326097.s21d-7.faiusrd.com/0/ABUIABA9GAAg1LbPtgUop5OMiwQ?f=Bluetooth+Multi-functional+smart+car+manual+for+arduino.pdf&v=1456724820, visitado el 27/09/2016.

[19] [elconfidencial.com]: *ProPILOT, la conducción autónoma de Nissan*, Jul 2016. http://www.elconfidencial.com/motor/2016-07-27/propilot-conduccion-autonoma-nissan_1238331/, visitado el 08/09/2016.

[20] [Friends-of-Fritzing Foundation]: *Fritzing Fritzing: electronics made easy.* http://fritzing.org/, visitado el 27/09/2016.

[21] R. Frisoni, A. Dall'Oglio, C. Nelson, J. Long, C. Vollath, D. Ranghetti y S. McMinimy: *Self-piloted cars: The future of road transport?* The European Commission (TRAN committee), Mar. 2016. http://www.europarl.europa.eu/RegData/etudes/STUD/2016/573434/IPOL_STU(2016)573434_EN.pdf, visitado el 29/09/2016.

[22] [General Motors Co.]: *2014: General Motors annual peport*, Feb. 2015. https://www.gm.com/content/dam/gm/en_us/english/Group4/InvestorsPDFDocuments/2014_GM_Annual_Report.pdf, visitado el 29/09/2016.

[23] C. Goerzen, Z. Kong y B. Mettler: *A survey of motion planning algorithms from the perspective of autonomous UAV guidance.* Journal of Intelligent and Robotic Systems, 57(1-4):65–100, 2010.

[24] [High Tech Systemen & Materialen (HTSM)]: *HTSM roadmap automotive 2016–2020*, Nov. 2015. http://www.rvo.nl/sites/default/files/2015/11/HTSM_Roadmap_Automotive_2016-2020.pdf, visitado el 09/09/2016.

[25] K. E. Hizon: *TINKBOX: Robotics training manual*, 2016. http://tinkbox.ph/sites/tinkbox.ph/files/downloads/Tinkbox%20Robotics%20Training%20Manual.pdf, visitado el 08/09/2016.

[26] [Intel Corporation]: *Galileo datasheet.* http://www.intel.com/newsroom/kits/quark/galileo/pdfs/Intel_Galileo_Datasheet.pdf, visitado el 29/09/2016.

[27] K. Jo, J. Kim, D. Kim, C. Jang y M. Sunwoo: *Development of autonomous car — Part I: Distributed system architecture and development process.* IEEE Transactions on Industrial Electronics, 61(12):7131–7140, 2014.

[28] K. Jo, J. Kim, D. Kim, C. Jang y M. Sunwoo: *Development of autonomous car — Part II: A case study on the implementation of an autonomous driving system based on distributed architecture.* IEEE Transactions on Industrial Electronics, 62(8):5119–5132, 2015.

[29] A. Kleyner y P. Sandborn: *Minimizing life cycle cost by managing product reliability via validation plan and warranty return cost.* International journal of production economics, 112(2):796–807, 2008.

[30] F. Lambert: *Tesla Autopilot 2.0: next gen Autopilot powered by more radar, new triple camera, some equipment already in production*, Ago. 2016. https://electrek.co/2016/08/11/tesla-autopilot-2-0-next-gen-radar-triple-camera-production/, visitado el 21/09/2016.

[31] T. H. Li y S. J. Chang: *Autonomous fuzzy parking control of a car-like mobile robot.* IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 33(4):451–465, 2003.

[32] F. M. López-Rodríguez y F. Cuesta: *Andruino-A1: low-cost educational mobile robot based on Android and Arduino.* Journal of Intelligent & Robotic Systems, 81(1):63–76, 2016.

[33] [Mercedes 500SEC.com]: *Distronic/Distronic PLUS*, Nov. 2015. http://500sec.com/distronicdistronic-plus/, visitado el 21/09/2016.

[34] W. C. Mitchell, A. Staniforth y I. Scott: *Analysis of Ackermann steering geometry.* Informe técnico., SAE Technical Paper, 2006.

[35] A. Mogelmose, M. M. Trivedi y T. B. Moeslund: *Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey.* IEEE Transactions on Intelligent Transportation Systems, 13(4):1484–1497, 2012.

[36] R. Müller-Fiedler y V. Knoblauch: *Reliability aspects of microsensors and micromechatronic actuators for automotive applications.* Microelectronics Reliability, 43(7):1085–1097, 2003.

[37] [National Highway Traffic Safety Administration (NHTSA)]: *Federal Automated Vehicles Policy. Accelerating the next revolution in roadway safety. Executive summary*, Sep. 2016. http://www.nhtsa.gov/nhtsa/av/pdf/Federal_Automated_Vehicles_Policy.pdf, visitado el 26/09/2016.

[38] T. Navarro Cosme: *Control de velocidad adaptativo (Adaptive Cruise Control) con SunFounder y 4WD*, Sep. 2016. https://youtu.be/_TQQ7HCs4M4, visitado el 29/09/2016.

[39] T. Navarro Cosme: *Evitación de obstáculos SunFounder sensores todo/nada algoritmo original*, Sep. 2016. https://youtu.be/GAtDEfyyeR4, visitado el 29/09/2016.

[40] T. Navarro Cosme: *Evitación obstáculos 4WD con único sensor móvil analógico por ultrasonidos*, Sep. 2016. https://youtu.be/3lcDnCP1lYA, visitado el 29/09/2016.

[41] T. Navarro Cosme: *Evitación obstáculos SunFounder con sensores binarios todo/nada. Algoritmo propio*, Sep. 2016. https://youtu.be/j_oXdmjQ268, visitado el 29/09/2016.

[42] T. Navarro Cosme: *Seguimiento de luz por el SunFounder (autocalibración, cambios bruscos iluminación)*, Sep. 2016. https://youtu.be/YKbGQjbif8U, visitado el 29/09/2016.

[43] T. Navarro Cosme: *Seguimiento de línea por el 4WD en circuito cerrado muy sinuoso (prueba 1)*, Sep. 2016. https://www.youtube.com/watch?v=u-5HUbMP2Kg&start=50, visitado el 29/09/2016.

[44] T. Navarro Cosme: *Seguimiento de línea por el 4WD en circuito cerrado muy sinuoso (prueba 2)*, Sep. 2016. https://youtu.be/xU5YbRDs1lA, visitado el 29/09/2016.

[45] T. Navarro Cosme: *Seguimiento de línea por el SunFounder en circuito cerrado (prueba 1)*, Sep. 2016. https://youtu.be/dQeV7wuW0k4&start=15, visitado el 29/09/2016.

[46] T. Navarro Cosme: *Seguimiento de línea por el SunFounder en circuito cerrado (prueba 2)*, Sep. 2016. https://youtu.be/LcfkrcTfUDE&start=, visitado el 29/09/2016.

[47] T. Navarro Cosme: *Seguimiento de línea.SunFounder+ArduinoMega2560+MegaShieldv2.0*, Sep. 2016. https://www.youtube.com/watch?v=VXMBFm3JHgY, visitado el 29/09/2016.

[48] [NHTSA]: *Vehicle safety: Recalls & defects*, 2016. http://www.nhtsa.gov/Vehicle+Safety/Recalls+&+Defects, visitado el 28/09/2016.

[49] [Nissan Motor Co. LTD.]: *Serena ProPILOT Tech makes autonomous Japan first*, Jul 2016. http://reports.nissan-global.com/EN/?p=17746, visitado el 08/09/2016.

[50] C. Nowakowski, S. E. Shladover, X. Y. Lu, D. Thompson y A. Kailas: *Cooperative adaptive cruise control (CACC) for truck platooning: Operational concept alternatives*, Mar. 2015.

[51] R. Okuda, Y. Kajiwara y K. Terashima: *A survey of technical trend of ADAS and autonomous driving*. En *Proceedings of Technical Program-2014 International Symposium on VLSI Technology, Systems and Application (VLSI-TSA)*, págs. 1–4. IEEE, 2014.

[52] OSEK/VDX: *OSEK/VDX. Operating system specification 2.2.3*, Feb. 2005. http://www.irisa.fr/alf/downloads/puaut/TPNXT/images/os223.pdf, visitado el 25/09/2016.

[53] T. Pilutti y A. G. Ulsoy: *Identification of driver state for lane-keeping tasks*. IEEE transactions on systems, man, and cybernetics-Part A: Systems and humans, 29(5):486–502, 1999.

[54] J. Ploeg, B. T. Scheepers, E. van Nunen, N. van de Wouw y H. Nijmeijer: *Design and experimental evaluation of cooperative adaptive cruise control*. En *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, págs. 260–265. IEEE, 2011.

[55] U. Pohlmann, M. Meyer, A. Dann y C. Brink: *Viewpoints and views in hardware platform modeling for safe deployment*. En *Proceedings of the 2nd Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling*, pág. 23. ACM, 2014.

[56] C. Pradalier, J. Hermosillo, C. Koike, C. Braillon, P. Bessière y C. Laugier: *The CyCab: a car-like robot navigating autonomously and safely among pedestrians*. Robotics and Autonomous Systems, 50(1):51–67, 2005.

[57] A. Pretschner, M. Broy, I. H. Kruger y T. Stauner: *Software engineering for automotive systems: A roadmap*. En *2007 Future of Software Engineering*, págs. 55–71. IEEE Computer Society, 2007.

[58] [Processing Foundation]: *Processing*. https://processing.org/reference/, visitado el 23/09/2016.

[59] [Processing Foundation]: *Processing: Environment IDE*. https://processing.org/reference/environment/, visitado el 23/09/2016.

[60] T. Schumann: *Standardization in automotive industry*. Proceedings of the 13th International CAN Conference iCC 2012, págs. 07–10, Mar. 2012.

[61] A. Sharmin y R. Wan: *An autonomous Lane-Keeping ground vehicle control system for highway drive*. En *9th International Conference on Robotic, Vision, Signal Processing and Power Applications*, págs. 351–361, Sep. 2016.

[62] [Shenzhen KEYES DIY Robot Co., Ltd]: *Bluetooth multi-functional smart car manual for Arduino (Robotale)*, 2016. http://7326097.s21d-7.faiusrd.com/0/ABUIABA9GAAg1LbPtgUop5OMiwQ?f=Bluetooth+Multi-functional+smart+car+manual+for+arduino.pdf&v=1456724820, visitado el 27/09/2016.

[63] V. Sivaji y M. Sailaja: *Adaptive Cruise Control systems for vehicle modeling using stop and go manoeuvres*. International Journal of Engineering Research and Applications (IJERA), ISSN, págs. 2248–9622, 2013.

[64] I. Skog y P. Handel: *In-car positioning and navigation technologies – A survey*. IEEE Transactions on Intelligent Transportation Systems, 10(1):4–21, 2009.

[65] Z. Sun, G. Bebis y R. Miller: *On-road vehicle detection: A review*. IEEE transactions on pattern analysis and machine intelligence, 28(5):694–711, 2006.

[66] [SunFounder]: *Smart car kit for Arduino: Tutorial*, 2016. `https://www.sunfounder.com/learn/category/Smart-Car-Kit-for-Arduino.html`, visitado el 27/09/2016.

[67] J. Thalen: *ADAS for the car of the future. Interface concepts for advanced driver assistant systems in a sustainable mobility concept of 2020*. Proyecto Fin de Carrera (*Bachelor Assignment*), University of Twente, Países Bajos, Jun 2006.

[68] J. Watson y G. Castro: *High-temperature electronics pose design and reliability challenges*. Analog Dialogue, 46(2):3–9, 2012.

[69] [Wikipedia Foundation]: *Exponential moving average*, Ago. 2016. `https://en.wikipedia.org/wiki/Moving_average#Exponential_moving_average`, visitado el 16/09/2016.

[70] [Wikipedia Foundation]: *Holonomic (robotics)*, Jul 2016. `https://en.wikipedia.org/wiki/Holonomic_(robotics)`, visitado el 24/09/2016.

[71] [Wikipedia Foundation]: *Nonholonomic system*, Sep. 2016. `https://en.wikipedia.org/wiki/Nonholonomic_system`, visitado el 24/09/2016.

[72] [Wikipedia Foundation]: *Ziegler–Nichols method*, Jul 2016. `https://en.wikipedia.org/wiki/Ziegler%E2%80%93Nichols_method`, visitado el 25/09/2016.

[73] S. J. Wu, H. H. Chiang, J. W. Perng, C. J. Chen, B. F. Wu y T. T. Lee: *The heterogeneous systems integration design and implementation for lane keeping on a vehicle*. IEEE Transactions on Intelligent Transportation Systems, 9(2):246–263, 2008.

[74] L. Xiao y F. Gao: *A comprehensive review of the development of adaptive cruise control systems*. Vehicle System Dynamics, 48(10):1167–1192, 2010.

[75] L. R. Yoshioka, M. C. Oliveira, C. L. Marte, C. F. Fontana, C. A. Sakurai y E. T. Yano: *Framework for designing automotive embedded systems based on reuse approach*. International Journal Systems Applications, Engineering & Development, 8:9–17, 2014.

[76] W. Yue, G. Guo, L. Wang y W. Wang: *Nonlinear platoon control of Arduino cars with range-limited sensors*. International Journal of Control, 88(5):1037–1050, 2015.

[77] Z. Zhou: *Modeling and verification of naturalistic Lane Keeping System*. Tésis, Texas A&M University, Texas, EE.UU., 2016.