Universidad de Córdoba
Instituto de Estudios de Postgrado

Trabajo Fin de Máster
Máster en Sistemas Inteligentes

Sistema de guiado automático para robot autónomo móvil basado en visión por computador y lógica difusa

---

Autor:
D. Daniel Pérez Rodriguez

Tutor:
Dr. D. Ángel Carmona Poyato

Diciembre, 2014

Este documento ha sido generado con LaTeX.

Para cualquier consulta, errata o sugerencia, por favor no dude en contactar con el autor en:

daniel@rtdevelopments.es | *z32perod@uco.es*

El autor quisiera recalcar que este documento se ha realizado con fines meramente didácticos y/o educativos y sin ánimo de lucro por lo tanto declina cualquier responsabilidad sobre el uso de sus contenidos.

El código fuente generado para el presente trabajo está disponible para su uso, estudio y modificación, bajo licencia MIT.

`https://github.com/rtdevelopments/ardubot.git`

Dedicado

A mi padre, por enseñarme el significado de la palabra valor.

AGRADECIMIENTOS

En primer lugar, me gustaría agradecer sinceramente al Dr. D. Ángel Carmona Poyato por su orientación y ayuda en el desarrollo del presente Trabajo Fin de Máster.

Mencionar sin ninguna duda el apoyo y conocimientos recibidos de los profesores del Master en Sistemas Inteligentes de la Universidad de Córdoba, ya que muchos de los conocimientos adquiridos en el desarrollo del Máster han podido ser plasmados en este trabajo.

Gracias también al resto de alumnos y compañeros del Máster, Miguel Ángel, Jacobo, Francisco y José Luis, que hicieron que las duras sesiones de los viernes y sábados de este curso pasado fueran mucho más amenas e interesantes.

Sin duda no se me ocurre mejor forma de casi acabar esta página de agradecimientos que dándole las gracias a mi familia, a mis padres y a mi hermano, ... porque siempre, siempre, están ahí.

El último gesto de agradecimiento no podría ser para otra persona que para mi ya futura esposa, María del Carmen. Así que Mari, aprovecho este pequeño párrafo para ir ensayando como decirte lo importante que eres para mi y que los proyectos futuros sí que van a ser maravillosos.

PREFACIO

En la página oficial del Máster, se define la Inteligencia Artificial (IA) como la disciplina que se ocupa de la investigación básica en la implementación de cada una de las habilidades relacionadas con la inteligencia humana. Así, los Sistemas Inteligentes se encargan de aplicar la investigación, tratando de solucionar problemas ya existentes o mejorar nuestra forma de trabajar o calidad de vida aplicando técnicas de Inteligencia Artificial.

Este Trabajo Fin de Máster (TFM) es sin duda un poquito de todo eso, ya que trata de aplicar dichas técnicas y metodologías a un problema universal del ser humano, como es el transporte, en concreto en la conducción de vehículos, un campo de trabajo que podría facilitar enormemente la calidad de vida de las personas, usándose como sistemas de asistencia a la conducción o incluso disminuir el número de accidentes y muertes en accidentes de tráfico.

Así este trabajo, es una pequeña aportación en este apasionante y creciente campo que es el de la conducción autónoma, en el que tanto universidades de prestigio y las más importantes compañías de la industria de la automoción están invirtiendo actualmente grandes cantidades de recursos y obteniendo resultados realmente sorprendentes.

Por ello, el primer capítulo de la memoria se dedica a describir la problemática actual en la conducción humana de vehículos y la evolución a los sistemas autónomos, realizando una breve introducción histórica al problema y describiendo los principios robóticos básicos que permiten la conducción autónoma.

El segundo capítulo se dedica a realizar una panorámica sobre el estado del arte de los dispositivos de asistencia a la conducción así como en el campo de la conducción autónoma.

El tercer capítulo recoge de forma específica los objetivos que fueron marcados para el desarrollo del trabajo fin de máster.

El cuarto capítulo realiza una descripción completa del diseño y funcionamiento del sistema de guiado automático desarrollado, recogiendo los principios teóricos en los que se basa el diseño así como las distintas técnicas y librerías software utilizadas.

El quinto capítulo recoge los principales resultados obtenidos tanto en los experimentos realizados como durante el resto del desarrollo del TFM,

mientras que el sexto capítulo recoge las principales conclusiones extraídas del trabajo así como las futuras líneas de investigación que pueden surgir a raíz del presente TFM.

Índice general

Índice de figuras

Índice de tablas

I'm really looking forward to a time when generations after us look back and say how ridiculous it was that humans were driving cars.

Sebastian Thrun.
Research professor at Stanford and a co-founder and CEO of Udacity.

Capítulo 1

Introducción

1.1. De la conducción humana a la conducción autónoma

La historia del automóvil puede ser dividido en distintas épocas, basadas en los medios prevalentes de propulsión. En 1769, el primer automóvil a vapor capaz de transportar seres humanos fue construido por Nicolas Joseph-Cugnot [21].

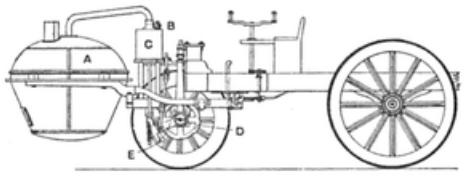

Figura 1.1: Diseño del carrito de vapor Cugnot.

En 1807, François Isaac de Rivaz diseñó el primer coche propulsado por un motor de combustión interna alimentado por hidrógeno y en 1886 Karl Benz inventó el que es considerado primer automóvil con motor a gasolina, el Benz Patent-Motorwagen. A finales del siglo 20 aparecieron los primeros diseños funcionales de automóviles eléctricos, aunque su desarrollo comercial aún es leve, comparado con los motores de combustión, y se espera que aumenten avanzado el siglo 21, aunque destacar también que sí han tenido un mayor éxito comercial los vehículos con propulsión híbrida, basados en motores eléctricos y motores de combustión.

Paralelamente a la evolución técnica en la propulsión de los motores, la industria de la automoción fue generando nuevos dispositivos y técnicas que han ido permitiendo aumentar la seguridad, comodidad y eficiencia de

los vehículos.

En el contexto que nos ocupa, nos centraremos en los denominados sistemas de asistencia a la conducción, que podríamos definir como los sistemas desarrollados para automatizar, adaptar o mejorar los sistemas del vehículo buscando aumentar la seguridad del mismo así como una mejor experiencia en la conducción.

Estas tecnologías, cuyos principales exponentes serán introducidos en el apartado 2.1 de este texto, básicamente alertan al conductor de posibles problemas, o para evitar colisiones mediante la aplicación de correcciones y pudiendo incluso tomar el control del vehículo.

Como veremos algunas de estas técnicas pueden automatizar la iluminación, proporcionar el control de crucero adaptativo, automatizar frenado, incorporar advertencias GPS o de estado del tráfico, conectar a los teléfonos inteligentes, alertar a otros vehículos, mantener al conductor en el carril correcto, o mostrar lo que está en los puntos ciegos.

El siguiente objetivo natural de la automoción, era crear el primer vehículo de conducción totalmente autónoma.

Así, desde los años 80, importantes compañías, universidades y centros de investigación han desarrollado prototipos operativos de vehículos autónomos, como Mercedes-Benz, General Motors, Bosch, Nissan, Toyota, Audi, Volvo, Universidad de Parma, Universidad de Oxford, Google y otros.

La conducción totalmente autónoma en entornos urbanos se ha mantenido como una meta importante pero difícil de alcanzar. A tal fin, muchos notables intentos han sido realizados, y se han alcanzado varios hitos importantes. Quizás el que haya supuesto el mayor punto de inflexión fue el 2007 DARPA Urban Challenge, competición en la que equipos de todo el mundo diseñaron el hardware y software para vehículos autónomos, que debían ser capaces de circular con obstáculos dinámicos, intersecciones, etc., en un entorno urbano. En el evento, más de 50 vehículos robóticos y no robóticos condujeron simultáneamente en un circuito cerrado durante todo un día, y seis robots completaron con éxito la carrera [27].

Tras el DARPA Urban Challenge, se han seguido consiguiendo importantes hitos en la conducción autónoma, por mencionar algunos:

- En 2012, un vehículo del *Programa AUTOPIA* recorrió 100 kilómetros por las carreteras de la comunidad de Madrid. Un vehículo guía generó

dinámicamente un mapa de alta precisión que inmediatamente recorrió el vehículo automático que lo seguía. El viaje incluyó una amplia variedad de escenarios de conducción, incluyendo zonas urbanas, vías secundarias y autopistas bajo situaciones normales de tráfico [6].

- En julio de 2013, Vislab presentó mundialmente *BRAiVE*, un vehículo que circuló de manera autónoma en una ruta abierta al tráfico público [23].

- Mención aparte merece el Proyecto de *Google Self-Driving Car*, que en abril de 2014 anunció que habían logrado la cifra de 700.000 millas de navegación autónoma libres de accidentes para posteriormente anunciar un primer prototipo de vehículo autónomo sin volante, ni pedales, siendo 100 % autónomo [11]

Es importante destacar también que se han hecho avances en el marco legal para este tipo de vehículos, así pues, varios estados de los Estados Unidos permiten actualmente la circulación de vehículos autónomos mientras que en Europa, Alemania, Holanda, España y Reino Unido han permitido pruebas de coches robóticos en tráfico real [26].

En los Estados Unidos, la National Highway Traffic Safety Administration (NHTSA) ha establecido un sistema de clasificación oficial [7]:

- **Nivel 0:** el conductor controla completamente el vehículo en todo momento.
- **Nivel 1:** ciertos controls individuales del vehículo están automatizados, como el control electrónico de estabilidad o de frenado automático.
- **Nivel 2:** por lo menos dos controles se pueden automatizar al unísono, como el control de crucero adaptativo (ACC), en combinación con el mantenimiento de carril.
- **Nivel 3:** el conductor puede ceder por completo el control de todas las funciones críticas para la seguridad en determinadas condiciones. El coche detecta cuando las condiciones requieren que el conductor retome el control y proporciona un *tiempo de transición suficientemente cómodo* para que el conductor lo haga.
- **Nivel 4:** el vehículo realiza todas las funciones de seguridad críticas para todo el viaje, con lo que no se espera en ningún momento la

intervención del conductor humano. Como este vehículo podría controlar todas las funciones desde el principio hasta el fin del trayecto, incluyendo todas las funciones de aparcamiento, se entiende que podría incluir vehículos sin conductor.

El uso y desarrollo de los vehículos autónomos podría conllevar algunas de las siguientes potenciales ventajas:

- Menos accidentes de tráfico, debido a un sistema autónomo de mayor fiabilidad y tiempo de reacción más rápido en comparación a los conductores humanos.
- El aumento de capacidad de la carretera y la reducción de la congestión del tráfico debido a la menor necesidad de brechas de seguridad y la capacidad para gestionar mejor el flujo de tráfico.
- El alivio de los ocupantes de los vehículos de las tareas de conducción y navegación.
- Límite superior de velocidad para los vehículos autónomos.
- Alivio de la escasez de aparcamiento, ya que los coches podrían dejar a los pasajeros, aparcar lejos, donde no escasea el espacio, y de retorno cuando sea necesario para recoger a los pasajeros.
- Eliminación de pasajeros redundantes, el coche robótico podría conducir desocupado a donde sea necesario, por ejemplo para recoger a los pasajeros o para ir a mantenimiento. Esto sería especialmente relevante para camiones, taxis y servicios de uso compartido del coche.
- Reducción del espacio necesario para el estacionamiento de vehículos.
- Reducción de la necesidad de que la policía de tráfico y seguro de vehículos.
- Reducción de la señalización vial física, ya que los coches autónomos podrían recibir la información necesaria electrónicamente (aunque las señales físicos deberían seguir disponibles para los conductores humanos).

Sin embargo, también aparecen una serie de potenciales obstáculos para el desarrollo de estos sistemas:

- La responsabilidad por daños en caso de accidente.

- La resistencia psicológica de las personas a perder el control de sus coches.
- La fiabilidad del software.
- La seguridad electrónica, pues el un vehículo potencialmente podría verse comprometida, utilizando como vector de ataque el sistema de comunicación entre los distintos vehículos.
- Aplicación del marco jurídico y el establecimiento de las regulaciones por parte de los gobiernos para los vehículos autónomos.
- La pérdida de puestos de trabajo relacionados con la conducción.
- La pérdida de la privacidad.
- El establecimiento de un espacio radioeléctrico exclusivo para ser utilizado por estos sistemas.

Llegados a este punto, es evidente que un vehículo autónomo es un robot y se hace por tanto necesario realizar una breve introducción a la robótica en general explicando sus principios y definiciones básicas, y a la robótica móvil en particular, explicando los principales problemas que debe superar.

1.2. Introducción a la robótica

En el término robot confluyen las imágenes de máquinas para la realización de trabajos productivos y de imitación de movimientos y comportamientos de seres vivos.

Los robots actuales son obras de ingeniería y como tales concebidas para producir bienes y servicios o explotar recursos naturales.

En el proceso de creación de un robot, confluyen numerosos conocimientos y ramas de la ciencia y la tecnología, entre los que caben destacar la mecánica, la electrónica, la informática, inteligencia artificial e ingeniería de control.

Por tanto, podríamos definir a los robots *como máquinas en las que se integran componentes mecánicos, eléctricos, electrónicos y de comunicaciones, dotados de un sistema informático para su control en tiempo real, percepción del entorno y programación* [31].

En la figura 1.2 se muestra el esquema básico de un robot genérico, esquema que por supuesto es perfectamente válido para un vehículo autónomo. En él se identifican un sistema mecánico, actuadores, sensores y el sistema de control como elemento básico necesario para cerrar la cadena Actuación - Medidas - Actuación.

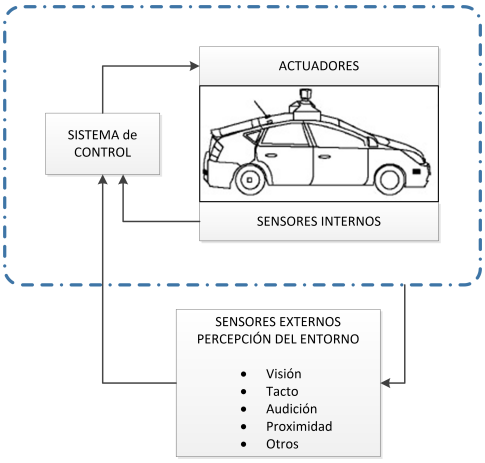

Figura 1.2: Robot y su interacción con el entorno.

Los sensores, se encargan de las labores de percepción y medida que necesita el robot para realizar su tarea, teniendo en cuenta que podemos realizar una clasificación de los sensores de un robot en externos e internos.

Los sensores internos miden el estado de la estructura mecánica del robot y, en particular, giros o desplazamientos relativos entre articulaciones, velocidades, fuerzas y pares.

Los sensores externos permiten dotar de sentidos al robot. La información que suministran es utilizada por el sistema de percepción para incorporar a su modelo la realidad del entorno.

Los sistemas de percepción sensorial hacen posible que un robot pueda adaptar automáticamente su comportamiento en función de las variaciones que se producen en su entorno, es decir, los sensores permiten al robot conocer el universo a su alrededor así como conocer su propio estado y actuar en consecuencia.

Existen diversos criterios para la clasificación de los distintos robots.

Según [31], atendiendo a su grado de autonomía, los robots pueden

clasificarse en teleoperados, de funcionamiento repetitivo y autónomos o inteligentes.

En los **robots teleoperados** las tareas de percepción del entorno, planificación y manipulación compleja son realizados por humanos. Es decir, el operador actúa en tiempo real cerrando un bucle de control de alto nivel. Los sistemas evolucionados suministran al operador realimentación sensorial del entorno (imágenes, fuerzas, distancias).

Los **robots de funcionamiento repetitivo** son la mayor parte de los que se emplean en cadenas de producción industrial. Trabajan normalmente en tareas predecibles e invariantes, con una limitada percepción del entorno. Son precisos, de alta repetibilidad y relativamente rápidos; incrementan la productividad ahorrando al hombre trabajos repetitivos y, eventualmente, muy penosos o incluso peligrosos.

Figura 1.3: Robot KUKA KR150.

Los **robots autónomos o inteligentes**, donde encontraríamos los vehículos autónomos, son los más evolucionados desde el punto de vista del procesamiento de información. Son máquinas capaces de percibir, modelar el entorno, planificar y actuar para alcanzar objetivos sin la intervención, o con una intervención mínima de supervisores humanos. Pueden trabajar en entornos poco estructurados y dinámicos, realizando acciones en respuesta a contingencias variadas en dicho entorno. Durante las últimas décadas se han realizado importantes esfuerzos en la aplicación de técnicas de inteligencia artificial. Se han empleado métodos simbólicos de tratamiento de la información basados en modelos geométricos del entorno.

De esta forma, se resuelven problemas basados en un modelo previo del entorno cuyas soluciones sólo son válidas si el modelo corresponde

exactamente a la realidad. La técnica obvia de reducir esta incertidumbre consiste en incrementar la información de que se dispone de dicho entorno mediante realimentación sensorial. Existen métodos que permiten intercalar la formulación y ejecución de planes con la captación de la información necesaria para asegurar que el modelo que se utiliza para la planificación sea lo suficientemente fiable. Las limitaciones vienen impuestas por el sistema de percepción y por la propia arquitectura del sistema de información y control del robot.

Desde el punto de vista de la planificación, existen diferentes arquitecturas diseñadas teniendo en cuenta especificaciones sobre el tiempo que tiene el sistema para responder y la disponibilidad de información potencialmente interesante.

La solución se sitúa normalmente entre dos extremos, en uno de los cuales está la planificación puramente estratégica. En este caso, se supone que la situación en la que va a ejecutarse el plan puede ser predicha de forma suficientemente precisa durante la planificación. En el otro extremo se sitúa la planificación puramente reactiva en la que se supone que el entorno es incierto, buscándose la mayor flexibilidad posible para reaccionar en cualquier instante lo suficientemente rápido a las discrepancias entre el modelo actual y la realidad observada en el entorno.

El problema puede plantearse también en términos de un compromiso entre eficiencia y flexibilidad. En efecto, las arquitecturas diseñadas para conseguir la mayor flexibilidad ante cualquier eventualidad del entorno son mucho menos eficientes que las que utilizan criterios de decisión basados en modelos del entorno suficientemente precisos sin tener demasiado en cuenta la posibilidad de generalizar el comportamiento. En este punto conviene poner de manifiesto el interés de las arquitecturas con capacidad de aprendizaje que combinan la planificación estratégica, basada en técnicas de búsqueda, con la planificación puramente reactiva.

Otra clasificación interesante de los robots puede realizarse atendiendo a su arquitectura. La arquitectura, que es definida por el tipo de configuración general del Robot, puede ser metamórfica. El concepto de metamorfismo, de reciente aparición, se ha introducido para incrementar la flexibilidad funcional de un Robot a través del cambio de su configuración por el propio Robot. El metamorfismo admite diversos niveles, desde los más elementales (cambio de herramienta o de efecto terminal), hasta los más complejos como el cambio o alteración de algunos de sus elementos o subsistemas estructurales. Los dispositivos y mecanismos que pueden agruparse bajo la denominación genérica del Robot, tal como se ha indicado, son muy diversos y es por tanto difícil establecer una clasificación coherente de los

mismos que resista un análisis crítico y riguroso. Así pues, una subdivisión de los Robots, con base en su arquitectura, se hace en los siguientes grupos: Poliarticulados, Móviles, Androides, Zoomórficos e Híbridos.

Los *robots poliarticulados* constituyen un grupo que incluye robots de muy diversa forma y configuración cuya característica común es la de ser básicamente sedentarios (aunque excepcionalmente pueden ser guiados para efectuar desplazamientos limitados) y estar estructurados para mover sus elementos terminales en un determinado espacio de trabajo según uno o más sistemas de coordenadas y con un *número limitado de grados de libertad*. En este grupo se encuentran los manipuladores, los robots industriales, y se emplean cuando es preciso abarcar una zona de trabajo relativamente amplia o alargada, actuar sobre objetos con un plano de simetría vertical o reducir el espacio ocupado en el suelo.

Figura 1.4: Robot móvil Pioneer 3-AT.

Los *robots móviles* son robots con grandes capacidad de desplazamiento, basados en carros o plataformas y dotados de un sistema locomotor de tipo rodante. Siguen su camino por telemando o guiándose por la información recibida de su entorno a través de sus sensores. Estos robots, en entornos industriales aseguran el transporte de piezas de un punto a otro de una cadena de fabricación. Guiados mediante pistas materializadas a través de la radiación electromagnética de circuitos empotrados en el suelo, o a través de bandas detectadas fotoeléctricamente, pueden incluso llegar a sortear obstáculos y están dotados de un nivel relativamente elevado de inteligencia.

La robótica móvil se extiende también más allá de los entornos industriales, apareciendo robots móviles dotados de una gran autonomía e inteligencia tanto para la navegación como para la evitación de obstáculos. A este respecto es muy interesante el campeonato y los equipos participan-

tes en el ya comentado DARPA Urban Challenge [27].

Por último, indicar que los *androides* son robots que intentan reproducir total o parcialmente la forma y el comportamiento cinemático del ser humano.

1.3. Robótica móvil

El desarrollo de robots móviles responde a la necesidad de extender el campo de aplicación de la robótica, restringido inicialmente al alcance de una estructura mecánica anclada en uno de sus extremos. Se trata también de incrementar la autonomía, limitando todo lo posible la intervención humana y aumentando en definitiva, las posibilidades y funcionalidades de los robots.

El objetivo final de la robótica móvil consiste en dotar al robot de la suficiente inteligencia como para reaccionar y tomar decisiones basándose en observaciones de su entorno, sin suponer que este entorno es perfectamente conocido a priori.

La autonomía de un robot móvil se basa principalmente en el sistema de navegación automática. En estos sistemas se incluyen tareas de planificación, percepción y control. En los robots móviles el problema de la planificación, en el caso más general, puede descomponerse en planificación global de la misión, de la ruta, de la trayectoria y finalmente, evitar obstáculos no esperados.

Existen numerosos métodos de planificación de caminos para robots móviles que se basan en hipótesis simplificadoras, tales como: entorno conocido y estático, robots omnidireccionales, con movimiento lento y ejecución perfecta de trayectoria. En particular hay muchos métodos que buscan caminos libres de obstáculos que minimizan la distancia recorrida en un entorno modelado mediante polígonos. En otros casos, se modela el espacio libre tratando de encontrar caminos por el centro del mismo. Para facilitar la búsqueda existen técnicas de descomposición del espacio en celdas, utilización de restricciones de varios niveles de resolución y búsqueda jerarquizada, que permiten hacer más eficiente el proceso con vistas a su aplicación en tiempo real, minimizando el coste computacional del mismo.

La planificación de la trayectoria puede realizarse también de forma dinámica, considerando la posición actual del vehículo y los puntos intermedios de paso definidos en la planificación de la ruta. La trayectoria se

corrige debido a acontecimientos no considerados. La definición de la trayectoria debe tener en cuenta las características cinemáticas del vehículo en cuestión. Por ejemplo, en vehículos con ruedas y tracción convencional, interesa definir trayectorias de curvatura continua que puedan ejecutarse con el menor error posible.

Además de las características geométricas y cinemáticas, puede ser necesario tener en cuenta modelos dinámicos de comportamiento del vehículo contemplando la interacción vehículo-terreno, es decir, el comportamiento de un mismo vehículo puede ser muy diferente según el tipo de terreno por el que se desplace. Por otra parte puede plantearse también el problema de la planificación de la velocidad teniendo en cuenta las características del terreno y del camino que se pretenda seguir.

Una vez realizada la planificación de la trayectoria, es necesario planificar movimientos concretos y controlar dichos movimientos para mantener al vehículo en la trayectoria planificada. De esta forma, se plantea el problema del seguimiento de caminos, que para vehículos con ruedas se concreta en determinar el ángulo de dirección teniendo en cuenta la posición y orientación actual del vehículo con respecto a la trayectoria que debe seguir. Asimismo, es necesario resolver el problema del control y regulación de la velocidad del vehículo durante todo el recorrido de la trayectoria.

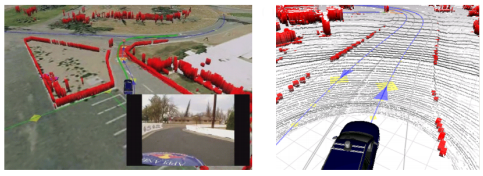

Figura 1.5: Robot Junior. Darpa 2007 Urban Challenge.

En cualquier caso, el problema del control automático preciso de un vehículo con ruedas puede resultar más complejo que el de los manipuladores debido a la presencia de restricciones no holónomas. Los bucles de control se plantean tanto en el espacio de las variables articulares como en coordenadas del mundo, y las ecuaciones de movimiento son complejas si se considera la interacción con el terreno. Mientras en manipuladores es relativamente fácil el cálculo y medida de los pares y fuerzas que se ejercen sobre la estructura mecánica, la determinación de estos pares en vehículos con ruedas es muy difícil. En la actualidad se emplean fundamentalmente métodos geométricos y modelos cinemáticos simplificados. No obstante, la consideración de aspectos dinámicos es necesaria cuando la velocidad es alta.

Nótese también que el control del vehículo requiere disponer de medidas de su posición y orientación, a intervalos suficientemente cortos. La técnica más simple consiste en la utilización de la odometría a partir de las medidas suministradas por los sensores situados en los ejes de movimiento, típicamente codificadores ópticos. Sin embargo la acumulación del error puede ser muy grande. Se emplean también sistemas de navegación inercial, Inertial Measurement Unit (IMU), incluyendo giróscopos y acelerómetros, aunque estos sistemas también acumulan error, especialmente en la determinación de la posición empleando los acelerómetros. No obstante, la combinación de las técnicas odométricas con la medida de los ángulos de orientación puede dar buenos resultados en intervalos de tiempo y distancia viajada suficientemente pequeños.

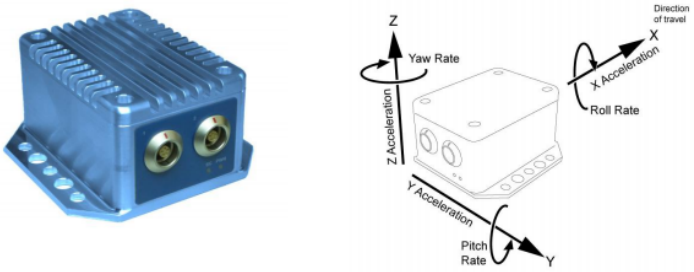

Figura 1.6: Racelogic IP66 VBOX Inertial Measurement Unit.

La corrección de la inevitable acumulación de error hace necesario el empleo de otros sensores. Con este fin, en aplicaciones de exteriores, en las que las distancias que recorre el vehículo autónomo son considerables, se emplean sistemas de posicionamiento global mediante satélites (GPS).

El sistema de percepción de un robot móvil o vehículo autónomo tiene un triple objetivo: permitir una navegación segura, detectando y localizando obstáculos y situaciones peligrosas en general, modelar el entorno construyendo un mapa o representación de dicho entorno (fundamentalmente geométrica), y estimar la posición del vehículo de forma precisa. Asimismo el sistema de percepción de estos robots puede aplicarse no sólo para navegar sino también para aplicaciones tales como el control de un manipulador situado en un el robot.

Para el diseño de estos sistemas de percepción deben tenerse en cuenta diferentes criterios, algunos de los cuales son conflictivos entre sí. De esta forma, es necesario considerar la velocidad del robot, la precisión, el alcance, la posibilidad de interpretación errónea de datos y la propia estructura de la representación del entorno.

En muchas aplicaciones se requiere tener en cuenta diversas condiciones de navegación con requerimientos de percepción diferentes. De esta forma, puede ser necesario estimar de forma muy precisa, aunque relativamente lenta, la posición del robot y a la vez, detectar obstáculos lo suficientemente rápido, aunque no se necesite una gran precisión en su localización.

Existen también arquitecturas en las que el sistema de percepción se encuentra integrado en el controlador de forma que, en entornos estructurados, es posible estimar de forma muy rápida la posición para navegar a alta velocidad.

Asimismo se han aplicado redes neuronales para generar el ángulo de dirección a partir del sistema de percepción.

Conviene mencionar también el interés del empleo de técnicas de procesamiento en paralelo para el tratamiento de imágenes en el guiado autónomo de vehículos.

Con respecto a los sensores específicos, además de las características de precisión, rango, e inmunidad a la variación de condiciones del entorno, es necesario tener en cuenta su robustez ante vibraciones y otros efectos originados por el vehículo y el entorno, su tamaño, consumo, seguridad de funcionamiento y desgaste.

Las cámaras de vídeo tienen la ventaja de su amplia difusión y precio, su carácter pasivo (no se emite energía sobre el entorno) y que no es necesario, en principio, el empleo de dispositivos mecánicos para la captación de la imagen. Las desventajas son los requerimientos computacionales, la sensibilidad a las condiciones de iluminación, y los problemas de calibración y fiabilidad.

La percepción activa mediante láser (LIDAR: Laser Imaging Detection and Ranging) es un método alternativo que ha cobrado una importante significación en robots móviles. Se utilizan dispositivos mecánicos y ópticos de barrido en el espacio obteniéndose imágenes de distancia y reflectancia a las superficies intersectadas por el haz.

Los sensores de ultrasonido son económicos y simples para la navegación. Se basan en la determinación del denominado tiempo de vuelo de un pulso de sonido (típicamente entre 30KHz y 1 MHz). Sin embargo, la influencia de las condiciones ambientales puede ser significativa, debiendo corregirse mediante una calibración adecuada.

Figura 1.7: Detección de líneas de carril.

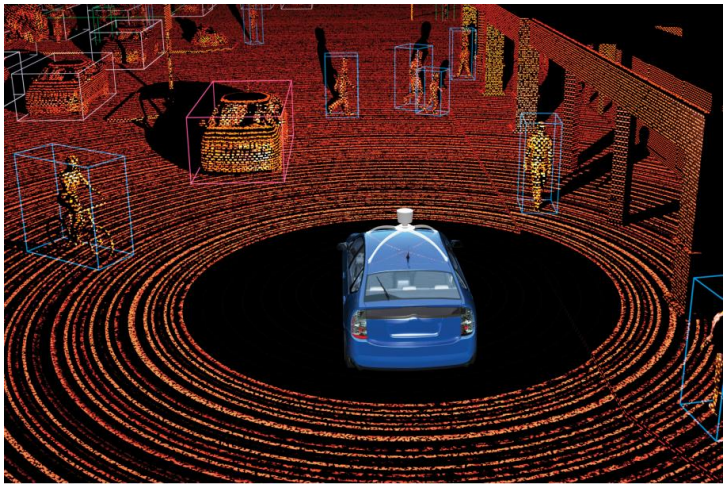

Figura 1.8: Google Self-Drive Car Project. Como ve el mundo un coche autónomo.

Por otra parte la relación señal/ruido es normalmente muy inferior a la de otros sensores, lo que puede hacer necesario el empleo de múltiples frecuencias y técnicas de filtrado y tratamiento de la incertidumbre de mayor complejidad computacional.

Asimismo, la resolución lateral es mala, existiendo para evitarlo técnicas de enfoque mediante lentes acústicas o transmisores curvos.

SLAM es la abreviatura en inglés para Simultaneous Localization And Mapping, es decir, localización simultánea y mapeo. Por mapeo entendemos el problema de integrar la información recopilada mediante los distintos sensores del robot para ser unificada en una representación única. Podría entenderse como la respuesta a la pregunta *¿cómo es el mundo a mi alrededor?* por supuesto, desde la perspectiva del robot.

Aspectos centrales en el mapeo son la representación del entorno y la interpretación de los datos de los sensores. En contraste a esta función, aparece la localización, que se corresponde con el problema de estimar la posición del robot respecto a un mapa.

En otras palabras, el robot debe responder en este caso a la pregunta *¿dónde estoy?* Es habitual distinguir entre seguimiento de la posición, dado que la localización inicial del vehículo es conocida y localización global, en donde no existe conocimiento a priori acerca de la posición inicial del robot.

Las técnicas de SLAM se definen por tanto como la técnica para construir un mapa mientras que al mismo tiempo el robot se localiza en dicho mapa.

En la práctica, ambos problemas no pueden resolverse de manera independiente, es decir, antes de que un robot pueda responder la cuestión acerca de cómo es su entorno tras una serie de observaciones, necesita conocer desde dónde se han realizado dichas observaciones.

Al mismo tiempo, es difícil estimar la posición actual del vehículo sin un mapa. Es por esto, que muchas veces SLAM es comparado con la paradoja del huevo y la gallina: se necesita un buen mapa para localizarse correctamente mientras que una estimación precisa de la posición es necesaria para construir un mapa fidedigno.

Figura 1.9: Mapa Campus Universidad de Freiburg (openslam.org).

1.4. Navegación en entornos urbanos

Los escenarios de aplicación de la robótica han evolucionado en las últimas décadas, desde entornos muy simples y controlados, típicamente entornos industriales, a entornos muy dinámicos en exteriores llegando al caso que nos ocupa, de la navegación autónoma de vehículos.

Al mismo tiempo, para afrontar ciertas aplicaciones, la cooperación en grupos de varios robots se ha convertido en una necesidad. Una tendencia en la actualidad es la investigación en sistemas que consideren la colaboración entre robots y sensores heterogéneos presentes en el entorno para multitud de aplicaciones, como robótica de servicio en entornos urbanos, o monitorización de desastres.

La razón fundamental es que estas aplicaciones involucran entornos dinámicos, como una ciudad, con condiciones cambiantes para la percepción, etc. En la mayoría de las ocasiones, un único agente (por ejemplo un robot o una cámara) no permite conseguir la robustez y eficacia necesarias. En estos casos, la cooperación de diferentes agentes (robots, sensores en el entorno) puede ser muy relevante.

Por otro lado, hay un creciente interés en la robótica de servicio por lo que cada vez son más frecuentes aplicaciones de robots en entornos ur-

banos, para tareas como el guiado y asistencia de personas, transporte de personas y objetos, etc. Ejemplo claro de esta tendencia es el anuncio que realizó Google en 2012, mostrando a un residente con un 90 % de ceguera del estado de California haciendo uso de uno de los vehículos autónomos de Google para atender a sus recados diarios.

Por citar otro ejemplo en el proyecto URUS [4] se obtuvieron interesantes resultados de robótica cooperativa en entornos urbanos, utilizando una flota de robots móviles, una red de cámaras fijas, así como una red inalámbrica de sensores.

Todos estos elementos pueden comunicarse entre sí de forma inalámbrica, y forman lo que se llama un sistema de robots en red (Network Robot System, NRS). Dicho sistema ha sido desplegado en un entorno urbano demostrando su utilidad. Por ejemplo, la fusión de la información de los distintos elementos permite un seguimiento más preciso, así como hacer frente a oclusiones, en tareas como el guiado de personas.

Una de las posibles aplicaciones de la robótica móvil en robótica de servicio es el guiado e incluso el transporte de personas y objetos en entornos urbanos.

Por ejemplo, en zonas peatonales o zonas que se convierten en peatonales en las ciudades para mejorar la calidad de vida de las ciudades. En este caso, los robots deben ser capaces de navegar a través de calles al mismo tiempo que las personas y posiblemente otros robots.

Figura 1.10: Robot móvil Romeo-4R. Experimento de evitación de obstáculo.

Adelantamos aquí que la navegación en entornos urbanos se enfrenta además a serios obstáculos, que no están presentes por ejemplo en un entorno industrializado. En efecto, la arquitectura y disposición de los distintos elementos en una zona urbana dista mucho de ser un entorno sencillo de modelar como una nave industrial: bordillos, aceras, tipos de adoquines, peatones, otros vehículos convencionales, rampas, escaleras, etc., situacio-

nes que todas ellas deben ser tenidas en cuenta, desarrollando estrategias para que el robot pueda solventarlas con éxito.

Capítulo 2

Estado del arte

2.1. Sistemas de asistencia a la conducción

El número de vehículos en las carreteras es una cifra que va en aumento cada año y desgraciadamente el número de accidentes en carretera tiende a ser proporcional a dicho incremento. Actualmente muchos vehículos son mejorados usando los denominados sistemas de asistencia a la conducción o Driver Assistance Systems (DAS). Dichos sistemas son implementados y diseñados para mejorar la seguridad en el tráfico y con la esperanza de reducir el número de accidentes de conducción, particularmente los mortales, causados por errores humanos en la conducción. Muchísimos esfuerzos y colaboraciones han sido llevados y se continuan realizando entre el mundo investigador y la industria de la automoción [18].

Por tanto, volvemos aquí a definir los denominados sistemas de asistencia a la conducción como los sistemas desarrollados para automatizar, adaptar o mejorar los sistemas del vehículo buscando aumentar la seguridad del mismo así como una mejor experiencia en la conducción.

A continuación repasamos los principales sistemas de asistencia a la conducción que ya han sido incorporados en vehículos comerciales.

2.1.1. Sistemas frontales de luces avanzados (AFS)

Desde principios del siglo XXI, se incrementaron los esfuerzos por optimizar el haz luminoso frontal del vehículo, siendo capaz éste de adaptarse, no sólo a las condiciones dinámicas de suspensión del vehículo sino también a las condiciones ambientales, de visibilidad del entorno, velocidad del vehículo y curvatura de la carretera, entre otras posibles variables.

Es evidente, que para la implementación de estos *Sistemas frontales de*

luces avanzados o Advanced Front-lighting Systems (AFS) son necesarios sensores de luminosidad, humedad, encoders para el radio de giro del volante y ruedas frontales, giróscopos, IMUs, etc.

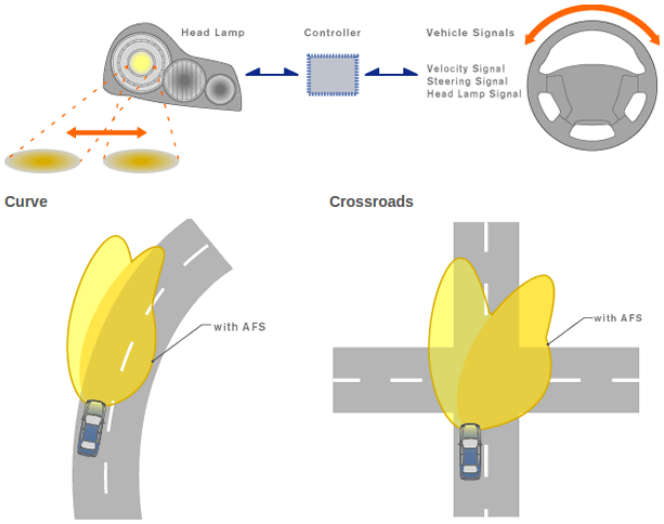

Figura 2.1: AFS - Nissan Technology.

Así, numerosos fabricantes han ido desarrollando y comercializando en sus modelos sistemas AFS, incorporando sistemas básicos, que pueden alternar entre luces cortas o largas en función de si existe algún vehículo delante del propio o que dirigen el foco luminoso en función del radio de curvatura del volante o más avanzados, capaces de usar señales y mapeados GPS para anticipar cambios en la curvatura de la carretera en vez de simplemente reaccionar ante los cambios.

2.1.2. Cambio involuntario de carril (LDW)

Los sistemas de *Cambio involuntario de carril o Lane Departure Warning (LDW)* alertan al conductor si abandona el carril por el que va circulando sin accionar antes los intermitentes, lo que se identifica como una distracción.

La detección de las líneas de demarcación del carril es un campo ampliamente trabajado en la literatura científica, habiéndose implementado y comprobado la validez de numerosas técnicas para este fin. Por ejemplo, en la figura 2.2 se muestra el trabajo [16], donde la detección de las líneas de carril se logra haciendo un seguimiento en cada imagen de las líneas detectadas y una aproximación posterior de los carriles usando un modelo basado en splines.

Figura 2.2: Detección de líneas de carril.

Los sistemas que incorporan los modelos comerciales actuales son básicamente de dos tipos, *pasivos*, que simplemente alertan al conductor mediante algún tipo de señal acústica, vibracional o visible o bien *activos*, pudiendo ejercer acciones correctoras sobre la dirección si fuese necesario.

Desde el punto de vista de la implementación, los sistemas utilizados para la detección de las líneas de la carretera se basan en varios tipos de sensores:

- *Cámaras de vídeo*, montadas habitualmente tras el espejo retrovisor interior del vehículo.
- *Sensores láser*, montados en el frontal del vehículo.
- *Sensores de infrarrojos*, montados habitualmente tras parabrisas o bajo el propio vehículo.

El trabajo realizado en el presente TFM podría interpretarse también como una implementación de este tipo de sistemas.

2.1.3. Head Up Display (HUD)

Es un sistema que procede del mundo de la aviación y que proyecta en el parabrisas del vehículo, a la altura de nuestros ojos, la información más importante del cuadro de instrumentos (velocidad, indicaciones de navegación...).

Estos sistemas únicamente suponen una asistencia al conductor por el hecho de que facilitan el acceso del mismo a cierta información importante del estado del vehículo, así, el conductor no tiene que apartar la vista de la carretera en ningún momento para atender o recibir dicha información.

2.1.4. Detector de ángulo muerto (BLIS)

Los sistemas de detección de ángulo muerto o Blind Spot Information System (BLIS), alertan, de manera visual o sonora, de la presencia de vehícu-

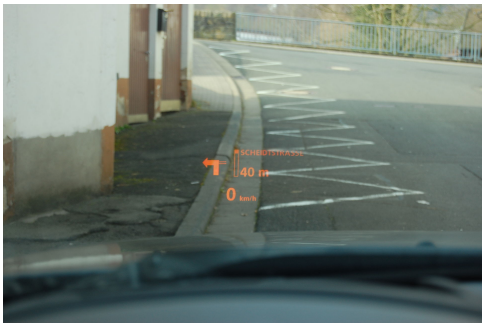

Figura 2.3: HUD en un modelo BMW E60.

los en el ángulo muerto. Algunos lo hacen de forma continua, mientras que otros se limitan a hacerlo sólo en el momento en el que el conductor activa el intermitente para efectuar un cambio de carril.

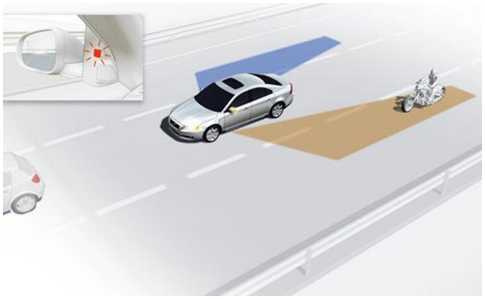

Figura 2.4: Sistema BLIS - Volvo.

Los sistemas BLIS se implementan utilizando principalmente sensores de ultrasonidos, tipo sonar, que pueden ser fácilmente integrados en la carrocería del vehículo.

Figura 2.5: Sensor de ultrasonido HC-SR04.

2.1.5. Control de velocidad de crucero adaptativo (ACC)

Los sistemas de control de velocidad se introdujeron inicialmente como una función para mantener la velocidad de un automóvil a un valor de re-

ferencia seleccionada por el conductor.

En los últimos años, estos sistemas han evolucionado y se han desarrollado los denominados controles de velocidad de crucero adaptativo o Adaptive Cruise Control (ACC), o control de crucero inteligente, que se caracterizan por tener una función adicional, en la que el vehículo es capaz de adaptar su propia velocidad en función de la velocidad del vehículo precedente, en caso de que este vaya a una velocidad inferior a la referencia dada.

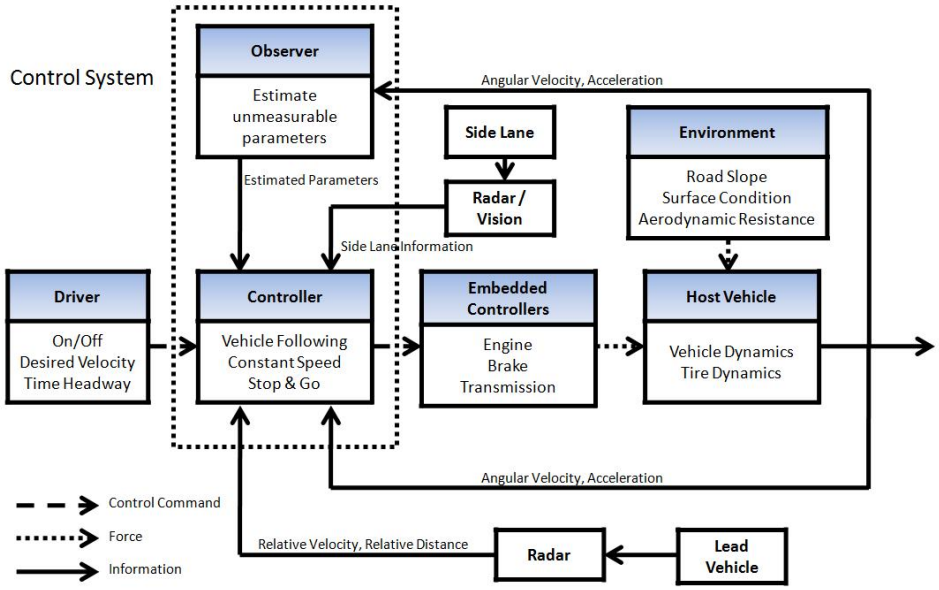

Figura 2.6: Diagrama de Flujo Sistema ACC - Universidad de Berkeley.

Los esfuerzos de investigación y desarrollo continúan tanto en el sector académico e industrial para mejorar aún más la capacidad de los sistemas ACC [8].

2.1.6. Asistente para el aparcamiento

Los sistemas comerciales actuales permiten que el vehículo detecta plazas de aparcamiento libres con el tamaño adecuado (longitudinal, transversal y diagonal). Aparca automáticamente el vehículo en la plaza tras la aprobación del conductor, utilizando independientemente el freno y la dirección.

Sin embargo, estos sistemas que han comenzado a comercializarse ha-

ce relativamente poco tiempo, llevan un largo recorrido en el mundo investigador. Así, en el trabajo [17], se define el concepto de maniobra en un robot móvil rodado. Para ello, se introduce el concepto de inversor. Un inversor es un punto de la trayectoria donde la velocidad del robot cambia de signo. De esta forma, para Latombe, una maniobra consiste en una concatenación de trayectorias separadas por inversores.

Por ejemplo, en el artículo [24] se presenta la aplicación de técnicas para la realización de maniobras con vehículos no holónomos[1]. En particular, se consideran de forma explícita la evitación de colisiones y la realización de dichas maniobras en entornos con poco espacio para maniobrar e incluyen resultados experimentales de la aplicación al aparcamiento de un coche eléctrico adaptado y de un vehículo con remolque.

Figura 2.7: Aparcamiento paralelo robot Romeo-4R - Universidad de Sevilla.

2.1.7. Frenada automática de emergencia (AEB)

Los *sistemas de frenada automática de emergencia o Autonomous Emergency Braking (AEB)* permiten que el vehículo pueda prevenir colisiones por alcance mediante una frenada automática, o al menos mitigar sus efectos

[1]Restricción de movimiento que viene expresada matemáticamente por una desigualdad. Por ejemplo, cuando una partícula resbala sobre la parte de superior de una esfera, debido a que, llegado a un punto, la partícula se separa de la superficie de ésta, por lo que no se cumple que el movimiento de la partícula esté restringido a su superficie.

en caso de colisión inminente.

Para la detección de obstáculos pueden utilizarse sistemas LIDAR, RADAR, ultrasonidos e incluso sistemas de reconocimientos de imágenes mediante cámaras.

El sistema detecta la presencia de obstáculos en la calzada y, en centésimas de segundo, es capaz de analizar si con la velocidad a la que nos movemos y la distancia al obstáculo, la colisión es inminente. Si es el caso, y antes de que el conductor reaccione, el sistema acciona el freno para evitar el golpe.

2.1.8. Sistema de reconocimiento de señales (TSR)

Los sistemas de reconocimiento de señales o Traffic Sign Recognition (TSR) muestran al conductor las diferentes señales que van siendo detectadas durante la circulación.

Estos sistemas se basan en el uso de una cámara de vídeo, situada generalmente a la altura del espejo retrovisor interior, que procesa las imágenes de la carretera y las muestra, bien en el cuadro de instrumentos o en el sistema de información proyectada en el parabrisas (HUD).

La detección de ciertas señales permitiría a los vehículos acondicionar su circulación, por ejemplo, reduciendo automáticamente la velocidad máxima al límite permitido en dicho tramo.

Mientras que las tareas de detección de vehículos o peatones pueden ser realizadas mediante sensores activos o sensores mixtos (activos y pasivos), el reconocimiento de señales de tráfico únicamente puede ser realizado mediante sistemas de visión por computador. Para este objetivo es habitual utilizar técnicas de reconocimiento basadas en el color y la forma de las señales [18], aunque es posible encontrar en la literatura científica técnicas más avanzadas para mejorar los resultados y problemáticas asociadas a estás técnicas básicas.

2.2. Conducción autónoma

A pesar de que es actualmente cuando se está empezando a hacer publicidad sobre estos vehículos o robots autónomos, haciendo especulaciones sobre cuál de los grandes fabricantes del mercado de la automoción será el primero en comercializar un modelo totalmente autónomo la conduc-

ción autónoma es una realidad desde hace bastante tiempo gracias a los trabajos realizados por investigadores y desarrolladores en todo el mundo.

En este apartado vamos a describir los principales proyectos e hitos alcanzados en esta temática, tanto por universidades de todo el mundo como por iniciativas privadas.

2.2.1. DARPA Grand Challenge

DARPA acrónimo de la expresión en inglés Defense Advanced Research Projects Agency (Agencia de Proyectos de Investigación Avanzados de Defensa), es una agencia del Departamento de Defensa de los Estados Unidos responsable del desarrollo de nuevas tecnologías para el uso militar.

DARPA ha sido responsable de la financiación de muchísimas tecnologías que han tenido un gran impacto en todo el mundo,siendo posiblemente la más conocida de todas el desarrollo de la actual Internet.

DARPA Grand Challenge es una competición para vehículos autónomos creada para fomentar el desarrollo de las tecnologías necesarias para crear los primeros vehículos terrestres autónomos capaces de completar un cierto recorrido en un tiempo limitado.

Se han realizado hasta la fecha tres ediciones de la competición en los años 2004, 2005 y 2007, siendo la de 2007 la única en la que el recorrido se situaba en un entorno urbano, como veremos más adelante.

Actualmente y tras los increíbles éxitos obtenidos en las ediciones anteriores, DARPA ha centrado sus objetivos en la robótica humanoide, habiendo convocado el 2012 DARPA Robotics Challenge, donde robots humanoides deben superar ciertas pruebas en entornos que simulan situaciones de desastres.

2004 Grand Challenge

La primera competición DARPA Grand Challenge se realizó el 13 de marzo de 2004 en el desierto de Mojave de los Estados Unidos, constando de una ruta de unos 240 Km, desde la interestatal 15 desde Barstow, California hasta la frontera entre California y Nevada en Primmm.

Ninguno de los vehículos robotizados fue capaz de finalizar la ruta. El equipo rojo de la La Universidad Carnegie Mellon, con un vehículo modelo Humvee modificado lograron recorrer la mayor distancia, completando

11.78 Km antes de quedar atrapado por una roca.

En esta primera edición el ganador del premio, de 1 millón de dólares quedó desierto, siendo programada una nueva edición para el año siguiente.

2005 Grand Challenge

La segunda competición DARPA Grand Challenge tuvo lugar el 8 de octubre de 2005. En ella, todos excepto uno de los 23 finalistas superaron la marca de 11.78 Km logrados en la anterior edición. 5 vehículos completaron con éxito el recorrido.

El robot de la Universidad de Stanford, Stanley, finalizó el recorrido empleando un tiempo de 6 horas y 53 minutos y 68 segundos siendo declarado vencedor del DARPA Grand Challenge 2005, que en esta ocasión estaba premiado con 2 millones de dólares.

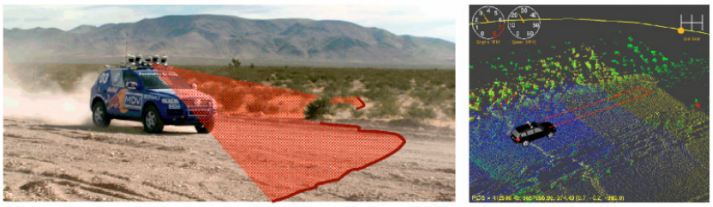

Figura 2.8: Stanley. Robot ganador del DARPA Grand Challenge 2005.

Stanley fue desarrollado por un equipo de investigadores para hacer avanzar el estado del arte en la conducción autónoma. El éxito de Stanley es el resultado de un intenso esfuerzo de desarrollo liderado por la Universidad de Stanford que involucró expertos de Volkswagen, Mohr Davidow Ventures, Intel Research y otras instituciones.

EL mayor reto tecnológico en el desarrollo de Stanley fue crear un sistema altamente fiable capaz de conducir a relativas altas velocidades a través de ambientes diversos, des estructurados y rurales (sin carretera) , todo ello con gran precisión. Todos estos requerimientos llevaron a lograr numerosos avances en el campo de la navegación autónoma [33].

Sin entrar en grandes detalles, es interesante comentar que el software de Stanley está dividido en seis grupos funcionales: Interfaz de sensores, percepción, control, interfaz con el vehículo e interfaz de usuario y servicios globales. Esta arquitectura software modular ha sido utilizada como guía o inspiración en multitud de trabajos posteriores.

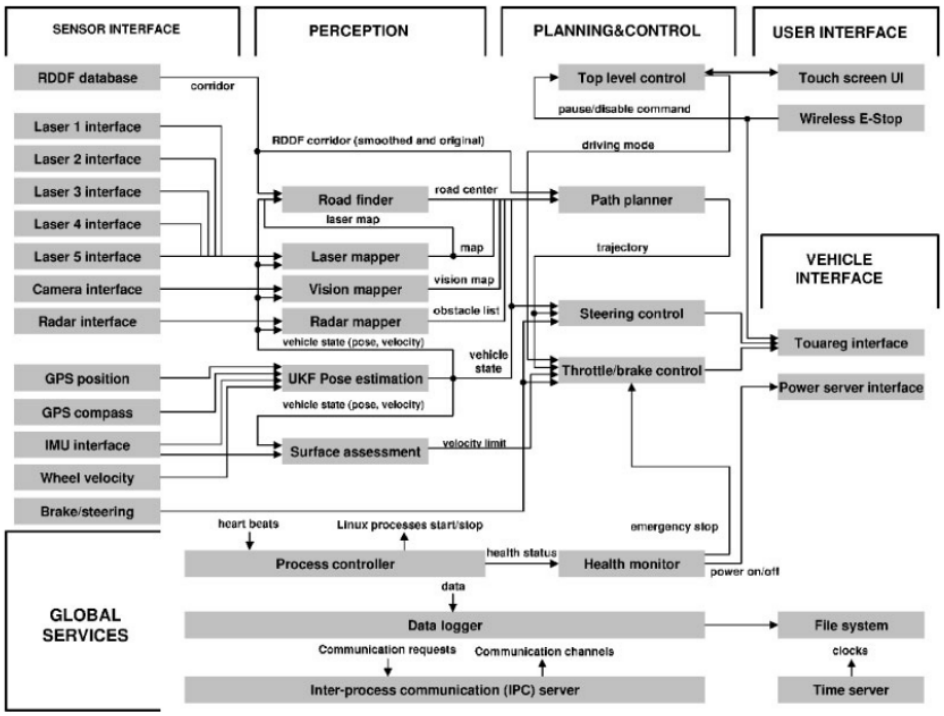

Figura 2.9: Diagrama de flujo del software de Stanley.

2007 Urban Grand Challenge

La tercera competición del DARPA Grand CHallenge, conocida como el Urban Challenge, o reto urbano, tuvo lugar el 3 de noviembre de 2007 en Victorville, California. El recorrido consistían en 96 km en zona urbana a ser completados en menos de 6 horas. Las reglas de la competición incluían obedecer todas las regulaciones de circulación mientras se circulaba con otro tráfico no robótico y obstáculos.

Es decir, en este caso los robots debían ser capaces de realizar tareas comunes como seleccionar sus rutas a destinos, cambios de carril, giros en U, o incluso aparcar, todo ello circulando con tráfico tanto robótico como no robótico.

El equipo Tartan Racing se alzó victorioso en la competición, logrando el primer premio de 2 millones de dólares con su robot Boss, un Chevy Tahoe. El segundo puesto, premiado con 1 millón de dólares lo alcanzó el equipo Stanford Racing, con su Volkswagen Passat, Junior mientras que el tercer puesto y sus 500 mil dólares correspondieron al equipo VictorTango, Odín, un híbrido Ford Escape de 2005. Otros tres equipos más lograron finalizar el reto.

Figura 2.10: Instante del 2007 DARPA Urban Challenge.

Mientras que las convocatorias de 2004 y 2005 tenían mayores requerimientos físicos para los vehículos (los recorridos eran mucho más duros y peligrosos), la convocatoria de 2007 requirió a los diseñadores construir robots capaces de obedecer todas las leyes de tráfico mientras detectaban y evitaban a otros vehículos, tanto humanos como robóticos.

Estos requerimientos suponen un mayor reto para el software de los robots participantes ya que los vehículos deben tomar decisiones inteligentes en función de las acciones de otros vehículos, en un entorno más complejo que algunos más controlables como la circulación en autovías o autopistas donde se minimiza la interacción con otros vehículos.

Capítulo 3

Objetivos

Los objetivos específicos a alcanzar en el presente TFM se han definido como:

- Implementación de un **sistema de visión por computador**, aplicado a los sistemas navegación o conducción de vehículos. Dicho sistema de visión deberá ser capaz de detectar los límites izquierdo y derecho de la carretera así como el horizonte, permitiendo *extraer características de la imagen*, estimando la localización relativa del vehículo obteniendo la distancia paralela hacia los límites de la carretera así como la orientación respecto a los mismos.

  Para el correcto diseño y ajuste del algoritmo, se utilizarán una serie de datasets públicos sobre imágenes de carreteras utilizados en otros proyectos de investigación recogidos en el apéndice A.

- Implementación de un **controlador basado en lógica difusa**, que a partir de los parámetros de localización relativa del vehículo respecto a los bordes de la carretera garantice que el vehículo nunca se salga de la carretera y detenga el vehículo en caso de peligro.

- Diseño de una **arquitectura software y hardware** completa que permita incorporar los anteriores objetivos en un prototipo de robot autónomo basado en un coche de radio control comercial.

- Implementación de una **interfaz gráfica**, que permita monitorizar el estado del robot, enviarle comandos de dirección y velocidad así como mostrar en pantalla información relativa a la geolocalización del robot e información proveniente del resto de sensores del vehículo.

- Como objetivo extra opcional, se incluye la realización de una **batería de experimentos** que permita comprobar la viabilidad del prototipo. Los experimentos propuestos son:

  - Recorrido en línea recta superior a 50 metros.

- Recorrido en línea recta superior a 100 metros.
- Recorrido en L tomando curva hacia la izquierda superior a 50 metros.
- Recorrido en L tomando curva hacia la derecha superior a 50 metros.

Capítulo 4

Materiales y metodología

En este capítulo explicaremos en primer lugar la arquitectura o plataforma hardware utilizada y posteriormente la arquitectura software del mismo, explicando inicialmente los distintos módulos desarrollados y posteriormente haciendo un breve resumen de las principales librerías software que dan soporte a dichos módulos.

4.1. Arquitectura Hardware

La plataforma robótica está formado por un vehículo de radiocontrol comercial, con motor eléctrico DC, modificado para poder ser controlado de manera autónoma.

Figura 4.1: Prototipo de Robot.Vista superior y lateral

Para ello se utilizará como controlador de bajo nivel una placa Arduino [2] y como sistema de adquisición de imágenes y control de alto nivel una plataforma smartphone con sistema operativo Android [1], ambos elementos que deben ser instalados en la plataforma robótica, junto con alguna electrónica adicional de seguridad.

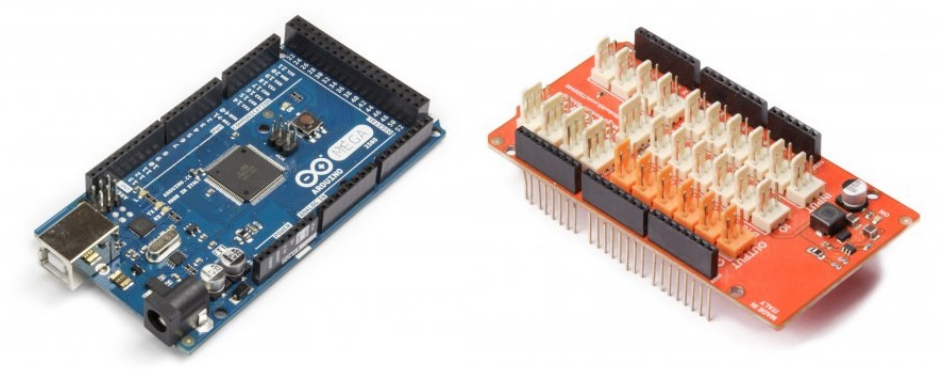

Figura 4.2: Arduino Mega 2560 Rev3 y TinkerKit Mega Sensor Shield V.2

El modelo utilizado Arduino Mega 2560 [9] es una placa microcontroladora basada en el chip ATMega2560. Tiene 54 pines de entrada/salida digitales, 14 de los cuales pueden ser utilizados como salidas PWM, 16 entradas analógicas, 4 UARTs, cristal de reloj a 16 MHz, conexión USB e interfaz ICSP.

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (15 PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB (8 KB bootloader)
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
Length	101.52 mm
Width	53.3 mm

Tabla 4.1: Arduino Mega 2560 Rev3. Especificaciones técnicas.

Para facilitar el conexionado de periféricos las placas Arduino constan

de las llamadas shields. EN nuestro caso hemos utilizado el modelo Mega Sensor Shield v.2 [15] del fabricante TinkerKit, el cual incluye un conector USB para interactuar con dispositivos Android a través del módulo Android Accessory Development Kit (ADK).

Tiene 22 conectores estándar Futaba J de 3 pines. Los pines I0 a I9 son entradas analógicas, los pines O0 a O5 son salidas analógicas, conectadas a los pines PWM de la placa Arduino, aunque también pueden configurarse como entradas digitales.

En la figura 4.3 se recoge el diagrama de conexiones utilizado.

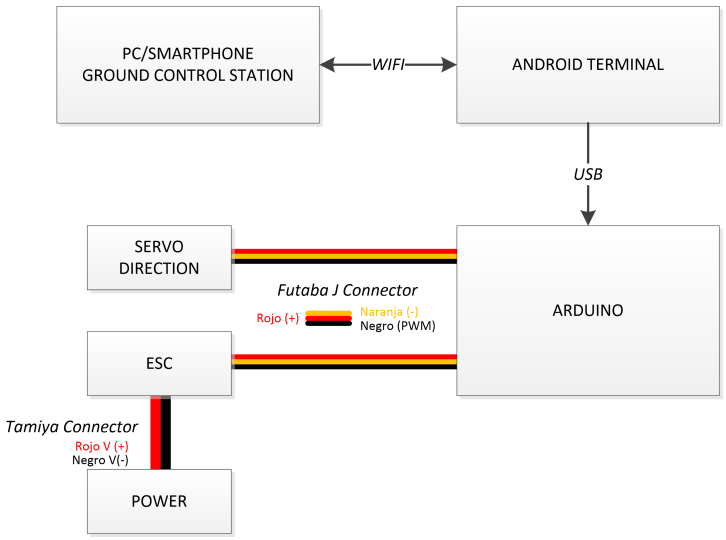

Figura 4.3: Diagrama de Conexiones.

4.2. Arquitectura Software

En la figura 4.4 se recoge la arquitectura software jerárquica del robot. En la parte izquierda del modelo se han unificado los elementos relacionados con las funciones cognitivas del robot, mientras que en la parte derecha se recogen las funciones de planificación y actuación.

La arquitectura software del robot se basa en el modelo de tres capas, que consisten en una capa reactiva, una capa ejecutiva y una capa deliberativa [32], que serán introducidas en los próximos apartados.

El desarrollo para este TFM ha estado centrado en el desarrollo de los módulos de percepción, esto es, sistema de geolocalización y sistema de

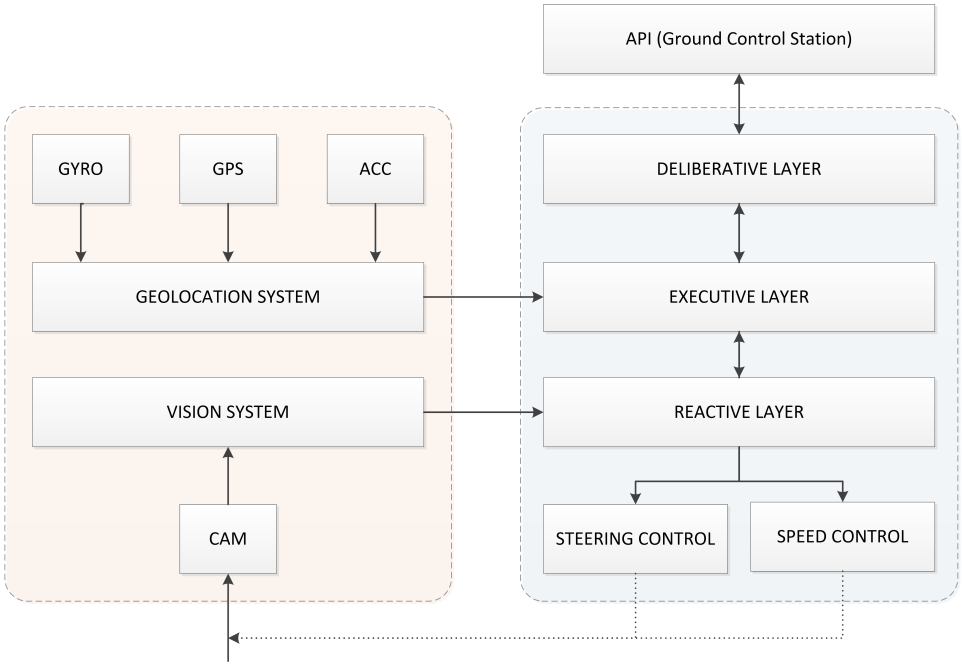

Figura 4.4: Arquitectura del robot.

visión, así como de una capa reactiva mínima, que incluye el control en velocidad y dirección mediante un controlador FLC. El desarrollo de las capas más inteligentes de la arquitectura robótica está fuera del ámbito de este trabajo.

4.2.1. Sistema de Visión

Sistema General

El sistema de visión se ha diseñado siguiendo el diagrama de bloques mostrado en la figura 4.5, donde vemos un primer bloque destinado al preprocesado de imagen, un segundo bloque de segmentación y el último bloque de análisis de imagen propiamente dicho, que podemos definir como un **Módulo de Extracción de Características**, capaz de estimar la posición y orientación de la cámara, y por tanto del robot, respecto de las líneas laterales de la carretera.

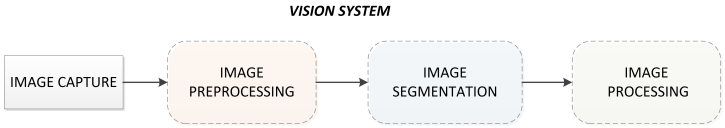

Figura 4.5: Vision System Workflow.

Es importante antes de continuar hacer hincapié en que el escenario contemplado para este trabajo es muy sencillo. Así, vamos a suponer siempre la existencia de las líneas de la carretera, correctamente iluminadas y con suficiente contraste respecto al resto de la carretera y el exterior de la misma, lo cual nos permitirá detectarlas y hacer un seguimiento de las mismas.

Debemos tener en cuenta que de no existir estas líneas, el enfoque utilizado sería inútil y para que una cámara detectara la zona por la que debe circular habría que usar un enfoque radicalmente distinto como por ejemplo diferencia de contrastes por regiones, o integración con otros sensores, como se muestra en la figura 4.6, donde el robot Stanley encuentra superficies navegables proyectando áreas navegables obtenidas del análisis LIDAR en la imagen obtenida por la cámara [33].

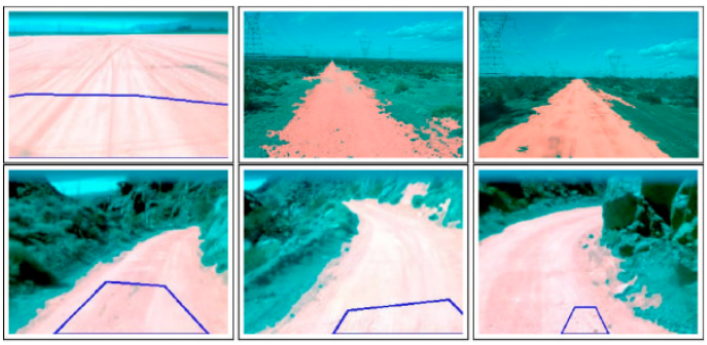

Figura 4.6: Imágenes procesadas por Stanley - 2005 DARPA Grand Challenge.

A continuación pasamos a describir todos y cada uno de los módulos implementados.

Preprocesado de Imagen

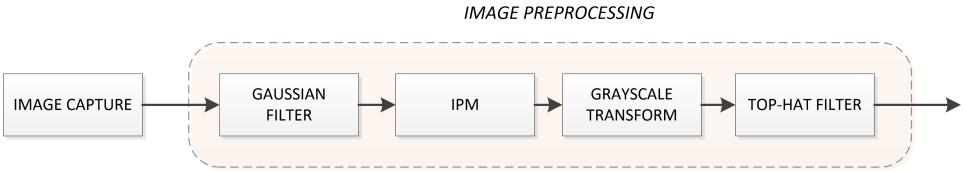

Figura 4.7: Bloque de preprocesado de imagen.

- **Filtro Gaussiano.** El primer paso del preprocesado consiste en una operación de suavizado de la imagen, mediante la aplicación de un

filtro gaussiano, con el objetivo de reducir el posible ruido de la imagen o algunos artefactos introducidos por la cámara. Es un paso muy habitual en sistemas de tratamiento de imagen.

- **Transformación Inverse Perspective Mapping (IPM).** Los sistemas automáticos de seguimiento de carril basados en visión, requieren información como las líneas de carril, la curvatura de la carretera o detección del vehículo precedente antes de capturar la siguiente imagen. Al colocar una cámara en el vehículo y tomar imágenes desde el morro o salpicadero del vehículo estamos tomando imágenes de la carretera con una cierta perspectiva. Dicha perspectiva distorsiona en cierta medida la forma de la carretera, afectando a su anchura, altura y profundidad, por lo que se hace necesario cierto preprocesado de la imagen para compensar estos efectos [28].

Figura 4.8: Ejemplo de transformación IPM.

Una técnica habitual utilizada es la denominada Inverse Perspective Mapping (IPM), con la que se intenta conseguir una imagen a vista de pájaro (birds-eye) de la zona de la imagen más cercana al vehículo, tal y como se muestra en la figura 4.8, donde a la izquierda, resaltada en amarillo tenemos la ROI sobre la que hemos aplicado la transformación IPM, cuyo resultado se muestra en la parte derecha.

Esta transformación es considerada como una transformación homográfica plano a plano en geometría proyectiva. Para implementar esta funcionalidad nos hemos basado en el siguiente proyecto [12].

- **Selector de canal de color (R/G/B).** Separando la información de la imagen proveniente en los distintos canales de color de la misma, se ha intentado mejorar los resultados en segmentación, sin embargo, la información parece encontrarse uniformemente distribuida en los tres canales, pues no ha habido mejoras significativas ejecutando los algoritmos usando como entrada de información uno sólo de los

canales de color, por lo que al final para la se ha utilizado una **transformación a escala de grises** para trabajar con el nivel de intensidad lumínica en las siguientes etapas.

- **Filtrado TopHat.** El filtrado TopHat es útil para aislar manchas que son respectivamente más brillantes que sus vecinos inmediatos, por lo que parece una buena herramienta a incorporar en la detección de las líneas de carril.

  $$TopHat(A) = A - open(A)$$

  El operador TopHat sustrae de la imagen el resultado de realizar una operación de apertura (open) sobre la propia imagen. Recordar que la operación *open()* tiene el efecto de exagerar pequeñas grietas o manchas puntuales en la imagen.

Segmentación

Para la detección de las líneas de la carretera se ha utilizado el conocido algoritmo de Canny en la fase de segmentación de la imagen, lo que permite detectar una amplia gama de bordes.

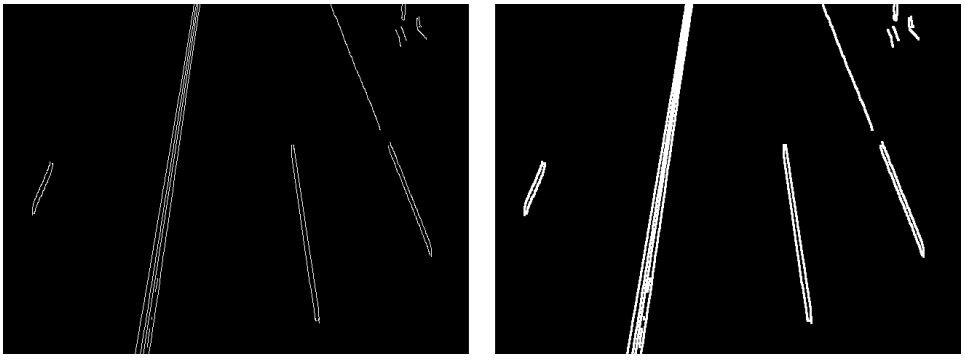

Figura 4.9: Dilatación en máscara filtro Canny.

Para facilitar la detección de las mismas, se ha utilizado una técnica que consiste en mejorar la máscara proporcionada a la salida del algoritmo Canny, aplicándole una operación morfológica de *dilatación*, como muestra la imagen 4.9 permitiendo obtener mejores resultados posteriormente con la Transformada de Hough.

Transformada de Hough

La Transformada de Hough[22] es una técnica para la detección de figuras en visión por computador con la que es posible encontrar figuras que

puedan ser expresadas matemáticamente como rectas, circunferencias o elipses.

En 1962, Paul Hough (1962) propuso y patentó[25] el método que toma su nombre, en el que debemos considerar un punto (x_i, y_i) en el plano (x,y) y la ecuación general de una recta del tipo pendiente-intersección, $y_i = ax_i + b$. Infinitas líneas pasan a través del punto (x_i, y_i) que satisfacen la ecuación anterior, variando los valores de las constantes a y b.

Sin embargo, la ecuación puede reescribirse como $b = -ax_i + y_i$ y considerando el plano (a,b), llamado espacio de parámetros, reduce la ecuación de una única linea a un par fijo (x_i, y_i).

Además, existe un segundo punto (x_j, y_j) que también posee una línea en el parámetro espacio asociado con él, y que, a no ser que las líneas sean paralelas, intersecta la línea asociada con (x_i, y_i) en algún punto (a', b'), donde a' es la pendiente y b' la intersección de la línea que contiene ambos puntos (x_i, y_i) y (x_j, y_j) en el plano (x,y).

En realidad, todos los puntos en esta línea tienen líneas en el espacio de parámetros que intersectan con (a', b').

En principio, pueden ser pintadas las líneas del espacio parámetros correspondientes a todos los puntos (x_k, y_k) del plano (x,y), y las líneas principales de dicho plano pueden encontrarse identificando puntos en el espacio de parámetros donde intersectan un gran número de líneas del espacio de parámetros.

No obstante, una dificultad práctica con este enfoque reside en que la pendiente de una recta, a, se aproxima a infinito conforme la línea se aproxima a la dirección vertical. Una forma de superar esta dificultad consiste en utilizar la representación normal de una recta, según la ecuación: $x \cdot cos\theta + y \cdot sin\theta = \rho$.

La figura 4.10 ilustra la interpretación geométrica de los parámetros ρ y θ, que conforman el espacio de parámetros (ρ, θ), también llamado espacio de Hough. Una recta en dirección horizontal se caracteriza por tener $\theta = 0^o$ mientras que una recta en dirección vertical se caracteriza por tener $\theta = \pm 90^o$.

Cada curva sinusoidal en el espacio de Hough representa la familia de rectas que atraviesan un punto particular del plano (x,y). El punto de intersección (ρ', θ') se corresponde con la línea que pasa por los puntos (x_i, y_i) y (x_j, y_j) simultáneamente.

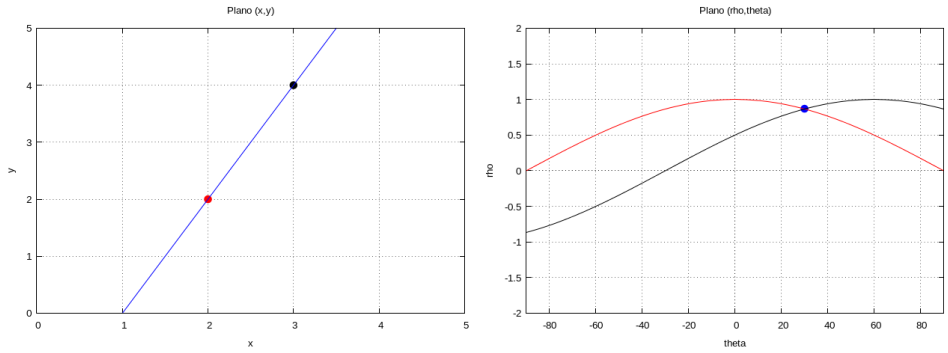

Figura 4.10: Transformada de Hough. Del plano (x, y) al plano (ρ, θ).

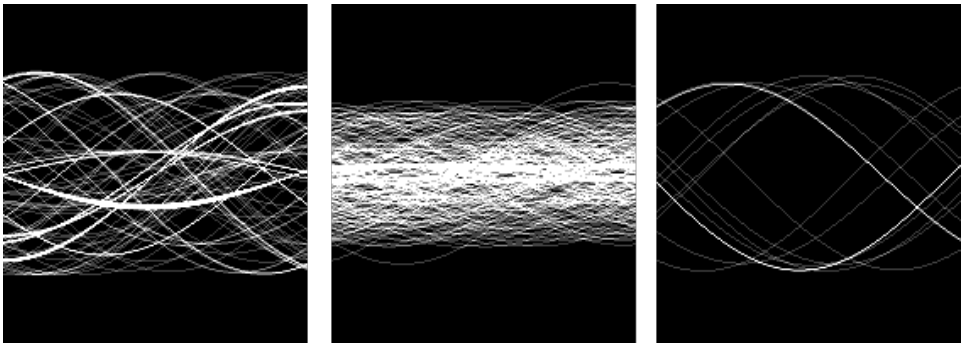

Figura 4.11: Diferentes espacios de Hough obtenidos en simulación.

El atractivo computacional de la Transformada de Hough proviene de la capacidad de poder subdividir el espacio de parámetros (ρ, θ) en las llamadas células acumuladoras, donde encontramos los límites (ρ_{min}, ρ_{max}) y $(\theta_{min}, \theta_{max})$, siendo el rango de $-90^o \leqslant \theta \leqslant 90^o$ y $-D \leqslant \rho \leqslant D$, donde D es la máxima distancia entre las esquinas opuestas de una imagen.

La célula en las coordenadas (i,j), qque tiene un valor de acumulador A(i,j), se corresponde cn la región asociada las coordenadas del espacio de parámetros (ρ_i, θ_j). Inicialmente las células tienen un valor nulo.

Entonces, para cada punto (x_k, y_k) no perteneciente al fondo del plano (x,y) hacemos θ igual a cada uno de los valores de subdivisión en el eje θ y resolvemos para el correspondiente valor de ρ usando la ecuación $x \cdot cos\theta + y \cdot sin\theta = \rho$. Los valores resultantes de ρ se redondean a la célula permitida más cercana a lo largo del eje ρ.

Si una elección θ_p resulta en una solución ρ_q, incrementamos en uno el

valor del acumulador A(p,q). Al final de este procedimiento, un valor de P en el acumulador A(i,j) implica que P puntos en el plano (x,y) satisfacen la ecuación $x \cdot cos\theta_j + y \cdot sin\theta_j = \rho_i$.

El número de subdivisiones en el plano (ρ, θ) determina la precisión en la colinearidad de dichos puntos. Se puede demostrar fácilmente, aunque cae fuera del ámbito del texto, que el número de iteraciones en el método discutido es lineal con respecto al número de puntos no pertenecientes al fondo en el plano (x,y).

Aunque hemos recogido la base teórica para la transformada de Hough para rectas, la transformada de Hough es aplicable para cualquier función de la forma $g(\overrightarrow{v}, \overrightarrow{c}) = 0$, donde $\overrightarrow{v}$ es un vector de coordenadas y $\overrightarrow{c}$ es un vector de coeficientes. Por ejemplo, si queremos encontrar puntos pertenecientes a una circunferencia aplicaríamos la misma discusión a la ecuación $(x - c_1)^2 + (y - c_2)^2 = c_3^2$.

En este caso, la diferencia estriba en este caso en la presencia de tres parámetros (c_1, c_2, c_3) que resultan en un espacio de parámetros tridimensional, con células en forma de cubo y acumuladores de la forma A(i,j,k). El procedimiento entonces consistiría en aumentar c_1 y c_2, resolviendo c_3 para la ecuación $(x - c_1)^2 + (y - c_2)^2 = c_3^2$, y aumentando el acumulador asociado a la tripleta (c_1, c_2, c_3).

Como es fácil intuir, la complejidad de la Transformada de Hough depende del número de coordenadas y coeficientes existentes en la representación funcional de partida.

Extracción de Características para el controlador

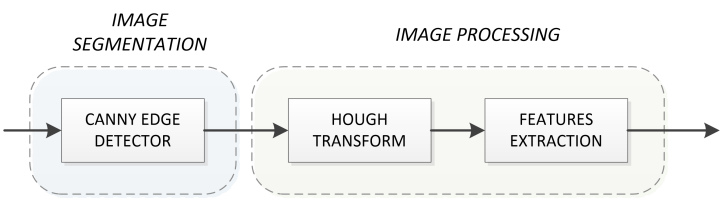

Figura 4.12: Bloque de segmentación y procesado de imagen.

En este apartado daremos las definiciones para las características que han sido medidas. Dichas características serán las variables de entrada del controlador, que definiremos en la sección 4.2.5.

Para facilitar el desarrollo del trabajo y evitar problemas con calibraciones de cámaras haremos las siguientes suposiciones:

- La correspondencia entre el eje central del vehículo y la imagen tomada es conocida, es decir, sabremos qué coordenada X de la imagen se corresponde con el eje central del vehículo y también con la posición teórica de las líneas de carril.
- La correspondencia entre el ancho del vehículo y el número de píxeles en horizontal en la cámara es también conocido.

Estas suposiciones nos permiten simplificar el modelo y definir las características de la trayectoria del vehículo a partir de la imagen capturada:

- **Error Lateral:** definido como la distancia horizontal existente entre el eje central del vehículo y el centro del carril detectado.
- **Error angular:** definido como el ángulo que se separa el eje central del vehículo respecto a la proyección del centro del carril detectado en el mismo sentido de la carretera, donde suponiendo que las líneas de los carriles son siempre perfectamente paralelas.

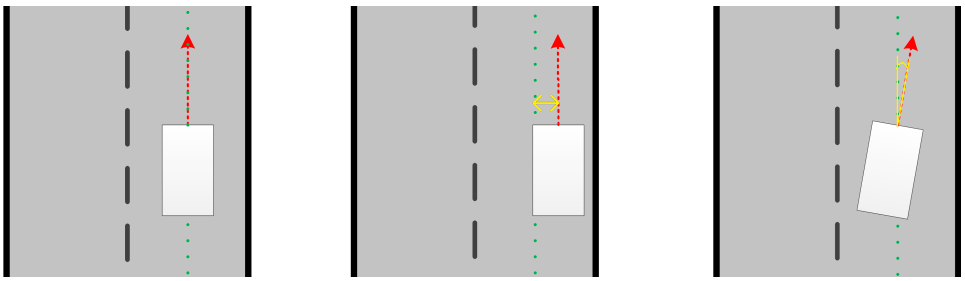

Figura 4.13: Extracción de características.

En la figura 4.13 se muestran ejemplos de las características extraídas. En la figura de la izquierda encontramos un vehículo perfectamente orientado, en el que tanto el error lateral como el angular son nulos. En la parte central se muestra un ejemplo del error lateral mientras que en la figura de la derecha se muestra un ejemplo del error angular. En todos los caso hemos representado en verde punteado el eje central del carril, con la línea roja la dirección del vehículo y en amarillo los respectivos errores medidos.

Se ha establecido el criterio de intentar obtener siempre las medidas respecto a la línea de carril derecha y en caso de no ser detectada, obtener la medida respecto a la línea de carril izquierdo, con el consiguiente cambio de signo en ambas medidas. Por simplificar el modelo nos es suficiente

con detectar correctamente una línea de carril, suponiendo la otra aproximadamente simétrica habiendo estimado anteriormente el ancho del carril respecto al ancho del vehículo.

4.2.2. Geolocation System

El sistema de geolocalización se basa en la API proporcionada por la SDK del sistema operativo Android [1].

En general, se utiliza un sistema de coordenadas estándar de 3 ejes para expresar los valores de los datos. Para la mayoría de los sensores, el sistema de coordenadas está definido de manera relativa a la pantalla del dispositivo en su orientación estándar, como muestra la figura 4.14.

Cuando un dispositivo se encuentra en su orientación por defecto, el eje X es horizontal y apunta hacia la derecha, el eje Y es vertical, apuntando hacia arriba mientras que el eje Z apunta hacia el exterior de la pantalla, así las coordenadas que se encuentran detrás de la pantalla tendrán valores negativos para el eje Z.

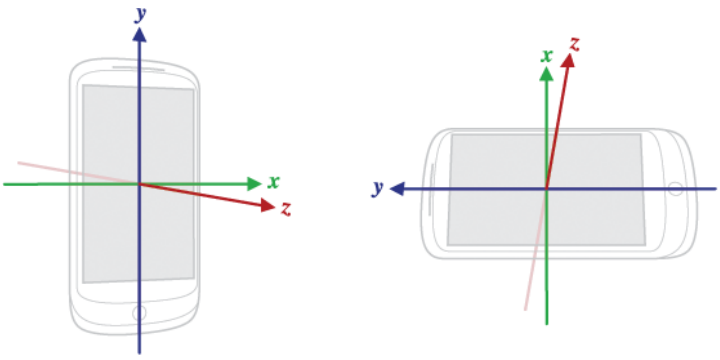

Figura 4.14: Sistema de Coordenadas de Sensores Terminal Android.

Es importante a la hora de interpretar los datos proporcionados por el dispositivo que los ejes nunca cambian independientemente de la orientación de la pantalla del dispositivo, es decir, el sistema de coordenadas nunca cambia conforme se mueve el dispositivo.

El último aspecto importante para la implementación de estos servicios es tener en cuenta que si una aplicación mapea los datos de los sensores con la pantalla del dispositivo, es necesario utilizar el método *getRotation()* para determinar la rotación de la pantalla, y luego usar el método *remapCoordinateSystem()* para transformar las coordenada del sensor a coorde-

nadas de la pantalla. Este procedimiento debe realizarse incluso si en el *manifest.xml* de la aplicación se especifica que la orientación de la aplicación es únicamente en modo retrato.

Para implementar cada uno de los módulos se ha desarrollado un *servicio* independiente. La documentación para desarrollo en Android define un servicio como un componente de aplicación que puede realizar operaciones en background sin proporcionar interfaz de usuario.

Los módulos implementados son:

- **ACCLoggerService.** El sensor de aceleración mide la aceleración aplicada al dispositivo, incluyendo la fuerza de la gravedad. En el siguiente ejemplo, vemos como configurar el sensor.

```
1 mSensorManager = (SensorManager) getSystemService(Context.
      SENSOR_SERVICE);
2 mAccelerometer = mSensorManager.getDefaultSensor(Sensor.
      TYPE_ACCELEROMETER);
3 mSensorManager.registerListener(this, mAccelerometer,
      SensorManager.SENSOR_DELAY_NORMAL);
```

Listing 4.1: *Configuración ACCLoggerService.*

  El acelerómetro utiliza el sistema de coordenadas por defecto ya explicado antes. y conceptualmente, un sensor de aceleración determina la aceleración aplicada al dispositivo A_d midiendo las fuerzas aplicadas al sensor mismo, usando la siguiente relación donde ya se contempla la influencia continua de la fuerza de la gravedad sobre el dispositivo.

$$A_d = -g - \sum F_s/mass$$

  Es decir, si el dispositivo está apoyado en una mesa sin moverse, el sensor de aceleración seguirá midiendo una magnitud de $9{,}81 m/s^2$. Por lo tanto, para medir la aceleración real del dispositivo, se debe eliminar la gravedad de los datos proporcionados por el acelerómetro, lo cual puede lograrse aplicando un filtro paso alto a los datos obtenidos.

- **GYRLoggerService.** El giróscopo mide la tasa o la rotación en rad/s en torno a los ejes X,Y,Z del dispositivo. Los giróscopos estándar proveen datos rotacionales brutos sin ningún tipo de filtrado or corrección por ruido o deriva. En la práctica, tanto el ruido como la deriva introducen

errores que deben ser compensados por la propia aplicación. El ruido y la deriva pueden ser determinados monitorizando otros sensores como el acelerómetro.

- **GPSLoggerService**. El sistema Android proporciona a las aplicaciones acceso a los servicios de localización del dispositivo a través del paquete *android.location*. El componente central de este paquete es el servicio del sistema *LocationManager*, el cual proporciona una API para determinar la ubicación y el rumbo del dispositivo.

  Para ello, la aplicación debe indicar que quiere recibir actualizaciones de localización del *LocationManager* llamando a *requestLocationUpdates()*, pasándole un *LocationListener* de la propia aplicación, que debe implementar métodos que serán llamados por el *LocationManager* cuando se actualiza la localización del dispositivo o cambia el estado del servicio. Se debe indicar el tipo de proveedor de localización a utilizar entre los disponibles (GPS, WiFi, Cell-ID) así como la frecuencia estimada a la que se quieren recibir las actualizaciones. En nuestro caso únicamente hemos utilizado el sistema GPS como fuente de datos para localización en exteriores.

- **ORILoggerService**. El vector de rotación representa la orientación del dispositivo como una combinación de ángulos y ejes, en la que el dispositivo ha rotado un ángulo θ sobre un eje X,Y o Z. Los tres elementos del vector de rotación son expresados como $x * sin(\theta/2)$, $y * sin(\theta/2)$, $z * sin(\theta/2)$.

  La magnitud del vector de rotación es igual a $sin(\theta/2)$, y su dirección es igual a la dirección del eje de rotación. Los tres elementos del vector de rotación son iguales a los tres últimos componentes de un quaternión $cos(\theta/2)$, $x * sin(\theta/2)$, $y * sin(\theta/2)$, $z * sin(\theta/2)$, careciendo de unidades. Los ejes X,Y,Z se definen como en el sistema de referencia estándar, pero el sistema de coordenadas de referencia se define como una base directa ortonormal, como muestra la figura 4.15, donde:

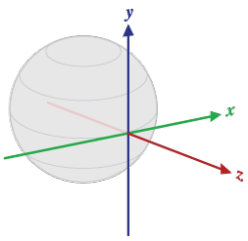

Figura 4.15: Sistema de Coordenadas Sensores de Orientación.

- X se define como vector producto Y x Z. Es tangencial a la superficie de la tierra en la localización aproximada del dispositivo apuntando aproximadamente hacia el este.
- Y es tangencial a la superficie de la tierra en la localización actual del dispositivo y apunta hacia el norte geomagnético.
- Z apunta directamente hacia el cielo y es perpendicular al plano de tierra.

4.2.3. Deliberative Layer

La capa deliberativa genera soluciones globales para tareas complejas utilizando planificación. Debido a la complejidad computacional necesaria para la generación de tales soluciones, su ciclo de decisión suele encontrarse en el rango de varios minutos. La capa deliberativa (o capa planificadora) usa modelos para la toma de decisiones. Estos modelos pueden ser presuministrados o aprendidos de los datos, y normalmente utilizan información de estado recolectada en la capa ejecutiva.

4.2.4. Executive Layer

La capa ejecutiva sirve como unión entre las capas reactivas y deliberativa. Acepta directivas de la capa deliberativa y las encadena para la capa reactiva. Por ejemplo, la capa ejecutiva podría manejar un conjunto de puntos de ruta generados por un planificador de rutas deliberativo y tomar decisiones como por ejemplo que comportamiento reactivo invocar. Los ciclos de decisión en la capa ejecutiva se encuentran normalmente en el orden del segundo de tiempo. La capa ejecutiva es también responsable de integrar la información de los sensores en una representación interna del esto. Por ejemplo, puede almacenar la localización del robot y rutinas de mapeado.

4.2.5. Reactive Layer

La capa reactiva provee de control de bajo nivel al robot, caracterizada por el ciclo de sensor-acción. Su ciclo de decisión se encuentra en el orden de los milisegundos.

La capa reactiva del robot en nuestro escenario es muy sencilla y está formada únicamente por el controlador basado en lógica difusa que toma como entradas los valores de las estimación de orientación y desplazamiento respecto a los bordes de la carretera obtenidas por el sistema de visión.

En este estadio del desarrollo el sistema es muy sencillo, pero puede hacerse algo más interesante si utilizara, tal y como muestra la figura 4.4, información o restricciones procedentes de las capas superiores.

Controlador FLC

La conducción de un vehículo es un problema de control especial dado que sus modelos matemáticos asociados son altamente complejos y son difícilmente linealizables con precisión. Se ha optado por utilizar un controlador basado en lógica difusa o Fuzzy Logic Controler (FLC) [30], ya que se ha comprobado en numerosas ocasiones que es un método apropiado para tratar con este tipo de sistemas, obteniendo buenos resultados a la par que permite incorporar conocimiento de procedimientos humanos a los algoritmos de control, es decir, nos permite imitar el comportamiento de la conducción humana hasta cierto punto [29].

El objetivo del **sistema de control de dirección** es seguir una determinada trayectoria, dada por los límites de la carretera. Para modelar el seguimiento de las desviaciones angulares y laterales percibidas por un conductor humano, utilizaremos dos variables difusas: **Lateral_Error** y **Angular_Error.**

Estas variables representan la diferencia entre la correcta y actual orientación y posición del vehículo respecto a una determinada referencia de trayectoria, que vendrá dada por los parámetros extraídos de la imagen de la carretera mediante el sistema de visión.

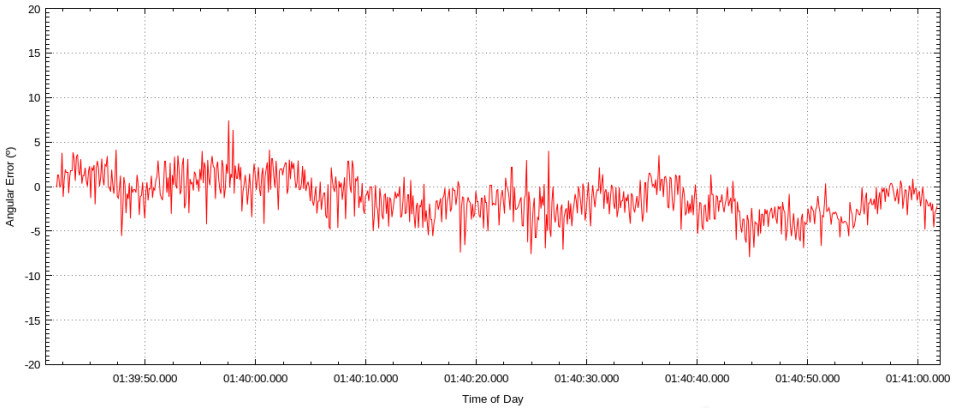

Figura 4.16: Valores de la variable Angular_Error obtenidas del sistema de visión aplicado al Dataset 4.

Ambas variables pueden tomar valores lingüísticos de izquierda o de-

recha. La variable *Angular_Error* representa el ángulo entre la orientación y el vector velocidad del vehículo. Si dicho ángulo tiene sentido antihorario, su valor es LEFT. Si el ángulo tiene sentido horario su valor lingüístico es RIGHT, como muestra la figura 4.17.

Destacar que para la realización de estos gráficos se ha utilizado la herramienta *jFuzzyLogic*, una librería de lógica difusa de código abierto orientada a facilitar el desarrollo de sistemas difusos [20] [19] y cuyo código se encuentra recogido en el apéndice B.

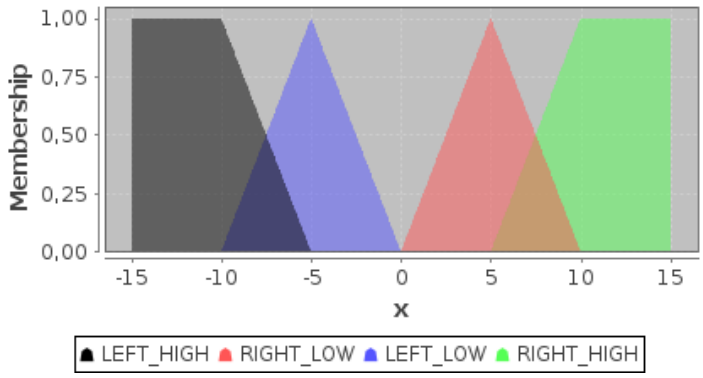

Figura 4.17: Variable de entrada difusa: *Lateral_Error*.

La variable *Lateral_Error* representa la distancia del vehículo a la trayectoria de referencia. Si el vehículo está posicionado a la izquierda de la trayectoria, Lateral_Error toma el valor LEFT mientras que si se encuentra a la derecha de la trayectoria tomará el valor RIGHT.

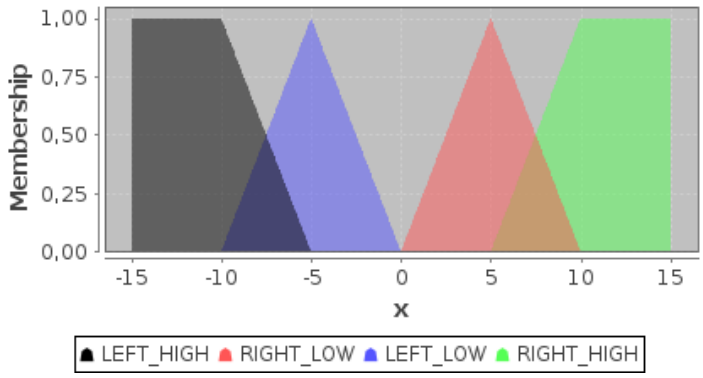

Figura 4.18: Variable de entrada difusa: *Angular_Error*.

- IF Lateral_Error IS LEFT_LOW OR Angular_Error IS LEFT_LOW THEN Steering_Wheel IS RIGHT_LOW;

- IF Lateral_Error IS RIGHT_LOW OR Angular_Error IS RIGHT_LOW THEN Steering_Wheel IS LEFT_LOW;
- IF Lateral_Error IS LEFT_HIGH OR Angular_Error IS LEFT_HIGH THEN Steering_Wheel IS RIGHT_HIGH;
- IF Lateral_Error IS RIGHT_HIGH OR Angular_Error IS RIGHT_HIGH THEN Steering_Wheel IS LEFT_HIGH;

A pesar de que las reglas son extremadamente sencillas, los resultados generados deben ser bastante cercanos a la conducción humana. Sí es importante para conseguir un control y por tanto una conducción suave establecer correctamente los valores límites para las funciones miembro de las variables difusas e incluso se puede proponer un modelo ligeramente más complejo donde se definan dos regiones de error a izquierda y derecha, como recogen las figuras 4.17, 4.18 y 4.19.

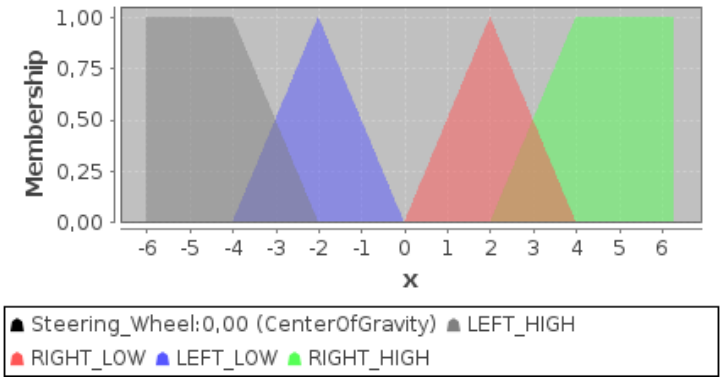

Figura 4.19: Variable de salida difusa: *Steering_Wheel*.

Respecto al control en velocidad, comentar que la implementación se basa en un controlador todo-nada, en el que mientras el seguimiento de la trayectoria sea correcto, se continua enviando el comando de velocidad mínima constante y en caso de perderse en la trayectoria se detiene el vehículo.

Dado que no se dispone de encoders para estimar la velocidad instantánea del vehículo y el GPS no es práctico para realizar un control en velocidad se ha preferido usar esa aproximación algo conservativa.

4.2.6. Interfaz gráfica

Para facilitar el desarrollo del TFM se ha implementado una interfaz gráfica que ha permitido desarrollar los algoritmos de visión en C++, usando

la librería openCV, en lugar de diseñar los algoritmos programando directamente sobre la plataforma Android.

Esta decisión ha venido determinada para mejorar el tiempo de desarrollo, pues con esta interfaz ha sido fácil cambiar los valores de los parámetros de los filtros en tiempo de ejecución, mostrar máscaras intermedias y realizar otro tipo de funciones que han mejorado mucho el desarrollo de los algoritmos.

Figura 4.20: Captura de pantalla de la interfaz gráfica desarrollada.

La interfaz permite ejecutar los algoritmos de visión sobre un dataset cualquiera, pero también permite ejecutarlos sobre los datos tomados por el dispositivo Android, para lo que se ha habilitado una función, tanto en la interfaz como en la aplicación del dispositivo que permite enviar las imágenes tomadas de la cámara del dispositivo a la aplicación para ser analizados.

La interfaz permite también controlar el robot, enviando comandos en velocidad y dirección al dispositivo Android, usando para ello un mando convencional de juego por USB o bien los comandos extraídos a partir de la información obtenida de la cámara. A continuación recogemos un listado con las principales características de la interfaz:

- Interfaz de usuario realizada en C++, basada en las librerías Qt, Marble y OpenCV. Ejecutado sobre PC convencional, con sistema operativo Linux, Ubuntu 14.04 en versiones de 32 y 64 bits.

- Implementación del módulo de visión descrito usando las librerías OpenCV. Incluye función de logueo de los valores obtenidos con marca de tiempo para su posterior tratamiento.

- Parametrización de los valores de los filtros usados en los algoritmos de visión en tiempo de ejecución.
- Capacidad para mostrar las imágenes intermedias y máscaras utilizadas.
- Implementación del módulo controlador FLC descrito, para poder controlar el robot. Incluye función de logueo con marca de tiempo de las señales de control generadas para su posterior tratamiento.
- Comunicación mediante TCP/IP con el dispositivo Android para:
  - *Función de logueo:* a través del puerto 4040 recibir toda la información proveniente del módulo de geolocalización del dispositivo (localización, orientación, etc.). Dicha información es guardada en un fichero de log incluyendo marcas de tiempo para su posterior análisis y tratamiento.

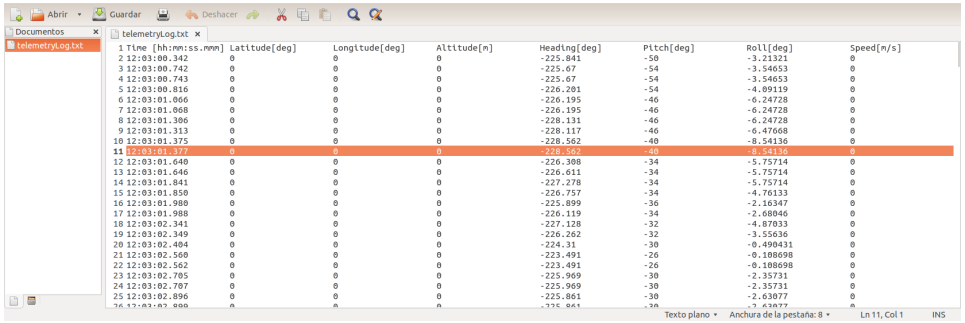

Figura 4.21: Fichero de log de Telemetría.

  - *Función de toma de imágenes:* a través del puerto 4042 la aplicación recibe las imágenes tomadas por el dispositivo situado en el robot para poder ejecutar los algoritmos de visión sobre ellas. La aplicación guarda los frames originales capturados, permitiendo así también generar datasets de imágenes sobre los que probar los distintos algoritmos de visión.
  - *Función de control del robot:* ya sea en modo manual utilizando el mando de control o bien usando el controlador automático FLC.
- Implementación de módulo de representación de la localización del robot usando la librería Marble, que usando los servicios proporcionados por el proyecto Open Street Map muestra la localización exacta del robot en el planeta.
- Implementación de módulo HUD, que muestra la información de orientación del robot en modo gráfico sobre las imágenes tomadas por el mismo.

4.3. Librerías software

4.3.1. Qt

Qt [5] es una librería multiplataforma, desarrollada inicialmente por la empresa Trolltech, y actualmente propiedad de Digia Plc [10] orientada principalmente al diseño de interfaces de usuario, aunque también admite desarrollo software sin interfaces gráficas.

Figura 4.22: Logo Qt.

Qt está actualmente soportado en una gran variedad de plataformas de 32 y 64 bits, y puede ser compilada en cada plataforma con el compilador GCC, ya sea uno suministrado el proveedor o por un tercero.

Desktop Platforms	Windows, Linux/X11, Mac OS X
Embedded Platforms	Ebedded Android, Embedded Linux,Windows Embedded (Compact and Standard),Real-Time Operating Systems, such as QNX, VxWorks and INTEGRITY
Mobile Platforms	Android, iOS, WinRT (including Windows Phone), Blackberry 10, Sailfish OS

Tabla 4.2: Plataformas soportadas por Qt.

Qt es licenciada bajo una licencia comercial y bajo licencias abiertas, concretamente la GNU General Public License (GPL) version 3 y GNU Lesser General Public License (LGPL) version 2.1.

Algunas de las principales características de la librería son:

- Basado en una librería de clases de C++ intuitiva.
- Portabilidad entre sistemas operativos empotrados y de escritorio.
- Herramientas integradas de desarrollo multiplataforma (IDE).
- Alto rendimiento en ejecución y poco espacio en disco y memoria en sistemas empotrados.

Qt se compone de diferentes módulos, donde cada uno de los cuales reside en una librería independiente. Según las funcionalidades que se quieran implementar en la aplicación se irán incluyendo dichos módulos o minilibrerías, los cuales se recogen en la tabla 4.3.

MÓDULO	**DESCRIPCIÓN**
QtCore	Clases de no GUI usadas por otros módulos
QtGui	Componentes de las interfaces gráficas de usuario
QtNetwork	Clases para programación de aplicaciones en red
QtOpenGL	Clases con soporte para OpenGL
QtSql	Clases para integración con SQL
QtScript	Clases para evaluación de Sripts Qt
QtSvg	Clases par amostrar el contenido de ficheros SVG
QtXml	Clases para el manejo de XML
QtDesigner	Clases para QtDesigner
QtUiTools	Clases para el manejo de formularios de QtDesigners en las aplicaciones
QtAssistant	Soporte para ayuda en Línea
Qt3Support	Clases con soporte de compatibilidad para Qt 3
QtTest	Clases con herramientas para pruebas

Tabla 4.3: Módulos Qt.

Los módulos principales en cualquier aplicación gráfica Qt son QtCore y QtGui, así el módulo QtCore forma parte de todas las ediciones de las librerías Qt, dependiendo el resto de módulos de la librería de este módulo.

En primer lugar incluye la definición del namespace <Qt>, donde se incluyen numerosos identificadores usados en toda la librería. Incluye asimismo la definición de numerosas clases y objetos que son básicas para usar en otros elementos o widgets del resto de la librería mientras que el módulo QtGui extiende las funcionalidades del módulo QtCore incorporando todas las definiciones de clases utilizadas para el desarrollo de interfaces gráficas de usuario.

Para incluir las definiciones de ambos módulos bastaría con la sentencia:

```
1 #include <QtCore>
2 #include <QtGui>
```

Comentar en último lugar que la versión más actual del desarrollo ha sido lanzada en su versión Qt 5.3 en el momento de escritura de este texto,

mientras que la aplicación de la Estación de Control y monitorización fue desarrollada con la versión 4.8.0 de la librería.

4.3.2. OpenCV

OpenCV (Open Source Computer Vision) [3] es una librería de código abierto para la programación de sistemas de visión por ordenador en tiempo real.

Figura 4.23: OpenCV Logo.

OpenCV está escrito en C y C++ y es multiplataforma, ejecutándose bajo Linux, Windows y Mac OS X. Actualmente existen desarrollos en activo trabajando en interfaces para Python, Ruby, Matlab y otros lenguajes de programación.

OpenCV fue diseñado para obtener eficiencia computacional en ejecución y con un fuerte enfoque para las aplicaciones en tiempo real. Está escrito en lenguaje optimizado en C y además puede beneficiarse del uso de procesadores multinúcleo.

Uno de los objetivos del proyecto OpenCV es proveer de una infraestructura de visión por computador fácil de usar, que facilite a los desarrolladores construir sofisticadas aplicaciones de visión de manera rápida y eficiente. La librería OpenCV contiene más de 500 funciones que incluyen muchas áreas de la visión por computador, tales como inspección en la fabricación de productos, imágenes médicas, seguridad, interfaces de usuario, calibración de cámaras, visión estéreo, así como aplicaciones específicas para robótica.

OpenCV está estructurado en cinco componentes principales, cuatro de las cuales se muestran en la figura 4.24.

El componente CV contiene el procesamiento base de imágenes y algoritmos de nivel superior de visión por computador. MLL (Machine Learning Library) es la librería de aprendizaje automático, que incluye a muchos clasificadores estadísticos y herramientas de clustering. HighGUI contiene rutinas de salida y funciones para el almacenamiento y carga de vídeo e

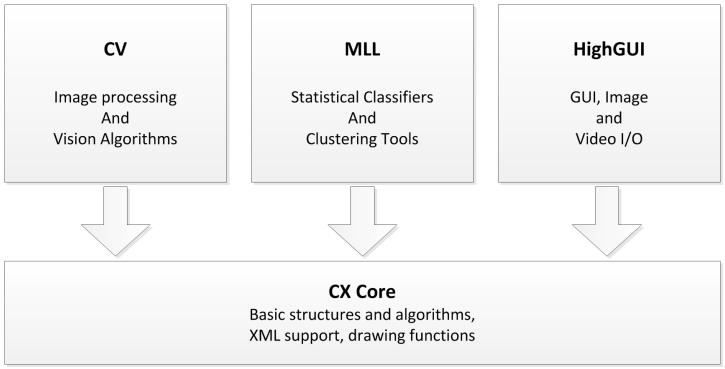

Figura 4.24: Estructura librería OpenCV.

imágenes, y CXCore contiene las estructuras de datos básicos y el contenido.

La figura 4.24 no incluye el último módulo CvAux, que contiene áreas abandonadas del proyecto (integrado HMM[1] y reconocimiento facial) así como otros algoritmos experimentales (segmentación de fondo/primer plano).

4.3.3. Marble

Marble [13] es una aplicación implementada en Qt que muestra una vista de la Tierra con multitud de plugins y módulos listos, pero que ha sido también liberada de forma que se pueden integrar en otras aplicaciones Marble widgets como cualquier otro widget de Qt.

La librería ha sido diseñada de forma que sus componentes sean fáciles de usar, potenciando cualquier aplicación integrando los geoservicios que ofrece Marble.

Las principales clases y aplicaciones de la librería son:

- **MarbleWidget**, es el widget principal en cualquier aplicación que use la librería Marble. En él se muestra una vista del globo terráqueo (o de cualquier otro planeta) por el cual el usuario puede navegar utilizando algún widget de control como por ejemplo MarbleControlBox, o el propio ratón.
- **MarbleModel**, es la clase de almacenamiento de datos que guarda los datos visualizados en le MarbleWidget. Es una clase creada internamente que puede ser accedida mediante el método *model()*.

[1] Hidden Markov Models.

Este modelo de datos contiene datos heterogéneos, como las imágenes que forman el fondo del mapa, vectores con la información de fronteras, lineas de costa, puntos de interés como montañas, ciudades, etc.

- **MarbleControlBox**, es el widget de control avanzado para el MarbleWidget. Se puede usar para navegar alrededor del globo, buscar elementos, elegir la información a renderizar, y muchas más opciones.

Estas tres clases proporcionan el núcleo de las clases que conforman la librería Marble. Para la aplicación de interfaz de usuario se ha usado la versión 0.18.3 estable de la librería y únicamente se ha utilizado la clase MarbleWidget, creando una clase que la extiende y que permite indicar la posición y orientación del robot en cada instante.

La librería está escrita en C++ y proporciona bindings para Qt Quick (QML) y Python, siendo realmente sencillo integrarla en otra aplicación Qt o Python, como muestra el ejemplo 4.2.

```
#include <QtGui/QApplication>
#include <marble/MarbleWidget.h>

int main(int argc, char** argv)
{
    QApplication app(argc, argv);

    // Load Marble using OpenStreetMap in Mercator projection
    Marble::MarbleWidget *mapWidget = new Marble::MarbleWidget;
    mapWidget->setProjection(Marble::Mercator);
    mapWidget->setMapThemeId("earth/openstreetmap/openstreetmap
        .dgml");

    mapWidget->setWindowTitle("Hello Marble!");
    mapWidget->show();
    return app.exec();
}
```

Listing 4.2: *Aplicación Qt básica con MarbleWidget.*

4.3.4. Microbridge

MicroBridge [14] es una implementación del Android Debug Bridge (ADB) orientada a su uso con microcontroladores. MicroBridge permite a dispositivos Android convencionales, sin acceso administrador, comunicarse de manera directa con unidades microcontroladoras (MCU) con capacidades

Figura 4.25: Hello Marble

de host USB, permitiendo por tanto que los dispositivos Android puedan actuar servos, motores DC, etc.

La librería MicroBridge funciona con dispositivos android a partir de su versión 1.5 y hasta la versión 4.2.1. En la versión 4.2.2 de Android, por defecto se securizaron las conexiones USB mediante un cifrado RSA y la librería MicroBridge no ha sido actualizada.

El protocolo ADB soporta la apertura de un shell en el teléfono o el envío de comandos de shell directos, subiendo archivos, leyendo ficheros de logs (logcat) y reenviando puertos TCP y sockets Unix. Usando sockets TCP es posible establecer líneas de comunicación bidireccionales entre una placa Arduino y un dispositivo Android. Así, la aplicación Android escucha en un puerto, al cual se conecta la placa Arduino mediante ADB.

Si la conexión se pierde, bien por fallo de la aplicación o por desconexión del dispositivo USB, la librería MicroBridge periódicamente trata de reconectar hasta que el enlace es finalmente recuperado.

La librería MicroBridge es completamente funcional y permite hacer prácticamente lo mismo que la línea de comandos de ADB.

Capítulo 5

Resultados

5.1. Resultados en simulación con datasets públicos

En la tabla 5.1 recogemos los resultados principales del comportamiento del extractor de características del módulo de visión para los distintos datasets.

Dataset	Resolution	Frames	BLD	RLD	LLD	ND	APT[ms]	ACC[%]
1-cordova1	640x480	250	183	38	10	19	71.0536	92.4
1-cordova2	640x480	406	128	13	176	89	62.8805	78.0788
1-washington1	640x480	337	211	52	37	37	82.5303	89.0208
1-washington2	640x480	232	177	19	9	27	89.1139	88.3621
2-highway_qcif	176x144	2000	742	897	104	256	10.4127	87.15
3-viov_amelie	320x240	800	178	141	472	9	21.7352	98.875

Tabla 5.1: Resultados Datasets. ***(BLD, Both Lanes Detected. RLD, Right Lane Detected. LLD, Left Lane Detected. ND, No Lane detected). APT, Average Processing Time. ACC, Accuracy.***

Como vemos en dicha tabla, la precisión, entendida como el tanto por ciento de frames en los que hemos detectado al menos una de las líneas de nuestro carril, es bastante alta para la mayoría de los datasets, funcionando especialmente bien en el dataset 3.

El primer detalle importante que podemos extraer de estos resultados es la influencia directa en el tiempo de procesado promedio o Average Processing Time (APT) del tamaño de la imagen. La resolución clásica VGA de 640x480 píxeles parece excesiva para poder ser utilizada en un controlador, a falta por supuesto de realizar una optimización intensiva el código ejecutado. Como se extrae de los resultados del dataset 3, parece que la

resolución 320x240 es la solución de compromiso en términos de tiempo de procesado, resolución de las imágenes y resultados obtenidos.

Por otro lado, en la figura 5.1 representamos los valores promedios y la desviación estándar de las variables difusas obtenidas por el extractor, que merecen un análisis detallado.

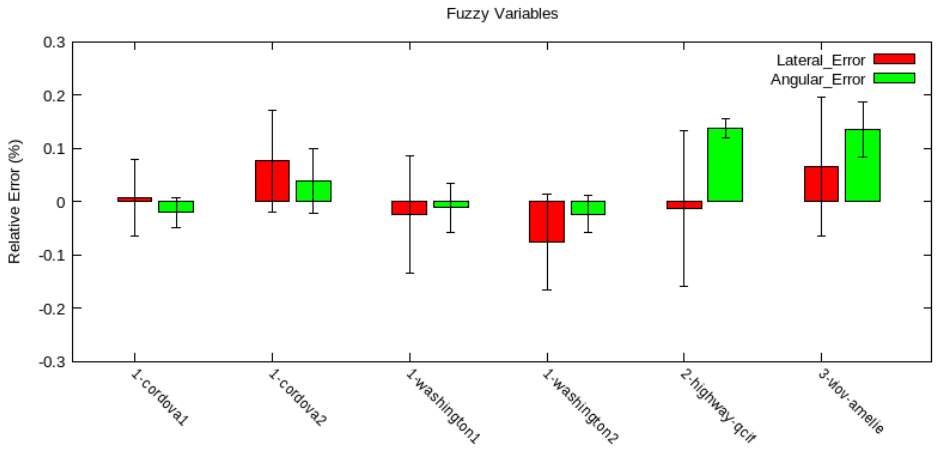

Figura 5.1: Extracción de características. Estadísticas Variables Difusas.

Lo primero que observamos es que los resultados obtenidos para las cuatro secuencias del dataset 1 son bastante dispersos. Los motivos de dicha dispersión pueden ser muy variados, siendo especialmente grave en el caso del error lateral. El dataset 1 corresponde a unas secuencias grabadas en entornos urbanos y por tanto aparecen una serie de problemas a solucionar en las imágenes tales como sombras de árboles y edificios, pavimento agrietado, ausencia de líneas de carril por tramos, señales de tráfico pintadas sobre el asfalto que hacen que los datos obtenidos tengan una dispersión alta.

Respecto a los resultados del dataset 2 destacamos en primer lugar su buen comportamiento a pesar de baja resolución de las imágenes de entrada, encontrándose algunas de ellas incluso pixeladas, ya que la fuente original de datos era un vídeo en formato YUV al que hubo que extraer los frames individuales. Los errores promedios obtenidos son muy bajos y con una baja dispersión para el error angular, algo que no ocurre con el error lateral. Esta alta dispersión en el error lateral puede deberse a la baja resolución y pixelación de las imágenes.

El último dataset presenta una precisión altísima, del 98.875 % y al igual

Figura 5.2: Imagen ejemplo dataset 1. Resolución 640x480.

Figura 5.3: Imagen ejemplo dataset 2. Resolución 176x144.

que el anterior, presenta un gran comportamiento en error angular aunque una dispersión alta en el error lateral. Este error puede deberse al hecho de que existe un desplazamiento lateral de la cámara hacia la izquierda del coche y al haber supuesto simetrías en nuestro modelo, la variabilidad de los errores medidos puede aumentar, ya que en esta secuencia, como se comprueba en la tabla 5.1 en la mayoría las ocasiones se ha utilizado el carril izquierdo como referencia, y al existir el desplazamiento de la cámara, aumenta la dispersión del conjunto en las ocasiones en las que se usa el carril derecho como referencia.

Figura 5.4: Imagen ejemplo dataset 3. Resolución 320x240.

Es importante destacar que en todos los datasets han aparecido detecciones de carril espúreas por los problemas de las imágenes ya comentados. Como medida para combatir estos errores se ha establecido un límite del 30 % en los errores, desestimando cualquier carril detectado con alguno de los errores por encima de esta cifra.

5.2. Resultados de experimentación

Por motivos de tiempo y logística, los experimentos para comprobar la viabilidad del prototipo y por tanto del controlador FLC no han podido ser ejecutados para la fecha de entrega de la memoria.

Capítulo 6

Conclusiones y Líneas Futuras

6.1. Conclusiones

Tras la implementación de los sistemas y, sobre todo, tras la realización de los distintos experimentos, se han podido extraer las siguientes conclusiones del presente TFM.

- **Primera:** El sistema de visión en su estado actual ha demostrado ser muy robusto para realizar su tarea en los datasets correspondientes a vías interurbanas (tipo autovía o carretera) e incluso obtener unos resultados aceptables en vías urbanas, donde el entorno es mucho más aleatorio y confuso donde se han detectado la presencia de excesivos artefactos (líneas de carril inverosímiles) en la detección de los carriles. Más allá de la dispersión obtenida en los resultados, hemos de fijarnos en los datos de la precisión obtenida que es suficientemente alta y que permite diseñar el controlador FLC sin más que ajustar los parámetros de su rango de trabajo.

- **Segunda:** El sistema de visión parece ser lo suficientemente rápido como para ser integrado en una plataforma completa y aplicado como entrada al controlador difuso descrito en la memoria.

- **Tercera:** la arquitectura modular del sistema permite el desarrollo en paralelo de los diferentes sistemas y la fácil ampliación del mismo desarrollando nuevos módulos independientes.

6.2. Líneas futuras

Tas finalizar los trabajos correspondientes a los objetivos del TFM, surgen de manera natural muchas líneas de trabajo e investigación que permiten continuar y ampliar el trabajo realizado. Por citar algunas:

- *Ejecución de los algoritmos de testeo del prototipo.* Como ya hemos comentado, a 5 de diciembre de 2014, fecha de entrega de la presente memoria del TFM, ha sido físicamente imposible realizar los experimentos de navegación, requisito casi indispensable para comprobar la viabilidad del robot prototipo.

- *Implementación del sistema completo utilizando únicamente la plataforma Android.* Actualmente, para facilitar el desarrollo del prototipo, los algoritmos de visión y control han sido desarrollados en un PC basado en Linux, usando concretamente la distribución Ubuntu 14.04 (en arquitecturas tanto de 32 como de 64 bits). Asi mismo, como ya se ha comentado, se introduce un retraso en el sistema al realizar el procesado de las imágenes en el PC que ejecuta la Ground Control Station, al tener que enviar desde el terminal Android las imágenes al PC y luego tener que recibir las actuaciones de control. El sistema debería mejorar notablemente el funcionamiento evitando este envío y recepción de información.

- *Implementación completa de las capas ejecutiva y deliberativa.* Estas capas aún no han sido implementadas, por lo que la inteligencia del robot es muy reducida. La implementación de estas capas permitiría al robot recorrer trayectorias y ejecutar funciones más complejas e incluso interactuar en un sistema con múltiples robots, redes de sensores inalámbricas (WSN).

- *Implementación completa de la estación de control (Ground Control Station)*. La GCS desarrollada es una versión de desarrollo, centrada en poder visualizar una cantidad alta de información y poder parametrizar correctamente los parámetros de los algoritmos de visión y control. Sería interesante desarrollar una GCS enfocada a un usuario final para que pudiese interactuar con el vehículo autónomo de forma natural, pudiendo definir un trayecto a recorrer por el vehículo, integración con distintos tipos de servicios web, etc.

- *Implementación del módulo de visión utilizando otras estrategias.* En este caso hemos planteado el uso de una región de la zona capturada, transformándola mediante IPM para poder aplicar la transformada de Hough y detectar los bordes pintados de la carretera. Sin embargo esta estrategia es inútil en ciertos escenarios como carreteras sin señalizar, carreteras de travesía o sin asfaltar, o interiores de edificios. El uso de otras técnicas, como cambio de contrastes, uso del espacio de colores y otras, para detectar la zona por la que puede moverse el robot puede dotar de una gran versatilidad a la plataforma.

Apéndice A

Datasets públicos de imágenes y videos de carreteras

Agradecer al doctor Marcos Nieto por haber enlazado en blog personal este completo listado de datasets públicos[1].

- **Dataset 1.** Caltech California Institute of Technology.
  `http://www.vision.caltech.edu/malaa/research/lane-detection`
- **Dataset 2.** YUV Arizona State University.
  `http://trace.eas.asu.edu/yuv`
- **Dataset 3.** IJRS International Journal of Robotics Research.
  `http://www.ijrr.org/contents/23_04/abstract/513.html`

[1] `http://marcosnietoblog.wordpress.com`

Apéndice B

Código Controlador Difuso FLC

```
FUNCTION_BLOCK Ardubot_Controller        // Block definition

VAR_INPUT                               // Define input variables
        Lateral_Error : REAL;
        Angular_Error : REAL;
END_VAR

VAR_OUTPUT                              // Define output variable
        Steering_Wheel : REAL;
END_VAR


// Fuzzify input variable 'Lateral_Error':
// { 'LEFT_HIGH', 'LEFT_LOW', 'RIGHT_LOW', 'RIGHT_HIGH' }

FUZZIFY Lateral_Error
        TERM LEFT_HIGH := (-15,1) (-10,1)(-5,0) ;
        TERM LEFT_LOW := (-10, 0) (-5,1)(0,0) ;
        TERM RIGHT_LOW := (0, 0) (5,1) (10,0);
        TERM RIGHT_HIGH := (5, 0) (10,1) (15,1);
END_FUZZIFY

// Fuzzify input variable 'Angular_Error':
// { 'LEFT_HIGH', 'LEFT_LOW', 'RIGHT_LOW', 'RIGHT_HIGH' }
FUZZIFY Angular_Error
        TERM LEFT_HIGH := (-15,1) (-10,1)(-5,0) ;
        TERM LEFT_LOW := (-10, 0) (-5,1)(0,0) ;
        TERM RIGHT_LOW := (0, 0) (5,1) (10,0);
        TERM RIGHT_HIGH := (5, 0) (10,1) (15,1);
END_FUZZIFY

// Defuzzify output variable 'Steering_Wheel':
// { 'LEFT_HIGH', 'LEFT_LOW', 'RIGHT_LOW', 'RIGHT_HIGH' }
DEFUZZIFY Steering_Wheel
        TERM LEFT_HIGH := (-6,1) (-4,1)(-2,0) ;
        TERM LEFT_LOW := (-4, 0) (-2,1)(0,0) ;
```

```
37          TERM RIGHT_LOW := (0, 0) (2,1) (4,0);
38          TERM RIGHT_HIGH := (2, 0) (4,1) (6,1);
39          METHOD : COG;    // Center Of Gravity
40          DEFAULT := 0;    // Default value is 0
41  END_DEFUZZIFY
42
43  RULEBLOCK No1
44          AND : MIN;       // Use 'min' for 'and'
45          ACT : MIN;       // Use 'min' activation method
46          ACCU : MAX;      // Use 'max' accumulation method
47
48          RULE 1 : IF Lateral_Error IS LEFT_LOW OR Angular_Error
                IS LEFT_LOW THEN Steering_Wheel IS RIGHT_LOW;
49          RULE 2 : IF Lateral_Error IS RIGHT_LOW OR Angular_Error
                 IS RIGHT_LOW THEN Steering_Wheel IS LEFT_LOW;
50          RULE 3 : IF Lateral_Error IS LEFT_HIGH OR Angular_Error
                 IS LEFT_HIGH THEN Steering_Wheel IS RIGHT_HIGH;
51          RULE 4 : IF Lateral_Error IS RIGHT_HIGH OR
                Angular_Error IS RIGHT_HIGH THEN Steering_Wheel IS
                LEFT_HIGH;
52  END_RULEBLOCK
53
54  END_FUNCTION_BLOCK
```

Listing B.1: *Código Controlador Difuso FLC*

ACRÓNIMOS

ACCEL	Accelerometer
ABS	Antilock Brake System
ACC	Adaptive Cruise Control
AC/DC	Alternating Current/Direct Current
ADB	Android Debug Bridge
ADC	Analog-to-Digital Converter
ADK	Android Accessory Development Kit
AEB	Autonomous Emergency Braking
AFS	Advanced Front-lighting Systems
API	Application Programming Interface
ASR	Anti-Slip Regulation
AUV	Autonomous Underwater Vehicle
BLIS	Blind Spot Information System
DAS	Driver Assistance Systems
DARPA	Defense Advanced Research Projects Agency
DDS	Data Distribution Service
EBD	Electronic Brakeforce Distribution
EEPROM	Electrically Erasable Programmable Read Only Memory
ESC	Electronic Speed Control
ESP	Electronic Stability Program
FZL	Fuzzy Logic Controller
GCC	GNU Compiler Collection
GCS	Ground Control Station
GPL	GNU General Public License
GPS	Global Positioning System
GUI	Graphic User Interface
GYRO	Gyroscope
HUD	Head Up Display
I2C	Inter-Integrated Circuit
ICSP	In Circuit Serial Programming
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
IP	Internet Protocol
IPM	Inverse Perspective Mapping
KDE	K-Desktop Environment
LDW	Lane Departure Warning
LGPL	GNU Lesser General Public License
LIDAR	Laser Imaging Detection and Ranging
NHTSA	National Highway Traffic Safety Administration

NRS	Network Robot System
PC	Personal Computer
PWM	Pulse Width Modulation
QML	Qt Meta Language
ROI	Region Of Interest
SDK	Software Development Kit
SLAM	Simultaneous Localization And Mapping
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
TCP	Transmission Control Protocol
TFM	Trabajo Fin de Master
TSR	Traffic Sign Recognition
UART	Universal Asynchronous Receiver-Transmitter
UAV	Unmaned Aerial Vehicle
UDP	User Datagram Protocol
UGV	Unmaned Ground Vehicle
UI	User Interface
USB	Universal Serial Bus
VGA	Video Graphics Array
VOIP	Voice Over IP
WSN	Wireless Sensor Networks

Bibliografía

[1] The android sdk. [Online]. Available: http://developer.android.com/sdk/index.html

[2] Arduino, a tool for making computers that can sense and control. [Online]. Available: http://www.arduino.cc/es/

[3] Opencv. librería para visión por computador. [Online]. Available: http://opencv.org

[4] Proyecto urus: Ubiquitous networking robotics in urban settings. [Online]. Available: http://www.urus.upc.es

[5] (2008) Qt. documentación de referencia. [Online]. Available: http://qt-project.org/

[6] (2012) Programa autopia. [Online]. Available: http://www.car.upm-csic.es

[7] (2013) U.s. department of transportation releases policy on automated vehicle development. [Online]. Available: http://www.nhtsa.gov/About+NHTSA/Press+Releases/U.S.+Department+of+Transportation+Releases+Policy+on+Automated+Vehicle+Development

[8] (2014) Adaptive cruise control systems with stop-and-go. [Online]. Available: http://msc.berkeley.edu/past_research/automotive/acc.html

[9] (2014) Arduino mega 2560 v.3. [Online]. Available: http://store.arduino.cc/product/A000067

[10] (2014) Digia plc company. [Online]. Available: http://www.digia.com/

[11] (2014) Google self-driving car project. [Online]. Available: https://plus.google.com/+GoogleSelfDrivingCars/

[12] (2014) Inverse perspective mapping (ipm) project. [Online]. Available: http://ipmapping.sourceforge.net/

[13] (2014) Marble library. [Online]. Available: https://techbase.kde.org/Projects/Marble

[14] (2014) Microbridge. android debug bridge (adb) implementation for microcontrollers. [Online]. Available: https://code.google.com/p/microbridge/

[15] (2014) Tinkerkit mega sensor shield v.2. [Online]. Available: http://store.arduino.cc/product/T020040

[16] M. Aly, "Real time detection of lane markers in urban streets," in *IEEE Intelligent Vehicles Symposium*, 2008.

[17] J. Barraquand and J.-C. Latombe, "Robot motion planning. a distributed representation approach," *International Journal of Robotics Research*, vol. 10, no. 6, pp. 628–649, 1991, cited By (since 1996)352. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-0026369737&partnerID=40&md5=990b3b9ab14ddeb8f6da96143bfe8a9b

[18] I. Buciu, A. Gacsádi, and C. Grava, "Vision based approaches for driver assistance systems," 2010, pp. 92–97, cited By (since 1996)2. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-79952613753&partnerID=40&md5=8d241bcc8ee3ef080fa7bdd238148a74

[19] P. Cingolani and J. Alcala-Fdez, "jfuzzylogic: a robust and flexible fuzzy-logic inference system language implementation," in *Fuzzy Systems (FUZZ-IEEE), 2012 IEEE International Conference on*. IEEE, 2012, pp. 1–8.

[20] P. Cingolani and J. Alcalá-Fdez, "jfuzzylogic: a java library to design fuzzy logic controllers according to the standard for fuzzy control programming," pp. 61–75, 2013.

[21] E. Eckermann, *World History of the Automobile*. SAE Press, 2001.

[22] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006.

[23] P. Grisleri and I. Fedriga, "The braive platform," in *Procs. 7th IFAC Symposium on Intelligent Autonomous Vehicles*, 2010, pp. 497–502. [Online]. Available: http://vislab.it/

[24] F. Gómez-Bravo, F. Cuesta, and A. Ollero, "Parallel and diagonal parking in nonholonomic autonomous vehicles," *Engineering Applications of Artificial Intelligence*, vol. 14, no. 4, pp. 419–434, 2001, cited By (since 1996)33. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-0035435172&partnerID=40&md5=703992dfe87505a1ed312f6cbcd02f3b

[25] P. V. Hough, "Method and means for recognizing complex patterns," Dec. 18 1962, uS Patent 3,069,654. [Online]. Available: http://www.google.co.in/patents/US3069654

[26] M. Kim, Y. Heled, I. Asher, and M. Thompson, "Comparative analysis of laws on autonomous vehicles in the u.s. and europe," vol. 1, 2014, pp. 740–751, cited By (since 1996)0. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84907057687&partnerID=40&md5=7076669a9801c3de8c26575baf28ff3e

[27] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, and S. Thrun, "Towards fully autonomous driving: Systems and algorithms," 2011, pp. 163–168, cited By (since 1996)23. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-79960817711&partnerID=40&md5=c151fcf1cf795b638cd5101f396c5271

[28] A. Muad, A. Hussain, S. Samad, M. Mustaffa, and B. Majlis, "Implementation of inverse perspective mapping algorithm for the development of an automatic lane tracking system," in *TENCON 2004. 2004 IEEE Region 10 Conference*, vol. A, Nov 2004, pp. 207–210 Vol. 1.

[29] J. Naranjo, C. González, R. García, T. De Pedro, and M. Sotelo, "Using fuzzy logic in automated vehicle control," *IEEE Intelligent Systems*, vol. 22, no. 1, pp. 36–45, 2007, cited By (since 1996)28. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-33847647149&partnerID=40&md5=e71e5a3d9f4d6eb54d2ff06e2d5ce5a5

[30] N. Nedjah and L. de M. Mourelle, "Introducing you to fuzziness," in *Fuzzy Systems Engineering*, ser. Studies in Fuzziness and Soft Computing, N. Nedjah and L. d. Macedo Mourelle, Eds. Springer Berlin Heidelberg, 2005, vol. 181, pp. 1–21. [Online]. Available: http://dx.doi.org/10.1007/11339366_1

[31] A. Ollero, *Robótica : manipuladores y robots móviles*. Marcombo, 2006.

[32] S. J. Russell, P. Norvig, J. F. Candy, J. M. Malik, and D. D. Edwards, *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.

[33] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney, "Stanley: The robot that won the

darpa grand challenge," *Journal of Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006, cited By (since 1996)499. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-33750024797&partnerID=40&md5=fef675e6b0d4a1fd7c94b99797e934ae