

TRABAJO DE FIN DE MASTER EN I.A. AVANZADA

**DEPARTAMENTO DE INTELIGENCIA ARTIFICIAL
UNED, 2010**

DESARROLLO DE UN SISTEMA DE VISIÓN PARA LA CONDUCCIÓN AUTOMÁTICA, APLICANDO EL MÉTODO DEL CENTRO DE ÁREAS

Fernando Aparicio Galisteo

ÍNDICE

1	RESUMEN.....	4
2	INTRODUCCIÓN.....	6
2.1	Hechos históricos relevantes.....	6
2.2	Paradigmas	6
3	DESCRIPCIÓN DEL PROBLEMA.....	9
3.1	Motivación y enfoque	9
3.1.1	Método general del centro de áreas.....	9
3.1.2	Método parcial del centro de áreas	12
3.1.3	El método parcial del centro de áreas aplicado al sistema de visión	15
3.1.4	Predicción en situaciones de discontinuidad: Algoritmo de Kalman	17
3.1.5	Métodos geométricos basados en la perspectiva plana.....	18
3.2	Estructura de la memoria	19
4	DESCRIPCIÓN DEL ENTORNO DE TRABAJO	21
4.1	Características del robot.....	21
4.2	Descripción general de las librerías utilizadas.....	22
4.2.1	Robótica	22
4.2.2	Visión artificial.....	23
4.3	Software para la simulación en Windows	23
5	DESCRIPCIÓN DE LOS MÓDULOS DESARROLLADOS	25
5.1	Estructura general.....	25
5.2	Conjuntos cliente-servidor utilizados.....	25
5.2.1	Pruebas de comunicación con el software de MobileRobots.....	25
5.2.2	Cliente y servidor desarrollados a medida: <i>ModuloRoboticaV0</i>	25
5.2.3	Servidor desarrollado a medida: <i>ModuloRoboticaV1</i>	26
5.2.4	Servidor desarrollado a medida: <i>ModuloRoboticaV2</i>	27
5.2.5	Servidor y Cliente desarrollados a medida: <i>ModuloRoboticaV3</i>	28
5.2.6	Servidor y Cliente desarrollados a medida: <i>ModuloRoboticaV4</i>	30
5.3	Aplicación para el tratamiento de imágenes	32
5.3.1	Etapas de preprocesado	33
5.3.2	Etapas de segmentación y selección de objetos.....	36
5.3.3	Etapas de seguimiento	37
6	PROGRAMACIÓN ORIENTADA A OBJETOS	43
6.1	Aplicación a sistemas reactivos	43
6.2	Aplicación a sistemas deliberativos e híbridos	48
6.3	Aplicación a sistemas de visión	50
7	MÉTODO DEL CENTRO DE ÁREAS Y CONDUCCIÓN AUTÓNOMA	53
7.1	Desarrollo del método del centro de áreas o baricentro	53
7.2	Movimiento hacia el centro de áreas	55
7.3	Conducción autónoma apoyada en el sistema de visión.....	56
8	PRUEBAS Y ANÁLISIS DE LOS RESULTADOS.....	59
8.1	Traspaso de puertas centradas	59
8.1.1	Método general del centro de áreas.....	59
8.1.2	Método parcial del centro de áreas	60
8.2	Traspaso de puertas no centradas	60
8.2.1	Método general del centro de áreas.....	60
8.2.2	Método parcial del centro de áreas	62
8.3	Salida de pasillos	62
8.3.1	Método general del centro de áreas.....	62
8.3.2	Método parcial del centro de áreas	63
8.4	Trazabilidad hacia zonas de espacio libre.....	64
8.4.1	Método general del centro de áreas.....	64
8.4.2	Método parcial del centro de áreas	66
8.5	Situaciones de posicionamiento en zonas ocupadas	66
8.6	Comportamiento ante posibles fallos de los sensores	69
8.7	Pruebas de visión artificial	70

8.7.1	Pruebas de preprocesado de imágenes	70
8.7.2	Pruebas de segmentación de imágenes para la detección de objetos ..	74
8.7.3	Pruebas de seguimiento como contingencia ante discontinuidades	78
9	CONCLUSIONES Y TRABAJO FUTURO.....	84
10	APÉNDICE I: INSTALACIÓN DEL SOFTWARE PARA LA SIMULACIÓN EN WINDOWS.....	86
10.1	Comunes.....	86
10.2	Robótica.....	86
10.3	Visión artificial	87
11	APÉNDICE II: EJECUCIÓN DE LA SIMULACIÓN DEL ROBOT	88
12	APÉNDICE III: INSTALACIÓN Y EJECUCIÓN DE <i>RESEARCH ARTIFICIAL VISION TOOL</i>	91
13	REFERENCIAS	95

1 RESUMEN

El objetivo de este trabajo es la obtención de mecanismos que posibiliten la conducción autónoma de un robot en situaciones reales. Para lograr cumplir una tarea tan ambiciosa, se ha tomado la decisión de dividirla en objetivos menores:

- I. En primer lugar, el desarrollo y estudio del método general de cálculo del centro de áreas, tratando de darle continuidad al trabajo realizado en [1], donde se introduce por primera vez el concepto, continuándose en [2], [3], [4] y [6], tratado como una técnica reactiva que, complementando a otro tipo de sistemas de navegación que usen conocimiento externo, permita el movimiento autónomo del robot. Otra referencia importante se puede encontrar en [5], donde se presenta esta idea en el contexto de un sistema de apoyo a la movilidad de personas invidentes hacia zonas de espacio libre. El estudio dará lugar a la propuesta de una arquitectura de software que permita la reutilización de los componentes (los comportamientos) en sistemas reactivos o sistemas híbridos.
- II. En segundo lugar, utilizando la misma arquitectura, la adaptación y comparación con el método parcial del centro de áreas, continuando con el trabajo desarrollado en [7] y [8]. El uso de los sensores frontales ayuda a paliar algunas de las limitaciones encontradas cuando se utilizan sensores alrededor de todo el robot, en los 360°.
- III. En tercer lugar, aplicar el concepto de centro de áreas sobre las imágenes obtenidas a través de un sistema de visión mono-cámara, mostrando algunas de las posibilidades de aplicación del concepto del centro de áreas. El desarrollo de este segundo sistema de visión por computador, dará lugar a un software de código libre distribuido bajo licencia GPL ([32]), permitiendo así a otros investigadores, que inicien su andadura con sistemas de visión, acceder de forma práctica y rápida a las peculiaridades de algunas de las etapas del procesado de imágenes.
- IV. En cuarto lugar, proponer métodos geométricos basados en la perspectiva plana que, apoyados en el cálculo del baricentro, podrían abrir las puertas a diferentes técnicas para llevar a cabo la conducción autónoma en entornos reales. Este estudio geométrico se realizará con los resultados de las pruebas realizadas sobre el software de visión desarrollado.

Para conseguir cada uno de estos subobjetivos, el trabajo se ha ido desenlazando en diferentes etapas, cada una de las cuales pretende clarificar tanto la utilidad del método del centro de áreas como la generalidad alcanzable por la arquitectura final. Estas etapas se pueden delimitar del siguiente modo:

- En una primera aproximación, se lleva a cabo el desarrollo de un sistema cliente-servidor cumpliendo: (1) El sistema servidor está a la espera de órdenes de posicionamiento, incluyendo la distancia mínima considerada suficiente para considerar alcanzada la posición objetivo (las coordenadas serán relativas a la posición inicial del robot). (2) Una vez el servidor ha recibido la orden, dejará lo que esté haciendo para calcular la ruta y dirigirse hacia el objetivo, a través de mecanismos reactivos capaces de evitar obstáculos imprevistos. (3) Una vez el robot ha llegado a su posición, se detiene y deja de enviar su posición.
- En el interior de una arquitectura servidora más elaborada, se calcula la posición del centro de áreas, guardando los datos en un fichero - o salida de texto editable - para el estudio de los puntos obtenidos. Esta arquitectura se mostrará de forma evolutiva e irá incluyendo mayor versatilidad en cada una de las versiones presentadas, tratando de manifestar las posibilidades de uso y extensión en trabajos futuros.
- Se diseña una interfaz de usuario propio que permita el envío de parámetros al servidor, adaptando este último para que entienda este tipo de comunicación e interprete las órdenes adecuadamente.

- Se adaptan las funciones del servidor para ejecutar el método parcial del centro de áreas y, del mismo modo, se dota a la interfaz de usuario de una opción para elegir entre la aplicación del método general o el método parcial.
- Se lleva a cabo una batería de pruebas apoyadas en estos módulos y en un conjunto de mapas generados para este trabajo de investigación. Al finalizar esta etapa se cumplen los objetivos I y II.
- Se estudian las imágenes procedentes de un sistema de visión mono-cámara con el objetivo de localizar los límites de la vía por la que se circula, con los que poder aplicar el método del centro de áreas, considerándolos como límites físicos reales (tal y como si se tratase de la lectura de los sensores). Esta etapa consiste fundamentalmente en el desarrollo de una aplicación de visión artificial que permita ensayar con diferentes estrategias de preprocesado, segmentación y seguimiento.
- Se realiza una amplia batería de pruebas para estudiar el comportamiento de los diferentes filtros, de donde se extraerán las configuraciones con las que se obtengan los mejores resultados de detección.
- Sobre estos resultados se estudiará la aplicabilidad del método del centro de áreas, apoyado en el algoritmo de Kalman como técnica predictiva en caso de discontinuidad. Con esta etapa se cubre el objetivo III.
- A partir del estudio geométrico de las figuras sobre las que aplicar el método, se proponen mecanismos complementarios para la detección (a través del estudio de la convergencia de las formas) y el seguimiento (utilizando el algoritmo de Kalman) de los puntos de interés, en situaciones reales de tráfico en sentido contrario y curvaturas de la trayectoria (como son el estudio de la forma trapezoidal en perspectiva o de los puntos de fuga). Con esta etapa finaliza el objetivo IV y la totalidad de este trabajo de investigación.

2 INTRODUCCIÓN

2.1 Hechos históricos relevantes

La palabra robot procede de la palabra checa *robota*, que significa *servidumbre, trabajo forzado o esclavitud*, utilizada por primera vez por Karel Čapek en su obra de teatro *Rossum's Universal Robots* (1920). Sin embargo, para encontrar las primeras referencias al concepto contenido en este término hay que remontarse hasta la mitología griega, con las leyendas de Camus o Pygmalion, la cultura egipcia, con las máquinas pensantes denominadas oráculos, los babilonios, con el primer reloj de agua del que hay evidencia, la mitología hebrea o la cultura china entre otras.

En el año 1206 D.C. encontramos lo que se ha considerado la invención de los autómatas programables, por el árabe Ibn Ismail Ibn al-Razzaz (llamado Al-Jazari debido al nombre de la zona donde nació, el norte de Mesopotamia, Al-Jazira), aprovechando la energía hidráulica para realizar la tarea. En torno a 1495, Leonardo Da Vinci realiza un diseño de robot humanoide, no sabiéndose si hubo algún intento de construir el diseño.

El desarrollo de la robótica en el marco de la revolución industrial puede considerarse iniciada en torno al siglo XVIII, con la máquina textil programable mediante tarjetas perforadas inventada por Joseph Jacquard. A mediados de este siglo, Jacques de Vaucansos construyó varios músicos de tamaño humano y, a principios del XIX, Henri Maillardert construye una muñeca mecánica capaz de dibujar.

A lo largo del siglo XX existen infinidad de referencias de aportaciones al desarrollo de la robótica autónoma, destacándose los trabajos de Alan Turing, con aportaciones tan importantes que impregnan a la mayor parte de las ciencias y del pensamiento. En 1942, el espléndido escritor Isaac Asimov, utiliza por primera vez el término *robótica* con el significado que se le sigue dando actualmente, en su obra *Círculo Vicioso (Runaround)*, donde también menciona las famosas *tres leyes de la robótica*, que completaría con la *ley cero* en 1985. En 1948, el tempranísimo Norbert Wiener acuña el término *cibernética*, en su obra *Cibernética o El control y comunicación en animales y máquinas* ([22]). Las aportaciones en la década de los 50 de Rosenblatt con las redes neuronales, apoyado en los estudios de Warren McCulloch, Walter Pitts, Luria, Lashley y Hebb, entre otros, han ayudado a enfrentar algunas de las limitaciones sobre el conocimiento que se posee en relación al funcionamiento del cerebro humano. La figura de John Von Neumann también se puede destacar en esta época, elaborando el concepto de computadoras reprogramables. Hay infinidad de investigadores que participaron de este desarrollo incipiente de lo que después se englobaría dentro de las áreas de estudio de la *Inteligencia Artificial*, nombre que se acuña en una conferencia impartida por Marvin Minsky y John McCarthy, en 1956. En 1963, Karl Popper en su obra *Conjeturas y Refutaciones: el Crecimiento del Conocimiento Científico*, trata de dar generalidad al modelo hipotético-deductivo propuesto por Clark Hull y Thomas Ross unos 30 años antes, con el que intentaban diseñar robots con capacidad de aprendizaje.

La gran evolución tecnológica en la que estamos inmersos ha obligado a reinventar una y otra vez los mecanismos y las herramientas utilizadas para resolver los problemas a los que se enfrenta la ciencia. Desde mediados del siglo XX, tanto las ciencias computacionales como todas las que se apoyan en ellas, han estado continuamente alimentadas con nuevas mejoras tecnológicas que han ido abriendo nuevos horizontes en los planteamientos con los que se ha trabajado en cada momento, exigiendo a cambio un gran esfuerzo de re-inventación.

2.2 Paradigmas

Al afrontar una investigación en el área de la Inteligencia Artificial con robots, se hace bajo un condicionante fundamental: la carencia de marcos de trabajo

comunes. En el modelo de desarrollo propuesto se ha pretendido tener como objetivo la construcción de una arquitectura capaz de integrar los diferentes paradigmas reconocidos hasta el momento, que serán descritos brevemente.

Las capacidades crecientes de los robots autónomos aplicando los conocimientos de Inteligencia Artificial, están limitadas fundamentalmente por la falta de conocimiento en áreas como la neurociencia, así como la falta de herramientas adecuadas debido a la complejidad del estudio del cerebro humano ([14]). A pesar de las limitaciones actuales de los robots, la variedad de logros y aplicaciones ha sido muy notable en las tres últimas décadas. De hecho, el desarrollo práctico de los robots como verdaderos mecanismos autónomos, comienza a germinar en los años 70 con el desarrollo del robot Shakey, el primer robot móvil que es capaz de navegar e interactuar con su entorno, pero al igual que en otras ciencias, su verdadero crecimiento no comienza hasta que en la década de los 80 se tomara como referencia el medio biológico, con Valentino Bratenberg y Michael Arbib como importantes referentes. El planteamiento utilizado para llevar a cabo la autonomía del robot Shakey, en el que se supone la existencia de una representación global del entorno en el que se sitúa el robot, se enfrentaba por un lado a los fallos ante situaciones imprevistas y, por otro, a la imposibilidad de detallar ese mundo tanto como sería deseable. Estas problemáticas se solventaban con la consideración de los comportamientos de Arbib, basados en los comportamientos reflejos de los animales, surgiendo problemas asociados a la eliminación de toda planificación en los mismos, considerándose como una tarea que emerge de su composición. A partir de los años 90 se comienza a trabajar con el camino más directo para solventar los problemas de uno y otro planteamiento, fusionándolos, siendo reseñable el desarrollo de la arquitectura AuRA (*Autonomous Robot Architecture*) por Ronald Arkin (i.e. [23]).

La construcción de robots capaces de aprender por sí mismos a resolver cualquier tipo de problema que se les presente, se puede considerar que está aún en sus inicios, por lo que es de vital importancia la elección adecuada de los mecanismos utilizados para tratar el problema particular que se desea resolver. Lo dicho en el párrafo anterior resume, desde el punto de vista de la organización de la inteligencia en los robots, los paradigmas existentes en la actualidad:

<p><u>JERÁRQUICO</u> Procesa la información secuencialmente: datos de los sensores, planificación y acciones.</p>	
<p><u>REACTIVO</u> Elimina la fase de mayor consumo de tiempo en el paradigma jerárquico (planificación <i>emergente</i>), a través de acoplamientos del tipo "Dato del sensor - Acción", denominados comportamientos.</p>	
<p><u>HÍBRIDO DELIBERATIVO-REACTIVO</u> Ejecuta los comportamientos de modo reactivo tras trazar un plan dividido en subtareas.</p>	

En la parte derecha de la tabla se han dibujado las relaciones entre tres primitivas comúnmente aceptadas, que representan una división para las diferentes funciones que lleva incorporadas el robot: Las que reciben datos de los sensores para realizar una transformación pertenecerían a la primitiva "SENTIR", aquéllas que producen una tarea pertenecerían a la primitiva "PLANEAR", mientras que las que ejecutan comandos de movimiento pertenecerían a la primitiva "ACTUAR".

El par SENTIR-ACTUAR connota comportamientos puramente reflejos en el paradigma reactivo, mientras que también incluye comportamientos innatos y aprendidos en el paradigma híbrido (ambos relacionados con el aprendizaje, el primero con un aprendizaje "evolutivo" y el segundo con un aprendizaje "individual").

3 DESCRIPCIÓN DEL PROBLEMA

3.1 Motivación y enfoque

El movimiento de los robots sin necesidad de intervención humana puede ayudarnos en infinidad de campos, cuya evolución está continuamente yendo más allá de la imaginación. Algunos ejemplos relevantes pueden ser las ciencias médicas (soporte en hospitales, ayuda a la movilidad de enfermos,...), la exploración espacial, las ciencias biológicas (estudio de lugares inaccesibles como el fondo de los océanos o zonas de temperaturas extremas), ayuda durante acciones de salvamento (edificios con peligro de derrumbe, situaciones de catástrofe en general), transporte, entretenimiento, etc.

En los últimos años se ha incrementado de forma notable el interés por la conducción autónoma de los automóviles, sobre todo desde dos ámbitos de aplicación: El militar (retos como los planteados por DARPA, en Estados Unidos, o M-ELROB, en Alemania) y el civil (como C-ELROB). La utilidad de estos estudios, tanto para sistemas civiles como militares, cubren un amplio espectro de posibilidades: desde tareas de soporte sobre determinadas acciones que aumenten la seguridad, hasta tareas de conducción autónoma entre un punto y otro.

El objetivo de estos sistemas está inmerso en un objetivo común mayor: replicar el funcionamiento del cerebro humano o de todo el sistema nervioso. Los procesos que se tratan de replicar no requieren un conjunto de sensores excesivamente exacto para obtener soluciones válidas a los problemas, tal y como se puede comprobar por lo limitado de nuestros sentidos (cuando tenemos que guiar nuestro coche por la carretera, en raras ocasiones necesitaremos hacer cálculos de distancia tan exactos como los que se consiguen a través de los sensores electrónicos, al menos de forma consciente). Por otro lado, se posee de una tremenda red neuronal que lleva almacenando conocimiento e información desde que nacemos y que, aún hoy día, sigue siendo una gran desconocida. Las dificultades para replicar el conocimiento almacenado por el cerebro son muchas, más aún (si cabe) si lo que se busca son respuestas en tiempo real, de ahí que se considere relevante el estudio de técnicas que den respuestas reactivas, posteriormente aplicables a arquitecturas híbridas que posean más conocimiento del mundo externo.

La idea de trasladar el método físico del centro de masas como uno de estos métodos de representación endógena del mundo, tal y como se explica en [1], procede de la idea sugerida por Walter Pitts y Warren S. McCulloch en su obra de 1947 "How we know universals: the perception of auditory and visual forms": un método biológicamente viable para obtener conocimiento del entorno a través de los sentidos, podría consistir en aplicar un conjunto de transformaciones a través de las que se puedan extraer representaciones invariantes, que permitan al cerebro producir versiones estándar centradas en el centro de gravedad.

3.1.1 Método general del centro de áreas

El objetivo del Método General del Centro de Áreas es construir una representación interna del robot, que le permita orientarse en la navegación en una arquitectura global que utilice los conceptos comunes a los tres paradigmas (de [2]): Espacio sensorial, espacio topológico, espacio de decisión y espacio efector. Esta arquitectura sirve de referencia para la usada en este estudio:

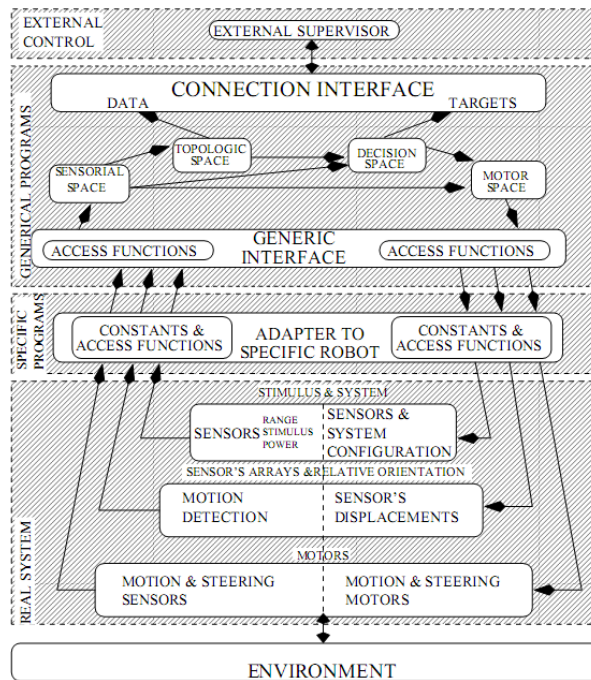


Figura recogida de [2]

La integración de la arquitectura híbrida puede integrar tanto las tareas y métodos aplicados en la robótica autónoma, como otras procedentes de la ingeniería del conocimiento, como son las ontologías (de [3]).

El método general del centro de áreas ayuda a resolver problemas de mínimos locales (de [4] y [6]), comparado con el método VFF (en la parte izquierda de la figura):

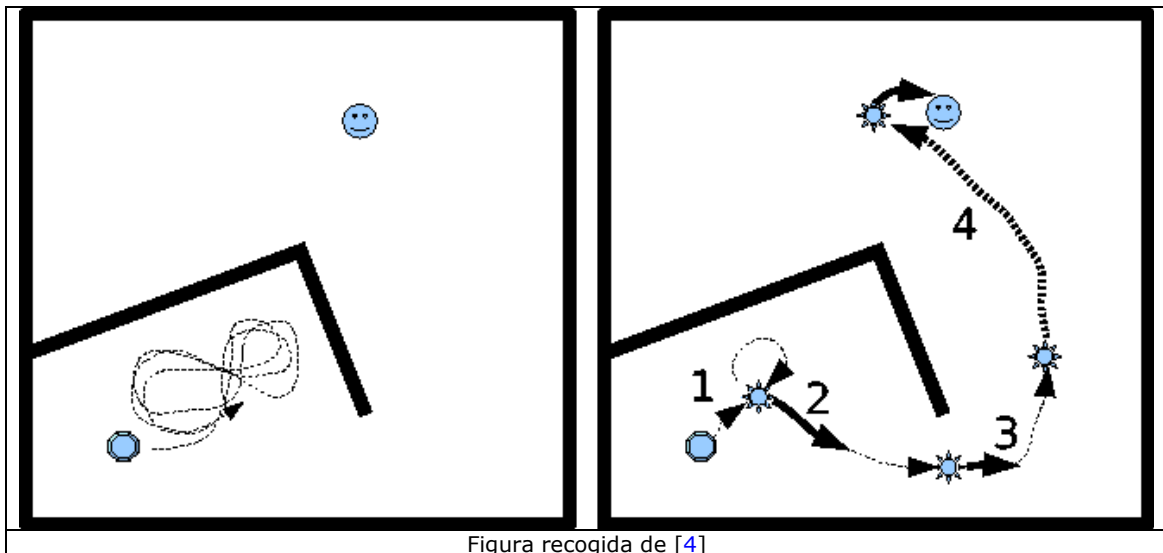
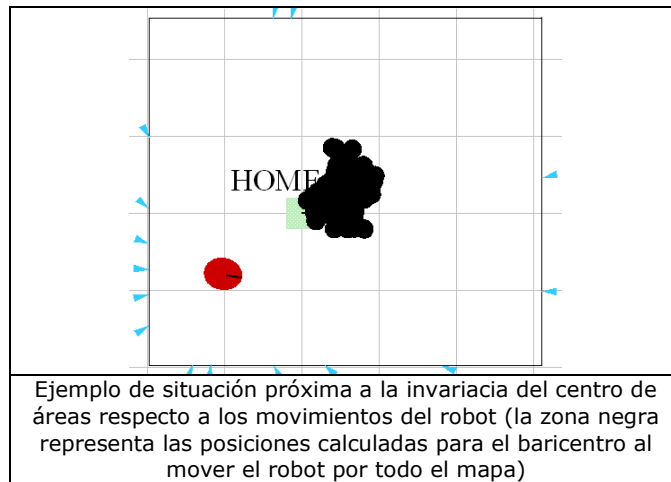


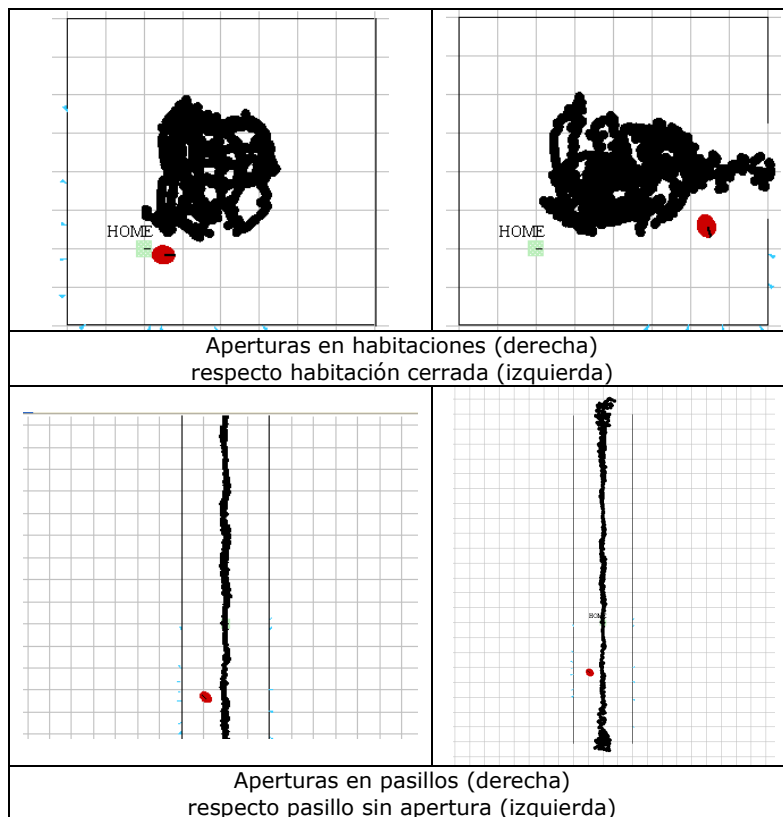
Figura recogida de [4]

El concepto físico en el que se basa el método del centro de áreas es el centro de masas, intercambiando el concepto de masa por el de área o, en el caso más general, por el volumen, consiguiéndose obtener un punto de referencia desde el robot con unas características muy interesantes:

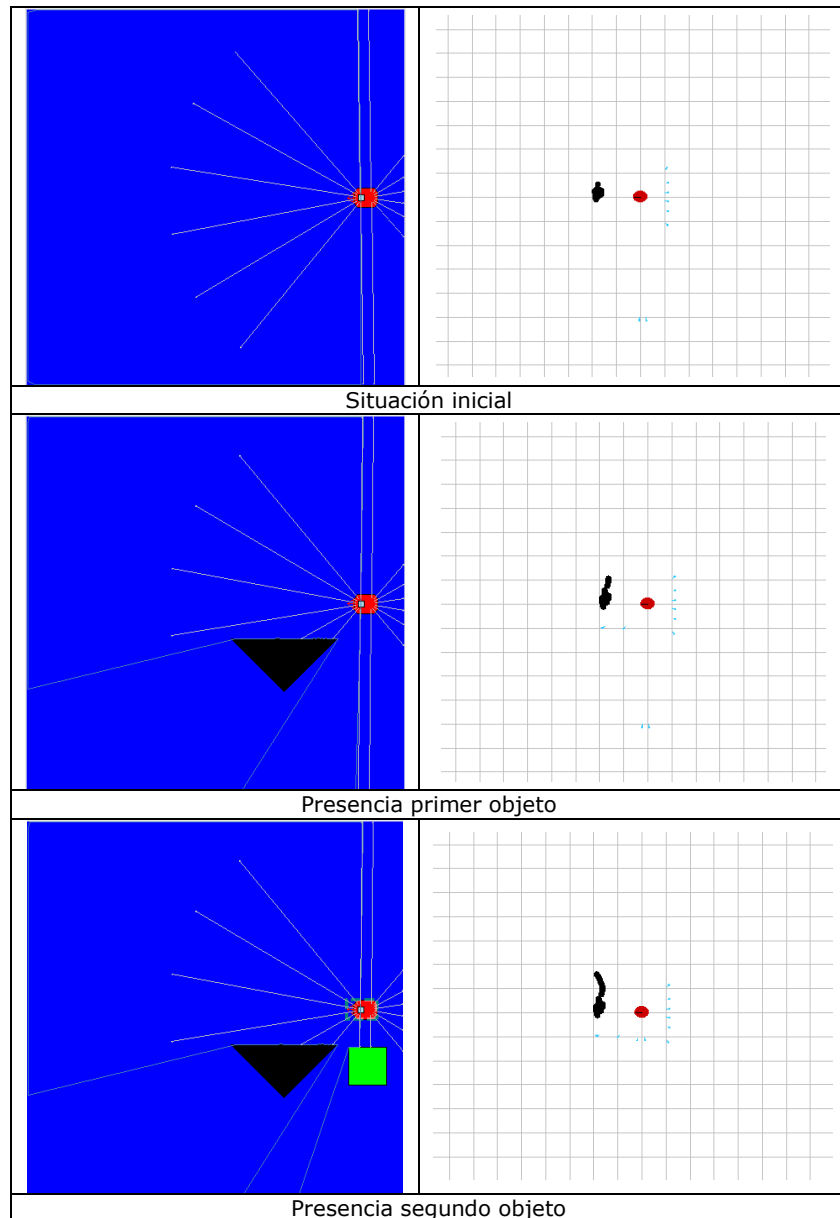
- Valor aproximadamente invariante ante los movimientos del robot, en contornos claramente definidos por la lectura de los sensores



- Cambios de patrón ante la existencia de aperturas:



- Cambios de patrón ante la aproximación de objetos (idea procedente de [5]):



- El hecho de ser un punto que aporta subjetividad al robot, respecto a la situación del espacio que le rodea, aportando un conocimiento que puede ser utilizado para construcciones en el espacio topológico o desde estructuras de un nivel superior en general ([2]).

[Estas pruebas se han podido realizar utilizando con el software desarrollado, que será descrito más adelante. En concreto, se usa el módulo *ModuloRoboticaV2*, a través de la opción "Wander" desde el *MobileEyes* y almacenando todos los puntos del baricentro en un vector, que es el conjunto de puntos que se pintan en *MobileEyes*, desde la opción incluida en el menú "Map → Draw All Barycenter points".]

3.1.2 Método parcial del centro de áreas

El movimiento del robot hacia el centro de áreas, calculado con el método general, finaliza cuando el robot alcanza la zona ocupada por el punto, llegándose a situaciones de equilibrio que rompen con la continuidad del movimiento. Para romper esta estabilidad se pueden usar otros controles de un nivel superior ([2]) o se puede emplear el método parcial del centro de áreas, presentado por primera vez en [7], donde se demuestra la utilidad del método para recorrer las zonas de espacio libre y evitar obstáculos al dirigirse hacia un punto objetivo:

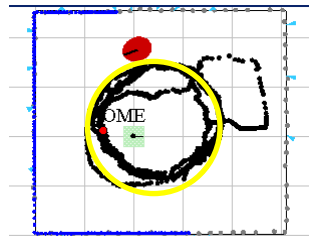


Figura recogida de [Z]: capacidad del método parcial del centro de áreas para recorrer zonas de espacio libre, posicionando el robot en diferentes orígenes

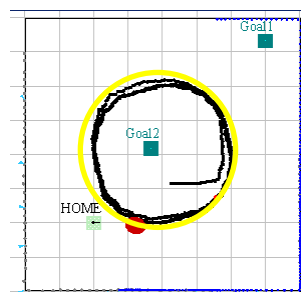
Además se realizan experimentos con sensores ideales y sensores reales (considerando que se pierden al menos el 50% de las lecturas) encontrándose diferencias poco significativas teniendo en cuenta el margen de error considerado.

Las propiedades obtenidas para el método general mantienen su interés en el método parcial, con algunas variaciones:

- Estabilización de los valores a lo largo de figuras geométricas, no de un único punto, en contornos claramente definidos por la lectura de los sensores:

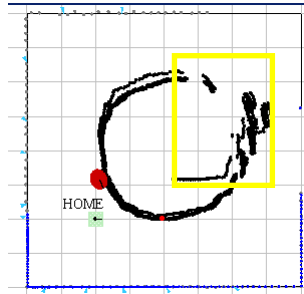


Ejemplo de estabilización de las posiciones del baricentro y el movimiento del robot a lo largo de una figura geométrica, destacada en amarillo. Al ser una habitación cuadrada la figura emergente es circular. Las perturbaciones iniciales se deben a las colisiones, que no se producen en una habitación mayor, tal y como se muestra en la siguiente figura.

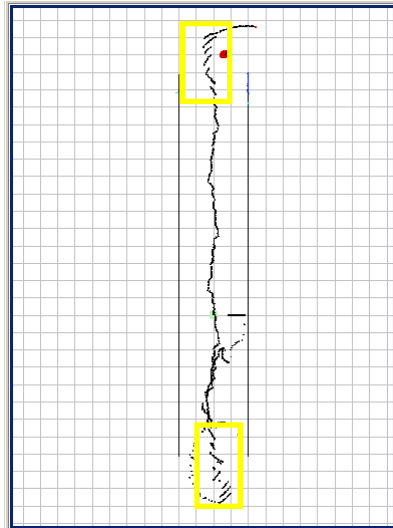


Ejemplo de estabilización de las posiciones del baricentro a lo largo de la figura geométrica en una habitación suficientemente grande para evitar colisiones iniciales.

- Existencia de aperturas:

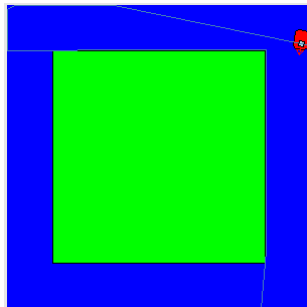


El cambio de patrón ante la presencia de aperturas, en este caso, se muestra con la presencia de discontinuidades en la forma geométrica circular obtenida sin la presencia de la apertura



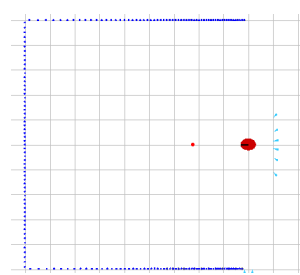
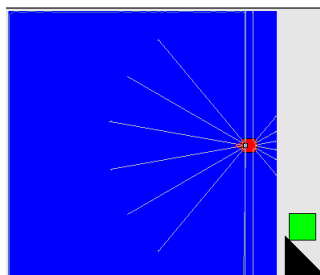
Discontinuidades en pasillos

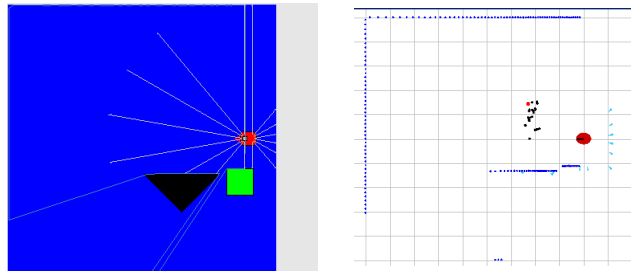
- Esquinas y colisiones:



Discontinuidades en esquinas, encuadradas en amarillo, y en las colisiones que se producen tanto al iniciar el movimiento como al girar en las esquinas, encuadradas en naranja

- Presencia de objetos





Discontinuidades ante la presencia de objetos

[Estas pruebas se han podido realizar utilizando con el software desarrollado, que será descrito más adelante. En concreto, se usa el módulo ModuloRoboticaV4, con la opción "Wander" del MobileEyes, tras conectar el cliente QT y enviar las coordenadas y el método radar "front" iniciales.]

3.1.3 El método parcial del centro de áreas aplicado al sistema de visión

Una evolución del método parcial del centro de áreas se puede encontrar en [8], donde se propone, como hipótesis de trabajo, la idea fundamental en la que está basada el desarrollo del sistema de visión para la conducción autónoma en entornos reales, esta es: los humanos fundamentalmente modelan el área de espacio libre, no el área ocupada con obstáculos. Entre las fuentes de inspiración se puede encontrar, precisamente, la idea de la búsqueda de espacio libre cuando se encuentra tráfico en sentido contrario:

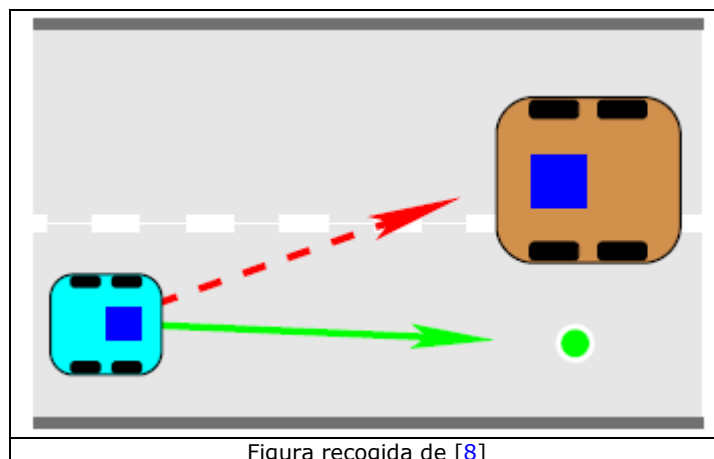


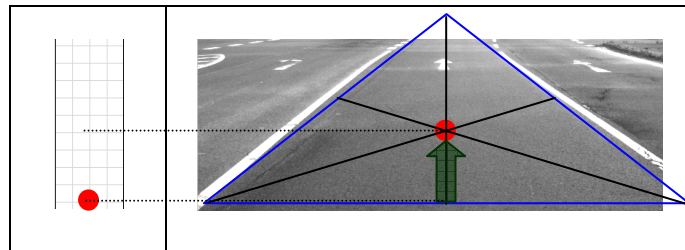
Figura recogida de [8]

El sistema de visión puede constituir un nuevo esquema sensor-acción acoplable al resto de la arquitectura reactiva o híbrida. En lugar de utilizar la forma geométrica resultante de la lectura de los sones, se utiliza la representación obtenida en base a las imágenes para ejecutar el esquema motor, que sería idéntico al desarrollado ya para la lectura de los sensores.

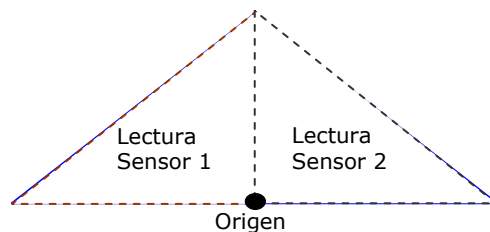
La visión aérea de una carretera no deja de ser la concatenación de una serie de curvas geométricas, más o menos bien definidas, que pueden resumirse en tres tipos: rectas, círculos y coloides (i.e. [45]). El diseño de carreteras es un campo de estudio muy amplio y no es recomendable trivializar la complejidad topológica inherente, además se saldría del alcance de este trabajo. Sin embargo, sí es importante tener en cuenta la regularidad que se busca en los trazados desde su diseño, ya que de lo contrario la casuística de las posibles situaciones en el tratamiento se volvería excesivamente amplia. Esto no impide que existan trazados ajenos a la regularidad establecida en el diseño, como se puede observar en multitud de situaciones, como son las intersecciones y muchos de los caminos que podemos encontrar al circular por cualquier zona (o las que se manifiestan como consecuencia de la visión en perspectiva de una superficie ondulada).

Disponer de una visión en perspectiva es, en efecto, una limitación para la representación y comprensión, por parte del sistema, de la escena visual, que podría tratar de suplirse utilizando otra cámara para simular la visión estereoscópica de la que disponemos muchos seres vivos (i.e. [46], [47] en relación al problema de la navegación, [48] en relación a la aplicabilidad en sistemas a tiempo real). Sin embargo, utilizar una sola cámara para el uso del método del centro de áreas puede resultar muy provechoso por tres cuestiones fundamentales:

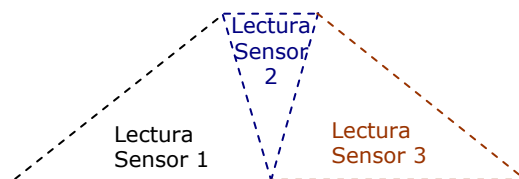
1. La convergencia de los bordes hacia el punto de fuga de la perspectiva frontal paralela, convierte el baricentro en una zona aún más fiable (rompiendo la estabilidad encontrada en recorridos con paredes paralelas para el método general del centro de áreas, sin necesidad de romper la simetría para darle continuidad al movimiento)



2. La forma geométrica se puede aproximar a un polígono similar a la aproximación obtenida con los sensores de rango, dos ejemplos serían:
 - a. Triangular: Se puede tratar como el polígono de un único sensor, con origen en el punto de fuga, o como dos triángulos, tomando como origen la posición de la cámara en el eje de abscisas



- b. Trapezoidal: Se puede tratar como un polígono procedente de la lectura de 3 sensores, utilizando de nuevo como origen la posición de la cámara en el eje de abscisas.



3. Al menos a priori, al trabajar con una sola cámara, se reduce la complejidad del problema, simplificando el cálculo que corresponde a la construcción tridimensional del mundo a una representación bidimensional, que concuerda con la representación estudiada con las lecturas de los sensores de rango.

Si en el caso de la figura trapezoidal nos quedásemos con la forma geométrica denominada "Lectura Sensor 2", se dispondría de una visión similar a la presentada en [8]:

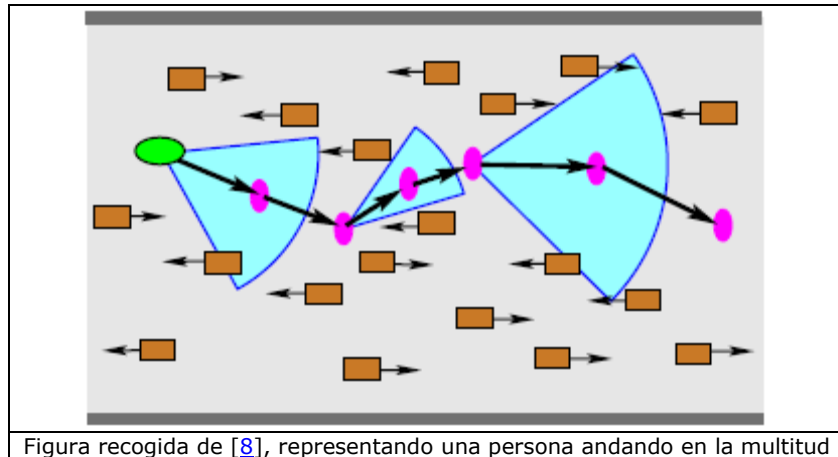


Figura recogida de [8], representando una persona andando en la multitud

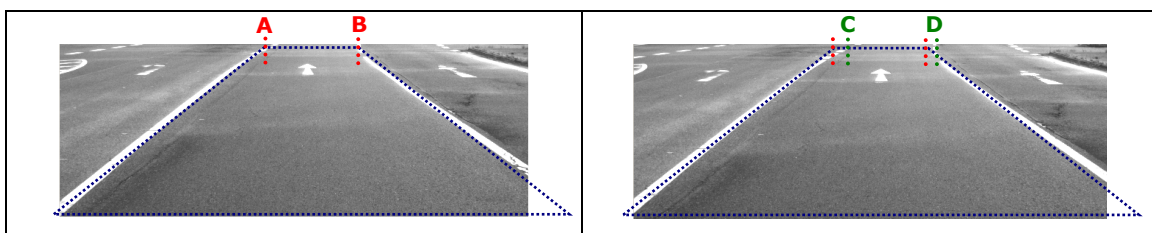
A través del sistema de visión se podrían localizar los extremos más próximos de los objetos, de manera que sea aplicable la misma técnica de división utilizada para evitar que el centro de áreas se encuentre en zonas ocupadas.

3.1.4 Predicción en situaciones de discontinuidad: Algoritmo de Kalman

La extracción de los límites de la vía en situaciones reales, a través del procesado digital de las imágenes, constituye una tarea ardua, debido a la gran cantidad de factores que pueden modificar las condiciones visuales. Es frecuente encontrarse con situaciones en las que se pierde la continuidad de detección de los límites de la vía (ya sea por las carencias del algoritmo, por las limitaciones de las condiciones lumínicas, por las características del dispositivo que capta las imágenes o por otras). Para mitigar las consecuencias que provocaría la pérdida de referencia durante estas situaciones (por ejemplo, a lo largo de una curva), se propone el uso del algoritmo de Kalman como mecanismo predictivo.

El uso del algoritmo de Kalman está justificado por su aplicabilidad en multitud de dominios, tanto en su versión lineal como en la no lineal (existen infinidad de ejemplos, una pequeña muestra se puede encontrar en [38], [39], [40], [41], [42]). Hay guías excelentes para comprenderlo y realizar los cálculos en diferentes entornos (i.e. [37]).

En el caso de la detección de los márgenes de la carretera, la forma esperada para la visión de la carretera es aproximadamente la de un trapecio, con la perspectiva que ofrece una cámara frontal. Esto se visualiza muy bien cuando se utiliza cualquier simulador para juegos de coches (i.e. [43 a] y [43 b]) o sencillamente cuando vamos conduciendo en el coche (aunque no sea recomendable). Construir este trapecio y calcular su centro de áreas nos proveería de un mecanismo de utilidad para conducción automática. Al avanzar en la carretera, el ajuste del trapecio requiere una serie de transformaciones, ante irregularidades del asfalto, cambios de pendiente, presencia de obstáculos y la propia curvatura de la trayectoria. Por ejemplo, ante una situación en la que se inicia una curva a la derecha, si tenemos localizados los vértices superiores (puntos A y B), necesitaremos irlos desplazando hacia la derecha para ajustar correctamente la representación de la escena (hacia los puntos C y D), cada uno de ellos de forma independiente en la cantidad adecuada.

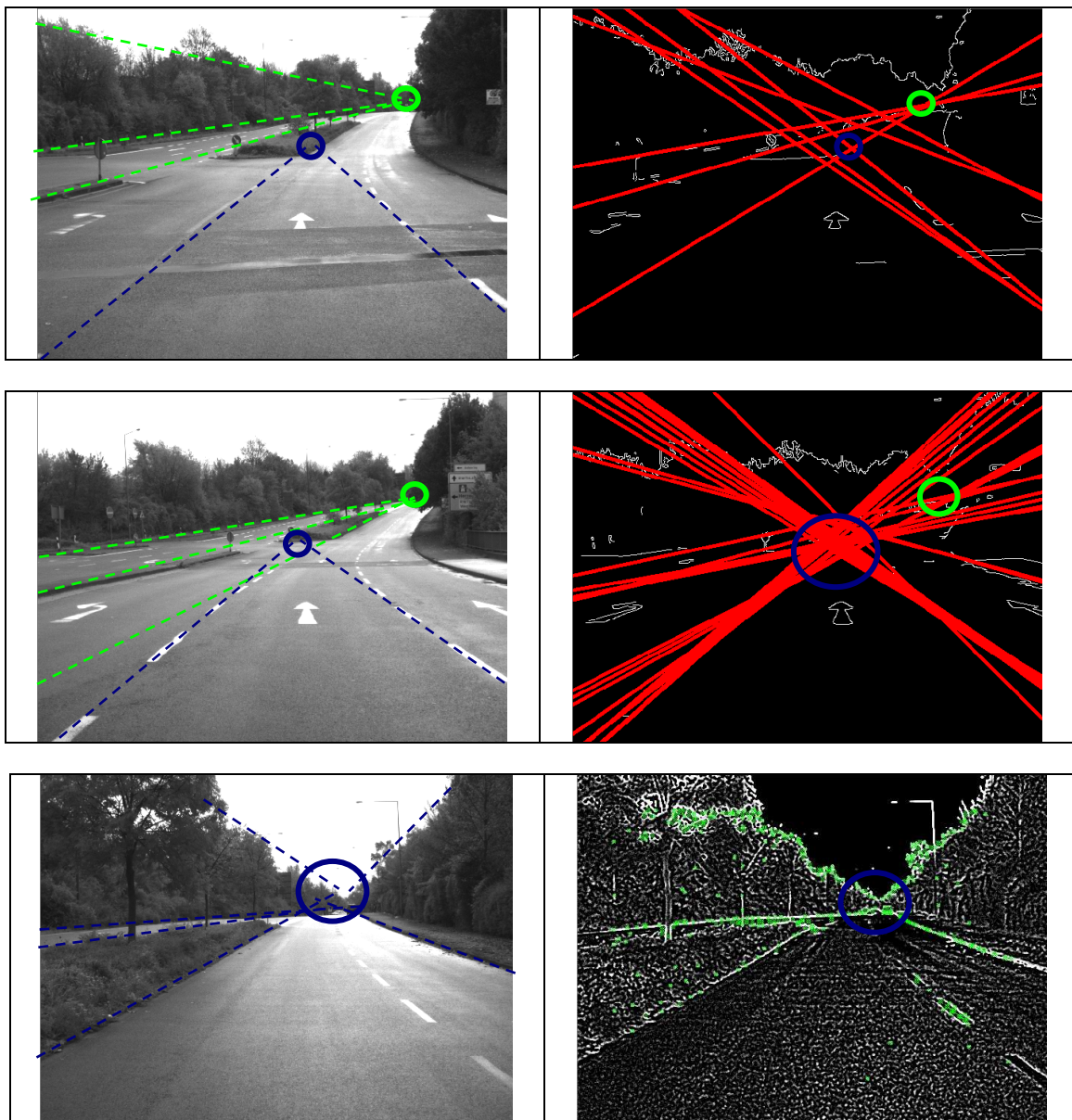


Como se puede advertir en este caso concreto, lo que nos interesaría, ante una posible discontinuidad en la detección, sería el poder predecir a lo largo del eje de abscisas el desplazamiento estimado, correspondiente al cálculo del filtro unidimensional.

3.1.5 Métodos geométricos basados en la perspectiva plana

A partir de las pruebas de procesamiento de imágenes, se han detectado con frecuencia comportamientos convergentes hacia los puntos de fuga que da la perspectiva plana (obtenido del cruce de las líneas aproximadas para los límites de la vía y el contorno de los objetos situados en los márgenes o en el horizonte). Sin llegar a hacer un tratamiento adecuado, a través de la matemática proyectiva, sí se ha querido destacar la localización de estos puntos (como un acercamiento muy modesto) que podrían ser útiles para detectar las desviaciones de la linealidad de la trayectoria.

Se han extraído algunos ejemplos para ilustrar algunas de estas situaciones:



Un estudio en mayor detalle que el presentado aquí de estos puntos podría dilucidar otras técnicas de apoyo para la aplicación de los métodos del centro de áreas.

3.2 Estructura de la memoria

El resto del trabajo está estructurado tal y como se explica a continuación.

En el siguiente apartado '*Descripción del entorno de trabajo*' se detallan las características de las herramientas utilizadas para el desarrollo del software y de las pruebas (robot, librerías de software, lenguaje de programación, plataformas, instalación, simulación y modo de ejecución de las pruebas), que no deja de ser decisivo al iniciar una investigación en general y, especialmente, en robótica, donde es complicado encontrar entornos de trabajo cómodos con los que poder centrarse únicamente en la investigación, dejando de lado dificultades que van desde la falta de integración (entre plataformas de software, sistemas operativos, lenguajes de programación, dispositivos electrónicos, etc.) hasta los problemas físicos con el robot (en relación a su traslado, las desviaciones del comportamiento en diferentes situaciones, limitaciones al tratar de llevar a cabo el experimento simulado, etc.). Como complemento a este capítulo se han agregado 3 apéndices.

En el capítulo titulado '*Descripción de los módulos desarrollados*' se pretende dar a conocer todo el proceso de desarrollo de los módulos de software, con un nivel de detalle suficiente para su uso y posible extensión en trabajos futuros. Está dividido en dos etapas fundamentales:

- (1) La creación de los componentes que permiten ejecutar algoritmos en el robot y la comunicación con el mismo, donde se despliegan las características de los diferentes módulos compuestos de forma general por el conjunto Cliente-Servidor.
- (2) La creación de una herramienta para el procesado de imágenes, basada en la descomposición del tratamiento en diferentes etapas. En concreto se proponen tres etapas:
 - a. Preprocesado
 - b. Segmentación y selección de objetos
 - c. Seguimiento

Una vez se ha descrito, por un lado, la necesidad de unificación de arquitecturas en base a los diferentes paradigmas y, por otro, el entorno de trabajo y los módulos desarrollados, se ha dedicado el siguiente capítulo '*Programación orientada a objetos*' a la descripción de lo que se puede considerar un marco óptimo para la estructura de los algoritmos o, yendo más allá de este trabajo, para estructurar información sobre un área de conocimiento particular. En lo que se refiere a los módulos de robótica, este capítulo está dividido como sigue:

- (1) Una primera parte que se encarga de relacionar la filosofía de la programación orientada a objetos con el marco teórico de los sistemas reactivos.
- (2) Esta misma relación extendida a los sistemas híbridos, incluyendo así el marco teórico de los sistemas deliberativos.

En lo que se refiere al sistema de visión, se propone una estructura relacionada con las diferentes etapas propuestas para el procesado de imágenes.

A continuación, en el capítulo '*Método del centro de áreas y conducción autónoma*', se puede encontrar la descripción del desarrollo asociado al método principal del que es objeto este estudio, su inclusión en la arquitectura y su aplicabilidad a la conducción autónoma a través del sistema de visión. Una vez se ha agregado el método en la arquitectura general propuesta, es necesario decidir de qué modo va a actuar el robot en su movimiento hacia el centro de áreas, proponiéndose aquí un método de ejemplo que no deja de ser, en ningún momento, susceptible de sufrir infinidad de posibles modificaciones. El último apartado de este capítulo describe algunas de las aportaciones del procesado digital de imágenes al método del centro de áreas, integrado como un mecanismo de soporte al desarrollo realizado con los sensores de rango.

El siguiente paso consiste en una amplia batería de pruebas de diferentes tipos. Esto se encuentra en el capítulo '*Pruebas y análisis de los resultados*'. Las pruebas se han agrupado en dos categorías:

- (1) Las realizadas sobre un conjunto de mapas, con objetos cuya geometría da lugar a conclusiones interesantes sobre el comportamiento de los métodos general y parcial del centro de áreas.
- (2) Las realizadas sobre secuencias de imágenes grabadas desde vehículos conducidos en entornos reales.

Por último, en el capítulo '*Conclusiones y trabajo futuro*' se plasman las revelaciones más importantes obtenidas durante el desarrollo de la investigación, seguidos de posibles caminos que den continuidad a estos resultados.

4 DESCRIPCIÓN DEL ENTORNO DE TRABAJO

Se ha trabajado con un modelo de robot PIONEER 3-AT, a través de una simulación ([18]), que ha permitido la realización de las pruebas de cálculo y la recogida de datos sobre diferentes tipos de mapas, estáticos y con objetos móviles. Con el objetivo de poder reutilizar los desarrollos con robots diferentes, se ha hecho uso de las librerías ARIA ([17]), que suponen un soporte importante al llevar a cabo la comunicación entre los distintos componentes del sistema, haciéndola independiente del robot a través del firmware ARCOS instalado en el robot. No obstante, debido a que los algoritmos estarán asociados con configuraciones concretas de los sensores, que no va a poder ser idénticas para todos los robots, no habrá una total independencia.

Como clientes para la monitorización remota del servidor desarrollado en el robot, se han utilizado dos tipos: (1) Clientes desarrollados a medida. (2) El cliente MobileEyes ([19]).

Para el desarrollo del software se ha utilizado el lenguaje C++, por motivos de eficiencia a la hora de incorporar la librería ARIA y por ser un lenguaje Orientado a Objetos, ya que este tipo de lenguajes permiten poner en práctica las ideas de la teoría de esquemas (i.e. pp. 993-998 de [24]).

El tratamiento de imágenes está fundamentalmente basado en las funciones incluidas en la librería *Open Computer Vision* ([25], [26]), con la que también se obtiene un rendimiento óptimo trabajando con el lenguaje de programación C++, así como la modularización de las distintas etapas del procesado. Además de la eficiencia y del ahorro, en líneas de código, que se consigue utilizando esta librería, se consideran cruciales otros dos aspectos:

- El tipo de licencia de código libre, facilitando su posterior distribución y difusión.
- Permite una continuidad del desarrollo en C++, a pesar de no ser triviales las cuestiones relacionadas con la unicidad de un Framework que admita la compilación conjunta de todas las piezas.

El diseño y la implementación del interfaz gráfico de usuario se han llevado a cabo a partir de la versión de código libre 4.6.1 de las librerías QT ([27]), tanto en uno como en otro.

4.1 Características del robot

El robot del que se dispone para realizar las pruebas y, por otro lado, que se trata de simular a través del software, es el PIONEER 3-AT, del que se puede encontrar información en la página Web de MobileRobots ([31]). No obstante, a continuación se muestran algunas de sus características:

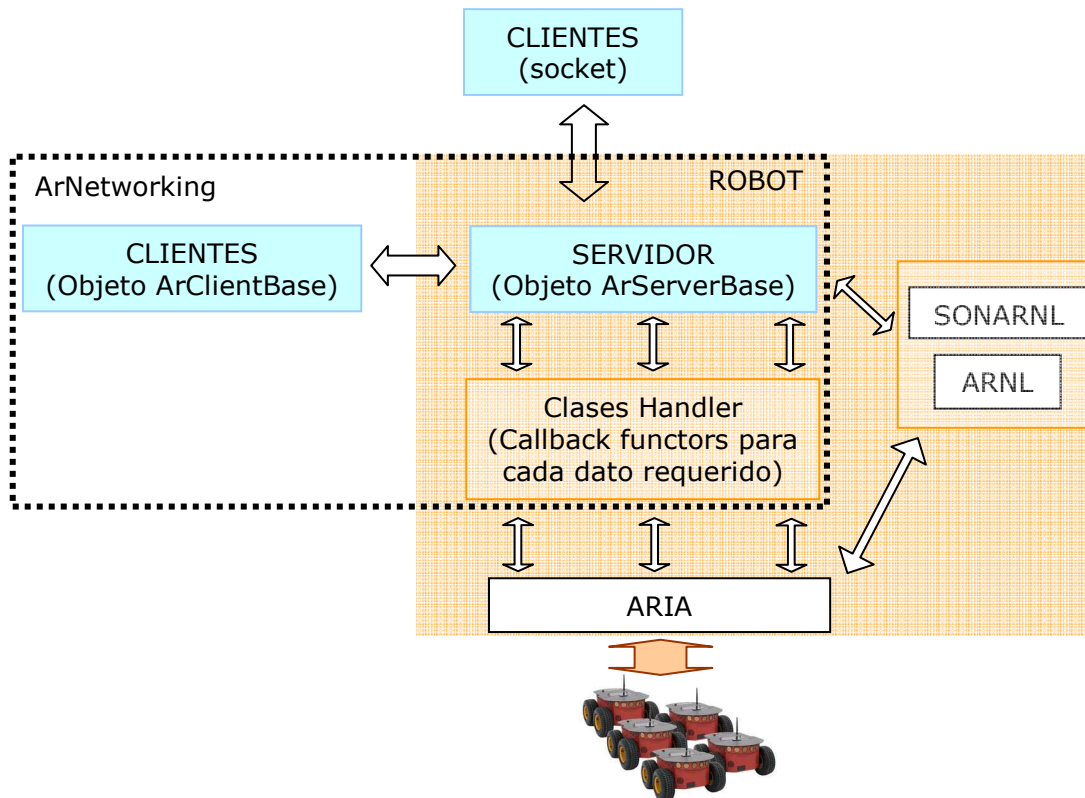
CARACTERÍSTICAS	
 <p>Pioneer 3-AT con toda la gama de accesorios</p>	<p><u>Sensores</u></p> <ul style="list-style-type: none"> (1) Sónares con visión entre 15cm y 5m y 500 codificadores de posición. Frontales: puede tener 8 (6 cada 15° + uno en cada lado). Traseros: la misma distribución que el frontal. (2) Láser. (3) Sensor inerciales: Giróscopo, acelerómetro, brújula, etc. (4) GPS. (5) Cámara PTZ, que permite el tracking de Blobs. (6) Cámara estéreo. (7) Bumpers. <p><u>Sistemas motores:</u></p> <ul style="list-style-type: none"> (1) 4 ruedas de 21.5cm de diámetro y 8.8cm de ancho (o ruedas macizas). (2) Relaciones de engranajes: 38.1:1 (engranaje de 381 dientes conectado a uno de 10 dientes). (3) Fuerza de empuje de 18kg. (4) Velocidad máxima de traslación de 1.2m/s. (5) Pasa por huecos que dejen 10cm por cada lado. (6) Sube escalones de 15.2cm. (7) Sube pendientes de hasta 40%. (8) Se puede mover en asfalto, arena, suciedad, pero no en alfombras (hay ruedas que permiten esto último) <p><u>Baterías:</u> con autonomía desde 4 hasta 8 horas para el robot básico, sin accesorios.</p>

4.2 Descripción general de las librerías utilizadas

4.2.1 Robótica

ARIA es una interfaz orientada a objetos preparada para facilitar el desarrollo de aplicaciones sobre robots móviles inteligentes de MobileRobots, entre los que se encuentra el PIONEER 3-AT. Incluye una librería denominada ArNetworking que permite la comunicación cliente-servidor, adecuada para la monitorización remota del robot, así como el uso de librerías adicionales de comunicación con los sensores (ARNL, SONARNL, ACTS, etc.).

Por sí sola, la librería ARIA es suficiente para desarrollar con facilidad un cliente que se conecte de forma remota al servidor del robot. Sin embargo, si deseáramos distribuir la arquitectura con un cliente remoto accediendo a las funciones situadas en el robot, por ejemplo, el desarrollo se agiliza a través del uso de la librería ArNetworking, que viene con la lógica necesaria para el despliegue de la funcionalidad en un servidor capaz de comunicarse con los efectores del robot a través de la librería ARIA, permaneciendo a la escucha de peticiones desde uno o varios clientes. El cliente servirá de GUI para el establecimiento de órdenes, la parametrización de los algoritmos y el acceso a recursos externos al robot. Este esquema de funcionamiento se puede resumir en la figura (el software desarrollado para las pruebas se corresponde con las zonas de fondo azul):



La conexión entre el cliente y el servidor puede realizarse a través de una conexión Wi-Fi, el resto de la arquitectura se sitúa en el robot permitiendo la comunicación con la librería ARIA, que será la que, a su vez, se comunique con el dispositivo y sus accesorios utilizando un conjunto de funciones que pueden ser lanzadas en diferentes hilos de ejecución (Callback functors). El servidor también se puede comunicar con librerías accesorias de ARIA, tales como ARNL o SONARNL.

4.2.2 Visión artificial

OpenCV es una librería de código libre (bajo licencia BSD) desarrollada en C y C++, lo que permite continuar desarrollando el código en C++, sin modificar la esencia del Framework preparado para los módulos de robótica. Además, provee un marco de desarrollo eficiente para ejecución en tiempo real y proporciona una gran cantidad de algoritmos plenamente funcionales, parametrizables, o interfaces para su creación personalizada (como es en este caso el conjunto desarrollado para la inclusión del algoritmo de Kalman, utilizado para realizar un seguimiento predictivo). Otra de las características que lo hace muy atractivo es el hecho de poder utilizarse en cualquier plataforma, ayudando a independizar el desarrollo del sistema operativo y del hardware.

Una buena referencia para comprender los fundamentos de OpenCV se puede encontrar en el libro [26].

Se ha detectado a lo largo del desarrollo de esta parte del software, que el hecho de que esta librería contenga programación en C la hace especialmente sensible en temas relacionados con la liberación de la memoria, sobre todo cuando lo que se pretende es trabajar en entornos reales, donde hay que tener un cuidado especial para que el proceso de tratamiento sobre una secuencia larga no se vaya ralentizando a medida que evoluciona en el tiempo.

4.3 Software para la simulación en Windows

La totalidad del software desarrollado supera las 13.000 líneas de código (divididas aproximadamente a partes iguales entre la programación dedicada al entorno robótico y la dedicada a la visión por computador) y, a pesar de no ser un factor

esencial, da una idea sobre el volumen y la complejidad que ha ido adquiriendo todo el entorno.

Se ha dedicado el '*Apéndice I*' para explicar con mayor detalle las herramientas de software en las que se apoya esta investigación, dividiéndose entre: Los elementos comunes utilizados en la parte de simulación del robot y en la de visión por computador, en un primer apartado, pasando a explicar las cuestiones específicas de ambas partes en otros dos apartados.

La ejecución de la simulación robótica se realiza a través de la consola en su mayor parte, para lo que se han implementado scripts que facilitan el arranque. Esto se explica con mayor detalle en el '*Apéndice II*'.

Para los ensayos de visión por computador se utiliza la herramienta de desarrollo propio *Research Artificial Vision Tool*, el proceso de instalación y ejecución se ha generado siguiendo el patrón general del sistema Windows. Una explicación con mayor detalle, necesaria no tanto para su instalación sino para comenzar a utilizarla, se ha llevado a cabo en el '*Apéndice III*'.

5 DESCRIPCIÓN DE LOS MÓDULOS DESARROLLADOS

5.1 Estructura general

En ambos desarrollos existen un conjunto de carpetas comunes, sin embargo es necesario distinguir entre los módulos desarrollados para la simulación del robot y la aplicación para el tratamiento de imágenes:

- (1) Los módulos desarrollados para la simulación del robot, tienen la nomenclatura general *MóduloRoboticaV[n]*, siendo *n* es la versión del módulo. En la carpeta con el mismo nombre del módulo se incluye:
 - a. Carpeta *uml*, con los diagramas de diseño UML.
 - b. Carpeta *doc*, con la documentación de tipo Doxygen.
 - c. Carpeta *bin*, con los ejecutables.
 - d. Códigos fuente en carpetas (según el caso): *client*, *server*, *guiClient*.
- (2) La aplicación para el tratamiento de imágenes se ha denominado *ArtificialVision*, en el directorio del mismo nombre se incluye:
 - a. Carpeta *uml*, con el diagrama UML.
 - b. Carpeta *doc*, con la documentación de tipo Doxygen.
 - c. Carpeta *bin*, con la última versión del instalable (*ArtificialVisionInstallerV03.exe*).
 - d. Carpeta *dll*, con los archivos requeridos en la instalación de la aplicación, en caso de no estar presentes en el sistema.
 - e. Carpeta *libs*, con las versiones de *cvBlobsLib* compiladas.
 - f. Carpeta *include*, con las cabeceras necesarias para el uso de *cvBlobsLib*.
 - g. En el directorio raíz se puede encontrar:
 - i. El código fuente
 - ii. El generador del instalable, denominado *setup.nsi*
 - iii. El archivo *Readme.txt*, que formará parte de la instalación a título informativo sobre las librerías instaladas

5.2 Conjuntos cliente-servidor utilizados

A lo largo del desarrollo se ha trabajado con diferentes servidores y clientes, que han sido de utilidad para diferentes etapas y que han servido para valorar las capacidades reales de la arquitectura que se pretende desarrollar.

5.2.1 Pruebas de comunicación con el software de MobileRobots

- (1) Conjunto instalado con la distribución de ARIA: *serverDemo* – *clientDemo*
- (2) Servidores instalados en ARNL: *arnlServer* y *sonarnlServer*, con el cliente *MobileEyes*

5.2.2 Cliente y servidor desarrollados a medida: *ModuloRoboticaV0*

Situado en el directorio *ModuloRoboticaV0*, está compuesto por:

- (1) Cliente por consola: con él se consiguen mandar las coordenadas de destino y la distancia mínima a la que se detendría el robot, lo que ha requerido el desarrollo de un servidor que se encargara de recoger estos datos para procesarlos de la forma deseada. Este cliente contiene dos clases:
 - a. *InputHandler*: Encargada de la entrada de datos procedentes de la consola y del envío al servidor.
 - b. *OutputHandler*: Encargada de la salida de datos por consola y de la recepción de paquetes procedentes del servidor.

InputHandler	OutputHandler
<pre>+goalX: int +goalY: int +minDist: int #myClient: ArClientBase #myKeyHandler: ArKeyHandler #myPrinting: bool #myUpCB: ArFuncorC<InputHandler> #myDownCB: ArFuncorC<InputHandler> #myLeftCB: ArFuncorC<InputHandler> #myRightCB: ArFuncorC<InputHandler> #mySpaceCB: ArFuncorC<InputHandler> #myGoToCB: ArFuncorC<InputHandler> #myListDataCB: ArFuncorC<InputHandler> #myLogTrackingTerseCB: ArFuncorC<InputHandler> #myLogTrackingVerboseCB: ArFuncorC<InputHandler> #myResetTrackingCB: ArFuncorC<InputHandler></pre>	<pre>#myX: double #myY: double #myTh: double #myVel: double #myRotVel: double #myVoltage: double #mySonarNumber: int #mySonarReading: std::vector<int> #myClient: ArClientBase #myHandleOutputCB: ArFuncor1C<OutputHandler, ArNetPacket *> #myNeedToPrintHeader: bool</pre>
<pre><<create>>-InputHandler(client: ArClientBase, keyHandler: ArKeyHandler) <<destroy>>-InputHandler(: void) +up(: void): void +down(: void): void +left(: void): void +right(: void): void +space(: void): void +goTo(: void): void +listData(): void +logTrackingTerse(): void +logTrackingVerbose(): void +resetTracking(): void</pre>	<pre><<create>>-OutputHandler(client: ArClientBase) <<destroy>>-OutputHandler(: void) +handleOutput(packet: ArNetPacket): void</pre>

- (2) Servidor, con una estructura similar, que utiliza un algoritmo simple para establecer la velocidad y el giro del robot de forma reactiva:
- InputHandler: Encargada de la entrada de datos procedentes del cliente y del envío al robot.
 - OutputHandler: Encargada del envío de datos al cliente y de la recepción de paquetes procedentes del robot.

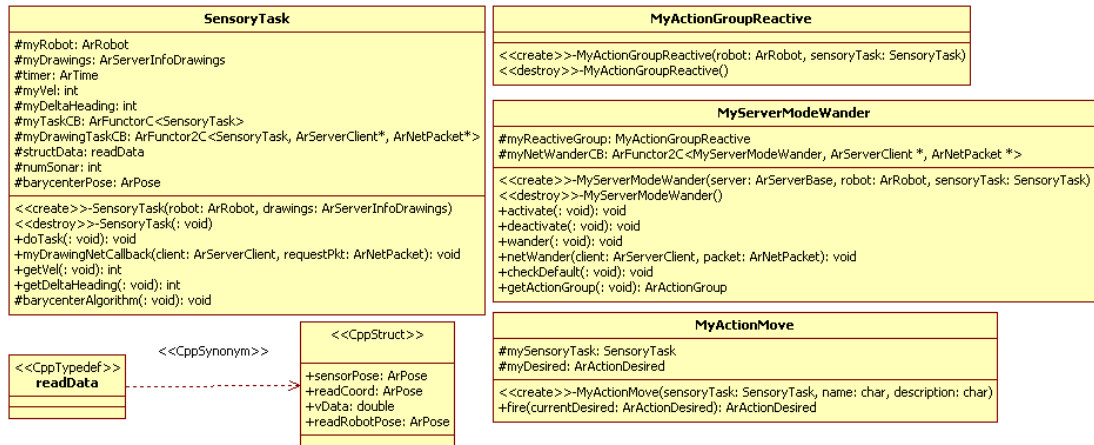
InputHandler	OutputHandler
<pre>#myRobot: ArRobot #myServer: ArServerBase #myActionInput: ArActionInput #myRecoverAct: ArActionStallRecover #myBumpAct: ArActionBumpers #myActionGoto: ArActionGoto #myActionAvoidFront: ArActionAvoidFront #myActionAvoidSide: ArActionAvoidSide #go: ActionGo #turn: ActionTurn #mySetVelCB: ArFuncor2C<InputHandler, ArServerClient *, ArNetPacket *> #mySetCoordCB: ArFuncor2C<InputHandler, ArServerClient *, ArNetPacket *> #myDeltaVelCB: ArFuncor2C<InputHandler, ArServerClient *, ArNetPacket *> #myDeltaHeadingCB: ArFuncor2C<InputHandler, ArServerClient *, ArNetPacket *></pre>	<pre>#myServer: ArServerBase #myRobot: ArRobot #myPacketMutex: ArMutex #myBuiltPacket: ArNetPacket #mySendingPacket: ArNetPacket #myOutputCB: ArFuncor2C<OutputHandler, ArServerClient *, ArNetPacket *> #myTaskCB: ArFuncorC<OutputHandler></pre>
<pre><<create>>-InputHandler(server: ArServerBase, robot: ArRobot) <<destroy>>-InputHandler(: void) +setVel(client: ArServerClient, packet: ArNetPacket): void +setCoord(client: ArServerClient, packet: ArNetPacket): void +deltaVel(client: ArServerClient, packet: ArNetPacket): void +deltaHeading(client: ArServerClient, packet: ArNetPacket): void +goToPosition(iCoordX: int, iCoordY: int): void</pre>	<pre><<create>>-OutputHandler(server: ArServerBase, robot: ArRobot) <<destroy>>-OutputHandler(: void) +buildOutput(: void): void +buildOutputGoTo2(: void): void +sonarPrinter(packet: ArNetPacket): void +output(client: ArServerClient, packet: ArNetPacket): void</pre>

Para iniciar la simulación, en primer lugar se arrancarí­a la simulación del robot (por ejemplo *mobilesim_map[n].bat*), a continuación se inicia el servidor y por último se arranca el cliente.

El cliente permite la teleoperación del robot, a través de los cursores de movimiento, y la activación de la tarea *goTo* (con los parámetros indicados en la ejecución).

5.2.3 Servidor desarrollado a medida: *ModuloRoboticaV1*

Situado en el directorio *ModuloRoboticaV1*, preparado para comunicarse con el cliente *MobileEyes*. Su estructura es la siguiente:



En este caso el movimiento del robot se sigue haciendo con un algoritmo de movimiento simple, incluido en el método *doTask* de la clase *SensoryTask*, mientras se hace el cálculo del centro de áreas (método *barycenterAlgorithm*) para pintar el punto mapa del cliente MobileEyes (con el método *myDrawingNetCallback*). La librería ArNetworking está preparada para la conexión con este entorno (MobileEyes) y evitar el envío personalizado de comandos, que puede convertirse en una tarea tediosa. De este modo se consigue teleoperar al robot desde un entorno gráfico (objeto *ArServerModeRatioDrive*), ofreciéndonos la opción de dibujar en el mapa (objeto *ArServerInfoDrawings*), tanto si se ha cargado en nuestro servidor (en cuyo caso las tres capas poseerán el mismo mapa: MobileSim ó Robot, Servidor, Cliente MobileEyes), como si no (pintando en un espacio vacío)

Para iniciar la simulación, en primer lugar se arranca la simulación del robot (mobilesim_map[n].bat, mobilesim_wcuadrado.bat ó mobilesim_wvarios.bat), después se arranca el robot indicándole el mapa o no (suponiendo que el mapa es de tipo ActivMedia y se encuentra en el mismo directorio que el ejecutable: server -map ./map[n].map; también se pueden almacenar los mapas en la ruta c:/MobileRobots/maps evitando tener que indicar la ruta: server -map map[n].map) y finalmente se arranca el cliente MobileEyes.

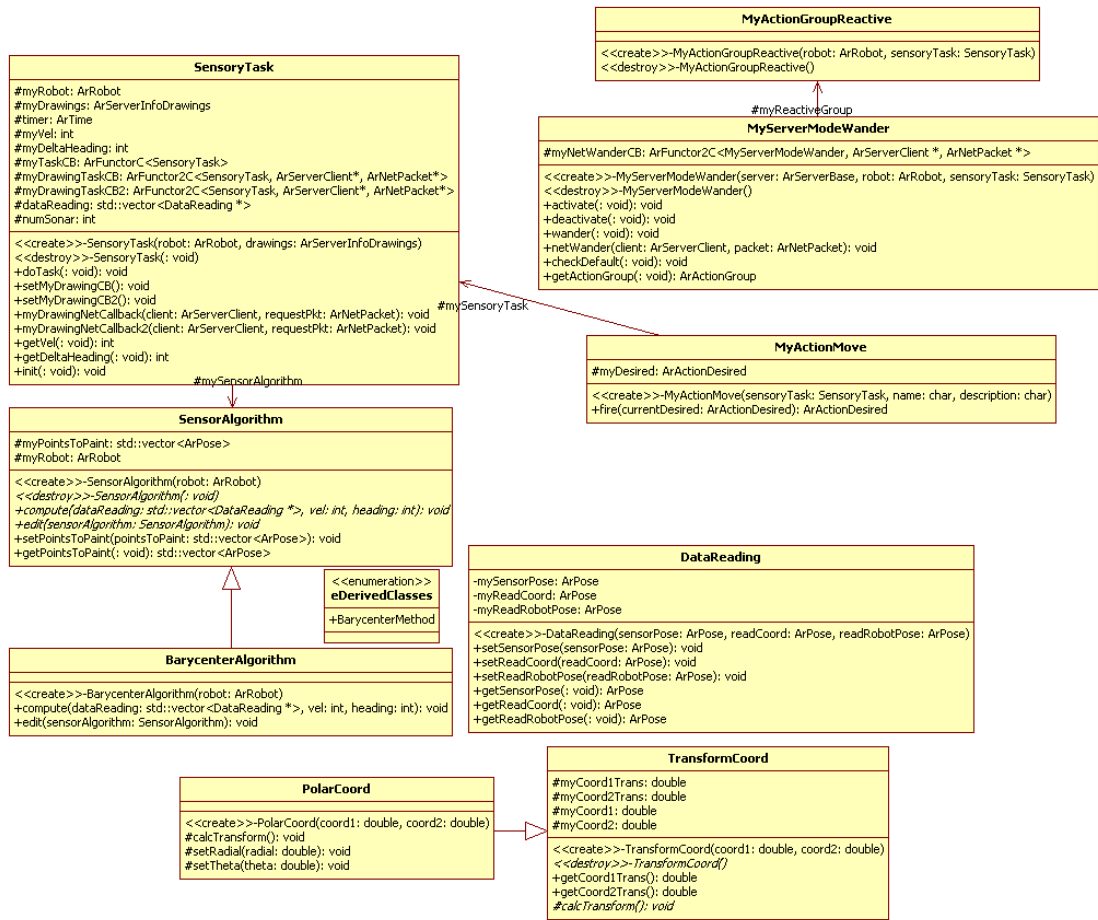
Permite a través de la teleoperación o del modo "Wander" del MobileEyes.

5.2.4 Servidor desarrollado a medida: *ModuloRoboticaV2*

Situado en el directorio *ModuloRoboticaV2*, preparado para la comunicación con el cliente MobileEyes. En este servidor se han agregado las siguientes funcionalidades respecto al anterior:

- (1) Se ha incluido una clase padre para transformar coordenadas (*TransformCoord*), que es heredada por la que realiza la transformación a coordenadas polares (*PolarCoord*). Con ella se obtienen las coordenadas polares del baricentro.
- (2) Se ha creado la clase padre de los algoritmos a aplicar (*SensorAlgorithm*), como es el del baricentro que se ha separado en otra clase (*BarycenterAlgorithm*). Se fuerza a estas clases a implementar los métodos *compute* y *edit*, el primero permite la ejecución polimórfica mientras el segundo es útil cuando se requiera modificar algo sobre el algoritmo.
- (3) Una clase para gestionar los datos de los sensores (*DataReading*) utilizados desde la clase donde se gestiona la tarea síncrona (*sensoryTask*) en la que se ejecuta el algoritmo

La estructura del servidor se muestra a continuación:



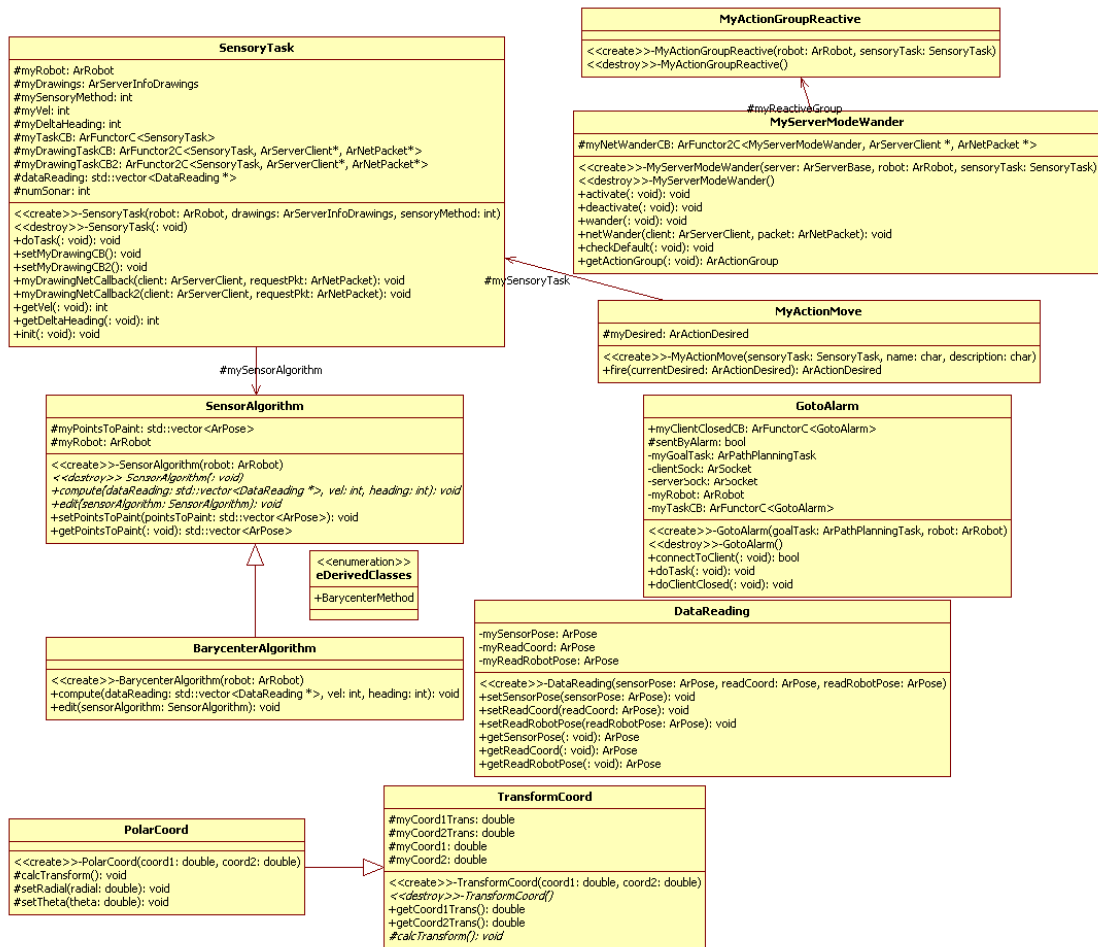
Para el movimiento del robot se hace uso del centro de áreas de forma reactiva, tratando de perseguir su posición y deteniéndose en el momento en el que lo alcanza, con un margen de 100 mm. En la misma tarea se incluye la posibilidad de pintar sobre el MobileEyes el punto actual o un histórico de las posiciones del baricentro.

Para iniciar esta simulación, en primer lugar se arranca la simulación del robot (mobilesim_map[n].bat), después se arranca el robot indicándole el mapa o no, en caso de ser de tipo stage (server -map map[n].map) y finalmente se arranca el cliente MobileEyes. Permite la interacción a través de la teleoperación o del modo "Wander" del MobileEyes, siendo en el segundo caso cuando se consigue el movimiento basado en la búsqueda del centro de áreas.

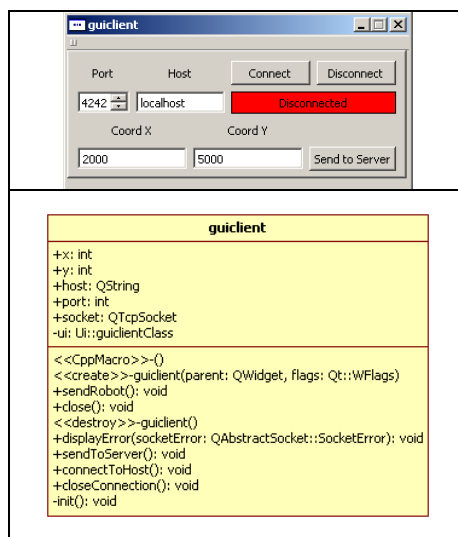
5.2.5 Servidor y Cliente desarrollados a medida: *ModuloRoboticaV3*

Servidor y cliente con interfaz de usuario QT, situados en el directorio *ModuloRoboticaV3*, preparados para trabajar conjuntamente con el cliente MobileEyes.

- (1) Servidor: En el que se han añadido dos tareas que tienen como objeto la planificación de la ruta de forma deliberativa (desde el *main*), una de localización y otra para la planificación, siempre y cuando se haya incluido el mapa al arrancar el servidor (de lo contrario la tarea devuelve un mensaje de objetivo fallido), manteniéndose las posibilidades del servidor anterior. Para planificar la ruta hacia unas coordenadas determinadas se ha creado la clase *gotoAlarm*, que se encarga de escuchar y leer por el puerto 4242 las coordenadas objetivo. La estructura es la siguiente:



(2) Cliente QT: En este caso el cliente desarrollado consiste en una interfaz que facilita el envío de coordenadas objetivo al servidor. Para enviar las coordenadas, basta indicar el puerto y el servidor, conectar, indicar las coordenadas de destino y enviar al servidor. En la consola del servidor saldrá una advertencia al recibirlas y tratará de planificarla. El interfaz QT y la estructura son las siguientes:



Para iniciar la simulación, en primer lugar se arranca la simulación del robot, en segundo lugar se arranca el servidor (queda a la espera de la conexión del cliente

por el puerto 4242), en tercer lugar se arranca el cliente QT y se conecta (en este momento se activa la escucha para la conexión del cliente MobileEyes), en cuarto lugar se arranca el MobileEyes. Al utilizar la ruta planificada, el algoritmo del baricentro se podría utilizar como mecanismo reactivo en el conjunto arquitectónico híbrido (este tipo de pruebas no se han llevado a cabo en este trabajo). Si se utiliza el modo *Wander* del MobileEyes, el método del baricentro se utiliza como mecanismo reactivo en una arquitectura reactiva.

5.2.6 Servidor y Cliente desarrollados a medida: *ModuloRoboticaV4*

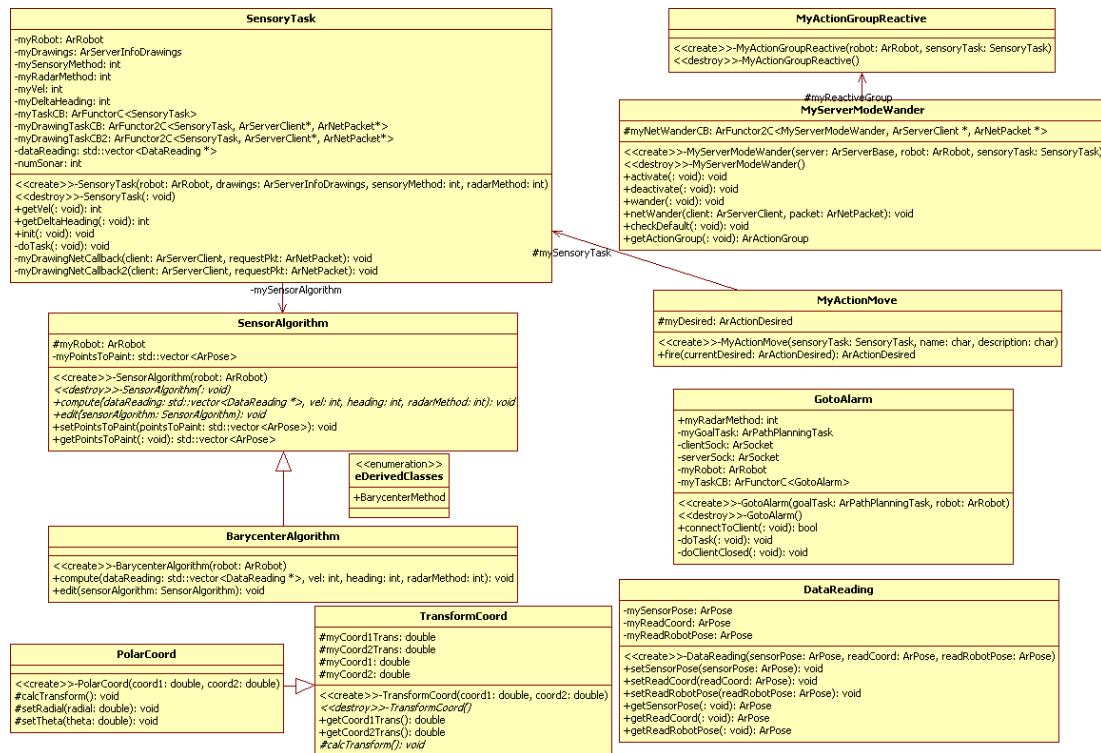
Servidor y cliente con interfaz gráfico de usuario QT, situados en el directorio *ModuloRoboticaV4*, de estructura similar al anterior pero con algunas mejoras tanto en el servidor como en el cliente.

- (1) Servidor: Las diferencias estructurales respecto al módulo anterior están relacionadas con la visibilidad de las variables. Sin embargo, merece la pena observar el esquema de nuevo cuya diferencia funcional es sutil, consistiendo en la inclusión de una variable para seleccionar la lectura de los radares. Esta variable se actualizará desde la clase *GotoAlarm*, cuya tarea es la que se encarga de la lectura de los datos del socket cliente para añadir la posición objetivo a la planificación de la ruta, siendo utilizada posteriormente desde el algoritmo sensor (en este caso el del baricentro, que es el único implementado).

Del mismo modo que el servidor del *ModuloRoboticaV3*, una vez se inicia queda a la espera de la conexión del cliente por el puerto 4242, por lo que hasta que no se conecte este cliente no estará disponible para conectarse con el cliente MobileEyes. Justo antes de quedar a la espera se agrega una funcionalidad que afectará a las pruebas desde el MobileEyes, habilitando el uso de los "goals" referenciados en el propio mapa o seleccionados en tiempo de ejecución (sobre el mapa del MobileEyes, con la tecla "control" pulsada seleccionando un punto con el ratón).

Por último mencionar que, al cambiar el método de radar (uso de todos los radares o únicamente los frontales) desde el cliente conectado al puerto 4242, también quedará reflejado en los puntos dibujados sobre el mapa del MobileEyes, pudiéndose comparar las diferencias entre los patrones de uno u otro método. El segundo método puede ser utilizado cuasi-indefinidamente a través del modo "Wander" del MobileEyes, debido a que bajo muy pocas circunstancias el robot va a conseguir encontrar el baricentro obtenido de la lectura de los radares frontales, no habiéndose dado este hecho en ninguna de las pruebas realizadas.

El esquema es el siguiente:



(2) Cliente QT: Este nuevo cliente se mantiene continuamente a la escucha de los mensajes del servidor, información que va incluyendo en el cuadro de texto (enmarcado en negro en Fig. 5.2.6.1), relacionados fundamentalmente con el comportamiento de la planificación. Los mensajes de este mismo cuadro también dan la información de los datos enviados hacia el servidor, esto es, las coordenadas de posicionamiento objetivo (enmarcadas en amarillo en Fig. 5.2.6.1) y el método radar (enmarcado en rojo en Fig. 5.2.6.1) seleccionado para el cálculo sensorial. El interfaz y la estructura son las siguientes:

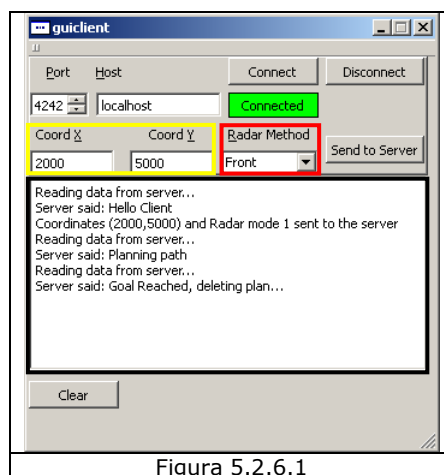


Figura 5.2.6.1

```

guiclient
+ x: int
+ y: int
+ r: int
+ host: QString
+ port: int
+ socket: QTcpSocket
- ui: Ui::guiclientClass

<<CppMacro>>-()
<<create>>-guiclient(parent: QWidget, flags: Qt::WFlags)
+ sendRobot(): void
+ close(): void
<<destroy>>-guiclient()
+ displayError(socketError: QAbstractSocket::SocketError): void
+ readFromServer(): void
+ sendToServer(): void
+ connectToHost(): void
+ closeConnection(): void
- init(): void

```

5.3 Aplicación para el tratamiento de imágenes

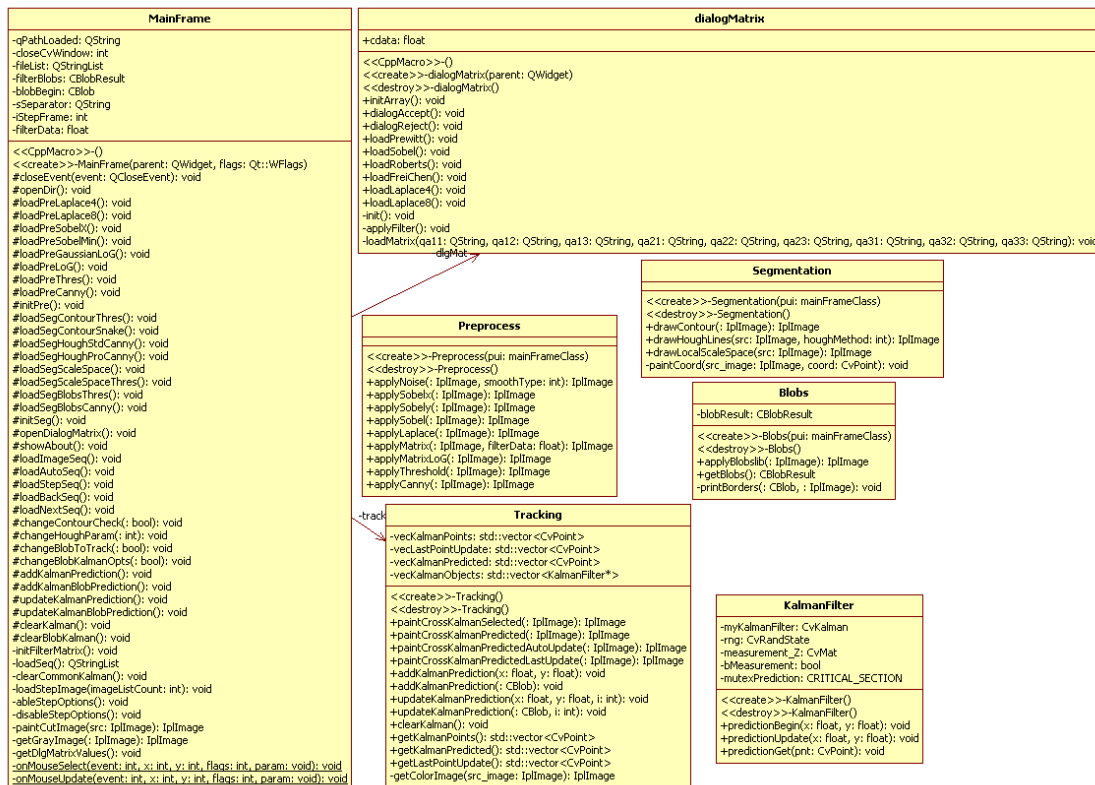
Las tareas relacionadas con el seguimiento de objetos, de utilidad en muchos campos científicos, tienen en común un conjunto de etapas que podrían agruparse en tres pasos fundamentales (un estado del arte puede ser encontrado, por ejemplo, en [28]):

- (1) Identificar los objetos de interés.
- (2) Seguimiento de los objetos a lo largo de la secuencia.
- (3) Analizar el seguimiento para reconocer su comportamiento.

La aplicación desarrollada se ha tratado de dividir en diferentes etapas que permitan reutilizar los diferentes módulos cuando:

- A. El objetivo sea la detección de otro tipo de situaciones.
- B. Se quiera ampliar la funcionalidad con otros filtros, algoritmos de aprendizaje, etc.

El esquema general es el siguiente:



En el momento de redacción del trabajo, las etapas de procesado de la imagen se han dividido en (Fig. 5.3.1):

- 1) Preproceso (Preprocess, enmarcado en rojo en Fig. 5.3.1), que permite

- a. Recortar la imagen (marco negro en Fig. 5.3.2).
- b. Filtros de ruido (marco naranja en Fig. 5.3.2).
- c. Filtros para la detección de bordes (marco violeta en Fig. 5.3.2).
- 2) Segmentación (Segmentation and Blobs, enmarcado en verde en Fig. 5.3.1)
 - a. Detección de contornos (marco negro en Fig. 5.3.3).
 - b. Detección de líneas con Hough (marco naranja en Fig. 5.3.3).
 - c. Detección de los extremos Scale-Space (marco violeta en Fig. 5.3.3).
 - d. Detección de blobs with blobslib (marco amarillo en Fig. 5.3.3).
- 3) Seguimiento (Tracking, , enmarcado en azul en Fig. 5.3.1)
 - a. Algoritmo lineal de Kalman aplicado a puntos seleccionados (marco negro en Fig. 5.3.4).
 - b. Alinear y pintar (marco naranja en Fig. 5.3.4).
 - c. Algoritmo lineal de Kalman aplicado los blobs detectados (marco violeta en Fig. 5.3.4).

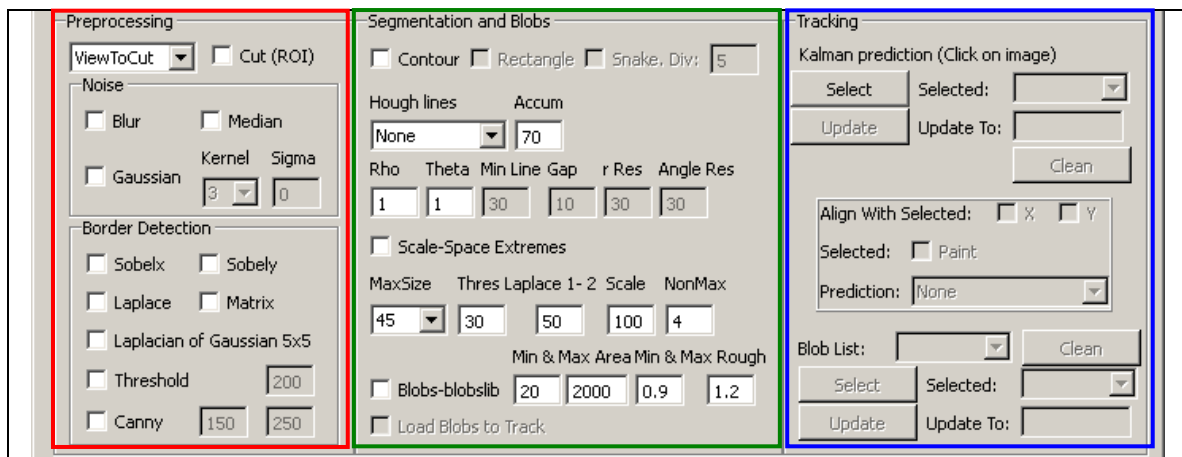
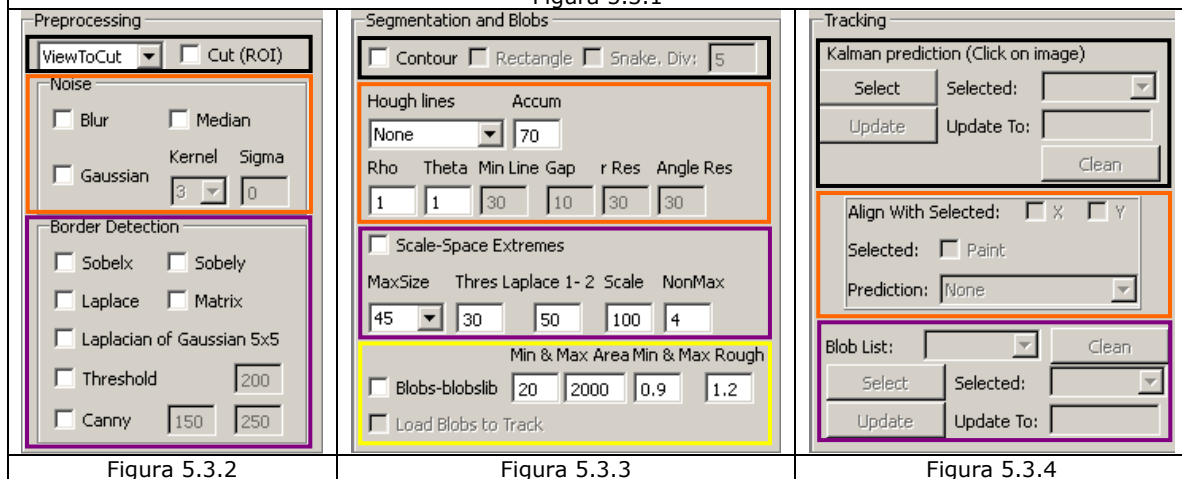


Figura 5.3.1



5.3.1 Etapa de preprocesado

Con la primera opción de esta etapa, se puede establecer como región de interés una fracción prefijada de la imagen, optimizando el tratamiento al limitarse el procesado únicamente a esta zona. Para ello es necesario seleccionar la fracción en el desplegable y marcar la opción *Cut*. Esta opción está codificada en la clase principal *MainFrame*, bajo el nombre *paintCutImage*. La decisión de dividir la escena en $1/2$ ó $2/3$ está relacionada con el interés particular de este estudio en la zona del asfalto. No obstante, el control de esta selección se encuentra bien delimitado en el método indicado (*paintCutImage*), donde se podría personalizar para otros casos particulares o, de forma más general, extender al interfaz de

usuario desde donde se introducirían los parámetros de recorte deseados para la altura y la anchura.

El resto de opciones, los filtros de ruido y de detección de bordes, se han encapsulado en la clase *Preprocess*:

Preprocess
<pre> <<create>>-Preprocess(pui: mainFrameClass) <<destroy>>-Preprocess() +applyNoise(: IplImage, smoothType: int): IplImage +applySobelx(: IplImage): IplImage +applySobely(: IplImage): IplImage +applySobel(: IplImage): IplImage +applyLaplace(: IplImage): IplImage +applyMatrix(: IplImage, filterData: float): IplImage +applyMatrixLoG(: IplImage): IplImage +applyThreshold(: IplImage): IplImage +applyCanny(: IplImage): IplImage </pre>

Todos los filtros de ruido se han apoyado en la función *cvSmooth*, mientras que los filtros de detección de ruido lo han hecho en: *cvSobel*, *cvLaplace*, *cvFilter2D*, *cvThreshold* y *cvCanny*.

Los filtros de ruido se aplican por defecto con una matriz de convolución de 3x3, sin embargo, la opción de Gauss habilita la posibilidad de modificar el tamaño (a través del desplegable *Kernel*) y, aún no siendo muy intuitivo, hay que tener en cuenta que el valor que quede seleccionado será el que se aplique a todos estos los filtros, aún cuando quede desactivado al desmarcar el filtro de Gauss.

Cuando se selecciona *Blur*, lo que se está haciendo es sumar los valores de los píxeles vecinos agrupados de 9 en 9 (en el caso por defecto) y dividiéndolos después por 9.

Al seleccionar la opción *Median* se reemplaza la vecindad correspondiente por el valor del pixel central.

Cuando se selecciona la opción *Gaussian* se aplica la matriz de convolución correspondiente al tamaño indicado en el desplegable *Kernel* (al igual que en los casos anteriores), usando la desviación típica introducida en la opción *Sigma*, si es distinta de 0. En caso de ser 0 es calculada del siguiente modo: $\sigma = 0,3 \cdot (n/2 - 1) + 0,8$, siendo n el tamaño de la matriz de convolución (si no fuese simétrica sería distinta para cada eje).

El filtro de Sobel se puede aplicar de forma independiente a cada eje de coordenadas:

- Aplicándolo al eje de abscisas, con *Sobelx*, se detectarán mejor los contornos verticales. En este caso la matriz de convolución aplicada es:

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

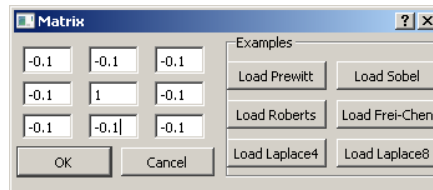
- Aplicándolo al eje de ordenadas, con *Sobely*, se detectarán mejor los contornos horizontales. La matriz de convolución se corresponde con:

$$\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Cuando se aplican ambos filtros simultáneamente se ha utilizado la función *cvMin* (en el método *applySobel* de la clase *Preprocess*) para seleccionar el pixel de menor valor de los resultantes de aplicar cada convolución.

El filtro de Laplace procesa la imagen con la matriz de convolución de vecindad-4.

Cuando se utiliza la opción *Matrix*, se pueden seleccionar los valores de la matriz 3x3 con la que procesar la correlación cruzada, muy similar a la convolución (la diferencia fundamental es que da como resultado diferencias entre las distribuciones de probabilidad, en lugar de sumas). Para ello, existe una ventana específica para la inserción de las componentes de la matriz, mostrada a través del menú de la barra de herramientas *MatrixFilter* → *Matrix*:



La Laplaciana de una gaussiana, aplicada con la opción *Laplacian of Gaussian 5x5*, se

obtiene a partir de la ecuación: $G(x, y) = \left(\frac{1}{\pi\sigma^4} \right) \left(\frac{x^2 + y^2}{2\sigma^2} - 1 \right) e^{-\frac{(x^2+y^2)}{2\sigma^2}}$

Considerando $\sigma^2 = 0,391$, la matriz 5x5 resultante tiene los siguientes valores:

$$\begin{pmatrix} 0 & 0,02 & 0,05 & 0,02 & 0 \\ 0,02 & 0,25 & 0,16 & 0,25 & 0,02 \\ 0,05 & 0,16 & -2,08 & 0,16 & 0,05 \\ 0,02 & 0,25 & 0,16 & 0,25 & 0,02 \\ 0 & 0,02 & 0,05 & 0,02 & 0 \end{pmatrix}$$

Se puede normalizar con un factor 50 a la siguiente*:

$$\begin{pmatrix} 0 & 1 & 3 & 1 & 0 \\ 1 & 13 & 8 & 13 & 1 \\ 3 & 8 & -104 & 8 & 3 \\ 1 & 13 & 8 & 13 & 1 \\ 0 & 1 & 3 & 1 & 0 \end{pmatrix}$$

[* Esta matriz ha sufrido muchas modificaciones y puede no coincidir con la última versión del ejecutable distribuida en Sourceforge. Estos valores se corresponden con los utilizados para las pruebas en este trabajo. Para replicarlos bastaría ajustarlos en el método *applyMatrixLoG* de la clase *Preprocess*, estableciendo los valores de la matriz con la función *cvmSet* – que está dispuesta en el código siguiendo el orden matricial.]

En el caso del filtro *Threshold*, se trata de un filtro binario aplicado a cada pixel que da como resultado una imagen en color blanco, allí donde el valor en la escala de grises supera el parámetro de corte (valor introducido en el cajón de texto que se encuentra a la derecha), y negro, en las zonas con valor menor. Este es un filtro muy práctico en este desarrollo a pesar de su simplicidad, puesto que la mayoría de

las líneas que delimitan el contorno de la carretera son de color blanco o, en todo caso, de colores claros (en contraste con el color del asfalto).

$$pixel_final = \begin{cases} 255 & \Leftarrow pixel_origen > umbral \\ 0 & \Leftarrow pixel_origen \leq umbral \end{cases}$$

Con el filtro de *Canny* se consigue filtrar entre dos parámetros límite: si los píxeles tienen un gradiente menor al límite inferior (primer parámetro) no son mostrados, en caso de ser mayor al límite superior (segundo parámetro) son mostrados, y en caso de estar comprendidos entre los dos límites sólo se mostrarán si están conectados a otro píxel que tenga un valor por encima del límite superior.

5.3.2 Etapa de segmentación y selección de objetos

Con el desarrollo de esta etapa se consiguen delimitar los posibles objetos de interés, siendo especialmente delicada por la necesidad de conocimiento del dominio en el que se ubique el problema que se desea resolver ([9], [10]). Una vez se dispone de una imagen con los contornos localizados, es imprescindible decidir qué tipo de contornos son los que nos interesan. En las escenas de conducción, por ejemplo, aparecerán bordes relacionados con los arcones de la vía pero también otros como: señales de tráfico, otros vehículos, farolas, objetos del horizonte y en los laterales, viandantes, manchas y defectos del asfalto, etc. Para suplir en la medida de lo posible esta dificultad se han tomado algunas decisiones:

- Una alta parametrización de cada uno de los algoritmos.
- Modularizar esta etapa en diferentes clases.
- Detener la distribución de código libre en este aspecto, es decir, no llegar a incluir ningún tipo de especialización en el proyecto, dejándolo en manos de cada usuario.

Los tres primeros procesos de esta etapa están encapsulados en la clase *Segmentation*:

Segmentation
<pre> <<create>>-Segmentation(pui: mainFrameClass) <<destroy>>-Segmentation() +drawContour(: IplImage): IplImage +drawHoughLines(src: IplImage, houghMethod: int): IplImage +drawLocalScaleSpace(src: IplImage): IplImage -paintCoord(src_image: IplImage, coord: CvPoint): void </pre>

La detección de contornos está basada en la función *cvFindContours*, obteniéndose la totalidad de contornos como una lista, dándose también la posibilidad de embeberlos en un rectángulo (opción *Rectangle*) y, posteriormente, ajustar la snake con un número de divisiones a elegir (basado en la función *cvSnakeImage*).

Para la detección de líneas se ha utilizado la transformada de Hough, basándolo en la función *cvHoughLines2*, que se puede aplicar bajo su formato estándar ([35]) o probabilística ([34]).

La detección basada en la teoría de "espacio escalado" ([36]) denominada *Scale-Space Extremes*, está construida utilizando la función *cvGetStarKeypoints*.

El último proceso elaborado en esta etapa hace uso de [30] para la detección de blobs, habiéndose encapsulado en la clase *Blobs*:

Blobs
-blobResult: CBlobResult
<<create>>-Blobs(pui: mainFrameClass) <<destroy>>-Blobs() +applyBlobslib(: IplImage): IplImage +getBlobs(): CBlobResult -printBorders(: CBlob, : IplImage): void

Los parámetros utilizados para delimitar el número de blobs, forman parte de la versatilidad de la librería base, que permite otros filtros adicionales. En concreto, a través del interfaz se pueden introducir el intervalo de área mínima / máxima y el intervalo de rugosidad mínima / máxima (la rugosidad se define como la relación entre el perímetro y la convexidad). Estos límites pueden ayudarnos a limitar el número de objetos detectados, que, dependiendo de los filtros aplicados en la etapa de preproceso y del tipo de escena, podrían ser excesivamente numerosos (con un efecto notable de ralentización del proceso). La última opción que aparece, denominada *Load Blobs to Track*, enlaza la detección de blobs con la etapa de seguimiento, esta conexión se controla desde la clase *MainFrame*.

Es interesante mencionar en este punto que considerar que cada blob se corresponde con un objeto simple en movimiento puede resultar una limitación demasiado restrictiva, por varios motivos (ver por ejemplo [44] en relación con el seguimiento de personas): (1) Un blob puede contener varios objetos (2) Un único objeto puede estar dividido en varios blobs (3) Los blobs pueden contener píxeles de sombras y reflexiones.

5.3.3 Etapa de seguimiento

El desarrollo de esta etapa permite ensayar con situaciones de discontinuidad de los contornos. Idealmente la situación se corresponde con la siguiente: Una vez tenemos localizados los contornos de interés, nos interesa disponer de una técnica que, ante la posible discontinuidad de aparición del objeto, prediga o anticipe de algún modo el lugar aproximado en el que está situado, a pesar de no estar presente en la imagen actual. En la situación real, la presencia continua del contorno y su identificación ya es complicada de obtener aún cuando el objeto se encuentra presente.

Apoyado en tres asunciones fundamentales (linealidad del sistema, ruido blanco y naturaleza gaussiana del mismo) se ha llevado a cabo el filtro de Kalman en la clase *KalmanFilter*, basado en la estructura *CvKalman*. Esta estructura se genera a partir de la función *cvCreateKalman*, se borra con *cvReleaseKalman* y se computa con *cvKalmanPredict* (para obtener la predicción en el instante siguiente) y *cvKalmanCorrect* (con la que se introduce la corrección con la nueva medida).

KalmanFilter
-myKalmanFilter: CvKalman -rng: CvRandState -measurement_Z: CvMat -bMeasurement: bool -mutexPrediction: CRITICAL_SECTION
<<create>>-KalmanFilter() <<destroy>>-KalmanFilter() +predictionBegin(x: float, y: float): void +predictionUpdate(x: float, y: float): void +predictionGet(pnt: CvPoint): void

El filtro de Kalman es un filtro recursivo que estima el estado de un sistema dinámico lineal, discretizado en el dominio del tiempo, a partir de medidas y acciones de control. Se asume que el estado actual verdadero cumple (I) y la relación entre la medida y el valor real viene dada por (II):

$$\begin{aligned} \text{(I)} \quad \vec{x}_k &= F_k \vec{x}_{k-1} + B_k \vec{u}_k + \vec{w}_k \\ \text{(II)} \quad \vec{z}_k &= H_k \vec{x}_k + \vec{v}_k \end{aligned}$$

Siendo:

- \vec{x}_k valor verdadero del vector de magnitudes en el estado (o momento) k .
- \vec{u}_k vector de control (no utilizado, igual a cero).
- \vec{w}_k vector de ruido del proceso, considerada como una distribución normal de media 0 y matriz de covarianza Q_k .
- \vec{z}_k vector de medidas.
- \vec{v}_k vector de ruido observado, considerada como una distribución normal de media 0 y matriz de covarianza R_k .
- F_k matriz de transición del estado k .
- B_k matriz aplicada al vector de control (no utilizado).
- H_k matriz de observación aplicada al vector verdadero para obtener la medida observada.

El filtro de Kalman debe superar dos dificultades:

1. Por un lado actuar como filtro del ruido (\vec{w}_k y \vec{v}_k que son desconocidos y no medibles).
2. Actuar como observador, puesto que H_k podría no ser invertible, no se podría inferir el estado del sistema directamente con la medida, así que habría que construirlo con los valores históricos de \vec{z}_k y los parámetros conocidos (F_k, B_k, H_k , desviaciones de \vec{w}_k y \vec{v}_k).

Pasos seguidos para su desarrollo:

- Se consideran las variables de posición (x,y), es decir $\vec{x}_k = (x_k, y_k, x'_k, y'_k)$:
 - myKalmanFilter crea la estructura de Kalman con 4 parámetros (posición y velocidad de cada coordenada) y las 2 medidas (la posición)
 - measurement_Z = \vec{z}_k Vector de medidas, de 2 posiciones
- Se inicializan las matrices de la estructura de Kalman:

$$\circ F_k \text{ a } \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Al considerar las ecuaciones iniciales de posición en función de la velocidad, tomando $dt = 1$:

$$x_1 = x_0 + x'_0 \cdot dt$$

$$y_1 = y_0 + y'_0 \cdot dt$$

$$x'_1 = x'_0$$

$$y'_1 = y'_0$$

- H_k a la identidad $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- Q_k es la matriz de covarianza del ruido, permite calcular el estado a priori de la matriz de covarianza del error (define el paso predictivo del algoritmo calculándose a partir de la del paso anterior $k-1$). $P'_k = F_k P_{k-1} F_k^T + Q_k$. Se inicializa la diagonal a valores pequeños, del orden de 10^{-5}
- R_k es la matriz de covarianza del ruido de la medida, con la que se obtiene la ganancia a través de la ecuación: $K_k = P'_k H_k^T (H_k P'_k H_k^T + R_k)^{-1}$. La matriz de covarianza del ruido de la medida se inicializa a una matriz diagonal con valores del orden de 10^{-1} .
- K_k es la Ganancia de Kalman, utilizada para calcular el valor del vector de estado cuando llegue una nueva medida y la matriz de covarianza del error corregida. La ganancia nos dice el peso con el que agregar la información nueva.
- P_k es la matriz de covarianza de error corregida, a partir de la estimada $P_k = (I - K_k H_k) P'_k$. Se inicializa a la identidad.

En la clase *KalmanFilter*, en caso de no introducirse ninguna medida a lo largo del ciclo de ejecución del objeto, se toma como nueva medida la predicción anterior (lo que conduce a una convergencia errónea, en mayor medida cuanto mayor sea el ritmo de cambio de la posición del contorno). No obstante, este ciclo de ejecución asíncrono se ha integrado posteriormente en el ciclo síncrono, facilitando la intervención manual y el control de la actualización en el tiempo (establecido a partir de los FPS o de la acción sobre el botón *Show Next Step*).

Para generar tantos objetos predictivos como se necesite se ha creado la clase *Tracking*, siendo el puente entre la clase principal *MainFrame* (que recibe la acción del usuario sobre el interfaz gráfico) y las instancias de la clase *KalmanFilter*. La clase *Tracking* dispone de varias utilidades a través de sus métodos:

- 1) Iniciar una nueva predicción (addKalmanPrediction):
 - a. Sobre puntos seleccionados directamente sobre la imagen (opción *Select* de la zona enmarcada en negro en Fig. 5.3.4)
 - b. Sobre los blobs detectados en la etapa de segmentación, una vez está seleccionada la opción *Loads Blobs to Track* (de la zona enmarcada en amarillo en Fig. 5.3.3), esta selección se puede realizar usando el desplegable *Blob List* junto con la opción *Select* (de la zona enmarcada en violeta en Fig. 5.3.4).
- 2) Actualizar la medida de una predicción iniciada anteriormente (updateKalmanPrediction):
 - a. Sobre los puntos seleccionados, previa elección en el combo *Selected* de la zona enmarcada en negro en Fig. 5.3.4, se procede pulsando el

botón *Update* de esta misma zona y, acto seguido, se presiona allí donde queramos fijar la medida. Para siguientes actualizaciones no es necesario pulsar de nuevo el botón *Update*, siempre y cuando no se vuelvan a seleccionar nuevos puntos con la opción *Select*.

- b. Sobre los blobs seleccionados, actuando de modo análogo pero en la zona enmarcada en violeta en Fig. 5.3.4. En lugar de seleccionar la medida sobre la imagen tras pulsar el botón *Update*, se trata de seleccionar el blob que actualizará la medida del desplegable *Blob List*, seleccionar el blob originalmente seleccionado del desplegable *Selected* y a continuación pulsar *Update*. Lógicamente, este conjunto de acciones nos va a llevar un tiempo, así que será viable cuando estemos en el modo *Show Step-By-Step* o, en caso contrario, cuando los FPS tengan un valor suficientemente alto (de lo contrario la acción manual se hace inaplicable).
- 3) Obtener el conjunto de puntos seleccionados para la predicción (*getKalmanPoints*), el último conjunto de puntos predictivos correspondiente a cada uno de los seleccionados (*getKalmanPredicted*) y el último conjunto de puntos utilizados para la actualización de la medida correspondiente al conjunto de puntos seleccionados (*getLastPointUpdate*)
 - 4) Pintar una cruz sobre la imagen en las coordenadas de los puntos correspondientes a (todas las opciones son aplicables tanto a la selección manual con el puntero sobre la imagen como a la selección a través de los Blobs, en cuyo caso la coordenada se corresponde con el punto central):
 - a. Los puntos seleccionados inicialmente para hacer el seguimiento (*paintCrossKalmanSelected*). Desde el interfaz de usuario esta opción se habilita marcando la opción *Paint* referido a *Selected* de la zona central (marco naranja en Fig. 5.3.4).
 - b. Los puntos correspondientes a la última predicción, sin ejecutar de nuevo el ciclo del filtro (*paintCrossKalmanPredicted*). Esto se consigue eligiendo *Just Paint* en el desplegable *Prediction* de la ya mencionada zona central.
 - c. Los puntos correspondientes a la última predicción, volviendo a lanzar de nuevo el ciclo de Kalman, actualizando la medida con la predicción obtenida a partir de la ejecución del ciclo anterior (*paintCrossKalmanPredictedAutoUpdate*). Se puede utilizar seleccionando, en el desplegable indicado en el apartado b., el valor *Paint&Auto-Update*.
 - d. Los puntos correspondientes a la última predicción, volviendo a ejecutar el ciclo y, en este caso, actualizando la medida con la última medida introducida por el usuario (*paintCrossKalmanPredictedLastUpdate*). Para hacer uso de esta función desde el interfaz hay que seleccionar, del desplegable, el valor *Paint&LastUpdate*.

Cualquiera de las opciones de actualización del ciclo, ya sea a través de la última predicción o de la última medida (4c, 4d), conduce, como es de esperar, a una convergencia sobre esta posición, en mayor medida cuanto mayor sea el número de ciclos en el que se repita.

A continuación se muestra la estructura de la clase *Tracking*:

Tracking
-vecKalmanPoints: std::vector<CvPoint> -vecLastPointUpdate: std::vector<CvPoint> -vecKalmanPredicted: std::vector<CvPoint> -vecKalmanObjects: std::vector<KalmanFilter*>
<<create>>-Tracking() <<destroy>>-Tracking() +paintCrossKalmanSelected(: IplImage): IplImage +paintCrossKalmanPredicted(: IplImage): IplImage +paintCrossKalmanPredictedAutoUpdate(: IplImage): IplImage +paintCrossKalmanPredictedLastUpdate(: IplImage): IplImage +addKalmanPrediction(x: float, y: float): void +addKalmanPrediction(: CBlob): void +updateKalmanPrediction(x: float, y: float, i: int): void +updateKalmanPrediction(: CBlob, i: int): void +clearKalman(): void +getKalmanPoints(): std::vector<CvPoint> +getKalmanPredicted(): std::vector<CvPoint> +getLastPointUpdate(): std::vector<CvPoint> -getColorImage(src_image: IplImage): IplImage

Cuando seleccionamos manualmente un punto sobre la imagen, en ocasiones es interesante observar únicamente cómo ese punto oscila a lo largo de uno de los ejes de coordenadas.

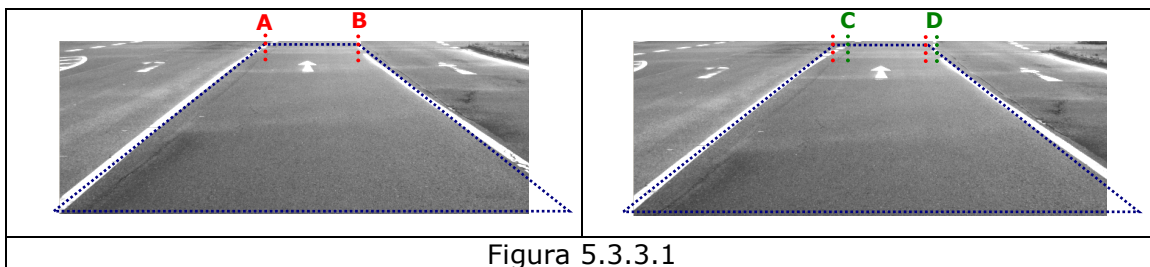


Figura 5.3.3.1

En el caso presentado en la Fig. 5.3.3.1 (mencionado anteriormente), deseamos aplicar la predicción sobre el eje de abscisas ante una posible discontinuidad en la detección. Para poder llevar a cabo este tipo de predicción se ha incluido la opción del interfaz *Align With Selected* situado en la zona central de esta etapa. Esto posibilita que al actualizar una medida (a través de la selección manual, no a través de la detección de blobs), la coordenada de actualización marcada se mantenga idéntica a la del punto original. Es decir, continuando con el ejemplo representado en la Fig. 5.3.3.1, para estimar a lo largo del eje de abscisas habría que mantener constante la componente del eje de ordenadas, marcando la opción *Y* de *Align With Selected*. Este alineamiento es controlado en la clase *MainFrame*, en el método que responde al evento de selección sobre la imagen (*onMouseUpdate*).

Es en esta misma clase donde se controla el evento de selección sobre la imagen (*onMouseSelect*), así como el resto de opciones para el control del ciclo general de procesado y otras cuestiones importantes para las pruebas como la carga (*load[etapa][nombre]* siendo *etapa*=Pre (preproceso), Seg (segmentación) y *nombre*=nombre de la opción) y la inicialización (*initPre* e *initSeg*) de las configuraciones predefinidas:

MainFrame

-qPathLoaded: QString
-closeCvWindow: int
-fileList: QStringList
-filterBlobs: CBlobResult
-blobBegin: CBlob
-sSeparator: QString
-iStepFrame: int
-filterData: float

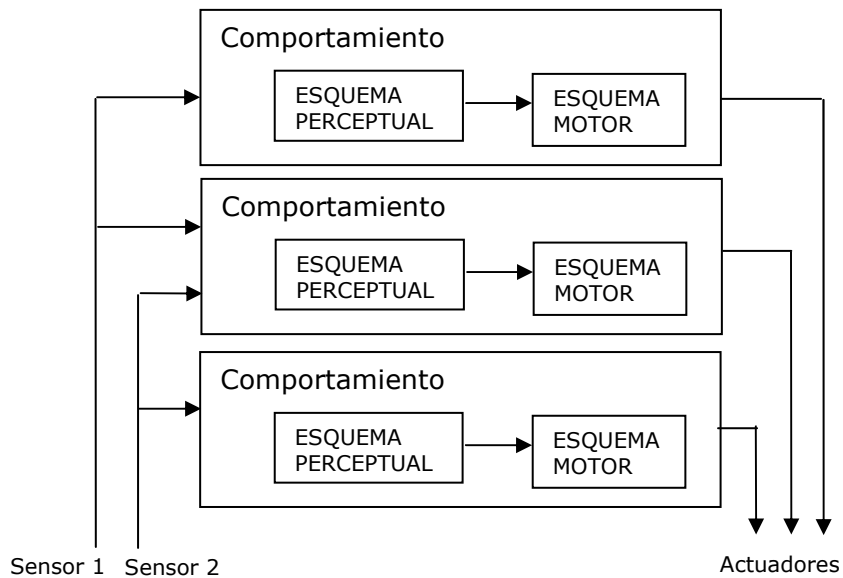
```
<<CppMacro>>-()  
<<create>>-MainFrame(parent: QWidget, flags: Qt::WFlags)  
#closeEvent(event: QCloseEvent): void  
#openDir(): void  
#loadPreLaplace4(): void  
#loadPreLaplace8(): void  
#loadPreSobelX(): void  
#loadPreSobelMin(): void  
#loadPreGaussianLoG(): void  
#loadPreLoG(): void  
#loadPreThres(): void  
#loadPreCanny(): void  
#initPre(): void  
#loadSegContourThres(): void  
#loadSegContourSnake(): void  
#loadSegHoughStdCanny(): void  
#loadSegHoughProCanny(): void  
#loadSegScaleSpace(): void  
#loadSegScaleSpaceThres(): void  
#loadSegBlobsThres(): void  
#loadSegBlobsCanny(): void  
#initSeg(): void  
#openDialogMatrix(): void  
#showAbout(): void  
#loadImageSeq(): void  
#loadAutoSeq(): void  
#loadStepSeq(): void  
#loadBackSeq(): void  
#loadNextSeq(): void  
#changeContourCheck(: bool): void  
#changeHoughParam(: int): void  
#changeBlobToTrack(: bool): void  
#changeBlobKalmanOpts(: bool): void  
#addKalmanPrediction(): void  
#addKalmanBlobPrediction(): void  
#updateKalmanPrediction(): void  
#updateKalmanBlobPrediction(): void  
#clearKalman(): void  
#clearBlobKalman(): void  
-initFilterMatrix(): void  
-loadSeq(): QStringList  
-clearCommonKalman(): void  
-loadStepImage(imageListCount: int): void  
-ableStepOptions(): void  
-disableStepOptions(): void  
-paintCutImage(src: IplImage): IplImage  
-getGrayImage(: IplImage): IplImage  
-getDlgMatrixValues(): void  
-onMouseSelect(event: int, x: int, y: int, flags: int, param: void): void  
-onMouseUpdate(event: int, x: int, y: int, flags: int, param: void): void
```

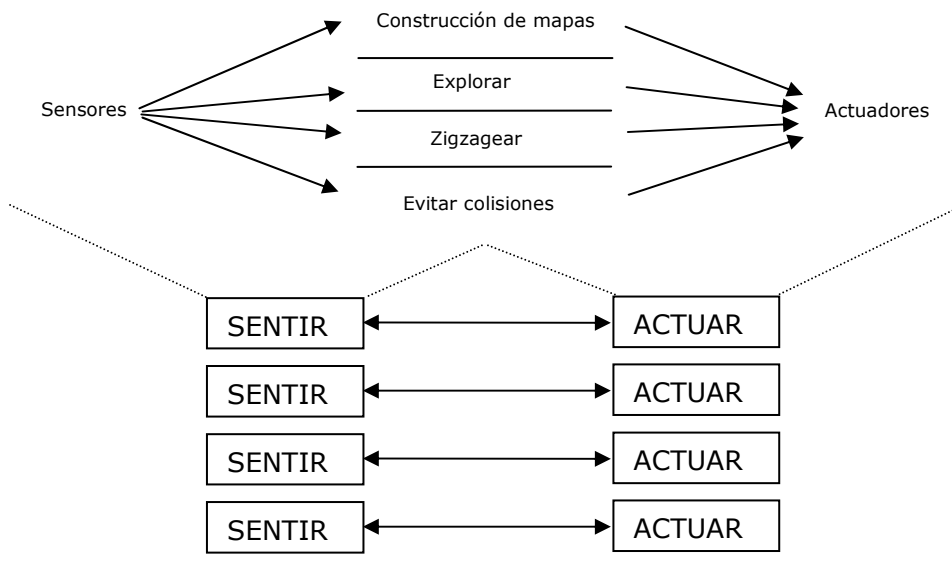
6 PROGRAMACIÓN ORIENTADA A OBJETOS

En psicología, los esquemas se refieren a estructuras mentales que representan aspectos del mundo real, idea vinculada con el estudio en el modo en el que la memoria es almacenada y después recuperada, trabajos iniciados por Bartlett en torno a 1930. La programación orientada objetos es capaz de representar la abstracción que representa a estos esquemas a partir de un conjunto de datos, capaces de almacenar el conocimiento, y unos métodos o funciones, que recogen la capacidad para procesar la información percibida y actuar.

6.1 Aplicación a sistemas reactivos

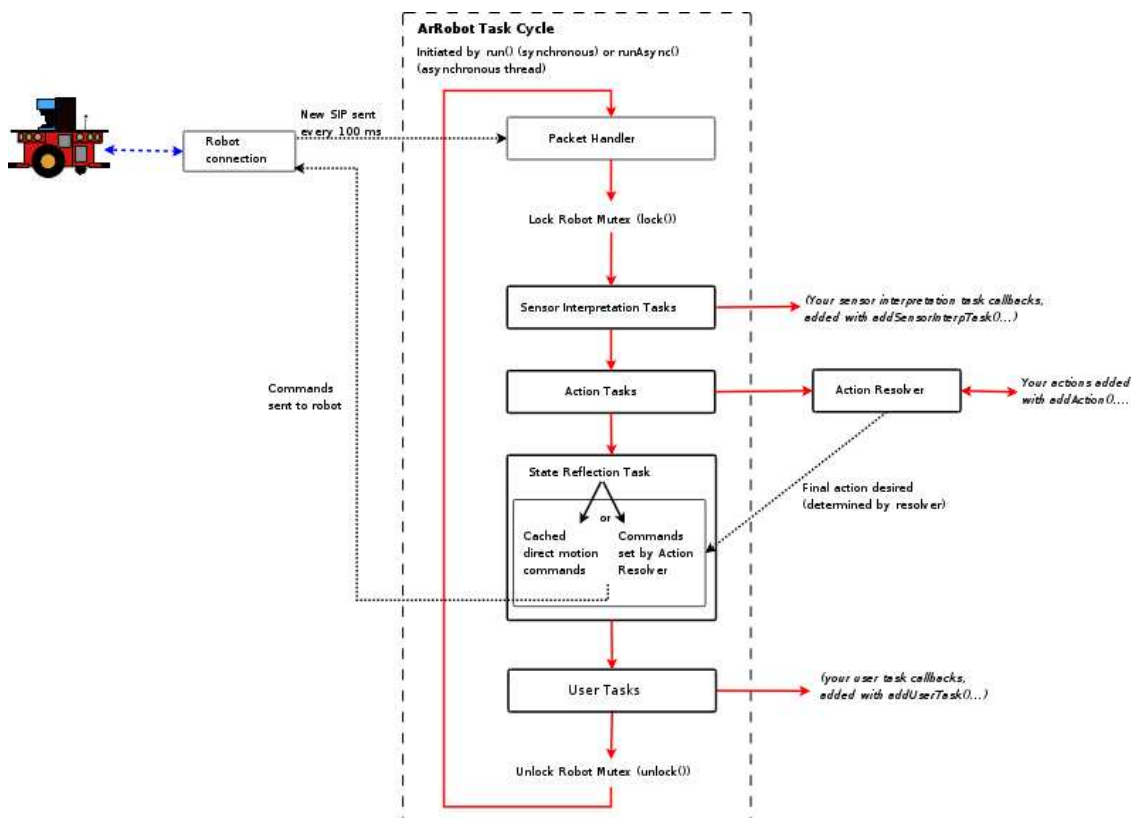
En un sistema reactivo - proceso vertical - el control se realiza partiendo de los comportamientos, que se pueden dividir en un esquema de percepción y un esquema motor. La suma de éstos es lo que controlaría al robot, con cada uno actuando de forma independiente, y teniendo acceso rápido a los datos de los sensores. Cada comportamiento tiene asociado los datos de un sensor, desconociendo la existencia de otros comportamientos, permitiendo la utilización fusionada de los sensores:





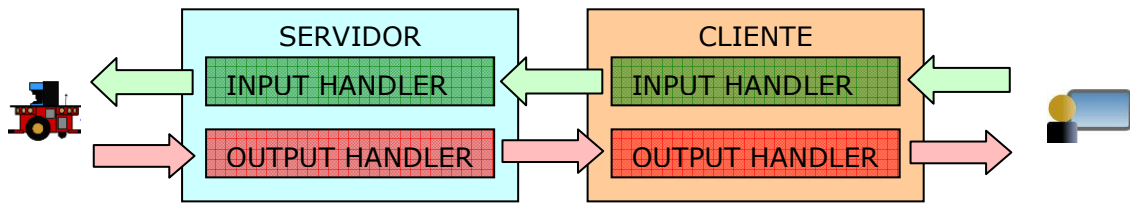
Proceso vertical (Brooks, [8])

La arquitectura propuesta a través de la librería ARIA es capaz de incluir estos esquemas a través de tareas de interpretación de los datos del sensor y acciones, que se pueden incorporar a un ciclo sincronizado de tareas (el ciclo se puede invocar en el mismo hilo de ejecución o en un hilo independiente):



Ya se ha descrito el primero de los módulos desarrollados a medida para este trabajo (*ModuloRoboticaV0*) que, a través del servidor a la espera de las órdenes de posicionamiento y de distancia enviadas desde el cliente, ha servido para desarrollar con muy poco esfuerzo un esquema reactivo de movimiento hacia un punto, únicamente utilizando el potencial de las funciones que posee las librerías

ARIA y ArNetworking. El esquema de flujo de la información para este módulo es el siguiente:



En el servidor se añaden las acciones, cuya funcionalidad fundamental reside en la utilización de un "ir a", a través del objeto *ArActionGoto*, conjuntamente con otros u otros para evitar obstáculos, con una prioridad mayor a la asignada al algoritmo "ir a".

Una vez se ha iniciado el simulador MobileSim del robot P3AT con el mapa deseado, se ejecuta el servidor que automáticamente conectará con el simulador (siempre y cuando haya una única instancia del mismo, usándose el puerto por defecto), tras lo que se puede ejecutar el cliente con los parámetros de posicionamiento y distancia mínima de detección (por ejemplo: `client -x 0 -y 4000 -d 100`, utilizando el mapa cuadrado.world, arrancado con `mobilesim_wcuadrado.bat`). Una vez se encuentra arrancado el cliente, que conectará automáticamente con el servidor, pulsando la tecla *g* se enviarán los paquetes con los parámetros introducidos hacia el servidor, que se encargará de añadir las acciones en el ciclo síncrono del robot y, por tanto, ejecutar reactivamente las acciones teniendo en cuenta su prioridad.

Pruebas:

- Se ejecuta MobileSim con el mapa de tipo stage: `mobilesim_wcuadrado.bat`
- Se ejecuta el servidor: `server`
- Se ejecuta el cliente con las coordenadas y la distancia mínima en milímetros
 - `client -x 0 -y 4000 -d 100`
Esta ejecución permite comprobar el funcionamiento del algoritmo *go* a lo largo del pasillo.
 - `client -x -8000 -y 0 -d 100`
Con este ejemplo se observa cómo el robot, al no disponer de la información del mapa y tener que ir justo al otro extremo, oculto tras el objeto cuadrado, va continuamente cabeceando hasta encontrar los pasillos, mostrando la dificultad del movimiento puramente reactivo.

A partir del *ModuloRoboticaV1* en adelante, no se han desarrollado procedimientos para la navegación hacia un punto objetivo basadas únicamente en sistemas reactivos. Para comprobar el comportamiento reactivo del sistema se utiliza el modo "Wander" del MobileEyes, que realiza una trayectoria aleatoria guiada por acciones reactivas. La ejecución en el ciclo síncrono del robot, estableciendo prioridades sobre las acciones da como resultado esta estructura de control. La interpretación de los sensores y las acciones se han agregado al ciclo a través de las denominadas "tareas de usuario", haciendo uso de los *functors*, incluidos en la clase definida para la interpretación sensorial *SensoryTask*.

A partir del *ModuloRoboticaV2*, esta clase se encarga también de ejecutar polimórficamente el algoritmo particular seleccionado al instanciarlo (a través del método *compute*, necesario para los algoritmos sensoriales, desde el método *doTask*):

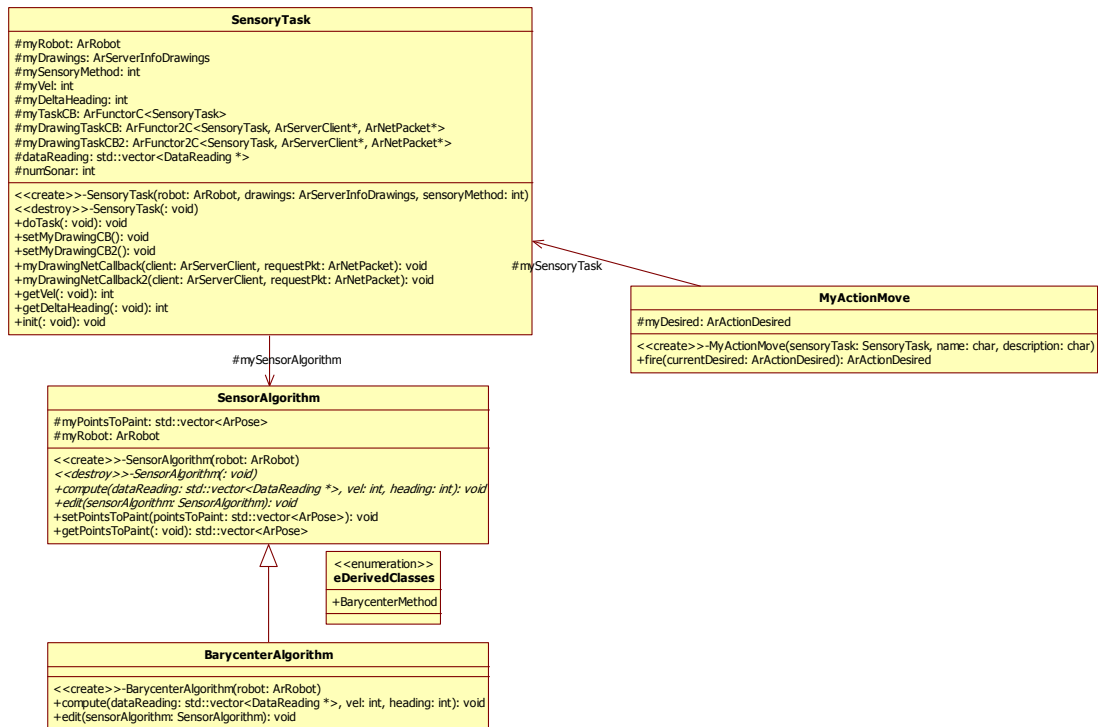
SensoryTask
<pre> #myRobot: ArRobot #myDrawings: ArServerInfoDrawings #mySensoryMethod: int #myShape: char #myLabelToPaint: char #myVel: int #myDeltaHeading: int #myTaskCB: ArFunctorC<SensoryTask> #myDrawingTaskCB: ArFunctor2C<SensoryTask, ArServerClient*, ArNetPacket*> #dataReading: std::vector<DataReading *> #numSonar: int #mySensorAlgorithm: SensorAlgorithm #timer: ArTime <<create>>-SensoryTask(robot: ArRobot, drawings: ArServerInfoDrawings, sensoryMethod: int, shape: char, labelToPaint: char) <<destroy>>-SensoryTask(: void) +doTask(: void): void +myDrawingNetCallback(client: ArServerClient, requestPkt: ArNetPacket): void +getVel(: void): int +getDeltaHeading(: void): int +init(: void): void </pre>

Las tareas incluidas en el ciclo síncrono del robot se encargan de ejecutar las funciones *doTask* y *myDrawingNetCallback*. La primera se encarga de capturar los datos de los sensores e instanciar el método de interpretación, tal y como se ha mencionado anteriormente, mientras que la segunda se ha llevado a cabo para poder dibujar un vector de puntos sobre el mapa del cliente MobileEyes. Tanto en *ModuloRoboticaV2* como en *ModuloRoboticaV3* se han incluido dos métodos para pintar sobre el mapa del MobileEyes, el ya mencionado (que se encarga de pintar el último baricentro) y otro nuevo denominado *myDrawingNetCallback2* (encargado de pintar el histórico de puntos).

Las acciones motoras se llevan a cabo con un objeto hijo de *ArAction*, llamado *MyActionMove*:

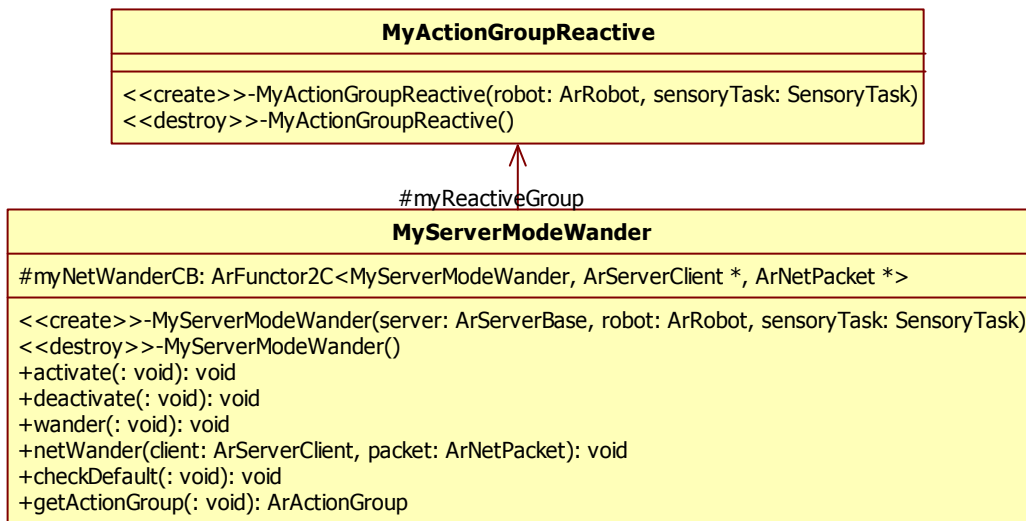
MyActionMove
<pre> #mySensoryTask: SensoryTask #myDesired: ArActionDesired <<create>>-MyActionMove(sensoryTask: SensoryTask, name: char, description: char) +fire(currentDesired: ArActionDesired): ArActionDesired </pre>

La estructura que las relaciona es la siguiente:

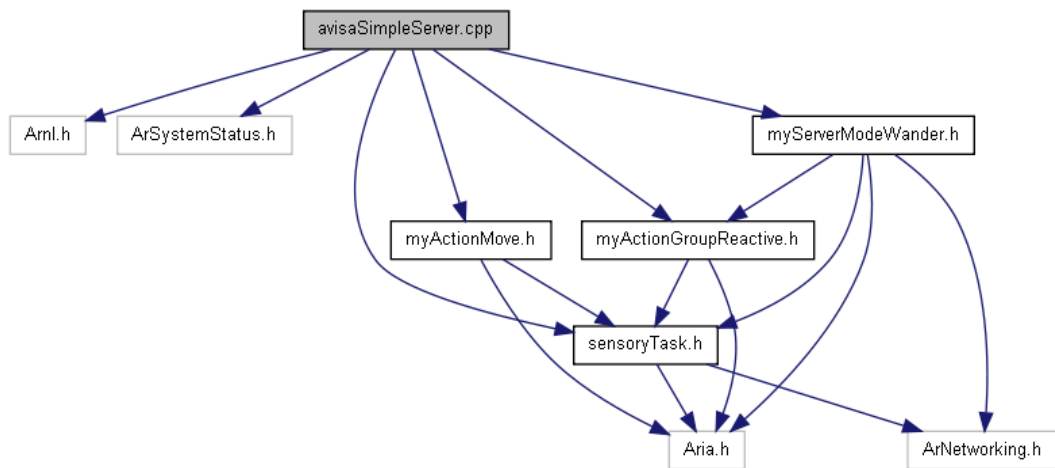


Esta acción motora es añadida a través de un objeto hijo de *ArActionGroup* (siendo el lugar apropiado para incluir acciones adicionales), denominado *MyActionGroupReactive*.

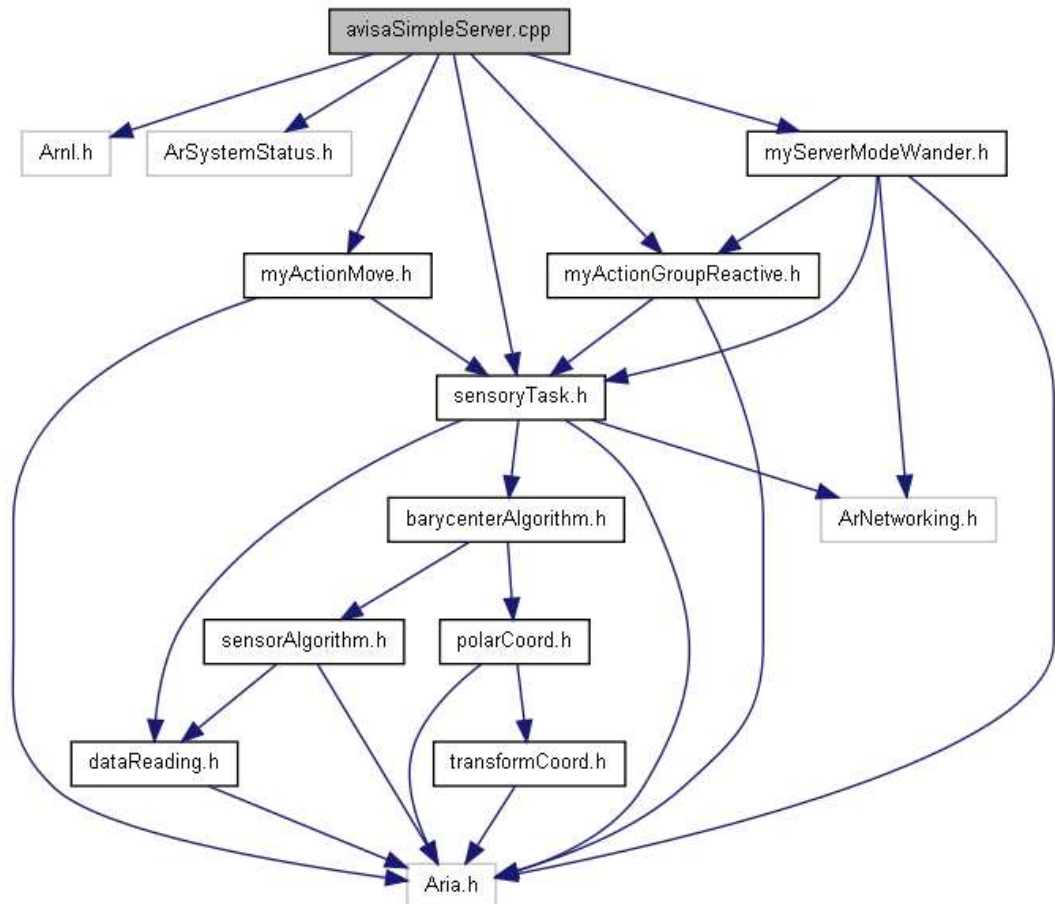
Para instanciarlo se hace a través de un objeto hijo de *ArServerModeWander*, denominado *MyServerModeWander*.



Las dependencias de ejecución son establecidas desde la función *main* en el fichero *avisaSimpleServer.cpp*, el esquema de cada módulo es el siguiente:



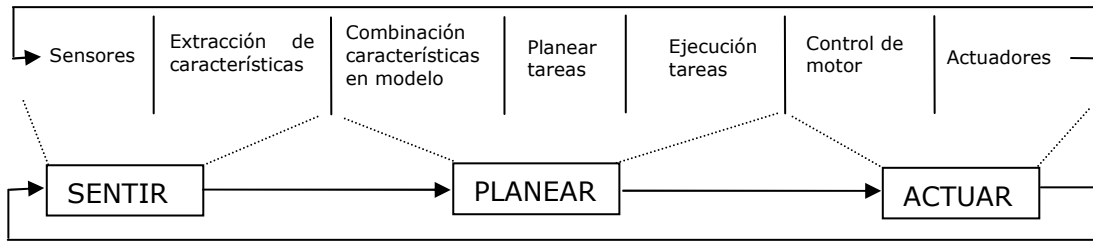
ModuloRoboticaV1



ModuloRoboticaV2

6.2 Aplicación a sistemas deliberativos e híbridos

En un sistema deliberativo - proceso horizontal - el control se realiza partiendo de un modelo global del mundo, comparando los cambios entre el mundo actual y el inicial, construyendo la planificación para llegar a un punto objetivo en base a los operadores definidos, disponiendo de acceso a los datos de los sensores tras cada operación únicamente.



Proceso horizontal (Brooks, [15])

Para incorporar esta componente deliberativa y construir un sistema híbrido, son necesarias técnicas de procedimiento asíncronas, desacoplando la primitiva PLANEAR de la parte reactiva evitando que le afecte el coste en tiempos y computacional que requieren las tareas deliberativas. Para crear estas tareas ARIA dispone de un objeto *ArASyncTask*, que puede ser derivado para sobrescribir su método *runThread*, creando un hilo de ejecución independiente al iniciarlo con el método *create*. La comunicación entre los threads del ciclo síncrono reactivo y los threads independientes deliberativos, se puede apoyar en los objetos *ArMutex* y *ArCondition*.

En el servidor desarrollado a medida incluido en *ModuloRoboticaV3*, desde la función principal *main*, se ha aprovechado una de las funcionalidades incluidas en la librería ARNL, denominada *ArPathPlanningTask*. Esta tarea se inicia en modo asíncrono para planificar la ruta, incluyendo también acciones para evitar obstáculos y recalcular la ruta. El algoritmo requiere exactitud en la localización, por lo que se inicia otra tarea asíncrona dedicada a la localización, denominada *ArLocalizationTask*. La referencia al objeto *ArPathPlanningTask* es mandada a la clase *GotoAlarm*, siendo desde ahí donde se le indicará la posición a alcanzar, mandada desde el cliente QT conectado al puerto 4242.

GotoAlarm
+myClientClosedCB: ArFunctorC<GotoAlarm> #sentByAlarm: bool -myGoalTask: ArPathPlanningTask -clientSock: ArSocket -serverSock: ArSocket -myRobot: ArRobot -myTaskCB: ArFunctorC<GotoAlarm>
<<create>>-GotoAlarm(goalTask: ArPathPlanningTask, robot: ArRobot) <<destroy>>-GotoAlarm() +connectToClient(: void): bool +doTask(: void): void +doClientClosed(: void): void

En el *ModuloRoboticaV4*, antes de enviar la referencia de "pathPlaning" a *GotoAlarm*, se enviado al modo "Goto" que permite el uso de "Goals" desde el *MobileEyes*, a través del objeto *ArServerModeGoto*.

Estos modos de funcionamiento se corresponden con una arquitectura híbrida, regida por las deliberaciones dedicadas a la planificación y las acciones reactivas utilizadas para evitar obstáculos o el propio algoritmo del baricentro, incluido a través del constructor de la clase *MyActionGroupReactive*.

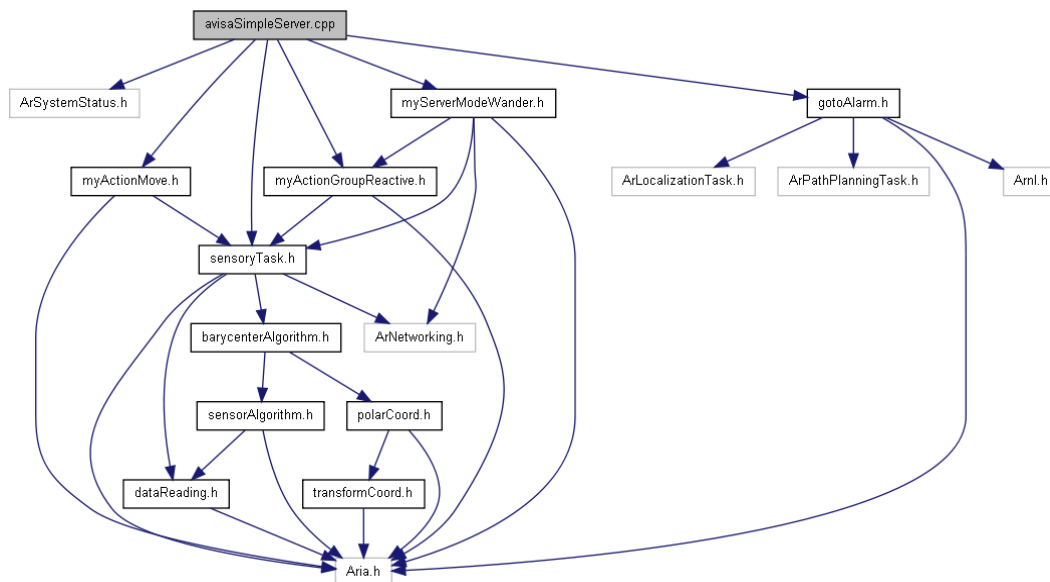
Su funcionamiento ante obstáculos imprevistos se puede comprobar ejecutando una simulación del robot con un mapa que presente obstáculos (*mobilesim_map2.bat*, *mobilesim_map4.bat*, *mobilesim_map5.bat*, *mobilesim_map6.bat*) asignándole al servidor uno similar pero vacío (*mobilesim_map1.bat*, para los ejemplos indicados), siendo este último el mapa

utilizado también por el cliente MobileEyes. El posicionamiento objetivo se puede mandar de tres formas:

- 5) Desde el cliente QT.
- 6) A través de puntos objetivo incluidos en los mapas, "Goals", tal y como se ha hecho en el mapa map1.map. Éstos pueden ser utilizados desde MobileEyes.
- 7) Seleccionarlos en tiempo de ejecución a través del mapa presentado en el MobileEyes.

En caso de no enviar las coordenadas desde el cliente QT (únicamente conectándolo al servidor para que continúe la ejecución del servidor) y utilizar el modo "Wander" desde el cliente MobileEyes, el movimiento del robot se corresponderá con una arquitectura puramente reactiva, utilizando como acción para el movimiento el método del baricentro. En esta arquitectura también es aplicable el método seleccionado para el uso de los radares, en el cliente QT, pero únicamente después de haber enviado al menos una planificación desde el mismo, ya que es conjuntamente con las coordenadas justo el momento en el que el servidor recibe este parámetro. Por defecto se utilizarán todos los radares.

Las dependencias desde la función *main* de estos módulos, tiene la siguiente forma:



ModuloRoboticaV3 y ModuloRoboticaV4

6.3 Aplicación a sistemas de visión

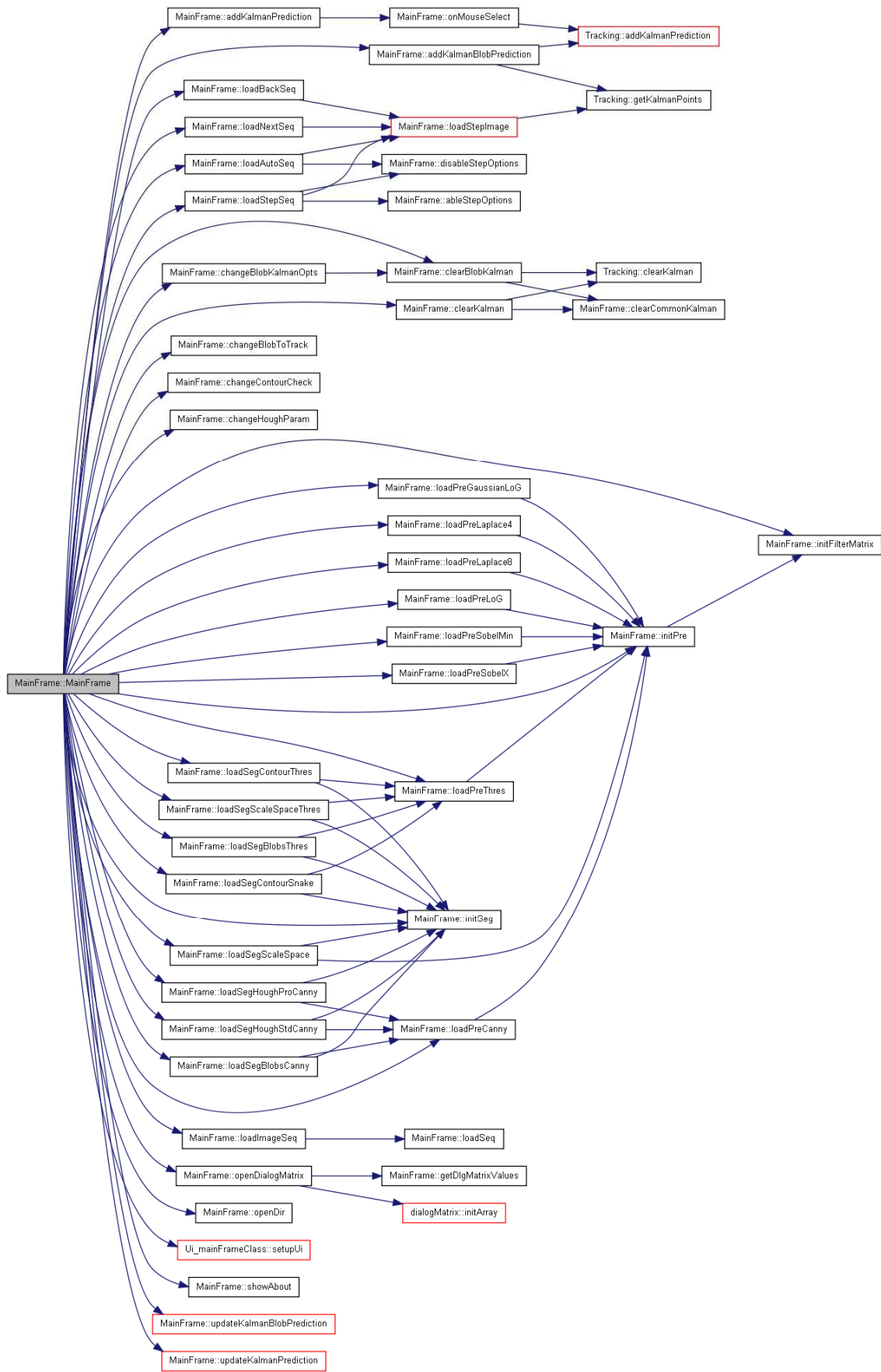
Para conseguir una cierta modularidad se han considerado las siguientes etapas del proceso de visión (en [10] se puede encontrar un análisis de mayor profundidad en el ámbito de desarrollo de un sistema de video-vigilancia):

- Visión de bajo nivel
 - Preproceso: Eliminación de ruidos, realce de características de interés, etc.
 - Segmentación: División en regiones, sin usar conocimiento
- Visión de nivel medio:
 - Segmentación con conocimiento de las zonas de interés
 - Descripción de las características: Representación (encapsula la estructura de datos de los objetos), extracción de características (que los distinguen del resto de objetos)
- Visión de alto nivel
 - Reconocimiento: Se asigna a cada vector de características (objeto) un significado del mundo real

- Interpretación: Descripción simbólica de la escena (esta es la etapa ligada con la IA)

El software desarrollado propone una estructura que pretende afrontar dos dificultades fundamentales:

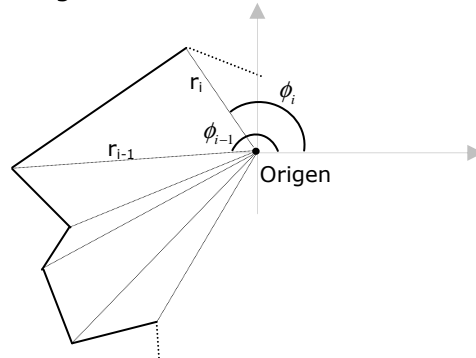
- 1) A pesar de que las tareas de análisis de imágenes tienen una estrecha relación con las particularidades de los objetivos, siendo dependientes del problema a resolver, las etapas mencionadas podrían generalizarse para el seguimiento de cualquier tipo de objeto.
- 2) Modelando el tratamiento en base a etapas, que en ningún caso tienen que ser totalmente independientes, se puede crear un software escalable, en el que ir incluyendo elementos de las diferentes etapas de forma progresiva, así como elementos para la interrelación configurable de las mismas (como sería, por ejemplo, el hecho de poder seleccionar el orden en el que se llevan a cabo los filtros)



7 MÉTODO DEL CENTRO DE ÁREAS Y CONDUCCIÓN AUTÓNOMA

7.1 Desarrollo del método del centro de áreas o baricentro

Puesto que la lectura de los sónares ofrece la posibilidad de construir una figura geométrica poligonal, se puede descomponer en triángulos sobre los que se realizará el cálculo del baricentro individual, datos que nos permitirán calcular la posición del baricentro total de la figura. De hecho, dado el polígono de $i=1, \dots, l$ lados, se puede dividir subjetivamente (desde la posición central del robot) en l triángulos desde un punto que esté en el interior de la forma. El robot aquí se está aproximando a un punto origen:



Las coordenadas de su baricentro se pueden calcular a partir de los baricentros de cada triángulo, descrito por las ecuaciones:

$$x_i = \frac{1}{3}(r_{i-1} \cos \phi_{i-1} + r_i \cos \phi_i)$$

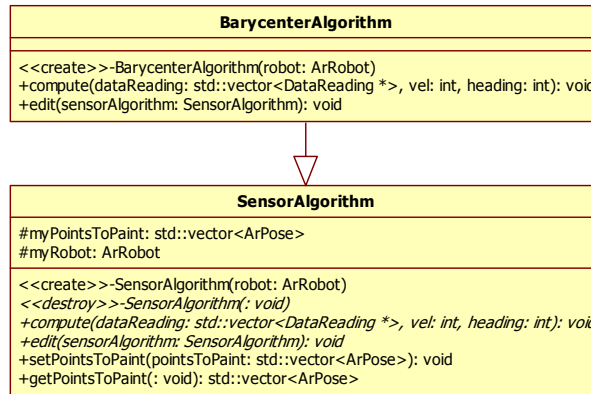
$$y_i = \frac{1}{3}(r_{i-1} \sin \phi_{i-1} + r_i \sin \phi_i)$$

De la misma forma, el área, con el que se pondera el valor del baricentro, vendría dado por la ecuación: $s_i = \frac{1}{2} r_i r_{i-1} \sin(\phi_{i-1} - \phi_i)$

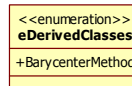
Las coordenadas del centro de áreas percibidas vienen dadas por los sumatorios ponderados, normalizados con el área total:

$$X_{CA} = \frac{\sum_{i=1}^l x_i s_i}{\sum_{i=1}^l s_i}, \quad Y_{CA} = \frac{\sum_{i=1}^l y_i s_i}{\sum_{i=1}^l s_i}$$

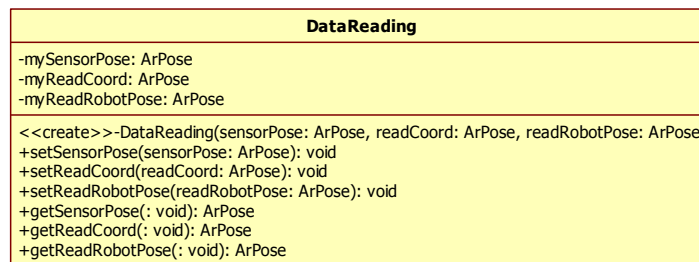
Estos cálculos están incluidos a partir del segundo de los módulos desarrollados (*ModuloRoboticaV1*). A partir del tercer módulo desarrollado (*ModuloRoboticaV2*), la estructura del algoritmo que realiza los cálculos tiene la siguiente estructura: Está desarrollada a partir de un objeto abstracto, permitiendo así la reutilización de la aplicación para incluir otros métodos con los que llevar a cabo la interpretación de la lectura de los sensores. La función *compute* es la que asume este papel, necesitando para ello los datos de lectura procedentes de los sensores, incluyendo por referencia la velocidad y el movimiento de rotación que actuarán sobre los motores del robot:



En el mismo fichero que el objeto *SensorAlgorithm* se incluye una enumeración que requiere ser actualizada si se llevan a cabo nuevos métodos, de momento únicamente el método asociado con la clase derivada *BarycenterAlgorithm* (denominándolo *BarycenterMethod*):



En lugar de utilizar directamente la lectura de los sensores, se ha creado un objeto denominado *DataReading* en el que se almacena la posición y lectura de un sensor, medidos desde el sistema de referencia robot, así como la posición del robot en el momento de la lectura, medida en el sistema de referencia global:

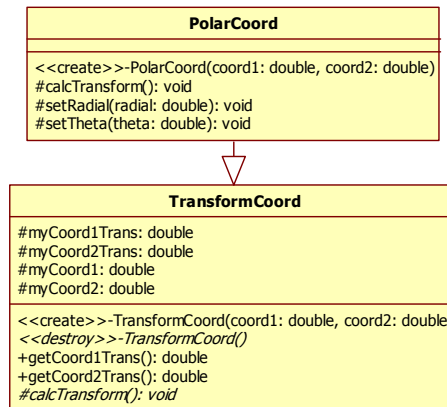


Al llevar a cabo un nuevo método de interpretación de sensores, podría resultar de utilidad alguna otra magnitud sensorial, que habrá que añadir a la estructura *DataReading*, no pareciendo necesario extender la jerarquía de la estructura. Al representar los datos de un único sensor, será necesario almacenar los datos en un vector de objetos *DataReading*, en el que se guarda la lectura de todos los sensores en cada ciclo.

La librería ARIA permite obtener estos datos con facilidad, a través del objeto *ArRobot*, con el que se pueden conseguir el número de sensores, función *getNumSonar*, y un objeto *ArSensorReading* por cada sensor, función *getSonarReading*, que da acceso a la posición obtenida por el sensor, función *getPose* (cuya posición desde el robot se obtiene aplicándole una transformación con el objeto *ArRobot*, la función *getToLocalTransform*), así como a la posición del robot en el momento de realizar la lectura, a través de la función *getPoseTaken*.

Una vez se dispone de un vector con los datos de lectura desde los sensores, en un sistema de referencia situado en el robot y en coordenadas cartesianas, es necesaria una transformación a coordenadas polares para poder aplicar las ecuaciones de posición del centro de áreas. Para ello se ha creado una nueva jerarquía que facilite el desarrollo de nuevas transformaciones sin necesidad de

cambios en lo que ya se ha desarrollado, aunque esto no incluye transformaciones en 3 dimensiones:



En concreto, dadas las coordenadas del punto desde el origen del sistema robot (x_r, y_r) las coordenadas polares se obtienen con las siguientes ecuaciones:

$$r_r = \sqrt{x_r^2 + y_r^2}, \theta_r = \arctg\left(\frac{y_r}{x_r}\right)$$

7.2 Movimiento hacia el centro de áreas

En el tercer módulo desarrollado (*ModuloRoboticaV2*), la clase derivada *BarycenterAlgorithm* se encarga, en el mismo método en el que se ha desarrollado el mecanismo para la interpretación de las lecturas (*compute*), de modificar por referencia la velocidad y el giro del robot, que acto seguido serán aplicadas a través del objeto *MyActionMove*.

Una vez se dispone de la posición del baricentro en coordenadas cartesianas desde el sistema robot, haciendo uso de la clase *PolarCoord* es realizada una transformación a coordenadas polares (denominando *dist* a la distancia o coordenada radial y *barycenterTh* a la coordenada angular), aplicándose una velocidad (variable *vel*) y un movimiento angular (variable *heading*) distintos de 0, únicamente si la distancia es mayor que una determinada cantidad:

```

if (dist > stopDistance){
    int turn = ArMath::roundInt(barycenterTh * turnRate);
    int velTemp = dist * distRate;
    vel = (velTemp > maxVel) ? maxVel : velTemp;
    if (ArMath::fabs(barycenterTh) > minAngle)
        heading = turn;
}else{
    vel = 0;
    heading = 0;
}
  
```

Donde se ha parametrizado como *stopDistance* la distancia mínima considerada para lograr el objetivo, *maxVel* la velocidad máxima del robot aplicable cuando el cálculo de la velocidad lo supere, *minAngle* la distancia angular mínima (de la orientación del robot respecto a la posición del baricentro) para aplicar un movimiento angular, *turnRate* para establecer la proporción de la diferencia angular que realizará el movimiento (permitiendo así movimientos más pequeños cuanto más cercana a cero) y *distRate* para realizar un cálculo de velocidad proporcional a la distancia al objeto (consiguiendo velocidades tanto más pequeñas cuanto más cerca se encuentre el punto).

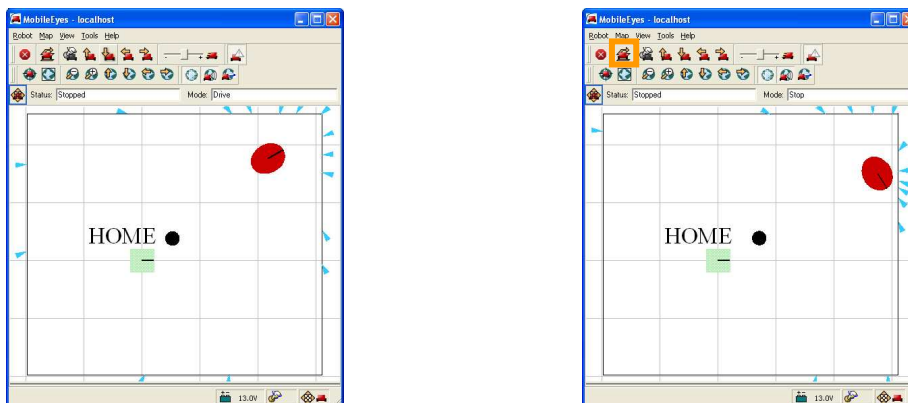
Tras realizar algunas pruebas de comportamiento con diferentes valores en la parametrización, se han seleccionado los siguientes valores:

```
stopDistance = 100
maxVel = 400
minAngle = 30
turnRate = 0.3
distRate = 0.3
```

El comportamiento obtenido es parcial debido a la estabilidad del centro de áreas (que se puede visualizar como un mínimo local, i.e. [12]), necesitando forzosamente de otros puntos de referencia (invariantes) o cualquier otro método para generar un momento sobre el robot, evitando la estabilidad al alcanzar los baricentros de cada zona. De una forma muy simplificada esto es lo que se ha pretendido con el método de medida frontal incluido en el *ModuloRoboticaV4*.

No obstante, puesto que la aplicación desarrollada incluye la posibilidad de utilizar el control remoto del programa MobileEyes, las pruebas se apoyarán en el posicionamiento teleoperado del robot.

Por otro lado, cabe decir que el comportamiento de este algoritmo de movimiento no incluye ningún control para evitar obstáculos (pudiendo incluirse como acción en el constructor del objeto *MyActionGroupReactive*), por lo que serían críticas (en cuanto al peligro de impacto) las situaciones en las que la orientación del robot es opuesta a la de la posición del centro de áreas, que se encuentra a distancia suficiente para permitir velocidades altas, y se encuentra en frente de una pared, ya que el giro se realiza con una curvatura grande que causa su impacto:



Sin embargo, disponiendo de otros mecanismos para controlar la colisión, no parece de interés para el método el incluir un control del tipo: Si el ángulo es mayor que una cierta cantidad, que minimice la velocidad y agudice el giro. Ya que esto evitaría también interesantes comportamientos (curvaturas amplias) detectados en diferentes configuraciones, tal y como al traspasar puertas situadas en las esquinas de habitaciones.

7.3 Conducción autónoma apoyada en el sistema de visión

Ya se ha mencionado la posibilidad de incluir un nuevo esquema sensor-motor sobre los módulos desarrollados basado en un sistema de visión, utilizando las ideas expuestas en [8]. Imagínese, por ejemplo, la situación en la que se localizan los márgenes de la vía completa y se encuentra tráfico en sentido contrario:

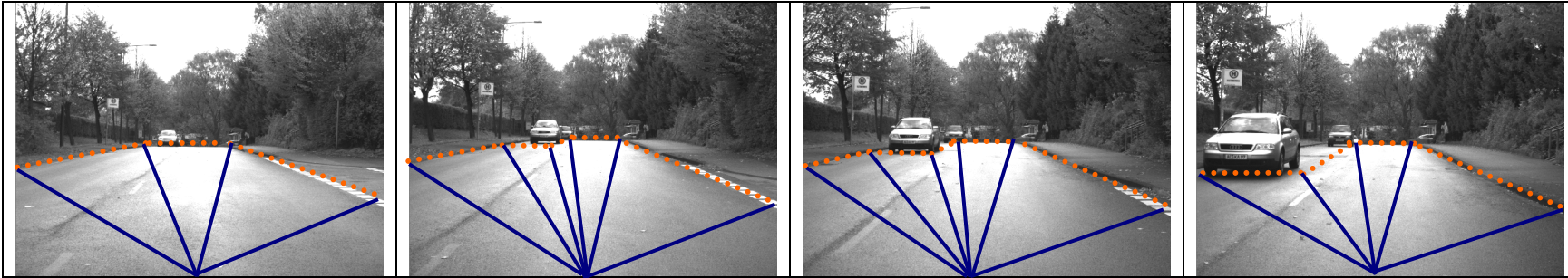


Tabla 7.3.1

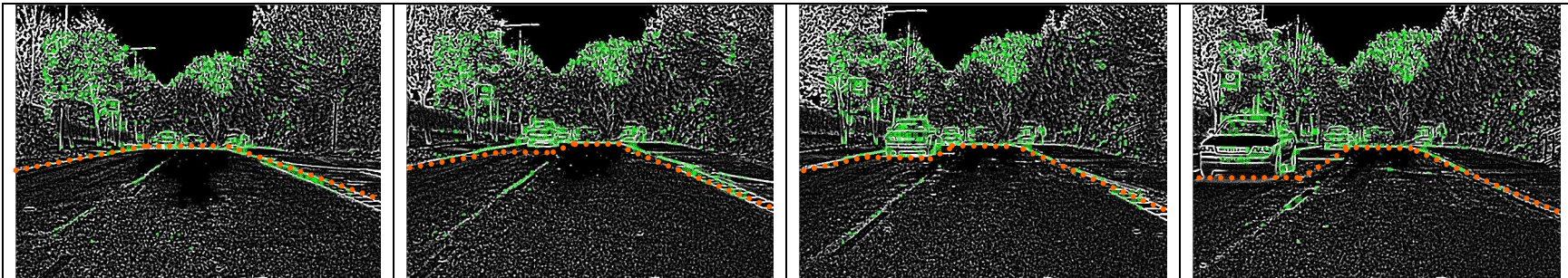


Tabla 7.3.2

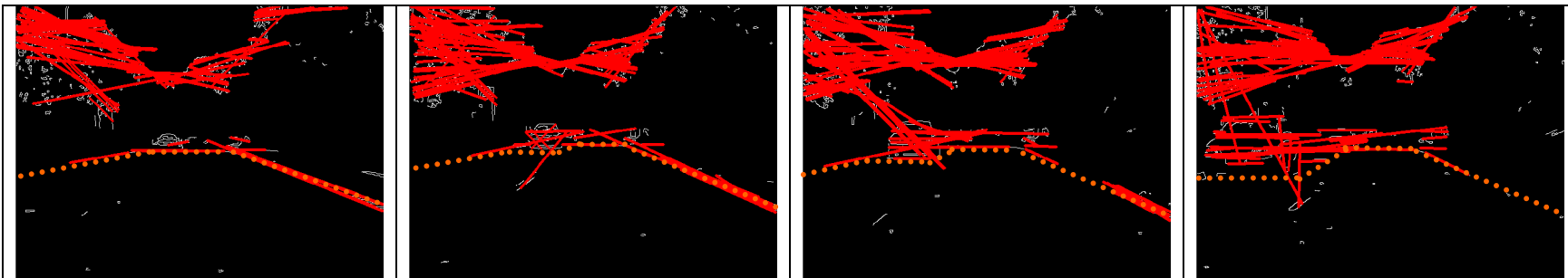
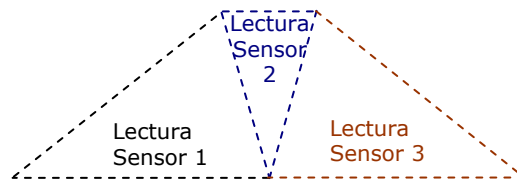
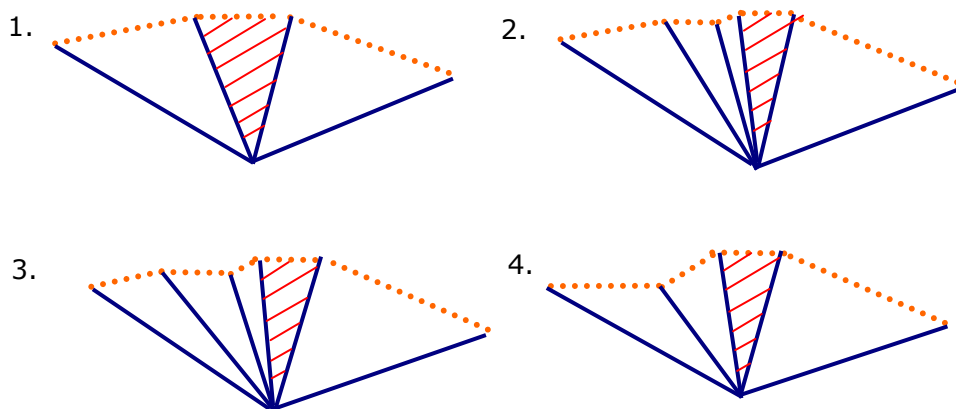


Tabla 7.3.3

Si se opta por trabajar con la figura geométrica trapezoidal, que simula la lectura de los sensores de rango:



A medida que aparece el vehículo en sentido opuesto, sería necesario rectificar de algún modo la forma trapezoidal, esto se muestra en la Tabla 7.3.1 con contornos naranja punteados. Si se aísla la forma y se simula la lectura de los sensores de rango con línea azul continua se obtiene la siguiente evolución:



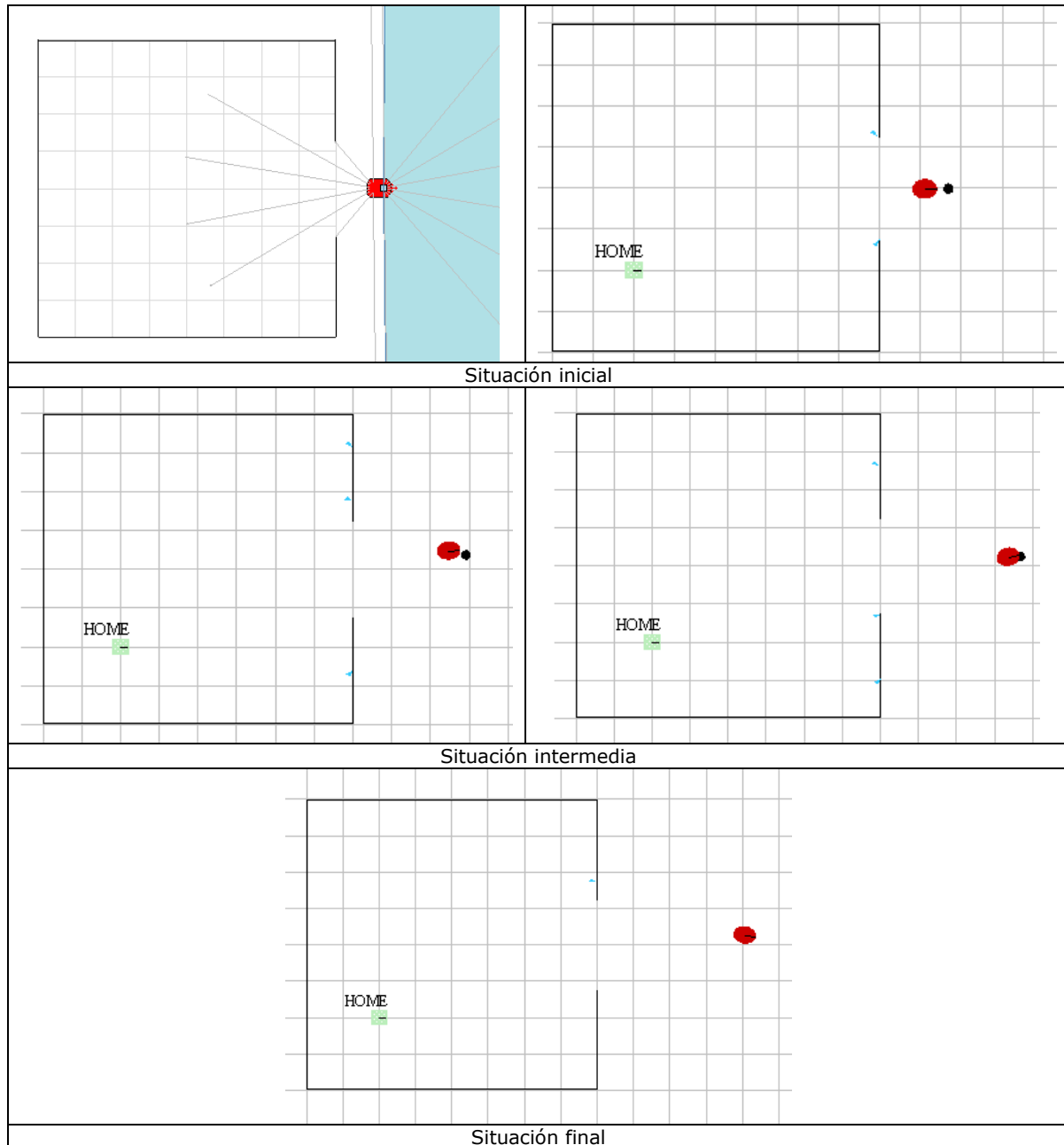
Estas formas reflejan la gran similitud que se puede llegar a obtener a través del análisis visual (en las Tablas 7.3.2 y 7.3.3. se ha superpuesto el resultado de la forma a la obtenida a través de dos ejemplos concretos de preprocesado y segmentación) en relación a los métodos utilizados a través de los sensores de rango. Se han rayado en rojo las zonas seguras para el movimiento, aunque en la elección cabe la posibilidad de utilizar cualquiera de ellas (bajo la hipótesis de una detección correcta, todas las zonas serían seguras, sin embargo si, por ejemplo, se sabe que la conducción se realiza por el carril derecho, la zona derecha será más segura que las situadas a la izquierda).

8 PRUEBAS Y ANÁLISIS DE LOS RESULTADOS

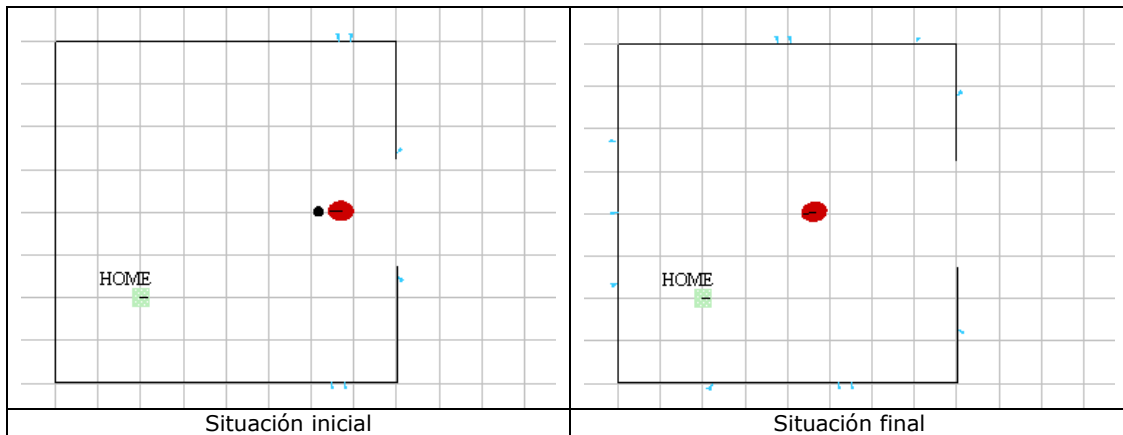
8.1 Traspaso de puertas centradas

8.1.1 Método general del centro de áreas

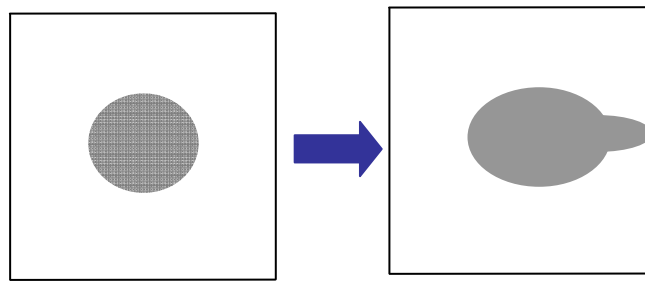
Al ser aplastada la forma geométrica construida a partir los sensores traseros, el baricentro se puede utilizar como punto de referencia para salir, en una dirección que dependerá de la simetría de la superficie externa de la pared de la habitación y de la orientación del robot:



La situación inversa es equivalente, ya que el origen del desplazamiento del centro de áreas es el mismo, siempre y cuando la habitación sea suficientemente amplia.

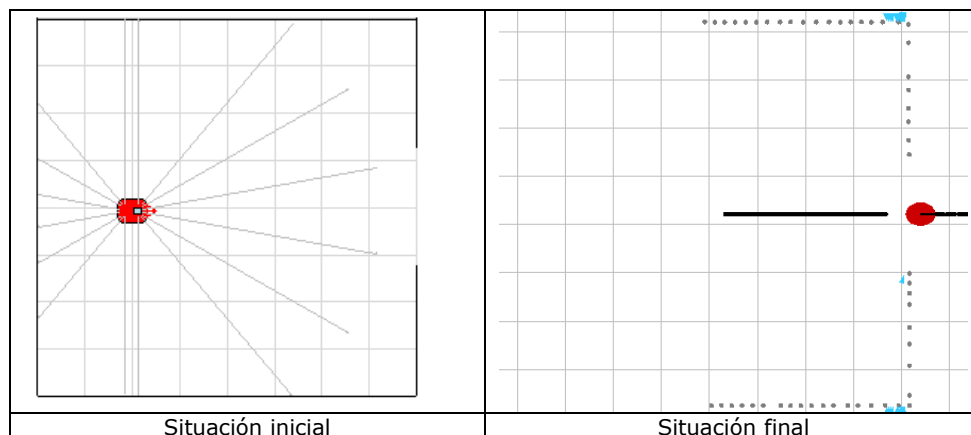


Desde el punto de vista topológico también es interesante la forma geométrica de los puntos que constituyen el baricentro, ya mostrados, que adoptan un estiramiento hacia la cavidad, con respecto a la forma cerrada, cuando realizamos un paseo por todas las regiones de la habitación original:



8.1.2 Método parcial del centro de áreas

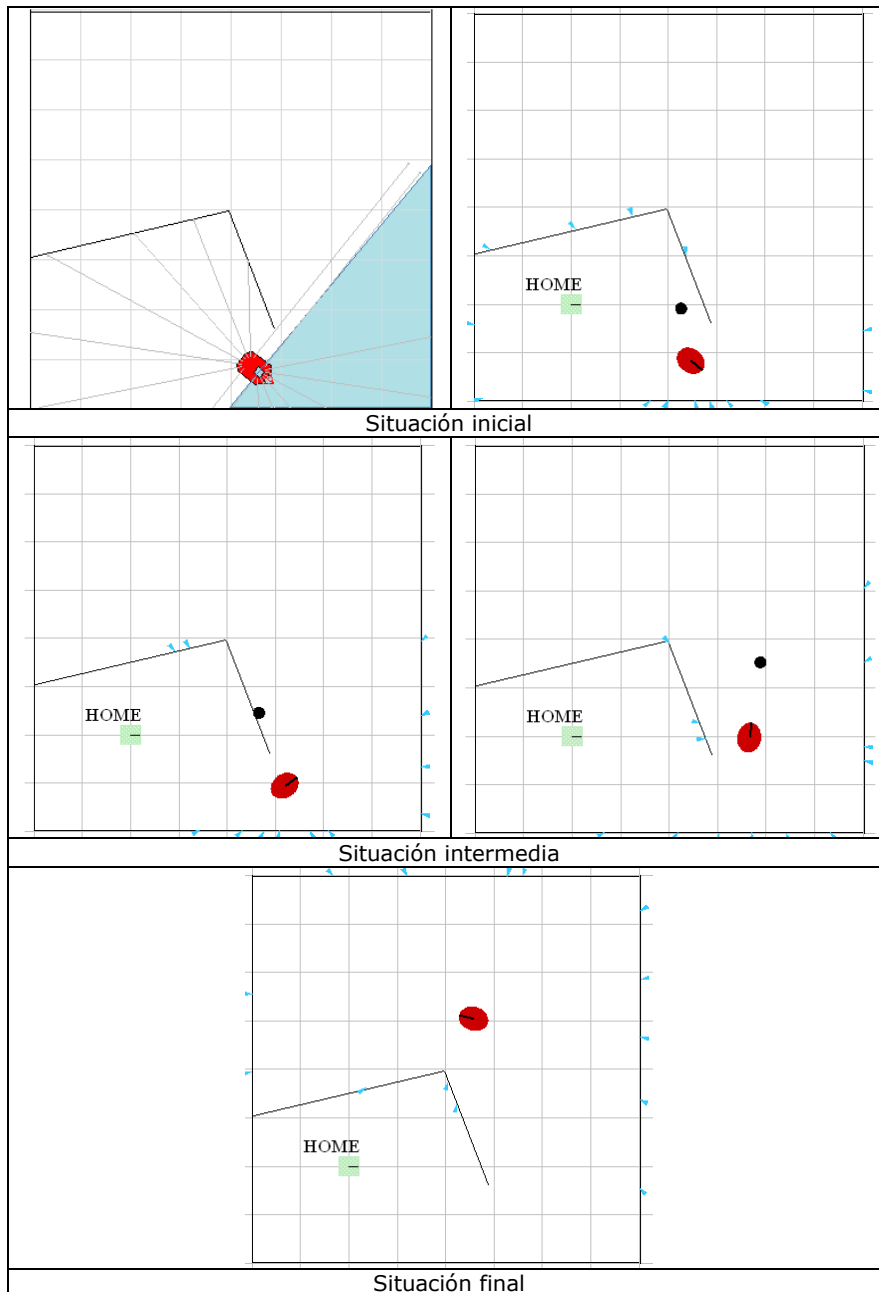
Con el método de radares frontales el comportamiento es similar, permitiendo situar el robot, con el mismo alineamiento hacia la puerta, a una distancia mayor, asegurando igualmente la salida:



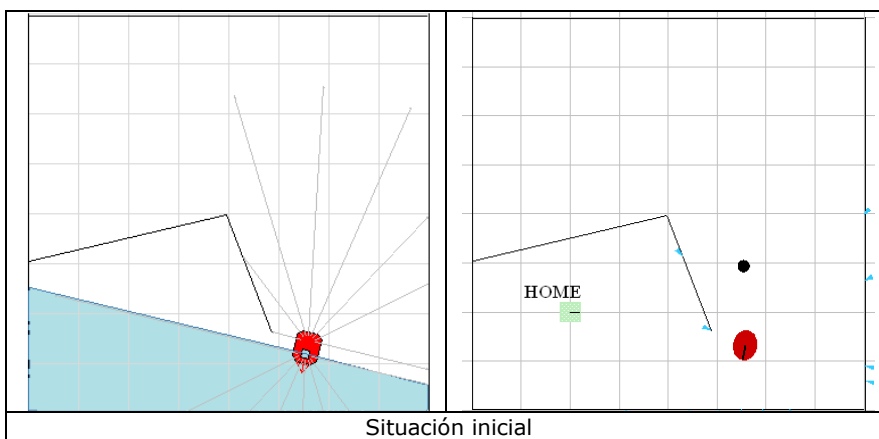
8.2 Traspaso de puertas no centradas

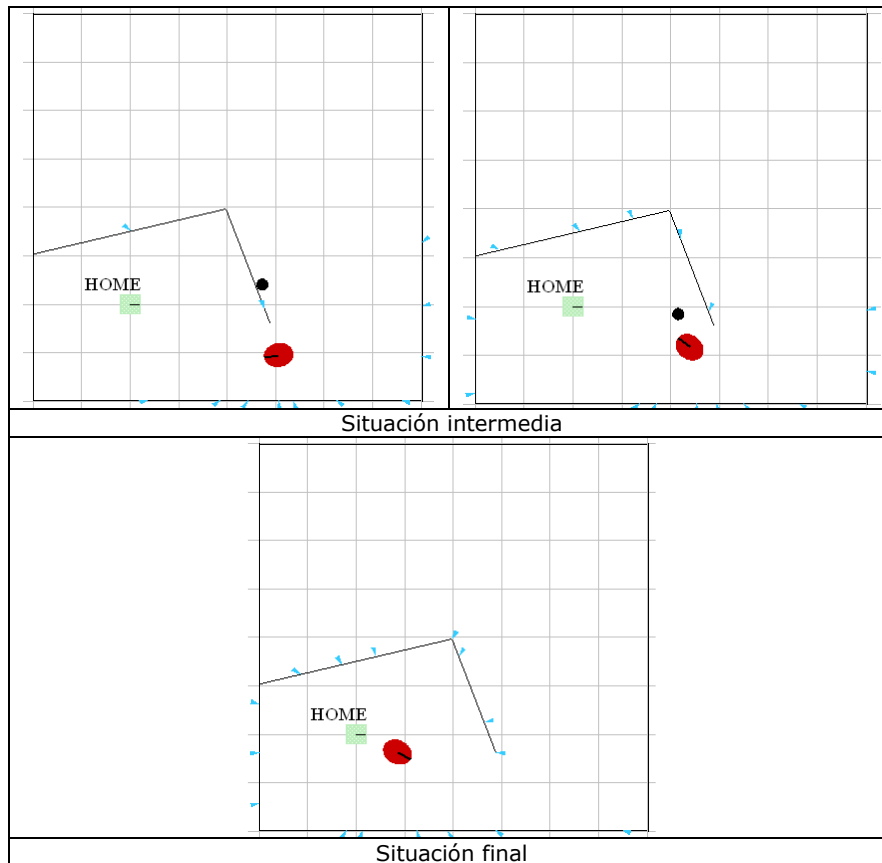
8.2.1 Método general del centro de áreas

Cuando la puerta se sitúa cercana a una esquina de la habitación (cuya forma oprime la forma geométrica construida desde los datos de los sensores), mientras que el robot se encuentra situado hacia la esquina, aún en la habitación, en el primer giro el centro de áreas se desplaza hacia el exterior, haciendo salir al robot:



La situación inversa produce un movimiento equivalente hacia el interior:

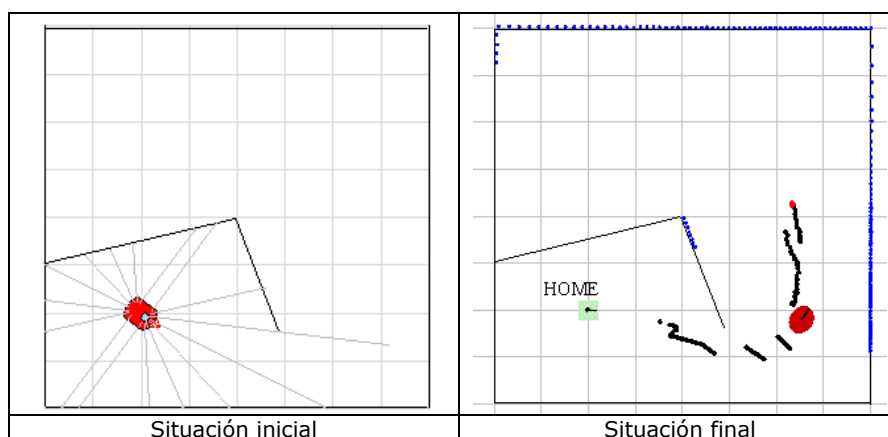




Debido a la vibración del punto en este caso, el algoritmo incurre en un camino sin fin si la distancia mínima no tiene un valor suficientemente alto (hay que tener en cuenta que no se ha restado el radio del robot).

8.2.2 Método parcial del centro de áreas

Con el método de radares frontales se consigue el mismo efecto, es decir, manteniendo el alineamiento hacia la puerta, permite asegurar la salida del robot emplazándolo a mayor distancia:

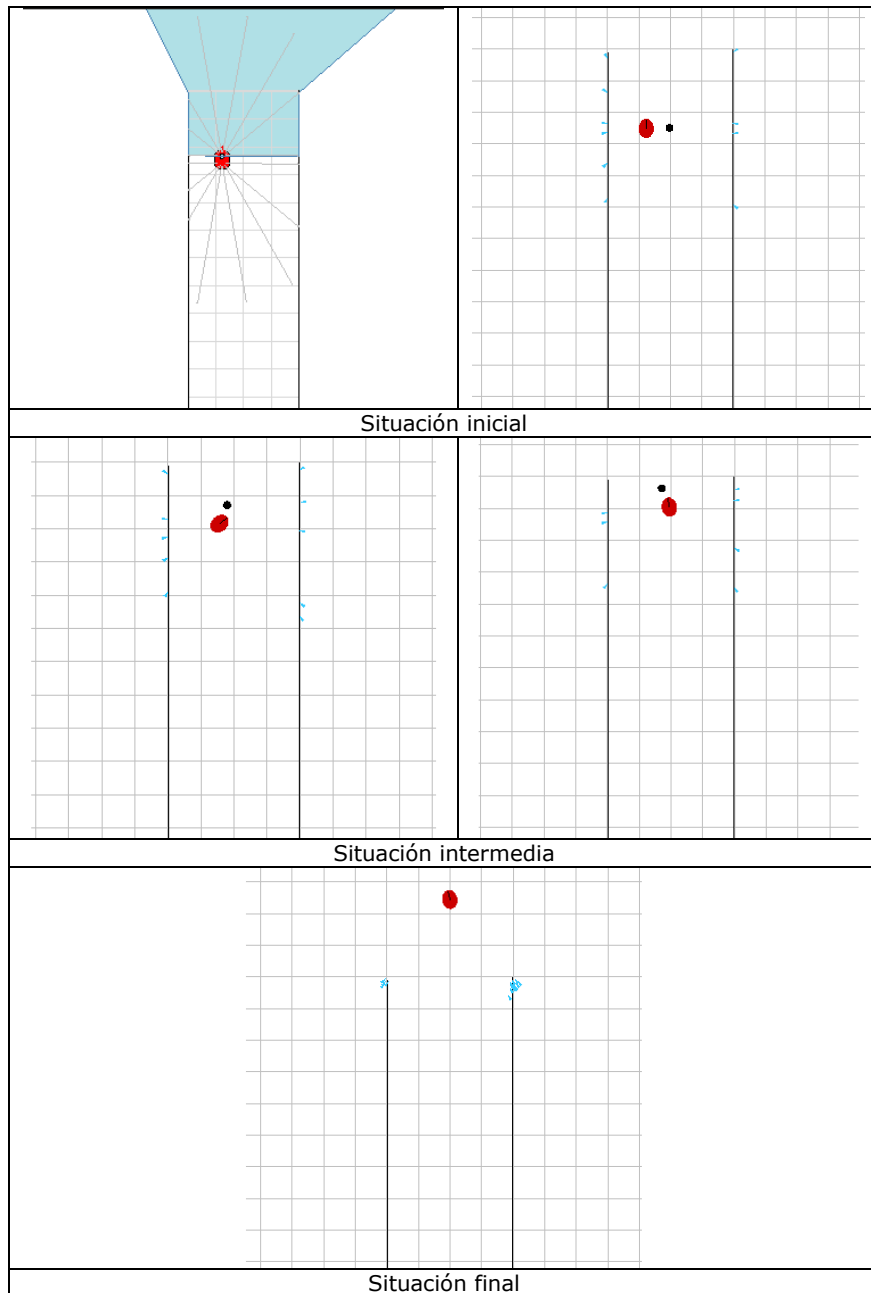


8.3 Salida de pasillos

8.3.1 Método general del centro de áreas

Cuando la situación del robot es cercana a la salida de un pasillo, hacia un espacio de mayor amplitud, no encontrándose sobre el punto de equilibrio, tenderá a salir

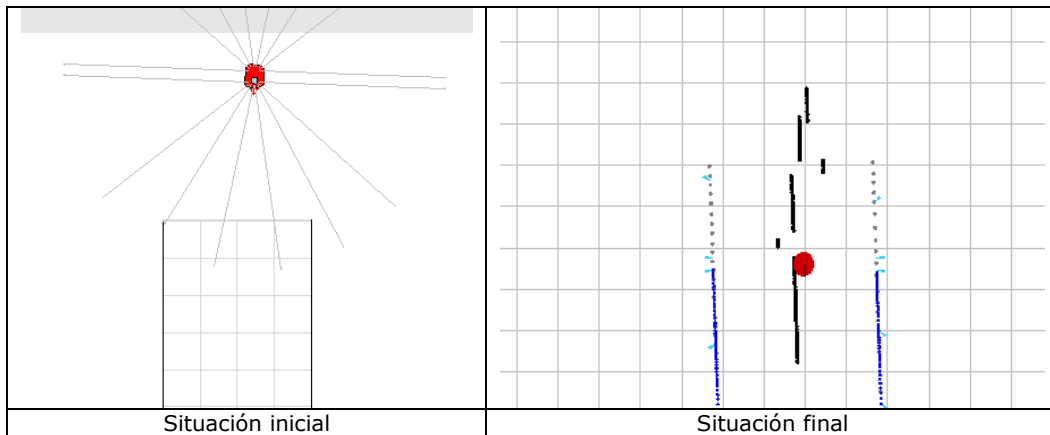
independientemente de la orientación que posea (siempre y cuando se encuentre suficientemente cerca de la salida):



A diferencia de los anteriores escenarios, esta situación no posee simetría, el centro de áreas se sitúa en el espacio libre, ya que la lectura procedente del pasillo comprime la forma geométrica.

8.3.2 Método parcial del centro de áreas

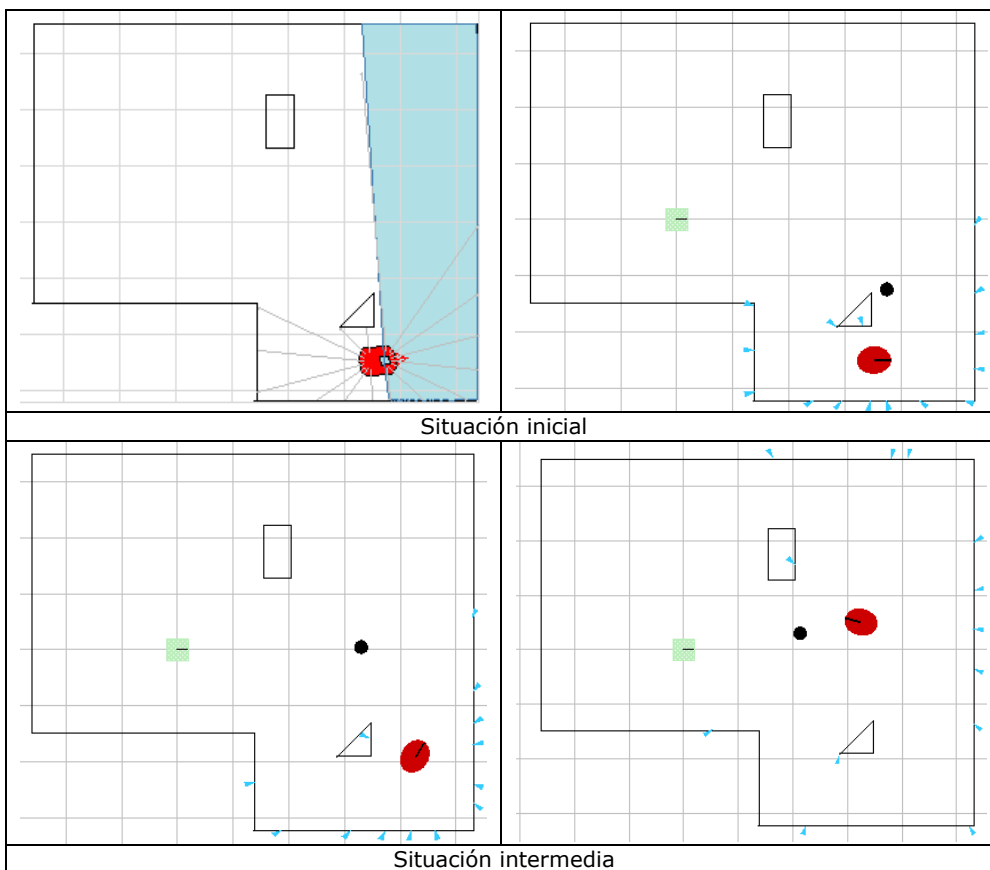
Esta problemática no se da con el método frontal de los radares, ya que la parte que comprime el pasillo orienta la posición del punto hacia la zona central:

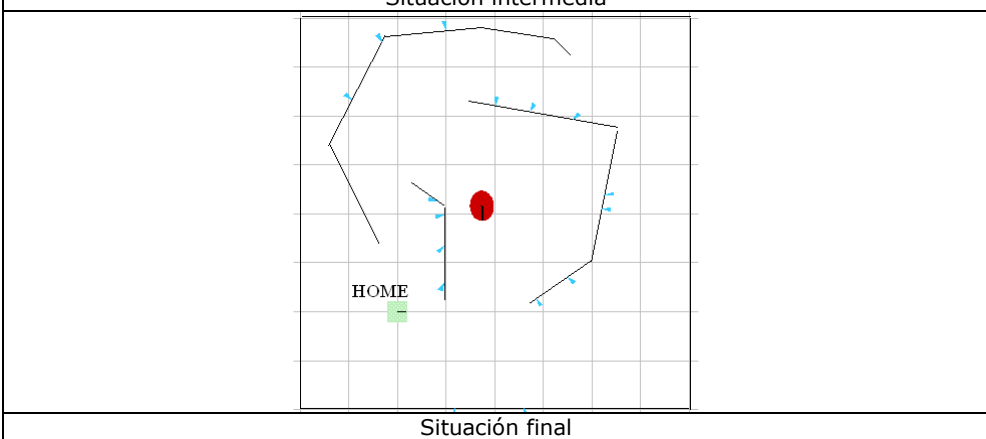
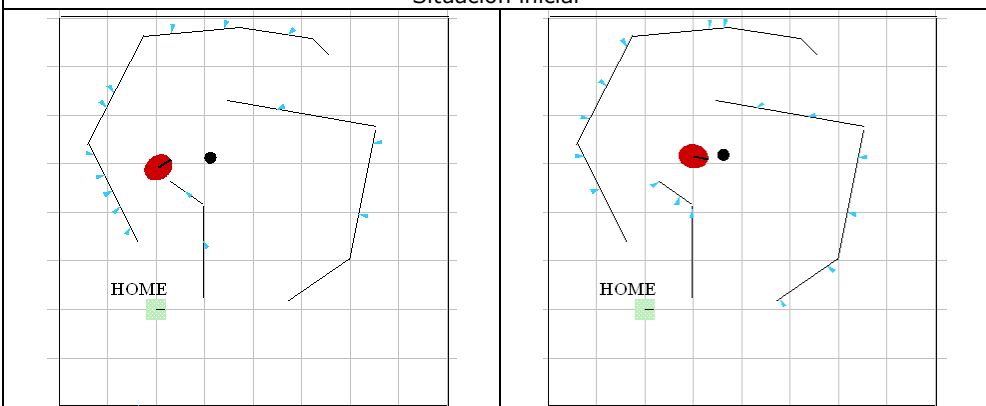
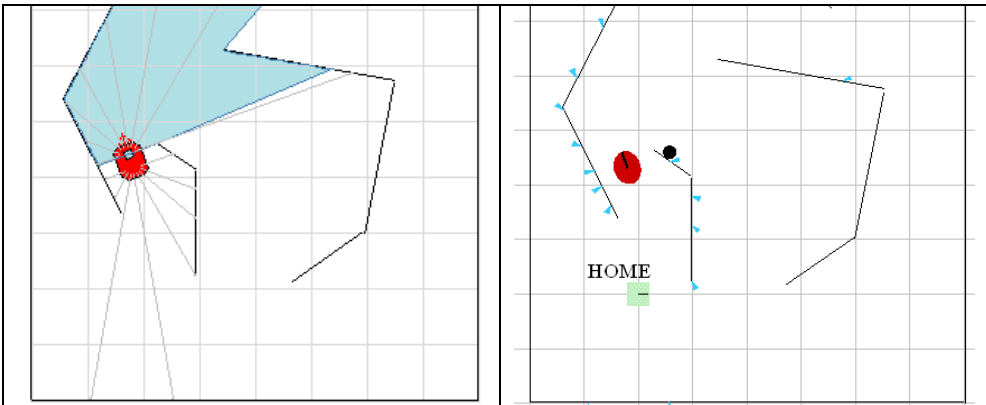
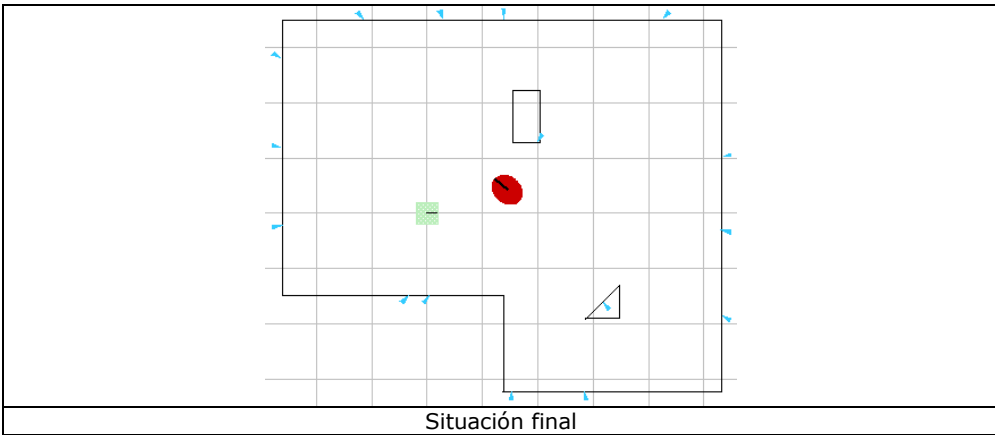


8.4 Trazabilidad hacia zonas de espacio libre

8.4.1 Método general del centro de áreas

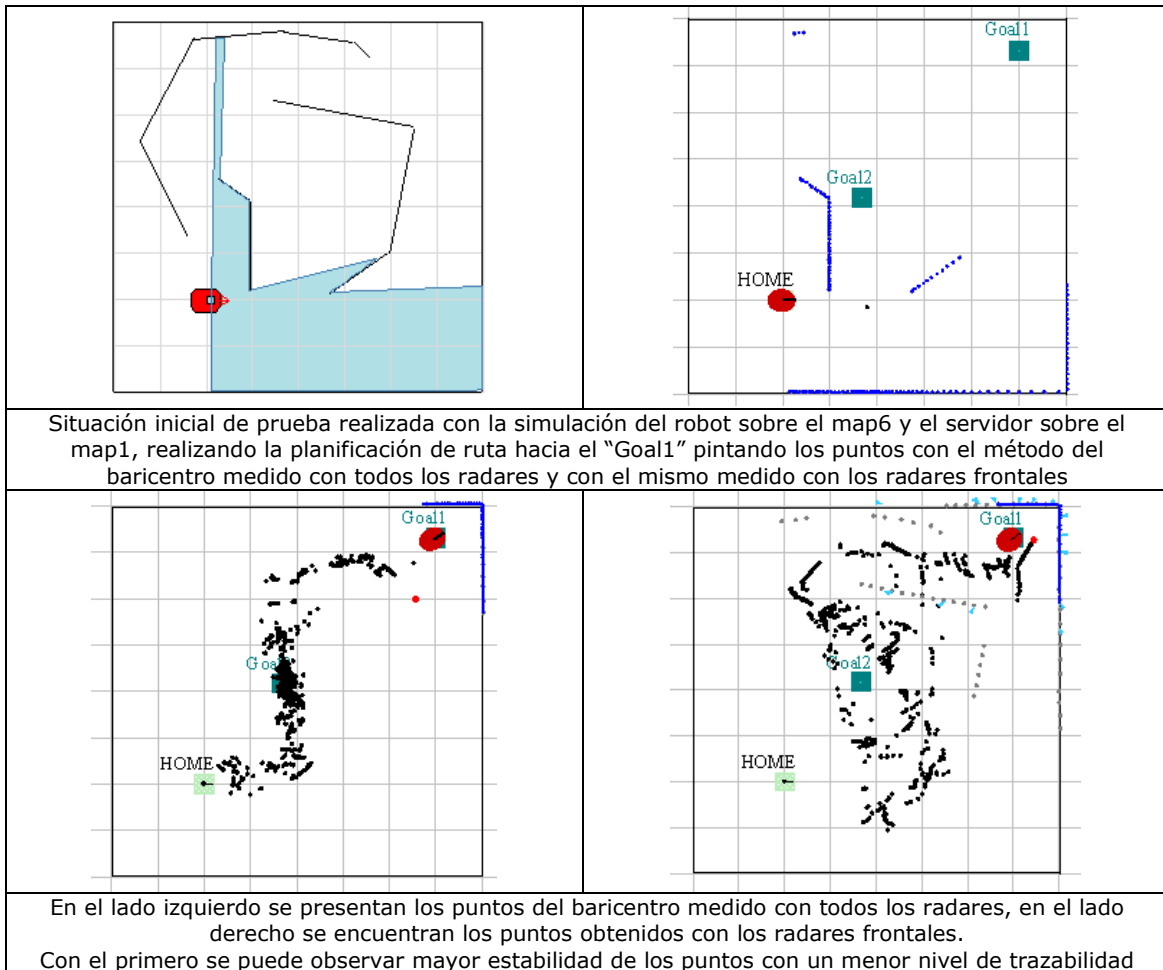
En la siguientes dos situaciones se pretende ilustrar la utilidad de este método para trazar rutas libres de obstáculos, complementándolo con acciones para evitar obstáculos:





8.4.2 Método parcial del centro de áreas

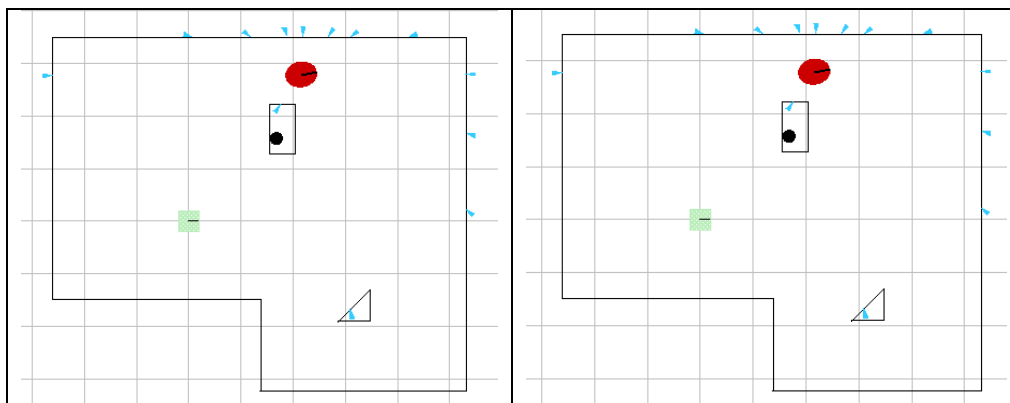
A continuación se muestra una prueba comparativa entre los métodos general y parcial del centro de áreas:

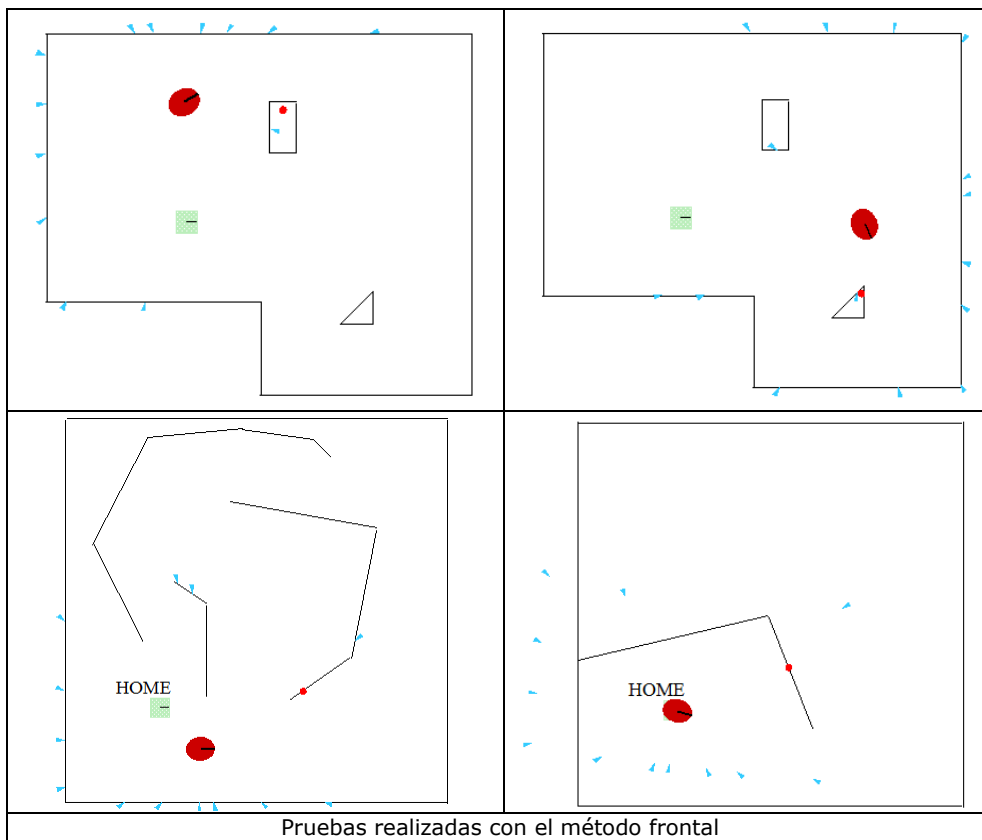
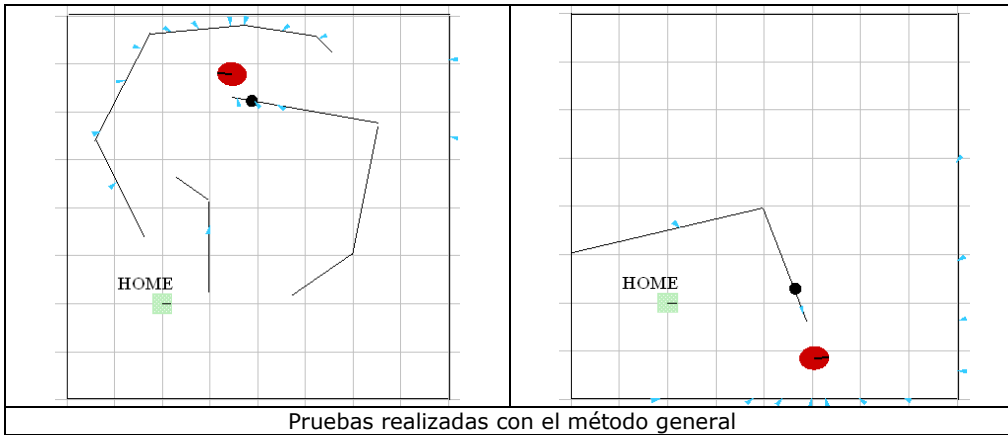


Esta última prueba también se puede realizar enviando el posicionamiento objetivo desde el cliente QT, en concreto hacia el punto 5000 - 5000 seleccionando el método radar para pintar.

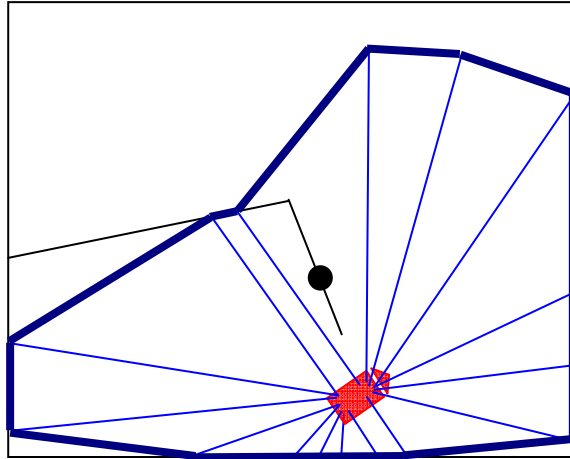
8.5 Situaciones de posicionamiento en zonas ocupadas

Tanto utilizando el método general como el método parcial, pueden encontrarse situaciones particulares en las que el baricentro se sitúa en zonas de espacio ocupado:

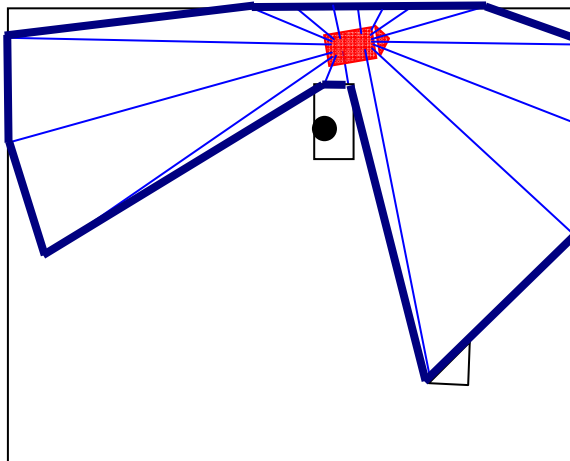




Hay detecciones que se deben a las limitaciones propias de la implementación llevada a cabo, haciendo que los sensores en posiciones concretas no detecten una zona ocupada (los muros reales tendrían grosor, el ejemplo sólo sirve para ilustrar una situación posible debida al número limitado de los sensores):



Otros casos de posicionamiento en zonas inaccesibles están más relacionados con la angulosidad de los objetos, obteniéndose que las lecturas más lejanas tienen rangos muy distantes de las lecturas más cercanas, existiendo un hueco profundo en la forma geométrica que puede dar lugar a baricentros sobre los objetos:



Estas situaciones han sido estudiadas con mayor profundidad en [8] para el método parcial del centro de áreas, donde se propone un proceso de "división" en los casos en los que el baricentro se sitúa en una zona inaccesible. Este proceso consiste en dividir en dos la forma geométrica justo por la zona angulosa:

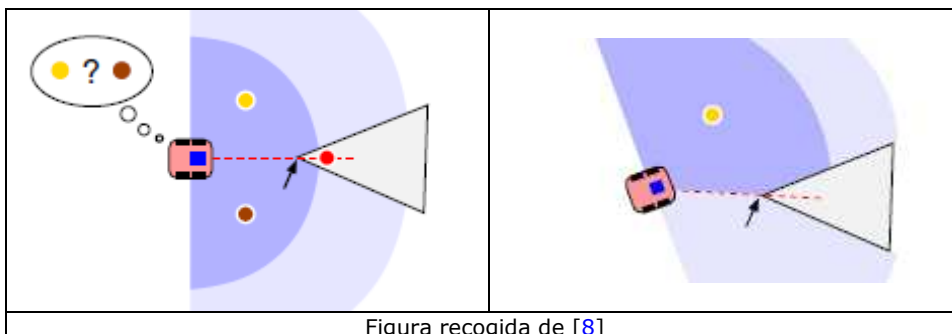


Figura recogida de [8]

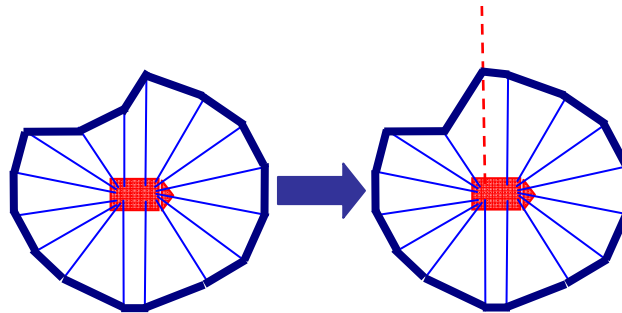
En caso de no existir simetría entre las zonas divididas se selecciona aleatoriamente una de ellas, de lo contrario se elegiría la zona de área mayor (en la figura sería la zona superior, ya que el objeto triangular está ligeramente rotado en el sentido de las agujas del reloj). Con esta zona se calcula el baricentro y se continúa el movimiento. Este proceso se repetiría tantas veces como fuese necesario, teniendo

siempre en cuenta el número finito de sensores de rango. Por ello, a través de las simulaciones realizadas en [8], se ha comprobado un buen funcionamiento de este mecanismo a partir de 36 sensores para cubrir los 360°, disponiendo de 18 sensores para el proceso de división (ya que con 8 sensores, el proceso de división da lugar a situaciones en las que el cálculo se tiene que realizar con uno o dos sensores).

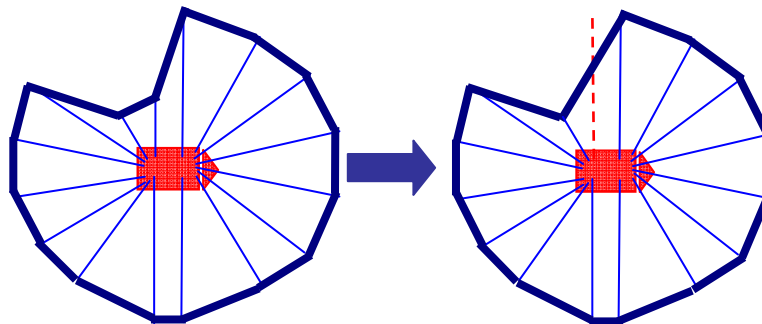
8.6 Comportamiento ante posibles fallos de los sensores

Cuando la lectura de un sensor falla, en la mayoría de las ocasiones devuelve el valor máximo. En [7] se realiza un análisis comparativo entre sensores ideales y sensores reales, considerándose en los últimos un porcentaje de fallo de referencia del 50% de las lecturas. Se utilizan dos mecanismos fundamentales:

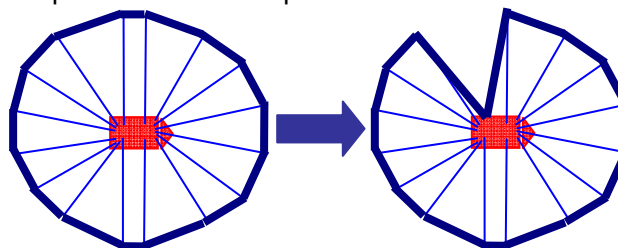
1. Se toma como medida máxima de los sensores una distancia menor a la permitida por los mismos, de manera que cuando el sensor devuelve el valor máximo (ya sea por fallo o sea por la detección de un objeto), la lectura se igualaría a la máxima posible. El inconveniente fundamental de este procedimiento, mantenido de forma constante, es que se limita el alcance de detección.



2. Cuando el sensor devuelve el valor máximo, adoptar un mecanismo de interpolación y extrapolación que calcule el valor de la lectura en base a las lecturas colindantes. El inconveniente fundamental de este mecanismo es el aumento de probabilidad de colisión, sobre todo ante la presencia de objetos de pequeño tamaño que no fuesen detectados con el balanceo del robot.



Si la lectura del sensor fuese la mínima, ante fallos puntuales la forma geométrica sufre un desequilibrio para ese sensor que tiene la forma:



A la que se le pueden aplicar los mismos procedimientos.

8.7 Pruebas de visión artificial

Para la ejecución de estas pruebas se han utilizado conjuntos de datos consistentes en secuencias de conducción en entornos reales, en concreto:

- (1) La secuencia [33 a], formada por un conjunto de 3674 imágenes en blanco y negro.
- (2) Las secuencias [33 b] consisten en dos conjuntos, cada uno de ellos compuesto de 2867 imágenes en color, uno de los cuales está capturado con una cámara frontal y el otro con una cámara trasera del mismo recorrido. Esta última se ha considerado desde un principio debido al uso de las ecuaciones de Kalman, ya que permite estudiar el comportamiento predictivo para el alejamiento de las líneas (tratando de anticipar la evolución hacia el horizonte), aunque finalmente no se ha aplicado de este modo. Por otro lado resulta de interés para la generalización de los filtros.

A continuación se muestra la primera imagen de cada una de las secuencias:







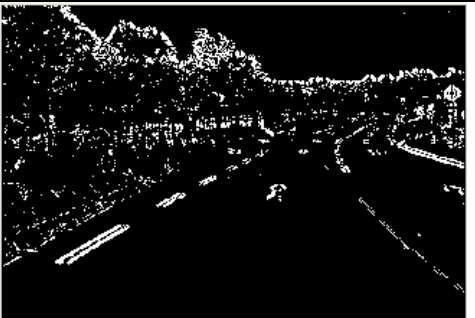




8.7.1 Pruebas de preprocesado de imágenes



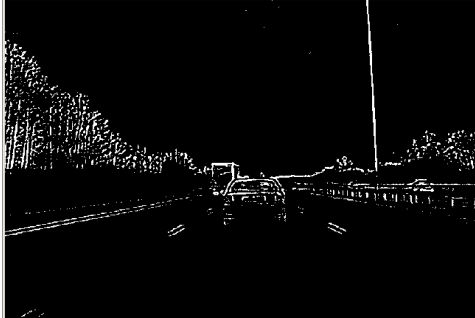
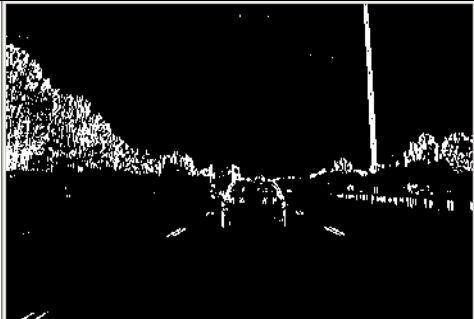





De forma predefinida se han ajustado (en base a una amplia batería de pruebas) un conjunto de configuraciones para la detección de bordes (menú *Predefined* de la barra de herramientas, apartado *Preprocessing*). Del mismo modo que sucede con otras configuraciones, debido al alto número de modificaciones, no todas estarán reflejadas con exactitud en el instalable que se puede descargar desde Sourceforge. Son las que se muestran a continuación:




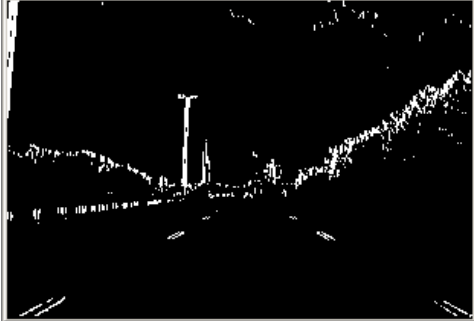
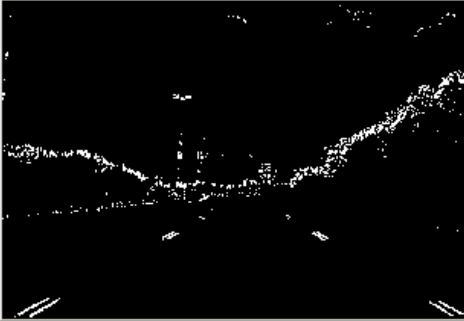

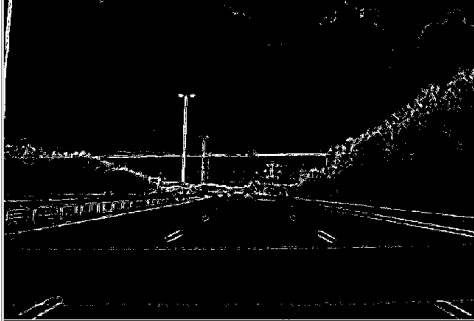


- *Laplace-4*: Blur, Median, Gaussian (3,0), Matrix (correlación cruzada con los valores cargados con la opción *Load Laplace4*), Threshold (5)
- *Laplace-8*: Blur, Median, Gaussian (3,0), Matrix (correlación cruzada con los valores cargados con la opción *Load Laplace8*), Threshold (15)
- *Sobel-X*: Blur, Median, Gaussian (3,0), Sobelx, Threshold (40)
- *Sobel-Min*: Blur, Median, Gaussian (3,0), Min(Sobelx|Sobely), Threshold (30)
- *Gaussian-LoG*: Blur, Median, Gaussian (5,0), Laplacian of Gaussian, Threshold (100)
- *LoG*: Blur, Median (5,0), Laplacian of Gaussian, Threshold (150)
- *Threshold*: Blur, Median, Gaussian (3,0), Threshold (220)
- *Canny*: Gaussian (7,0), Canny (50,100)

La aplicación directa de estas configuraciones se muestra para la primera imagen de la secuencia [33 a] en la Tabla 8.7.1.1, para la primera imagen frontal de la secuencia [33 b] en la Tabla 8.7.1.2 y para la primera imagen de la secuencia trasera en la Tabla 8.7.1.3.

Revisando las imágenes se puede observar que el filtro *Threshold*, a pesar de su utilidad y rapidez, requiere ser ajustado individualmente para cada secuencia, ya que es muy dependiente de los cambios de la luminosidad y color (para las secuencias [33 b] se comporta bien con un valor umbral de 110 o inferior, que será el utilizado para las pruebas de segmentación).

<p>Original, 1ª Imagen secuencia [33 a]</p> 	<p>Laplace-4</p> 	<p>Laplace-8</p> 
<p>Sobel-X</p> 	<p>Sobel-Min</p> 	<p>Gaussian-LoG</p> 
<p>LoG</p> 	<p>Threshold</p> 	<p>Canny</p> 
<p>Tabla 8.7.1.1</p>		

<p>Original, 1ª Imagen sec [33 b] frontal</p> 	<p>Laplace-4</p> 	<p>Laplace-8</p> 
<p>Sobel-X</p> 	<p>Sobel-Min</p> 	<p>Gaussian-LoG</p> 
<p>LoG</p> 	<p>Threshold</p> 	<p>Canny</p> 
<p>Tabla 8.7.1.2</p>		

Original, 1ª Imagen sec [33 b] trasera	Laplace-4	Laplace-8
		
Sobel-X	Sobel-Min	Gaussian-LoG
		
LoG	Threshold	Canny
		
Tabla 8.7.1.3		

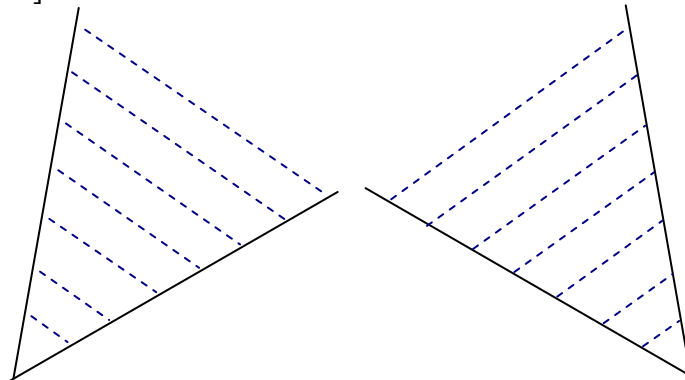
8.7.2 Pruebas de segmentación de imágenes para la detección de objetos

De un modo similar se han preconfigurado algunas opciones para la segunda etapa, haciendo uso del conocimiento anterior. Estas opciones se muestran aplicadas a las primeras imágenes de cada secuencia en las tablas 8.7.2.1 (secuencia [33 a]), 8.7.2.2 (secuencia frontal [33 b]) y 8.7.2.3 (secuencia trasera [33 b]). Se corresponden con los siguientes valores:



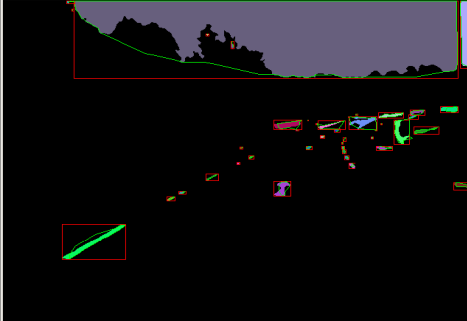
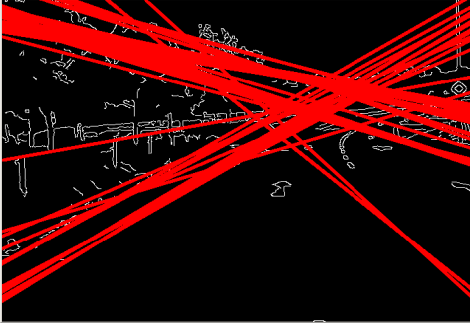
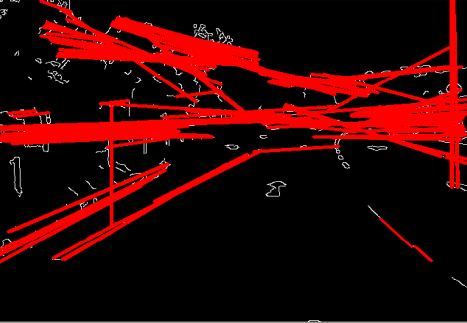

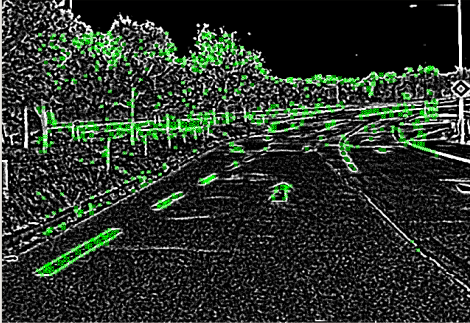
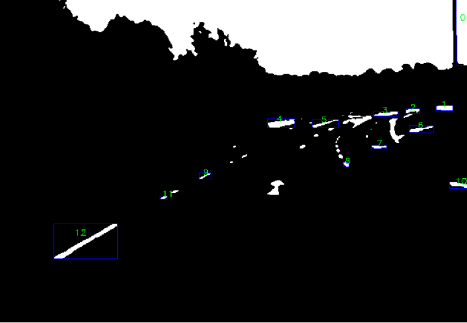
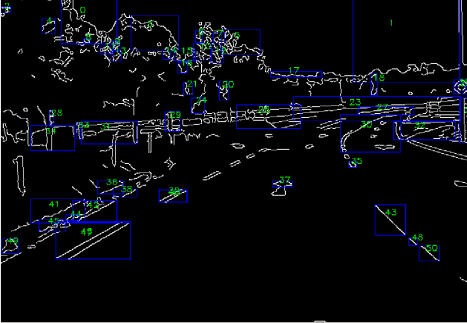
- *Contour-Threshold*: Opción de preprocesado *Threshold* (con valor por defecto, 220, para la primera secuencia y modificado para la segunda, a 110) con la detección de contornos basada en la función *cvFindContours*
- *Contour-Snake*: La opción anterior utilizando la función *cvSnakeImage* para delimitar los contornos (dividiendo el rectángulo en 5 partes)
- *HoughStd-Canny*: Opción de preprocesado de Canny con Hough estándar (acumulador = 70, rho=theta=1) y agregando la siguiente limitación en el método *drawHoughLines* de la clase *Segmentation* (basada en el conocimiento del dominio):



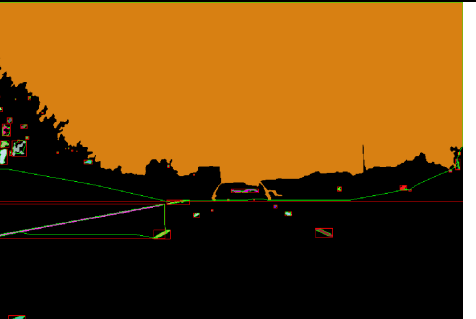
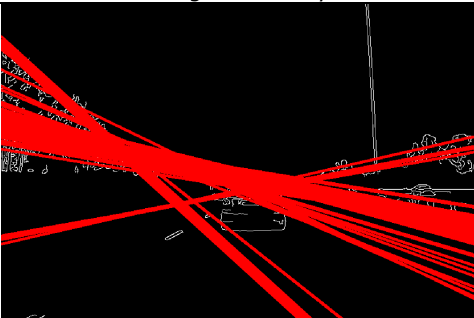
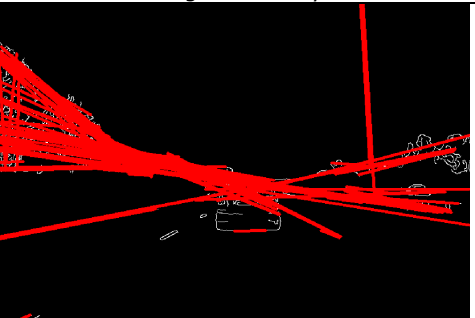

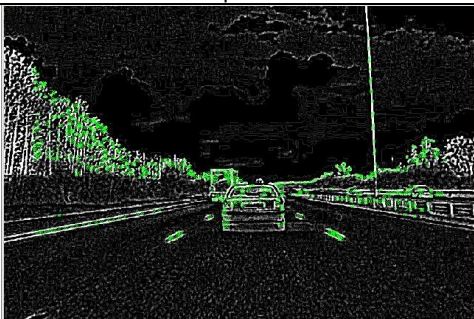
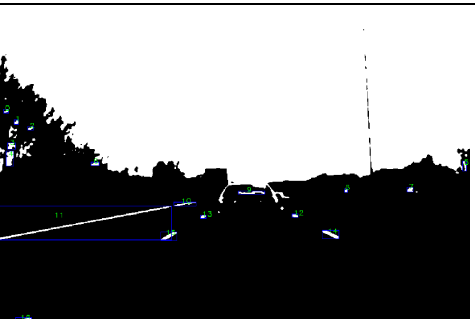
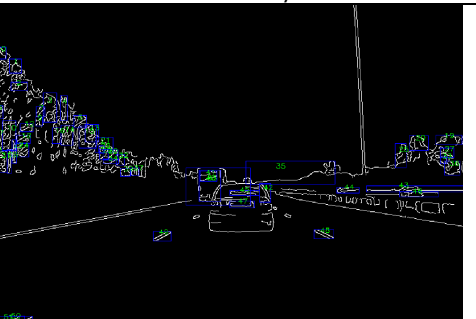
```
double thetaminleft = 30*CV_PI/180
double thetamaxleft = 80*CV_PI/180
double thetaminright = 100*CV_PI/180
double thetamaxright = 150*CV_PI/180
```




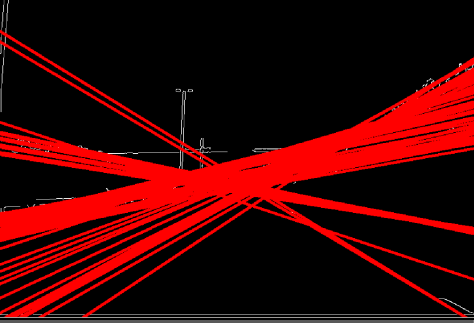
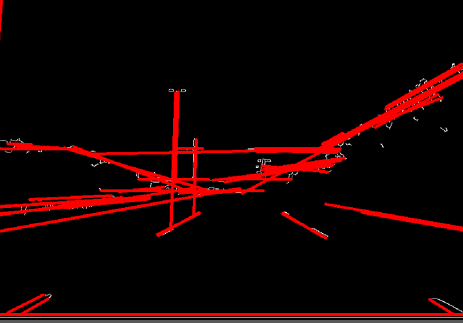



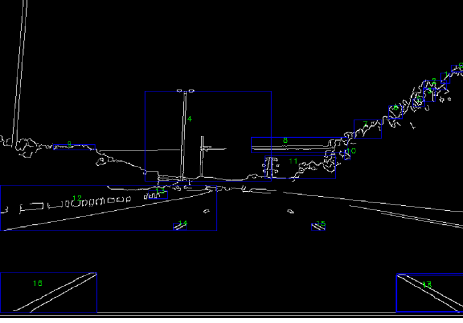
Limitándose la detección a los ángulos situados entre los rangos [30°,80°] y [100°,150°]



- *HoughPro-Canny*: Opción de preprocesado de Canny con Hough probabilístico (acumulador = 50, rho=theta=1, Min Line=20, Gap=50)
- *ScaleSpace*: La opción *Scale-Space Extremes* (con la parametrización: 6, 20, 100, 100, 2)
- *ScaleSpace-LoG* (versión mejorada del *ScaleSpace-Threshold* que se puede encontrar en la versión 0.3 de la aplicación): Se utiliza el preprocesado de los tres filtros de ruido (con Kernel=3 y Sigma=0) y el *Laplacian of Gaussian 5x5*. En la siguiente etapa se usa *Scale-Space Extremes* (con la parametrización: 32, 100, 200, 50, 2)
- *Blobs-Threshold*: Configuración de preprocesado *Threshold* con la opción de detección *Blobs-blobslib* (con los parámetros: 20, 2000, 0.9, 1.2)
- *Blobs-Canny*: Configuración de preprocesado *Canny* con la opción *Blobs-blobslib* (esta vez con los parámetros: 10, 2000, 0.9, 3)

<p>Original, 1ª Imagen secuencia [33 a]</p> 	<p>Contour-Threshold</p> 	<p>Contour-Snake</p> 
<p>HoughStd-Canny</p> 	<p>HoughPro-Canny</p> 	<p>ScaleSpace</p> 
<p>ScaleSpace-LoG</p> 	<p>Blobs-Threshold</p> 	<p>Blobs-Canny</p> 
<p>Tabla 8.7.2.1</p>		

<p>Original, 1ª Imagen sec [33 b] frontal</p> 	<p>Contour-Threshold</p> 	<p>Contour-Snake</p> 
<p>HoughStd-Canny</p> 	<p>HoughPro-Canny</p> 	<p>ScaleSpace</p> 
<p>ScaleSpace-LoG</p> 	<p>Blobs-Threshold</p> 	<p>Blobs-Canny</p> 
<p>Tabla 8.7.2.2</p>		

Original, 1ª Imagen sec [33 b] trasera	Contour-Threshold	Contour-Snake
		
HoughStd-Canny	HoughPro-Canny	ScaleSpace
		
ScaleSpace-LoG	Blobs-Threshold	Blobs-Canny
		
Tabla 8.7.2.3		

De la infinidad de ensayos que se han realizado para ajustar lo mejor posible estas configuraciones, cabe destacar la independencia obtenida en relación a las diferentes secuencias de prueba y a sus distintas escenas, el de las siguientes:

- Hough (tanto la estándar como la probabilística) que, además, muestra en muchas ocasiones la forma triangular o trapezoidal característica de la visión en perspectiva frontal hacia el punto de fuga, con la que se calcularía el centro de áreas.
- Scale Space, sobre todo con la aplicación previa de la transformación Laplaciana de Gaussiana, que posibilita el cálculo del mismo modo que la anterior, aunque requeriría el ajuste de los puntos.

Las configuraciones en las que se utiliza el filtro *Threshold* son tremendamente dependientes del tipo de imagen, incluso trabajando con la misma escena, por lo que sería necesario añadir algún método de ajuste automático adicional, que mitigaría su eficacia.

8.7.3 Pruebas de seguimiento como contingencia ante discontinuidades

Para la prueba de seguimiento se ha seleccionado una escena que posee varios ingredientes de interés, como son:

- Se produce una curva a la izquierda.
- Los márgenes de la carretera no se encuentran bien delimitados entre un carril y otro.
- Existe tráfico en sentido contrario.
- En la zona de los arcones después de la curva hay coches aparcados que comprimen la zona de espacio vacío.
- En lo que se refiere a los aspectos visuales, existe un cambio en la iluminación sobre el asfalto (debido a los distintos reflejos de la luz por la ondulidad existente antes de tomar la curva).

El conjunto de imágenes, procedente de la secuencia [33 a], está compuesta por 180 elementos: Desde la imagen 01m_24s_108957u.pgm hasta la imagen 01m_35s_029757u.pgm. A medida que se ha ido realizando la prueba, se plasma también el resultado de la imagen tras aplicar dos de las configuraciones de segmentación que han dado los mejores resultados: ScaleSpace-LoG y HoughPro-Canny.

El procedimiento seguido consiste en los siguientes pasos:

1. Se sitúa manualmente la aplicación (usando la ejecución paso a paso) sobre la primera imagen (01m_24s_108957u.pgm) y se seleccionan dos puntos (aproximadamente en la zona detectable por las configuraciones seleccionadas para la segmentación):
 - a. Margen izquierdo, de coordenadas (309,244)
 - b. Margen derecho, de coordenadas (373,243)



2. Se seleccionan:

- a. La opción que pinta una cruz verde sobre los puntos seleccionados originalmente (*Selected Paint*) para mantenerlos como referencia.
 - b. El alineamiento de la ordenada al hacer la predicción, para estudiar la evolución de la curva hacia la izquierda (*Align With Selected: Y*), haciendo el cálculo unidimensional de Kalman sobre el eje de abscisas.
 - c. La opción que pinta una cruz roja sobre la predicción calculada y actualiza el valor de la medida con el último valor introducido (*Prediction: Paint&LastUpdate*), confirmando al algoritmo que, hasta la siguiente medida, el valor introducido es válido.
3. Con esta configuración se avanza hasta que se comienza a percibir una divergencia entre los bordes y los puntos originales (que hasta entonces coincidirán con la predicción). Obviando una pequeña oscilación debida al bacheado, esto sucede en la imagen 33 (el archivo 01m_26s_004935u.pgm anotado como +32 respecto a la inicial). En este momento se actualizan las posiciones de las coordenadas de abscisas de ambos puntos y se cambia el tipo de actualización del filtro para que se vaya actualizando con el valor de la predicción (*Prediction: Paint&AutoUpdate*):
- a. 309 se actualiza a 301
 - b. 373 se actualiza a 363



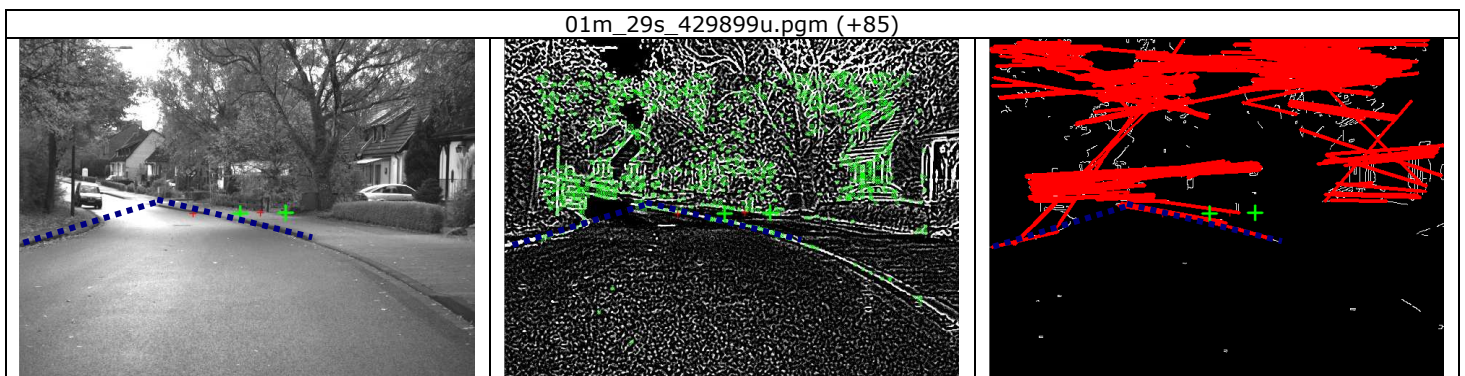
4. Manteniendo la configuración, se vuelven a actualizar las coordenadas, rectificándolas en el sentido de la curva, en las siguientes imágenes:
- a. En 01m_26s_060829u.pgm (+33)
 - i. 309 se actualiza a 296
 - ii. 373 se actualiza a 358
 - b. En 01m_26s_866901u.pgm (+46)
 - i. 309 se actualiza a 257
 - ii. 373 se actualiza a 350
 - c. En 01m_27s_430807u.pgm (+51)
 - i. 309 se actualiza a 234
 - ii. 373 se actualiza a 342
 - d. En 01m_29s_429899u.pgm (+85)
 - i. 309 se actualiza a 198
 - ii. 373 se actualiza a 270
5. A partir de aquí se deja evolucionar la escena, únicamente se realiza una última actualización para rectificar el valor a la salida de la curva. Esta actualización se produce sobre el margen derecho, ya que no se encuentra ninguna referencia sobre la que actualizar el valor del margen izquierdo. Se hace en la imagen 01m_31s_505231 (+120), 373 se actualiza a 338.

A continuación se muestra una tabla con las seis actualizaciones realizadas sobre el conjunto total de 180 imágenes:

Imagen / posición	Actualización de x=309	Actualización de x=373
01m_26s_004935u / +32	301	363
01m_26s_060829u / +33	296	358
01m_26s_866901u / +46	257	350
01m_27s_430807u / +51	234	342
01m_29s_429899u / +85	198	270
01m_31s_505231u / +120	-	338

En las tablas 8.7.3.1, 8.7.3.2 y 8.7.3.3 se ha pretendido resumir el proceso de prueba sobre las 180 imágenes, tomando escenas relevantes que muestren la utilidad del filtro de Kalman. A continuación se muestran algunas conclusiones:

- Únicamente con 6 actualizaciones (lo que supone una frecuencia de actualización media de 0,03) se consigue disponer de puntos de referencia con los que construir la zona segura sobre la que se realizaría el cálculo del centro de áreas.
- Las 4 primeras actualizaciones se realizan en un intervalo pequeño en relación al total (en 1.5 segundos aproximadamente frente a los 11 segundos que dura la escena, disponiendo de unas 16 imágenes por segundo), siendo las que marcan el ritmo predictivo para casi la totalidad de la curva. El comportamiento del algoritmo en relación a estas escenas se puede observar en la tabla 8.7.3.1.
- La última imagen de la tabla 8.7.3.1 muestra una desviación significativa, precisamente al acentuarse la curvatura, por lo que se necesita realizar la siguiente actualización. El criterio para actualizar el valor para el margen izquierdo (manualmente en el ensayo) podría ajustarse, en este caso concreto, a través del punto de fuga (línea punteada en azul). Para el margen derecho existen puntos de referencia detectados en el margen.



- En la tabla 8.7.3.2 se puede observar el buen comportamiento del filtro a lo largo de toda la curva, más aún teniendo en cuenta que no se está rectificando el giro que va dando el coche (que sería conocido puesto que la trayectoria se iría computando y no estaría previamente fijada, como sucede al utilizar los Datasets de conducción). No obstante, se ha optado por actualizar de nuevo el valor del margen derecho para ajustar la desaceleración a la salida de la curva, ya que se disponía para ello de una buena detección del borde.
- En la tabla 8.7.3.3 se presenta el resto de la salida de la curva, sin ninguna actualización se puede observar que la seguridad de la zona supera la que se detectaría incluso si se tomara como referencia el coche aparcado a la izquierda.

01m_24s_108957u.pgm	01m_28s_249495u.pgm (+65)	01m_28s_845289u.pgm (+75)	01m_29s_429899u.pgm (+85)

Tabla 8.7.3.1

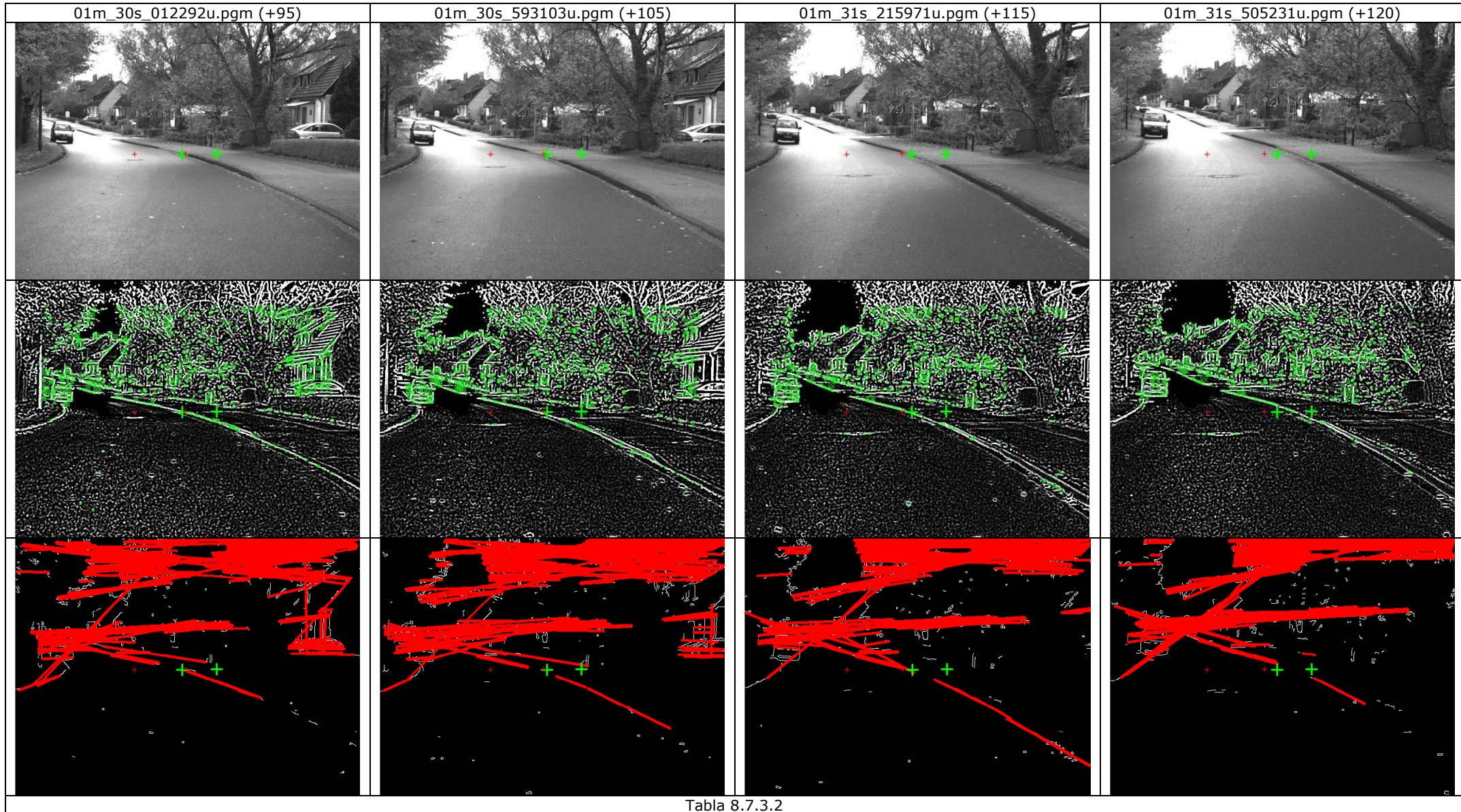


Tabla 8.7.3.2

01m_32s_673098u.pgm (+140)	01m_33s_792280u.pgm (+159)	01m_34s_395119u.pgm (+169)	01m_35s_029757u.pgm (+179)

Tabla 8.7.3.3

9 CONCLUSIONES Y TRABAJO FUTURO

Parece evidente que la inquietud del ser humano por la creación de objetos artificiales con el objetivo de realizar tareas de forma autónoma, ha formado parte del desarrollo de todas las culturas de las que tenemos conocimiento, cada una con sus motivaciones particulares y bien diferenciadas, enmarcándose en alguno de los siguientes ámbitos: (1) Religioso: La creencia de seres superiores creadores, los dioses, conlleva implícitamente la creación de seres vivos a partir de la materia que nos rodea. (2) Práctico: La creación de instrumentos que sirven para el control automático de tareas, la diversión, la guerra, la obtención y el tratamiento de los recursos, el conocimiento científico, etc.

Las aplicaciones prácticas de esta idea original en nuestros días son innumerables y las aportaciones al conocimiento científico han adquirido un papel fundamental. A lo largo del desarrollo de este trabajo, sin ir más lejos, se ha recurrido al conocimiento de varios ámbitos científicos, que da una idea de la multidisciplinariedad del área si se tiene en cuenta la especificidad del objetivo.

Uno de los problemas más relevantes en el estudio de los robots autónomos es el que tiene como objetivo que el robot pueda realizar simultáneamente la localización en un entorno desconocido mientras realiza la construcción del mapa, sabiéndose posicionar en él, denominado SLAM - Simultaneous Localization And Mapping - y sobre el que actualmente existen soluciones propuestas ([20], [21]) que están pendientes de verificación a gran escala y en entornos en los que se requiera mucha precisión sin la utilización de sistemas de localización como los GPS, tal y como podría ser la conducción de un coche. El método presentado en este trabajo permite valorar la utilidad de disponer de puntos de referencia para el robot, cuyas características de invariancia los convierten en información para su uso subjetivo, que puede ser utilizado de forma complementaria al desarrollo de sistemas que resuelven el problema SLAM. La arquitectura, construida a través de los estándares y herramientas mencionados, da cabida a la utilización del método de interpretación que se desee conjuntamente con otros módulos para un comportamiento híbrido, tal y como sería un sistema que resuelve SLAM.

El refuerzo de la percepción sensorial endógena, procesada a través del centro de áreas (a partir de esquemas de comportamiento reactivo que hacen uso de las medidas recogidas con los sensores de rango), con un sistema de visión artificial, ha mostrado posibles aplicaciones justo allí donde el cálculo con otros sensores se convierte en una tarea muy compleja. Ejemplo de estas situaciones son: La detección de límites sin relieve, el cálculo de la angulosidad de objetos para la localización de zonas libres o el uso de la visión en perspectiva plana para localizar los puntos de fuga. Estas técnicas, elaboradas con mayor detalle, podrían servir para complementar los métodos general o parcial del centro de áreas y aumentar la probabilidad de acierto.

El trabajo futuro mostrado a continuación únicamente pretende reflejar cuestiones que han ido surgiendo a lo largo del desarrollo del trabajo de investigación:

- Integración, bajo el mismo ciclo de ejecución, de los datos procesados con el sistema de visión y de otros sensores para salvaguardar al sistema de los errores propios de los s3nar: reflexión especular, recorte del contorno y acoplamiento accidental. D3ndole continuidad a [8], extracci3n de las zonas angulosas ofreciendo soporte para el c3lculo en el proceso de divisi3n, con el que se evita el movimiento hacia zonas ocupadas.
- Desarrollo de la matem3tica proyectiva subyacente al estudio de los puntos de fuga. Localizaci3n y estudio de las trayectorias.
- An3lisis num3rico detallado de todos los resultados, as3 como el estudio de las distribuciones y figuras geom3tricas obtenidas (triangular y traapezoidal). An3lisis de nuevas formas.

- Inclusión de mecanismos topológicos utilizables para la localización y la planificación de ruta, teniendo en cuenta que contengan las funciones que dan respuesta a las siguientes preguntas (de [11]): ¿Hacia dónde voy?, ¿Cuál es el mejor camino?, ¿Por dónde he estado? y ¿Dónde estoy?
- Mejoras del interfaz:
 - Integrar el desarrollo realizado en un GUI propio y único que permita de forma visual el manejo de toda la parametrización.
 - Posibilidad de modificar sobre el interfaz de usuario el orden de ejecución de los filtros de visión. Extensión de estos filtros y las etapas en las que están distribuidos.
 - Conexión del software de visión artificial con algún tipo de interfaz de simulación (se ha valorado el desarrollo de escenas en 3D con los que simular la conducción, sustituyendo las secuencias de prueba).

10 APÉNDICE I: INSTALACIÓN DEL SOFTWARE PARA LA SIMULACIÓN EN WINDOWS

10.1 Comunes

Todo el software desarrollado se ha apoyado en dos herramientas con las que se ha pretendido algo más que aportar una mera documentación del código, esto es, han sido elegidas para tratar de dar visibilidad, a modo de esquemas conceptuales, a factores como la utilidad o las posibilidades de las diferentes capas, así como ayudar a mejorar la comprensión sobre las mismas. Estas herramientas son:

- StarUML, para los diagramas UML (<http://staruml.sourceforge.net>)
- Doxygen, para la documentación de navegación HTML en formato API (www.doxygen.org), conjuntamente con Graphviz para la generación de esquemas gráficos (<http://www.graphviz.org>)

En un ámbito más general, se ha elegido la herramienta de código libre ampliamente utilizada 'Eclipse' ([29]) para la implementación de la primera versión de la aplicación de visión artificial, mientras que, por dificultades de compatibilidad y soporte de algunas de las librerías utilizadas en robótica, se ha seleccionado el Visual Studio 2005 para llevar a cabo los diferentes módulos de comunicación con el robot y, por extensión, para el método del centro de áreas. Finalmente, la aplicación de visión artificial también se ha compilado en Visual Studio 2005.

No obstante, se ha realizado un esfuerzo notable para conseguir una integración estable en relación a lo siguiente:

- El uso del Visual Studio 2010 (ya que una parte de este trabajo ha coincidido en tiempo con su lanzamiento de prueba gratuito 'beta 2'), consiguiéndose una estabilidad similar realizándose cambios de configuración poco significativos.
- La compatibilidad del código en otros sistemas operativos, tal y como es el sistema Linux debian, que se consigue con pequeños esfuerzos relacionados sobre todo con los cambios de estructura y referencias a algunas de las utilidades (librerías) embebidas.
- Migración íntegra a la plataforma Eclipse: no se ha dejado con este formato por la falta de soporte del middleware de MobileRobots e inestabilidades de difícil solución, incluyendo algunas del entorno visual (QT), quedando lejos de la robustez que se logra con el Framework definitivo.

10.2 Robótica

Instalación inicial en Windows:

- En la carpeta *c:/MobileRobots*, se ha instalado el siguiente software:
 - MobileSim 0.4.0, en la carpeta *MobileSim4*
 - Aria 2.7.2, en la carpeta *Aria*
 - ARNL Base Library 1.7.2, ARNL 1.7.0 y SONARNL 1.7.0, en la carpeta *ARNL*
 - Mapper3 2.2.5, en la carpeta *Mapper3*
 - MobileEyes 2.2.4, en la carpeta *MobileEyes*
 - ACTS 2.3, en la carpeta *ACTS*
 - Demo de ACTS 2.3.0, en la carpeta *ACTSDemo*
- En la misma carpeta *c:/MobileRobots* se ha creado un directorio *maps*, para almacenar los mapas

Correcciones de la primera instalación para evitar problemas de funcionamiento:

- MobileSim 0.4.0, en la carpeta *MobileSim4*
- Se cambia la versión anterior de ARIA por la versión 2.5.1
- Se cambia a las versiones de ARNL y SONARNL 1.5.1

Es lógico pensar que en el intervalo de tiempo que vaya desde la redacción de este trabajo hasta el momento actual, sobre todo teniendo en cuenta el ritmo de desarrollo de MobileRobots y la GUI QT, algunos de los problemas encontrados se hayan solucionado. Otro factor a tener en cuenta es la variabilidad de los diferentes sistemas Windows.

10.3 Visión artificial

El proceso seguido para la instalación de OpenCV ha sido el siguiente:

- Se descarga la última versión existente en el momento de desarrollo (2.0)
- Es necesario compilar las librerías, para lo que se descarga la herramienta CMake (<http://www.cmake.org/cmake/resources/software.html>) versión 2.8.0. A través del interfaz gráfico de este programa, se selecciona el directorio donde se encuentra openCV y el directorio de salida vs2005, en el que se generará una solución preparada para compilar. A continuación se configura el Visual Studio de manera general (menú *Tools* → *Options*, sección *VC++ Directories*):
 - Seleccionando *Include files*, se agrega `c:\OpenCV2.0\include\opencv`
 - Seleccionando *Library files*, se agregan `c:\OpenCV2.0\vs2005\lib\debug` y `c:\OpenCV2.0\vs2005\lib\release`
 - Seleccionando *Source files*, se incluyen `C:\OpenCV2.0\src\cv`, `C:\OpenCV2.0\src\cvaux`, `C:\OpenCV2.0\src\cxcore`, `C:\OpenCV2.0\src\highgui` y `C:\OpenCV2.0\src\ml`
- Compilar `cvblobslib` para integrarla en el proyecto:
 - Se abre el proyecto y se transforma de forma automática
 - El control de errores se comenta (código asociado con `_SHOW_ERRORS`)
 - Se compila la librería de release como `cvblobslib.lib` y la de debug como `cvblobslibd.lib`
 - Estas librerías se incluyen en el proyecto de visión, en el directorio *libs*, que se agrega como directorio adicional de librerías a todas las configuraciones
 - Las cabeceras se incluyen en el directorio *include*

11 APÉNDICE II: EJECUCIÓN DE LA SIMULACIÓN DEL ROBOT

Para la simulación se han utilizado diferentes modos de ejecución en el simulador MobileSim de MobileRobots:

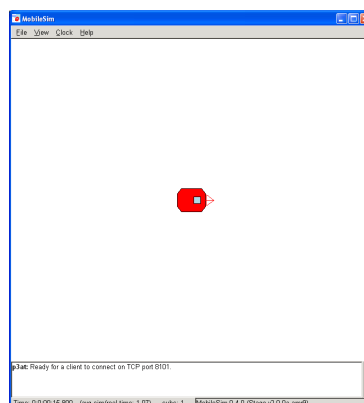
- Robot sin mapa (para pruebas de comunicación): El simulador del robot se arranca desde la consola, situándose en el directorio *bin* de MobileSim, ejecutando: `MobileSim --nomap -r p3at`
Para este tipo de ejecución se ha preparado el bat: `mobilesim_sinmapa.bat`
Para comprobar la comunicación entre el robot, el servidor y el cliente, se ejecuta el programa *serverDemo* incluido en Aria (directorio *bin*). Una vez conectado el servidor al robot, se puede probar la comunicación con el cliente a través del software *MobileEyes*, indicando el servidor "localhost":



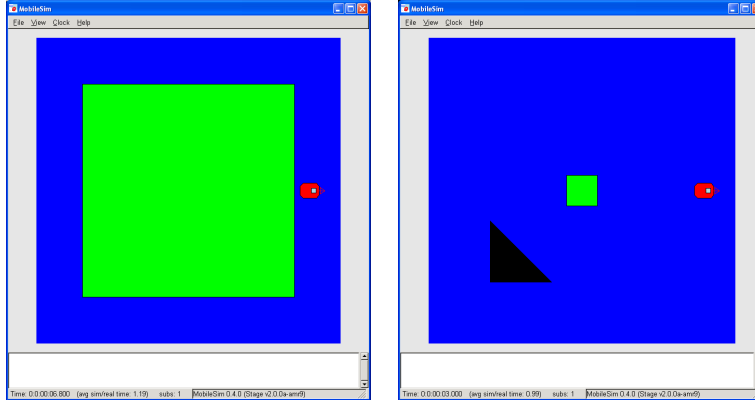
A través de *MobileEyes*, conectado al servidor, se pueden realizar acciones sobre el robot (como Tele-operación o Wander).

- Robot con mapa de tipo stage (algo más complicado de construir pero muy útil si se desean mover los objetos). El archivo *world* incluye la definición del contorno total en un archivo aparte `[nombre_archivo].inc`, siendo necesario incluirlo en el directorio raíz para evitar utilizar referencias absolutas. Su ejecución sería en este caso:
`MobileSim -W [ruta_al_archivo_world]`
La ruta utilizada al archivo *world* hace referencia al directorio `..\maps\`
Para la ejecución del mundo llamado *cuadrado.world* se ha creado el archivo `mobilesim_wcuadrado.bat`
Para la ejecución del mundo llamado *varios.world* se ha creado el archivo `mobilesim_wvarios.bat`
- Robot con mapas MobileRobots/ActivMedia. La ejecución de cada mapa se encuentra separada en un archivo bat, con la siguiente sintaxis: `mobilesim_map[n].bat`, siendo `[n]` el número de mapa.

Estos modos de ejecución son los siguientes:



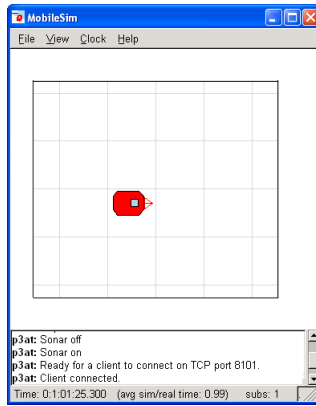
mobilesim_sinmapa.bat



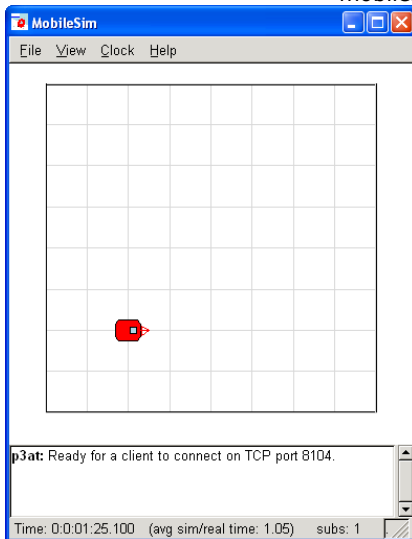
mobilesim_wcuadrado.bat

mobilesim_wvarios.bat

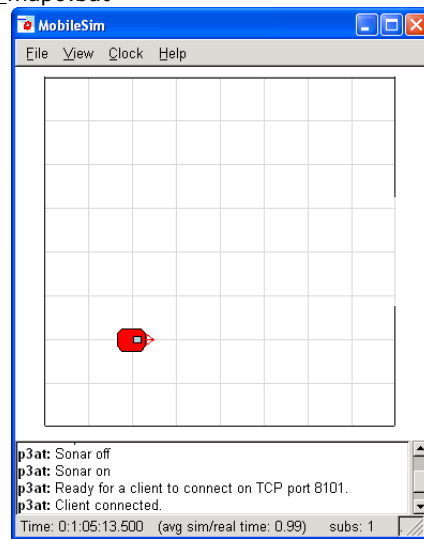
Los archivos.inc asociados a los mapas .world se sitúan en el directorio raíz de MobileSim



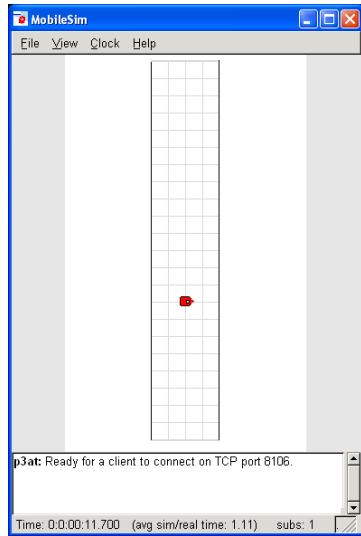
mobilesim_map0.bat



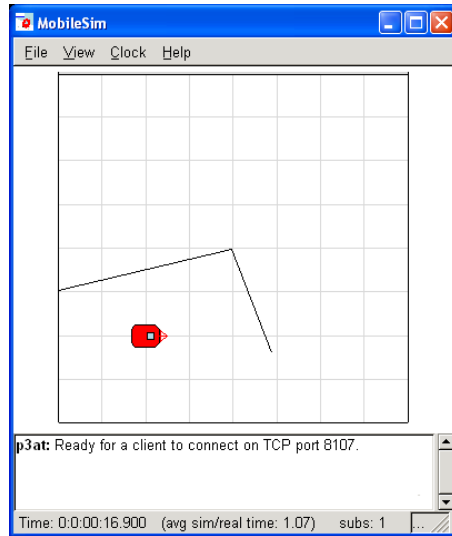
mobilesim_map1.bat



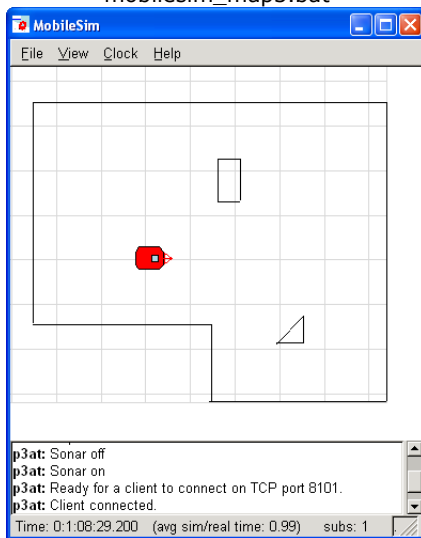
mobilesim_map2.bat



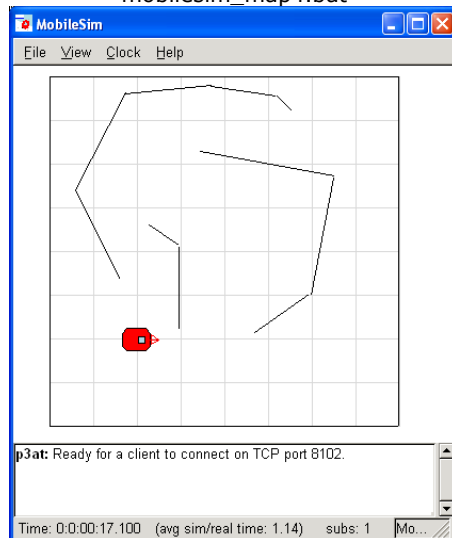
mobilesim_map3.bat



mobilesim_map4.bat



mobilesim_map5.bat



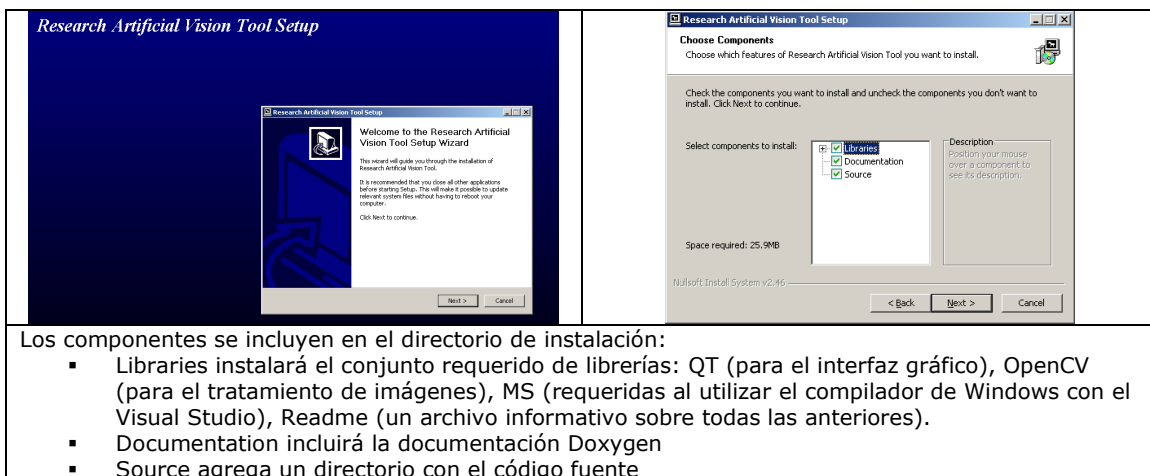
mobilesim_map6.bat

12 APÉNDICE III: INSTALACIÓN Y EJECUCIÓN DE RESEARCH ARTIFICIAL VISION TOOL

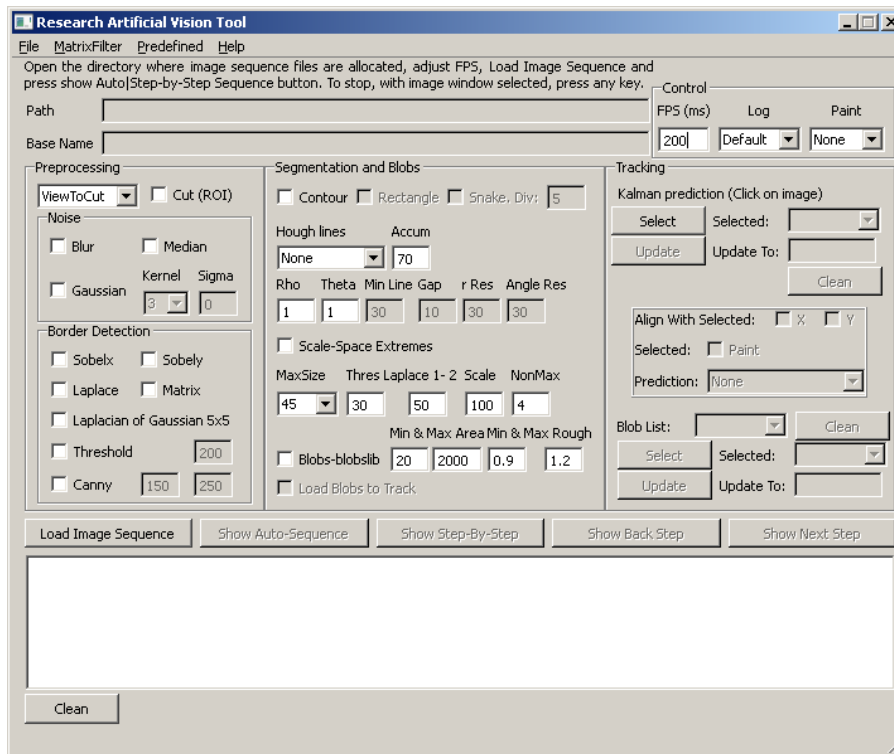
Como ya se ha comentado, el lenguaje de programación utilizado es C++, permitiendo la integración de la librería OpenCV ([25]) y de un desarrollo para la detección de blobs (Blob Extraction Library, [30]) cuyo buen comportamiento ha sido comprobado en estos y otros ensayos. El entorno de trabajo ha pasado de ser *Eclipse* ([29]) con el compilador de mingw y las librerías para el diseño del interfaz gráfico QT integradas, a ser Visual Studio 2005 con el compilador de Microsoft y el módulo QT también integrado, buscando la máxima semejanza con los módulos de robótica.

La instalación se realiza a través de un ejecutable que se encarga de situar las librerías necesarias en el directorio seleccionado durante el proceso, evitando así obligar al usuario a tener que instalar una cantidad sustancial de software adicional, que sería necesario únicamente si lo que se desea es realizar un desarrollo propio. La generación de este instalable se ha llevado a cabo con el plugin *EclipseNSIS*, en el caso de la versión construida bajo *Eclipse*, o utilizando directamente el software *NSIS* (que en cualquier caso es requerido) en el caso de la versión construida bajo Visual Studio.

La aplicación resultante se ha denominado '*Research Artificial Vision Tool*' ([32]), quedando a disposición de todos aquellos que quieran usarla. Se ha publicado con licencia GPL y se ha alojado en el repositorio *Sourceforge*, destinado a este tipo de programas de código libre. Una vez se dispone del ejecutable (*ArtificialVisionInstallerV03.exe*) su instalación es semejante a la de cualquier otra aplicación para Windows, mostrando un conjunto de pantallas aproximadamente estándar:



Una vez instalada se puede ya ejecutar como cualquier otra aplicación instalada en el sistema, desde el menú Inicio de Windows (bajo el directorio de instalación aparecerá una opción *Run Research Artificial Vision Tool*). Casi la totalidad de la funcionalidad incluida en la aplicación está integrada en la pantalla principal, que tiene el siguiente aspecto:



Lo siguiente que se necesita, antes de poder realizar ninguna prueba, es la secuencia de imágenes sobre la que verse nuestro experimento. Para realizar las pruebas en este trabajo se han seleccionado los Datasets [33 a] y [33 b], que consisten en grabaciones realizadas desde vehículos en circulación por diferentes tipos de carretera. Los formatos de imagen admitidos por código son jpg/jpeg, png y pgm, que podrían ser ampliados con cierta facilidad a otros formatos de interés. La secuencia de imágenes se muestra siguiendo el orden establecido por los nombres de los archivos, siendo recomendable que tengan el mismo número de caracteres, ya que de otro modo puede dar lugar a ordenaciones inesperadas. Para abrir la secuencia basta con ir al menú *File* y seleccionar *Open Directory*. Se selecciona el directorio en el que se encuentra la secuencia de prueba y, en el marco inferior, se presiona el botón *Load Image Sequence* (enmarcado en rojo en Fig. 12.1). Durante la carga aparecerá un mensaje informativo en el cuadro de texto del marco inferior (enmarcado en rosa en Fig. 12.1), una vez estén cargadas las imágenes se habilitan dos opciones (enmarcadas en azul en Fig. 12.1):

- (1) *Show Auto-Sequence*, con la que se consigue mostrar la secuencia al ritmo indicado en el marco de control, cuadro de texto correspondiente a los FPS (enmarcado en violeta en Fig. 12.1). Lo que se indica aquí es el intervalo de tiempo, en milisegundos, que el programa queda en espera entre la ejecución de una imagen y la siguiente. El funcionamiento de este parámetro es dependiente de la carga computacional, a medida que ésta aumenta, el tiempo se va convirtiendo en un retraso añadido. Es editable en tiempo de ejecución, salvo en el momento en el que se está tratando la imagen, proceso que bloquea las interfaces. La máxima velocidad del programa se consigue poniéndolo a vacío o a cero.
- (2) *Show Step-By-Step*, permite la ejecución paso a paso de cada imagen de la secuencia a través de la intervención manual del usuario, que se consigue a través de los botones *Show Back Step* / *Show Next Step* (enmarcados en negro en Fig. 12.1), una vez se encuentran habilitados. La computación sobre la imagen, en este caso, no se produce sobre la imagen actual sino sobre la siguiente o la anterior, dependiendo del botón pulsado. Esta misma cuestión es aplicable en el caso (1), las funciones implementadas para el tratamiento de la imagen se pueden ir modificando en tiempo de ejecución,

sin embargo son aplicables a la siguiente imagen cargada, nunca a la actual, puesto que lo que se muestra por pantalla ya es el resultado de aplicar los algoritmos seleccionados.

En caso de desear detener la ejecución actual, es necesario marcar la ventana destinada a mostrar las imágenes como pantalla activa y, acto seguido, pulsar cualquier tecla (la cruz de cierre situada en el borde superior derecho únicamente funcionará si la opción seleccionada es la (2)).

Otros datos de interés para entender la ejecución son los siguientes:

- Si se desea cargar otra secuencia de prueba es necesario repetir el proceso de nuevo. Esto puede dar lugar a alguna confusión debido a que en la zona del marco superior, en la opción denominada *Path* (enmarcada en verde en Fig. 12.1), lo que se almacena es la última ruta señalada desde el menú *File*, no la última secuencia cargada con el botón *Load Image Sequence*.
- El nombre de la imagen actual se puede obtener de dos modos, siendo la más directa la zona del marco superior denominada *Base Name* (enmarcada en amarillo en Fig. 12.1). Otro modo es activar el log "Simple" desde el marco de control (enmarcado en naranja en Fig. 12.1), pero esta opción incluirá más datos a medida que se vayan agregando funciones al tratamiento de las imágenes (en concreto, irá indicándonos cada uno de los filtros aplicados).
- Una de las simplificaciones más importantes que se han llevado a cabo, consiste en la conversión a escala de grises antes de iniciar cualquier rutina de tratamiento de las imágenes. Por este motivo, aunque la secuencia de prueba consista en imágenes en color, al marcar cualquier filtro se realizará esta transformación, siendo la imagen resultante la correspondiente a esta versión sustituta de la original. El procesamiento de los tres canales para imágenes a color se ha decidido dejar como posible mejora futura.

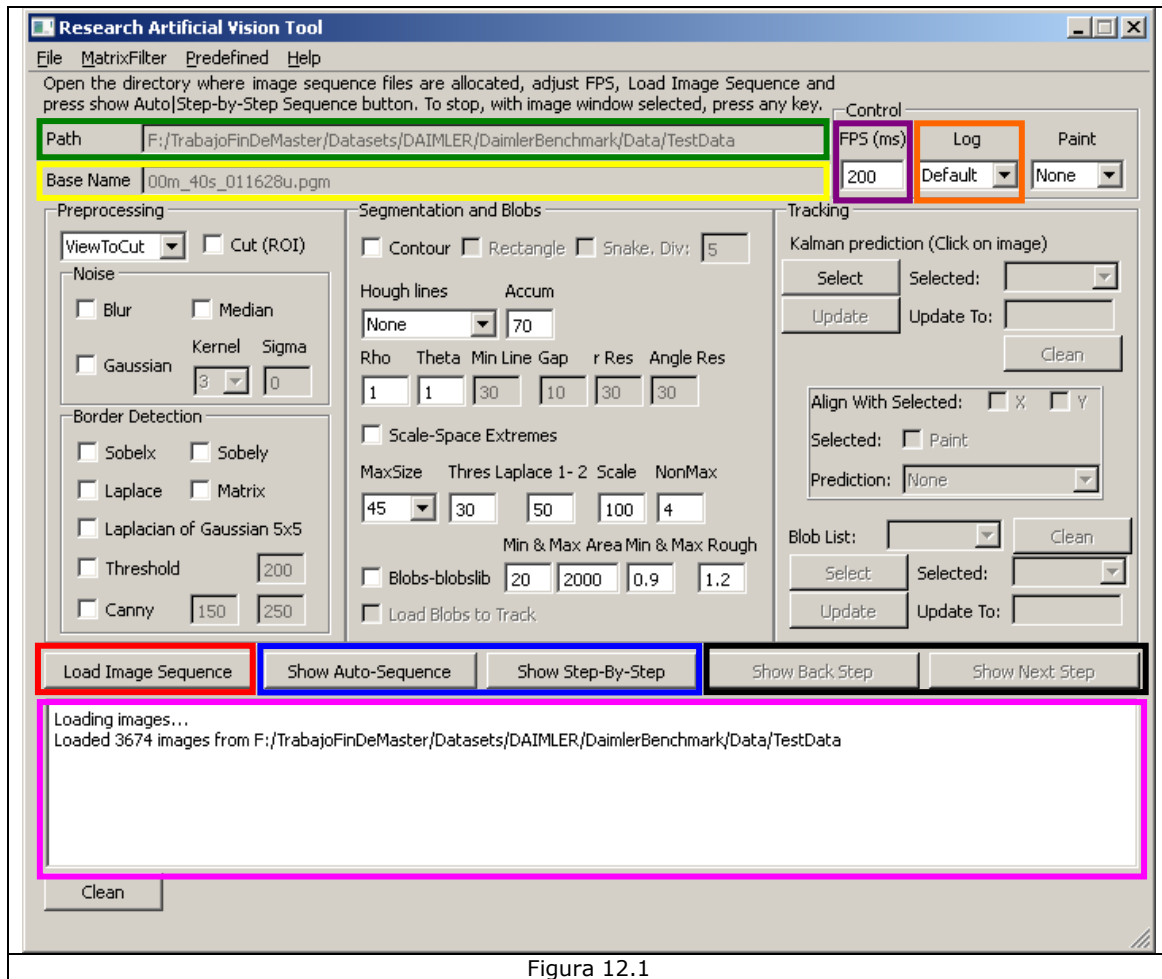


Figura 12.1

El marco central está destinado a las etapas de procesamiento y sus posibles parametrizaciones. En la barra de herramientas destaca la pestaña *Predefined*, en la que se han preconfigurado combinaciones de prueba para las dos primeras etapas, ya que marcar las diferentes opciones y ajustar sus parámetros se acaba convirtiendo en una tarea tediosa, sobre todo cuando se quieren repetir el proceso de forma sistemática.

13 REFERENCIAS

- [1] José R. Álvarez, Félix de la Paz and José Mira. On Virtual Sensory Coding: An Analytical Model of the Endogenous Representation. In José Mira and Juan V. Sánchez-Andrés editors, *Engineering Applications of Bio-Inspired Artificial Neural Networks*, volume 1607 of *Lecture Notes in Computer Science*, pages 526-539. International Work-Conference on Artificial and Natural Neural Networks, IWANN'99, Springer-Verlag, June 1999.
- [2] Félix de la Paz López y José Ramón Álvarez Sánchez. Topological Maps for Robot's Navigation: A Conceptual Approach. En José Mira y Alberto Prieto, directores, *Bio-Inspired Applications of Connectionism*, volumen 2085 de *Lecture Notes in Computer Science*, páginas 459-467. International Work-Conference on Artificial and Natural Neural Networks, IWANN 2001, Springer-Verlag, junio 2001.
- [3] Jose Mira, José Ramón Álvarez Sánchez, y Félix de la Paz López. The Knowledge Engineering Approach to Autonomous Robotics. En José Mira y José R. Álvarez, directores, *Artificial Neural Nets Problem Solving Methods*, volumen 2687 de *Lecture Notes on Computer Science*, páginas 161-168. International Work-Conference on Artificial and Natural Neural Networks, IWANN 2003, Springer-Verlag, junio 2003.
- [4] José Ramón Álvarez Sánchez, José Mira and Félix de la Paz López. El método del centro de áreas como mecanismo básico de representación y navegación en robótica situada. In Antonio Fernández Caballero, María Gracia Manzano, Enrique Alonso and Sergio Miguel Tomé editors, *Una Perspectiva de la Inteligencia Artificial en su 50 Aniversario*, volume 1, pages 103-117. CMPI-2006, Universidad de Castilla La Mancha, July 2006.
- [5] José R. Álvarez Sánchez, Félix de la Paz and José Mira. A Robotics Inspired Method of Modeling Accessible Open Space to Help Blind People in the Orientation and Traveling Tasks. In José Mira and Jose R. Álvarez editors, *Mechanisms, Symbols and Models Underlying Cognition*, volume 3561 of *Lecture Notes in Computer Sciences*, pages 405-415. International Work-conference on the Interplay between Natural and Artificial Computation, IWINAC 2005, Springer-Verlag, June 2005. http://dx.doi.org/10.1007/11499220_42
- [6] J. R. Álvarez Sánchez, J. Mira Mira, F. de la Paz López, J. M. Cuadra Troncoso. The centre of area method as a basic mechanism for representation and navigation. 2007 Elsevier, doi:10.1016/j.robot.2007.07.008
- [7] José Ramón Álvarez Sánchez, Félix de la Paz López, José Manuel Cuadra Troncoso and José Ignacio Rosado Sánchez. Partial Center of Area Method Used for Reactive Autonomous Robot Navigation. J. Mira et al. (Eds.): IWINAC 2009, Part II, LNCS 5602, pp. 408-418, 2009.
- [8] José Ramón Álvarez-Sánchez, Félix de la Paz López, José Manuel Cuadra Troncoso, Daniel de Santos Sierra. Reactive navigation in real environments using partial center of area method. *Robotics and Autonomous Systems* (2010), doi:10.1016/j.robot.2010.05.009
- [9] Encarnación Folgado, Mariano Rincón, Margarita Bachiller and Enrique J. Carmona. A Block-Based Human Model for Visual Surveillance. J. Mira et al. (Eds.): IWINAC 2009, Part II, LNCS 5602, pp. 208-215, 2009.
- [10] José Mira, Rafael Martínez, Mariano Rincón, Margarita Bachiller and Antonio Fernández-Caballero. Towards a Semi-automatic Situation Diagnosis System in Surveillance Tasks.
- [11] Robin R. Murphy. *Introduction to AI Robotics*. The MIT Press, 2000.
- [12] Howie Choset, Kevin Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia Kavraki y Sebastian Thrun. *Principles of Robot Motion*. The MIT Press, 2005.
- [13]
 - [a] Bruce Eckel. *Thinking in C++*. Volume One: Introduction to Standard C++. 2000. <http://mindview.net/Books/TICPP/ThinkingInCPP2e.html>

- [b] Bruce Eckel, Chuck Allison. Thinking in C++. Volume Two: Practical Programming. 2003. <http://mindview.net/Books/TICPP/ThinkingInCPP2e.html>
- [14] J. Mira. La Inteligencia Artificial Como Ciencia Y Como Ingeniería. 2006.
- [15] Rodney A. Brooks. A robust layered control system for a mobile robot. MIT, 1985. <http://people.csail.mit.edu/brooks/papers/AIM-864.pdf>
- [16] UNED and UCLM. AVISA: Un sistema semiautomático de diagnóstico de situaciones en tareas de vigilancia basado en técnicas de Atención Visual Selectiva y dinámica con capacidad de Aprendizaje. <http://www.ia.uned.es/proyectos/avisa/>.
- [17] ARIA (Advanced Robot Interface for Applications), es un conjunto de herramientas orientadas a objetos, escritas en lenguaje C++, que permiten controlar el servidor del robot. La distribución es de código libre, descargable desde <http://robots.mobilerobots.com/ARIA/>. Se ha utilizado la versión 2.5-1 para Windows.
- [18] Plataforma software de simulación de robots MobileSim, distribución de código libre, descargable desde <http://robots.mobilerobots.com/MobileSim/>. Se ha utilizado la versión 0.4.0-1 para Windows.
- [19] MobileEyes, Aplicación gráfica que sirve para monitorizar de forma remota tanto el robot como sus accesorios. El uso de esta aplicación requiere haber adquirido los derechos de la licencia. <http://robots.mobilerobots.com/MobileEyes/>.
- [20] Hugh Durrant-Whyte y Tim Bailey. Simultaneous Localization and Mapping: Part I. IEEE Robotics & Automation Magazine, junio 2006.
- [21] Hugh Durrant-Whyte y Tim Bailey. Simultaneous Localization and Mapping: Part II. IEEE Robotics & Automation Magazine, septiembre 2006.
- [22] Norbert Wiener. Cibernética o El control y comunicación en animales y máquinas. Escrito en 1948, editorial Tusquets, 2ª edición de 1998.
- [23] Ronald C. Arkin. Intelligent Mobile Robots in the Workplace: Leaving the Guide Behind. Proceedings of the 1st international conference on Industrial and engineering applications of artificial intelligence and expert systems - Volume 1 IEA/AIE '88. ACM, 1988.
- [24] Michael A. Arbib. The Handbook of Brain Theory and Neural Networks, Second Edition. MIT Press, 2002.
- [25] OpenCV (Open Computer Vision) es una librería de funciones de apoyo a la programación de aplicaciones de visión artificial. La distribución es de código libre, pudiéndose descargar desde <http://opencv.willowgarage.com/wiki/>. También se puede encontrar en <http://sourceforge.net/projects/opencvlibrary>
- [26] Gary Bradski and Adrian Kaehler. Learning OpenCV. O'Reilly, September 2008.
- [27] QT es un entorno de desarrollo para la construcción de interfaces de usuario. Dispone de distribución de código libre, que se puede descargar desde <http://qt.nokia.com>.
- [28] Alper Yilmaz, Omar Javed y Mubarak Shah. Object Tracking: A Survey. 2006, ACM Computing Surveys, Vol. 38, Nº. 4, Article 13, Publication date: December 2006.
- [29] Plataforma de código libre Eclipse, sostenida por la Fundación Eclipse. <http://www.eclipse.org>
- [30] cvBlobsLib (Blob Extraction Library) es una librería para la detección de blobs inicialmente distribuida entre los ficheros enviados a los foros de OpenCV, posteriormente incluida como componente independiente de OpenCV (tal y como se puede ver en las FAQs <http://opencv.willowgarage.com/wiki/FullOpenCVWiki>). Se puede encontrar en: <http://opencv.willowgarage.com/wiki/cvBlobsLib>
- [31] En el sitio Web de MobileRobots (www.mobilerobots.com) se puede encontrar información detallada sobre el modelo Pioneer 3-AT, en la url: <http://www.mobilerobots.com/ResearchRobots/ResearchRobots/P3AT.aspx>

- [32] Research Artificial Vision Tool. Software de código libre desarrollado como consecuencia de este trabajo de investigación. Se puede descargar desde: <http://artificialvis.sourceforge.net>
- [33] Datasets
- [a] Daimler Monocular Pedestrian Detection Benchmark. Secuencia TestData_part1 de descargada desde: <http://www.lookingatpeople.com/download-daimler-ped-det-benchmark/index.html>
- [b] Las dos secuencias de prueba del Dataset 5 del PETS2001: <ftp://ftp.pets.rdg.ac.uk/pub/PETS2001/DATASET5/TESTING>
- [34] J. Matas, C. Galambos, and J. Kittler. Progressive probabilistic Hough Transform. In Proc. British Machine Vision Conf. BMVC98, Sep. 1998.
- [35] R. Duda and P. Hart. Use of the hough transformation to detect lines and curves in pictures. In Communications of the ACM 15, 1, 11-15, 1972.
- [36] Motilal Agrawal, Kurt Konolige and Morten Rufus Blas. CenSurE: Center Surround Extremas for Realtime Feature Detection and Matching. D. Forsyth, P. Torr, and A. Zisserman (Eds.): ECCV 2008, Part IV, LNCS 5305, pp. 102-115, 2008.
- [37] Mohinder S. Grewal, Angus P. Andrews. Kalman Filtering: Theory and Practice Using MATLAB, Second Edition. 2001 John Wiley & Sons, Inc.
- [38] Huang S., & Dissanayake G. Convergence and consistency analysis for extended Kalman filter based SLAM. (2007) IEEE Transactions on Robotics, 23(5), 1036-1049.
- [39] David Ribas, Pere Ridao, Juan Domingo Tardós and José Neira. Underwater SLAM in Man-Made Structured Environments. Journal of Field Robotics, 1-24 (2008), DOI: 10.1002/rob.20249
- [40] Oleg V. Komogortsev, Javed I. Khan. Eye Movement Prediction by Kalman Filter with Integrated Linear Horizontal Oculomotor Plant Mechanical Model. 2008 ACM 978-1-59593-982-1/08/0003
- [41] James G. Malcolm, Martha E. Shenton and Yogesh Rathi. Two-Tensor Tractography Using a Constrained Filter. G.-Z. Yang et al. (Eds.): MICCAI 2009, Part I, LNCS 5761, pp. 894-902, 2009.
- [42] Rachid Deriche, Jeff Calder, Maxime Descoteaux. Optimal real-time Q-ball imaging using regularized Kalman filtering with incremental orientation sets. 2009 Elsevier B.V. doi:10.1016/j.media.2009.05.008
- [43] Simuladores para juegos de coches de carrera de código libre
- [a] The Open Racing Car Simulator (TORC). Se puede encontrar en <http://torcs.sourceforge.net>
- [b] VDrift. Se puede encontrar en <http://vdrift.net>
- [44] T. Zhao y R. Nevatia. Tracking Multiple Humans in Complex Situations. *PAMI, Vol. 26, No. 9, pp. 1208-1221, 2004.*
- [45] Topografía de obras. Libro de Ignacio del Corral y Manuel de Villena. Eds UPC, 2001.
- [46] D. Murray and J. Little. Using real-time stereo vision for mobile robot navigation. In Proceedings of the IEEE Workshop on Perception for Mobile Agents, Santa Barbara, CA, June 1998.
- [47] Stephen Se, David Lowe and Jim Little. Vision-based mobile robot localization and mapping using scale-invariant features. In International Conference on Robotics and Automation, 2001, Seoul, Korea, pp. 2051-58.
- [48] Bruce D. Lucas and Takeo Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. Proc. DARPA IU Workshop, 1981, pp. 121-130.
- [49] Robotics and automation handbook. Book edited by Thomas R. Kurfess. ISBN 0-8493-1804-1