SIVI: SISTEMA INTELIGENTE MULTI-ROBOT PARA LA VIGILANCIA MÓVIL DE ENTORNOS

UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

INGENIERÍA
EN INFORMÁTICA

PROYECTO FIN DE CARRERA

SIVI: Sistema Inteligente multi-robot para la vigilancia Móvil de Entornos

Roberto Pozuelo Domínguez

Septiembre, 2014

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Departamento de Tecnologías y Sistemas de Información

PROYECTO FIN DE CARRERA

SIVI: Sistema Inteligente multi-robot para la vigilancia Móvil de Entornos

Autor: Roberto Pozuelo Domínguez
Director: Dr. David Vallejo Fernández

Septiembre, 2014

Roberto Pozuelo Domínguez

Ciudad Real – Spain

E-mail: Rovertpd@gmail.com
Teléfono: 669 124 286
Web site:

TRIBUNAL:

Presidente:

Secretario:

Vocal:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	**SECRETARIO**	**VOCAL**
Fdo.:	Fdo.:	Fdo.:

Resumen

Los sistemas de vigilancia están cada día más demandados debido a la necesidad de una mayor seguridad. Hasta hace unos años solo se instalaban sistemas de seguridad en lugares concretos, para preservar de robos, atracos o incendios. Hoy en día se utilizan en hogares, pequeños negocios, fábricas, además de lugares de alto riesgo, como bancos y joyerías. Por ello, este tipo de sistemas deben de ser tolerantes a fallos y no deben producir falsas alarmas, ya que estos sistemas son tan poco eficaces como uno que se pueda vulnerar con facilidad. Además, cada vez son más las empresas que emplean todo tipo de sensores para aumentar la seguridad, para complementar a las cámaras de videovigilancia.

El uso de elementos móviles en los sistemas de vigilancia añade mayor grado de vigilancia en este tipo de sistemas. Gracias a la movilidad que ofrecen este tipo de elementos robóticos se puede reducir los puntos ciegos del sistema y el acceso a zonas que mediante videocámaras convencionales no se podría acceder (como en entornos con muchas pareces y pequeñas zonas cerradas) y sobretodo añade mayor seguridad a los operarios y vigilantes de seguridad al no ser necesaria su presencia física. Además gracias a la inclusión de agentes inteligentes se consigue dotar de mayor autonomía a los sistemas de vigilancia, lo que conlleva una reducción de costes y una mejor vigilancia, gracias a que el procesamiento de imágenes de de los sistemas electrónicos es más rápida y eficaz que el ojo humano. Si además se introducen algoritmos de coordinación se consigue mayor eficacia en el sistema al poder emplear más elementos robóticos al mismo tiempo, consiguiendo una vigilancia más efectiva al poder estar en varios sitios a la vez.

A partir de esto surgió la motivación de crear un sistema que sirviera de base para este tipo de sistemas de vigilancia, multi-agente, dedicados a la vigilancia de entornos creando un mecanismo de comunicación y coordinación básico.

SIVI es un sistema que se apoya en una arquitectura adaptativa, extensible y modular, que hace sencilla la introducción y modificación de las funcionalidades que ofrece, así como nuevos dispositivos móviles, como robots más complejos. Además, SIVI está creado empleando herramientas y estándares libres consiguiendo una fácil portabilidad.

Para probar la eficacia del sistema SIVI, se ha utilizado un sistema a pequeña escala haciendo uso de pequeños robots capaces de moverse por el entorno y, gracias a un sistema de localización y posicionamiento, capaces de vigilar pequeños objetos de colores, donde cada color implica una prioridad de objetivo. Además se ha comprobado que el sistema ofrece una buena visualización del entorno a través de una interfaz de usuario que ofrece la funcionalidad del aumento de la información de robots y objetivos gracias a la realidad aumentada.

Abstract

Surveillance systems are increasingly demanded due to the need for greater security. Until recently only security systems were installed in specific locations, to preserve burglary, robbery or fire. Today are used in homes, small businesses, factories, and high-risk places such as banks and jewelers. Therefore, these systems must be fault tolerant, and must not produce false alarms, as these systems are so inefficient as one that can easily breach. In addition, more and more companies use all kinds of sensors to increase security, to complement the video surveillance cameras.

The use of moving parts in systems adds greater vigilance monitoring in such systems. Thanks to the mobility they offer this type of robotic elements can reduce the blind spots of the system and access to areas by conventional camcorders could not be accessed (as in environments with many walls and small enclosed areas) and above adds more security operators and security guards to his physical presence will not be required. Also thanks to the inclusion of intelligent agents is achieved giving greater autonomy to surveillance systems, which leads to lower costs and better surveillance, thanks to the image processing of electronic systems is faster and more effective than the human eye. If coordination algorithms are also introduce greater efficiency is achieved in the system to be able to employ more robotic elements while getting a more effective monitoring.

From this emerged the motivation to create a system as a basis for this type of surveillance systems, multi-agent, dedicated to creating a monitoring mechanism for communication and basic coordination environments.

SIVI is a system that relies on an adaptive, extensible and modular architecture which makes it easy introduction and modification of features it offers, and new mobile devices, as more complex robots. Furthermore, SIVI is created using free tools and getting easy portability standards.

To test the effectiveness of SIVI system used a small-scale system using small robots able to move around the environment and thanks to a tracking and positioning, capable of monitoring small colored objects, where each color means a priority target. Furthermore it has been found that the system provides good visualization environment through a user interface that provides increased functionality robots information and targets with enhanced reality.

Índice general

Índice de tablas

Índice de figuras

Índice de listados

Listado de acrónimos

CCTV	Circuito Cerrado de TeleVision
PUD	Proceso Unificado de Desarrollo
WPM	Wireless Programming Module
RAD	Desarrollo Rápido de Aplicaciones
ICE	The Internet Communications Engine
RGB	Red Green Blue
HSV	Hue Saturation Value
HSL	Hue, Saturation y Luminance
LGPL	Lesser General Public License
IVS	Sistemas de Videovigilancia Inteligentes
GIS	Sistema de Información Geográfica
RAIOM	Realidad Aumentada para la Identificación de Objetivos Militares

Agradecimientos

Se dice que *'Basta un poco de espíritu aventurero para estar siempre satisfechos, pues en esta vida, gracias a dios, nada sucede como deseábamos, como suponíamos, ni como teníamos previsto'*. Efectivamente esas frases que empiezan por creí, pensé, esperaba siempre acaban por no ocurrir. Pero con un poco se espíritu aventurero la vida se ve de otra manera.

A mi familia en primer lugar, que sin pedir nada a cambio siempre me lo han dado todo y sin ellos nunca habría conseguido acabar lo que empecé. También son los que más me han sufrido durante todo este tiempo, mi estrés y mis cambios de humor. Gracias por vuestro apoyo y vuestros ánimos.

A Javier, Jesús, José y Laura, que gracias a ellos estoy donde estoy y han compartido conmigo todos los momentos importantes de la carrera.

A mis amigos de siempre, de toda la vida, que aunque no estuviéramos en la misma ciudad siempre han estado ahí para apoyarme.

A Alba, nunca olvidaré tus palabras de ánimo y apoyo.

Y por último y más importante a David, mi director de proyecto. No se qué habría hecho sin ti. Gracias por estar ahí día sí y día también.

Roberto

A mis padres y mi hermano que, aunque no se lo diga muy a menudo, les quiero mucho

Capítulo 1

Introducción

En la actualidad la inversión de las empresas en los sistemas de vigilancia está aumentando considerablemente. Las empresas cada vez guardan más objetos de gran valor y por ello cada vez se gastan más en grandes sistemas de vigilancia que garanticen su seguridad. Ya no sólo contratan a un guardia de seguridad, para que vigile las pantallas en las que se muestran los vídeos de las cámaras de seguridad, sino que también se instalan **sistemas más avanzados** como sensores de movimiento o sensores térmicos. También hay que tener en cuenta que los sistemas de vigilancia no sólo se utilizan para proteger los objetos importantes de las empresas, también se utilizan para vigilancia de entornos, como parques nacionales, para vigilar que no haya riesgos para los animales, detectar a cazadores furtivos, etc.

Con todo esto podemos observar que estos sistemas se utilizan mucho en la actualidad, lo que nos lleva a plantearnos una cuestión: ¿Podemos **mejorar la eficiencia** de estos sistemas? La respuesta es sí. Podemos mejorar los sistemas de vigilancia si introducimos elementos automáticos, que ayuden a los humanos en la tarea de vigilancia. Con esto conseguiríamos reducir problemas de **dependencia de los humanos**, puesto que un sistema completamente dependiente de los seres humanos tiene ciertas fisuras en la seguridad. Los principales problemas de esta dependencia respecto al operador humano son la fatiga y el cansancio derivados de la observación continua de los monitores de televisión donde se muestran las imágenes de las cámaras [Smi04], debido sobre todo a la gran cantidad dispositivos de vigilancia que tiene que observar, los cuales normalmente superan las capacidades de los operadores humanos a la hora de monitorizarlos todos. Con un sistema automatizado reduciríamos de forma considerable estos problemas y aumentaríamos la eficacia del sistema de vigilancia.

Una manera de **automatizar** el sistema es mediante la vigilancia inteligente, que se puede entender como la aplicación de técnicas, algoritmos y métodos de Inteligencia Artificial (IA) con el objetivo de desarrollar sistemas de seguridad avanzados que lleven a cabo las tareas de vigilancia y monitorización realizadas tradicionalmente por operadores humanos [VV05b]. En este sentido se hace especialmente importante el estudio de **robots inteligentes** capaces de automatizar esa labor. La implicación del uso de robots para sistemas de vigilancia inteligente supone no sólo una mejora en la seguridad de las empresas, sino también en

la seguridad de los operarios. Podemos ver un ejemplo de esto en entornos de vigilancia de seísmos o de volcanes, en los que los operarios se juegan la vida cada vez que necesitan recopilar datos. Es en esos momentos cuando el uso de un robot, capaz no solo de recopilar información sino también de vigilar posibles cambios, aumentaría la seguridad de los operarios humanos.

En los últimos tiempos se están dejando cada vez más tareas monótonas que tienen que realizar las personas en manos de dispositivos robóticos móviles simplemente porque un humano no podría sin ponerse en riesgo, como el ejemplo anterior de los volcanes y terremotos.

Y aunque la **robótica móvil** está en pleno auge, el componente inteligente en los sistemas multi-robot continúa siendo un reto fascinante y relevante dentro de la comunidad científica y de desarrolladores. En concreto, no sólo se trata de generar un comportamiento específico dependiendo del entorno para un agente (entendido como una entidad software autónoma), sino que disponemos de varios agentes (cada robot sería un agente), por lo que habría que generar un comportamiento de grupo (no se pueden comportar todos los robots de la misma manera). El simple hecho de tener varios **robots coordinados** por un único servidor ya nos presenta el problema concreto de la comunicación y coordinación [Bur00] entre los distintos robots móviles.

Por ello el **siguiente proyecto** trata de establecer mecanismos generales de coordinación entre los robots. El proyecto consiste en crear un sistema de vigilancia inteligente a pequeña escala. Un sistema de vigilancia inteligente mediante robots móviles sería muy interesante para la vigilancia en edificios, con objetos importantes, como pueden ser museos, joyerías, etc. Además al ser un sistema inteligente multi-robot no solo se puede utilizar para la vigilancia móvil, sino que se puede utilizar en otros dominios como puede ser la domótica, puesto que se podría utilizar, por ejemplo, en aplicaciones como la limpieza de viviendas.

1.1 Sistemas de vigilancia

1.1.1 Introducción

Desde tiempos inmemoriales, el ser humano ha estado haciendo uso de sistemas de vigilancia aun sin tener conciencia de ello.

El **proceso de vigilancia** es cualquier proceso que implique la captación de información del exterior de forma organizada, selectiva y durante un período de tiempo. Entonces, ¿por qué decimos que el ser humano ha hecho uso de estos procesos desde hace tanto tiempo? Muy sencillo; el simple hecho de estar pendiente de nuestros hijos ya se considera un proceso de vigilancia.

A pesar de que en francés la palabra vigilancia significa literalmente 'mirar por encima', el término vigilancia suele aplicarse a toda forma de observación o monitorización, no sólo

la observación visual.

Lejos han quedado ya aquellos tiempos en los que la vigilancia se hacía de forma presencial, observando nuestro entorno en busca de cierta información considerada importante por alterar el estado natural de dicho entorno. En la actualidad todo el proceso se hace de forma más automatizada, haciendo uso de las nuevas tecnologías. Y lo más importante es que todavía no se ha llegado al techo. Cada día surgen nuevas herramientas capaces de hacernos más sencillas las tareas de vigilancia.

Y no sólo nos tenemos que fijar en las herramientas utilizadas en los sistemas de vigilancia. Hay que hacer especial hincapié en las necesidades, cada vez mayores, de aumentar el estado de seguridad, puesto que el principal objetivo de los sistemas de vigilancia es el de aumentar la seguridad.

La forma de aumentar la **sensación de seguridad** se está consiguiendo con la mejora de los sistemas de vigilancia. Si no se mejoras las herramientas (o se crean nuevas) a la larga dejarán de ser útiles las actuales herramientas, aunque se aumente su número.

Uno de los métodos de obtención de información más extendidos actualmente es el uso de cámaras de videovigilancia. Sin embargo el uso intensivo de este tipo de herramientas puede convertirse en un problema en otra etapa importante del proceso de vigilancia, en este caso el análisis de la información obtenida para seleccionar la información relevante.

En la mayoría de grandes instalaciones con cámaras Circuito Cerrado de TeleVision (CCTV), con cientos de ellas, sólo una pequeña fracción de las mismas se vigilan. Este hecho se debe fundamentalmente a dos factores. Por una parte, el número de operarios es reducido con respecto al número de cámaras y, por otra, los flujos de vídeo almacenados por determinadas cámaras sólo se revisan en caso de que haya ocurrido algún tipo de incidente. Este último fenómeno se suele denominar análisis forense . Por lo tanto, la conclusión es que aunque en teoría todas las cámaras se monitoricen de alguna manera, aunque sea como parte de un procedimiento de prueba, sólo un pequeño porcentaje se hace en tiempo real.

Otro de los motivos por el que no se consigue un alto porcentaje vigilancia en tiempo real en estos sistemas superpoblados es la cuestión relacionada con la dependencia respecto a los operadores humanos. Los seres humanos no son máquinas, con lo que siempre va a aparecer el tema de la fatiga y el cansancio. Estos síntomas son derivados de la observación continua de este tipo de dispositivos, así como de la necesidad de tratar con varios problemas simultáneos cuando se dé una situación caótica.

Todo esto es lo que nos está llevando de manera incesante e imparable a desarrollar sistemas de vigilancia cada vez más automáticos, más tecnológicos, que nos permitan reducir los problemas derivados de los operarios humanos y conseguir un mayor período de tiempo de vigilancia en tiempo real.

Es el momento de un cambio. Es el momento de la **vigilancia inteligente**. Es el momento en el que se empiezan a desarrollar sistemas de vigilancia inteligentes que automaticen todo el proceso. Sin embargo estos sistemas de vigilancia deben cumplir otras características, que son las que se exponen a continuación [Fer09]:

- **Escalabilidad**, con el objetivo de que el sistema de vigilancia pueda expandirse en función de los requisitos del entorno a monitorizar.
- **Portabilidad**, para permitir que el sistema pueda adaptarse a distintos entornos y condiciones de funcionamiento.
- **Modularidad**, que permite diferenciar claramente qué partes del sistema proporcionan cada uno de los servicios del sistema de vigilancia, al mismo tiempo que se garantiza la interoperabilidad de los mismos.
- **Robustez**, para que el sistema sea tolerante a fallos y siga funcionando cuando se produzca cualquier tipo de error en la infraestructura que le da soporte.

1.1.2 Evolución

Seguro que todo el mundo conoce los sistemas de vigilancia de hoy en día. ¿Quién no ha visto una videocámara en lo alto de una esquina en un centro comercial? o ¿quién no ha oído hablar de los sensores de movimiento instalados en los bancos y las casas para localizar a los ladrones?

Los **primeros sistemas de vigilancia** se podría decir que eran los que integraban los propios vigilantes de seguridad. Ellos eran los encargados de observar el perímetro y avisar si de las situaciones adversas con las que se encontraban. Evidentemente, eso tenía muchos problemas de fiabilidad, entre otros los muchos puntos ciegos que dejaban cada vez que salían a hacer la guardia.

Con la aparición de la **telefonía** se empezó a extender el uso de herramientas para hacer escuchas telefónicas y sentar las bases de lo que más tarde serían las escuchas secretas a través de micrófonos ocultos. Este tipo de sistemas de vigilancia en particular tuvieron un uso muy importante en los sistemas de espionaje.

Pero, sin embargo, uno de los pasos más grandes que se han dado en la evolución de los sistemas de vigilancia ha sido la creación de las videocámaras. Si se dijera que los primeros sistemas de videovigilancia datan de 1965 seguramente nadie se lo creería. Pero así es; se han encontrado informes de prensa que indican que la policía de los Estados Unidos habrían empezado a utilizar estos sistemas videovigilancia en lugares públicos en 1965. Sin embargo no fue hasta 1969, cuando se empezaron a instalar estos sistemas en el edificio municipal de Nueva York, que sentó un fuerte precedente y se empezó a extender a otras ciudades y los agentes de policía vigilaban de cerca en áreas clave, con el uso de CCTV.

Sistemas de circuito cerrado de TV analógicos usando VCR	
4*Características	Cintas de una grabadora doméstica
	No se comprime el vídeo
	Máximo de 8 horas de grabación
	Es necesario un monitor analógico
Sistemas de circuito cerrado de TV analógicos usando DVR	
2*Ventajas	No es necesario cambiar las cintas
	Calidad de imagen constante
Sistemas de circuito cerrado de TV analógicos usando DVR de red	
2*Ventajas	Monitorización remota de vídeo a través de un PC
	Funcionamiento remoto del sistema
Sistemas de vídeo IP que utilizan servidores de vídeo	
4*Ventajas	Utilización de red estándar y hardware de servidor de PC para la grabación y gestión de vídeo
	El sistema es escalable en ampliaciones de una cámara cada vez
	Es posible la grabación fuera de las instalaciones
	Preparado para el futuro, ya que este sistema puede ampliarse fácilmente incorporando cámaras IP
Sistemas de vídeo IP que utilizan cámaras IP	
5*Ventajas	Cámaras de alta resolución (megapíxel)
	Calidad de imagen constante
	Alimentación eléctrica a través de Ethernet y funcionalidad inalámbrica
	Funciones de Pan/tilt/zoom, audio, entradas y salidas digitales a través de IP, junto con el vídeo
	Flexibilidad y escalabilidad completas

Tabla 1.1: Evolución de la videovigilancia desde el punto de vista tecnológico.

Podemos ver un pequeño resumen de la evolución de los sistemas de vigilancia mediante videocámaras en el Cuadro 1.1:

El siguiente paso en la evolución es la **vigilancia móvil**. Es un paso más a la hora de poder vigilar entornos dejando el menor número de puntos muertos. Estos nuevos sistemas de vigilancia incorporan elementos robóticos que dotan a los dispositivos de vigilancia (como pueden ser las cámaras CCTV) de cierta movilidad, lo que permite obtener un grado de visión casi completo. En el siguiente punto se desarrollará esta primera aproximación a los sistemas móviles.

1.1.3 Vigilancia móvil

En un mundo en el que te encuentras tecnología en cada esquina, en cada persona, en cada edificio, se hace indispensable la evolución tecnológica de las herramientas. Y ya no sólo

Vigilancia fija	Vigilancia móvil
Lugares o puntos determinados	Sitios o áreas de grandes dimensiones
Posición fija	Acceso a cualquier posición
Depende de un operario	No es necesario un operario
Más popular	Cada vez más conocida
En declive	En constante crecimiento
Coste bajo	Elevado coste

Tabla 1.2: Vigilancia fija vs Vigilancia móvil.

basta con evolucionar esas herramientas, sino con dar un paso más y cambiar la manera de hacer las cosas.

En los sistemas de vigilancia, uno de los cambios más importante se dio con la aparición de la vigilancia móvil. Se empezaron a utilizar sistemas robotizados, que permitían un ligero movimiento al principio y una movilidad casi total a día de hoy.

Si nos fijamos en la tabla 1.2 podemos ver una pequeña comparación entre los sistemas de vigilancia fija y los sistemas de vigilancia móvil. A pesar de que los sistemas de vigilancia móvil nos pueden reportar una gran cantidad de ventajas y beneficios, en muchas ocasiones su elevado coste lleva a las empresas a utilizar las cámaras de vigilancia fija.

Este tipo de sistemas son muy importantes en situaciones de riesgo para el ser humano. En los entornos comprometidos para el ser humano, en los que la simple ubicación de una persona implica una exposición a un peligro tanto para el operarios como para el resto de personas cercanas. Es en esas situaciones en las que cobran mayor importancia los sistemas de vigilancia móviles, en las que se podrían introducir robots en dicho entorno capaces de obtener información mediante cámaras y sensores sin poner en peligro a nadie.

Pongamos como ejemplo el caso de un volcán activo. Cuando encontramos un volcán activo, se hace necesario hacer un seguimiento de dicho volcán para evitar que entre en erupción de forma inesperada, con el peligro que ello conlleva para las poblaciones adyacentes. Para realizar dicho seguimiento habría que estar midiendo constantemente la temperatura en el cráter, además de medir la composición química de los gases. Para una persona sería un peligro entrar en cráter constantemente para hacer estas mediciones, sin contar con la alta temperatura que hay en el interior y los gases tóxicos. Sin embargo, se podría mandar una unidad robótica cargada con sensores y cámaras dirigida por un operario. Esa unidad robótica sería capaz de llegar muy profundo en el cráter sin ningún riesgo, y hacer las mediciones necesarias a través de los sensores. También se podría hacer un seguimiento parcial a través de satélites para monitorizar el aumento de la actividad fumarólica.

Otro ejemplo lo podemos encontrar en las misiones espaciales. En este tipo de misiones se hace muy necesaria la utilización de sistema de vigilancia móviles, puesto que no se podría

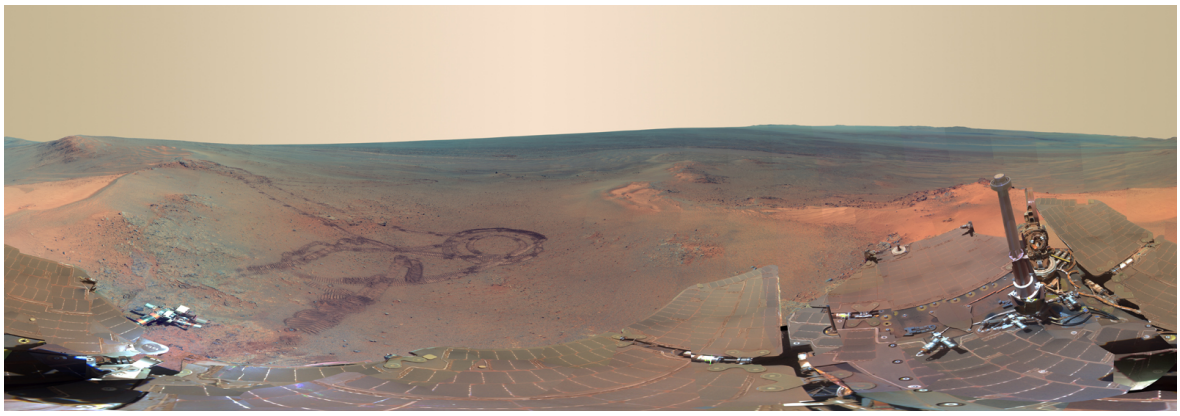

Figura 1.1: Combinación de imágenes (817) tomadas por la Mars Rover Opportunity entre el Diciembre de 2011 y Mayo de 2012.

asegurar la integridad personal. Podemos poner como ejemplo especifico la utilización de los robots Spirit y Opportunity, que forman parte del Programa de Exploración de Marte de la NASA. Gracias a estos sistemas de vigilancia móvil se ha conseguido información muy importante sin ser necesaria la presencia del ser humano en un territorio totalmente desfavorable. Entre los éxitos cosechados por estos robots podemos destacar:

- Evidencia de antiguos manantiales en ebullición.
- Evidencia de una atmósfera densa y de agua dulce
- Evidencia de un ciclo activo de agua

Así pues, podemos deducir que la inclusión de estos nuevos sistemas de vigilancia otorgan otro aire a los anticuados sistemas basados en cámaras fijas, totalmente dependientes de la posición en la que están situadas y con área de observación mucho más reducida.

Sin embargo, hemos observado como en la mayoría de los casos, estos sistemas se utilizan en proyectos muy ambiciosos. Esto se debe a la complejidad que acarrean. Al ser sistemas independientes de la posición pero dependientes de sistemas robóticos que doten de movilidad los antiguos sistemas nos encontramos con el consiguiente aumento de costes y de complejidad a la hora de manejar dichos sistemas. A continuación entraremos en el detalle de esta problemática.

1.2 Problemática

Cuando aparece la idea de diseñar un sistema de vigilancia móvil el principal problema a paliar es el de hacer un buen diseño de la inteligencia artificial que tendrá dicho sistema, sobre todo la parte de que afecta a los sistemas móviles (robots). En este aspecto hay que tener muy presente los aspectos más importantes como son los cálculos de trayectorias, el seguimiento y corrección de trayectorias de los robots, y en el caso de sistemas multi-robots se hace indispensable una buena coordinación entre ellos, esto es, la creación de los mecanismos necesarios para que los robots se comuniquen y se coordinen entre ellos.

Tenemos que tener en cuenta que los robots móviles que operan en grandes entornos siempre deben enfrentarse con un gran **grado de incertidumbre** en cuanto a la identificación de objetos y a la posición (tanto de los objetos como de los propios robots). El grado de incertidumbre es tal que lo que para un operador humano o un manipulador robótico industrial seria una actividad relativamente trivial, como es trasladarse desde un punto A hasta un punto B, para un robot móvil es una actividad arriesgada, no sabe con qué se puede encontrar. Así pues, como resultado de tener que enfrentarse con más incertidumbres del entorno, no se espera que un robot móvil siga trayectorias o alcance su destino final con el mismo nivel de precisión que se espera de un manipulador industrial (en el orden de las centésimas de milímetro).

El principal problema a resolver en un robot móvil es generar trayectorias y guiar su movimiento según éstas, en base a la información proveniente del sistema de sensores externos (ultrasonidos, infrarrojos, videocámaras), permitiendo al vehículo desplazarse entre dos puntos cualesquiera del ambiente de trabajo de manera segura, sin colisiones. Esto exige diseñar sistemas de control de trayectorias (posición, dirección, velocidad) en diversos niveles jerárquicos, de manera que el procesamiento de la información proveniente de los sensores externos asegure la mayor autonomía posible[Bam08].

Para hacer un sistema de vigilancia móvil realmente eficiente es conveniente la utilización de robots móviles inteligentes o autónomos.

Los **robots móviles autónomos** se caracterizan por una conexión inteligente entre las operaciones de percepción y acción, que define su comportamiento y le permite llegar a la consecución de los objetivos programados sobre entornos con cierta incertidumbre. El grado de autonomía depende en gran medida de la facultad del robot para abstraer el entorno y convertir la información obtenida en órdenes, de tal modo que, aplicadas sobre los actuadores del sistema de locomoción, garantice la realización eficaz de su tarea. Las características principales de estos robots son la **percepción** y el **razonamiento**.

Al tratar con este tipo de robots tenemos que tener presente que son muy influenciables por el entorno, son totalmente dependientes del entorno, y se pueden enfrentar a situaciones insospechadas, como se indicaba antes debido sobretodo a la incertidumbre. Pero también nos podemos encontrar ante problemas mecánicos, como pueden ser fallos en los motores, problemas de deslizamiento o patinaje.

Si se quiere diseñar un buen sistema de vigilancia se hace necesario el uso de varios robots inteligentes, para minimizar el número de puntos ciegos. Sin embargo esto plantea otro gran problema que es el de la comunicación entre ellos, la coordinación.

El tema de la coordinación entre robots genera la duda de cuál sería la mejor forma de que estos sistemas autónomos se comuniquen entre sí. Se podría utilizar la forma más sencilla, que sería programar a los robots para que, mediante los sensores disponibles en los propios

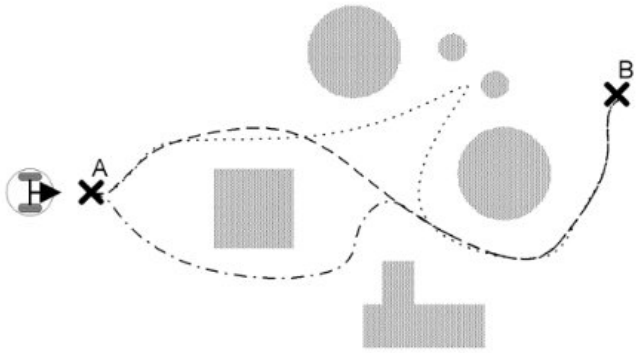

Figura 1.2: Posibles trayectorias a seguir por un robot móvil.

robots, detectarán la posición de los demás elementos del entorno y en caso de una posible colisión se detuviera.

Pero este planteamiento no nos asegura un buen sistema de vigilancia, puesto que realmente no hay una comunicación entre ellos, no se coordinan, con lo que se daría el caso de que varios robots fueran al mismo punto a vigilar, o que se quedaran sin vigilar los elementos más importantes.

Una manera de conseguir un sistema de vigilancia móvil mediante uso de robots inteligentes sería con la creación de un sistema central que, a través de toda la información obtenida del entorno, sea capaz de tomar decisiones y mandar señales a los robots para que se muevan. Con esto se consigue una base de coordinación entre los robots y se establecen las comunicaciones necesarias entre ellos.

Este sistema central debe de saber, en todo momento, toda la información relativa al entorno en el que se trabaja. Y no solo la información del entorno, sino también la información de los elementos que vigilaran ese entorno, que en este caso son los robots.

Además debe de ser un sistema fornido y tolerante a fallos puesto que en este tipo de sistemas de vigilancia automatizados un fallo que nos provoque un paro en la vigilancia puede ser fatal.

También hay que tener en cuenta a la hora de definir los medios de coordinación el entorno en el que se va a localizar el sistema, puesto que en entorno con un gran alto grado de incertidumbre se antoja un trabajo arduo el poder coordinar los elementos robóticos desde un sistema central. Además se puede hacer difícil la comunicación en entornos con grandes bloques de hormigón, los cuales dificultan la comunicación inalámbrica.

1.3 Propuesta de Proyecto Fin de Carrera

Una vez que se ha expuesto la problemática de este tipo de sistemas, pasaremos a detallar la propuesta de Proyecto Fin de Carrera.

Como hemos visto, los sistemas de vigilancia móvil son muy complejos, y no siempre son fáciles de conseguir ni baratos. Lo que se propone con este trabajo es la creación de un sistema de vigilancia móvil que mediante el uso de una sola videocámara sea capaz de construir un sistema inteligente de robots que colaboren entre sí.

Se trata de un sistema de vigilancia móvil cuya características principales son la completa escalabilidad y su modularización. Esta modularización nos permitirá añadirle nuevas funcionalidades con el mínimo cambio que, añadido al algo grado de escalabilidad, nos permitirá crear un sistema con un gran abanico de posibilidades.

El sistema se encargará de la vigilancia de un determinado entorno mediante una serie de robots. El sistema tratará de de identificar una serie de objetos que, dependiendo del color que tengan, tendrán una determinada prioridad. Mediante una serie de algoritmos de coordinación, el sistema será capaz de mantener el mayor número posible de objetos vigilados.

El grado de escalabilidad del proyecto hace que sea adaptativo tanto al número de robots como al número de objetos a vigilar. Si en algún momento se queda algún robot ocioso por no tener nada que vigilar, este se mantendrá dando vueltas al espacio, como forma alternativa de vigilancia, mientras espera la aparición de otro elemento.

A través de una cámara cenital se hará la fase de obtención de la información. Será la encargada de obtener toda la información referente al entorno, como son toda la información de los objetos a vigilar, así como toda la información referente a los robots y los límites espaciales.

Cada robot, independientemente del número de estos, se moverá de forma autónoma dirigidos por el sistema central, que alberga toda la información del espacio. Se hace necesaria la creación de un coordinador que, a partir de la información, sea capaz de tomar decisiones y de coordinar a los robots. Será el encargado de priorizar los objetos a vigilar, así como de decidir en cada momento cuál es el mejor robot (tanto si está ocioso o, en caso de estar todos ocupados, el que esté vigilando un objeto con menor prioridad) y de coordinar los robots para evitar colisiones entre ellos.

A todo esto, hay que añadir una interfaz en la que se muestra en todo momento, mediante realidad aumentada, información relativa a los objetos (como la posición o la prioridad), a los robots (como el ángulo de visión) y de los límites del entorno.

1.4 Estructura del resto del documento

La estructura que se sigue en el presente documento se ajusta la normativa vigente para la realización del Proyecto Fin de Carrera en la titulación de Ingeniería Superior Informática por la Escuela Superior de Informática de Ciudad Real, aprobada en Junta de Centro el 8 de noviembre de 2007 y modificada en Junta de Centro el 1 de Marzo de 2013.

Capítulo 1: Introducción

Se expone una breve introducción del área en la que se enmarca el presente Proyecto Fin de Carrera. Además, se detalla la problemática que supone el desarrollo de un sistema inteligente dentro de los sistemas de vigilancia multi-robot, así como la propuesta de Proyecto Fin de Carrera.

Capítulo 2: Objetivos

Se describen los objetivos planteados para el presente Proyecto Fin de Carrera. En este capítulo se describe el objetivo general del sistema, así como, los objetivos específicos que deberá resolver el sistema desarrollado.

Capítulo 3: Antecedentes

Se realiza un estudio de las áreas de conocimiento necesarias para entender y desarrollar este proyecto.

Capítulo 4: Método de trabajo

Se recoge y expone la metodología utilizada, así como el software y el hardware necesario para la construcción de este proyecto.

Capítulo 5: Arquitectura

se describe la arquitectura del sistema, analizando la funcionalidad de cada uno de los submódulos en los que se divide la arquitectura.

Capítulo 6: Evolución, Resultados y Costes

Se detalla la evolución del proyecto, defiendo las iteraciones necesaria para el desarrollo del mismo. También, se exponen y contrastan los resultados obtenidos por el sistema. Por último, se detalla una estimación del coste del proyecto.

Capítulo 7: Conclusiones y Propuestas

En este capítulo se realiza una valoración general tanto del desarrollo del sistema, como de los resultados obtenidos. Se exponen las posibles líneas de trabajo futuro en el sistema. El capítulo concluye con una valoración personal acerca de lo que ha supuesto el desarrollo del proyecto.

Anexos

En el apartados de anexos se incluyen un manual de usuario, un manual de desarrollo y una parte final con las imágenes de SIVI en color.

Capítulo 2

Objetivos

La finalidad del presente proyecto es la de crear una sistema que ayude a mejorar el proceso de vigilancia. Para ello se introducirán robots móviles que aumenten la movilidad del sistema y los rangos de vigilancia, así como dotar de cierta autonomía al sistema mediante el uso de algoritmos de inteligencia artificial. Además crearán mecanismos de coordinación básicos que mejoren el rendimiento de los sistemas de vigilancia cuando sean multi-robots, consiguiendo evitar situaciones de peligro como posibles colisiones y espacios sin vigilar.

2.1 Objetivo general

El objetivo principal del proyecto Sivi es el de mejorar los sistemas de vigilancia actuales creando un sistema base a partir de una serie de robots móviles que mediante el uso de algoritmos de inteligencia artificial establezcan un protocolo de coordinación y comunicación para aumentar el grado de autonomía y escalabilidad.

Teniendo en cuenta que en la actualidad los sistemas de vigilancia se encuentran muy demandados, debido a la creciente inseguridad ciudadana y al aumento de los robos en establecimientos, podemos entender que cada vez se apueste más por mejorar estos sistemas. La gente piensa que simplemente aumentando el número de unidades se mejora, pero no siempre es correcto. En el caso de sistemas de vigilancia basados en robots móviles el aumento de de las unidades, en este caso de robots, aumenta la complejidad del problema, puesto que ya no solo existe comunicación con un solo robot, sino que existe esa misma comunicación con varios robots (que debería ser en paralelo para que el sistema fuera lo suficientemente autónomo y eficaz) y además tiene que existir una comunicación entre los robots que los coordine entre sé, no solo para mejorar el sistema sino para evitar fallos como posibles colisiones.

Por este motivo surgió la idea de crear SIVI, un sistema que implementara mecanismos básicos de coordinación entre entidades robóticas, con motivo de ayudar a desarrollar sistemas multi-robots para la vigilancia de entornos. Estos mecanismos se encargarán de monitorizar la situación de los robots y de realizar las comunicaciones pertinentes con los robots para conseguir que éstos se comporten de la manera esperada.

Además, al tratar con entidades físicas que además dependen de baterías, el grado de incertidumbre del sistema se agrava por el hecho de no conocer con exactitud cuál va a ser el comportamiento de estas entidades, puesto que al moverse mediante motores puede haber una gran diferencia entre la potencia de los motores con las baterías llenas y la potencia de las baterías muy desgastadas.

Por tanto el presente proyecto deberá analizar en todo momento el comportamiento de las entidades presentes, en nuestro caso robots móviles, comprobando si se comportan de la manera esperada o no. Teniendo en cuenta esto, se hace necesario que el sistema sea capaz de controlar los robots y, en caso de que no se comporten como se esperaba, ser capaz de mandarle las ordenes precisas para que vuelva a un estado controlado, es decir, a un comportamiento esperado.

Puesto que se trata de un sistema de vigilancia, SIVI deberá poseer mecanismos por los cuales se obtenga la información del entorno. Esta información se guardará en el sistema de modo que éste tenga una foto del entorno con toda la información necesaria en todo momento. Por consiguiente se deberá construir una sistema que permita obtener información útil del entorno en forma de objetos y 'vigilantes'.

Desde el punto de vista de la visualización, el usuario debe poder tener visibilidad del entorno y tener la alternativa de seleccionar un elemento de dicho entorno y obtener información del elemento en cuestión en tiempo real. Esta información será información útil mostrada mediante herramientas de realidad aumentada y superposición de elementos en vídeo. Además dicha información servirá al operador del sistema de vigilancia como ayuda en su labor, puesto que contiene información relacionada con prioridades de los objetivos, ángulos de visión de los robots, etc..

En resumen, en el presente proyecto el principal objetivo es el de crear un sistema que integre tanto una sistema de coordinación entre las entidades, como la funcionalidad necesaria para dotar de cierta inteligencia autónoma a los robots, como el sistema de vigilancia íntegro capaz de ser adaptarse a cualquier entorno.

2.2 Objetivos específicos

A continuación se exponen los distintos objetivos específicos que se pretenden alcanzar con la finalización del Proyecto Fin de Carrera denominado SIVI.

- **Mejorar la monitorización de entornos** a través de la creación de un sistema inteligente multi-robot.
- **Aumentar la coordinación** entre elementos robóticos para eliminar posibles situaciones de peligro, como pudieran ser las colisiones y eliminar irregularidades en los sistemas de vigilancia multi-robot (varios robots vigilando la misma zona, objetos sin vigilancia habiendo robots ociosos, etc.) mediante la creación de un sistema central

capaz de coordinar los elementos móviles del sistema, los elementos vigilantes, de manera independiente.

- Conseguir un alto grado de **modularidad** que aumente la independencia del dominio y nos permita aplicar el sistema en otro entorno distinto con el mínimo esfuerzo, teniendo que modificar una parte mínima relativa a la lógica.
- Crear un sistema un **escalable** que permita añadir nuevas funcionalidades y módulos sin que suponga cambios en la arquitectura.
- Mejorar la **visualización** de los sistemas de vigilancia convencionales mediante la creación de un sistema de información visual basado en realidad aumentada.
- Favorecer la **flexibilidad**, **modularidad**, **escalabilidad**, **adaptabilidad** y posterior **mantenimiento** del proyecto mediante el uso de diversos patrones de diseño.
- Incrementar las **posibilidades de utilización** en diversas plataformas con el uso de estándares y tecnologías multi-plataforma.

Capítulo 3

Antecedentes

En este capítulo se realiza un estudio del estado del arte de aquellas áreas de trabajo vinculadas directamente con la realización del presente Proyecto Fin de Carrera. Primero se muestran los estudios de las áreas independientes y en el apartado final se hace un estudio de la aplicación de estas áreas de trabajo en los sistemas de vigilancia.

3.1 Visión artificial

La **visión artificial** o visión computacional es una disciplina que tiene como finalidad la extracción automática de información del mundo real a partir de imágenes, utilizando como herramienta un ordenador. Abarca muchos y muy diversos usos, como pueden ser la detección y reconocimiento de objetos, evaluación de resultados, mapeo de imágenes...etc.

Desde un punto de vista técnico, un sistema de visión computacional o artificial es un conjunto de elementos que permiten obtener imágenes del entorno, procesarlas y tomar ciertas decisiones basadas en la evaluación de las imágenes adquiridas. Es decir, un **sistema autónomo** que sea capaz de realizar alguna de las tareas de visión humana.

En la actualidad la visión artificial se ha convertido en una herramienta utilizada en gran parte de trabajos de inspección industrial y de vigilancia en general; pues, lo que busca es determinar en el menor tiempo posible, que grandes bloques de producción cumplan estándares fijos en los que se disminuya en lo posible los errores de tipo humano.

En la búsqueda de la automatización de estos problemas, se encuentran dos campos distintos que son, la visión computacional y la visión artificial. La visión computacional surgió con la idea de conectar una cámara de vídeo a un computador, para que las imágenes capturadas fueran interpretadas y así obtener una representación simbólica de los objetos presentes en las escenas analizadas [MF82]. Esto con el propósito de brindar al sistema de visión, ver y entender el mundo en tiempo real, sin ninguna intervención humana. Mientras que la visión artificial cumple la misma función pero adaptándose a las limitaciones tecnológicas que impiden imitar fielmente la visión humana.

A continuación se entra en el detalle de lo que son los sistemas de visión artificial y el procesamiento de digital de imágenes.

3.1.1 Sistemas de visión artificial

Un sistema de visión artificial puede ser definido como un sistema que permite adquirir y analizar imágenes de forma automática, en las que se encuentran los datos necesarios para controlar un proceso o actividad definida con anterioridad. La finalidad de éste sistema es imitar en lo posible el sistema de visión humana [BW97].

Para considerar que un sistema sea de visión artificial debe cumplir con las siguientes tareas:

- **Realizar captura y análisis de imágenes**, dado que el sistema debe realizar estas actividades en conjunto de tal forma que el grado de procesamiento dependerá en gran medida de la calidad de la información adquirida.
- **Debe tener definido el problema previamente**, es decir, establecer las variables que se desean analizar o cuantificar dentro de la solución del problema.
- El sistema **debe estar capacitado para obtener los datos de interés**, ya que dentro de su implementación se deben definir algoritmos que delimiten los datos de tal forma que el costo computacional sea reducido.
- Por último, el sistema **debe ser automático**, lo que implica que el sistema de visión actúa de forma autónoma al momento de realizar los análisis de la información presente en las imágenes obtenidas.

3.1.2 Imagen digital

La vista es nuestro sentido más avanzado, y no es sorprendente que las imágenes jueguen el papel más importante en la percepción humana.

Aunque los seres humanos estemos limitados a la banda visible del espectro electromagnético, las máquinas pueden percibir casi el espectro completo, desde los rayos gamma a las ondas de radio.

Las máquinas también pueden procesar imágenes generadas por fuentes que los seres humanos no asociamos con imágenes, como es el caso del ultrasonido.

Las imágenes captadas por las máquinas se denominan imágenes digitales.

Una **imagen digital** es la proyección en dos dimensiones de una escena tridimensional. La reducción de una dimensión supone la pérdida de gran cantidad de información lo cual dificulta el proceso de visión. Este es el principal inconveniente que nos encontramos al intentar reproducir el resultado de la visión humana a través de un sistema de percepción electrónico.

La representación de la información de una imagen se puede realizar de varias formas pero independientemente de su formato, se representa por una matriz de puntos (denominados pixels).

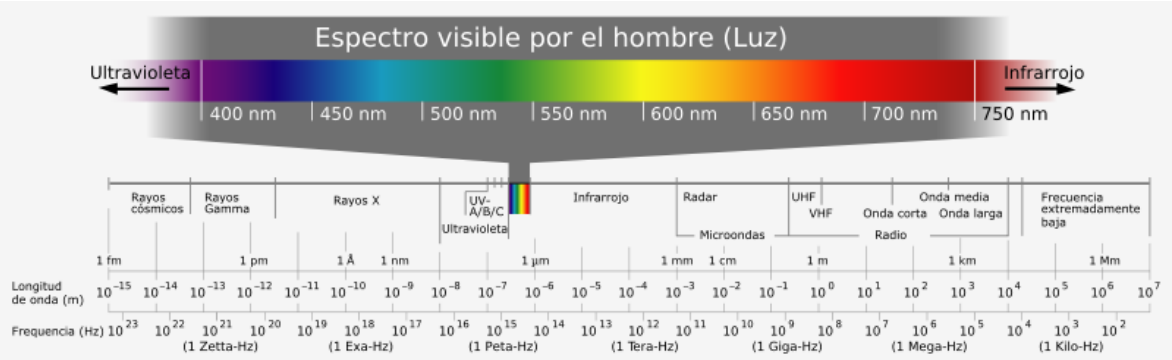

Figura 3.1: Espectro de percepción[1].

Las imágenes se pueden clasificar en 3 tipos: Imágenes Binarias, Imágenes en Escala de Grises e Imágenes a Color. La distinción entre estas lo hace posible la cantidad de colores que pueden ser representados por cada pixel, obteniendo una **imagen binaria** cuando a cada uno de sus pixels se le puede asignar solamente 2 colores: blanco o negro. Para las **imágenes en escala de grises**, cada pixel toma un valor dentro de una gama de grises que van desde el blanco hasta el negro. La cantidad de valores puede variar de acuerdo al número de bits asignado para el almacenamiento del valor del pixel. Para las **imágenes a color**, cada pixel varia en una gama más amplia de tonos, producto de la combinación de ciertos valores. Cada pixel de una imagen a color es definido como un vector, donde cada elemento del vector indica el aporte de una tonalidad.

El color de un objeto es dado físicamente por la reflexión de longitudes de ondas en el espectro electromagnético que comprende la luz.

El espectro de la luz visible por el ojo humano se compone de longitudes de onda entre 400 nm (violeta) y 700 nm (rojo) como se muestra en la figura 3.1.

Las imágenes a color son imágenes compuestas por cuatro valores almacenados en tres planos de pixels o matrices, que se encuentran definidos por los modelos de color, entre los que se encuentran: el modelo Red Green Blue (RGB), el modelo Hue, Saturation y Luminance (HSL) o el modelo Hue Saturation Value (HSV) [PlC01].

En el modelo RGB cada color aparece en sus componentes espectrales primarias rojo, verde, azul. La razón por la que se toman estos tres colores tiene que ver con la **teoría tricromática de color** que indica que tres estímulos cromáticos diferentes, combinados en diferentes proporciones, pueden producir el efecto equivalente a muchos más estímulos que una persona percibirá como muchos colores diferentes.

Este modelo está ubicado en el sistema de coordenadas cartesianas positivas del espacio tridimensional. El sub-espacio de color de interés es el tetraedro mostrado en la figura 3.2, en el que los valores RGB están en tres vértices; mientras que el cyan, magenta y amarillo

[1] https://youcanalso.wordpress.com/tag/percepcion/

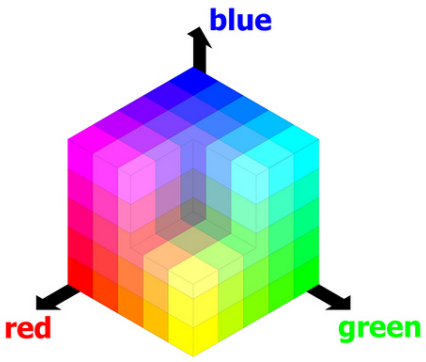

Figura 3.2: Modelo RGB[2].

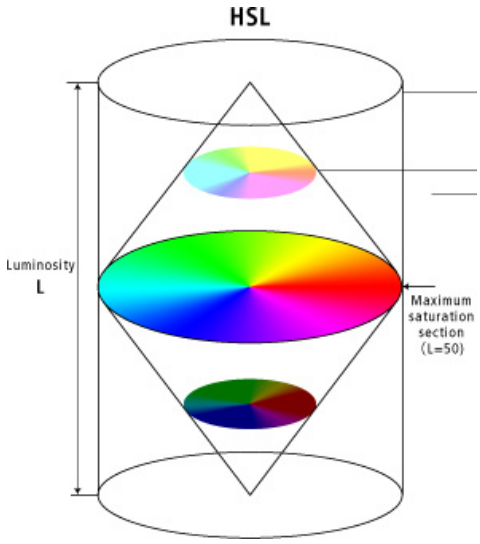

Figura 3.3: Modelo HSL[3].

se sitúan en otros tres vértices, el negro corresponde al origen y el blanco en el vértice más alejado del origen. En este modelo, los niveles de gris se extienden desde el negro hasta el blanco a lo largo de la diagonal que une estos dos puntos y los colores son vectores normalizados definidos desde el origen pajares.

En el modelo de color HSL ó HSI, que se representa como un doble cono, los ápices corresponden al blanco y el negro, los parámetros angulares son los colores, la distancia desde el eje a las esquinas del cono corresponde a la saturación y la distancia a lo largo del eje blanco-negro corresponde a la luminosidad (ver figura 3.3).

Sin embargo en el modelo de color HSV ó HSB, la intensidad varía de negro a blanco en un único cono, en el que se puede identificar tres parámetros que son color, saturación y brillo (ver figura 3.4).

[2]https://www.flickr.com/photos/ethanhein/3103830956/?rb=1
[3]http://www.canon.co.jp/imaging/picturestyle/editor/matters05.html

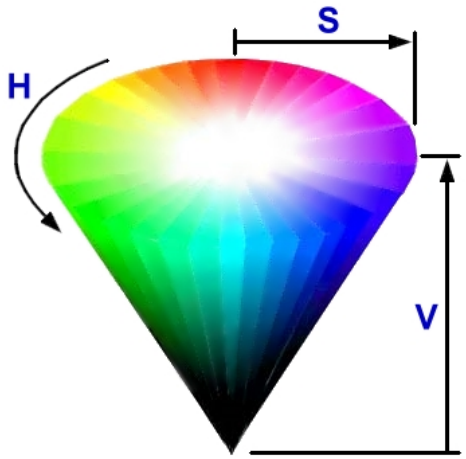

Figura 3.4: Modelo HSV[4].

3.1.3 Procesamiento digital de imágenes

Una vez identificadas las propiedades de las imágenes digitales, aparecen las distintas metodologías existentes para el **procesamiento digital** de dichas imágenes. Existen tres tipos:

- **Basadas en puntos de la imagen**: En esta técnica, cada pixel de la imagen de salida es producto de una operación realizada sobre el pixel respectivo de la imagen de entrada.

  Para la determinación de cada uno de los pixels de la imagen procesada, solo es necesario conocer el valor del pixel original y efectuar la operación deseada.

- **Basadas en la imagen global**: La operación realizada para la determinación de cada pixel de salida necesitará el valor de todos los pixels de la imagen. Cada pixel de la imagen procesada es función de todos los valores de los pixels de la imagen original.

- **Basadas en región de la imagen**: En esta técnica se necesita conocer el valor de los puntos de la región alrededor del pixel en estudio, para luego aplicar la operación determinada y obtener el valor pixel de salida. Cada punto de la imagen de salida es función de su valor en la imagen original y de la región circundante a él. A estos pixels se les denomina vecinos y el tamaño de la región (ventana) que los contiene varia dependiendo de la operación que se desea llevar a cabo.

3.2 Realidad aumentada

Es el término que se usa para definir una visión directa o indirecta de un entorno físico del mundo real, cuyos elementos se combinan con elementos virtuales para la creación de una realidad mixta en tiempo real. Es decir, la tecnología de **Realidad Aumentada** nos presenta la realidad y le añade información artificial.

[4]http://www.ayunor.com/hsv-color-model/

Generalmente esto es una captura de imágenes por un dispositivo de vídeo (ej.: una web-cam) y, tras varios cálculos, la implantación sobre la imagen de cierta información que puede ir desde figuras 2D y 3D, una secuencia animada de vídeo, texto estático o dinámico, etc.

Idealmente, el usuario debe tener la sensación de que los objetos virtuales y los objetos reales **coexisten en el mismo espacio**, es decir, sin distinguir la diferencia entre los objetos reales y los virtuales.

Los sistemas de Realidad Aumentada requieren el uso de unas tecnologías y dispositivos de visualización para aprovechar al máximo sus posibilidades. Una definición de realidad aumentada comúnmente aceptada es la siguiente, un sistema de Realidad Aumentada cumple [Azu97]:

- **Combinación de imagen real y virtual.**
- **Interacción en tiempo real.**
- **Localización 3D.**

A diferencia de la Realidad Virtual, que sumerge al usuario en un ambiente completamente artificial, la Realidad Aumentada permite al usuario mantener contacto con el mundo real mientras interactúa con objetos virtuales.

Un usuario necesita que el sistema responda en tiempo real para poder interactuar con el sistema de forma efectiva. Un sistema de Realidad Aumentada **complementa** el mundo real siendo necesario que el usuario mantenga el sentido de presencia en ese mundo. Las imágenes virtuales se mezclan con la vista real para crear el sistema de Realidad Aumentada.

Los objetos virtuales generados por ordenador deben estar ajustados con el mundo real en todas las dimensiones. Si existen errores en el ajuste, el usuario no tendrá la percepción de ver ambas imágenes, virtual y real, fusionadas. Además, el ajuste de las imágenes debe ser correcto en todo momento, incluso cuando el usuario se esté moviendo, los cambios en la visión debidos al movimiento se deben tener en cuenta y realizar las operaciones oportunas para la situación de los objetos virtuales. Discrepancias o cambios en estos ajustes harán el trabajo con la vista de la Realidad Aumentada mucho más difícil, llegando al punto de ser molesto para el usuario y haciendo que el sistema sea inutilizable.

Milgram describe una taxonomía de cómo la Realidad Aumentada y la Realidad Virtual se relacionan (Figura 3.5) [MK94]. El mundo real y un mundo totalmente virtual son los dos extremos de esta continuidad en cuyo punto intermedio se encuentra lo que Milgram denomina **Realidad Mezclada** (o **Realidad Mixta**). La Realidad Aumentada la encontramos cerca del extremo del entorno real, siendo el mundo real complementado con datos generados por ordenador. La Virtualidad Aumentada es un término creado por Milgram para identificar sistemas que son principalmente sintéticos pero que agregan ciertas imágenes del mundo real como vídeos y texturas sobre objetos virtuales.

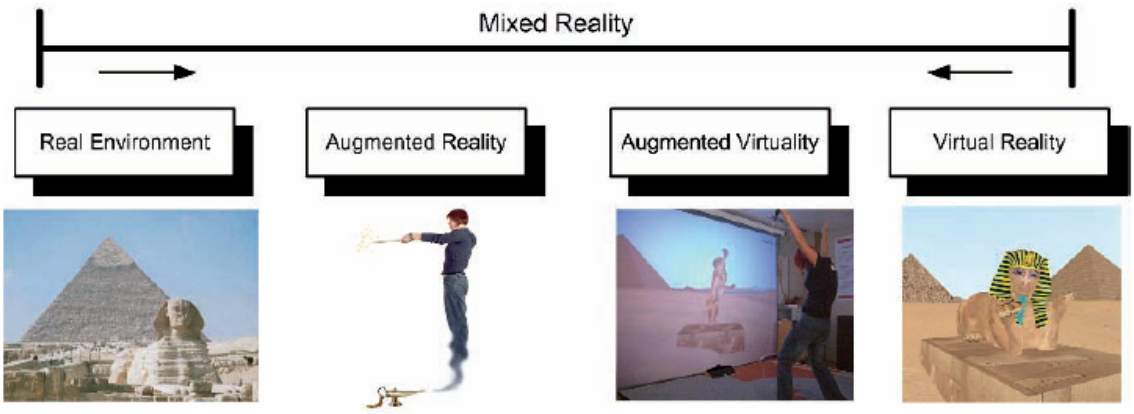

Figura 3.5: Continuo Realidad-Virtualidad de Milgram[5].

3.2.1 Niveles y tipos de Realidad Aumentada

En función del tipo de **activadores** de la información asociada a elementos podemos distinguir los siguientes niveles [Mu3]

- **Nivel 0 (Códigos QR)**:

  Utilizando los códigos QR (códigos que contienen un mensaje que puede ser leído por un lector de códigos QR) como activadores de la información asociada a un elemento, mayoritariamente hipervínculos, pero también texto, SMS, VCards o números de teléfono.

  Los códigos QR no son como los marcadores de Realidad Aumentada que únicamente pueden ser identificados por la aplicación para la que han sido diseñados. La información que se muestra en un marcador o una imagen, viene determinada por la aplicación que se ejecuta, sin embargo en un código QR la información o acción a realizar está codificada en el propio símbolo, pudiendo ser leído por cualquier lector de códigos QR.

- **Nivel 1 (Marcadores de formas geométricas)**:

  Usando Marcadores de formas geométricas sencillas, generalmente cuadrados,en los que se superpone algún tipo de información (imágenes,objetos 3D, vídeo,...) cuando son reconocidos por un software de determinado.

  El software en ejecución es capaz de realizar un seguimiento del marcador de tal manera que si el usuario lo mueve, el objeto 3D superpuesto también sigue ese movimiento, si se gira el marcador se puede observar el objeto 3D desde diferentes ángulos y si se acerca o se aleja, el tamaño del objeto aumenta o se reduce respectivamente.

[5] http://wibirama.com/dip/2010/10/19/image-processing-belajar-augmented-reality/

- **Nivel 2 (Sin marcadores)**:

  Si se emplea una imagen como 'marcador', el proceso es muy similar, tienes que ejecutar la aplicación correspondiente y captar la imagen en cuestión con la cámara, reconocida la imagen se producirá la acción que corresponda.

  En este nivel también encontramos la Realidad Aumentada basada en la localización. En los últimos años (desde el 2009) se han venido desarrollando aplicaciones para dispositivos móviles basados en este tipo de Realidad Aumentada llamadas navegadores de Realidad Aumentada.

  Estas aplicaciones utilizan el hardware de los smartphones o teléfonos inteligentes (gps, brújula y acelerómetro) para localizar y superponer una capa de información sobre puntos de interés de nuestro entorno.

  Cuando el usuario mueve el smartphone captando la imagen de su entorno, el navegador, a partir de un mapa de datos, muestra los puntos de interés cercanos.

- **Nivel 3 (Visión aumentada)**:

  Es lo que parece que puede ser el futuro de la Realidad Aumentada. En esta categoría tenemos por ejemplo las famosas gafas de Google. También encontramos unas lentes de contacto que proyectarían la Realidad Aumentada directamente a nuestros ojos, un experimento en el que está trabajando la Universidad de Washington donde ya han sido probadas en conejos sin que experimenten efectos adversos.

3.2.2 Aplicaciones

El uso de los sistemas de Realidad Aumentada se ha extendido a muchos sectores en los últimos años.

Cada vez es más evidente para los grupos de investigación, tanto en el ámbito docente como en el ámbito industrial, de la potencia que posee la Realidad Aumentada para transmitir de forma visual una información digital que **potencie** o **sustituya parcialmente** el entorno percibido.

Esto ha sido potenciado gracias a que las capacidades de procesamiento de imagen y de vídeo en tiempo real son cada vez más mayores y que van apareciendo nuevas posibilidades en los sistemas gráficos digitales se consigue cada vez efectos más realistas en los sistemas de Realidad Aumentada y esto permite reducir la sensación de artificialidad de los elementos virtuales añadidos.

Algunos de los sectores que se han hecho eco de la aplicabilidad de la Realidad Aumentada, que se mencionan a continuación:

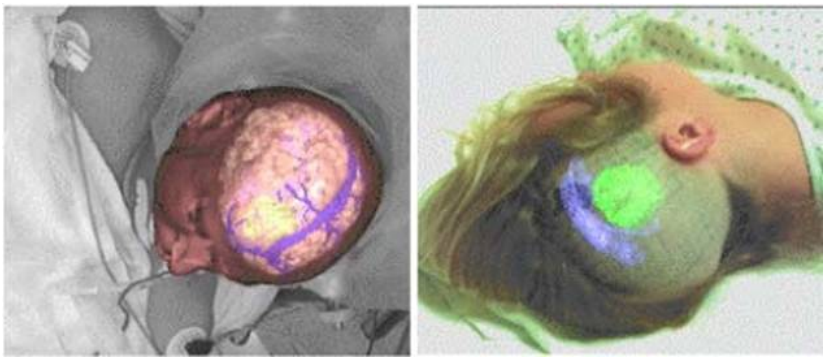

Figura 3.6: Ejemplo de Realidad Aumentada en el campo de la medicina como ayuda a cirujanos[6].

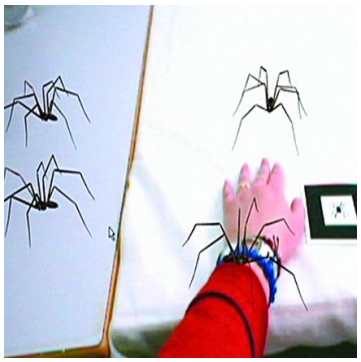

Figura 3.7: Ejemplo de Realidad Aumentada en el campo de la psicología para el tratamiento de fobias[7].

- **Medicina**

  Los médicos pueden usar la Realidad Aumentada como una ayuda a las prácticas durante la formación. También puede ser de ayuda a cirujanos durante la preparación de una operación o durante la misma (véase la figura 3.6). Utilizando estos sistemas, el tiempo de la intervención se reduce, consiguiéndose así un mayor beneficio y menor riesgo para el paciente [NUG+08].

- **Psicología**

  En psicología también se han desarrollado aplicaciones de Realidad Aumentada. Por ejemplo para el tratamiento de fobia a los animales pequeños(figura 3.7) o para la acrofobia, es decir fobia a las alturas [JJB+08].

- **Fabricación: mantenimiento y reparación**

  Otra categoría donde se pueden aplicar las posibilidades de la Realidad Aumentada es en el montaje, mantenimiento y reparación de maquinaria compleja. Por ejemplo un operario que necesite realizar una labor de montaje o mantenimiento puede tener con un sistema de Realidad Aumentada un manual visual que le indique cómo montar

[6] http://sabia.tic.udc.es/gc/Contenidos %20adicionales/trabajos/Imagenyvideo/Graficos$_e n_l a_m edicina/IntroduccionRA.$
[7] http://www.ai2.upv.es/es/mostrarnoticia.php?id=470

Figura 3.8: Ejemplo de Realidad Aumentada en el campo de la fabricación como manual para reparaciones[8].

Figura 3.9: Ejemplo de Realidad Aumentada en un parabrisas de General Motors para mejorar la visibilidad[9].

y desmotar una pieza, además de permitirle tener las manos mucho más libres que si tuviese que mirar manuales de texto tanto si son en papel como en un formato digital a través de un dispositivo móvil. La figura 3.8 muestra un ejemplo de cómo BMW está aplicando Realidad Aumentada a su cadena de montaje.

- **Anotación y visualización**

  También podemos usar los sistemas de Realidad Aumentada para comentar objetos y entornos ya sea con información pública o privada. Por ejemplo, un usuario podría ir en su coche con un sistema de visualización integrado sobre el parabrisas por una carretera y éste proporcionarle información acerca de los elementos de señalización conforme el usuario avanza o la pide, o bien señalándole dónde se encuentra un posible peligro en concreto. Esto podría ser especialmente aconsejable en días de escasa visibilidad. General Motors está desarrollando un sistema de estas características (ver figura 3.9.

  Esto puede aplicarse también al reconocimiento de la posición del usuario, haciendo que las anotaciones aparezcan si está cerca el usuario, o bien le sigan de tal forma que

[8] http://www.geeky.com.mx/2011/05/23/semana-de-realidad-aumentada-parte-1/
[9] http://www.diariomotor.com/2010/03/24/realidad-aumentada-en-un-parabrisas-de-general-motors/

Figura 3.10: Ejemplo de Realidad Aumentada mediante Chroma Key[10].

Figura 3.11: Ejemplo de Realidad Aumentada en aplicación Invizimals[11].

aparezcan siempre delante de su visión. Podemos conseguir este efecto mediante un dispositivo de rastreo que se colocaría el usuario y con lo que el ordenador conocería su localización. Esto permite que la información siempre aparezca a la vista del usuario aunque éste cambie de orientación o de posición con respecto al objeto.

- **Entretenimiento**

  Una forma simple de Realidad Aumentada se viene usando ya hace tiempo en el entretenimiento y en el mundo de las noticias en televisión. En cualquier noticiario de televisión, la información meteorológica se muestra mediante un presentador junto con las imágenes por satélite y los mapas del tiempo cambiando tras de él.

  También se emplea este método últimamente en la televisión para generar platos virtuales, colocan el presentador con una silla y una mesa reales y el resto es virtual. A esta técnica se le conoce con el nombre de Chroma Key. Recientemente se han desarrollado otros proyectos, dónde ya no es necesario que exista esta pantalla de fondo verde o azul. Se usa la temperatura del cuerpo de la persona para mantener la imagen de la persona y luego cambiar el fondo (Figura 3.10).

  Por supuesto, el campo del ocio ofrece un amplio abanico de posibilidades dónde aplicar los sistemas de Realidad Aumentada. Varios grupos y empresas están investigando

[10] http://neuronaviva.blogspot.com.es/2011/01/chroma-key-para-todos.html
[11] http://es.playstation.com/psn/games/detail/item158726/Invizimals %E2 %84 %A2/

Figura 3.12: Ejemplo de Realidad para la visualización de animales extintos[12].

Figura 3.13: Ejemplo de Realidad en situaciones militares[13].

y desarrollando juegos dentro de este campo. Existen una infinidad de posibilidades, desde juegos de mesa hasta simulaciones de juegos de combate, por ejemplo Invizimals (ver figura 3.11).

- **Aprendizaje**

  Para aprendizaje se han desarrollado sistemas para distintas materias. Por ejemplo, para contar historias interactivas, conocer animales exóticos o en peligro de extinción (véase la figura 3.12).

- **Aeronáutica y entrenamiento militar**

  Durante muchos años, la aeronáutica militar y los helicópteros han usado visores en la cabeza (Head-Up Displays) y en los cascos (Helmet-Mounted Sights) para superponer vectores gráficos en la visión del mundo real. También suelen proporcionar también información básica de vuelo y navegación, estas imágenes son también utilizadas en ocasiones a la hora de marcar el objetivo de las armas.

[12]http://nuevosmediosytelematica.bligoo.com/realidad-aumentada.U$_Y M - Lt3QYw$
[13]http://realidadaumentadaperu.blogspot.com.es/

Además de la aplicación en aeronáutica, los sistemas de Realidad Aumentada también pueden aplicarse para entrenamiento, tanto en la simulación del manejo de distintos aparatos como en el entrenamiento del personal militar en situaciones de combate (véase la figura 3.13).

3.3 Inteligencia artificial

La Inteligencia Artificial es una de las ciencias más recientes [RN04]. El trabajo comenzó poco después de la Segunda Guerra Mundial y el término se acuñó en 1956. En la actualidad, la Inteligencia Artificial abarca una gran variedad de campos, que van desde áreas de propósito general, como el aprendizaje y la percepción, a otras más específicas como la construcción de sistemas que determinen el comportamiento de elementos robóticos.

La principal función de la Inteligencia Artificial no es sólo intentar comprender cómo piensa el ser humano, sino que va más allá y también se esfuerza en construir entidades inteligentes.

Es difícil dar una definición de Inteligencia Artificial de una manera concreta, pero, podemos entenderla como la ciencia que establece los fundamentos necesarios para construir sistemas que realicen las siguientes acciones:

- **Piensan y actúan como humanos**: Estos sistemas miden su éxito en términos de la fidelidad en la forma de actuar de los humanos.
- **Piensan y actúan racionalmente**: Estos sistemas miden su éxito en en términos de racionalidad. Un sistema es racional si hace lo correcto, en función de su conocimiento.

3.3.1 Agentes inteligentes

En el presente Proyecto Fin de Carrera se considera que la inteligencia está relacionada principalmente con acciones racionales.

Un **agente** es cualquier entidad capaz de percibir su **medioambiente** con la ayuda de **sensores** y actuar en ese medio utilizando **actuadores** (ver figura 3.14)[RN04].

El término **percepción** se utiliza en este contexto para indicar que el agente puede recibir entradas en cualquier instante. La **secuencia de percepciones** de un agente refleja el historial completo de lo que el agente ha recibido. En general, un agente tomará una decisión en un momento dado dependiendo de la secuencia completa de percepciones hasta ese instante.

En términos matemáticos se puede decir que el comportamiento del agente viene dado por la **función del agente** que proyecta una percepción dada en una acción.

Un **agente racional** es aquel que hace lo correcto. Lo correcto es aquello que permite al agente obtener un resultado mejor que un comportamiento incorrecto.

Es necesario definir medidas de rendimiento que incluyen los criterios que determinen el

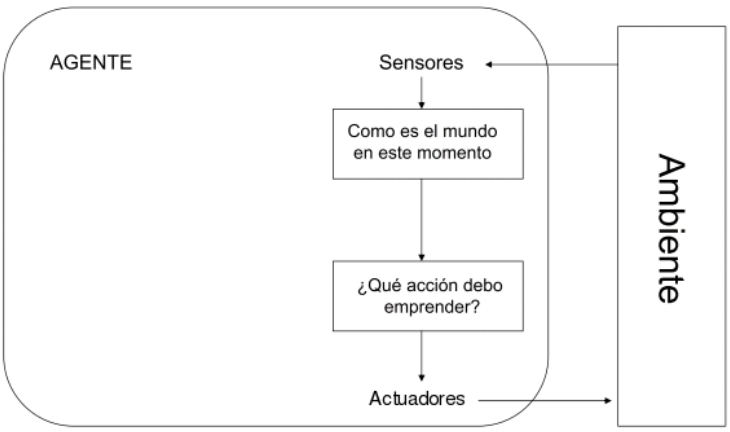

Figura 3.14: Arquitectura de un agente simple

éxito en el comportamiento del agente. La **racionalidad** [RN04] en un momento determinado depende de cuatro aspectos:

- La medida de rendimiento que define el criterio de éxito.
- El conocimiento del medio en el que habita acumulado por el agente.
- Las acciones que el agente puede llevar a cabo.
- La serie de percepciones obtenidas por el agente hasta ese momento.

Sin embargo, lo primero en lo que hay que centrarse es en los denominados **entornos de trabajo**. Se definen los entornos de trabajo como los 'problemas' a los que tienen que enfrentarse los agentes racionales para hallar sus 'soluciones'. En [RN04] puede encontrarse la siguiente clasificación en función de las propiedades del entorno:

- **Total o parcialmente observable**. Un entorno será totalmente observable si el agente es capaz de acceder a toda la información del medio. En cambio, un entorno será parcialmente aceptable si el agente sólo puede acceder a una parte de la información.
- **Deterministas o estocásticos**. Si el siguiente estado del medio está totalmente determinado por el estado actual y la acción ejecutada por el agente, entonces se dice que el entorno es determinista; de otra forma es estocástico.
- **Episódico o secuencial**. En un entorno episódico la elección de la acción por parte del agente depende sólo del episodio en sí mismo. Por otro lado, en entornos secuenciales, la decisión presente puede afectar a decisiones futuras. Los medios episódicos son más simples que los secuenciales porque el agente no necesita pensar con el tiempo.
- **Estático o dinámico**. Si el entorno puede cambiar cuando el agente está deliberando, entonces se dice que el entorno es dinámico para el agente; de otra forma se dice que es estático.
- **Discreto o continuo**. Un entorno discreto es aquel en que existe un número finito de acciones y percepciones en el mismo; en otro caso, se dice que el entorno es continuo.

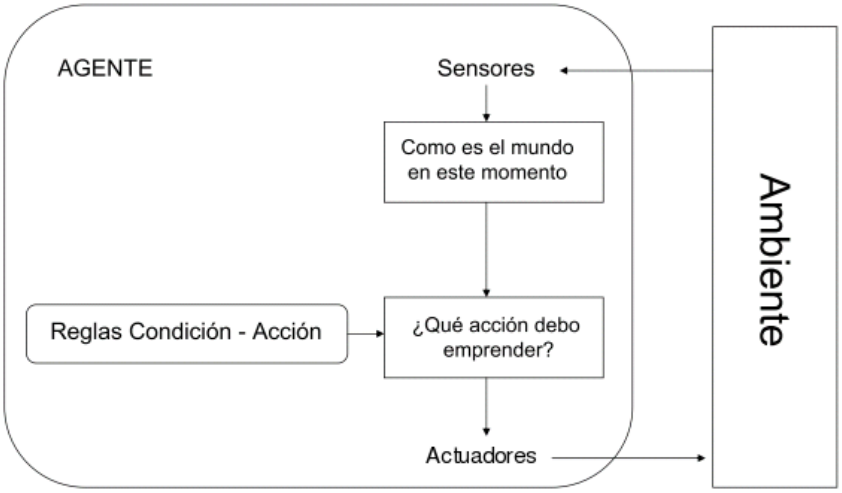

Figura 3.15: Arquitectura de un agente reactivo simple

- **Agente individual o multi-agente**.

El trabajo de la Inteligencia artificial es diseñar el programa del agente que implemente la función del agente que proyecta las percepciones en las acciones. Se asume que este programa se ejecutará en algún tipo de computador con sensores y actuadores, lo cual se conoce como arquitectura. Por tanto se puede definir un agente de la siguiente manera:

$$Agente = Arquitectura + Programa \tag{3.1}$$

Debido a esto existen varios modelos de agentes en función del tipo de programa. A continuación se puede el detalle estos tipos de agentes según [RN04]:

Agentes reactivos simples

Es el tipo de agente más sencillo, puesto que selecciona las acciones a tomar dependiendo del conjunto de percepciones actuales, ignorando las percepciones históricas. Utilizan reglas del tipo **condición-acción**, esto es, si el agente recibe una percepción entonces actúa (ver figura 3.15).

Agentes reactivos basados en modelos

El agente mantiene algún tipo de **estado interno** que dependa de la historia percibida y que, de ese modo, refleje por lo menos alguno de los aspectos no observables del mundo actual. Cuando el agente decida la acción que debe realizar tendrá que valorar la secuencia de estados almacenados, ver cómo evoluciona el mundo en función de la misma y conocer qué efectos pueden ocasionar sus acciones (ver figura 3.16).

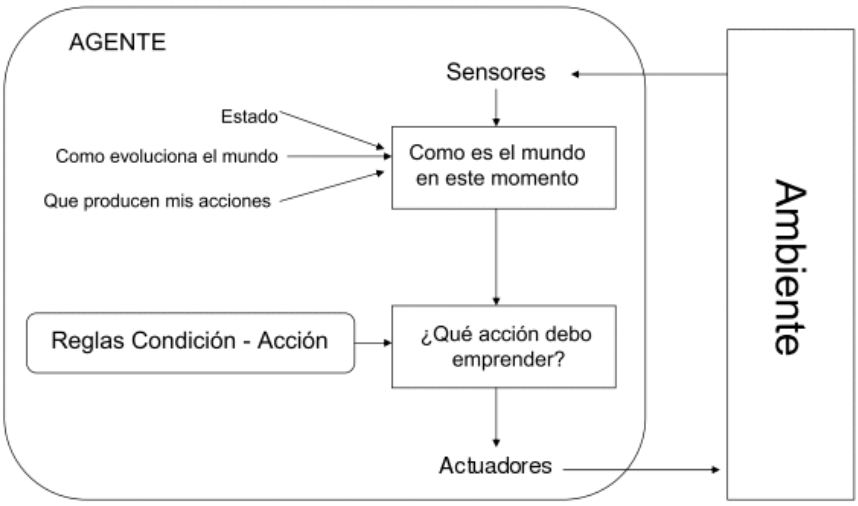

Figura 3.16: Arquitectura de un agente reactivo basado en modelos

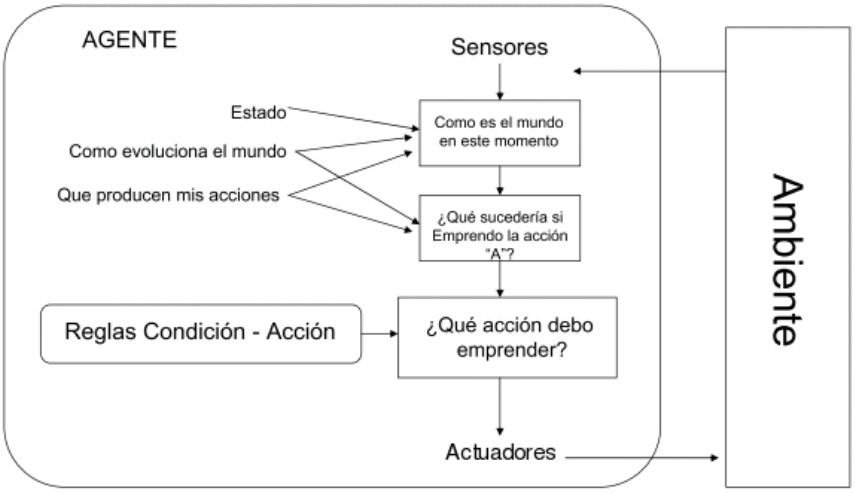

Figura 3.17: Arquitectura de un agente basado en objetivos

Agentes basados en objetivos

Este tipo de agentes complementa a los del tipo anterior, puesto que, además del estado interno, este tipo de agente almacena información sobre una **meta** que describa las situaciones que son deseables. Un agente basado en objetivos almacena información del estado del mundo, así como un conjunto de objetivos que intenta alcanzar, y que es capaz de seleccionar la acción que lo guiará hacia la consecución de sus objetivos (ver figura 3.17).

Agentes basados en utilidad

Este tipo de agente, además de la información sobre la meta, hace uso de una **función de utilidad** que calcula sus preferencias. Después, selecciona la acción que le lleve a alcanzar la mayor utilidad esperada.

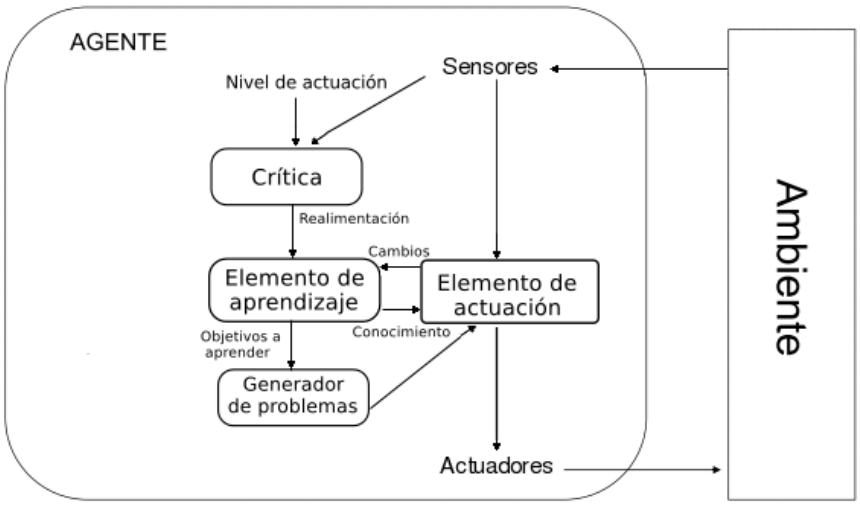

Figura 3.18: Arquitectura de un agente que aprende

Agentes que aprenden

Un agente que aprende se puede dividir en cuatro componentes conceptuales, tal y como se muestra en la figura 3.18. Estos componentes son los siguientes:

- **Crítica**. Indica al elemento de aprendizaje qué tal lo está haciendo el agente con respecto a un nivel de actuación fijo. Este componente es necesario porque las percepciones por sí mismas no prevén un acierto del agente.
- **Elemento de aprendizaje**. Este componente es el encargado de hacer mejoras en el agente. Se realimenta con las críticas sobre la actuación del agente y determina cómo se debe modificar el elemento de actuación para proporcionar mejores resultados.
- **Elemento de actuación**. El objetivo de este componente es la selección de acciones externas.
- **Generador de problemas**. Es el encargado de sugerir acciones que guiarán al agente hacia situaciones nuevas e informativas.

3.3.2 Planificación multi-agente

La planificación multi-agente es un mecanismo de coordinación en el que los agentes construyen secuencias de acciones o planes, los cuales especifican todas sus acciones futuras e interacciones con respecto a un objetivo particular. Cuando estos planes se ejecutan correctamente y en orden, llevan al sistema a un estado u objetivo deseado. Con esta aproximación los agentes saben de antemano qué acciones llevarán a cabo, qué acciones llevarán a cabo los restantes agentes, y qué interacciones se producirán. Lógicamente, esta técnica de coordinación tiene un horizonte temporal corto, debido a que los planes requieren una especificación de comportamiento completa, y pueden producirse en el entorno eventos inesperados que eviten la ejecución con éxito de los planes, siendo por tanto necesaria una replanificación.

La planificación clásica asumía que era un único agente el que llevaba a cabo la planificación siendo este la única fuente de cambio en su entorno, y por tanto descartando otras acciones inesperadas y concurrentes en el mismo. El ejemplo más conocido en este sentido es la aproximación STRIPS para el mundo de bloques. Sin embargo, y debido a la posible aparición de eventos en el entorno que vayan más allá del control de los agentes se desarrollaron extensiones a la planificación clásica. Las posibles contingencias fueron consideradas de forma explícita en los planes, la ejecución de los planes fue controlada, y los planes revisados teniendo en cuenta la posible aparición de eventos relevantes [RN04].

Sin embargo, la situación de la planificación multi-agente ocupa un lugar intermedio con respecto a las dos aproximaciones anteriores. Aunque un entorno multi-agente esta constantemente cambiando, los cambios son debidos a acciones llevadas a cabo por otros agentes, y estas no son necesariamente impredecibles e incontrolables. Generalmente estas acciones están encaminadas hacia la consecución de los objetivos de los agentes, y por tanto el agente de planificación puede predecirlas, e incluso influir en la forma en la que éstas ocurren.

La planificación multi-agente o distribuida se enfrenta al problema de cómo un agente debería formular un curso de acción que tuviera en cuenta las acciones deseadas e indeseadas que los restantes agentes podrían llevar a cabo concurrentemente:

- Cómo un agente debería averiguar qué cursos de acción llevarán los restantes agentes.
- Cómo un agente debería modelar cómo sus acciones son anticipadas por los otros.
- Cómo un agente puede influenciar a los otros cambiando sus modelos de acciones.
- Cómo un agente debería comprometerse con los modelos de él mismo.

Las primeras aproximaciones a la planificación distribuida operan en entornos estables, y por este motivo primero generan planes detallados y completos para todos los agentes y luego actúan o ejecutan dichos planes. Sin embargo aplicaciones más recientes de la planificación multi-agente se usan también para entornos dinámicos. Como consecuencia de esto los planes se construyen (y reconstruyen) basándose en el modelo parcial actual del agente acerca del entorno y acerca de los otros agentes, y este modelo se revisa continuamente durante el curso de acción de dicho agente. Esto significa que, a diferencia de la primera aproximación, la planificación debe ser intercalada con la ejecución del plan.

En una aproximación centralizada un agente es el responsable del reconocimiento y reparación de las interacciones entre planes. Este agente o coordinador puede ser definido estáticamente [Geo83] o dinámicamente [CMS83]. Los restantes agentes envían sus planes separados al coordinador, éste examina los planes con el objetivo de descubrir regiones críticas en las que interacciones entre los planes de los agentes podrían conducir a fallos, y los modifica de la forma más adecuada. Por ejemplo, en el trabajo de Georgeff se insertan mensajes de sincronización en los planes. Sin embargo, en la aproximación de Cammarata. los agentes envían al coordinador información acerca de sus acciones. A continuación el

coordinador construye un plan que especifica todas las acciones de los agentes, incluyendo las acciones que ellos o algún otro agente deberían adoptar para evitar colisiones.

En contraste con la anterior, en la aproximación distribuida el reconocimiento de interacciones y la reparación de planes se hace de forma distribuida, es decir el plan es desarrollado por varios agentes. Esto significa que no hay agentes individuales con una visión completa de las actividades de la comunidad, y por tanto detectar y resolver interacciones indeseables puede ser más difícil. En su sistema NOAH distribuido [Cor79] Corkill desarrolla versiones distribuidas de NOAH. Como antes, los sub-objetivos son asignados a diferentes agentes, y cada agente construye un plan para su sub-objetivo. Sin embargo, cada agente también tiene en cuenta el hecho de que los otros agentes están llevando a cabo concurrentemente sus planes. De esta forma, a medida que los planes se construyen localmente, los efectos de estos planes se propagan de modo que determinadas restricciones de orden sobre los pasos del plan, que prevean interacciones, puedan ser identificadas y resueltas (usando mensajes de sincronización).

3.4 Sistemas de vigilancia

Se entiende por sistema de videovigilancia a toda aquella actividad que implique la colocación de una o varias cámaras de grabación, fijas o móviles, que tengan la finalidad de **monitorizar** el comportamiento y actividades de un espacio o personas [4]. Los sistemas de videovigilancia pueden estar compuestos, simplemente, por una o más cámaras de vigilancia conectadas a uno o más monitores o televisores que reproducen las imágenes capturadas por las cámaras. Los sistemas de videovigilancia pueden clasificarse por el tipo de sensor (infrarrojo, audio y vídeo), por la multiplicidad del sensor (monocular o estéreo) y por el emplazamiento del sensor (centralizado o distribuido) [VV05a].

Según Valera y Velastin la forma más apropiada de clasificar los sistemas de videovigilancia es clasificarlos en tres generaciones que se distinguen por la tecnología empleada en ellos. La **primera generación** la forman los sistemas de CCTV. Se les conoce por el nombre de circuito cerrado porque todos los componentes que lo forman están enlazados entre sí. Este conjunto de sistemas están formados por un grupo de cámaras distribuidas que se sitúan en la zona de vigilancia y se conectan a un conjunto de monitores que normalmente se ubican en una sala central. Este tipo de sistemas son **completamente analógicos** y necesita de la **supervisión** de personas para hacer posible la detección de sucesos de interés. A pesar de sus limitaciones proporcionan una baja tasa de errores y son sistemas ampliamente utilizados debido en gran parte a la madurez de la tecnología que emplean. Sin embargo utilizan técnicas analógicas para el tratamiento, distribución y almacenamiento de las imágenes, y dependen demasiado de la actividad humana para detectar las anomalías que suceden en el entorno.

Los sistemas de **segunda generación** combinan la tecnología de los sistemas CCTV analógicos con los sistemas digitales de vigilancia-IP. El objetivo es intentar reducir la dependencia que existe de la actividad humana interpretando automáticamente los eventos y comportamientos que se producen en el entorno monitorizado. Estos sistemas permiten **incrementar la eficiencia** de los sistemas de seguridad, observando y analizando un mayor número de situaciones al mismo tiempo. También **reducen la presencia** del factor humano para detectar situaciones anómalas. Actualmente no existe una solución que permita realizar un razonamiento general sobre cualquier situación, solo existen soluciones parciales que interpretan soluciones muy concretas. Además existen situaciones anómalas imprevisibles que podrían no ser detectadas, y algunas de las soluciones propuestas no son demasiado robustas, dando lugar a un número elevado de falsas alarmas.

Por último los sistemas de **tercera generación** se caracterizan por ser altamente distribuidos. Estos sistemas emplean dispositivos más modernos que mejoran o complementan los aparatos utilizados en anteriores generaciones. Al ser sistemas más distribuidos que los anteriores, la carga de procesamiento ya no se encuentra centralizada, **reduciendo** así la cantidad de datos que hay que procesar en cada dispositivo y ofreciendo **tiempos de respuesta** más cercanos al tiempo real. Además este último tipo de sistemas son más **sólidos** dado que si algún equipo se daña el sistema puede seguir trabajando con normalidad. Sin embargo existen problemas para distribuir eficazmente la información, aparte de que la comunicación entre dispositivos puede ser complicada debido a la heterogeneidad de sus componentes.

3.4.1 Visión artificial

Existen varios tipos de videovigilancia basada en sistemas de visión artificial. Entre ellos podemos encontrar:

Vídeo-detección de intrusos

Un sistema de vídeo detección de intrusos [Ote06], es una tecnología capaz de detectar accesos no autorizados por personas en áreas de seguridad o restringidas como aeropuertos, centros de detención de máxima seguridad o instalaciones nucleares . La función principal de este tipo de sistemas es la de **detectar anomalías** que puedan ser indicios de intrusión o alarmas.

Para hacer posible la detección de intrusos, se recopila información de múltiples fuentes del sistema, analizando los datos obtenidos de diferentes maneras. Algunas de estas consisten en hacer un análisis estadístico sobre la información, buscando patrones que indiquen actividad anormal.

Figura 3.19: Ejemplo de Vídeo-detección de intrusos[14].

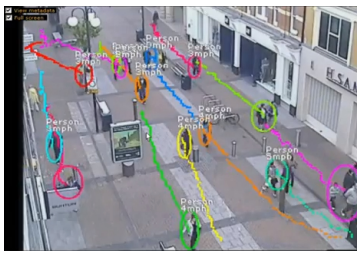

Figura 3.20: Ejemplo de Video tracking en seguridad vial[15].

En la figura 3.19 podemos ver un ejemplo de una aplicación basada en detección de intrusos.

Video Tracking

Se entiende por Video Tracking [TBP03] al proceso de localización de un objeto en movimiento (o varios) y su posterior seguimiento en el tiempo mediante el uso de cámaras de vídeo.

Se utiliza en gran cantidad de aplicaciones, por ejemplo: interacción hombre-máquina; seguridad y vigilancia (ver figura 3.20); compresión y comunicación de vídeo; realidad aumentada; control de trafico, imágenes médicas y edición de vídeo.

Los sistemas visuales de seguimiento tienen dos principales componentes: **representación y localización del objetivo**, y **filtrado de la información asociada**. El primera utiliza una gran variedad de algoritmos para identificar el objeto en movimiento, por ejemplo: 'Blob tracking', 'Kernel-based tracking', 'Contour Tracking', 'Feature matching' [KAK+03]. El segunda necesita conocer información a priori de la escena o del objeto para poder hacer su seguimiento . Este algoritmo permite seguir objetos complejos incrementado notablemente la complejidad computacional. Los algoritmos más comunes son: filtro de Kalman [SSC06] y 'Particle filter' [GAMC02].

[14] http://security.panasonic.com/pss/security/es/products/hd/i-VMD/index.html
[15] http://www.corpwatch.org/article.php?id=15767

Figura 3.21: Ejemplo de detección de objeto abandonado en un aeropuerto[16].

Detección de objetos abandonados

Esta tarea consiste en localizar automáticamente objetos que fueron **abandonados en una escena** (ver figura 3.21). Normalmente este tipo de objetos son bastante pequeños comparados con las personas y frecuentemente son ocultados por otras personas o vehículos que se mueven alrededor de la escena.

Su funcionamiento se basa en utilizar **algoritmos de segmentación** que actúan en el primer plano de la secuencia. Los pixels que no son parte del fondo y no se mueven se utilizan para encontrar regiones que contienen los posibles objetos abandonados. Sin embargo este tipo de sistemas son vulnerables a los problemas causados por fluctuaciones de iluminación, sombras y niveles de contraste de la escena [VSS07].

Conteo de objetos

Los sistemas de conteo de objetos en entornos de vídeo son aquellos que son capaces de llevar la cuenta del **número de objetos similares** que aparecen en una escena en un determinado espacio de tiempo [VV05a].

Estos sistemas se basan en las características del objeto para su identificación. Existe un amplio campo de aplicaciones para este tipo de sistemas entre las que destacan el análisis de tráfico, el conteo de personas en zonas vídeo-vigiladas y procesos de fabricación.

3.4.2 Realidad aumentada

Aunque el campo de la realidad aumentada esté en auge, sus aplicaciones derivan más en el entretenimiento y el aprendizaje que en sistemas de vigilancia. Sin embargo podemos encontrar algunos ejemplos de sistemas basados en Realidad aumentada dirigidos al campo de la vigilancia y la seguridad. A continuación se detallan algunos de ellos.

[16] http://www.madrimasd.org/informacionidi/noticias/noticia.asp?id=36255

Proyecto Hesperia

Una de las aplicaciones centradas en el mundo de la seguridad es el proyecto **Hesperia**[17].

Hesperia ha sentado las bases del que puede ser el sistema de vigilancia del futuro, a partir de la integración en una **única plataforma** de los sistemas más avanzados de representación 3D; sistemas de videovigilancia inteligente, que permiten **clasificar objetos** y **detectan cualquier movimiento anormal**; y un **Sistema de Información Geográfica (GIS)**, desarrollado por Indra. Este último sistema permite posicionar gráficamente todo tipo de elementos importantes para la seguridad, como cámaras o extintores, así como a los propios guardias de seguridad o intrusos detectados, cuyos movimientos pueden seguirse en la pantalla.

Gracias a la solución que permite la comunicación entre todos los sistemas integrados en la plataforma (middleware), toda esta información pasa al sistema de conocimiento, que da la alarma automáticamente ante cualquier incidencia detectada. Lo mismo sucede con la información procedente de los sensores y sistemas de audio cognitivo, que detectan, clasifican y localizan sonidos de rotura de cristales, disparos, golpes metálicos o sirenas.

La plataforma también integra **sistemas de control de accesos avanzados**, que ante cualquier incidencia incrementan su nivel de seguridad pasando, por ejemplo, de requerir para la identificación una tarjeta de empleado a exigir además controles biométricos de reconocimiento facial o de voz, que en este último caso pueden analizar no sólo la identidad de la persona sino si se encuentra sometida a alguna situación extraña que le genera pánico, ansiedad o enfado.

Cabe destacar igualmente los **sistemas de gestión de crisis**, que ante un incendio o una intrusión ofrecen las mejores alternativas de actuación, o los sistemas de movilidad, que permiten que los vigilantes puedan acceder a información de la plataforma a través de PDAs. Asimismo, el proyecto ha supuesto un importante avance en lo que se conoce como 'cableado virtual', que permite la conexión inalámbrica de sensores distribuidos, algo importante, por ejemplo, en el caso de un acto terrorista, ya que no se deshabilitan todos, como sí ocurre en el caso de sensores centralizados.

Proyecto RAIOM

Aunque este proyecto no esté enfocado a los sistemas de vigilancia de entornos, el **proyecto Realidad Aumentada para la Identificación de Objetivos Militares (RAIOM)**[18] sirve de referencia para el presente Proyecto de Fin de Carrera porque esté enfocado a la localización de objetivos.

[17] http://www.indracompany.com/noticia/el-proyecto-hesperia,-liderado-por-indra,-desarrolla-tecnologia-punta-para-seguridad-de-infraestructuras

[18] http://www.infodefensa.com/latam/2012/06/14/noticia-argentina-desarrolla-el-proyecto-realidad-aumentada-para-la-identificacion-de-objetivos-militares.html

Este proyecto consiste en el diseño de un '**framework**' específico para el desarrollo de software vinculado con la Realidad Aumentada en el área militar y relacionado al concepto de '**Guerra o Batalla Ubicua**', en donde cada soldado de un grupo de combate utiliza equipamiento de Realidad Aumentada individual (pantallas flexibles, tabletas, anteojos, etcétera), interconectados entre sí y con el centro de mando, que reciben información del entorno a través de sensores, satélites, vehículos aéreos no tripulados (UAV) y otros dispositivos que sirven como soporte a la conciencia situacional para la toma de decisiones.

Normalmente, una aplicación de Realidad Aumentada incluye un sistema de posicionamiento global (GPS), brújula electrónica y/o acelerómetros, vídeo en tiempo real y acceso directo a información geo-referenciada. Al final del Proyecto se tendrá un prototipo de software o versión inicial de Realidad Aumentada que posibilitará por un lado el desarrollo posterior de diversas aplicaciones totalmente operativas y funcionales para ser utilizadas por las fuerzas militares y potenciar en el futuro la implementación y utilización de esta tecnología en los diferentes ámbitos de la Defensa y en otros campos de aplicación, debido a su carácter dual.

3.4.3 Inteligencia artificial

Los sistemas de videovigilancia convencional pueden grabar lo que ven, pero no pueden procesar la imagen. Esa tarea normalmente es responsabilidad del personal de seguridad, que ha visto cómo su trabajo se ha ido haciendo más exigente al aumentar el número medio de cámaras de vigilancia.

Como apunta un experto del sector, 'En el pasado, el personal de seguridad visualizaba una cámara en un único monitor. Ahora no es raro encontrar 20 cámaras conectadas a una sola pantalla. Tras 20 minutos de vigilancia, la atención humana a los detalles del vídeo disminuye hasta niveles inaceptables y la videovigilancia deja de tener sentido. La vigilancia tradicional mediante vídeo ya no puede cumplir las demandas del sector, que cada vez son mayores.' Por supuesto, algunos sistemas de vigilancia utilizan cámaras que emplean la detección de movimiento, pero dependiendo de cómo y dónde se usen, estos sistemas pueden generar falsas alarmas con frecuencia. La detección de movimiento no distingue entre hojas que caen, un gato que salta o un ladrón.

El software de videovigilancia inteligente supera dichas limitaciones, permitiendo que los fabricantes de equipos de videovigilancia e integradores de sistemas creen soluciones de vídeo inteligentes que ven y procesan la información visual de forma similar a los humanos. Por ejemplo, estos sistemas de análisis de vídeo pueden distinguir entre una persona y un coche. Estos sistemas se pueden programar para seguir sólo los objetos identificados como humano y enviar un aviso en caso de que ese sujeto infrinja unas reglas predefinidas, como saltar una tapia.

En este sentido, podemos decir que estos sistemas permiten capturar imágenes en tiempo real cuando se produce una alarma y de esta manera acelerar el tiempo de respuesta en caso de emergencia, robo o incendio, evitar las falsas alarmas y garantizar la seguridad de hogares. Esta tecnología permite que el sistema envíe las imágenes al teléfono móvil o al email al menor signo de incidencia y permite saber en tiempo real qué sucede en su hogar mientras su propietario está ausente. El desarrollo de estos sistemas lleva aparejada la progresiva disminución de los servicios de 'acuda', que suponen el desplazamiento de un vigilante hasta el lugar de la alarma, lo cual no sólo implica mayores costes, sino un mayor tiempo de respuesta que puede ser clave en caso de emergencia.

Como se puede apreciar a continuación, la propuesta de valor de los sistemas de videovigilancia inteligentes es exhaustiva, lo que supone una mayor rentabilidad y una vigilancia de mayor calidad, así como una mayor escalabilidad[19]:

- **Más rentable**. Los entornos de videovigilancia convencional requieren personal de seguridad para pasar muchas horas viendo vídeos en directo o grabados, y analizar e identificar acontecimientos sospechosos.

  En contraposición, los sistemas de videovigilancia inteligentes pueden examinar miles de horas de datos de vídeo sin la intervención de personas. Si el sistema de videovigilancia inteligente detecta una actividad sospechosa, puede avisar automáticamente al personal de seguridad para que lo analice. Si se aumenta el número de transmisiones de vídeo que puede gestionar cada miembro del equipo de seguridad, los sistemas de videovigilancia inteligentes pueden recortar significativamente los costes de personal. Además, los sistemas de videovigilancia inteligentes permiten al personal de vigilancia realizar tareas de seguridad clave sin supervisión.

- **Más preciso**. Varias pruebas han demostrado que las personas pierden entre el 50 y el 90 % de su percepción visual tras 20 minutos de supervisión continua de vídeos. Cuantas más transmisiones de vídeo tenga que supervisar una persona en un determinado período de tiempo, más pronto se manifestarán los trastornos de la percepción visual.

  Por el contrario, los sistemas de videovigilancia inteligentes son inmunes a la fatiga, las distracciones y los lapsus de memoria inherentes a las personas. La revisión y el análisis de los vídeos en directo y grabados con los sistemas de videovigilancia inteligentes registran continuamente las imágenes de vídeo en busca de comportamientos y patrones seleccionados por el usuario (por ejemplo, intrusos, números de matrículas y equipaje desatendido), mientras que ignoran los datos superfluos que generan las falsas alarmas.

[19] http://www.seagate.com/es/es/tech-insights/video-security-gets-smarter-with-intelligent-video-surveillance-systems-master-ti/

- **Mayor capacidad de ampliación**. Los entornos de seguridad son dinámicos, cambian de tamaño y carácter con el crecimiento empresarial. Así, cualquier solución de videovigilancia debe permitir la escalabilidad y adaptación a las necesidades imperantes. Con un sistema de seguridad convencional, se debería contratar y formar a más personal para supervisar cualquier transmisión de cámara adicional, un proceso costoso y que conlleva una gran pérdida de tiempo.

  Los sistemas de Sistemas de Videovigilancia Inteligentes (IVS) permiten ampliar el área de videovigilancia de una empresa en un instante; simplemente se deberán conectar más cámaras y el software de vídeo inteligente del sistema empezará inmediatamente a supervisar y analizar los nuevos datos de vídeo añadidos. La escalabilidad es realmente ilimitada: sólo hay que implementar más cámaras y unidades de disco duro adicionales para almacenar los datos de vídeo.

3.4.4 Vigilancia móvil

Se puede ver el constante crecimiento de los sistemas de vigilancia móvil en la actualidad. No hay más que ver los nuevos sistemas de vigilancia de entornos forestales y agrícolas mediante el uso de disposivos móviles como los drones.

Se trata de sistemas robóticos móviles que se han equipado con todos los sistemas de vigilancia que el entorno o una aplicación concreta puedan exigir en cada caso. Su principal función es la de realizar vigilancia e inspección dentro de cualquier superficie que requiera de ella como, por ejemplo, museos, bibliotecas, almacenes, plantas industriales, plantas solares, etcétera.

Actualmente, la seguridad de este tipo de instalaciones se realiza por medio de un circuito de cámaras de vigilancia o por medio de personal humano encargado de realizar distintas rondas en busca de cualquier anomalía. Estos tipos de trabajo poseen un alto riesgo, debido a que son estas personas las encargadas de velar por la seguridad de los bienes custodiados.

El uso de este tipo de sistemas, en los que no se hace necesaria la presencia de los operadores 'in situ' provoca un mayor grado de seguridad, además de una reducción de los costes a largo plazo. Para este tipo de sistemas la inversión inicial es considerable, al tener que pagar todos los componentes electrónicos, pero sin embargo el mantenimiento es mucho menor.

Estos sistemas están gobernados por un software de teleoperación capaz de controlar una plataforma móvil dotada con varios sensores y cámaras que den soporte a sesiones de telepresencia. El término **telepresencia** se refiere a un conjunto de tecnologías y a una situación ideal que permiten a una persona sentirse presente en una localización distinta a la que se encuentra.

Los escenarios donde podemos operar este tipo de sistemas son a través de la red (internet) o a través de una red local (WIFI). Además, cualquier persona no experta es capaz de controlar el sistema de forma natural e intuitiva. Además con los últimos avances en el campo de la Inteligencia Artificial, se ha conseguido mejorar la interacción con este tipo de elementos robóticos, al ser estos cada vez más autónomos.

Se puede observar un ejemplo de este tipo de sistemas en un proyecto iniciado en San Luis, Argentina, en el que se hace uso de drones para la videovigilancia. Los drones son unos robots androides, utilizados en muchos lugares del mundo para combatir el delito y asistir en situaciones de emergencia.

Nicolás Anzulovich, jefe del Programa de Innovación Tecnológica, perteneciente al Ministerio de Seguridad aseguró: 'La implementación de los drones son para que cumplan una función preventiva en seguridad, ya que tienen una cámara móvil montada en una plataforma voladora con las aplicaciones específicas que servirá a la policía para llegar a lugares en los que no puede acceder fácilmente, también serán útiles para eventos deportivos, controles de seguridad vial e incluso catástrofes naturales.' [20]

Él mismo apunta que 'La diferencia con las otras cámaras de seguridad es que los drones son totalmente manejables y movibles y en cambio las demás cámaras de seguridad son fijas.' lo que corrobora que los sistemas de vigilancia móvil ofrecen mayor libertad a la hora de realizar las tareas de vigilancia que los sistemas convencionales mediante cámaras, aunque esto suponga un gran desembolso inicial.

[20] http://agenciasanluis.com/notas/2013/09/18/implementaran-drones-para-video-vigilancia/

Capítulo 4

Método de trabajo

En este capítulo se detalla el método de trabajo para el desarrollo del Proyecto Fin de Carrera Sivi. También se describen las herramientas software y hardware empleadas en la construcción del mismo.

4.1 Metodología de trabajo

A continuación se detallan las metodologías seguidas a lo largo de todo el ciclo de vida del desarrollo.

4.1.1 Scrum

Para la realización del presente proyecto se ha optado por el uso de metodologías ágiles de desarrollo software, en especial **Scrum**.

Scrum es un marco de trabajo basado en un proceso iterativo e incremental. Es un modelo de referencia en el que se definen distintos roles y practicas. Los roles principales relativos al proceso de scrum son:

- **Scrum master**. No es el líder del grupo, sino que simplemente es el encargado de que se cumplan las reglas y de que el equipo no se distraiga por ninguna influencia externa. Hace de escudo.
- **Product owner**. Es el encargado de hablar por el cliente. Se asegura de que el equipo trabaje correctamente desde el punto de vista de la empresa.
- **Scrum team**. Es el grupo de trabajo, los que tienen la responsabilidad de entregar el trabajo.

Todos los roles definidos anteriormente se corresponden con un grupo denominado 'cerdos'. También existe otro grupo denominado 'gallinas' que aunque no son parte del proceso de scrum hay que tener en cuenta. Estos otros roles son los del usuario, los Stakeholders (clientes, proveedores, inversores) y los managers.

En Scrum se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde

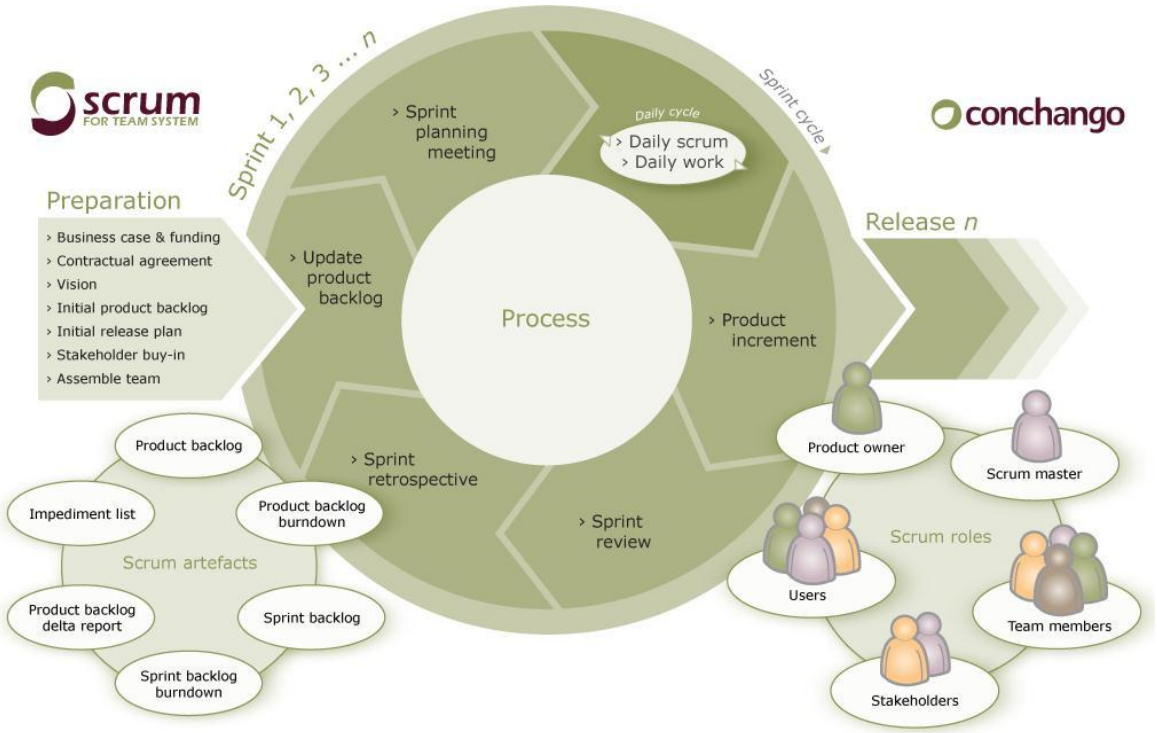

Figura 4.1: Resumen de metodología Scrum[1].

los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

Scrum también se utiliza para resolver situaciones en que no se está entregando al cliente lo que necesita, cuando las entregas se alargan demasiado, los costes se disparan o la calidad no es aceptable, cuando se necesita capacidad de reacción ante la competencia, cuando la moral de los equipos es baja y la rotación alta, cuando es necesario identificar y solucionar ineficiencias sistemáticamente o cuando se quiere trabajar utilizando un proceso especializado en el desarrollo de producto.

Como se ha mencionado anteriormente, scrum se basa en un procesa iterativo e incremental. Esto se consigue mediante los denominados sprints. Durante cada sprint el equipo crea un incremento de software potencialmente entregable, o lo que es lo mismo, utilizable. El software realizado en cada sprint se corresponde con una serie de requisitos, que se encuentran en el product backlog. En el product backlog se encuentran todos los requisitos de alto nivel, priorizados, que definen el trabajo a realizar. Los requisitos que van a formar parte de cada sprint se determinan en la reunión de sprint planning. En ella es el product owner el que especifica los requisitos que desea en dicho sprint y el scrum team decide qué cantidad de trabajo se compromete a realizar. Una vez decididos los requisitos del sprint, estos no pueden cambiar durante el sprint, se encuentran congelados. Los sprints pueden durar entre una y cuatro semanas.

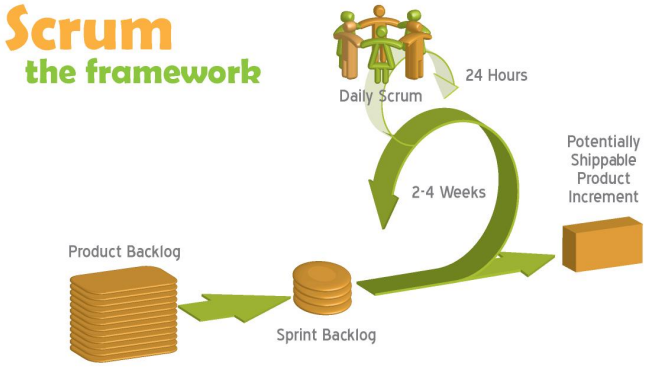

Figura 4.2: Resumen de un sprint en Scrum[3].

Por lo tanto podemos resumir los fundamentos de Scrum de la siguiente forma[2]:

- El **desarrollo incremental** de los requisitos del proyecto en bloques temporales cortos y fijos (iteraciones de un mes natural hasta de dos semanas, si así se necesita).
- La **priorización de los requisitos** por valor para el cliente y coste de desarrollo en cada iteración.
- El **control empírico** del proyecto. Por un lado, al final de cada iteración se demuestra al cliente el resultado real obtenido, de manera que pueda tomar las decisiones necesarias en función de lo que observa y del contexto del proyecto. Por otro lado, el equipo se sincroniza diariamente y realiza las adaptaciones necesarias.
- La **potenciación del equipo**, que se compromete a entregar unos requisitos y para ello se le otorga la autoridad necesaria para organizar su trabajo.
- La **sistematización** de la colaboración y la comunicación tanto entre el equipo como con el cliente.
- El **timeboxing** de las actividades del proyecto, para ayudar a la toma de decisiones y conseguir resultados.

Gracias a este proceso de desarrollo (que se puede ver en las figuras 4.1 y 4.2), podemos involucrar antes al equipo de pruebas lo antes posible. De esta forma, se facilitará la posibilidad de detectar y solucionar a tiempo todos los posibles errores y se podrá analizar la usabilidad en cada momento para ir mejorándola continuamente.

4.1.2 Aplicación de la metodología

La metodología ágil scrum está ideada para cualquier equipo de desarrollo. En este caso en particular, al tratarse de un Proyecto de Fin de Carrera, sólo habrá un miembro en el equipo, esto es, compondrá el scrum team entero. Además, siguiendo el Proceso Unificado de Desarrollo (PUD), será el encargado de desarrollar todas las fases del ciclo de vida, hará el papel de analista, desarrollador y tester.

[2] www.proyectosagiles.org/fundamenteos-de-scrum

El desarrollador del sistema es Roberto Pozuelo Domínguez. David Vallejo Fernández actúa como director del proyecto y usuario final de la aplicación, además de ser el product owner del sistema. A lo largo de todo el ciclo de vida se han ido haciendo entregables, y durante las reuniones de fin sprint con el product owner (David Vallejo) se han ido redefiniendo los requisitos del sistema. Además durante estas reuniones se han discutido los requisitos que serán cubiertos durante el siguiente sprint y que formarán parte del siguiente ejecutable. La duración de los sprints ha sido de dos semanas, aunque si los requisitos de un determinado sprint eran muy ambiciosos se posponían a tres semanas.

4.2 Herramientas

A continuación se detallan los recursos, tanto software como hardware, empleados en la construcción del presente proyecto.

4.2.1 Lenguajes de programación

Para la implementación del presente proyecto se han utilizado los siguientes lenguajes:

- **C++**: La gran mayoría del proyecto está codificado en este lenguaje. El lenguaje está orientado a objetos, sin ser un lenguaje orientado a objetos puro, lo que permite una capa de abstracción, encapsulamiento y modularización en la implementación. Además, Una particularidad del C++ es la posibilidad de redefinir los operadores, y de poder crear nuevos tipos que se comporten como tipos fundamentales, lo que hará más fácil el manejo de entidades como robots y objetos creándolas como tipos fundamentales.

- **C**: A la hora de implementar la inteligencia física de los robots (programar los robots internamente) se ha optado por el uso de C, puesto que es el lenguaje principal que utilizan los desarrolladores de sistemas Arduino. A parte es un lenguaje apreciado por la eficiencia del código que produce. Es un lenguaje fuertemente tipificado de medio nivel, pero con muchas características de bajo nivel.

- **Python**: Durante gran parte del ciclo de vida del proyecto se mantuvo la posibilidad de utilizar este lenguaje para la parte de la búsqueda de caminos para la inteligencia de los robots, por ser un lenguaje interpretado y no necesitar de compilación cada vez que se cambiaban los parámetros de búsqueda. Finalmente se descartó esta opción y se optó por el uso de C++ para seguir con la uniformidad y no ser necesario el cambio de parámetros en tiempo de ejecución, sino solo el paso de variables.

4.2.2 Software

A continuación se describen las herramientas y bibliotecas empleadas en la elaboración del proyecto:

Sistema operativo

- **Ubuntu**: Es una distribución del sistema operativo GNU/Linux basada en Debian. En Ubuntu se ha desarrollado la totalidad de la implementación del proyecto. La versión empleada ha sido 12.04, también conocida como Precise Pangolin. Las principales razones de la elección de esta distribución son su estabilidad y la facilidad de uso.
- **Windows**: Este sistema sólo se ha utilizado durante la fase de configuración de los WPM, puesto que la herramienta de configuración solo funcionaba bajo este sistema operativo.

Software de desarrollo

- **Codeblocks**: Se ha empleado este entorno de desarrollo para la implementación del código de los sistemas principales del proyecto. La principal razón por la que se ha utilizado es la facilidad de desarrollo y depuración, además de ser una herramienta optimizada para el desarrollo de aplicaciones en el lenguaje C++.
- **Make**: Es una herramienta que permite la automatización de tareas que resultan repetitivas para el usuario. En el proyecto se ha hace uso de esta herramienta generar todos los archivos auxiliares necesarios para la generación del ejecutable, así como la carga de las bibliotecas auxiliares necesarias y el linkado de estas.
- **TexMaker**: Es herramienta que permite la escritura de documentos en LaTeX de forma sencilla, aportando entras cosas un corrector ortográfico en varios idiomas y una ventana de estructura en forma de árbol desde la que acceder con facilidad a las partes del documento. Además nos permite previsualizar el documento mediante una compilación rápida.
- **Arduino**: Empleado para la programación de los elementos robóticos del sistemas, puesto que están basados en arduino.

Documentación y gráficos

- **LaTeX**: Es un lenguaje que permite la generación y maquetación de documentos técnicos. En el proyecto se ha utilizado para la elaboración de la presente documentación.Además, se han utilizado los paquetes arco-authors2[4] y arco-pfc3[5] que han facilitado la creación de la documentación.
- **Gimp**: Es un editor de gráficos de libre distribución. En la elaboración del proyecto se ha hecho uso de él para la generación de la gran mayoría de las imágenes de la presente documentación.

[4] https://bitbucket.org/arco$_group/arco-authors$

[5] https://bitbucket.org/arco-group/arco-pfc

- **Blender**: Es una herramienta dedicada especialmente al modelado, iluminación, renderizado, animación y creación de gráficos tridimensionales. Se ha hecho uso de esta herramienta para el modelado de los elementos 3D de la escena.
- **Umbrello UML Modeller**: Es una herramienta libre para crear y editar diagramas UML, que ayuda en el proceso del desarrollo de software. Se ha utilizado para crear todos los diagramas.

Interfaz gráfica

- **Glade**: Es una aplicación para el Desarrollo Rápido de Aplicaciones (RAD). Glade permite desarrollar de manera rápida y sencilla interfaces gráficas de usuario para el conjunto de herramientas GTK+ y el entorno de escritorio GNOME.

Bibliotecas

- **Build-Essential**: Es un conjunto de herramientas que permiten la creación de archivos binarios. Incluye la herramienta make, necesaria para la creación del presente proyecto.
- **ARToolKit**: Es una biblioteca que permite la creación de aplicaciones de realidad aumentada. La utilización de esta biblioteca es de gran importancia en la realización del presente proyecto puesto que se utiliza para el seguimiento de los robots.
- **OpenCV**: Es una biblioteca libre de visión artificial. En Sivi es la encargada de realizar el seguimiento de los objetos a vigilar, así como la encargada de mostrar el render de la cámara en la interfaz de usuario.
- **Ogre3D**: Es un motor de renderizado 3D, escrito en el lenguaje de programación C++ y software libre, lo que sigue en consonancia con el resto del proyecto. Es utilizado para mostrar en la interfaz de usuario la información referente al entorno de vigilancia.
- **Ice**: The Internet Communications Engine (ICE) es un conjunto de herramientas que permite crear aplicaciones distribuidas con el mínimo esfuerzo además de poseer un tipo básico de hilos de fácil manipulación, que han sido utilizados en el presente proyecto.

Sistema de control de versiones

Se ha utilizado un sistema de control de versiones **Git**[6] para gestionar los cambios y actualizaciones del software desarrollado. Un sistema de control de versiones permite la gestión de cambios realizados sobre los diferentes elementos que componen un producto.

- Los servicios más importantes que debe proveer un sistema de control de versiones son los siguientes:

[6] http://git-scm.com/

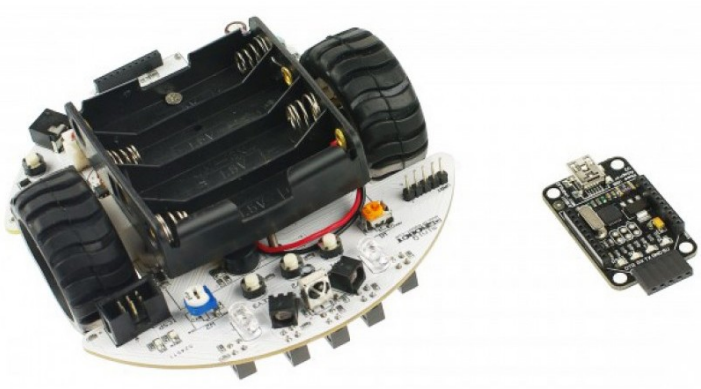

Figura 4.3: Robot MiniQ 2WD Complete Kit.

- Almacenamiento de elementos que componen un producto y sobre los que se, a priori, realizarán los cambios a gestionar.
- Realización de operaciones sobre los elementos almacenados en el sistema como: añadir o eliminar elementos, modificarlos o renombrarlos, entre otras opciones.

Subsistema de registro que permita almacenar las acciones llevadas a cabo en cada uno de los elementos, siendo posible extraer un estado anterior del producto.

En el campo de la ingeniería del software este tipo de herramientas tienen una importancia capital. Los sistemas de control de versiones resuelven el problema que supone el controlar las distintas versiones del código fuente. Es común que muchas de las organizaciones que desarrollan software tengan diferentes versiones, por ejemplo; una sobre la que se desarrollan nuevas funcionalidades, otra que es la que se ofrece al usuario y que está correctamente probada y otra versión que esté en periodo de pruebas.

En especial el servidor utilizado para el sistema de control de versiones e Github[7] donde aloca el proyecto entero.

4.2.3 Hardware

Para el desarrollo del presente proyecto se ha utilizado un computador personal portátil Sony Vaio VPCCW2S8E que posee un procesador Intel® Core™ i3 con cuatro núcleos a 2.13GHz y una memoria RAM de 3,9 GiB. El computador posee una tarjeta gráfica dedicada Nvidia GeForce 330M.

Desde el punto de vista del sistema se han utilizado tres robots MiniQ 2WD Complete Kit (figura 4.3) basados en arduino y seis chips para la programación y la comunicación inalámbrica WPM, todo ello de DFRobot.

También se ha utilizado una cámara web Trust Ceptor HD 720p para la obtención de la información mediante vídeo.

[7] https://github.com

Capítulo 5

Arquitectura

En este capítulo se describe detalladamente la arquitectura en la que se basa el presente Proyecto de Fin de Carrera. Se trata de una arquitectura modular y adaptativa, es decir, una arquitectura dividida en módulos interconectados entre sí, sobre la cuál se pueden realizar cambios e incorporar nuevas funcionales de una forma sencilla.

El detalle de la arquitectura se dará siguiendo una visión top-down, lo que quiere decir que se partirá de un nivel de detalle general a partir del cual se irá entrando en profundidad hasta llegar a un nivel más específico.

Cabe recordar que el objetivo principal del proyecto es el desarrollo de un Sistema Inteligente multi-robot para la mejora de los sistemas de vigilancia modernos, ofreciendo una herramienta tanto gráfica como física que facilite dicha tarea de vigilancia.

5.1 Descripción general

5.1.1 Arquitectura

La arquitectura del sistema (ver figura 5.1) está compuesta por un total de cinco módulos: Módulo de representación de la información, Módulo de análisis de vídeo y obtención de la información, Módulo de localización y posicionamiento, Módulo de comunicación y Módulo de coordinación y gestión del conocimiento. A continuación se definirán estos módulos para entender mejor las partes que componen el sistema.

- **Módulo de análisis de vídeo y obtención de información**: Es el encargado de obtener toda la información del entorno necesaria para poder tomar decisiones. La información se obtiene de una cámara cenital, capaz de enviar toda la información. La información se extrae mediante el análisis de vídeo gracias a las bibliotecas de OpenCV y de realidad aumentada ARToolKit.
- **Módulo de localización y posicionamiento**: Es el encargado de transformar la información obtenida en el módulo de obtención de información y convertirla en información útil para el sistema, como son las posiciones de los objetivos y de los robots disponibles, y las posiciones en las cuáles tendrían que situarse los robots para mantener el sistema protegido.

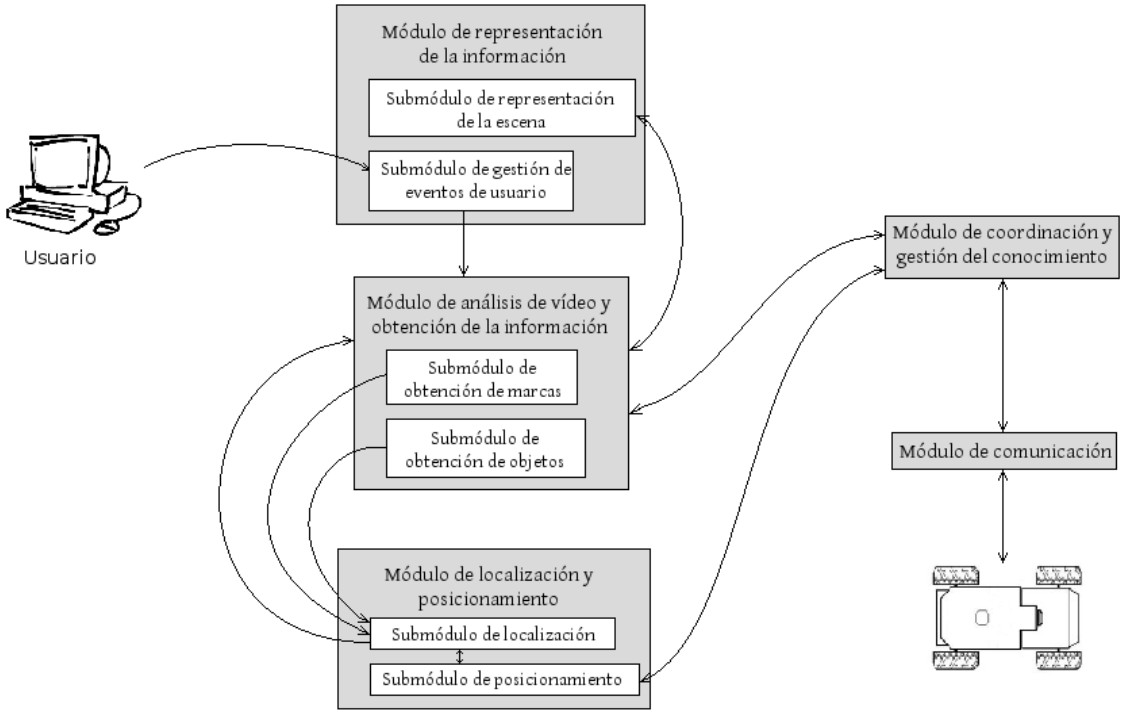

Figura 5.1: Arquitectura del sistema.

- **Módulo de coordinación y gestión del conocimiento**: Es el módulo principal del proyecto. Es el encargado de coordinar a los robots, decidiendo cuál es el robot que se situará en cada posición y qué recorrido tendrá que realizar, así como coordinar a los distintos elementos móviles para no colisionar entre ellos mismos y con los objetos. Para ello se comunicará con los robots a través del módulo de comunicación.

- **Módulo de comunicación**: Es el encargado de gestionar toda la comunicación existe entre los robots y el sistema, tanto de entrada como de salida. Se hace necesario este módulo por la necesidad de coordinar la comunicación entre los robots y abstraer al módulo de coordinación de la lógica propia de los robots proveiendo una interfaz de actuación.

- **Módulo de representación de la información**: Es el encargado de representar la información de manera gráfica de una forma atractiva y sencilla para que el usuario pueda interactuar con la herramienta. Es el encargado de mostrar de mostrar las imágenes tomadas por la cámara y la información que solicite el usuario referente a los objetos y los robots.

A continuación, una vez definidos los módulos que componen el sistema, se puede ver una descripción del flujo de trabajo del sistema, desde que se despliega hasta que se alcanzan los objetivos de vigilancia, como visión general del sistema, en el que se describen paso a paso los eventos encontrados y sus soluciones.

1. En un primer momento la cámara captura una imagen de la escena completa a través del módulo de análisis de vídeo y obtención de la información. En ese momento mientras se procesa la imagen pasa por el submódulo de obtención de objetos, puesto que se aprovecha la pasada de procesamiento de la imagen para buscar objetos de colores.
2. Mientras se acaba el procesamiento de la imagen se pasa al submódulo la información obtenida para localizar los objetos de colores. Una vez localizados en el espacio se pasa esa información al módulo de análisis de vídeo y obtención de la información.
3. Una vez procesada la imagen, esta pasa al módulo de representación de la información, más concretamente al submódulo de representación de la escena, para mostrar por la interfaz de usuario la escena, la imagen captada por la cámara.
4. Después se devuelve el flujo al módulo de análisis de vídeo y obtención de la información para que invoque la funcionalidad del submódulo de obtención de marcas.
5. Una vez obtenidas las marcas pasará lo mismo que con los objetos, se enviará esa información al submódulo de localización (dentro del módulo de localización y posicionamiento) para que devuelva la información obtenida relativa a la localización en la escena de las marcas asociadas a los robots.
6. Cuando ya se tiene la imagen completamente procesada y se ha obtenido toda la información, se tienen localizados los objetos y las marcas, el módulo de análisis de vídeo y obtención de la información notificará al módulo de coordinación y gestión del conocimiento el evento detectado.
7. Una vez que el módulo de coordinación y gestión del conocimiento ha procesado el evento y ha decidido la acción realizar por los robots localizados en la escena, llama al submódulo de posicionamiento con la información de cada robot y la acción a realizar.
8. Este submódulo de posicionamiento se encarga de decidir la lista de movimientos que deberá realizar cada robot para realizar con éxito la acción indicada por el módulo de coordinación y gestión del conocimiento, y se lo notifica al dicho módulo.
9. Es en ese momento cuando el módulo de coordinación (con toda la información ya disponible) le indica al cada robot que movimiento debe realizar a través del módulo de comunicación.
10. Cuando el módulo de comunicación indica que ya se ha realizado el movimiento requerido devuelve el control al módulo de coordinación, que se queda en constante comunicación con el submódulo de posicionamiento, que es el encargado de controlar que los robots sigan la trayectoria indicada.
11. Si en algún momento el robot no sigue la trayectoria indicada, el módulo de coordinación y gestión del conocimiento se comunicaría con el módulo de análisis de vídeo y obtención de la información para que lo localice y entonces volver a llamar al submó-

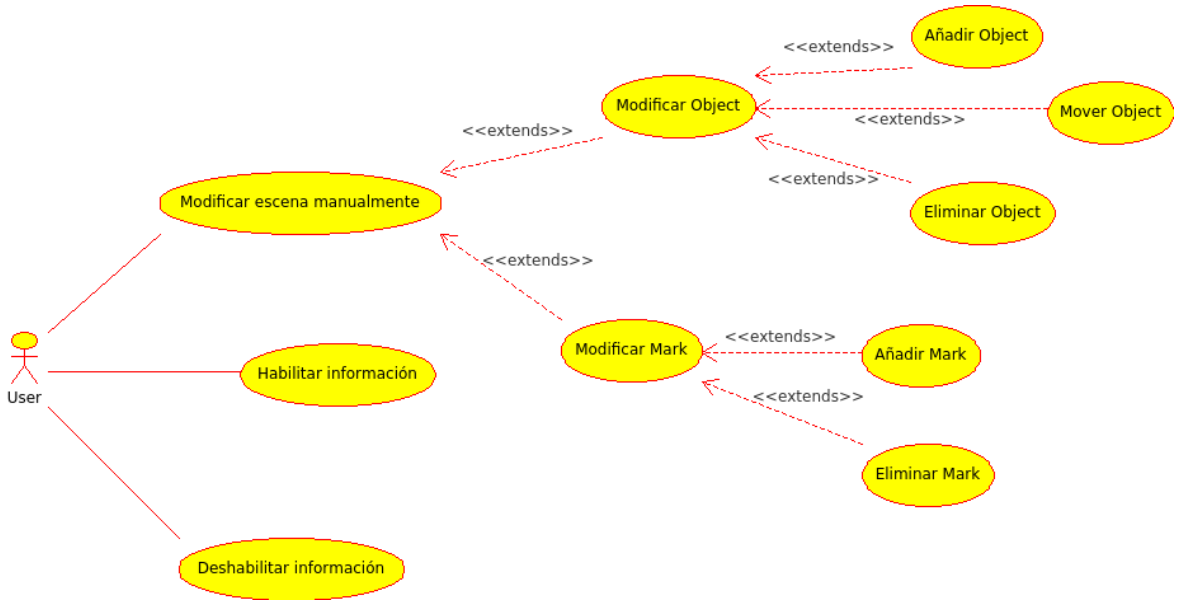

Figura 5.2: Diagrama de casos de uso.

dulo de posicionamiento con la nueva información para que vuelva a hacer los cálculos de trayectorias.

12. De la misma manera, si se detecta una situación de riesgo de colisión entre robots, el módulo de coordinación y gestión del conocimiento le indicará a unos robots que se detengan (a través del módulo de comunicación) y para el más prioritario realizará un recálculo de trayectoria (a través del submódulo de posicionamiento).

13. Además en cualquier momento se puede detectar un evento de usuario (un click en la interfaz de usuario), en cuyo caso el submódulo de gestión de eventos de usuario le pedirá al módulo de análisis de vídeo y obtención de la información la información referente a la posición del click (si es un objeto, o un robot y su información).

14. Después se notificaría esa información al submódulo de representación de la escena para que muestre dicha información, en caso de haber hecho click sobre un elemento de la escena, esto es, un robot o un objeto de color.

Este sería el esquema del flujo que sigue la aplicación a lo largo de cada iteración, puesto que el sistema se encuentra en constante procesamiento de imágenes, mientras la cámara siga en funcionamiento, y se considera una iteración el procesamiento de una sola imagen.

5.1.2 Diagrama de casos de uso

Los distintos módulos dotan a la arquitectura de la funcionalidad necesaria para cubrir las necesidades de interacción con el usuario definidos en los distintos casos de uso (figura 5.2).

La especificación de los casos de uso es la siguiente:

- **Modificar escena manualmente**: El usuario puede interactuar con el sistema modificando la escena manualmente, con lo que consigue modificar su comportamiento. Una vez que el usuario a modificado la escena, el sistema se coordina automáticamente para conseguir que el entorno esté vigilado. Para modificar la escena el usuario puede realizar una de las siguientes acciones:

  - **Modificar Object**: Cuando se realiza esta acción, nos encontramos con que un elemento de tipo Object ha sido modificado, lo que altera la escena. Esta modificación puede ser por cualquiera de sus especializaciones:

    Añadir Object: En cualquier momento, un usuario puede decidir añadir un nuevo objeto en la escena. Cuando esto suceda, el sistema lo detectará automáticamente y tomará las decisiones pertinentes para, en el caso en que se pueda y siempre dependiendo de la prioridad, vaya un robot a vigilar dicho objeto. Si hay un robot ocioso, le manda vigilar el objeto. Si no hay ninguno ocioso pero existe alguno cuya prioridad de su objetivo es menor que la del nuevo objeto, entonces cambia de objetivo.

    Mover Object: Si un usuario desea mover un objeto dentro del entorno puede hacerlo en todo momento. En ese caso el sistema detectará que el objeto ha cambiado de posición y redirigirá al robot que lo vigila actualmente, en caso de que lo haya.

    Eliminar Object: Cuando se realiza la acción de eliminar un objeto, el sistema tiene que adaptarse a la nueva situación. Primero comprueba si ese objeto estaba vigilado. Si lo estaba entonces elimina su referencia del robot que lo vigilaba y busca en la escena otro objeto sin vigilar. Si existe algún objeto sin vigilar entonces le pasa la referencia al robot y lo manda vigilar, sino lo manda de guardia, que es dando vueltas por el perímetro..

  - **Modificar Mark**: La acción de modificar Mark implica que alguna marca ha cambiado en la escena, de manera manual. Esto implica que se ha añadido o eliminado algún robot, con lo que es sistema tiene que planificarse con la nueva situación. Esta acción se realiza a través de cualquiera de sus dos especializaciones:

    Añadir Mark: Si un usuario decide introducir un robot en el sistema, este lo reconocerá automáticamente cuando detecte su marca. En ese momento puede pasar lo siguiente: que exista un objeto sin vigilar en cuyo caso le pasará la referencia al nuevo robot y le mandará que lo vigile, o que no existan objetos a vigilar, en cuyo caso se mandará al robot que haga la ruta de guardia por el perímetro. En cualquier caso, al introducir un nuevo robot (marca que lo identifica) el sistema lo orientará diciéndole lo que hacer.

Eliminar Mark: En el caso de que el usuario decida prescindir de un robot, lo que hará será eliminarlo de la escena eliminando su marca. Una vez que el sistema deje de detectar la marca del robot, lo marcará como no visible y dejará de mandarle órdenes. En ese momento comprobará si tenia algún objeto en vigilancia o por el contrario estaba de guardia. Si estaba vigilando un objeto, dejará de vigilarlo y perderá su referencia.

- **Habilitar información**: Al ejecutar este caso de uso, el usuario podrá visualizar la información referente a un elemento del sistema. A través de un click en el ratón, el usuario puede escoger uno de los elementos clave del sistema (Objetos o Robots) para obtener información de ellos, como la posición, o datos específicos como la prioridad de un objeto o el ángulo de rotación de un robot. Este caso de uso solo está disponible cuando no se esté mostrando ninguna información, puesto que solo se puede mostrar información de un elemento a la vez para no sobrecargar el interfaz gráfico.
- **Deshabilitar información**: Para poder realizar esta acción debe de estar activo el cuadro de información. Cuando el usuario pinche con el botón derecho del ratón, la información que se está mostrando en ese momento desaparecerá y solo se mostrará la salida de vídeo enviada por la cámara.

5.1.3 Diagrama de clases

El diagrama de clases de la arquitectura es el que se puede apreciar en la figura 5.3. Con el objetivo de no sobrecargar el diagrama, no se muestran los atributos y métodos de las clases. A continuación se hace una breve descripción de las clases que componen el presente proyecto, como una primera aproximación en la que solo de enumeran las clase. Su descripción completa se describirá en las sucesivas secciones:

- Clase **Sivi**: Esta clase es el main del sistema. Es la encargada de crear y mantener una instancia de OgreWindow con la que arrancar el sistema.
- Clase **OgreWindow**: Esta clase es la encargada de mantener la interfaz gráfica del usuario.
- Clase **OgreWidget**: Esta clase está definida como un widget para mostrar en una interfaz gtk la salida desde el motor de render Ogre3D. Muestra la salida de vídeo de la cámara además de la información de los elemento de la escena.
- Clase **ARTKDetector**: Esta clase se encarga de buscar en el entorno las marcas, utilizadas en realidad aumentada, y de posicionarlas en dicho entorno.
- Clase **VideoManager**: Esta clase se encarga de procesar las imágenes de vídeo en busca de objetos además de mandar las imágenes al OgreWidget.
- Clase **Scene**: Esta clase es la encargada de almacenar toda la información relativa a la escena.

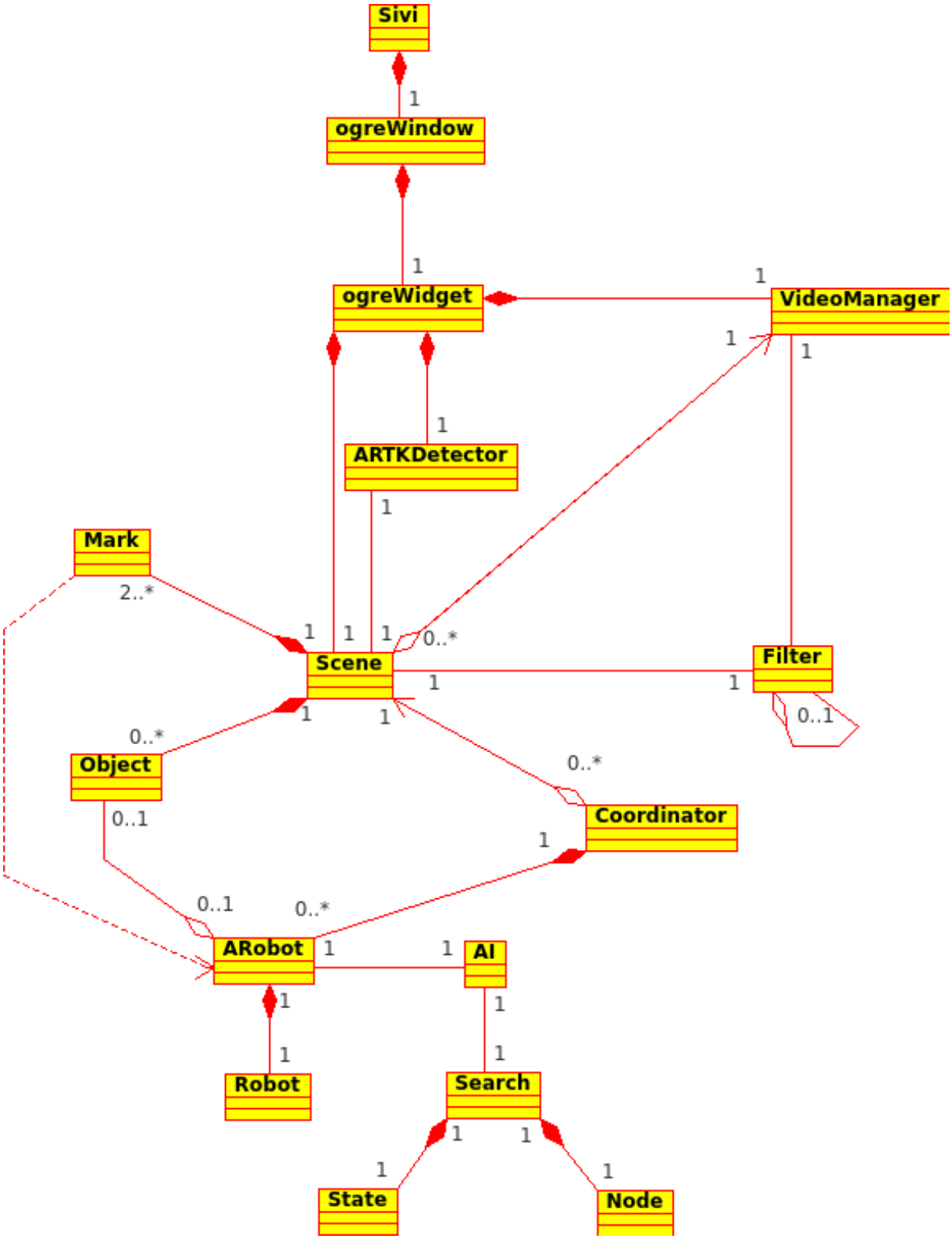

Figura 5.3: Diagrama de clases.

- Clase **Filter**: Esta clase define la funcionalidad necesaria para establecer en el sistema el filtro encargado de delimitar el espacio de actuación de los robots.
- Clase **Mark**: Esta clase es la encargada de almacenar la información relativa a las marcas de realidad aumentada detectadas las cuales se asociarán a los robots y las marcas de delimitación.
- Clase **Object**: Esta clase almacena la información referente los objetos encontrados en el entorno a vigilar.
- Clase **Coordinator**: Esta clase es la encargada de manejar la lógica de coordinación básica, puesto que es la encargada de asignar los objetos a cada robot además de controlar y evitar fallos en el sistema.
- Clase **ARobot**: Esta clase define la lógica de los agentes de los robots.
- Clase **Robot**: Esta clase hace de interfaz entre los elementos físicos y el sistema.
- Clase **AI**: Esta clase es la encargada de lanzar la búsqueda de caminos.
- Clase **Search**: Esta clase es la encargada de hacer la búsqueda del camino a seguir por los robots para llegar hasta una posición determinada.
- Clase **State**: Esta clase almacena la posición en x e y actual, además del movimiento seguido para llegar a esa posición.
- Clase **Node**: Esta clase define las propiedades de los elementos necesarios que se utilizarán en el pathfinding.

5.2 Módulo de representación de la información

La finalidad de este módulo consiste en presentar al usuario la información de una manera atractiva y sencilla. Este módulo está compuesto por dos submódulos que cooperan para representar la información de una forma dinámica.

Este módulo se compone no sólo del código fuente dedicado a la interfaz gráfica, sino que también incluye la parte de renderizado de la escena y el uso de realidad aumentada. Para la parte de la interfaz de utiliza la biblioteca de GTK, mientras que pare el renderizado de la escena se utilizan tanto las bibliotecas de OpenCV como las de Ogre3D.

5.2.1 Submódulo de representación de la escena

Para la representación de la escena se hace necesaria la creación de un submódulo, puesto que se tienen integrar varias tecnologías para lograrlo. Una de ellas es la biblioteca gtkmm, que es la interfaz C++ oficial para la popular biblioteca de interfaz gráfica GTK+. Además esta biblioteca es de software libre distribuido bajo licencia Lesser General Public License (LGPL). Otra es la biblioteca de OpenCV, que es una biblioteca libre de visión artificial que contiene más de 500 funciones que abarcas una gran gama de áreas del proceso de

visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras o visión estéreo (la extracción de información 3D de imágenes digitales). Por último tenemos la biblioteca de Ogre3D, que es un motor de renderizado 3D orientado a escenas, escrito en C++, diseñado para facilitar la producción de aplicaciones que utilizan gráficos 3D acelerados por hardware.

Como se ha visto, se utilizan tres bibliotecas diferentes, cada una con su finalidad específica y diferente, para la representación de la escena. Esto hace que su integración no sea sencilla ni intuitiva, de ahí la creación de este submódulo. Por ello la integración se hace por pares, encapsulando la funcionalidad de una biblioteca dentro de la siguiente.

```
1  // createBackground: Crea el plano sobre el que dibuja el vídeo
2  void VideoManager::createBackground(int cols, int rows){
3  Ogre::TexturePtr texture=Ogre::TextureManager::getSingleton().
4    createManual("BackgroundTex", // Nombre de la textura
5    Ogre::ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME,
6    Ogre::TEX_TYPE_2D, // Tipo de la textura
7    cols, rows, 0, // Filas, columas y Numero de Mipmaps
8    Ogre::PF_BYTE_BGRA,
9    Ogre::HardwareBuffer::HBU_DYNAMIC_WRITE_ONLY_DISCARDABLE);

11 Ogre::MaterialPtr mat = Ogre::MaterialManager::getSingleton().
12   create("Background",
13   Ogre::ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME);

15 mat->getTechnique(0)->getPass(0)->createTextureUnitState();
16 mat->getTechnique(0)->getPass(0)->setDepthCheckEnabled(false);
17 mat->getTechnique(0)->getPass(0)->setDepthWriteEnabled(false);
18 mat->getTechnique(0)->getPass(0)->setLightingEnabled(false);
19 mat->getTechnique(0)->getPass(0)->getTextureUnitState(0)->
      setTextureName("BackgroundTex");

21 // Creamos un rectangulo que cubra toda la pantalla
22 Ogre::Rectangle2D* rect = new Ogre::Rectangle2D(true);
23 rect->setCorners(-1.0, 1.0, 1.0, -1.0);
24 rect->setMaterial("Background");

26 // Dibujamos el background antes que nada
27 rect->setRenderQueueGroup(Ogre::RENDER_QUEUE_BACKGROUND);
28 Ogre::SceneNode* node = _sceneManager->getRootSceneNode()->
      createChildSceneNode("BackgroundNode");
29 node->attachObject(rect);
30 }
```

Listado 5.1: «Integración del vídeo desde OpenCV con render a textura en Ogre3D»

Lo primero a lo que hay que enfrentarse es a la integración entre OpenCV y Ogre3D. Al ser Ogre3D un motor de render y OpenCV la biblioteca para la visión por computador, vamos a encapsular esta última dentro de Ogre3D. Para ello lo primero que se ha hecho es crear una textura, del tamaño de la pantalla. Después se crea un material al que se le asigna la textura creada para finalmente crear un rectángulo que cubra toda la pantalla y al que le añadimos el material anteriormente creado (véase el listado 5.1).

Una vez que se tiene creado el rectángulo de background, sobre el que se van a renderizar las imágenes de la cámara, hay que obtener el puntero a la textura en cada frame e ir asignándole los valores de cada pixel (véase el listado 5.2). Con esto se consigue que dentro de una ventana de ogre tengamos de fondo el render de un vídeo obtenido por la cámara mediante la biblioteca de OpenCV.

```
1 Ogre::TexturePtr tex = Ogre::TextureManager::getSingleton().
     getByName("BackgroundTex",Ogre::ResourceGroupManager::
     DEFAULT_RESOURCE_GROUP_NAME);
2 Ogre::HardwarePixelBufferSharedPtr pBuffer = tex->getBuffer();
3 pBuffer->lock(Ogre::HardwareBuffer::HBL_DISCARD);
4 const Ogre::PixelBox& pixelBox = pBuffer->getCurrentLock();
5 Ogre::uint8* pDest = static_cast<Ogre::uint8*>(pixelBox.data);
6   .. // Se pinta cada valor del pixel obtenido por la cámara el el pixel
       destino de la textura
7 pBuffer->unlock();
```

Listado 5.2: «Render a textura de las imágenes capturadas»

El siguiente paso es de integrar una ventana de Ogre3D dentro de una interfaz gráfica creada con GTK. Actualmente dentro de la variedad de widgets disponibles en GTK no existe un widget propiamente dicho que integre una ventana de Ogre3D dentro de la interfaz gráfica. Por ello se ha tenido que crear un widget que implemente esta funcionalidad. En el listado 5.3 podemos ver la estructura de la clase OgreWidget, la cual es la encargada de implementar esta funcionalidad. Para ello se tienen que sobrecargar las funciones básicas de la clase widget de GKT con la funcionalidad requerida, que el caso de SIVI es la de crear una escena de Ogre3D, conectando la señal Glib::signal_idle con la de Ogre3D (listado 5.4).

Las clases que definen la funcionalidad de este submódulo (véase la figura 5.4) son las clases OgreWindow y OgreWidget, que son las encargadas de la interfaz gráfica, la clase Scene, al ser la clase que almacena toda la información de la escena y la clase VideoManager que es la encargada de sintetizar el vídeo dentro de una textura. La clase OgreWidget es la que integra Ogre3D con GTK para mostrar la ventana de Ogre dentro de una interfaz de usuario diseñada con GTK mientras que la clase VideoManager es la encargada de integrar las capturas de las imágenes de vídeo de OpenCV con la escena de Ogre3D.

```
class OgreWidget : public Gtk::Widget
{
public:
  OgreWidget();
  virtual ~OgreWidget();
protected:
  void loadResources();
  void createScene();
  virtual void on_size_request(Gtk::Requisition* requisition);
  virtual void on_size_allocate(Gtk::Allocation& allocation);
  virtual void on_map();
  virtual void on_unmap();
  virtual void on_realize();
  virtual void on_unrealize();
  virtual bool on_expose_event(GdkEventExpose* event);
  virtual bool on_idle();
  virtual bool on_motion_notify_event(GdkEventMotion *event);
  virtual bool on_button_press_event(GdkEventButton *event);
  virtual bool on_button_release_event(GdkEventButton *event);
  Glib::RefPtr<Gdk::Window> mRefGdkWindow;
  Ogre::RenderWindow* mRenderWindow;
  Ogre::SceneManager* mSceneMgr;
  Ogre::Viewport* mViewport;
  Ogre::Camera* mCamera;
};
```

Listado 5.3: «Estructura de la clase OgreWidget»

```
// Start idle function for frame update/rendering
Glib::signal_idle().connect(sigc::mem_fun(*this,&OgreWidget::
   on_idle));
```

Listado 5.4: «Función de conexión de los idles»

A continuación se especifican en detalle las clases involucradas:

- Clase **OgreWindow**: Esta clase es la encargada de mantener la interfaz gráfica del usuario. Mantiene una instancia de un widget de Ogre3D desde el que se mostrará el vídeo recibido por la cámara. Además mantiene un control sobre el teclado para habilitar.
- Clase **OgreWidget**: Esta clase está definida como un widget para mostrar en una interfaz gtk la salida desde el motor de render Ogre3D. Muestra las imágenes obtenidas desde la cámara mediante un render a textura en un plano además de la posible información que usuario necesite referente al entorno. También tiene el control de los eventos de click del ratón para controlar la zona en la que el usuario pincha para obtener información.

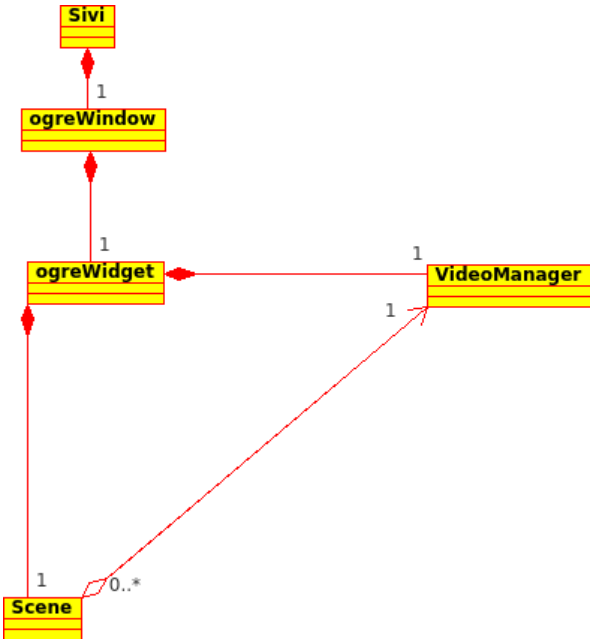

Figura 5.4: Diagrama de clases del submódulo de representación.

- Clase **VideoManager**: Esta clase procesa las imágenes devueltas por la cámara y las va dibujando pixel a pixel en un rectángulo de Ogre3D para poder ver desde la interfaz de usuario las imágenes de vídeo. Además si aplicando el filtro. el pixel se encuentra dentro del entorno, comprueba si su color corresponde con algunos de los guardados como color de objetivo, en cuyo caso hace un conteo de los pixels de dicho color. Después de procesar todos los pixels del frame, si ha detectado algún color, creará un objeto de tipo Object con su posición, sus puntos máximos y mínimos, su prioridad y su id.
- Clase **Scene**: Esta clase es la encargada de almacenar toda la información relativa a la escena. En esta clase se almacenan todos los objetos que se han descubierto en el entorno, en un vector de Objects, y todas las marcas en otro vector, de Marks. Esta clase es la encargada de comunicarse con el coordinador cada vez que haya habido un cambio en la escena, para aumentar la eficiencia y no estar haciendo comprobaciones constantemente. Almacena los valores del tamaño de la escena (el alto y el ancho del vídeo) y del entorno (posiciones de las marcas de delimitación del entorno) además del tamaño de las celdas del grid utilizado para el cálculo de trayectorias.

5.2.2 Submódulo de gestión de eventos de usuario

A la hora de integrar la interacción con el usuario se utiliza un submódulo de la representación de la información, llamado submódulo de gestión de eventos de usuario.

El usuario puede interaccionar con el sistema de dos maneras. Una de ellas es mediante la introducción de elementos en la escena, lo que genera un evento que el sistema trata como evento interno del sistema. En este caso el sistema automáticamente seguirá su curso, como si

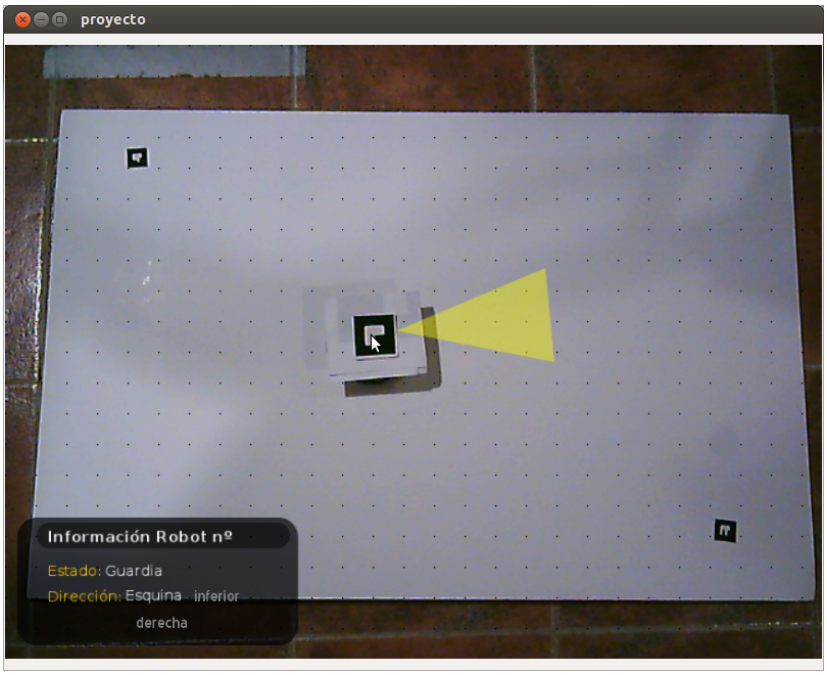

Figura 5.5: Con información(evento de pulsación sobre robot)

el usuario no hubiera hecho nada. La otra opción es mediante clicks en la interfaz de usuario. En este caso, el sistema registra los eventos como eventos externos, con lo que se lanza una lógica que genere la información requerida y la muestre o elimine según el caso.

Si el click se produce con el botón izquierdo del ratón, el sistema detecta el evento como pedida de información, y dependiendo de la posición sobre la que haya hecho el click mostrará la información de un objeto de un robot (siempre que el click se haya hecho sobre ese elemento).

Si el click ha sido con el botón derecho entonces el sistema detectará el evento como evento de deshabilitación de información, y eliminará de la interfaz de usuario el cuadro de diálogo con la información pertinente.

En las figuras 5.5 y 5.6 se puede observar cómo aparece una ventana con información relativa al robot cuando se pincha sobre él (imagen izquierda) y cómo desaparece dicho elemento cuando el usuario pincha con el botón derecho en cualquier posición de la escena (imagen derecha).

La clase encargada de manejar los eventos es la clase OgreWidget. Ésta es la clase manejadora porque es la que integra el widget con la ventana de ogre y el evento debe detectar sobre la escena en qué posición se ha pinchado para saber sobre qué elemento ha sido. Además al ser una clase que hereda de un widget, ya tenemos la necesidad de implementar esa funcionalidad. Si se hubiera hecho necesario el manejo de eventos de teclado, estos se controlarían en la clase OgeWindow, puesto que ese evento no tiene nada que ver la escena en particular.

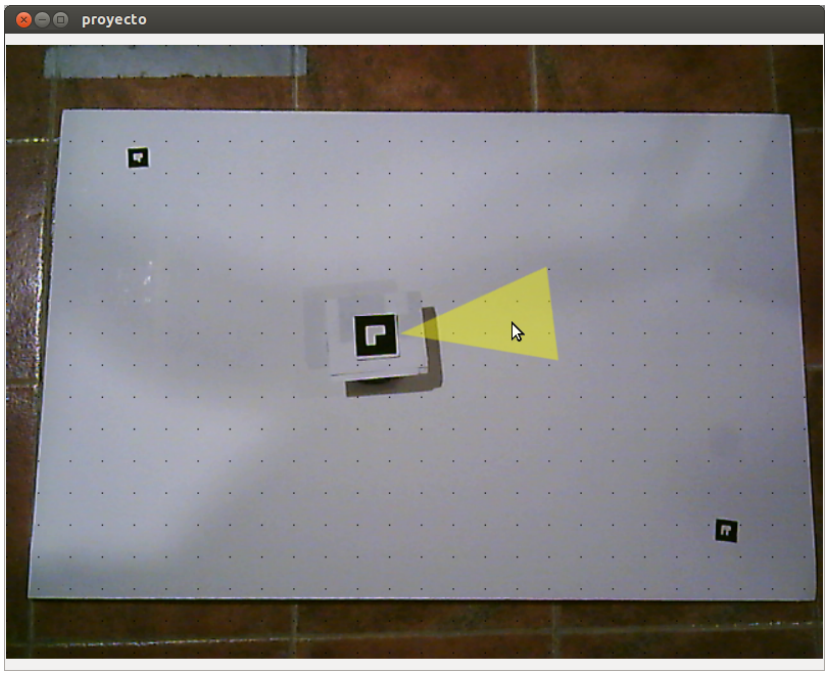

Figura 5.6: Sin información (evento de pulsación con botón derecho)

La especificación de la clase que integra la funcionalidad de este módulo la podemos encontrar en el anterior apartado 5.2.1.

5.3 Módulo de análisis de vídeo y obtención de la información

Este módulo es el encargado analizar todos los frames que va generando la cámara, y a partir de esos frames obtener la información importante para el sistema. Este módulo no provee de ninguna lógica al sistema, su única función es la de captar la información del entorno.

El sistema es capaz de captar en cada frame todos los pixels de colores y todas las marcas utilizadas en realidad aumentada. La funcionalidad de este módulo se divide a su vez en dos submódulos.

5.3.1 Submódulo de obtención de marcas

La primera pasada que se hace en cada frame es la de detección de marcas. Para ello el sistema hace uso de las bibliotecas de ARToolKit, que provee de la funcionalidad necesaria para la detección de marcas. Esta funcionalidad la hace la clase ARTKDetector, que busca en cada frame los patrones que se han cargado al iniciar el sistema, y en caso de encontrar una marca, actualiza sus valores.

Al iniciar el detector de marcas, la clase lee los parámetros guardados relativos a la calibración de la cámara, y se cargan los patrones de las marcas.

En cada frame se comprueba si se ha detectado cada marca, como podemos ver en el listado 5.5, donde se comprueba si hay alguna marca en el frame actual, guardando en _markerNum el número de marcas detectadas. Después se comprueba por cada marca si se ha encontrado su patrón, en cuyo caso actualizará la información de dicha marca con la obtenida en el patrón.

```
2  if(arDetectMarker(frame->data, _thres, &_markerInfo, &
       _markerNum) < 0){
3    return _detected;
4  }
5  for (i=0; i<_nMarks; i++) { //Por cada marca comprobamos si se
       encontro su patron
6    _mark = marks[i];
7    for(j = 0, k = -1; j < _markerNum; j++) {
8      if(_mark->getId() == _markerInfo[j].id) { //Si el id del patron
           coincide con el id de la marca actual
9        if (k == -1) k = j;
10       else if(_markerInfo[k].cf < _markerInfo[j].cf) k = j;
11     }
12   }
13   ..
```

Listado 5.5: «Detección de marcas»

En la figura 5.7 se puede observar como se detecta la marca superior izquierda, en modo debug, mediante la obtención del contorno de la marca, dibujada de amarillo.

Para la implementación de este submódulo se ha seguido el esquema descrito en la figura 5.8. En este caso las clases Scene y ARTKDetector se pasan información necesaria para las dos partes. Por una parte la clase ARTKDetector pide a la escena los patrones que tiene que buscar al arrancar la aplicación, para mantener el detector como un módulo a parte y mantener el desacoplamiento. Además es le detector el que notifica a la escena cuando ocurre algún evento, de manera que no esté siempre 'pidiendo' información la escena y así aumentar la eficiencia. Para ello la clase ARTKDetector pide información a la escena sobre el estado de las marcas antes de buscar, para tener un mapa actualizado y así efectuar eventos cuando las marcas cambien de estado.

La escena, por otro lado es la clase contenedora de la información de la escena, por lo que tiene que mantener una información actualizada. También necesita mantener la instancia del ARTKDetector para poder normalizar la información, puesto que es el detector el que tiene la funcionalidad de normalizar la información a información que pueda ser leída por Ogre3D.

Estas son las razones de que se haya establecido una relación de asociación entre ellas, con una relación de uno a varios, puesto que aunque una escena solo puede estar en contacto con

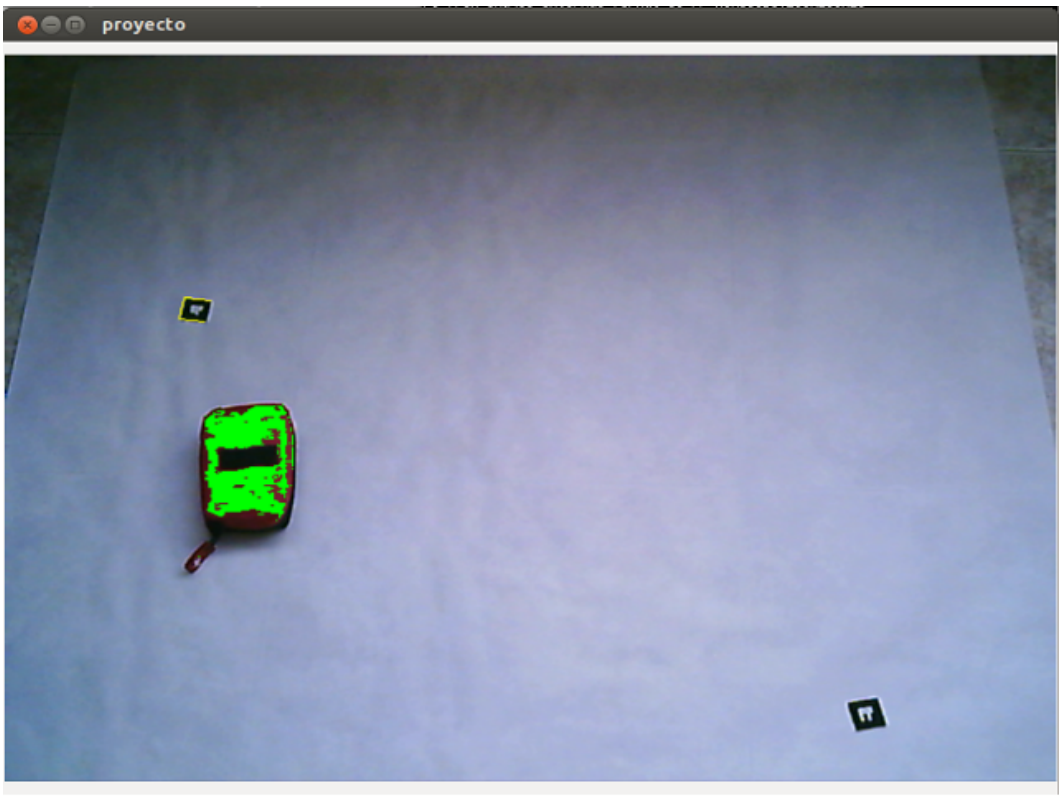

Figura 5.7: Imagen en modo debug de la detección de una marca.

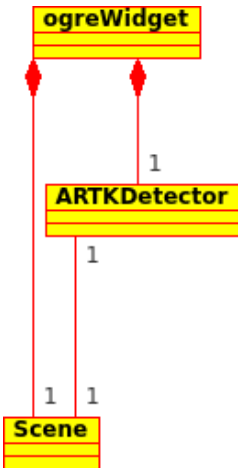

Figura 5.8: Diagrama de clases para la detección de marcas.

un detector, el detector puede tener varias escenas a las que notificar, y de las que necesitará saber qué tiene que buscar.

A continuación se especifican en detalle las clases involucradas:

- Clase **OgreWidget**: Ver apartado 5.2.1.
- Clase **ARTKDetector**: Esta clase se encarga de buscar en el entorno las marcas, utilizadas en realidad aumentada, y de posicionarlas en dicho entorno. Modifica los objetos de tipo Mark de la escena con su posición, además de proveer la funcionalidad del cálculo de la rotación de dicha marca, lo que se hace indispensable para la orientación de los robots, ya que su seguimiento se hace mediante estas marcas. Esta clase es capaz de transformar los valores de rotación y orientación de las marcas en coordenadas de Ogre3D de manera que se puedan colocar bien los objetos de realidad aumentada en la interfaz, integrada en Ogre3D.
- Clase **Scene**: Ver apartado 5.2.1.

5.3.2 Submódulo de obtención de objetos

La segunda pasada que se hace es para la búsqueda de objetos de colores, que formarán los objetivos de los robots, esto es, los objetos a vigilar.

Haciendo uso de las bibliotecas y de la funcionalidad de OpenCV hacemos la pasada al frame pixel a pixel. Lo primero que se hace es comprobar si el pixel se encuentra dentro de los límites del entorno, mediante llamadas a la funcionalidad de la clase Filter, más concretamente a la pertenencia al espacio dentro de las rectas el entorno(véase el listado 5.6). Después se comprueba el valor del pixel en los tres canales, puesto que el valor está en RGB, para ver si corresponde con los valores establecidos para los objetos definidos.

```
1 for(int j=0;j<_frameMat->rows;j++) {
2   for(int i=0;i<_frameMat->cols;i++) {
3     if(_filter->rect_sup(i,j)&&_filter->rect_der(i,j)&&_filter
        ->rect_inf(i,j)&&_filter->rect_izq(i,j)){
4         ..
```

Listado 5.6: «Filtro de los pixels»

Por motivos de eficiencia se ha mantenido el cálculo de los colores en el espacio RGB. Se estudió la posibilidad de hacer los cálculos con el espacio HSV, pero la necesidad de tener que procesar los frames dos veces (la captura normal y la conversión al espacio HSV) hace que aparezca una latencia que disminuye la eficiencia del sistema.

La funcionalidad de este submódulo viene definida por las clases VideoManager y Scene (figura 5.9). Pero la funcionalidad no estaría completa sin el uso del filtro (clase Filter). A la

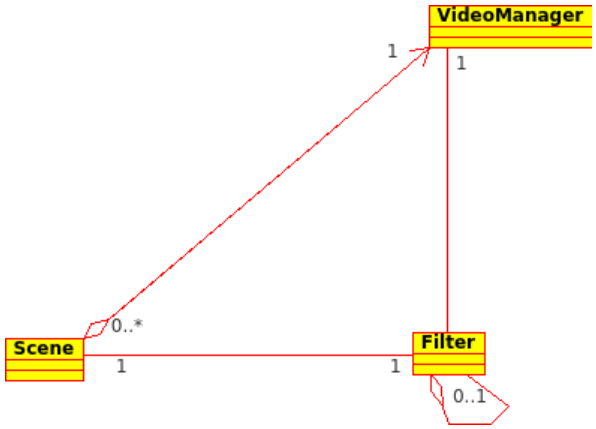

Figura 5.9: Diagrama de clases para la detección de objetos.

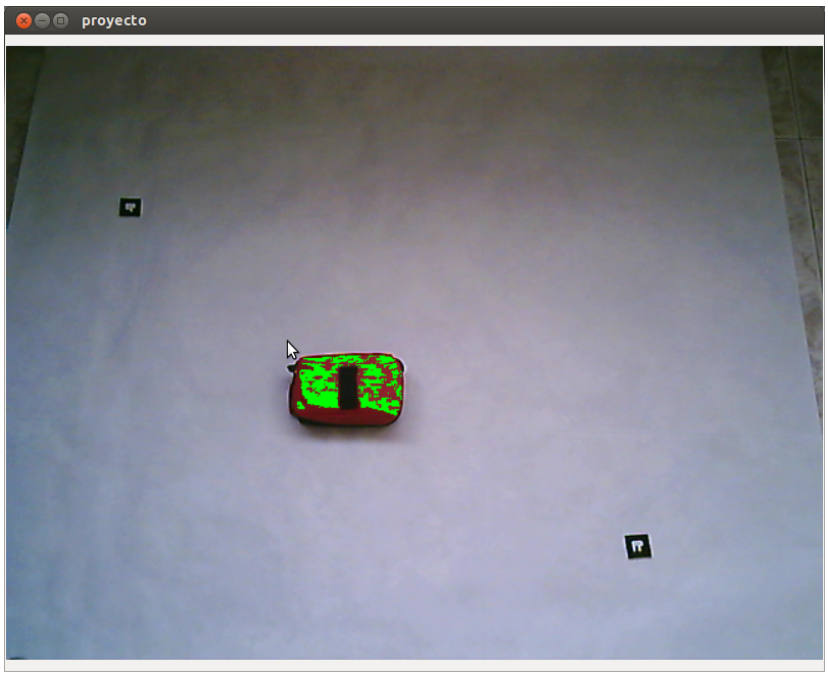

Figura 5.10: Imagen en modo debug de la detección de un objeto.

hora de realizar esta funcionalidad se aprovecha el hecho de tener que dibujar pixel a pixel la imagen de vídeo en la textura de ogre para comprobar cada pixel por separado. Lo primero que se hace es comprobar, mediante el uso del filtro, si el pixel se encuentra dentro del entorno de trabajo del sistema, el cual viene definido por las marca de delimitación situadas en las esquinas. Si el pixel se encuentra dentro del marco de actuación se descompone en los tres canales de que consta cada pixel (en formato RGB).

Una vez tenemos el valor del pixel en los tres canales hacemos la comprobación del color. Para ello se comprueban los valores de cada canal, y si sus valores se encuentran dentro de los limites de cada color entonces se actualizan las variables. En la figura 5.10 podemos observar cómo el sistema ha identificado una cantidad de pixels de un objeto (pintados de verde). Internamente el sistema hará los cálculos para componer el objeto.

En este caso se ha optado por implementar un patrón observador, que implique que cada vez que se detecte en el VideoManager un cambio en el estado de algún objeto, notifique a la clase Scene indicando el evento encontrado y el id del objeto en cuestión. De esta manera si el VideoManager tiene varias escenas a la escucha de la detección de objetos, se les notificará a todos de la misma manera.

Las razones de no utilizar el mismo tipo de relación entre el VideoManager y la Scene que entre el ARTKDetector y la Scene es que para el VideoManager no importa que escena esté pendiente, solo se encarga de identificar objetos de colores, mientras que el ARTKDetector necesita saber a qué escena notificar específicamente, puesto que cada una tiene unas marcas propias, que no tienen que ser las mismas en escenas diferentes.

A continuación se especifican en detalle las clases involucradas:

- Clase **VideoManager**: Ver apartado 5.2.1.
- Clase **Scene**: Ver apartado 5.2.1.
- Clase **Filter**: Esta clase define la funcionalidad necesaria para establecer en el sistema el filtro encargado de delimitar el espacio de actuación de los robots, o lo que es lo mismo, el entorno a vigilar. A partir de los puntos máximos y mínimos del sistema obtenido de los vértices de las marcas de posicionamiento establece los límites mediante el calculo de las rectas que lo delimitan. Esta funcionalidad sirve para que la detección de objetos no tenga lugar fuera del entorno y no se desestabilice el sistema.

5.4 Módulo de localización y posicionamiento

Este módulo se compone de otros dos módulos muy diferenciados pero a su vez muy relacionados entre sí. Estos son el submódulo de localización y el submódulo de posicionamiento. Aunque cada uno tiene una funcionalidad diferente, la cual se explicará con detalle más adelante, ambos están muy relacionados.

Ambos se encuentran relacionados puesto que a la hora de hacer el calculo de trayectorias se necesita tener todos los elementos de la escena bien localizados, para evitar posibles colisiones, tanto entre robots como entre los objetivos y los robots.

A continuación se describen los dos submódulos.

5.4.1 Submódulo de localización

Este submódulo es el encargado de almacenar la localización exacta de todos los elementos de la escena, guardando dicha información en la clase escena.

Todo empieza una vez que se han detectado un objeto o una marca o, en su caso, que ha desaparecido de la escena. En ese momento se notifica a la escena el evento ocurrido. La escena almacena un vector de objetos y un vector de marcas y cuando recibe un evento actualiza esa información.

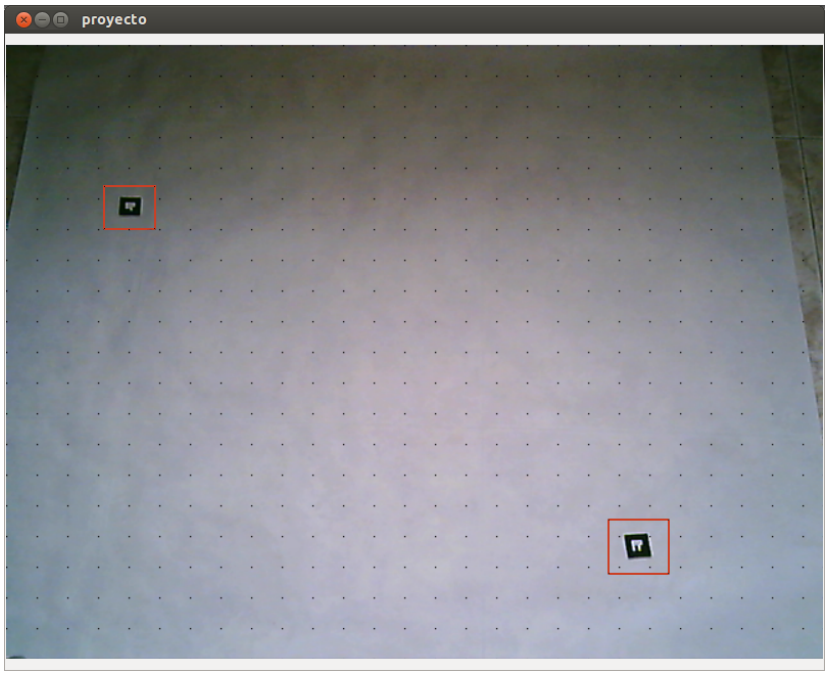

Figura 5.11: Localización de las marcas de delimitación del entorno y grid en modo debug.

Puede recibir eventos de varios tipos, pero se pueden agrupar en dos:

- Eventos de **objeto**: Cuando se recibe un evento relativo a la información de un objeto, la escena se encarga de buscar el objeto en cuestión (o de crearlo si se detecta un objeto nuevo). Una vez se ha localizado el objeto, se actualizan sus valores clave para la localización de dicho objeto, como son sus puntos máximo, mínimo y medio, así como su prioridad.
- Eventos de **marca**: Cuando se recibe un evento de marca lo que se está recibiendo es un cambio referente a un robot. En ese momento el sistema se encarga de buscar la marca correspondiente y de actualizar su información, como es el punto medio, los puntos máximo y mínimo y su orientación.

Esta información clave relativa a la localización se almacena en un mapa interno que mantiene en todo momento la escena. Este mapa esta dividido en cuadriculas, para reducir el espacio de búsqueda. El tamaño del grid lo determina la distancia entre vértices diagonales de las marcas de delimitación.

Durante la primera pasada del sistema se buscan las marcas de delimitación (véase figura 5.11), que definen el entorno de actuación. Estas marcas delimitan la zona de búsqueda de objetivos y además nos sirven de orientación para mantener a los robots ociosos en un estado de guardia, puesto que en ese caso éstos van desde una esquina a otra del entorno. En la misma figura se puede observar como se configura el grid de la escena (el que servirá de base para el mapa interno) en modo debug, sacado de los vértices en diagonal de las marcas de delimitación.

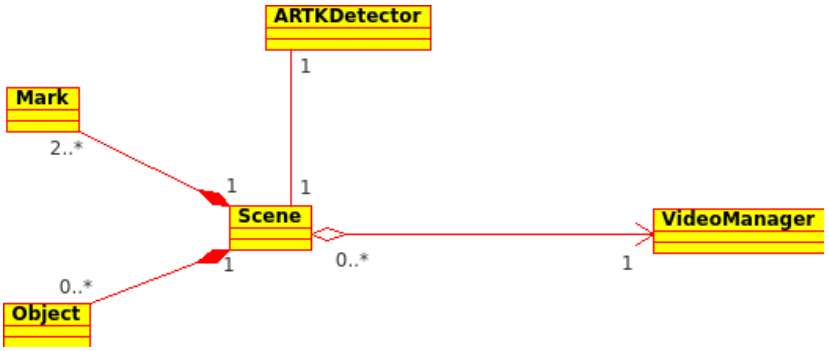

Figura 5.12: Diagrama de clases del submódulo de localización.

La funcionalidad de este submódulo sigue el esquema mostrado en la figura 5.12. Las clases involucradas son las de Mark, Object y las ya mencionadas en el módulo de análisis de vídeo y obtención de información. Puesto que lo primero que se tiene que hacer es la detección de los elementos se involucran dichas clases definidas anteriormente. Sin embargo hay que añadir el echo de necesitar las clases que almacenen dicha información. Para eso se han creado las clases específicas para los objetos y para las marcas. Como los objetos y las marcas tienen información diferente, se crean clases diferentes.

Además es la propia escena la que almacena los objetos de dichas clases, puesto que la funcionalidad principal de la clase Scene es la de almacenar toda la información del entorno. Cuando las clases ARTKDetector y VideoManager detectan un elemento y notifican a la escena mediante un evento, esta se encarga de modificar su mapa interno con los nuevos valores obtenidos. Una vez que se ha modificado el mapa, entonces se modifican los objetos que almacena para mantener la información de manera coherente.

En este submódulo se definen las relaciones de composición entre los objetos Mark y los objetos Objects con respecto a la clase Scene. Esto es así porque la escena se compone de un vector de cada uno de estos elementos, en los que almacena su información. Esos objetos no son utilizados, ni forman parte, de otra clase, con lo cual en el momento en el que desaparece la escena, desaparecen esos objetos Marks y Objects.

A continuación se especifican en detalle las clases involucradas:

- Clase **ARTKDetector**: Ver apartado 5.3.1.
- Clase **VideoManager**: Ver apartado 5.2.1.
- Clase **Scene**: Ver apartado 5.2.1.
- Clase **Mark**: Esta clase es la encargada de almacenar la información relativa a las marcas de realidad aumentada detectadas. Guarda estados como la visibilidad, identificador, la rotación o el markerInfo (clase interna de la biblioteca ARToolKit que guarda toda la información de la marca, como los vértices o la matriz de transformación). También almacena el ancho, el centro y la matriz de transformación asociada al

patrón, así como su posición y sus puntos máximo y mínimo.

- Clase **Object**: Esta clase almacena la información referente los objetos encontrados en el entorno a vigilar. Almacena tanto el identificador único de objeto como la prioridad de dicho objeto (obtenida a partir de color del objeto). También mantiene la información relativa a su posición, esto es el centro del objeto y los puntos máximo y mínimo.

5.4.2 Submódulo de posicionamiento

El submódulo de posicionamiento es el encargado de definir la posición y los movimientos a seguir por los elementos robóticos.

A la hora de localizar los movimientos a seguir y las posiciones finales se ha optado por implementar un algoritmo de búsqueda de caminos mediante A* (primero el mejor) [RN04].

Este algoritmo evalúa los nodos combinando g(n), el coste para alcanzar el nodo siguiente, y h(n), el coste de ir al nodo objetivo (final):

$$f(n) = g(n) + h(n) \tag{5.1}$$

Ya que la gn) nos da el coste del camino desde el nodo inicio al nodo n, y la h(n) el coste estimado del camino más barato desde n al objetivo, tenemos:

$$f(n) = coste\ mas\ barato\ estimado\ a\ traves\ de\ n \tag{5.2}$$

Así, si tratamos de encontrar la solución más barata, es razonable intentar primero el nodo con el valor más bajo de g(n) + h(n). Resulta que esta estrategia es más que razonable: con tal de que la función heurística h(n) satisfaga ciertas condiciones, la búsqueda A* es tanto completa como óptima.

La optimalidad de A* es sencilla de analizar si se usa con la Búsqueda-Árboles. En este caso, A* es óptima si hn) es una **heurística admisible** es decir, con tal de que la hn) nunca sobrestime el coste de alcanzar el objetivo.

Este algoritmo hace uso del mapa interno de la escena, el cual está actualizado y mantiene toda la información de localización de los elementos de la escena.

Para realizar la búsqueda de caminos se utiliza el mapa reducido de la escena simplemente por motivos de eficiencia. Si se utilizara un mapa con los valores originales el algoritmo de búsqueda se podría volver lento y tardar mucho tiempo en calcular la solución.

Por motivos de seguridad y evitar cuellos de botella, los algoritmos se búsqueda (la inte-

ligencia artificial) se lanzan en hilos de ejecución separados. De esta manera se evitan estos cuellos de botella y aumentamos la eficiencia.

Este módulo también es el encargado de comprobar que los robots siguen el camino indicado. Para ello cada vez que llega un evento de movimiento se comprueba si el robot se encuentra en una casilla válida. Un robot se encuentra en una casilla válida si está en la misma casilla en la que estaba antes, o si se encuentra en una casilla adyacente y que se corresponde con la marcada por el movimiento.

Mientras los robots sigan se muevan a través de casillas válidas el sistema estará actuando con normalidad. En el caso en que el submódulo de localización nos indique que la posición del robot no corresponde con la posición esperada, se encuentra en una casilla no valida, se realiza un recálculo del camino, para reorientar al robot 'descarriado'.

A la hora de realizar el pathfinding, se tienen en cuenta varios aspectos:

- **Radio del robot**: Para comprobar si se puede mover el robot a una casilla adyacente hay que tener en cuenta que las marcas de los robots ocupan más de una casilla, con lo que se deben de mover varias casillas, no es una relación de uno a uno. Para eso por cada tipo de movimiento hay que mirar que se puedan desplazar todas las casillas que componen el robot en esa dirección.
- **Casillas vacías**: Es necesario tener presente que en el entorno puede haber varios elementos, con lo que a la hora de calcular el camino hay que mirar el mapa para evitar localizar un robot en una posición que no está vacía, puesto que esto implicaría una acción de riesgo de colisión.
- **Casilla del objeto/posición de fin**: La situación de meta del problema será la que permita dejar al robot (teniendo en cuenta el radio) en una casilla adyacente al objetivo. Si el objetivo es una posición normal, esta será la casilla de meta. Si por el contrario es un objeto, la posición de meta será cualquiera en la que se encuentre dicho objeto.

En la figura 5.13 se puede apreciar la planificación hecha mediante el algoritmo A*. En este caso, el algotimo premia el menor número de casillas visitadas, esto es, el menor número de movimientos. Por ello, primero sigue unas casillas en horizontal para acabar la última fase del camino en diagonal, que es la forma más rápida de llegar hasta el objetivo. La lista de movimientos que genera el algoritmo es : derecha, derecha, derecha, derecha, diagonal, diagonal, diagonal, diagonal, diagonal.

Se puede apreciar cómo el robot se mantendría a una distancia del objeto, pero orientado hacia él para mantenerlo vigilado. Aunque esta es la primera planificación, debido a las posibles salidas de trayectoria y visitas de casillas no marcadas en el camino como por ejemplo por problemas físicos (una rueda influía más fuerza que la otra), se pueden realizar reposicionamientos, quedando el robot en una posición final distinta a la marcada en un primer momento.

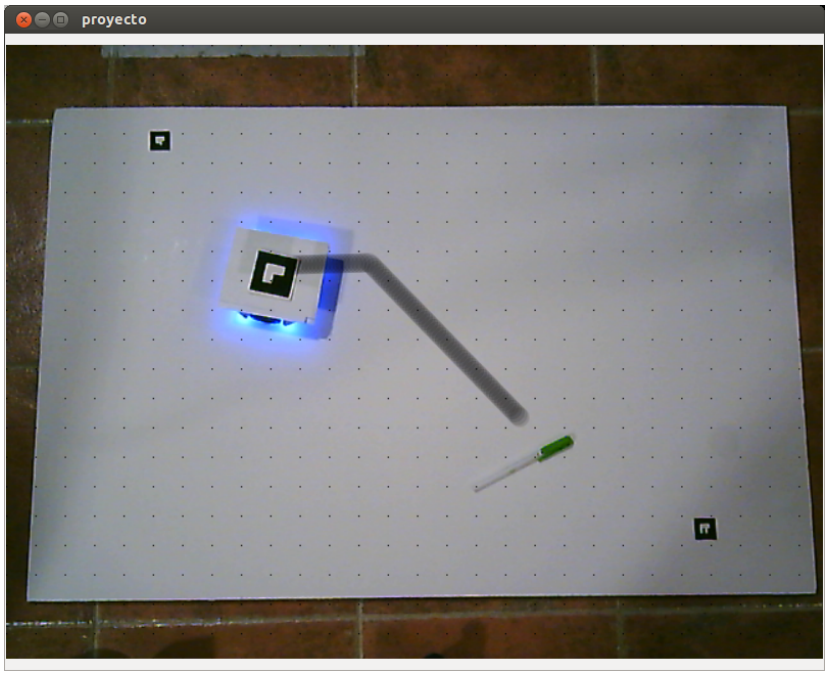

Figura 5.13: Visualización del posicionamiento a través del grid mediante el algoritmo A*.

La arquitectura seguida para modelar este submódulo es la que se puede ver en la figura 5.14. En ella podemos observar cómo a través de la clase ARobot, la cuál es un agente de robot, se realiza el posicionamiento.

Esta clase es la encargada de crear una instancia de la clase búsqueda, la cual será la encargada de realizar el pathfinding. Una vez creada la instancia se lanzará un hilo de la clase AI, la cual hereda de los hilos de ICE [Hen04]. Dicho hilo será el encargado de llamar a la funcionalidad de la instancia de búsqueda que implementa la búsqueda de caminos.

Las clases Object y Mark se utilizan para obtener la localización de los objetos y de la marca del robot respectivamente. Si el robot se encuentra vigilando un objeto, a través de la instancia que almacena el agente de dicho objeto, se obtiene la posición en la que se encuentra, que será la enviada a la instancia de búsqueda. Si no tiene ningún objeto a vigilar, las posiciones de fin del robot serán las esquinas del entorno.

A continuación se especifican en detalle las clases involucradas:

- Clase **Mark**: Ver apartado 5.4.1.
- Clase **Object**: Ver apartado 5.4.1.
- Clase **ARobot**: Esta clase define la lógica de los agentes de los robots. Tienen la lógica de actuación de cada robot. Almacena en todo momento el estado del robot(si está vigilando un objeto o está de guardia) y la posición del robot. Tiene un valor booleano que indica si tiene una trayectoria calculada o no, y en el caso de que tenga un camino calculado también almacena una lista con movimientos y la posición final para el movimiento actual y el ángulo necesario para llegar a dicha posición. En el caso de que el

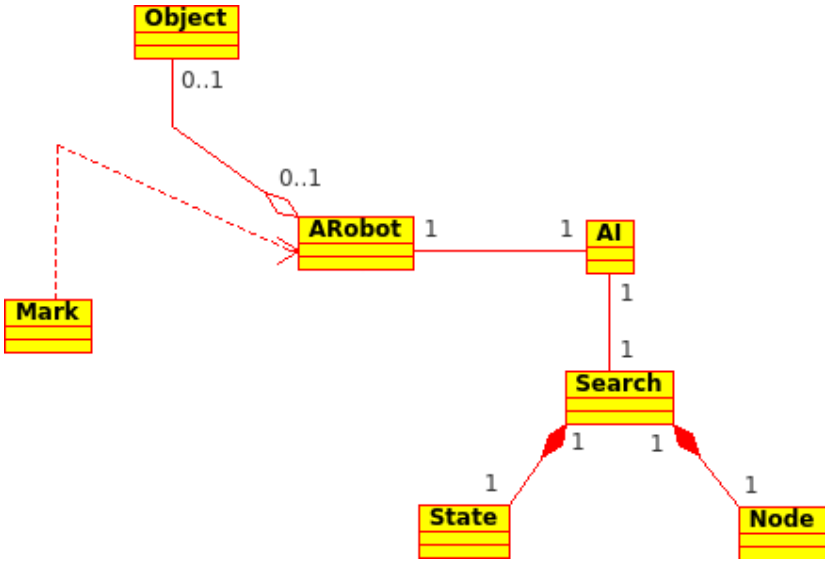

Figura 5.14: Diagrama de clases del submódulo de posicionamiento.

robot esté vigilando un objeto, esta clase mantiene una referencia a dicho objeto, para saber en todo momento qué robot vigila qué objeto. Esta clase también se encarga de gestionar los movimientos del robot, orientándolo y comprobando si se encuentra en una casilla válida dentro del camino calculado.

- Clase **AI**: Esta clase es la encargada de lanzar la búsqueda de caminos. Hereda de los hilos de ICE para ejecutarse en un hilo diferente al principal y no bloquear el sistema mientras se hace la búsqueda. Además mantiene una referencia al agente que lo ha llamado para hacerle una llamada indicando que ya se ha calculado el pathfinding y que puede empezar a manejar a su robot.
- Clase **Search**: Esta clase es la encargada de hacer la búsqueda del camino a seguir por los robots para llegar hasta una posición determinada. Antes de empezar el algoritmo de pathfinding (se sigue el algoritmo A*) genera una foto del estado actual, creando un mapa base además de otro con los elementos visitados, para saber en todo momento las posiciones visitadas y las que no se pueden visitar por encontrar obstáculos.
- Clase **State**: Esta clase almacena la posición en x e y actual, además del movimiento seguido para llegar a esa posición. Es utilizada durante el proceso de búsqueda de caminos.
- Clase **Node**: Esta clase define las propiedades de los elementos necesarios que se utilizarán en el pathfinding. Almacena el estado actual, el coste y una referencia al elemento padre, necesaria para construir la solución final al problema del cálculo de trayectoria.

5.5 Módulo de comunicación

El módulo de comunicación es el encargado de definir la funcionalidad necesaria para la comunicación entre los robots y el sistema.

Este módulo tiene las funciones necesarias para que desde otros módulos se pueda dar ordenes a los robots, independientemente del hardware que sea y de las llamadas que se tengan que hacer. Para el caso de estudio, las llamadas entre los robots y el sistema se hacen a través del paso de cadenas de caracteres. Este módulo implementa la funcionalidad de enviar un mensaje al robot o de recibir un mensaje desde él (véase el listado 5.7). La funcionalidad de recibir desde arduino se queda a la espera de nuevos caracteres hasta que el robot envía la señal de fin de mensaje.

```
1  void Robot::send_arduino(int port)
2  {
3    write(port, code_to_arduino, strlen(code_to_arduino));
4  }
5  char* Robot::recieve_arduino(int port)
6  {
7    int n;
8    n = 0;
9    strcpy(code_from_arduino, "");
10   for(;;){
11     strcpy(data, "");
12     read(port,data,1);
13     if(data[0] ==  10){
14       code_from_arduino[n] = '\0';
15       break;
16     }
17     else if(data[0] == 13);
18     else if(data[0] > 0 && n < 98)
19       code_from_arduino[n++] = data[0];
20   }
21   return code_from_arduino;
22 }
```

Listado 5.7: «Comunicación Robot-Sistema»

A la hora de enviar un mensaje hacia un robot, lo que se hace es escribir dicho mensaje directamente en el puerto de conexión entre el robot y el sistema. Esta conexión se trata como una conexión por puerto serie, aún tratándose de una conexión inalámbrica.

La recepción de los mensajes desde los elementos móviles se realiza a la inversa. En vez de escribir en el puerto, se queda a la escucha de nuevos datos en el puerto de conexión. Como se explicó anteriormente, la función se queda a la escucha de datos hasta que aparezca la señal de fin mensaje.

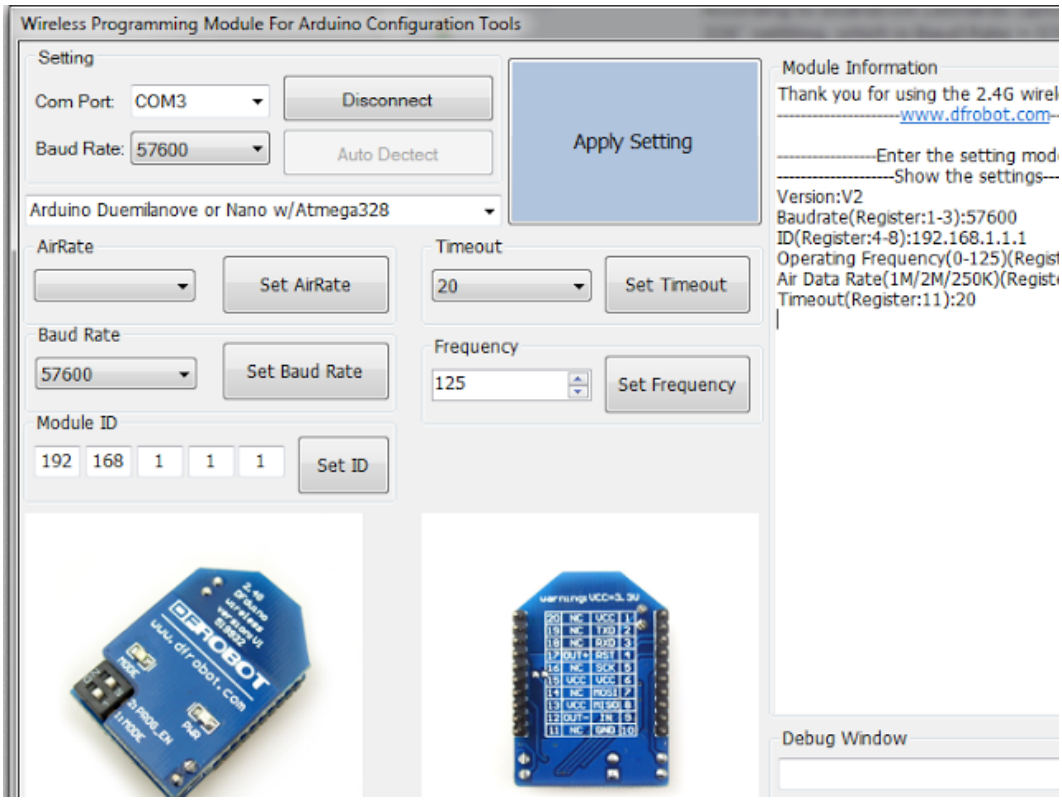

Figura 5.15: Interfaz de configuración de los módulos WPM.

Cada robot está programado para ser capaz de interpretar las órdenes enviadas desde el sistema en forma de caracteres. Dependiendo del carácter enviado, el robot imprime una fuerza a cada motor (cada rueda es impulsada por un motor) de manera independiente.

La comunicación se realiza mediante los sistemas WPM, que envían las señales mediante Wi-Fi. Para ello, el sistema necesita tener un módulo WPM por cada robot, configurados de tal manera que se establezca la conexión uno a uno, emparejándolos por un identificador y una frecuencia. Para ello tenemos que utilizar la misma configuración (como por ejemplo la de la figura 5.15) tanto en el módulo emisor (el del sistema) como en el módulo receptor (el del robot) en parejas. Además cada robot tendrá su propio puerto de conexión, con lo que será más fácil la intercomunicación entre ellos sin que aparezca 'ruido' procedente de los demás robots.

Toda la funcionalidad del módulo de comunicación se hace a través de la clase Robot. Esta clase hace de interfaz con el resto del sistema, para abstraerlo de la manera de comunicarse con los robots y presentar una manera sencilla que, independientemente del hardware utilizado, mantenga siempre la misma estructura. De este modo, si en algún momento se cambia el hardware de los sistemas móviles solo habría que modificar el contenido, la manera de establecer las conexiones, de las funciones en esta clase.

La especificación de la clase Robot, que define la funcionalidad del módulo de comunicación, es la siguiente:

- **Robot**: Esta clase hace de interfaz entre los elementos físicos y el sistema. Es la encargada de abrir el puerto correspondiente para conectarse al robot y de convertir la información recibida del agente en información útil para el robot, mediante chars. También es capaz de recibir información del robot, como los mensajes de confirmación de recepción de maniobra.

5.6 Módulo de coordinación y gestión del conocimiento

Gracias al módulo de coordinación y gestión del conocimiento se dota al sistema de un nivel básico de inteligencia artificial, capaz de adaptarse a diferentes situaciones de manera automática y sin la intervención de ningún usuario.

Cuando el módulo de análisis de vídeo y obtención de la información (apartado 5.3) detecta un cambio en la escena se encarga de notificarle a este módulo que ha habido un cambio, y qué cambio ha sido. A partir de ese momento, el módulo de coordinación y gestión del conocimiento se encarga de de implementar toda la funcionalidad necesaria para convertir el sistema en un sistema inteligente, organizando a los robots para una buena vigilancia del entorno.

Éste módulo lo primero que hace es interpretar el evento ocurrido, y notificado por el anterior módulo. Dependiendo de que evento se trate, el módulo se puede encontrar frente a 6 situaciones diferentes. Estas son:

- **Nuevo robot**: Cuando aparece un nuevo robot en la escena y el sistema lo ha localizado, entonces el módulo de coordinación y gestión del conocimiento obtiene todos los objetos de la escena y busca el objeto mas prioritario. Para ello, no solo busca el objeto con mayor prioridad, sino que además ese objeto no puede estar ya vigilado por otro robot, lo que llevaría a una situación incorrecta. Una vez ha hecho la búsqueda del objeto más prioritario pueden ocurrir dos cosas:
  - **Existe** un objeto disponible. Si se ha encontrado un objeto que no está siendo vigilado por ningún robot, entonces se asigna al nuevo robot la referencia de objeto que será el objetivo de vigilancia. Una vez se tiene asignado el objetivo, se mandará al robot que inicie la rutina de vigilancia de objetivo.
  - **No existe** un objeto disponible. Cuando no existe un objetivo se mandará al robot que inicie la rutina de guardia.

  En el listado 5.8 se puede observar el esquema básico de actuación del coordinador cuando recibe un evento de nuevo robot, tal y como se ha comentado antes.

- **Nuevo objeto**: Si el sistema detecta un nuevo objeto, que no estaba detectado antes entonces, después de que se haya localizado el objeto entero, se hacen varias comprobaciones. La primera es si existe algún robot ocioso, esto es, sin ningún objetivo, sin objeto que vigilar. Si ese es el caso, entonces se le asignará el nuevo objeto a dicho

```
if (event == NR){
  id = id - 2;
  int ob = -1;
  if ((ob = getPriority()) != -1){
    getARobot(id)->setObj(_scene->getObjects()[ob]);
    getARobot(id)->plan(_scene,1);       // Vigilar objeto
  }else{
    getARobot(id)->plan(_scene,3);       // Guardia
  }
}
```

Listado 5.8: «Evento de nuevo robot»

robot y se le mandará que inicie la rutina de vigilancia de objetivo. Sin embargo, si este no es el caso y no hay ningún robot que esté de guardia se comprobará la prioridad del objetivo de todos los robots visibles, y si alguno tiene una prioridad menor que la del nuevo objeto entonces, cogiendo el objeto de prioridad más baja, se cambiará de objetivo al robot en cuestión y empezará la rutina de vigilancia de objetivo (ver listado 5.9).

```
if(event == NO){
  int rob = -1;
  if ((rob = anyIdle()) != -1){ // Si hay algún robot ocioso
    getARobot(rob)->setObj(_scene->getObjects().back());
    getARobot(rob)->plan(_scene,1);      // Vigilar objeto
  }else{
    if ((rob = havePriority(_scene->getObjects().back())) !=
       -1){        // Si hay un robot vigilando un objeto menos
        prioritario
      getARobot(rob)->setObj(_scene->getObjects().back());
      getARobot(rob)->plan(_scene,1);    // Vigilar objeto
    }
  }
}
```

Listado 5.9: «Evento de nuevo objeto»

- **Movimiento de robot**: Otro de los posibles eventos que se detectan es el de movimiento de un robot. Cuando este es el caso, entonces lo primero que se hace es comprobar si existe una situación de alerta. Una situación de alerta equivale a una situación en la que existen dos robots muy próximos entre sí, lo que puede derivar en una futura colisión. Cuando se evalúa la existencia de alerta podemos encontrarnos con tres situaciones:
  - No se ha producido una situación de riesgo relativa al robot, son lo que puede seguir su actuación de manera normal.

- Si hay una situación de riesgo, pero la prioridad del objetivo del robot en cuestión es la más elevada de entre los robots que entran en conflicto. En este caso, al ser el que tiene más prioridad seguirá con su ejecución normal, pero primero se hará un reposicionamiento, se calculará de nuevo el camino a seguir, para adaptarse a la nueva situación y evitar la colisión con los robots circundantes.
- Si hay una situación de riesgo y además el robot no tiene la prioridad más alta. Al no ser el objeto con mayor prioridad, con relación a los objetos que vigilan los robots involucrados en la situación de riesgo, cobra menos importancia el tiempo gastado en llegar hasta él. Por este motivo la solución seguida ha sido la de mandar los robot que se paren y se mantengan a la espera hasta que la situación de riesgo se solucione.

En el listado 5.10 se puede observar el esquema básico de actuación del coordinador cuando recibe un evento de movimiento de un robot definido anteriormente.

```
1  if (event == MR){
2    id = id - 2;
3    int gc = getColision(id);
4    if (gc == 0){ // No existe riesgo colisión
5      getARobot(id)->plan(_scene,2);       // Movimiento normal
6    }
7    else if (gc == -1){    // Hay riesgo de colisión pero tengo la
        prioridad
8          if (getARobot(id)->getGuard()==-1){      // Recalcula la
              ruta según estado
9            getARobot(id)->plan(_scene,1);         // Vigila objeto
10         }else{
11       getARobot(id)->plan(_scene,3);     //Guardia
12     }
13   }else{         // Hay riesgo y además no tengo la prioridad
14     getARobot(id)->stop();       // Me detengo hasta nueva orden
15   }
16 }
```

Listado 5.10: «Evento de movimiento de un robot»

- **Movimiento de objeto**: Si el evento detectado ha sido el de un movimiento de un objeto lo primero que se hace es actualizar su posición y localización en el escenario. Si el objeto no está vigilado no pasa nada, puesto que no afecta al sistema, pero si está vigilado hay que tener en cuenta ese movimiento. Al tener cada robot la referencia de su objetivo, al cambiar la localización del objeto esa referencia interna al robot también se actualiza. Después lo que se hace es replanificar al robot, esto es, se vuelve a calcular la trayectoria a seguir por el robot (ver listado 5.11).

```
if (event == MO){
  for (vector<ARobot*>::iterator it = _arobots.begin(); it !=
     _arobots.end(); ++it){     // Se pregunta a todos los robots por
     si alguno vigila el objeto
    if (_scene->getMark((*it)->getId()+2)->getVisibility()){
      if ((*it)->getObj()->getId() == id){ // Si algún robot
         vigilaba el objeto se replanifica
        (*it)->plan(_scene,1);
      }
    }
  }
}
```

Listado 5.11: «Evento de movimiento de un objeto»

- **Eliminación de robot**: Si el evento detectado es el de la eliminación de un robot se comprueba si ese robot estaba vigilando algún objeto. En el caso en que no se encuentre vigilando ningún objeto el sistema seguirá igual, no habrá ningún impacto en la ejecución. Sin embargo si el robot se encontraba vigilando un objeto se procede a encontrar un nuevo vigilante para ese objeto. Para ello se busca un robot que esté en estado de guardia y si se encuentra alguno, se le asigna el nuevo objetivo. Si no hay ninguno pasará lo mismo que cuando entra un nuevo objeto a la escena, se le pregunta a todos los robots visibles si se encuentran vigilando un objeto y si se da el caso se les pregunta por la prioridad de dicho objeto. Si algún robot se encuentra vigilando un objeto con menor prioridad que el objeto que se acaba de quedar sin vigilar se le cambiará de objetivo y si no el objeto se quedará finalmente sin vigilar.

  En el listado 5.12 se puede observar el esquema básico de actuación del coordinador cuando recibe un evento de eliminación de un robot. Se puede observar cómo después comprobar si estaba vigilando un objeto, independientemente de si lo estaba o no, se reinicia la información del robot, pues la instancia de dicho robot sigue creada (con la asociación marca-robot).

- **Eliminación de objeto**: Cuando desaparece un objeto de la escena se comprueba si ese objeto estaba siendo vigilado o no. Si no estaba siendo vigilado, no pasará nada, el objeto desaparece y ya está, pero si estaba siendo vigilado, la cosa cambia. Lo primero que hay que hacer es buscar el robot que lo vigilaba. Una vez se tiene el robot en cuestión, se elimina la referencia que tiene al objeto que ha desaparecido y se procede como si fuera un nuevo robot. Se comprueba si hay algún objeto sin vigilar y en caso positivo se le asigna el nuevo objetivo al robot para que lo vigile a partir de ese momento. Sino existe ningún objeto sin vigilar se pone al robot en estado de guardia (ver listado 5.13).

```
if (event == BR){
  id = id - 2;
  if(getARobot(id)->hasObj()){  // Si vigilaba un objeto, se busca nuevo vigilante
    int rob = -1;
    if ((rob = anyIdle()) != -1){        // Si hay algún robot ocioso
      getARobot(rob)->setObj(_scene->getObjects().back());
      getARobot(rob)->plan(_scene,1);    // Vigilar objeto
    }else{
      if ((rob = havePriority(_scene->getObjects().back())) != -1){      // Si hay un robot vigilando un objeto menos prioritario
        getARobot(rob)->setObj(_scene->getObjects().back());
        getARobot(rob)->plan(_scene,1); // Vigilar objeto
      }
    }
  }
  getARobot(id)->reset();
}
```

Listado 5.12: «Evento de eliminación de robot»

```
if (event == BO){
  for (vector<ARobot*>::iterator it = _arobots.begin(); it != _arobots.end(); ++it){     // Se pregunta a todos los robots por si alguno vigila el objeto
    if (_scene->getMark((*it)->getId()+2)->getVisibility()){
      if ((*it)->getObj()->getId() == id){       // Si algún robot vigilaba el objeto se replanifica
        (*it)->deleteObj();
        int ob = getPriority();
        if (ob != -1){
          (*it)->setObj(_scene->getObjects()[ob]);
          (*it)->plan(_scene,1);         // Vigilar objeto
        }else{
          (*it)->plan(_scene,3);         // Guardia
        }
      }
    }
  }
}
```

Listado 5.13: «Evento de eliminación de objeto»

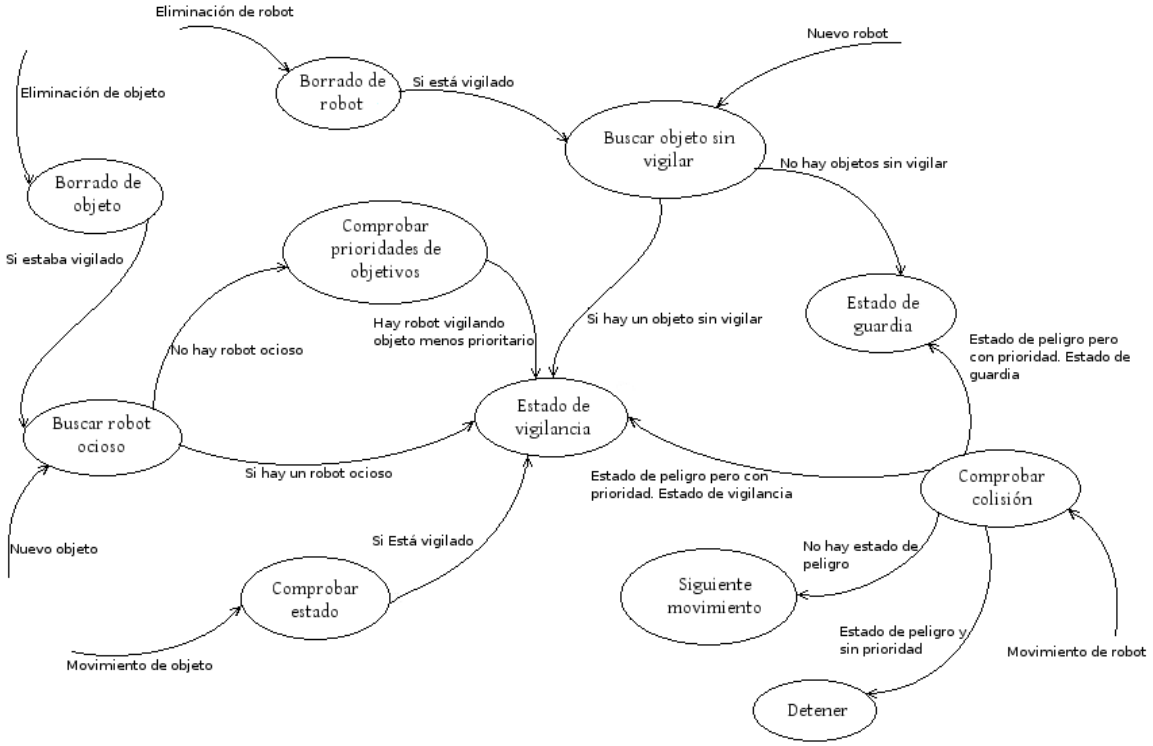

Figura 5.16: Resumen del flujo de coordinación general.

En la figura 5.16 podemos ver un esquema donde se resume la funcionalidad anteriormente descrita. Este es el flujo principal común al sistema, en el que se detectan los nuevos eventos y, a partir de ellos, se disparan una serie de acciones que desembocan en tres acciones principales, acción de **guardia** por el perímetro del entorno, acción de **vigilancia** de objetivo y **detención**.

Una vez que se ha detectado el evento y procesado la información necesaria se activa el agente con dicha información y éste a su vez se comunicará con el submódulo de posicionamiento (apartado 5.4.2). Pero antes de posicionar a los robots, tenemos que decidir hacia dónde tiene que ir el robot. Lo único que sabe el agente es que se ha producido un evento, del tipo que sea, y que tiene que decidir hacia dónde manda a su robot.

Para esto se definen tres casuísticas:

- Si la situación procesada informa al sistema que el robot asignado a un agente en particular debe de vigilar un nuevo objeto, por el motivo que sea, entonces se llamará al submódulo de posicionamiento pasándole la posición de dicho objetivo, para que este submódulo indique la lista de movimientos a seguir y sea capaz de posicionar al robot.

- Si la situación procesada informa al sistema que el robot asignado tiene que ponerse de guardia, entonces el sistema, en base a la posición actual del robot, se comunicará con el submódulo de posicionamiento con la posición una de las cuatro esquinas del

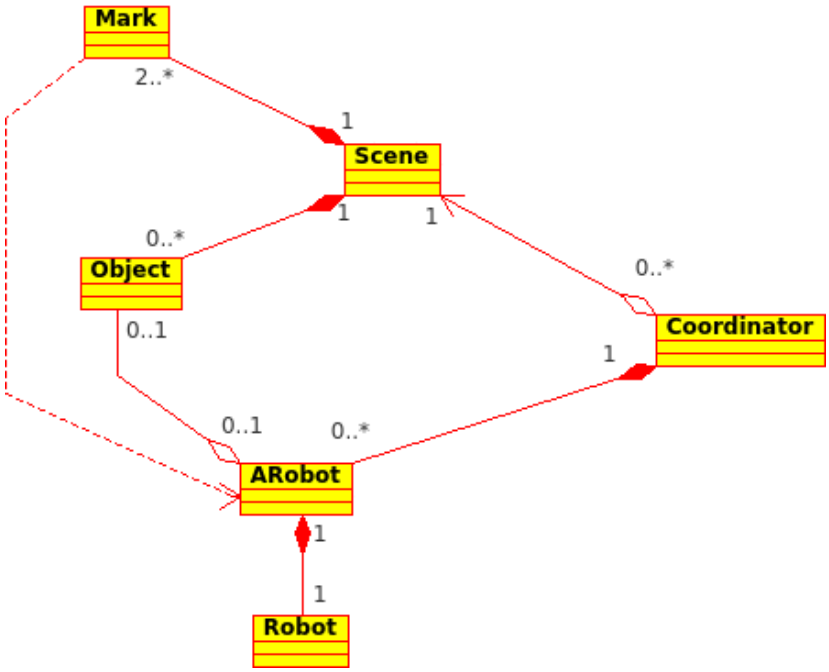

Figura 5.17: Diagrama de clases del módulo de coordinación.

perímetro del entorno. En ese momento el submódulo de posicionamiento se encargará de mantener al robot en constante movimiento, llevándolo de esquina a esquina para realizar la maniobra de guardia.

- Si la situación procesada informa al sistema que el robot asignado se ha movido en el entorno (se distinguen los movimientos de desplazamiento y de orientación), entonces se informará al submódulo de posicionamiento de la nueva situación para que compruebe si se encuentra en un estado esperado, si sigue la trayectoria indicada. Si no sigue es el caso se tiene que replanificar y recalcular el pathfinding para poder volver a orientar al robot y que siga el sistema en estado correcto.

El submódulo de posicionamiento va informando de los pasos seguidos, de los movimientos que hay que seguir, y este módulo es el encargado de comunicarse con el módulo de comunicación para comunicarle al robot los movimientos a seguir, identificados por el submódulo de posicionamiento.

Las clases que componen la funcionalidad de este módulo son las de Coordinator y ARobot, junto con las de Scene, Mark y Object (ver figura 5.17).

La clase Coordinator es la encargada de recibir los eventos, procedentes del submódulo de análisis de vídeo y obtención de la información, y de decidir que robot será el encargado de realizar la acción necesaria. Para ello se comunica con la clase Scene, la cual tiene los vectores de Marks y de Objects de la escena, para tener localizados (no espacialmente sino estructuralmente) tantos los robots como los objetos para poder realizar una buena decisión.

Cabe destacar que se ha establecido una relación de observador entre las clases Scene y Coordinator. En este caso es la clase Coordinator la que se subscribe a la escena para que esta sea la encargada de notificar al coordinador el momento en el que ha ocurrido algún cambio en la escena. Así conseguimos que el coordinador no esté en funcionamiento constantemente, sino solo cuando sea necesario, y aumentar la eficiencia del sistema.

Una vez que el coordinador ha elegido que robot va a realizar la acción se comunica con su agente (la clase ARobot) y le notifica el nuevo evento, la nueva situación de la escena, para que maneje al robot de la manera adecuada.

Por cada nuevo movimiento recibido la clase ARobot se comunicará con la clase fachada Robot, para que el robot atienda a los movimientos.

A continuación se especifica en detalle cada una de las clases involucradas en el módulo de coordinación y gestión del conocimiento:

- Clase **Scene**: Ver apartado 5.2.1.
- Clase **Mark**: Ver apartado 5.4.1.
- Clase **Object**: Ver apartado 5.4.1.
- Clase **Coordinator**: Esta clase es la encargada de hacer el emparejamiento entre las marcas y los agentes. Asigna cada Mark con un ARobot de manera única, asignando siempre el mismo agente con la misma marca. Esta clase maneja la información del entorno a nivel de un mapa minimizado a nivel del grid definido en la escena. tiene un vector de agentes a los cuales llama cada vez que ocurre un evento en la escena relativo a su robots específico (el del agente). Además es la encargada de manejar la lógica de coordinación básica, puesto que es la encargada de asignar los objetos a cada robots, dependiendo de prioridades y de robots ociosos, además de controlar y evitar fallos en el sistema como son las colisiones y los objetivos sin vigilar con robots ociosos, o situaciones en las que varios robots vigilen el mismo objetivo.
- Clase **ARobot**: Ver apartado 5.4.2.
- Clase **Robot**: Ver apartado 5.5

5.7 Patrones de diseño

Los **patrones de diseño** son una solución simple y elegante a problemas comunes del diseño orientado a objetos. Así pues, no es necesario volver a determinar una solución frente a uno de estos problemas. Simplemente se estudia el problema y se adopta la solución para ese problema.

Los requisitos que debe poseer una solución para que pueda ser considerada como un patrón de diseño son dos; **efectividad** y **reusabilidad**. Esto quiere decir que una solución debe

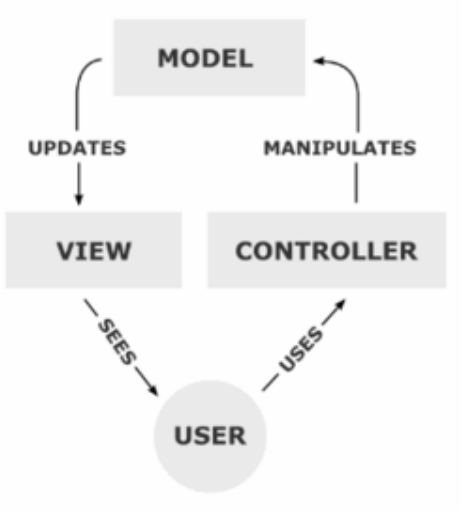

Figura 5.18: Esquema del patrón MVC[1].

haber resuelto problemas con éxito y, además, dicha solución se puede aplicar a problemas con semejantes restricciones.

Los patrones se dividen en tres tipos [GHJV96]:

- De **creación**: Este tipo de patrones propone soluciones para los problemas que supone la creación de objetos.
- De **estructura**: Los patrones de estructura se utilizan para una correcta composición de las clases.
- De **comportamiento**: Estos patrones caracterizan las formas en las que interactúan y reparten responsabilidades las distintas clases u objetos.

5.7.1 Patrones utilizados

En la arquitectura del presente proyecto se han utilizado diversos patrones de diseño, los cuales se enumeran y describen a continuación:

Modelo Vista Controlador (MVC)

Es un patrón de arquitectura de las aplicaciones. Este patrón separa la lógica de negocio de la interfaz de usuario, facilitando la evolución por separado de ambos aspectos, lo que supone un incremento de la reutilización y flexibilidad [Ree03].

El patrón se compone de varios módulos, como son los siguientes (figura 5.18):

- El **Modelo**: El modelo contiene el núcleo funcional de la aplicación. El modelo proporciona funciones para acceder a sus datos y que son utilizadas por la vista para adquirir dichos datos y mostrarlos al usuario.
- La **Vista**: La vista presenta la información al usuario, usualmente mediante la interfaz de usuario, por lo que requiere dicha información del modelo.

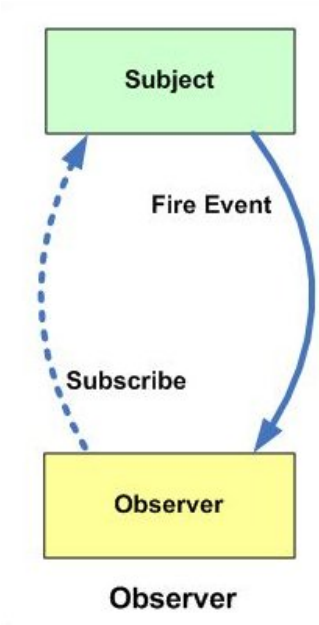

Figura 5.19: Esquema del patrón Observer.

- El **Controlador**: El controlador define las entradas del usuario como eventos. Un controlador implementa procedimientos de atención a eventos, dichos procedimientos serán invocados cuando se produzca un evento, como la interacción con el ratón o el teclado. Los eventos se convierten en peticiones al modelo, que procesará esa petición de una manera u otra.

El uso de este patrón define la arquitectura del sistema, por lo que se puede decir que es el más importante.

Observer

El patrón Observer es un patrón de diseño que define una dependencia del tipo uno-a-muchos entre objetos, de manera que cuando uno de los objetos cambia su estado, notifica este cambio a todos los dependientes [GHJV96]. En la figura 5.19 podemos ver el esquema básico del patrón.

Se trata de desacoplar la clase de los objetos clientes del objeto, aumentando la modularidad del lenguaje, creando las mínimas dependencias y evitando bucles de actualización (espera activa o polling). En definitiva, se usa el patrón Observer para implementar la funcionalidad cuando un elemento 'quiere' estar pendiente de otro, sin tener que estar preguntando de forma permanente si éste ha cambiado o no.

La principal ventaja de este patrón es que todo se logra sin recurrir a un acoplamiento estrecho. El sujeto solo sabe de una lista de observadores y en una sola llamada los notifica. Al sujeto no le interesa los efectos o desenlaces de los observadores, él simplemente emite. El resultado es código flexible y reusable.

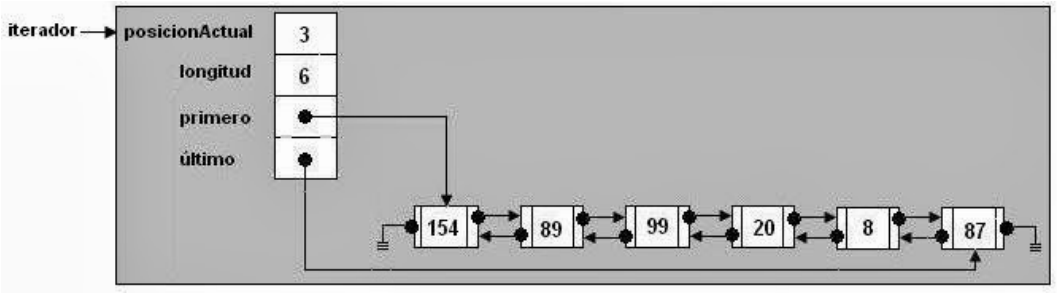

Figura 5.20: Ejemplo de patrón Iterator[2].

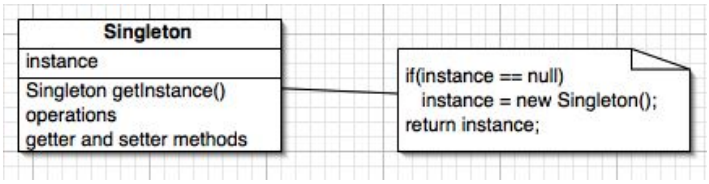

Figura 5.21: Esquema del patrón Singleton[3].

Iterator

Proporciona una manera de acceder a los elementos de un objeto secuencialmente [GHJV96]. En la figura 5.20 podemos ver el esquema básico del patrón.

La idea clave de este patrón es tomar la responsabilidad para el acceso y recorrido del objeto lista y ponerlo en un objeto iterador. La clase de iterador define una interfaz para poder acceder a los elementos de la lista. Un objeto iterador es responsable de hacer el seguimiento del elemento actual; es decir, sabe que elementos se han atravesado ya.

El principal uso de ese patrón se hace integrado con el patrón Observer para la notificación a los distintos subscriptores.

Singleton

La aplicación de este patrón asegura que una clase tenga únicamente una instancia, y proporciona un punto de acceso global a la misma [GHJV96]. En la figura 5.21 podemos ver el esquema básico del patrón.

Las situaciones más habituales de aplicación de este patrón son aquellas en las que dicha clase controla el acceso a un recurso físico único (como puede ser el ratón o un archivo abierto en modo exclusivo) o cuando cierto tipo de datos debe estar disponible para todos los demás objetos de la aplicación. En el caso de Sivi, su uso más importante es el de implementar el filtro, ya que debe ser usado por varias clases.

Facade

El patrón fachada proporciona una interfaz unificada para un conjunto de interfaces de un subsistema [GHJV96]. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar. En la figura 5.22 podemos ver el esquema básico del patrón.

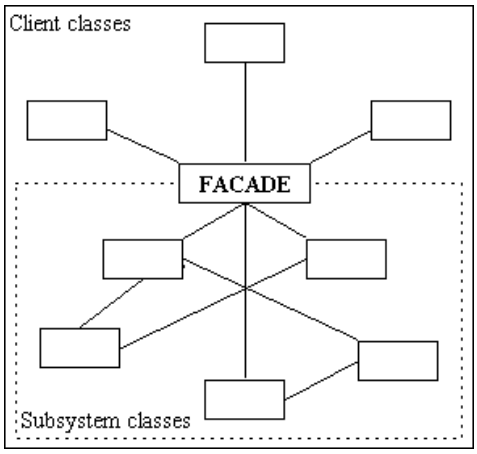

Figura 5.22: Esquema del patrón Facade[4].

El patrón fachada se ha utilizado en la aplicación para establecer una interfaz que haga más fácil el acceso a las llamadas de comunicación con los elementos robóticos.

Aplicar el patrón fachada tiene el principal objetivo de mantener un bajo acoplamiento entre clases y una cohesión alta para cada clase. Para mantener un bajo acoplamiento, es necesario que las relaciones entre clases sean las menores posibles. Una clase con alta cohesión significa que la clase es responsable de proveer funcionalidad, de tal manera, que no existen dos clases con responsabilidades similares o que se complementen.

Capítulo 6

Evolución, resultados y costes

ESTE capítulo evalúa el trabajo llevado a cabo desde que comenzó el proyecto hasta que llegó a la etapa de mantenimiento. Como parte de ello, se estudiará la aplicación de la **metodología** realizada, se analizarán los **resultados** obtenidos a la hora de desplegar el sistema en un caso de estudio concreto y se estimarán los **costes** del desarrollo.

6.1 Evolución

Esta sección pone de manifiesto las distintas iteraciones empleadas desde el inicio del proyecto hasta el final del mismo. Cada una de ellas describe, además de la información relevante, los diferentes hitos alcanzados.

En primer lugar, debido a las múltiples alternativas que se planteaban a la hora de la realización del Proyecto Fin de Carrera, se acordó el dominio del mismo. Tras determinar la temática del proyecto, se determinó que el primer problema a solucionar sería el de qué elementos robóticos se iban a utilizar, ya que siendo la temática del proyecto la de un sistema de vigilancia móvil se hace necesaria una buena elección en el soporte físico. Después de estudiar diferentes alternativas (robots más grandes, más pequeños, con cuatro ruedas, con dos ruedas) se decidió la utilización de los robots MiniQ 2WD Complete Kit (descritos en el apartado 4.2.3) por adaptarse a lo requerido y la buena relación calidad/precio.

Una vez definido el hardware a utilizar se acordaron los objetivos iniciales del proyecto, los cuales se enumeran a continuación y se definen en el apartado 2.2:

- Mejorar la monitorización de entornos.
- Aumentar la coordinación entre elementos robóticos.
- Conseguir un alto grado de modularidad.
- Crear un sistema un escalable.
- Mejorar la visualización de los sistemas de vigilancia convencionales.
- Favorecer la flexibilidad, modularidad, escalabilidad, adaptabilidad y posterior mantenimiento del proyecto mediante el uso de diversos patrones de diseño.

- Incrementar las posibilidades de utilización en diversas plataformas con el uso de estándares y tecnologías multi-plataforma.

Tras la definición de los requisitos iniciales, dio comienzo el desarrollo del Proyecto de Fin de Carrera (Agosto de 2013). El desarrollo de dicho proyecto ha seguido varias iteraciones. A continuación se detallan estas iteraciones.

6.1.1 Iteraciones

Iteración nº1

Una vez definidos los requisitos, y escogido el hardware que se iba a utilizar había que familiarizarse con él puesto que, al ser elementos físicos mecánicos, se mueven mediante impulsos, mediante fuerzas aplicadas a los distintos motores.

La primera iteración consistió en la creación de la base del módulo de comunicación. Se establecieron los valores necesarios para el movimiento correcto de los robots, así como su programación (en arduino).

Se definieron las funciones necesarias para el envío de ordenes a los robots y la recepción de sus respuestas y se estableció el protocolo de actuación de los robots, de manera interna, para cada mensaje recibido.

Seguidamente, se determinó que todo el proyecto debería estar gestionado por un sistema de control de versiones. Se siguieron los pasos necesarios para la implantación de un sistema de este tipo que soportara la estructura del proyecto, así como los cambios que iban sucediendo.

Como resultado de este iteración se creo una aplicación que, a través de la herramienta de arduino, fuera capaz de reconocer las teclas pulsadas y dependiendo de ellas se ejercía una función u otra. A parte de las funcionalidades de avanzar, retroceder, girar y parar la aplicación también diferenciaba entre un robot y otro, de forma que su pudieran mover varios robots de forma simultánea pero independiente.

Así pues el objetivo de esta iteración se resume en la definición e implementación del módulo de comunicación.

Iteración nº2

Cuando la primera iteración estuvo completa se procedió a la implementación del submódulo de representación de la información.

Debido a que este submódulo necesitaba integración de varias herramientas se procedió a implementarlo en una iteración a parte.

Lo primero fue crear una interfaz mediante GTK, incluyendo botones para poder manejar a los robots (siguiendo la aplicación de la iteración anterior). La interfaz era una interfaz simple, con cuatro botones para dirigir al robot.

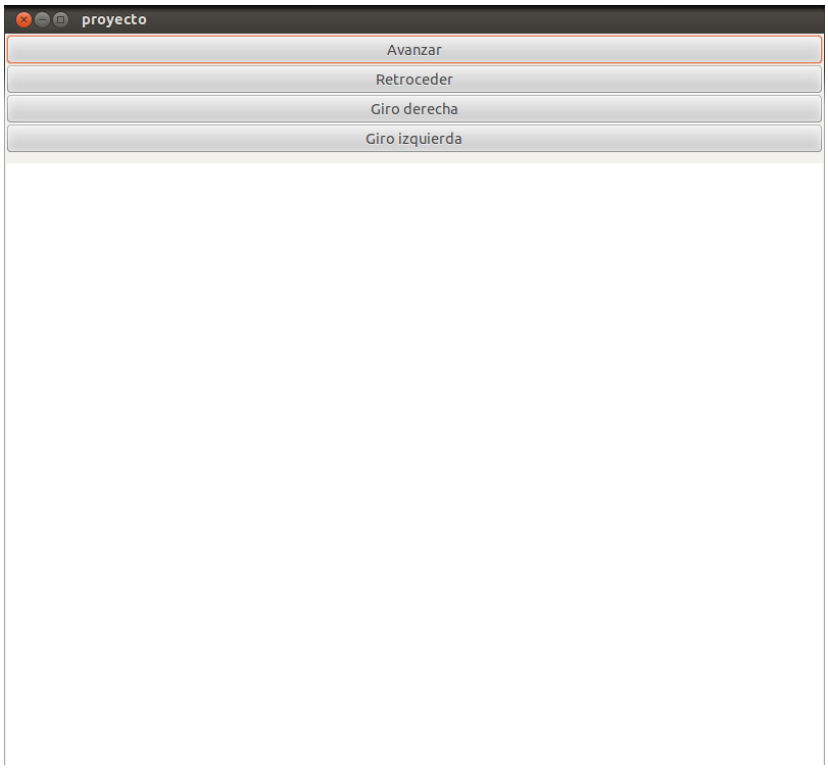

Figura 6.1: Estado de la interfaz en la iteración 2.

Después se procedió a la integración de las bibliotecas de Ogre3D y OpenCV, para la representación de las imágenes de la cámara en una textura en Ogre3D.

Una vez que se integraron estas dos bibliotecas y se veían las imágenes de OpenCV en una interfaz de Ogre3D se procedió a la creación de un widget para GTK en el que se pudiera mostrar la salida de la interfaz de Ogre3D. Para ello se creó la clase OgreWidget, que hereda de Gtk::Widget.

Como resultado de esta iteración se obtuvo una aplicación capaz de mover los robots mediante eventos de botones en una interfaz de Gtk, mientras se monitorizaba la escena a través de dicha interfaz con una ventana de Ogre3D. Podemos ver el estado de la aplicación (la interfaz anteriormente descrita) en la figura 6.1

Iteración nº3

Una vez acaba la segunda iteración, y establecido el primer prototipo de la interfaz, se procedió a la implementación del submódulo de detección de marcas.

Para ello se integraron las bibliotecas de ARToolKit en la aplicación a través de la implementación de la clase ARTKDetector. Esta clase es la encargada de realizar la detección de marcas, así como la funcionalidad de posición y rotación necesaria para representar la información en la interfaz (integrada con ogre a modo de realidad aumentada, información añadida a las imágenes de vídeo).

Además se acabó de implementar el módulo de comunicación de forma definitiva.

El resultado de esta iteración se resume en la creación de una aplicación que, integrando lo conseguido en la iteración anterior, sea capaz de reconocer las marcas de realidad aumentada (submódulo de detección de marcas), así como de mover los robots mediante el uso de botones en la interfaz, a través del módulo de comunicación.

Con esto se obtuvo un primer prototipo de la aplicación, el cual integraba todas las bibliotecas necesarias para un sistema mono-robot.

Iteración nº4

Acabada la iteración nº3, con la detección de las marcas, se procedió a la implementación del otro submódulo, el de detección de objetos.

Puesto que el sistema consiste en la vigilancia de entornos había que crear alguna manera de controlar la delimitación del entorno a monitorizar. Para ello se utilizaron dos marcas, de menor tamaño que las utilizadas para el seguimiento de los robots, situadas en esquinas opuestas, una en la esquina superior derecha y la otra en la esquina inferior derecha.

Una vez que se tenían localizadas dichas marcas, se guardaba en la escena su posición y se definían dichas posiciones como los puntos máximos y mínimos del entorno.

Con estos puntos se estableció un filtro, dedicado a abstraer al módulo de detección de objetos del posible ruido procedente de colores fuera del entorno delimitado. Esto se consigue estableciendo cuatro funciones (una para cada lado) que determinen si un determinado punto pertenece a una recta o bien se sitúa a un lado o al otro de dicha recta.

El siguiente paso fue el de crear el submódulo de detección de objetos. Primero se definió la manera de procesar las imágenes y después se decidió la opción de aprovechar que, para la representación de dichas imágenes, había que procesar pixel a pixel (para crear la textura) la imagen para procesar el color de cada pixel a la vez, en la misma pasada.

Utilizando el filtro se comprobaba si el pixel se encontraba dentro del entorno, en cuyo caso se comprobaba el color. Si se encontraban pixels de los colores definidos se iban guardando una serie de valores y contadores necesarios para, posteriormente, calcular el centro del objeto detectado, esto es su posición en el espacio.

Al final de esta iteración se obtuvo un prototipo en el cual se añadía a la funcionalidad del prototipo anterior la de la detección de los objetos. En este momento se quedaba implementado el módulo de Análisis de vídeo y obtención de la información, con los dos submódulos en una versión estable. Además se dio el primer paso para la creación del submódulo de localización (dentro del módulo de localización y posicionamiento)

Iteración nº5

Durante la siguiente iteración se procedió a definir e implementar el submódulo de localización de manera definitiva.

Lo primero que se tuvo en cuenta fue la capacidad que debía tener el sistema para soportar la inclusión en el escenario de multitud de robots y de objetos. Por ello se implementaron los primeros prototipos de estructuras de datos para almacenar dichos objetos. Durante el procesamiento de cada imagen se buscan todos los colores y marcas definidas, independientemente del número de estos elementos que aparezcan en la escena, y se almacenan dentro de la escena en las estructuras definidas, en este caso en vectores.

Una vez se tenían las estructuras necesarias, se procedió a la implementación del submódulo de localización. Este se encargaba de almacenar en los objetos de los elementos pertenecientes a la escena tanto su posición como sus puntos límites (máximo y mínimo) como la orientación.

Con esto se tenían todos los elementos clave de la escena completamente localizados, puesto que ya teníamos de la iteración anterior la delimitación del espacio a través del as marcas de delimitación.

Al final de la iteración nº5 se obtuvo un prototipo con la funcionalidad del prototipo anterior más el soporte para varios robots y varios objetos y la implementación del submódulo de localización completa.

Iteración nº6

En la iteración nº6 se procedió a la creación de los agentes y una primera versión de la inteligencia artificial básica.

Durante esta iteración se creó la clase agente que es la encargada de comunicarse con el módulo de comunicación (es la que decide los movimientos del robot). Además se generan instancias una por cada robot definido en el sistema. Con esto lo que se logra es la creación de una agente independiente para cada robot. Estos agentes se encargaban de decidir los movimientos de los robots, de comprobar su orientación y redirigirlos para seguir una dirección concreta.

Después de la creación de los agentes se integró en el sistema una primera versión de la inteligencia artificial. Ya no era el Agente el que decidía los movimientos, sino que se creó una clase capaz de calcular una trayectoria óptima desde un punto hasta otro. Para ayudar en el cálculo de trayectorias y optimizar este proceso se creó un grid, una rejilla, para minimizar el espacio de búsqueda.

Este grid se calcula a través de las marcas de delimitación. Puesto que las marcas de delimitación son fijas en el sistema, su tamaño no cambia, así que es una buena referencia para la creación de la rejilla. El tamaño de las casillas de la rejilla es el de la diagonal de la

marca de delimitación más grande.

Una vez se tenía definido el grid se creaba un mapa interno en la clase scene en el que se almacenaba la información de la escena. Si alguno de los pixels que caían dentro de cada casilla se encontraba un objeto, esa casilla pasaba a tener el valor del identificador del objeto. En el caso de las marcas de los robots se procedía de la misma forma. Una vez definido el mapa, éste se mandaba a la clase encargada de calcular el camino a seguir.

Si se encontraba un camino, este se devolvía en forma de lista de movimientos (los movimientos vienen definidos como caracteres, cada carácter corresponde a un movimiento). Con esta información el agente ya era capaz de posicionar los robots en las posiciones especificadas.

Cabe mencionar que en esta iteración la clase encargada del pathfinding (cálculo de caminos) se implementó como un scrpt en python.

Se puede decir que con la finalización de esta iteración se creó la primera versión del submódulo de posicionamiento.

Iteración nº7

A lo largo de la séptima iteración se procedió al desarrollo de la clase coordinator, además de la utilización de varios patrones observer.

Lo primero fue crear una clase que se encargara de la coordinación entre los agentes. Se ha estableció un sistema central encargado de esta coordinador como primer caso de estudio en el que la coordinación se tratara desde un sistema central.

Este es el sistema que se encargaba de obtener la información de la escena para manejar a los robots según fuese el caso.

A su vez se modificaron los agentes y la escena, para que a partir de ese momento la toma de decisiones fuera a través del coordinador. Aunque los agentes no dejaron de tomar decisiones, puesto que a ellos solo les llegaban los eventos de vigilancia de un objetivo o guardia, pero era el propio agente el que se encargaba de la toma de decisiones de acción tenía que realizar el robot.

Después, para aumentar la eficiencia, se decidió implementar dos patrones observador en el sistema. El primero de ellos era el que se encargaba de notificarle a la escena la aparición de nuevos objetos y nuevos robots, para que la escena no estuviera preguntado frame tras frame. De esta manera solo se activaba la escena cuando se detectaba un cambio en uno de los dos submódulos de detección.

Una vez que se notificaba a la escena un determinado evento, este se encargaba de notificar al coordinador, por la misma razón de eficiencia.

Iteración nº8

En la siguiente iteración se establecieron unas pequeñas modificaciones en el código con la intención de mejorar la calidad de dicho código y, además, de arreglar pequeños defectos.

Uno de esos cambios fue la introducción de la referencia del objeto objetivo dentro de los agentes.

También se precedió a cambiar todo el sistema de pathfinding. En vez de hacer el cálculo de las trayectorias a través de un script en python se procedió a la realización de dichos cálculos en el mismo lenguaje en que se encuentra todo el sistema (C++). De esta manera se consigue una mayor homogeneidad en el presente proyecto.

Se decidió hacer todos los cálculos de búsqueda de caminos en hilos de ejecución al margen del hilo principal. Esto es porque aumentar la simultaneidad de los cálculos, en el momento en el que se encuentran varios robots, y además se evita la latencia en el procesamiento de las imágenes.

El hilo principal seguirá su propia ejecución al margen de que se estén calculando las trayectorias, y en el momento en el que se haya calculado el camino, se activará el agente de nuevo para seguir con su procesamiento normal. Si se encuentra haciendo un calculo de trayectoria no debe de estar activo el agente, puesto que no puede realizar ninguna acción, no sabe cual debe de ser su próximo movimiento.

Iteración nº9

En la iteración nº9 se definieron los remates finales del submódulo de representación de la información y se creó el submódulo de gestión de eventos de usuario.

Lo primero que se definieron fueron los posibles eventos que se iban a controlar en el sistema. Como solución se propuso que los eventos detectados fueran solo los eventos de ratón. Con ellos el usuario podría pedir información al sistema relativa a los objetivos y a los robots de vigilancia.

Los eventos se separaron en dos tipos de eventos, los producidos con el botón derecho y los producidos por el botón izquierdo.

- Botón **derecho**: En el caso de obtener un evento de botón derecho se decidió que la aplicación debía de hacer desaparecer el cuadro de texto en cual se muestra la información, ya sea de un objeto o de un robot.
- Botón **izquierdo**: En el caso de obtener un evento de botón izquierdo se decidió que la aplicación debía mostrar la información relativa al elemento sobre el que se produzca el click del ratón. Para ello se comprobaba el valor que tenía el mapa de la escena en la posición x e y (normalizada al tamaño del grid). Dependiendo de si el elemento era un objeto o un robot, el cuadro de texto mostraba una información distinta.

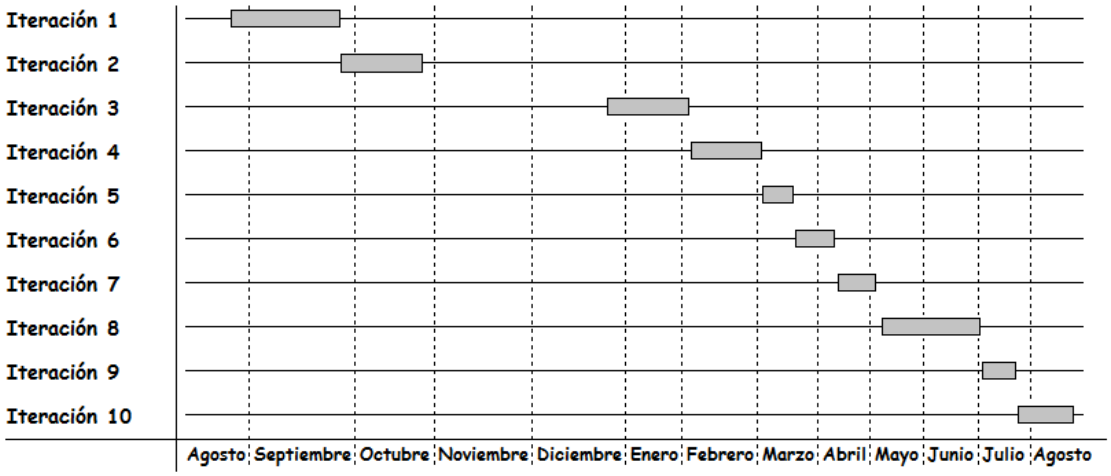

Figura 6.2: Diagrama de fechas para cada iteración.

Además, durante la realización de esta iteración se crearon los elementos de realidad aumentada utilizados en el sistema. No solo se crearon los cuadros de texto a modo de layout para mostrar la información básica, sino que también se crearon los indicadores de posición de los robots y los del ángulo de visión. Estos últimos elementos se realizaron con la herramienta Blender, para generar los objetos en 3D.

Con la realización de esta iteración se dio por cerrada la implementación del sistema, pues todos los módulos se encontraban ya implementados, y el sistema se encontraba en completo funcionamiento.

Iteración nº10

En esta iteración se genera la documentación correspondiente al presente Proyecto Fin de Carrera. Para la generación de la documentación se crean las imágenes que componen la documentación. Para la maquetación de la documentación se utiliza el paquete arco-pfc.

En esta iteración se construyen los vídeos de demostración de la herramienta, además de la presentación de la misma. Esta resulta la última iteración del presente Proyecto Fin de Carrera.

6.1.2 Fechas

El período de realización de cada una de las diferentes iteraciones puede observarse en la figura 6.2.

6.2 Resultados

Los resultados acerca del proyecto tratan de ayudar a evaluar en qué medida se ha logrado cumplir con los objetivos y requisitos impuestos. Debido a que el trabajo realizado consiste

en un Sistema Inteligente de multi-robot para la vigilancia Móvil de Entornos, ha sido necesario construir un caso de estudio del sistema con un modelo de coordinación básico basado en un sistema coordinador central.

En las siguientes secciones se detalla el caso de estudio realizado, analizando los resultados obtenidos. Además, se estudian diferentes estadísticas del proyecto, tales como el número de líneas o el uso del repositorio.

6.2.1 Caso de estudio

Introducción

Como caso de estudio se ha optado por un escenario en cual se emplean hasta tres robots, cada uno identificado con una marca propia, y tres objetivos diferentes, cada uno de un color diferente (verde, rojo y marrón).

Los robots son de un tamaño reducido, con una fuente de alimentación mediante pilas y unos chips de comunicación emparejados dos a dos. Esto supone un problema por la dificultad de mantener la potencia de las pilas y mantener estable la comunicación entre los chips de comunicación. A pesar de estar emparejados dos a dos, siempre que el sistema se apaga y los robots también, se desacoplan y necesitan de una comunicación previa al lanzamiento del sistema para su correcto funcionamiento.

Si el caso de estudio se quisiera escalar a un sistema de mayor envergadura con robot de mayor tamaño y un entorno de vigilancia mayor, no habría problema, los robots se podrían reemplazar otros robots de mayor tamaño, como pueden ser pequeños drones, mientras que en el entorno solo habría que hacer las marcas de mayor tamaño (el suficiente para que la cámara cenital las detectara). Además el escenario estará delimitado por dos marcas de delimitación situadas en diagonal (esquina superior izquierda y esquina inferior derecha).

Todo el entorno estará monitorizado a través de una cámara cenital, a través de la cual el sistema obtendrá la información del escenario. Esta cámara cenital en el caso de estudio ha sido una webcam normal y corriente, pero se podría sustituir por una cámara de CCTV perfectamente y el sistema seguiría en correcto funcionamiento.

SIVI trabaja con las herramientas de OpenCV y Ogre3D en colaboración para obtener una interfaz de usuario en la que se pueda ver tanto las imágenes de la cámara cenital como la información adicional. En este caso de estudio se ha implementado una serie de elementos de notificación, que ayuden al usuario en la tarea de vigilancia del entorno. Estos elementos de notificación son una serie de pantallas situadas en la parte inferior izquierda en la que se muestre la información solicitada por el usuario.

Para ello el usuario podrá hacer uso del ratón para, pinchando en un robot o en un objetivo, obtener la información importante como son el estado, el identificador y el objetivo/robot que lo vigila. Además en caso de que el elemento seleccionado sea un robot y este este de

guardia se mostrará la dirección que sigue dicho robot. Cuando el elemento seleccionado ha sido un objeto se mostrará en pantalla la prioridad de dicho objetivo junto con el resto de información anteriormente descrita. Además, en todo momento se podrá visualizar por la pantalla la orientación de los robots mediante un elemento 3D indicando el ángulo de visión de dichos robots. Este marcador se moverá a la vez que el robot para mantener una correcta visualización de los robots.

En este caso de estudio se ha realizado la coordinación entre los robots (indispensable en un sistema multi-robot) mediante un sistema central encargado de coordinar los robots. Sin embargo, aunque exista un sistema central coordinador, cada robot dispone de un agente capaz de tomar sus propias decisiones, dentro de las ordenes del coordinador. Esto es, el coordinador es el encargado de decidir que robot vigila que objeto, pero son los agentes los encargados de calcular el posicionamiento de los robots y seguir su movimiento para redirigirlos y reorientarlos. Además en el caso de que los robots estén de guardia son los propios agentes los encargados de decidir hacia dónde debe ir su robot para mantener la guardia a través del perímetro.

Todas las herramientas y bibliotecas utilizadas en el caso de estudio han sido de software libre, lo que ha añadido una cierta complejidad en algunos casos por su falta de documentación, o una maña API. Sin embargo esto hace que se pueda utilizar en cualquier entorno, independientemente del equipo.

Las marcas utilizadas para la detección de los robots y las marcas de delimitación son marcas de detección comunes, que aplicando pequeños cambios se pueden sustituir por marcas más sencillas de camuflar en un sistema real, para mejorar su visualización.

El simple hecho de mezclar tantas bibliotecas y tantas herramientas en un solo sistema hace aumentar la complejidad de este caso de estudio. Sin embargo esto ofrece la ventaja de poder crear un sistema más eficaz gracias a que se pueden utilizar multitud de funcionalidades, cada una orientada a un proceso concreto.

Cuando aparece la idea de diseñar un sistema de vigilancia móvil el principal problema a paliar es el de hacer un buen diseño de la inteligencia artificial que tendrá dicho sistema, sobre todo la parte de que afecta a los sistemas móviles (robots). En este aspecto hay que tener muy presente los aspectos más importantes como son los cálculos de trayectorias, el seguimiento y corrección de trayectorias de los robots, y en el caso de sistemas multi-robots se hace indispensable una buena coordinación entre ellos, esto es, la creación de los mecanismos necesarios para que los robots se comuniquen y se coordinen entre ellos.

Aunque todo esto puede disparar la complejidad del sistema, ofrece una experiencia de usuario más cercana a la que se quiere conseguir, que es la implementación de un sistema de vigilancia inteligente y automático, en el que el usuario (el vigilante) tenga que intervenir lo menos posible).

Objetivos de monitorización

En el caso de estudio propuesto existen dos tipos de objetivos de monitorización, los cuales se describen a continuación:

- **Robots**
- **Objetos de colores**

Aunque los robots son objeto de monitorización por parte del sistema, puesto que es necesario que el sistema tenga totalmente localizados los robots para su correcta coordinación y posicionamiento, los objetivos reales son los objetos de colores. Para el caso de estudio solo se ha estudiado la inclusión de tres tipos de objetivos, uno rojo, uno verde y uno marrón.

Cada objeto tiene una prioridad, dependiendo de su color, asociada de la siguiente forma:

- **Verde**: Un objeto de color verde significa que, aunque es necesaria su vigilancia, si en el entorno se encuentra cualquier otro objeto, éste (el verde) pasará a ser la última opción de guardia. or lo tanto podemos asegurar que los objetos de color verde son los menos prioritarios del sistema.
- **Marrón**: Un objeto de color marrón representa un posible objetivo, siempre y cuando no existe un objetivo muy prioritario. Los objetos de color marrón tienen prioridad media.
- **Rojo**: Un objeto de color rojo representa un objetivo muy importante, y por tanto, siempre que exista un robot en la escena deberá vigilarlo. Son los objetos de mayor prioridad del caso de estudio.

En el caso de estudio puede haber hasta tres robots, cada uno con su marca identificativa. Si en el sistema se encuentra algún robot y no existen objetivos de monitorización, los robots que se encuentren en la escena se encontrarán de guardia, esto es, dando vueltas al perímetro delimitado por las marcas de delimitación.

Si existen objetivos y robots, estos se encargarán de ir escogiendo objetivos (por prioridad de objetivo) hasta que no haya más robots (en cuyo caso quedarán los objetivos menos prioritarios sin vigilar) o no haya más objetivos que vigilar (en cuyo caso los robots restantes se dispondrás a hacer la ronda de guardia).

Flujo de trabajo

Una vez que arranca el sistema, lo primero que ocurre es que busca las marcas de delimitación del entorno. Una vez que el sistema ha reconocido las marcas de delimitación, éstas ya no se procesarán más. A partir de ese momento el sistema es completamente automático. Toda la escena recogida por la cámara cenital se muestra por la interfaz de usuario. En cualquier momento se puede introducir une elemento en la escena y el sistema lo detectará y se

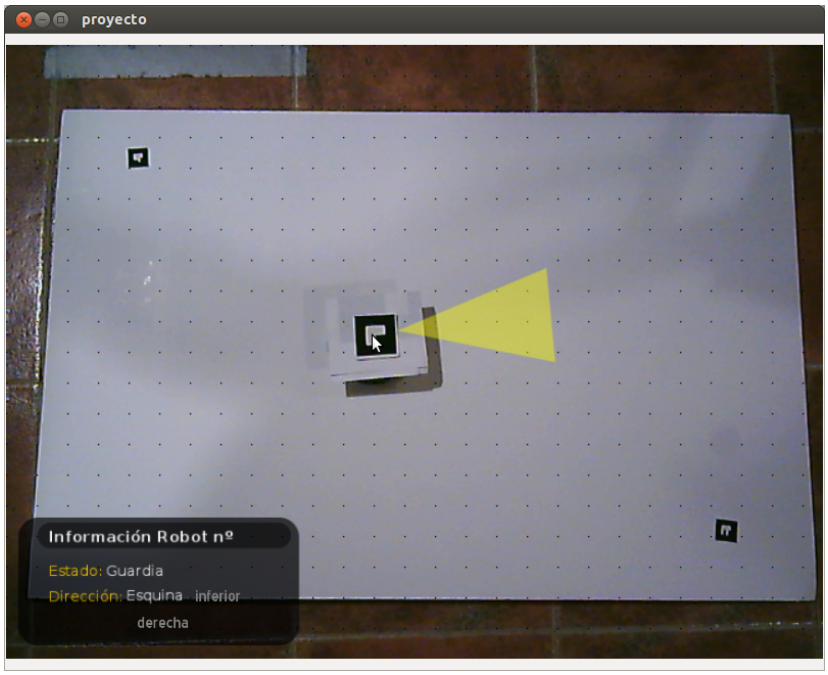

Figura 6.3: Robot en estado de guardia

pondrá en marcha.

Si se introduce un objeto, el sistema lo detecta y se queda ahí, puesto que al no haber robots disponibles nadie puede ir a vigilarlo. Cuando se introduce un robot, este busca objetivos a vigilar, y si no encuentra ninguno, este se dispone a seguir el patrón de guardia, empezando por la esquina inferior derecha. Después irá de esquina en esquina, siguiendo el patrón:

- Esquina inferior derecha.
- Esquina superior derecha.
- Esquina superior izquierda.
- Esquina inferior izquierda.

En la figura 6.3 se puede observar como el robot se dirige hacia la esquina inferior derecha al estar en estado de guardia. Es el primer estado de este robot en estado de guardia y una vez que llegue a una posición próxima cambiará de objetivo a la esquina superior derecha.

Sin embargo si ya se encuentra un objeto, el robot pasaría a ir hacia él, quedándose a una distancia prudencial de él. En ese momento se mantendrá en esa posición mientras siga existiendo el objeto. La única manera de que salga de ese estado de 'standby', a parte de la desaparición del objetivo, es que entre en el escenario otro objeto con más prioridad, en cuyo caso pasará a ir a vigilar a ese nuevo objeto. De manera independiente, aunque el robot se encuentre en estado de guardia, si se introduce un objetivo en el entorno, por ejemplo un objeto verde, él robot dejaría de estar en guardia y pasaría a vigilar ese objeto. A su vez, cuando se introduce un objeto en la escena, se busca cualquier robot disponible, en estado

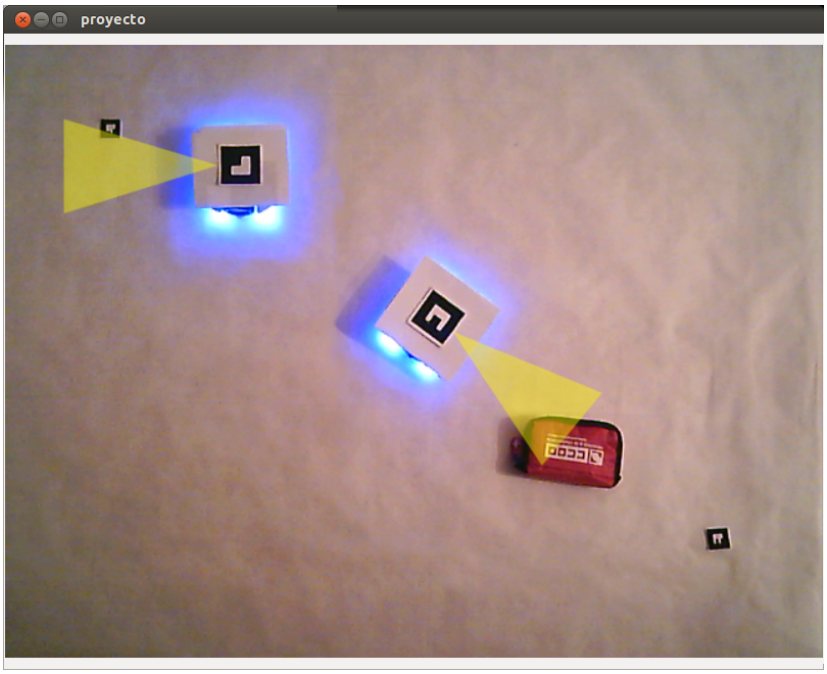

Figura 6.4: Cambio de estado del robot de guardia a vigilando (estado de guardia)

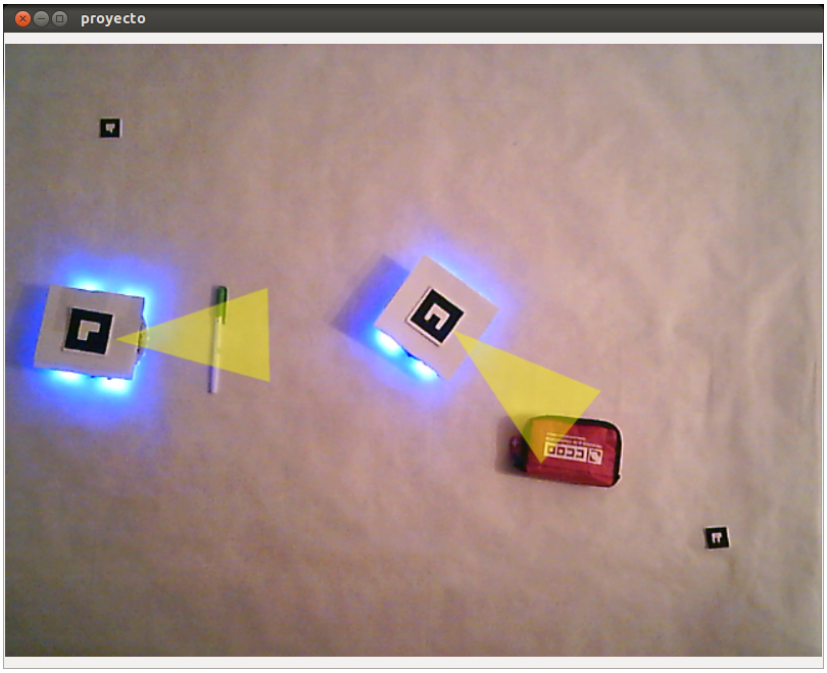

Figura 6.5: Cambio de estado del robot de guardia a vigilando (estado vigilando)

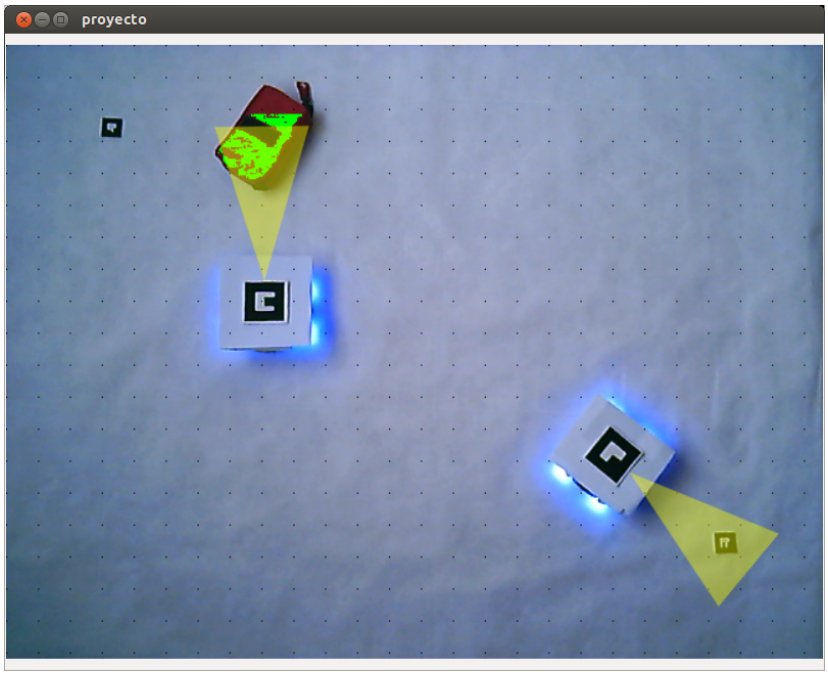

Figura 6.6: Movimiento de objeto al límite del entorno

de guardia. Si se encuentra alguno ese robot pasará a cambiar de estado y vigilar al nuevo objeto introducido en la escena.

En la figura 6.4 se puede observar cómo se tiene un robot vigilando un objeto (de color rojo, máxima prioridad) mientras otro robot se encuentra de guardia, en este caso su dirección es la esquina superior izquierda. Mientras cambia de dirección (esquina inferior izquierda) se introduce otro objeto de color, es este caso de color verde, que aún siendo el de menor prioridad es más prioritario que el estado de guardia. Por eso, el robot cambia de estado y pasa de estar en guardia a vigilar el nuevo objetivo de color verde, como se puede apreciar en la figura 6.5.

Hasta aquí todo es bastante intuitivo pero, ¿qué pasa si se coge un objeto y se cambia de posición? En ese caso, como el sistema está preparado para ello, detectará una modificación en el objeto y, siempre y cuando estuviera vigilado por un robot, se informará al agente del robot de la nueva posición y se moverá a una nueva posición desde la que poder vigilar el objeto. En la figura 6.6 se puede observar, en modo debug, como después de mover un objeto de la escena, éste se coloca en el límite del entorno de búsqueda, por lo que solo se detecta la parte que se encuentra dentro. Esto hace que el robot que se encuentra vigilando ese objetivo tenga como posición final la parte media del objeto de dentro del entorno.

¿Y si lo que se mueve y se cambia de sitio es un robot? Tampoco pasa nada. Los robot pueden desaparecer de la escena perfectamente, que en el momento en el que se vuelvan a colar en ella, se detectará, y se volverán a poner en funcionamiento, como si fuera un robot nuevo. Si el robot no llega a desaparecer de la escena, se tratará como un movimiento más de robot, se comprobará que no está en la casilla que debería y se recalculará su trayectoria.

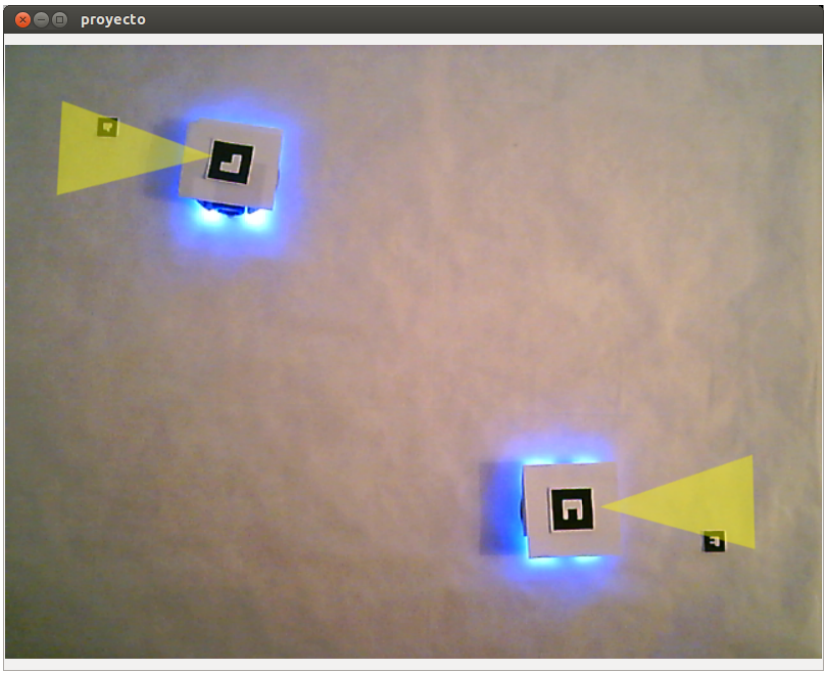

Figura 6.7: Coordinación entre robots con el mismo estado, cada uno manejado por un agente diferente

Aunque desaparecieran los elementos de la escena, en el caso de estudio que está discutiendo, se tienen en cuenta estas situaciones y están definidas como estados controlados.

Si desaparece un objeto se comprueba si está vigilado, en cuyo caso se cambiará de estado al robot correspondiente y se le intentará asignar otro objetivo. Si existiera algún objeto sin vigilar se mandaría a este robot a vigilarlo. Si no existiera ningún objetivo sin vigilar, se cambiaría al estado de guardia y procedería a ejecutar su rutina anteriormente descrita. Por el contrario, si lo que desaparece es un robot la cosa cambia. Aparece en la escena un objeto sin vigilar, y se buscará un robot en estado de guardia. Si no hubiera ninguno disponible se comprobaría la prioridad del objeto 'abandonado' por el robot con el de los objetivos del resto de robots disponibles en la escena y, en caso de que haya un robot vigilando un objeto menos prioritario se cambiará de objetivo a dicho robot (siempre teniendo en cuenta que ese objetivo tiene la prioridad menor de entre todos los disponibles).

Como se puede observar, el sistema es tolerante a cualquier modificación en el entorno. Estas modificaciones están reflejadas en los casos de estudio del capítulo 5.

Como se trata de un sistema multi-agente se hace necesaria una coordinación que responda de manera independiente para cada robot. En la figura 6.7 se puede observar como se manejan de manera independiente dos robots, que aún estando los dos en el mismo estado de guardia, cada uno sigue una dirección diferente, con lo que se consigue que cada robot sea orientado de manera independiente gracias a los diferentes agentes asignados a cada robot.

Al tratarse de un sistema de vigilancia, se propuso el uso de una interfaz en la que se muestre la imagen captada por la cámara. Pero no solo eso, sino que para completar el sistema en este caso de estudio se ha añadido la funcionalidad de ampliación de la información en pantalla a través de elementos de realidad aumentada y para ello el usuario puede en cualquier momento hacer una petición por pantalla pinchando sobre un elemento del sistema.

Se tratan como elementos físicos del sistema tanto los objetos de colores como los robots, dejando de lado las marcas de delimitación. Si el usuario pulsa sobre un objeto, aparecerá en pantalla la información relativa a ese objeto en ese momento, pero el sistema sigue funcionando. Si la pulsación se realiza sobre un robot se le mostrará la información en ese momento de ese robot, pero al estar en funcionamiento el sistema, el robot puede cambiar de estado, con lo que el usuario tendría que pedir de nuevo la obtención de información de ese robot.

También puede decidir eliminar el cuadro de texto con la información ampliada, por ejemplo si le molesta par ala visualización de un determinado robot (aunque estos elementos de realidad aumentada son traslúcidos, esto es, permiten ver a través de ellos). Para ello solo tendrá que hacer click con el botón derecho del ratón, esta vez no es necesario pinchar en el elemento específico.

Conclusiones

Como ya vimos antes, los sistemas de vigilancia móvil multi-robot son muy complejos. A la complejidad que conlleva la comunicación individual con un componente físico, que se mueve mediante fuerzas en los motores, hay que sumarle la complejidad de tener que comunicarte con varios robots a la vez y tener que coordinarlos. Y aunque se haga difícil la coordinación entre ellos, al final gracias a ella se consigue un escenario con un alto grado de independencia, puesto el sistema es capaz de coordinarse solo y evitar colisiones, algo indispensable en los sistemas de vigilancia modernos.

En este caso de estudio, las unidades móviles utilizadas añaden la complejidad de que su fuente de alimentación es mediante pilas, lo que conlleva posibles fallos en los movimientos por perdida de potencia. Además los módulos utilizados solo se pueden comunicar uno a uno, con lo que hay que tener un transmisor y un receptor para cada robot.

Pero esto a su vez es beneficioso en el sentido en el que al ser transmisores separados, es más fácil la comunicación, puesto que cada robot tendrá su puerto de enlace con el sistema, de forma única e inequivoca. Esto evita posibles fallos de comunicación, como el mandar una determinada orden a un robot erróneo.

Otro punto a tener en cuenta es que se han utilizado robot pequeños, lo que facilita el transporte y más importante, hace más dinámico el entorno. Gracias a su pequeño tamaño se consigue aumentar el espacio de movimiento en entornos pequeños (como en el caso de

Language	files	blank	comment	code
C++	16	272	277	2409
C/C++ Header	15	88	28	583
Python	1	21	54	257
SUM:	32	381	359	3249

Tabla 6.1: Relación de las líneas de código de SIVI

estudio) pero sin embargo ofrece la ventaja de poderse adaptar a entornos más grandes sin necesidad de cambios, lo que lo hace altamente escalable.

Uno de los puntos más difíciles es el del procesamiento de la imagen para la detección de los objetivos. Finalmente se optó por el uso del espacio de colors RGB. Aunque se demostró que era más sencilla la detección con el uso del espacio HSV, gracias a una buena calibración de los parámetros se consigue un resultado igual de bueno, con la ventaja de ser más rápido en proceso, puesto que no tiene que procesar la imagen dos veces.

Como conclusión final se obtiene que con este caso de estudio de un sistema multi-robot, con un sistema coordinador, se ha conseguido un sistema robusto capaz de tolerar las modificaciones en el entorno,y aunque sea un caso de estudio de dimensiones reducidas se han conseguidos unos resultados tan buenos que, gracias a su modularidad,se aplicar en entornos mucho mayores sin cambios en el sistema, puesto que con el cambio de parámetros sencillo como el tamaño de las marcas y los umbrales de los colores se puede escalar a sistemas de gran escala.

6.2.2 Estadísticas del proyecto

Líneas de código

A modo informativo, la tabla 6.1 determina la relación entre los ficheros y las líneas escritas empleadas para el desarrollo de SIVI. Para obtener estos datos se ha empleado la herramienta libre cloc[1], que contabiliza el número de líneas escritas de un proyecto en cada lenguaje.

Uso del repositorio

Este apartado permite determinar la actividad del desarrollador de la aplicación en el sistema de control de versiones empleado. La figura 6.8 indica, por un lado y de color verde, el número de líneas añadidas por semana y, en color rojo, el número de líneas eliminadas por semana. Estas estadísticas son generadas por semanas, lo que permite realizar un estudio de ello y observar como a medida que se avanza en el caso de estudio los cambios son cada vez menores, puesto que se va asentando la base del código y simplemente se van añadiendo pequeños cambios. También cabe destacar la falta de actividad existente entre la

[1] http://sourceforge.net/projects/cloc/

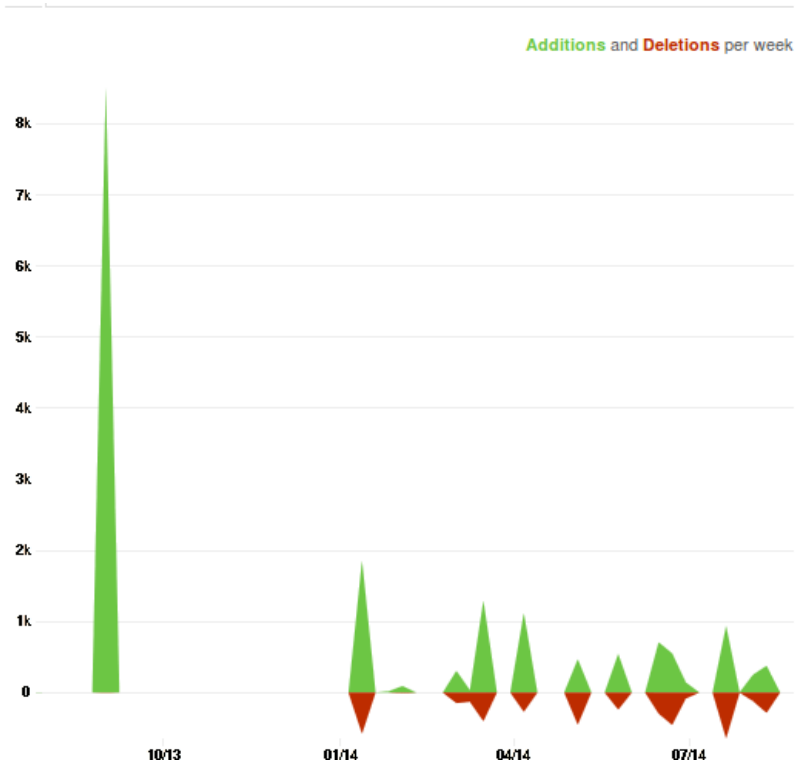

Figura 6.8: Cambios en el repositorio del proyecto.

Elemento	Precio	Unidades	Precio total
Desarrollador	25€/h	700	17500€
Computador portátil	636€/u	1	636€
Cámara Trust Ceptor HD	21,80€/u	1	21,80€
Robot miniq 2WD	58,96€/u	3	176,88€
Wireless Programming Module	18,58€/u	6	111,48€
Total			18445€

Tabla 6.2: Precio total del proyecto.

primera semana del desarrollo y la segunda semana de febrero. Esta falta de actividad solo fue representativa en el repositorio, puesto que aunque seguía el desarrollo, éste se redujo drásticamente por problemas personales.

Sin embargo, podemos observar como la primera semana de desarrollo fue la que más cambios introdujo. Esto se debe a que es la etapa del proyecto donde se implementaron las primeras aproximaciones de la mayoría de las clases que componen el proyecto.

6.3 Costes

El desarrollo del proyecto ha tenido una duración de 10 meses. Si se asume que son 3,5 horas / día la media de horas empleadas y que el sueldo de una persona con conocimientos de programador y de gestión de conocimientos es 25€/hora. El coste del desarrollo del proyecto es 17500€. A ese total hay que añadir el precio del computador utilizado, es decir, 636€, y el

de la cámara que es de 21,80€. También hay que añadir el precio de las unidades robóticas y los chips de programación y comunicación inalámbrica, esto es 58,96€cada robot más 18,58€por cada módulo, lo que hace un montante de 288,36€. El precio total del proyecto es 18445€(véase la tabla 6.2).

Capítulo 7

Conclusiones y propuestas

En este capítulo se expone una valoración acerca de los objetivos alcanzados, teniendo en cuenta la construcción de la arquitectura del sistema y los resultados obtenidos tras la aplicación del mismo. Además, se realizan una serie de propuestas futuras para ampliar y mejorar las funcionalidades de SIVI. Para concluir, se realiza una valoración personal acerca de lo que ha supuesto la realización del presente Proyecto Fin de Carrera.

7.1 Objetivos alcanzados

La principal motivación de este proyecto es la de ayudar en tareas tan importantes hoy en día como son las de vigilancia. Como se ha visto en capítulo 1, cada vez son más importantes los sistemas de vigilancia ya que ayudan a mantener la seguridad ciudadana y una buena calidad de vida.

A pesar de que los sistemas de vigilancia se encuentran en constante crecimiento (cada vez más empresas se especializan en este ámbito) y del hecho de que cada vez se estén centrando más en análisis de vídeo hace que los sistemas robóticos se queden un poco al margen de esta evolución.

Y aunque la aparición repentina de los drones supone un paso más en la introducción de elementos robóticos móviles en los sistemas de vigilancia, este campo sigue estando en estado experimental.

Esto hace la implementación de un sistema que integre ambas partes más difícil por parte del desarrollador, puesto que existen muy pocas herramientas de este tipo en la actualidad.

Una de las principales ventajas de las que ha gozado el desarrollo del proyecto han sido las continuas reuniones con el director de proyecto. Estas reuniones han sido necesarias tanto para idear soluciones a diferentes problemas como para evaluar la forma de implementación requisitos ya cubiertos. Esto dota al sistema de mayor calidad.

Uno de los requisitos del sistema era el de aumentar la **coordinación** entre elementos robóticos mediante la creación de un sistema central capaz de coordinar los elementos móviles del sistema, los elementos vigilantes, de manera independiente. Este objetivo se ha conse-

guido mediante la creación de dicho sistema central y la implementación de una serie de patrones se conducta.

Otro de los requisitos del sistema era la **modularidad**. Este objetivo se encuentra cubierto, puesto que se han creado diferentes módulos, los cuáles, tienen su propia responsabilidad e interactúan entre sí para la realización de las tareas del sistema. Así, el mantenimiento y la ampliación del sistema no resulta costosa.

También se ha conseguido mejorar la **visualización** de los sistemas de vigilancia convencionales con la creación de un sistema de información visual basado en realidad aumentada, en el que el usuario puede visualizar la información relevante del sistema relativa a los elementos del sistema como son los robots y los objetos de colores.

Otro de los requisitos era la creación de una arquitectura **adaptativa** y **extensible** basada en patrones de diseño. Tal y cómo se puede observar en el capítulo 5, este objetivo se ha cubierto utilizando diferentes patrones de diseño que se aplican en la ingeniería del software.

Se ha conseguido incrementar las posibilidades de utilización en diversas plataformas y se ha creado un sistema totalmente **portable**, puesto que, se han utilizado estándares para su implementación, así como, lenguajes de programación y bibliotecas portables.

Con todo esto podemos decir que se ha cumplido el objetivo de **mejorar la monitorización de entornos** a través de la creación de un sistema inteligente multi-robot, ya que además de los objetivos anteriores se ha conseguido implementar un sistema que detecte varios robots.

En el capítulo 6 se define un caso de estudio basado en un sistema central coordinador, a partir del cual se han obtenido resultados importantes para la evaluación del rendimiento del sistema. Se han introducido hasta tres elementos robóticos y hasta tres objetos de colores diferentes y se ha comprobado cómo el sistema ha sido capaz de adaptarse a las nuevas variables del entorno.

Aunque se ha visto que el comportamiento de los robots a la hora de seguir trayectorias no ha sido correcto del todo, puesto que hay veces que se sale de la trayectoria debido a la fuerza de los motores, se ha podido comprobar como el sistema se adapta a estos cambios y es capaz de reconducir los robots para que vigilen sus objetivos.

Dentro de dicho caso de estudio (con un sistema central) se puede observar cómo el comportamiento a través de dicho sistema coordinador ha sido el que se considera aceptable, puesto que ha evitado las situaciones de peligro como son objetivos prioritarios sin vigilar, varios robots con el mismo objetivo y las colisiones. Con todo ellos se puede decir que se ha conseguido un sistema **estable** y **tolerante** a posibles fallos.

En resumen, tanto el objetivo general, como los objetivos específicos del capítulo 2 han sido cubiertos en su gran mayoría. Esto demuestra el nivel de implicación y compromiso tanto del desarrollador de la herramienta, como del director de proyecto.

7.2 Propuestas y trabajo futuro

Aunque los objetivos propuestos se han cumplido en su mayoría, el proceso de desarrollo del software no es estático sino que evoluciona con el tiempo. Así, se ha identificado unas líneas de trabajo futuro que completen o mejoren las funcionalidades que ofrece SIVI, las cuales se describen a continuación:

- **Implementar nuevos modelos de coordinación**. El sistema actual cuenta con un sistema central que es el encargado de proporcionar la coordinación básica en el sistema. Sin embargo la tendencia actual en la robótica móvil es tener robots completamente autónomos, sin un sistema centralizado.

  Por ello se propone como línea de trabajo futuro la implementación de la misma funcionalidad existe actualmente en SIVI pero sin la aparición del sistema central de coordinación.

- **Hacer más amigable la interfaz**. Actualmente el sistema posee una interfaz sencilla en la que únicamente se muestra la salida de vídeo y, a través de eventos de usuario, información importante del sistema mediante realidad aumentada.

  Como línea de trabajo futuro se podrá abordar la implementación de una interfaz en la que apareciera la información de una manera más 'vistosa' y en la que aparecieran botones que permitieran interactuar con los elementos robóticos (por ejemplo que nos permitiera detener un determinado robot en una posición).

- **Implementar un sistema de log para el almacenamiento de los eventos detectados en la escena**. En el sistema se detectan constantemente eventos como la aparición de robots u objetos, o como el movimiento de los robots en estado de guardia. Sin embargo esa información no se guarda en ningún sitio.

  Se podría construir un sistema que escribiera un log cada vez que se generase un evento para, de esa manera, poder comprobar que detectó el sistema en determinado momento para que su comportamiento fuera el observado. Ayudaría sobretodo a la hora de solucionar problemas como puede ser un fallo en las baterías, en el que aunque la orden mandada sea ir hacia adelante, el robot no se puede comportar de esa manera y hay una rueda que gira y otra que no debido a la falta de potencia.

- **Mejorar la detección de los objetos**. Actualmente el sistema es capaz de detectar los objetos de colores mediante comprobaciones de los valores en el espacio RGB. Sin embargo está demostrado que para la detección de colores en imágenes es más preciso el uso del espacio HSV. La segmentación HSV de los objetos de colores es mucho

más precisa que la realizada para los objetos utilizando la segmentación RGB según se puede apreciar en el artículo 'Detección de objetos por segmentación multinivel combinada de espacios de color' [GTO04].

Como posible línea de trabajo futuro se propone la utilización de este espacio de colores para la detección de los objetivos, así como la localización de varios objetos del mismo color de manera independiente.

- **Adaptar el sistema a las tecnologías Web y móviles**. El sistema sólo se puede ejecutar como aplicación de escritorio, con lo que queda bastante aislada de las tecnologías que están copando el mercado informático en estos momentos, puesto que cada vez es más común el uso de estos dispositivos para la monitorización.

  Es normal que la mayoría de las personas que estuvieran interesadas en el uso de esta aplicación, posean dispositivos móviles o tengan conexión a Internet. Por tanto, sería interesante crear aplicaciones tanto móviles como web, que permitieran la monitorización del entorno con la información a modo de realidad aumentada. De esta manera, la aplicación podría llegar a ser utilizada por mayor número de personas.

- **Utilización de robots móviles más complejos**. Aunque el sistema es totalmente escalable, se propone el uso de robots móviles más complejos que mejores las tareas de vigilancia, añadiendo sensores y cámaras propias. Para ello se hará necesaria la modificación en SIVI tan solo del módulo de comunicación para cambiar los métodos de entrada y de salida de información entre el sistema y os nuevos dispositivos robóticos.

  Este pequeño cambio mejoraría la eficacia del sistema, puesto que se haría más sencilla la interacción con los robots, al poder utilizar robots más sensibles a los movimientos.

7.3 Valoración personal

El desarrollo del presente Proyecto de Fin de Carrera no solo me ha servido para completar mis estudios de ingeniería superior en informática, sino que me ha ayudado a comprender la complejidad que alcanzan los sistemas software de cierta envergadura y la manera de proceder a su construcción desde cero. A pesar de encontrarme ya dentro de un entorno laboral, de los proyectos en los que me he visto involucrado ninguno a llegado a esta envergadura, por lo menos no creando el sistema desde cero, sin ser simples mejoras. Pese a ello me he demostrado que todo esfuerzo vale la pena, y que si luchas por las cosas, por muy lejos que parezca el fin siempre aparece la luz al final del túnel.

En cuanto a la tecnología utilizada he de decir que he aprendido mucho. Desde el principio me puse a trabajar con herramientas que no había utilizado nunca, y con un hardware y un software poco intuitivo (y mal documentado, todo hay que decirlo) que no hacía sino plantear un reto tras otro. Sin embargo gracias a ello me he dado cuenta de realmente así es como se disfruta del trabajo que uno realiza, superando obstáculos.

Sin embargo me hubiera gustado poder dedicarme por completo a la realización del presente proyecto y dedicarle más tiempo. Cuando se tiene que compaginar el trabajo con la realización es todo mucho más complicado, aparece el estrés y el agotamiento mental lo que dificulta la obtención de resultados satisfactorios.

Pero en definitiva creo que el resultado obtenido ha sido satisfactorio, como prueban los resultados obtenidos. Espero que el sistema pueda seguir creciendo y ofreciendo nuevas funcionalidades, a través de las líneas propuestas de trabajo futuro, puesto que realmente creo que puede servir de base para sistemas potentes de vigilancia y seguridad.

ANEXOS

Anexo A

Fotos en color

A continuación se anexan las imágenes del caso de estudio en color para una mejor visión del sistema.

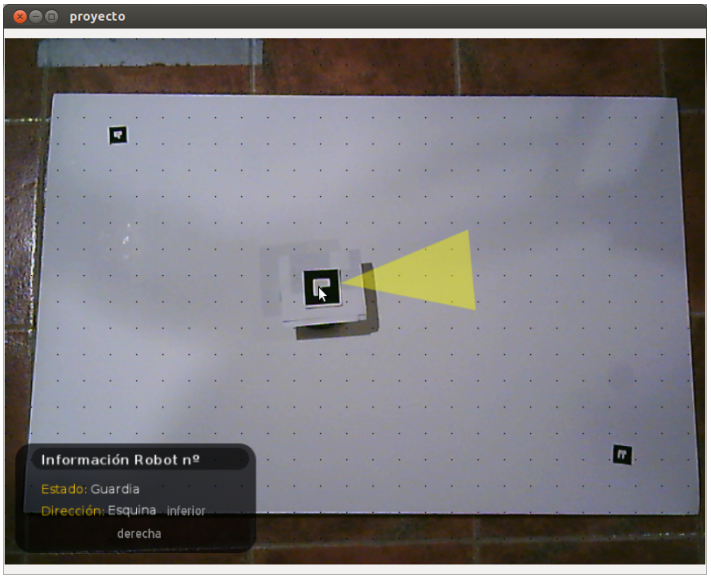

Figura A.1: Con información(evento de pulsación sobre robot)

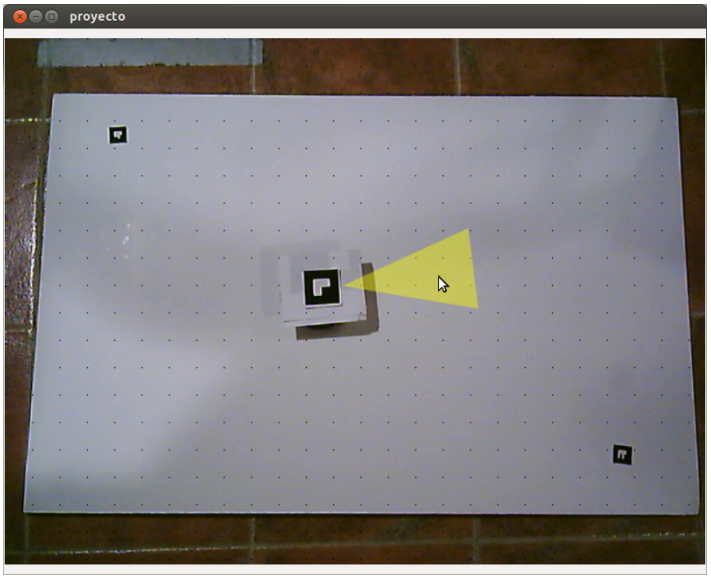

Figura A.2: Sin información (evento de pulsación con botón derecho)

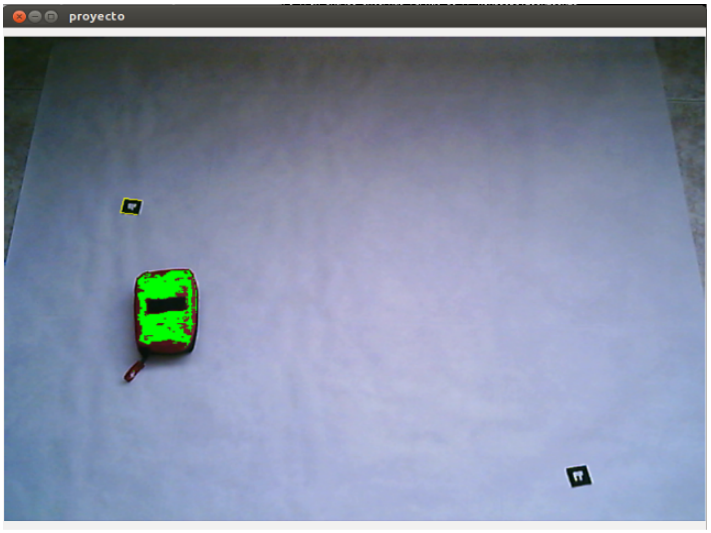

Figura A.3: Imagen en modo debug de la detección de una marca.

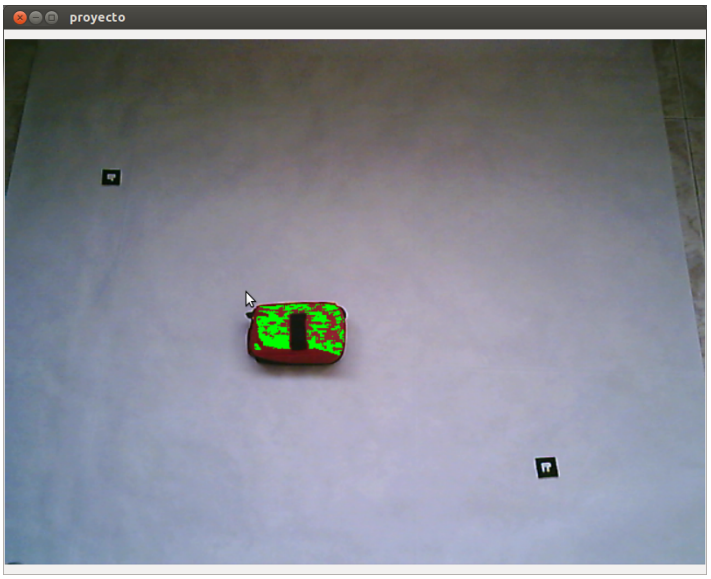

Figura A.4: Imagen en modo debug de la detección de un objeto.

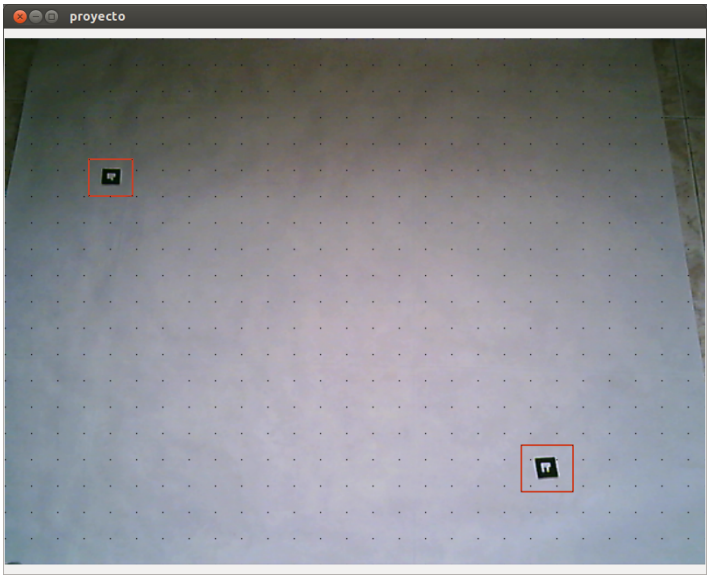

Figura A.5: Localización de las marcas de delimitación del entorno y grid en modo debug.

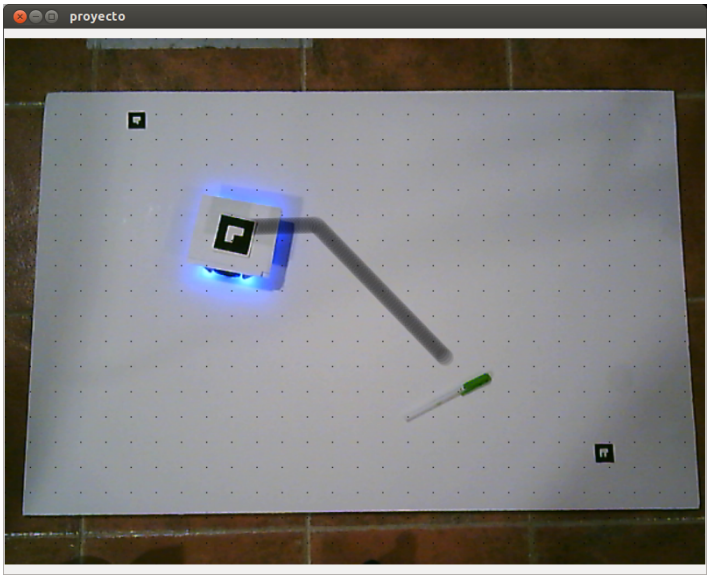

Figura A.6: Visualización del posicionamiento a través del grid mediante el algoritmo A*.

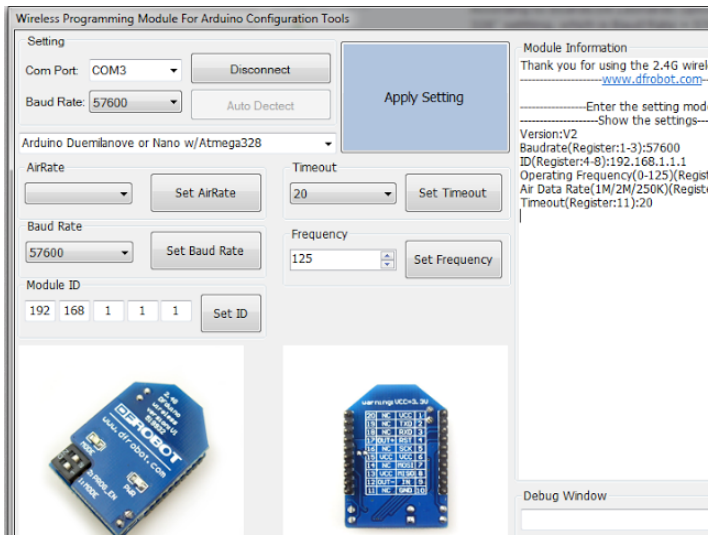

Figura A.7: Interfaz de configuración de los módulos WPM.

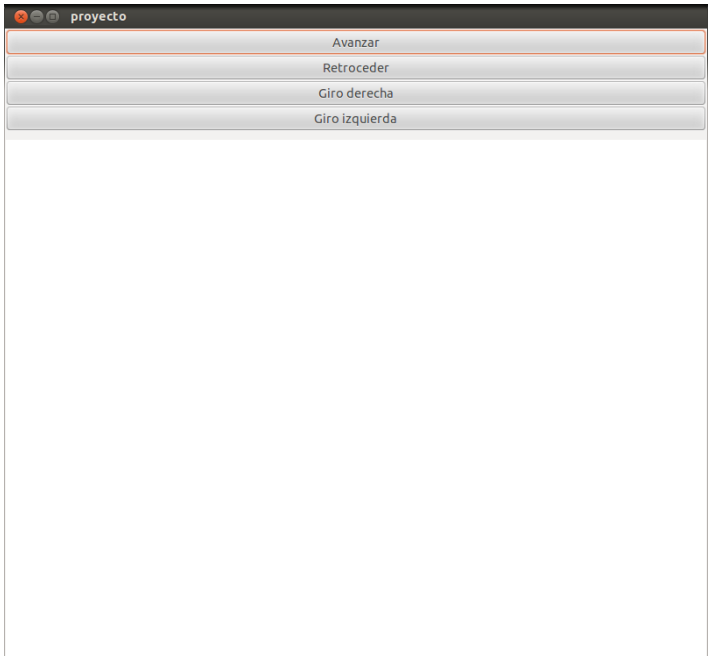

Figura A.8: Estado de la interfaz en la iteración 2.

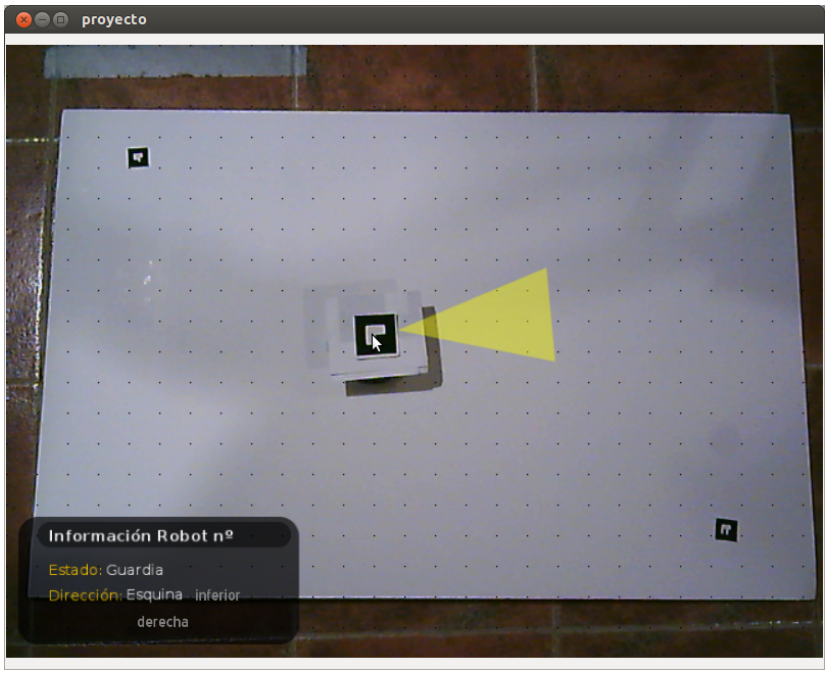

Figura A.9: Robot en estado de guardia

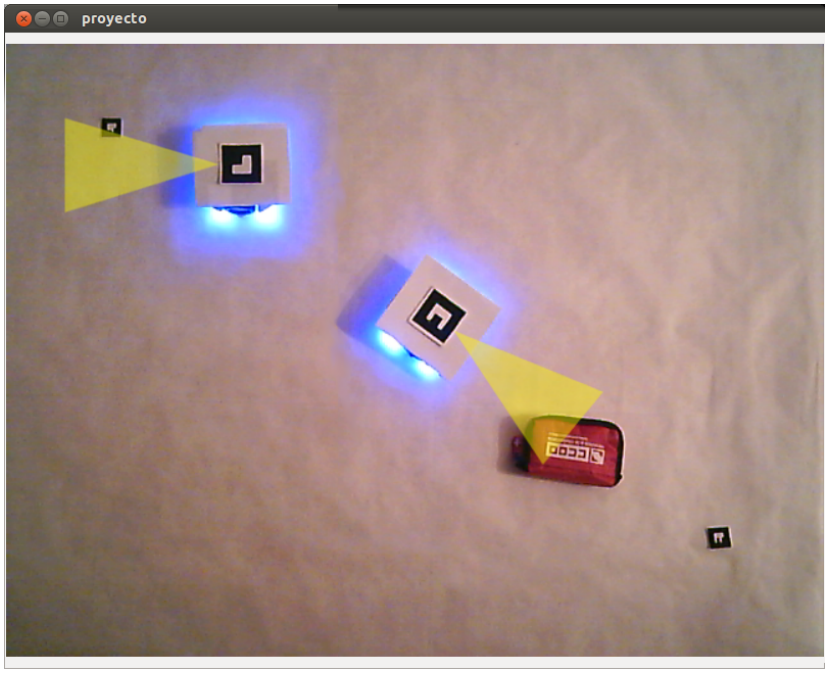

Figura A.10: Cambio de estado del robot de guardia a vigilando (estado de guardia)

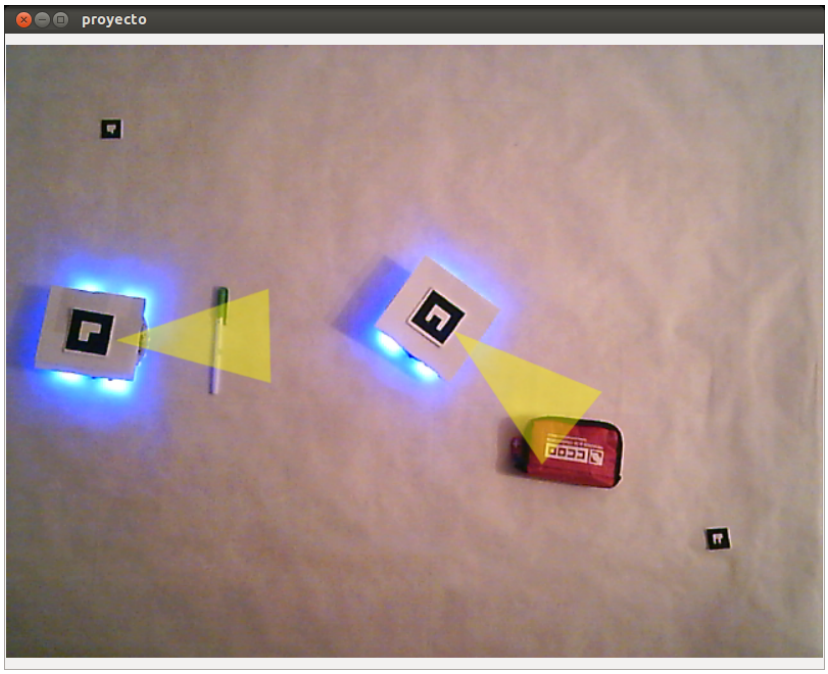

Figura A.11: Cambio de estado del robot de guardia a vigilando (estado vigilando)

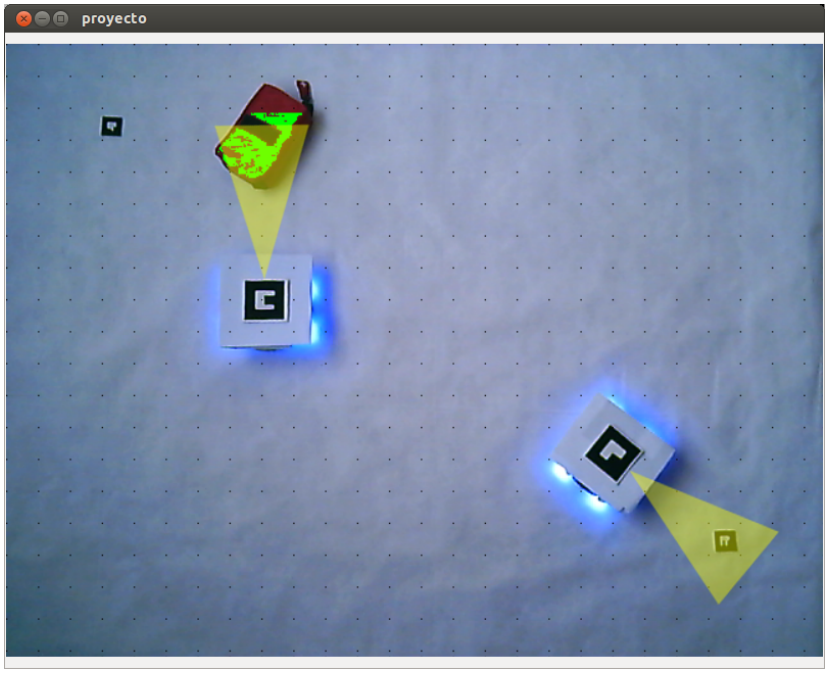

Figura A.12: Movimiento de objeto al límite del entorno

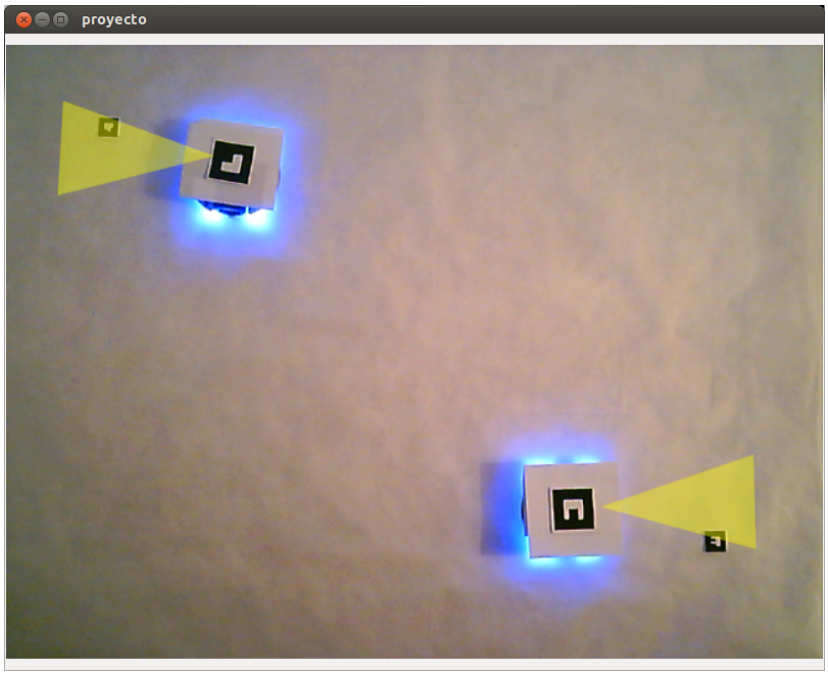

Figura A.13: Coordinación entre robots con el mismo estado, cada uno manejado por un agente diferente

Anexo B

Manual de usuario

Este anexo detalla los pasos a seguir para lanzar el sistema.

Lo primero es entrar en la carpeta contenedora del ejecutable y ejecutarlo. Si es la primera vez que se ejecuta la aplicación en ese determinado equipo aparecerá una ventana de configuración para Ogre3D como la que aparece en la figura B.1.

Una vez seleccionados los parámetros (los parámetros que vienen por defectos son los aconsejados para una correcta visualización) aparecerá la interfaz de usuario del sistema. Hasta que no se detecten las marcas de delimitación no se pondrá el funcionamiento el sistema, incluso aunque ya se encuentren robots y objetivos dentro del entorno delimitado.

Puede ser que la detección de esas marcas no sea instantánea, o que por una mala iluminación no sean detectadas. Si esto ocurriera, con aumentar la iluminación de la escena sería suficiente para que el sistema arranque.

Una vez que el sistema está lanzado el usuario no tendrá que hacer nada. La única interacción que puede hacer es la petición de la información por pantalla. Para ello sólo es necesario pinchar con el ratón sobre el elemento requerido.

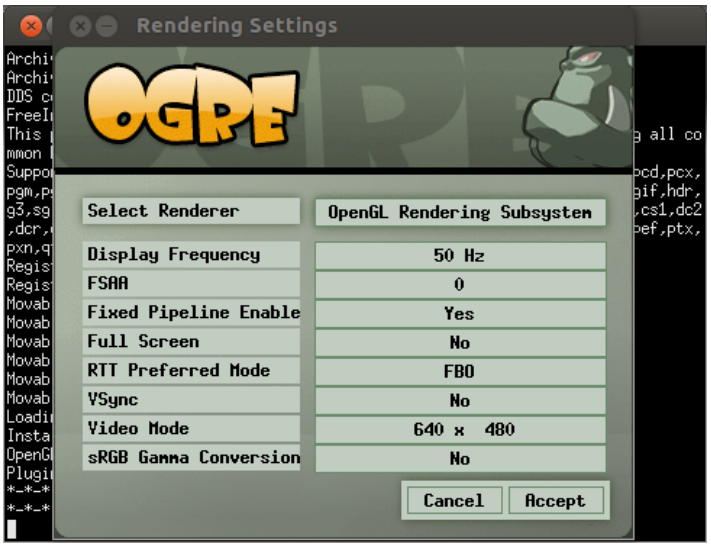

Figura B.1: Pantalla de configuración de Ogre3D

Anexo C

Manual de desarrollo

Este anexo detalla los pasos necesarios para la integración de nuevos comportamientos de los robots.

C.1 Modificación en el coordinador

Para definir un nuevo comportamiento en el robot lo primero que hay que hacer es introducir la detección de los eventos necesarios para disparar el nuevo comportamiento. Para ello hay que modificar la clase Coordinador, en la cual se definen los comportamientos en un primer nivel.

Si para la nueva funcionalidad es necesaria la generación de un evento relacionado con los robots, se modificará la clase ARTKDetector, que es la encargada de detectar las marcas identificativas de los robots. Si por el contrario, el evento necesario es un evento relacionado con los objetivos, entonces se modificará la clase VideoManager.

- **ARTKDetector**

  A la hora de generar un evento relacionado con la detección de las marcas, el lanzamiento de dicho evento se realizará dentro de la función detectMark. En esta función se controla tanto la situación actual de las marcas como un historial de dichas marcas.

  A partir de ahí se puede modificar y generar un nuevo evento y lanzar dicho evento a la clase coordinado mediante la notificación a la clase actualizar.

- **VideoManager**

  Cuando el evento necesario tiene relación con la detección de los objetos, entonces se modificará la clase VideoManager. Para ello se puede modificar la función color, si lo que se quiere es la detección de un nuevo objetivo, y la función DrawCurrentFrame, que es la encargada de procesar la imagen pixel a pixel.

  Desde este función se pueden introducir cambios como el procesamiento en un espacio de colores diferente, o la segmentación de objetivos para la detección de varios objetos del mismo color.

La clase coordinadora es la que recibe el nuevo evento y se encarga de generar el inicio del comportamiento, siendo la que da indicaciones a los agentes. En la función Actualizar se

recibirá el nuevo evento (como un número entero definido con antelación). Además también se recibe el identificador del elemento que genera el evento. Una vez recibido el nuevo evento, es esta la función que se encarga de generar las llamadas necesarias a los agentes para el nuevo comportamiento. Es en esta función don de se realizan los cálculos necesarios para la coordinación de los robots relativa al nuevo comportamiento.

A partir de este momento se hace necesaria la modificación de los agentes para que estos traten las llamadas del coordinado con el comportamiento deseado.

C.2 Modificación en los agentes

Cuando el evento llega a un determinado agente, este llega a través de la función planifica, que es la encargada de realizar la planificación del agente. El nuevo evento detectado será otro número entero definido en la clase. Para la definición del nuevo comportamiento solo se hará necesaria la incorporación de una nueva condición en el cuerpo de la función con la condición de que el evento detectado sea el del nuevo comportamiento.

Una vez hecha la comprobación se incluirá el código necesario para el nuevo comportamiento dentro de la nueva cláusula del if de esta función.

Bibliografía

[Azu97] R.T. Azuma. *A Survey of Augmented Reality*. Teleoperators and Virtual Environments, 1997.

[Bam08] Il Bambino. *Una introducción a los robots móviles*. 2008.

[Bur00] W. Burgard. *Collaborative multi-robot exploration. Robotics and Automation*. IEEE International Conference on. Vol. 1, 2000.

[BW97] B. G. Batchelor y P. F. Whelan. *Intelligent vision system for industry*. Springer, 1997.

[CMS83] S. Cammarata, D. McArthur, y R. Steeb. Startegies of Cooperation in distributed Problem Solving. En *Proc. of the 8th International Joint National Conference on Artificial Intelligence*, páginas 767–770, 1983.

[Cor79] D.D. Corkill. Hierarchical Planning in distributed environment. En *Proc. of the 6th International Joint National Conference on Artificial Intelligence*, páginas 168–175, 1979.

[Fer09] David Vallejo Fernández. *Service-Oriented Multi-Agent Architecture for a Cognitive Surveillance System*. PhD thesis, University of Castilla-La Mancha, 2009.

[GAMC02] N. Gordon, M. Arulampalam, S. Maskell, y T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. En *IEEE Trans. on Signal Processing*, 2002.

[Geo83] M.P. Georgeff. Communication and action in multi-agent planning. En *Proc. of the 3rd National Conference of Artificial Intelligence*, páginas 125–129, 1983.

[GHJV96] E. Gamma, R. Helm, R. Johnson, y J. Vlissides. *Dessign Patterns*. Addison-Wesley, 1996.

[GTO04] P. Gil, F. Torres, y F.G. Ortiz. Detección de objetos por segmentación multinivel combinada de espacios de color. En *XXV Jornadas de Automática Ciudad Real*, 2004.

[Hen04] Michi Henning. A new approach to object-oriented middleware. *Internet Computing, IEEE*, 8(1):66–75, 2004.

[JJB+08] M.C. Juan, D. Joele, C. Botella, R. Baños, M. Alcañiz, y Ch. Van der Mast. The use of a visible and/or an invisible marker augmented reality system for the treatment of phobia to small animals. En *Annual review of Cybertherapy and Telemedicin*, 2008.

[KAK+03] A. Koschan, B. Abidi, S. Kang, J. Paik, y M. A. Abidi. Real-time video tracking using ptz cameras. En *Proc. of SPIE 6th International Conference on Quality Control by Artificial Vision*, 2003.

[MF82] D. Mark y W. H. Freeman. *Vision*. McGraw-Hill, 1982.

[MK94] P. Milgram y F. Kishino. *A taxonomy of mixed reality visual displays*. Special issue on Networked Reality, 1994.

[Mu3] J. M. Muñoz. Realidad Aumentada, realidad disruptiva en las aulas. En *Boletín SCOPEO Nº 82*, 2013.

[NUG+08] M. Nakamoto, O. Ukimura, I.S. Gill, A. Mahadevanand T. Miki, M. Hashizume, y Y. Sato. Medical Imaging and Augmented Reality. En *4th International Workshop Tokyo, Japan*, 2008.

[Ote06] J.L. Días Otegui. Sistemas de detección perimetral en infraestructuras penitenciarias. En *Congreso Penitenciario Internacional*, 2006.

[PlC01] G. Pajares y J. D. la Cruz. *Visión por computador Imágenes Digitales y Aplicaciones*. Rama, 2001.

[Ree03] T. Reenskaug. *The Model-View-Controller (MVC)*. Its Past and Present, 2003.

[RN04] S.J. Russell y P. Norvig. *Inteligencia Artificial: Un enfoque moderno*. Pearson, 2004.

[Smi04] G.J.D. Smith. *Behind the screens: Examining constructions of deviance and informal practices among cctv control room operators in the UK*. Surveillance and Society, 2004.

[SSC06] S.Tu, S.Weng, y C.Kuo. Video object tracking using adaptive kalman filter. En *Visual Communication and Image Representation*, 2006.

[TBP03] T.Ellis, B.James, y P.Rosin. A novel method for video tracking performance evaluation. En *Pattern Recognition*, 2003.

[VSS07] V.Chandran, S.Denman, y S.Sridharan. Abandoned object detection using multi-layer motion detection. En *Proceeding of International Conference on Signal Processing and Communication Systems*, 2007.

[VV05a] M. Valera y SA Velastin. Intelligent distributed surveillance system: a review. En *Vision, Image and Signal Processing*, 2005.

[VV05b] M. Valera y S.A. Velastin. *Intelligent distributed surveillance systems: a review*. IEE Proceedings Vision, Image and Signal Processing, 2005.

Este documento fue editado y tipografiado con LaTeX
empleando la clase **arco-pfc** que se puede encontrar en:
`https://bitbucket.org/arco_group/arco-pfc`