UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO

TFG Grado en Ingeniería Electrónica Industrial y Automática:

PROGRAMACIÓN DE ROBOT MÓVIL PARA VIGILANCIA CON VISIÓN ARTIFICIAL

Autor: Abel Martínez Martínez

Director:
Houcine Hassan Mohamed

Codirector:
Carlos Pascual Domínguez Montagud

Julio 2014

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Dedicatòria

Dedicat al meu volgut fill Marc.

Que vingueres al món dies abans de l'inici de la meua carrera. El teu creixement m'ha acompanyat durant estos anys. Estic molt orgullós dels teus avanços i de la felicitat que tens. Espere que este treball et servisca com a referència a la teua vida escolar i professional. Però per a tot açò falta molt, així que de moment com sempre et dic "Juga molt i passa-ho bé".

Agradecimientos

Me gustaría agradecer en este apartado a todas las personas que en algún modo han estado conmigo durante la carrera y en este trabajo.
A mi familia y en especial a mi madre, por darme el apoyo necesario para continuar trabajando día a día.
A los directores del trabajo, por permitirme trabajar con vosotros en este proyecto tan interesante. También por hacer que me sienta acompañado durante todo el desarrollo del trabajo. Por atender las dudas que me han surgido en cualquier momento.
Mi agradecimiento también va dirigido a los compañeros del grupo ARA, con los que he compartido muchos momentos de diversión, tensión y alegría. Compaginando estudios con proyectos extra-escolares, deportes y alguna que otra quedada.
Agradecer también a todos los profesores y compañeros de la ETSID.
Agradezco también a los compañeros que he conocido en el DISCA. Han sido muchos los ratos buenos que hemos pasado juntos, y espero que con continúen siendo en el futuro. También dedico este trabajo a mis amigos. Amigos recientes y antiguos que han estado conmigo en todo momento, y porque este éxito es también suyo.

Índice general

Resumen

En este trabajo se ha implementado un robot de vigilancia con visión artificial. El robot vigilante realiza las funciones de un portero de vigilancia. El robot tiene una base de datos con los usuarios que tienen permiso para entrar en al zona restringida. En la base de datos se introduce el nombre del usuario y fotografías del mismo desde diferentes ángulos. Cuando el robot detecta a un usuario que está en la base de datos, le permite el acceso y lo acompaña.

La aplicación se ha programado en lenguaje C# y se ha utilizado código propio y de terceros. Para el desarrollo del trabajo se ha programado un módulo de creación de mapas de ocupación. Este módulo utiliza información de los sensores para generar un mapa del entorno.

También se ha diseñado una interfaz de control desde la que el usuario puede controlar el robot, indicar los puntos de vigilancia o añadir usuarios a la base de datos. La aplicación combina diferentes sensores para hacer un uso eficiente de la energía.

Por último se ha realizado una evaluación del prototipo final, además de un presupuesto de la realización del trabajo.

Palabras clave: Robot vigilante, reconocimiento facial, mapa de ocupación, cámara IP, programación C#, software distribuido.

Resum

En este treball s'ha implementat un robot de vigilància amb visió artificial. El robot vigilant realitza les funcions d'un porter de vigilància. El robot té una base de dades amb els usuaris que tenen permís per a entrar a al zona restringida. A la base de dades s'introdueix el nom de l'usuari i fotografies desde diferents angles. Quan el robot detecta a un usuari que està a la base de dades li permiteix l'accés y l'acompanya.

L'aplicació s'ha programat amb llenguatge C# y s'ha utilitzat codi propi i de tercers. Per al desenvolupament del treball s'ha programa un mòdul de creació de mapes d'ocupació. Este mòdul utilitza informació dels sensors per a generar un mapa de l'entorn.

També s'ha dissenyat una interfície de control desde la qual l'usuario pot controlar el robot, indicar els punts de vigilància o afegir usuaris a la base de dades. L'aplicació combina diferents sensors per a fer un us eficient de la energia.

Per últim s'ha realitzat una evaluació del prototip final, ademés d'un pressupost de la realització del treball.

Palaules clau: Robot vigilant, reconeixement facial, mapa d'ocupació, càmera IP, programació C#, programari distribuït.

Abstract

In this work were implemented a surveillance robot with artifitial vision. The robot perform gatekeeper functions. The robot has a database with the user list that are allowed to enter in the restricted area. User's name and photos from different angles are added to the database. When the robot detects a user existing in the database, it allows to enter and guide the user to the restricted area.

The application is programmed in C# language. It is used own code and third-party code. For the development of the work has been programmed a occupancy map module. This module uses information from the sensors to generate a map of the environment.

An user interface is designed. The user can control the robot, indicate monitoring points and also add users to the database. The application combines several sensors to perform efficient use of energy.

Finally it is presented an assessment of the final prototype, and a budget of job costs.

Keywords: Surveillance Robot, face recognition, occupancy grid, IP camera, C# code, distributed software.

Capítulo 1

Introducción

En el presente artículo se va a exponer el trabajo que se ha realizado para programar un robot móvil para que realice tareas de vigilancia utilizando visión artificial. A continuación se introducen los diferentes conceptos que se van a tratar en este trabajo.

Se va a utilizar una plataforma robótica comercial, a la que se añadirán los componentes software y hardware necesarios cumplir su misión. Su misión consiste en controlar el acceso a una zona restringida, de modo que solo los usuarios que se encuentren en la base de datos podrán pasar de la entrada. Además de permitir el paso, el robot acompañará a las personas que se encuentren en la base de datos al despacho donde desee ir el usuario.

Uno de los problemas habituales que se encuentran las plataformas robóticas es la localización, para que un robot realize con éxito labores más complejas es necesario que tenga una localización precisa en el entorno. Una de las soluciones comúnmente empleada es el uso de SLAM. SLAM es el acrónimo del inglés "Simultaneous Location And Mapping", que significa localización y creación de mapas simultanea. Lo que hacen los robots que utilizan SLAM es que cuando exploran un entorno desconocido, crean un mapa que les sirve para localizarse. Dentro del SLAM existen diferentes algoritmos que ofrecen solución al problema.

El SLAM también puede utilizar imágenes para localizarse en el entorno, este es un tema que todavía se está investigando en diferentes centros de investigación. Las técnicas de visión por ordenador aparecieron en los años 50 y la mayoría de algoritmos de visión artificial que utilizamos actualmente se desarrollaron en los años 80. En la aplicación se utilizará la visión artificial para detectar las caras de los usuarios, y compararlas con las caras guardadas en la base de datos.

1.1. Pliego de condiciones

En el laboratorio del DISCA se dispone de una plataforma robótica. Se desea programar esta plataforma robótica para que realice labores de vigilancia. Para ello se diseñarán una serie de módulos necesarios para el vigilante. Los módulos que se requieren son:

- Reconocimiento facial.
- Planificador de trayectorias.
- Interfaz usuario.
- Exploración.
- Mapa de ocupación.
- Vigilancia.

También se diseñará una aplicación genérica para el robot de vigilancia. El robot será capaz de restringir el acceso a un determinado área a los usuarios que tengan permiso para acceder a él. Para ello se delimitará un área en la cual el robot realizará las labores de vigilancia. Se programará una barrera de entrada que representará el punto de control. Cuando una persona intente acceder a la zona restringida, el robot deberá detectarla, informando a la persona de que está siendo detectado para acceder. El agente buscará su imagen en la base de datos de usuarios con acceso, y le dará acceso en el caso de que encuentre su imagen en la base de datos. Si no la encuentra guardará en el sistema algunas de sus imágenes y le informará de que no tiene permiso para acceder al área restringida. Tras esta aplicación genérica de robot de vigilancia, se plantea una aplicación del robot de vigilancia en hospitales. El robot dispondrá de una base de datos con los pacientes del día y las consultas donde deben ir. El robot reconocerá a cada uno de los pacientes, para acompañarlo a la consulta adecuada.

Para verificar el funcionamiento de los módulos, así como el de la aplicación de vigilancia genérica y vigilancia en el hospital, se plantearán los siguientes experimentos:

1. Verificación del módulo de odometría.

2. Verificación del módulo de reconocimiento facial.

3. Validación de la aplicación de vigilancia genérica.

4. Validación de la aplicación de vigilancia en hospitales.

En la Figura 1.1 se puede ver una imagen de la zona en la que se ha probará la aplicación.

Figura 1.1: Zona vigilada por el robot.

Capítulo 2

Estado del arte

A continuación se pretende introducir al lector en los robots de servicio y dar una visión global de las diferentes aplicaciones que realizan dichos robots, así como de un informe de la situación actual del mercado. Por último, se presentan algunas alternativas software y hardware para realizar tareas de vigilancia con robots de servicio.

2.1. ¿Que son los robots de servicio?

Un robot de servicio es un sistema robótico que es utilizado para ayudar y colaborar con las personas. Los robots de servicio son móviles o movibles y realizan tareas automática o semiautomáticamente. Su propósito no es trabajar en industria o fabricación. Están construidos para realizar servicios para el bienestar y comodidad de las personas. Existen robots de servicio de diferentes tipos y formas, y trabajan en varios campos desarrollando diferentes servicios para humanos o el bienestar de los humanos. Un modo de dividir robots de servicio es según el cliente al que va destinado, distinguiendo entre robots de servicio profesionales y robots de servicio domésticos. Este método es utilizado por la IFR, "International Federation of Robotics", en el "World Robotics Report". La IFR es una organización profesional, sin animo de lucro que promueve la investigación y desarrollo en todos los aspectos de la robótica. Simultáneamente actúa como lazo entre organizaciones, instituciones y gobiernos interesados en robótica. La IFR es una fuente de información bien conocida en robótica. Una significativa cantidad de la información de esta sección se ha obtenido de fuentes IFR.

2.2. Robots de servicio profesionales

Los robots de servicio profesionales están orientados a las empresas. Están construidos para el propósito de realizar tareas específicas en ambientes laborales. Estos robots no realizan servicios personales para ninguna persona, en lugar de ello trabajan para mejorar el bienestar de las personas realizando servicios necesarios. La IFR divide los robots de servicio profesionales en distintas categorías, incluyendo:

1. Robots subacuáticos.
2. Robots de limpieza.
3. Robots de defensa, rescate y seguridad.
4. Robots de construcción y demolición.
5. Robots para agricultura.
6. Plataformas robóticas móviles.
7. Robots de medicina.
8. Robots de logística.
9. Otros robots.

1. Robots subacuáticos

Los robots subacuáticos están construidos para resistir y operar bajo el agua. Son utilizados en exploración y navegación submarinas. Además, compañías petrolíferas y de gas los utilizan para controlar y reparar tuberías submarinas. También son utilizadas por otras compañías para inspeccionar sistemas de comunicación submarinos y otros equipos.

2. Robots de limpieza

Existen robots de limpieza de varios tipos diseñados para completar diferentes tareas. Algunos robots de limpieza están diseñados para escalar por los edificios para limpiar las ventanas, las paredes y los suelos. Otros inspeccionan y limpian cañerías, tuberías, tanques, etc. Otros están diseñados para limpiar piscinas y por lo tanto comparten algunas características con los robots subacuáticos. Básicamente, los robots de limpieza realizan tareas

y trabajos que resultan difíciles y/o peligrosos para las personas. Finalmente los robots de limpieza profesionales no realizan servicios domésticos para familia. Estos robots realizan tareas de limpieza a gran escala.

3. Robots de defensa, rescate y seguridad

Los robots de esta categoría también realizan variedad de tareas. En el sector de defensa, las compañías de robots diseñan sistemas que pueden lanzar misiles, observar y supervisar. Además, hay muchos sistemas robóticos que ayudan en combates aéreos e incluso en desactivación de explosivos, permitiendo a las personas permanecer seguras en la distancia. También hay otros robots que se utilizan para seguridad y vigilancia. Estos están diseñados para patrullar automáticamente y controlar dentro y fuera de las empresas. Pueden recorrer el área programada sin supervisión.

4. Robots de construcción y demolición

Los robots de construcción y demolición son utilizados en obras para derribar paredes y retirar suelos. Son utilizados para romper el cemento y derribar bloques de oficinas enteros. Están diseñados con herramientas para completar su trabajo. Pueden taladrar a través del cemento, azulejos, tierra, etc. Además, están diseñados para trabajar en una variedad de ambientes, incluso nucleares o plantas de materiales peligrosas.

5. Robots para agricultura

Los robots para agricultura se utilizan en exteriores. Trabajan en el campo ofreciendo servicios y productos más rápida y efectivamente. Por ejemplo, pueden distribuir pesticidas y fertilizantes, peligrosos, o alimentar ganado. Además, estos robots trabajan en la industria forestal, desplazándose entre obstáculos, talando y remolcando troncos. Otros robots están diseñados para extraer y procesar la leche de vacas, ovejas y cabras. Los robots para agricultura ofrecen velocidad y precisión en aplicaciones de exteriores.

6. Plataformas robóticas móviles

Las plataformas móviles son ampliamente utilizadas en diferentes sectores. Diferentes prototipos se han diseñado para transportar personas en lugares públicos y también para actuar como conductores. Las plataformas robóticas también transportan bienes y productos en almacenes, factorías y

otras empresas. Estos robots también se utilizan en entornos de defensa y seguridad.

7. Robots de medicina

Los robots de medicina asisten a la personas en el campo médico, ofreciendo mayor precisión y exactitud y a menudo menor tiempo de intervención. Estos robots ayudan en rehabilitación y terapias. Además estos robots se utilizan como asistentes quirúrgicos. A parte de aumentar la precisión y destreza que las personas pueden alcanzar, pueden ofrecer visualización 3D ayudando a los cirujanos y doctores todavía más.

8. Robots de logística

Los robots de logística trabajan en almacenes, fábricas, oficinas y hospitales. Ellos actúan como sistemas de repartos entregando correo, información y alimentos. Transportan bienes y productos. Los robots trabajan en almacenes gestionando, manejando y distribuyendo bienes. Los robots de logística se encargan a menudo de manejar y organizar piezas en el entorno de trabajo. Estos robots pueden estar total o parcialmente automatizados.

9. Otros robots

Los robots clasificados en esta sección representan una pequeña porción del mercado. Son robots poco utilizados, que no encajan en ninguna de las categorías anteriores.

2.3. Análisis del mercado de robots de servicio profesionales

La IFR produce un estudio en profundidad y comprensivo de robots industriales y de servicio anualmente. Dicho estudio se titula "World Robotics Report"[11]. El estudio contiene información detallada y estadísticas sobre tipos de robots, variables económicas y aspectos técnicos. Además el artículo describe el mercado en varios países. Las estadísticas que se muestran a continuación se han extraído del "World Robotics Report 2013".

Según el "World Robotics Report 2013", en 2012 se vendieron un total de 16.067 robots de servicio profesionales, con un impacto en el mercado de 3.42 mil millones de dolares. La categoría con más unidades vendidas es la de robots de defensa, con 6.200 unidades. Lo que representa casi el 40 % de

todos los robots de servicio vendidos en 2012. La mayor parte de los robots de defensa son vehículos aéreos no tripulados (UAV), con un total de 5.453 unidades vendidas. No obstante el número de estos robots podría ser muy superior. Los robots para agricultura consiguen el 33 % del total, alcanzando 5.300 unidades vendidas. Las ventas de robots para medicina ha supuesto el 8 % de los robots de servicio profesionales, con 1.308 unidades. Dentro de esta categoría destacan los robots de asistencia en cirugía y terapia con 1.053 ventas. Fueron instalados 1.376 sistemas automatizados de logística, lo que representa el 9 % de las ventas de robots de servicio profesionales. Se vendieron 796 vehiculos guiados automáticamente para entornos industrializados y 557 para no industrializados. Otros robots de servicio profesionales fueron robots de construcción y demolición, robots de limpieza profesionales, sistemas de inspección y mantenimiento, robots de rescate y seguridad, plataformas móviles y robots submarinos.

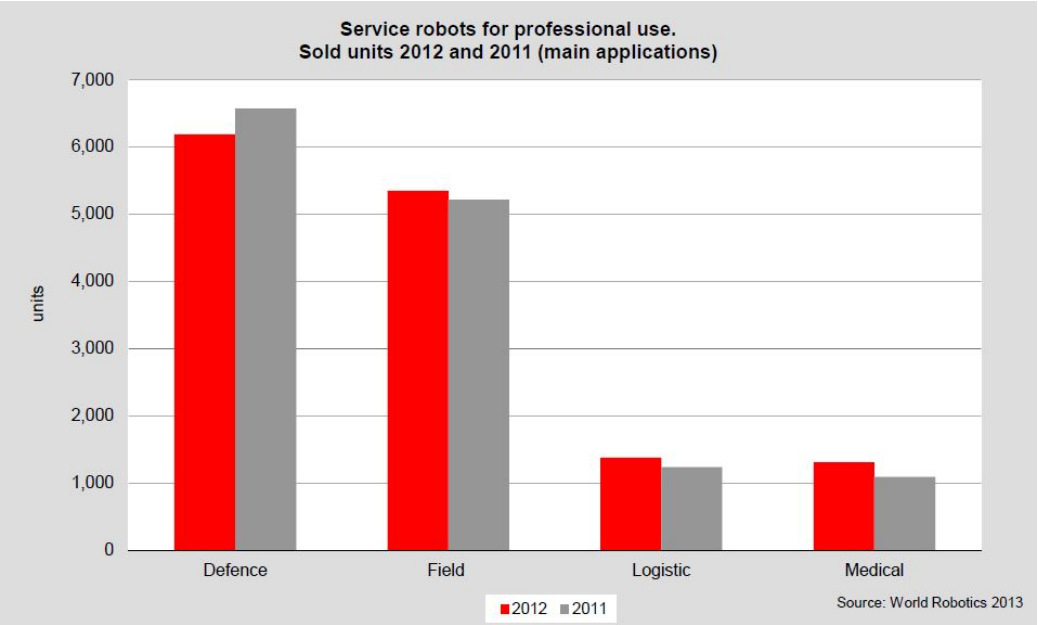

Figura 2.1: Ventas robots de servicio profesionales 2011/2012.

2.4. Soluciones para robots de vigilancia

A continuación se listan diferentes alternativas software y hardware que podrían incorporarse en un robot de vigilancia con visión artificial.

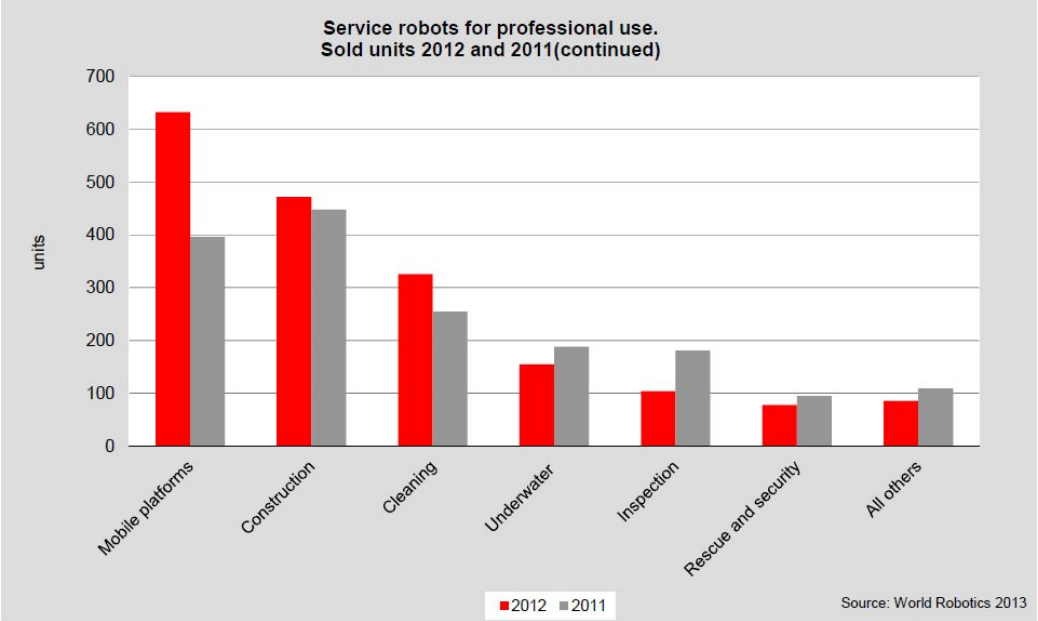

Figura 2.2: Ventas robots de servicio profesionales 2011/2012.

2.5. Plataformas robóticas alternativas

Robots con ruedas

Configuración ackerman

Los robots construidos con configuración ackerman disponen al menos de dos ruedas fijas alineadas en la misma dirección y de otras dos ruedas que giran coordinadamente. La tracción de los vehículos en configuración ackerman puede ejercerse sobre las ruedas direccionales o fijas indistintamente, aunque también se pueden accionar la totalidad de las ruedas, estos vehículos son conocidos como de tracción total. El robot Summit [10]desarrollado por Robotnik dispone de configuración ackerman. Además está diseñado para labores de vigilancia, educación, militares, monitorización remota o acceso a zonas peligrosas. El robot cuenta con una cámara PTZ para la transmisión de vídeo en tiempo real. Además está diseñado para funcionar en el interior y en el exterior. Se puede ver el robot Summit en la Figura 2.3

Figura 2.3: Plataforma robótica Summit.

Configuración diferencial

El robot que se ha utilizado en el desarrollo de este trabajo tiene configuración diferencial, pero existen más fabricantes que ofrecen robots con esta configuración. En la Figura 2.4 se puede ver el robot PowerBot, que dispone de configuración diferencial. Este robot está diseñado para llevar a realizar tareas de mapeado, reconocimiento, localización, monitorización, reconocimiento, visión, manipulación y cooperación. Una característica de este robot es el nivel de personalización sobre el mismo. Se puede elegir entre algunos de los siguientes accesorios: Ordenador de abordo, comunicación Ethernet inalambrica, giróscopo de corrección, módulo integrado con inclinación en dos ejes, zoom y cámara, sensor láser y paquete de navegación, cámara estéreo a color, módulo de audio y voz, entre otros.

Robots voladores

Otro sector en el que los robots están empezando a demostrar su potencial es el de la vigilancia desde el aire. Las labores que realizan son diversas y van desde el control de incendios, labores de búsqueda de desaparecidos o de defensa entre otras.

Aviones

El uso de aviones para vigilancia presenta ventajas y desventajas. Entre las ventajas cabe destacar la velocidad que pueden alcanzar y la economía

Figura 2.4: Plataforma robótica PowerBot.

de su vuelo. Alguna desventaja podría ser que no pueden detenerse en el aire o que necesitan una pista para despegar. En la Figura 2.5 se puede ver un avión no tripulado comercial [12]. Tiene un ancho de 1.2 m y un largo de 2.4m, su peso es de 2.72 kg. Tiene una autonomía de 55 minutos y vuela a una velocidad media de 60km/h.

Cuadcópteros

La aparición de los potentes motores bruce-less y las baterías de alto rendimiento de litio han permitido el desarrollo de los cuadcópteros. Los quadcópteros resultan más estables que los helicópteros convencionales. Actualmente existen controles tan precisos que permiten su vuelo incluso en interiores. En la Figura 2.6 se muestra un cuadcóptero comercial para su uso en investigación y enseñanza [13]. Este robot viene preparado para incorporar una cámara y su peso es de 4.5kg. El fabricante ofrece soporte para multitud de opciones de cámara, como por ejemplo cámaras térmicas, cámaras con zoom o cámaras de alta definición.

Figura 2.5: Avión no tripulado MicroPilot.

Robots con patas

Bipedos

Desde la primera aparición de robots en novelas y películas se ha perseguido el sueño de fabricar robots con apariencia y comportamiento similares a los humanos. Este sueño está cada día más cerca gracias a trabajos como los de la empresa Pal Robotics, que ha desarrollado el robot bípedo [14] de la Figura 2.7. Este robot que mide 1.65m y pesa 80kg es capaz de levantar hasta 10kg de carga extra o caminar a 1.5k/h. El desarrollo de este tipo de robots tiene la motivación en que a las personas nos resulta más cómodo interactuar con robots que tienen nuestra misma apariencia. El robot Reem-c está indicado para labores de telepresencia, asistencia a personas mayores o con discapacidades y ayuda para realizar tareas.

2.6. Sensores alternativos

Brújula digital

Una brújula digital es un dispositivo electrónico que permite conocer la orientación absoluta del robot. Para conocer la orientación del robot estos

Figura 2.6: Cuadcóptero.

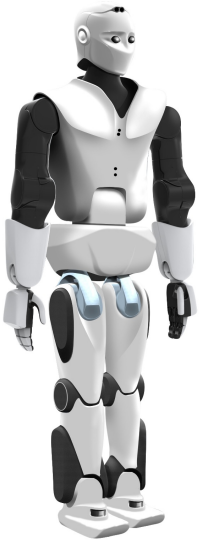

Figura 2.7: Robot para investigación Reem-c.

dispositivos son capaces de medir con precisión el campo magnético de la tierra, y calcular la orientación del sensor con respecto a este. Por ejemplo al brújula electrónica CMPS03 que se muestra en la Figura 2.8 tiene una precisión de 3-4 grados y una resolución de décimas. Además ofrece distintas opciones de salida de datos como anchura modulada o conexión I^2C.

Láser de rango

Los sensores láser de rango son sensores ampliamente utilizados en mapeado y vehículos autónomos. Son capaces de obtener gran cantidad de puntos del entorno en muy poco tiempo. El láser de rango que se presenta en la

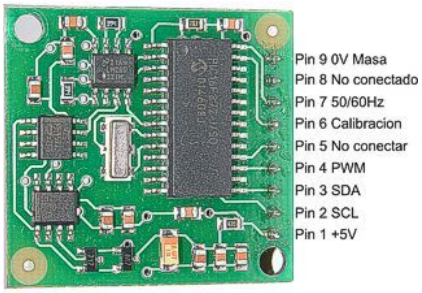

Figura 2.8: Brújula electrónica.

Figura 2.9: Láser de rango Hokuyo.

Figura 2.9 es del fabricante Hokuyo [15], es capaz de escanear 1024 puntos del entorno en 28 milisegundos. Tiene un ancho de escáner de 240 ° y su peso es de 260g.

Receptor GPS

La instalación de un GPS resulta útil cuando el robot trabaja en el exterior. Los dispositivos GPS son capaces de estimar la posición GPS con cierta precisión. Calculan la posición en función de la distancia que a la que se encuentran de los satélites.

GPS de interior

El GPS de interior también sirve para posicionar al robot en el entorno, pero su funcionamiento es algo diferente al del GPS convencional. Se trata de colocar en el entorno una balizas que sirven al robot para saber su ubicación. El robot puede calcular su ubicación en función de su orientación con respecto a cada una de las balizas o con la distancia desde el robot a cada una de las balizas. Algunas de las tecnologías que utilizan los GPS de interior son ultrasonidos combinados con algún tipo de sincronización, como por ejemplo radiofrecuencia.

Visión estéreo

Figura 2.10: Cámara estéreo comercial.

Las cámaras estéreo disponen de dos o más puntos de visión. Con este tipo de cámaras se puede calcular la distancia a la que se encuentran los objetos. Se utiliza el mismo principio que el ojo humano para obtener a partir de dos imágenes tomadas a una distancia conocida la imagen 3D. La cámara de la Figura 2.10 es una Bumblebee XB3 que tiene una resolución de 1.280 x 960 y es capaz de tomar imágenes a 15 fps.

Cámara con sensor de profundidad

Las cámaras con sensor de profundidad ofrecen una salida similar a la de las cámaras estéreo pero su funcionamiento es distinto. Para calcular la distancia a la que se encuentran los objetos, las cámaras de profundidad dibujan una rejilla invisible al ojo humano. El cálculo de la profundidad se realiza a partir de la deformación que presenta esta rejilla por la forma de los objetos. La cámara Xtion que se muestra en la Figura 2.11 funciona en un rango de 0.8 a 3.5m, y puede procesar entre 30 y 60 fps dependiendo del

Figura 2.11: Cámara con sensor de profundidad Xtion.

tamaño de las imágenes. Xtion funciona en Windows, Linux y Android y dispone de una librería de programación abierta llamada openNI.

Cámara cenital

Las cámaras cenitales son cámaras que se colocan en el techo y que se utilizan para estimar la posición del robot, o bien para detectar objetos o personas en el entorno.

Cámara térmica

El uso de cámaras térmicas como sensores aporta grandes beneficios en cuanto a fiabilidad, seguridad y prevención. Aportan información que no podría ser obtenida de otro modo, como por ejemplo la temperatura de las ruedas de un coche que se puede ver en la Figura 2.12. El uso de cámaras térmicas en robots de vigilancia puede ayudar a detectar intrusos en entornos controlados, detectar cambios de temperatura en pacientes en hospitales o detectar rápidamente focos de incendios forestales.

Figura 2.12: Imagen de coche de carreras tomada con cámara térmica.

2.7. Software alternativo

openTLD

OpenTLD es una librería de visión artificial que ofrece la posibilidad de seguir un objeto sin que sea necesario un modelo previo del mismo. Según su acrónimo del ingles "Trade, Learn and Detect", openTLD es capaz de seguir objetos, aprender de cómo evoluciona el objeto y detectar al objeto si sale del campo de visión de la cámara y vuelve a entrar nuevamente. OpenTLD fue creada en Zdenek Kalal y programada en Matlab.

Alternativas SLAM

tinySlam

Los algoritmos de SLAM tratan de resolver el problema de posicionar al robot en un entorno desconocido a partir de la información que reciben de los sensores. En los sistemas de SLAM se utiliza la información previa para corregir errores de posición debidos a deslizamientos y errores acumulados. El algoritmo tinySLAM tiene como entrada los datos de un láser de rango y los datos de la odometría del robot. En la Figura 2.13 se puede ver una imagen creada con a partir del algoritmo tinySLAM.

Alternativas control

Librerías y herramientas ROS

ROS es un entorno de trabajo para escribir código para robots, es una colección de herramientas, librerías y convenios para simplificar la tarea de

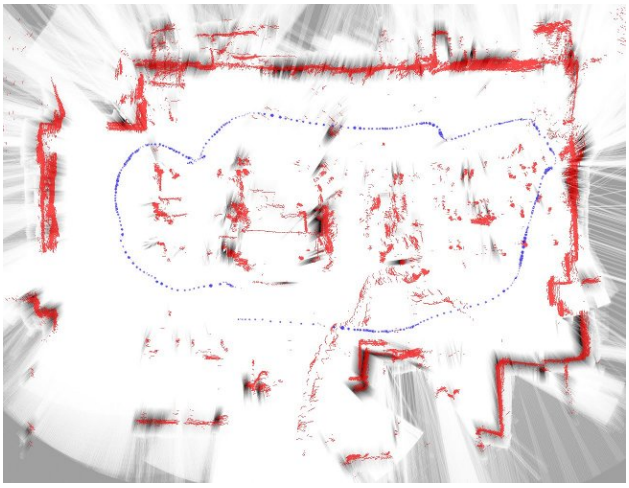

Figura 2.13: Imagen generada con tinySlam.

crear aplicaciones complejas para todo tipo de robots. Desde su aparición en 2007 ha revolucionado la fabricación de robot, siendo muchas las empresas que incorporan ROS a sus productos, como por ejemplo Fraunhofer IPA, Aldebaran Robotics, Lego NXT, Shadow Robot Company, Willow Garage, iRobot y Robotnik Automation entre otras.

Capítulo 3

Plataforma robótica

3.1. El robot

Figura 3.1: Robot X80 Pro

El robot elegido para desarrollar este proyecto es el X80 Pro del fabricante Dr. Robot [9]. Se trata de una plataforma robótica especialmente diseñada para su uso en labores de monitorización, telepresencia y navegación autónoma. El robot X80 Pro cuenta con diferentes periféricos distribuidos principalmente entre sensores, actuadores y multimedia, que se explicarán a continuación. Puede conectarse al computador a través de comunicación

serie RS-232, inalámbrica WI-FI o mediante Bluetooth.

En la Figura 3.2 se puede ver un esquema de la arquitectura Dr. Robot. Para realizar la comunicación, el PC, procesador o DSP se conectan al módulo de comunicación. Posteriormente el módulo de comunicación se conecta con el módulo de comunicación del robot por cable RS-232 o inalámbrico. El módulo de comunicación del robot se conecta a la placa multimedia PMB5010 por cable o en este caso con el sistema "board to board", que significa que una placa está introducida en la siguiente. Finalmente la placa multimedia PMB5010 está conectada a la placa de control de motores y sensores PMS5005. A la pla-

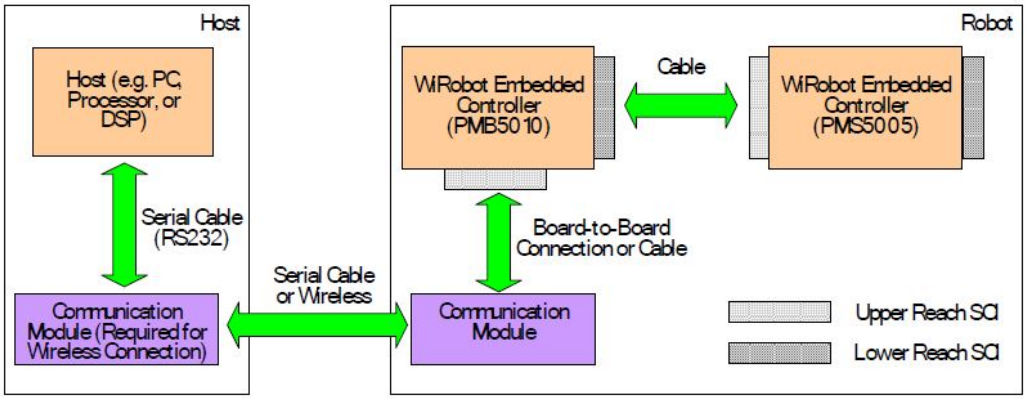

Figura 3.2: Arquitectura Dr. Robot.

ca multimedia PMB5010 de la Figura 3.3 se conectan entre otros dispositivos el altavoz y el micrófono que pueden reproducir y registrar audio continuamente, las conexiones de comunicación, la imagen de vídeo de la cámara, la conexión a la pantalla LCD o el acceso a las entradas-salidas digitales. En la Figura 3.4 está la placa de sensores PMS5005 a la cual se conectan los sensores disponibles en el robot: conexiones para sensor de corriente o de inclinación, sensor de temperatura ambiente o de presencia humana. También tiene conexiones para los sensores ultrasonidos, servomotores, codificadores rotativos de cuadratura, control de motores o retroalimentación.

3.1.1. Actuadores

Motores

El robot dispone de dos motores en configuración diferencial, estos motores tienen un par de 40kg·cm. Los motores pueden controlarse independien-

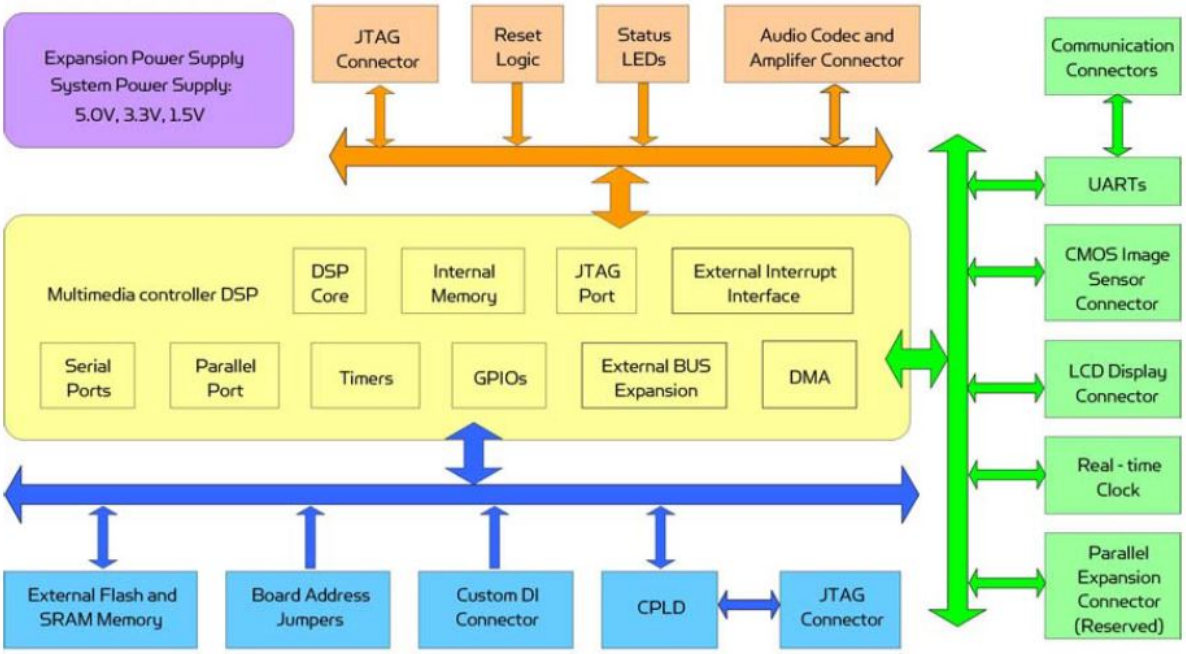

Figura 3.3: Placa multimedia PMB5010.

temente permitiendo al robot moverse en linea recta o giros.

El robot se desplaza en linea recta cuando la velocidad y sentido de ambas ruedas es idéntico. El robot realiza una trayectoria en forma de arco cuando los motores se mueven a velocidades diferentes.

Brazo Cámara

En la parte frontal del robot se encuentra el brazo robótico, tiene una cámara integrada en su extremo, y se puede utilizar para orientar la cámara al punto o zona de interés. Está construido con dos servos que aportan al brazo dos grados de libertad.

Los servos que otorgan movimiento al brazo tienen internamente un sistema electrónico de control. La señal de entrada que reciben los servos es una señal modulada de ancho de pulso (PWM). El periodo de este tipo de señales suele ser de 20ms. El tiempo a nivel alto en de estas señales suele ser de entre 1 y 2 ms, el control interno del servo posiciona la salida a 0 y $180^{\underline{o}}$ respectivamente.

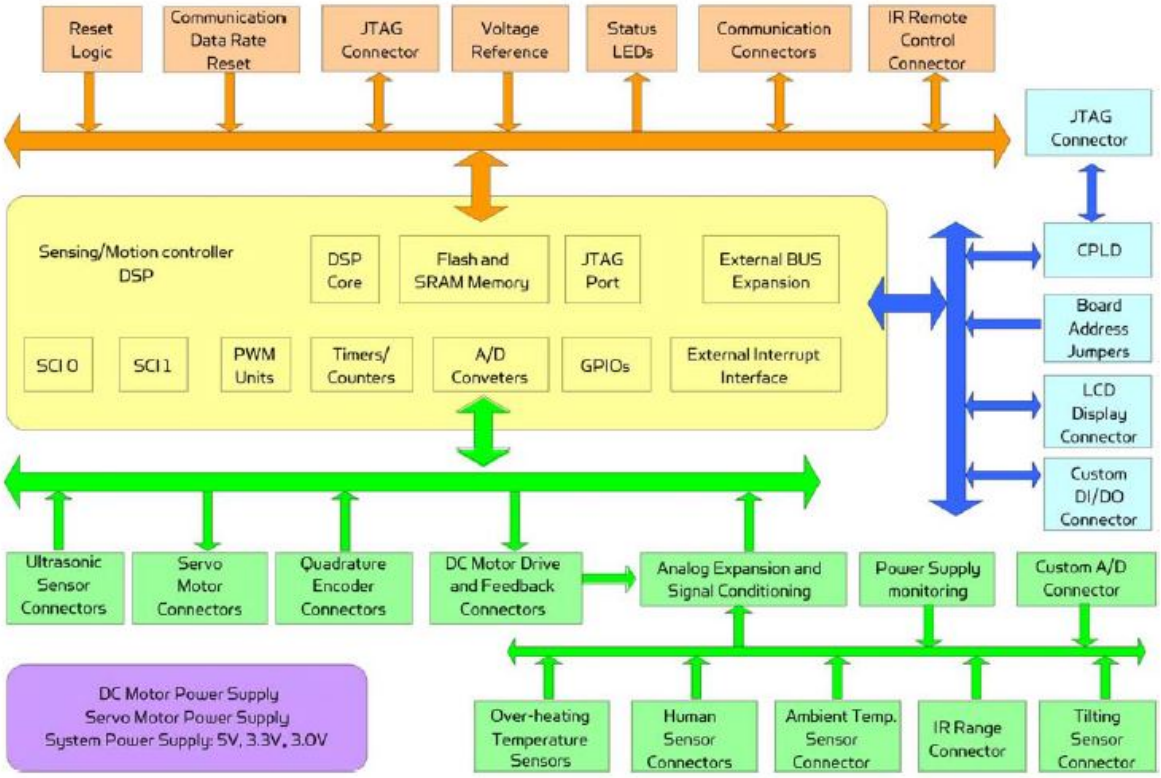

Figura 3.4: Placa sensores PMS5005.

3.1.2. Sensores

Codificadores rotativos

En cada una de las ruedas hay un codificador rotativo que permite conocer la posición de las ruedas en cada momento, tiene una resolución de 800 cuentas por vuelta.

Los codificadores rotativos se utilizan para ofrecer una retroalimentación de la posición de las ruedas. Así es posible compensar factores del entorno como desniveles o irregularidades en el suelo disminuyendo el error que se obtendría en bucle abierto.

Sensores de Intensidad y voltaje

El robot tiene sensores de intensidad conectados a la batería y a los motores. Es posible por tanto medir la intensidad que consume el robot y la intensidad que consumen los motores. También tiene sensores de voltaje conectados a la batería para controlar su nivel y gestionar su recarga en el momento óptimo.

Los sensores de intensidad pueden utilizarse para implementar sistemas rebundantes de detección de colisiones, en conjunto con los sensores de proximidad. Por ejemplo un incremento inesperado en el sensor de intensidad podría significar que el robot ha colisionado, ha subido algún desnivel o bien alguna de sus piezas se ha deteriorado.

Los sensores de voltaje se pueden utilizar para controlar el consumo de batería del robot, y en trabajos con múltiples robots, podría gestionarse de un modo eficiente la recarga de las baterías de los diferentes robots.

Sensores de Inclinación-Aceleración

El robot tiene un sensor de inclinación de dos ejes con los cuales se puede conocer la inclinación del robot. Este sensor está alineado al eje de las ruedas y perpendicular a este. El sensor de inclinación puede captar aceleraciones estáticas relacionadas con la fuerza de la gravedad o fuerzas causadas por colisiones. El módulo de inclinación-aceleración combina un acelerómetro de tecnología Cmos en circuito integrado con filtros de paso bajo y amplificadores de señal. Algunas de las aplicaciones típicas de este tipo de sensores son aplicaciones robóticas, monitorización de vibración, medida de impactos y aceleraciones, medida de inclinación y orientación, estabilización de imágenes en cámaras o sistemas de diagnostico electrónicos.

El sensor de inclinación-aceleración ofrece una salida lineal. En la ecuación 5.11 se puede ver la salida que presenta el sensor en función de la inclinación. En la ecuación 5.2 podemos ver la salida que presenta el sensor en función de la aceleración.

$$V_{OUT} = V_{ZEROG} + \left(\frac{\Delta V}{\Delta G} \times G \times Sin\theta\right) \tag{3.1}$$

$$V_{OUT} = V_{ZEROG} + \left(\frac{\Delta V}{\Delta G} \times Acc\right) \tag{3.2}$$

Donde:

Figura 3.5: Sensor de inclinación 2 ejes.

V_{OUT}: La tensión de salida
V_{ZEROG}: La tensión de salida a 0g
$\frac{\Delta V}{\Delta G}$: La sensibilidad
G: La gravedad de la tierra
θ: Angulo de inclinación
Acc: Aceleración

Sensores de Proximidad

El robot dispone de 7 sensores de proximidad infrarrojos con un alcance de 81 cm. También tiene 6 sensores de ultrasonidos con un alcance de 2.85 m. Los

Figura 3.6: Sensor de distancia por ultrasonidos.

sensores de ultrasonidos son frecuentemente utilizados en robots para calcular distancias hasta los objetos cercanos a ellos. Su funcionamiento está basado en el calculo del tiempo de vuelo (TOF). Para conocer la distancia hasta el obstáculo más cercano los sensores de ultrasonidos emiten una señal a una velocidad determinada, 343m/s a temperatura 20 °C. Seguidamente miden el tiempo que transcurre desde que emiten la señal hasta que llega el eco del objeto más próximo.

Los sensores de ultrasonidos emiten la señal en forma cónica, la distancia que recibirá el sensor será la del obstáculo más cercano dentro del cono acústico y no existe especificación de que posición ocupa el objeto. Deberá tenerse en cuenta esta condición a la hora de procesar la información de los sensores de ultrasonidos. Otro problema que presenta la medición con sensores de ultrasonidos es el conocido como "cross-talking". Este efecto ocurre cuando por circunstancias del entorno, un sensor recibe la señal que ha generado otro sensor, generando una medida errónea. Las formas de evitar el cross-talking son dejar el tiempo suficiente entre sensor y sensor para que una señal no pueda interferir en otra o agrupar en el tiempo medidas de sensores que por su ubicación en el robot tienen difícil sufrir cross-talking.

Figura 3.7: Sensor de distancia por infrarrojos.

Los sensores de proximidad por infrarrojos tienen otro modo de funcionamiento con respecto a los ultrasonidos. Los sensores de proximidad por infrarrojos efectúan medidas puntuales de los elementos del entorno, envían un haz de luz infrarroja en linea recta de forma perpendicular al sensor, si el haz encuentra un obstáculo, este reflejará la luz del sensor de modo especu-

lar. Si el objeto se encuentra en el área de acción del sensor, el haz llegará al sensor. El sensor es capaz de medir con precisión la posición de retorno del haz para calcular la posición a la que se encuentra el obstáculo. En la Figura

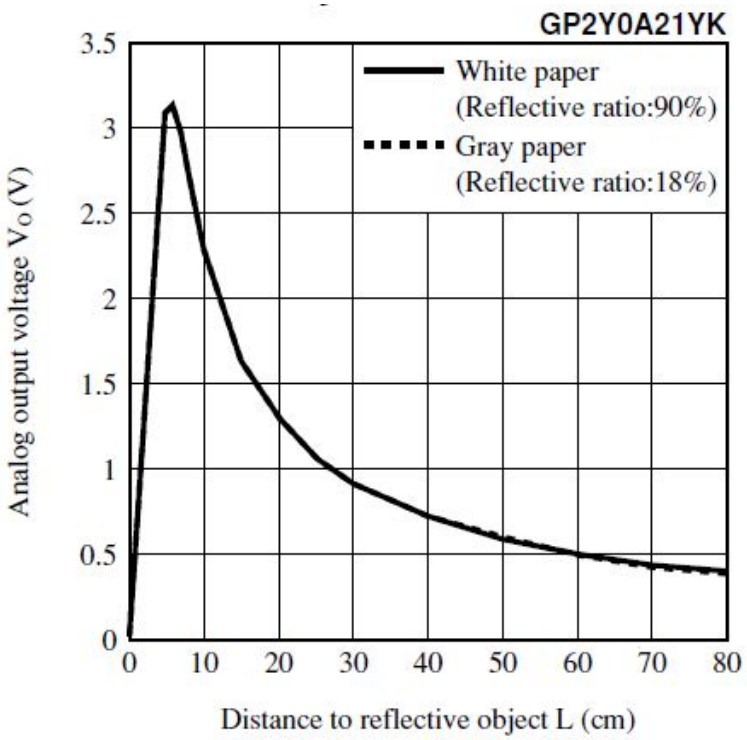

Figura 3.8: Gráfica de salida sensor infrarrojo.

3.8 podemos ver una gráfica con la salida en tensión frente a la distancia de detección en centímetros.

Sensores piroeléctricos de Presencia Humana

El robot tiene dos sensores de presencia piroeléctricos que puede utilizar para detectar la presencia de personas. Los sensores piroeléctricos de presencia humana basan su funcionamiento en captar la radiación infrarroja que emiten el cuerpo humano. La radiación infrarroja existe en el espectro electromagnético a una longitud de onda que es más larga que la longitud de onda visible. La longitud de onda del ser humano tiene una longitud de 9.4 micras.

Figura 3.9: Sensor pyroeléctrico de presencia humana.

3.1.3. Multimedia

El robot cuenta con una serie de dispositivos multimedia para la comunicación entre usuarios y robot. A continuación se explican las características y utilidades de la cámara, sistema de audio y pantalla LCD.

Cámara

La cámara disponible en el robot es una CAM3908 que permite el envío de imágenes hasta 15 fps de velocidad de transferencia y tamaño de imagen de 352 x 288 píxeles. No obstante la configuración en el robot es de velocidad de transferencia 4 fps y tamaño de imagen de 176 x 144 píxeles. Esta configuración no se puede modificar y no es suficiente para labores de reconocimiento y vigilancia. Más adelante en la Sección 2.2 se describirán las características de la cámara elegida para llevar a cabo la vigilancia y reconocimiento facial.

Audio

El robot dispone de micrófono y altavoz que le permiten enviar y recibir señal de audio continuamente. El micrófono y el altavoz se pueden utilizar en aplicaciones de videovigilancia. El audio registrado por el robot también puede ser procesado por el robot o en aplicaciones distribuidas para realizar acciones, como por ejemplo controlar el robot por voz.

Pantalla LCD

El último elemento de multimedia es la pantalla LCD, permite escribir imágenes en blanco y negro con un tamaño de 128 x 64 píxeles. La pantalla LCD puede utilizarse para realizar consultas al usuario, mostrar el estado del robot o el nivel de carga que le queda en la batería.

Figura 3.10: Display gráfico LCD.

3.2. La cámara IP inalámbrica.

Pese a que el robot dispone de una cámara integrada, esta no ofrece ni la calidad ni la velocidad de transferencia necesarios para la videovigilancia. Es por este motivo por el que se optó por incorporar otra cámara al robot. La cámara elegida debía ser un dispositivo inalámbrico, por lo que se escogió una cámara inalámbrica. Concretamente la cámara inalámbrica TV-IP672WI de la marca Trendnet dispone de conexión inalámbrica IP, puede transmitir imágenes y sonido a velocidades de transferencia de hasta 30 fps. Dispone además de dos grados de movimiento que permiten a la cámara enfocar 340º horizontal y 115 verticalmente. Puede enviar imágenes de hasta 1.280 x 800 píxeles [7][8].

Una característica representativa de la cámara es que puede registrar imágenes a partir de 0 lux. El fabricante ofrece un software de control de la cámara desde el cual es posible programar la grabación de imágenes, envío de correos electrónicos en caso de detección de movimiento o mover manualmente el cabezal de la cámara.

Figura 3.11: Cámara TRENDnet TV-IP672WI

3.3. Librería openCV

La librería openCV [1] fue desarrollada por Intel originalmente en 1999, en la actualidad se trata de una librería multiplataforma. Puede ejecutarse en diferentes sistemas operativos, está compuesta por multitud de funciones que abarcan una gran área en el proceso de visión, como son el reconocimiento de objetos, calibración de cámaras, visión estéreo y visión robótica.

OpenCV se ha utilizado para desarrollar aplicaciones de visión artificial [17], para identificar señales del trafico [18] o algoritmos de detección de objetos [19].

3.4. Librería emguCV

La librería emguCV [2] se trata de un *wrapper* multiplataforma que permite el uso de funciones de la librería openCV en lenguajes compatibles con .NET, como son C#, VB, VC++ o IronPython entre otros. La librería emguCV resulta imprescindible para el uso de openCV desde las aplicaciones desarrolladas en .NET, puesto que .NET es un lenguaje interpretado que no puede llamar directamente a funciones escritas originalmente en C++.

3.5. FaceRecPro

FaceRecPro [16] es un software desarrollado por Sergio Andrés Gutiérrez Rojas en 2012 que permite la detección y reconocimiento de múltiples caras en una imagen. Está escrito íntegramente en C#.

FaceRecPro es un software que permite agregar usuarios a una base de datos, asociando sus fotografías y sus nombres.
Podemos ver el interfaz gráfico de FaceRecPro en la Figura 3.12. La intención

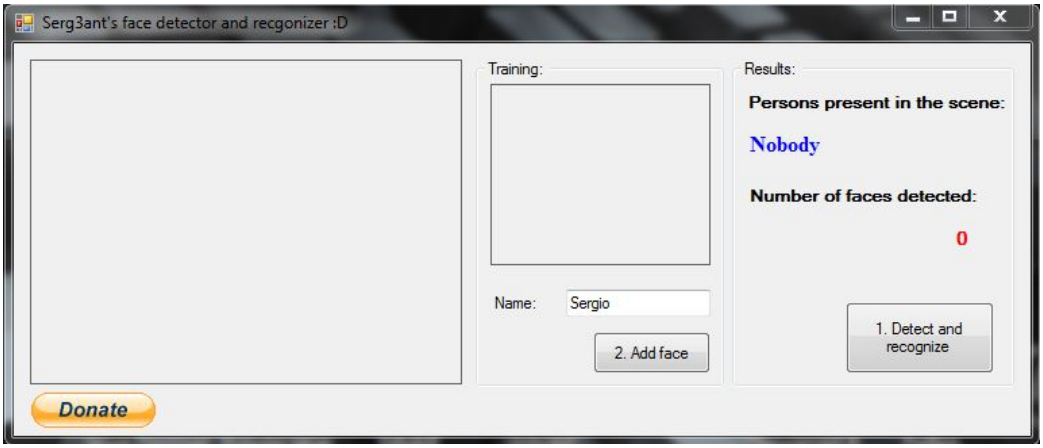

Figura 3.12: Software FaceRecPro

del desarrollador era la de realizar un software sencillo de entrenar y utilizar. A continuación se presentan los pasos que deben seguirse para entrenar el reconocedor.

1. Iniciar el software de reconocimiento.

2. Pulsar el botón "1. Detect and recognize.". Esta acción habilitará la adquisición de imágenes de la cámara web. A partir de este momento las imágenes tomadas por la cámara serán reproducidas en el cuadro de imagen de la izquierda.

3. Escribir el nombre de usuario en el cuadro de texto "Name".

4. Cuando el detector detecte la cara del usuario podremos pulsar el botón "2. Add face". Para añadir al usuario asociado a su nombre.

5. A partir de este momento el programa estará en condiciones de reconocer al usuario cuando aparezca por pantalla.

6. Se pueden añadir diferentes fotos del usuario para aumentar la probabilidad de detección y/o añadir diferentes usuarios para que el software los detecte a todos.

3.6. Instalación

3.6.1. Instalación software robot

Para la instalación del robot se debe descargar e instalar los siguientes programas.

Software	Descripción
IDE Visual Studio 2012	Se utiliza para desarrollar y compilar el programa del robot.
Software DirectX SDK	APIs para el manejo de tareas multimedia.
Microsoft .Net 3.5 Framework	Framework para aplicaciones de Windows.
DRROBOTSSentinel-CONTROL.OCX	Librería de programación del robot X80Pro.
WiRobotGateWayforWiFi.exe	Puerta de acceso entre PC y Robot.
VitaminCtrl.dll	Controlador de la cámara.

Tabla 3.1: Software necesario para programar el robot.

Para chequear el robot y comprobar que todo funciona correctamente se instalan los programas X80Pro Control y WiRobotGateway.exe, disponibles tanto en el CD de instalación como en la web del fabricante. Tambien se deben instalar directX y el Framework .Net 3.5 de microsoft, se pueden descargar fácilmente de internet. Una vez instalados se procede a ejecutar WiRobotGateway.exe, se abrirá un cuadro de dialogo donde se debe configurar el nombre del robot, por defecto Drrobot1, y en el caso de estar conectando por conexión WI-FI escribir IP: 192.168.0.201, puerto 10001 (Ver Figura 3.13. Ahora que ya se ha probado que el robot funciona correctamente se puede instalar la librería para la programación del robot DRROBOTSentinelCONTROL.OCX. A continuación se puede abrir un proyecto de X80 Pro con Microsoft Visual Studio 2012 para empezar a programar el robot.

3.6.2. Instalación Librería openCV

Para instalar la librería openCV existen dos opciones, se puede descargar los ejecutables necesarios para la versión del sistema operativo y arquitectura del microprocesador, o bien se pueden descargar los archivos y compilarlse por el usuario. El primer paso es descargar openCV, se debe elegir la última versión acorde con nuestra arquitectura y sistema operativo. A continuación

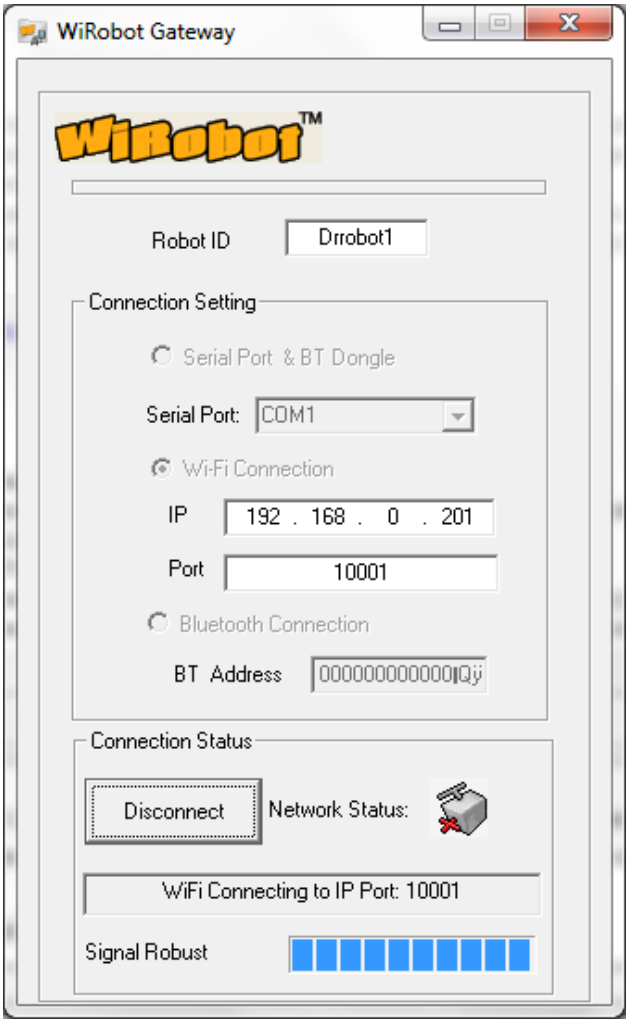

Figura 3.13: Software WiRobotGateway

se inicia la instalación, eligiendo la carpeta donde se quiere instalar la librería. Por último se deben configurar las variables del sistema con los parámetros adecuados. Como se va a utilizar el sistema operativo Windows 7, la configuración es la siguiente:

- Seleccionar Equipo en el menú inicio.
- Seleccionar Propiedades del sistema en el menú contextual.
- Hacer click en Configuración avanzada del sistema y a continuación en la ficha de Opciones avanzadas.

- Hacer clic en Variables de entorno, en Variables del sistema, buscar PATH y hacer clic en él.
- En las ventanas Editar, modificar PATH agregando la ubicación de la clase al valor de PATH. Si no dispone del elemento PATH, se puede optar por agregar una nueva variable y agregar PATH como el nombre y la ubicación de la clase como valor.
- En el campo Valor de la variable se ponen las rutas "C:\openCV; C:\openCV\build;" con la ubicación donde se encuentra openCV

3.6.3. Instalación Librería emguCV

La librería emguCV puede descargarse desde la web www.emgu.com e instalarse en el ordenador siguiendo los pasos indicados en la web.

3.6.4. Instalación Microsoft Visual Studio 2012

Para instalar Visual Studio 2012 se puede descargar el programa desde la plataforma de software de la UPV. A continuación se instala el programa siguiendo los pasos del instalador [4].

3.6.5. Instalación Cámara TrendNet

Para instalar la cámara TRENDNet utilizaremos el asistente de instalación que se encuentra en el CD de instalación de la cámara. Tras ejecutarlo se pulsa el botón *Install* y se siguen los pasos del instalador. Se utiliza la conexión WPS disponible tanto en el router como en la cámara IP. Tras finalizar con el asistente la cámara queda instalada.

Capítulo 4

Desarrollo de aplicación de vigilancia.

4.1. Diseño

El proyecto constaba de diferentes partes que convenía diferenciar para el correcto desarrollo del mismo. Por una parte el robot estaría en un entorno sin marcas especiales, por lo tanto se programaría un módulo que le permitiría posicionarse en el espacio utilizando los sensores de que dispone, incluyendo la cámara y los sensores de proximidad. El robot debería desplazarse por el entorno sin colisionar con los elementos fijos del mismo, ni tampoco con los eventuales elementos que podrían encontrarse en el entorno. Otra parte necesaria del proyecto sería la implementación del algoritmo al agente, que le permita detectar y recibir a los usuarios interesados en acceder, y acompañarlos a la entrada del área restringida en caso de reconocimiento correcto. A continuación se implementaría el módulo de reconocimiento facial, capaz de localizar en una base de datos previamente establecida, los usuarios que tienen acceso a la zona restringida. Para finalizar se implementaría una interfaz de usuario que permitiría el control del robot así como el envío de ordenes. Para conseguir el propósito de este experimento se programarían los siguientes módulos, conforme a lo que se ha descrito anteriormente: Interfaz de usuario, módulo localizador en el entorno, módulo generador de trayectorias, módulo agente de vigilancia y módulo de reconocimiento facial. En la Figura 4.1 se puede ver la relación entre módulos.

La Figura 4.2 muestra las diferentes capas de que consta la aplicación. En el lugar más alto está la aplicación X80Pro control. A un nivel inferior está la capa .NET 3.5 Framework. Un nivel por abajo está la capa del sistema operativo. El sistema operativo utiliza las funciones de las diferentes librerías.

Figura 4.1: Interacción entre módulos.

Seguidamente se detallan las características técnicas y principales funciones de cada módulo.

4.1.1. Módulo interfaz de usuario.

El módulo interfaz de usuario tiene la misión de monitorizar el comportamiento del robot, añadir usuarios a la base de datos, gestionar las acciones que el robot debe llevar a cabo en cada momento y introducir los parámetros necesarios para las labores de vigilancia (Punto de vigilancia y punto de acceso). Desde el interfaz de usuario también es posible controlar el robot de manera manual.

En el modulo interfaz de usuario se añaden los usuarios con permiso de acceso, para ello se entra en la pestaña *"Reconocimiento"* y una vez allí se escribe el nombre del nuevo usuario en el cuadro de texto, se inicia el reconocimiento de caras pulsando en el botón *"Detectar y reconocer"* y a continuación se pulsa el botón *"Añadir cara"*. Para controlar el robot hay que hacer click en la pestaña *"Vigilancia"* donde tenemos diferentes opciones, controlar el robot manualmente, iniciar una exploración del entorno, iniciar la vigilancia, ir al punto de entrada o detenerse. Seguidamente se explica como se pueden hacer las citadas acciones:

- **Mover el robot manualmente.** Existen cuatro botones para mover el robot hacia adelante, hacia atrás y girar a ambos lados. También

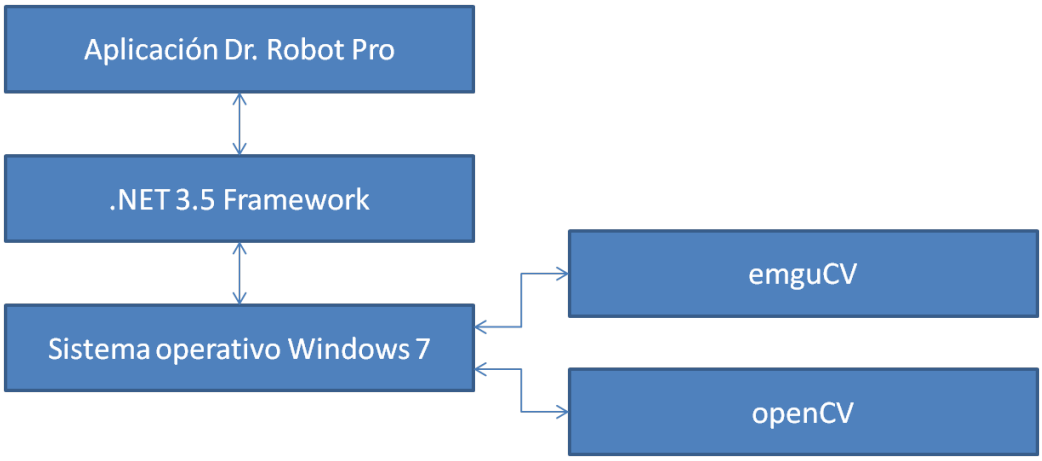

Figura 4.2: Organización capas de software.

hay un botón central para detener el movimiento. Antes de utilizar el movimiento manual, se deben detener los movimientos automáticos, para ello debemos pulsar el botón *"Parar"*

- **Exploración del entorno.** La exploración del entorno permite crear y mostrar por pantalla un mapa de ocupación, esto nos permitirá entre otras cosas introducir los parámetros de entrada al robot. Para iniciar el módulo de vigilancia bastará con pulsar el botón explorar y el robot empezará a recorrer el entorno. Es posible detener en todo momento la exploración pulsando el botón *"Parar"*.

- **Iniciar la vigilancia.** Antes de iniciar la vigilancia es necesario indicar los puntos de vigilancia y de entrada. El punto de vigilancia es donde se posicionará el robot para impedir el paso de personas no autorizadas. El punto de entrada es el punto al que el robot acompañará al usuario. Para indicar el punto de entrada se hace click en el botón *"Punto de entrada."* y a continuación en el punto del mapa donde se encuentra el punto de entrada. De igual modo se introduce en punto de vigilancia haciendo click en el botón *"Punto de vigilancia"* y pulsando en el mapa en el punto de vigilancia. Cuando ya se han introducido los puntos de entrada y vigilancia se puede iniciar la vigilancia pulsando el botón *"Controlar"*.

- **Ir al punto de entrada.** Para enviar el robot al punto de entrada se debe pulsar el botón *"Ir a entrada"*.

4.1.2. Módulo agente de vigilancia.

El agente de vigilancia es el módulo principal de la aplicación, su función es la de coordinar al resto de módulos para conseguir que la plataforma robótica se comporte como un agente de seguridad, utiliza el módulo localizador en el entorno para saber si está en el punto adecuado para recibir usuarios, detecta la llegada de los mismos y utiliza el módulo de reconocimiento facial para permitir o denegar el acceso. También acompaña a los usuarios a la entrada utilizando el generador de trayectorias. Para implementar el módulo de vigilancia se implementa una máquina de estados que gestione el comportamiento que debe tener el robot en cada momento.

El módulo agente de vigilancia se ha configurado para que se desplace al punto de vigilancia, una vez allí activa los sensores piroeléctricos de presencia humana. Cuando detecta a una persona se activa el módulo de reconocimiento. Si se reconoce una cara de la base de datos se acompaña al usuario al punto de entrada. El módulo se ha implementado con ayuda de un temporizador y una máquina de estados que determina el comportamiento del robot en cada momento.

4.1.3. Módulo localizador en el entorno.

Este módulo sirve para ubicar al robot en el entorno y para ajustar los errores de posición que el robot acumule en el desarrollo de su actividad. Para ello el robot utiliza como entrada los sensores de proximidad, la información odometrica de los codificadores rotativos de las ruedas y la cámara para determinar en que punto del entorno se encuentra.

El módulo localizador en el entorno permite al robot saber que posición ocupa en el entorno donde debe trabajar. Se ha implementado un localizador que funciona mediante tres tipos de entradas que se definen a continuación.

1. **Odometría:** La odometría se basa en sensores propioceptivos, que miden magnitudes internas al dispositivo, como las vueltas que realiza un motor. Se utiliza la odometría para conocer la posición del robot sabiendo el camino que ha recorrido previamente el robot. El módulo de odometría que se ha utilizado es el que ofrece el fabricante en el programa DrRobotX80Demo.

2. **Entradas sensores:** Las entradas de los sensores se utilizan para explorar el entorno sin colisionar también para generar un mapa de ocupación que utilizaremos para rectificar los errores acumulados debido a la odometría.

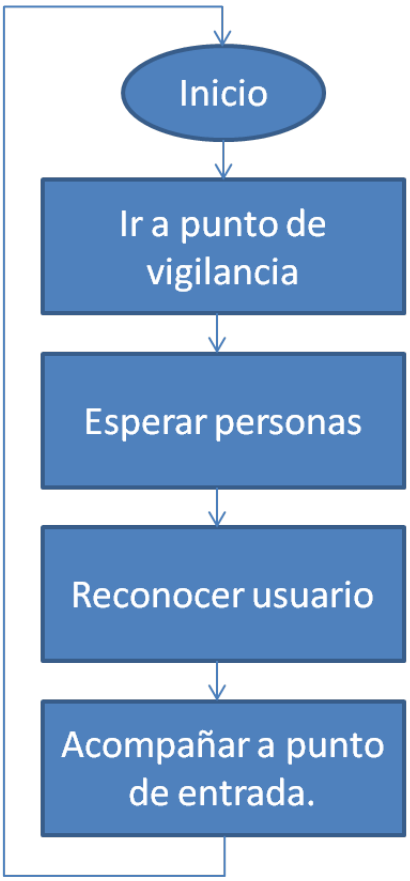

Figura 4.3: Flujograma módulo de vigilancia.

3. **Mapa de ocupación:** El mapa de ocupación es un registro de puntos en los que el robot ha detectado un objeto. Lo que se hace es guardar los puntos en los que los sensores detectan objetos, a la vez que se les asigna un grado de confianza. Cuando un objeto sea detectado en sucesivas ocasiones se irá incrementado su grado de confianza, es decir, se trata de un objeto fijo del entorno como una pared. Si el objeto no es detectado cuando el robot pasa por la zona, significará que se trataba de un objeto móvil, y se eliminará de la lista de objetos fijos. Una vez completado el mapa de ocupación del entorno, el robot será capaz de actualizar su posición en base a los datos obtenidos por los sensores y del mapa de ocupación.

En la primera ejecución del programa no existe mapa de ocupación, por lo que las primeras celdas que se ocuparán serán las que vean los sensores en su exploración del terreno. Cuando se cierra la aplicación se guarda el mapa de

ocupación en un archivo .XML que posteriormente se utilizará como experiencia previa del robot. Pasos realizados para calcular mapa de ocupación:

1. Si la medida está en el rango estimar añadir obstáculo.
2. Obtener posición y orientación del sensor.
3. Calcular posición y orientación del robot según orientación del robot.
4. Calcular posición obstáculo.
5. Si existe un obstáculo en el mapa a una distancia determinada añadir confianza al obstáculo.
6. Si no existe un obstáculo en el mapa a una distancia determinada añadir obstáculo.

4.1.4. Módulo generador de trayectorias.

El módulo generador de trayectorias es el encargado de guiar al robot para alcanzar un objetivo. Cuando se desee ir de un punto a otro del entorno, éste módulo generará la trayectoria en base a la información que conoce del entorno. Durante el desplazamiento del robot el generador de trayectorias controlará los sensores en todo momento, calculando rutas alternativas si es necesario con el fin de alcanzar el objetivo sin colisionar. En la Figura 4.4 se muestra un ejemplo de trayectoria calculada y trayectoria realizada.

Se ha utilizado el generador de trayectorias que ofrece el fabricante del robot, tanto el módulo agente de vigilancia como el operador del robot utilizan el módulo generador de trayectorias. El operador del robot lo puede utilizar si pulsa el botón *"Ir a entrada"*.

4.1.5. Módulo de reconocimiento facial.

El módulo de reconocimiento facial es el encargado de comprobar si el usuario que desea acceder está registrado en la base de datos. Para ello se

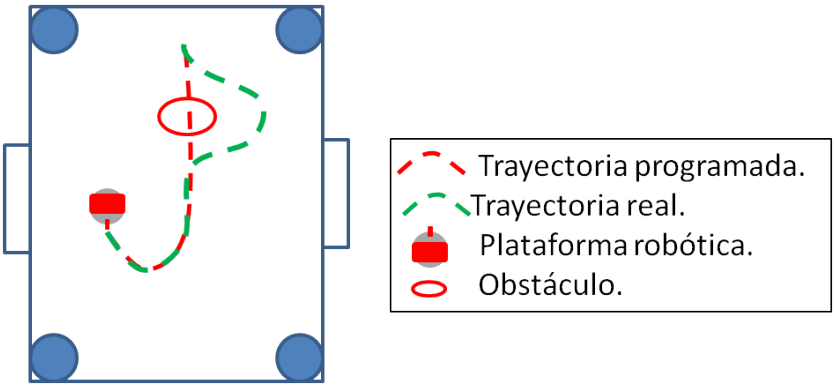

Figura 4.4: Planificación y ejecución de una trayectoria.

implementan funciones que permiten gestionar los usuarios que pueden entrar en un determinado área. Cuando la plataforma robótica se encuentra trabajando como agente permitirá el acceso cuando el usuario concuerde con uno de la base de datos.

Para el desarrollo del modulo de reconocimiento facial se ha partido del software FaceRecPro [16]. El citado código utiliza la librería emgu para utilizar las funciones de la librería openCV en el lenguaje C#. FaceRecPro compara cada cara detectada en cada uno de los fotogramas con las imágenes guardadas en la base de datos. Si alguna de las imágenes supera un determinado umbral de coincidencia, el robot acompaña al usuario a la entrada del laboratorio. Para el correcto funcionamiento del reconocimiento facial el sistema debe ser entrenado, es decir tener las suficientes imágenes en la base de datos con diferentes ángulos y niveles de iluminación.
Para añadir usuarios a la base de datos hay que hacer click en la pestaña de configuración, y a continuación introducir el nombre de usuario en el campo de texto. Seguidamente se inicia el reconocedor pulsando el botón iniciar reconocimiento y se añaden las imágenes que se deseen del nuevo usuario pulsando el botón *"Añadir imagen"*. Para cerrar el reconocedor se pulsa el botón creado a tal efecto. Cuando se pulsa el botón *"Añadir imagen"* se ejecuta el reconocimiento facial.

El proceso de reconocimiento del usuario se muestra en la Figura 4.6.

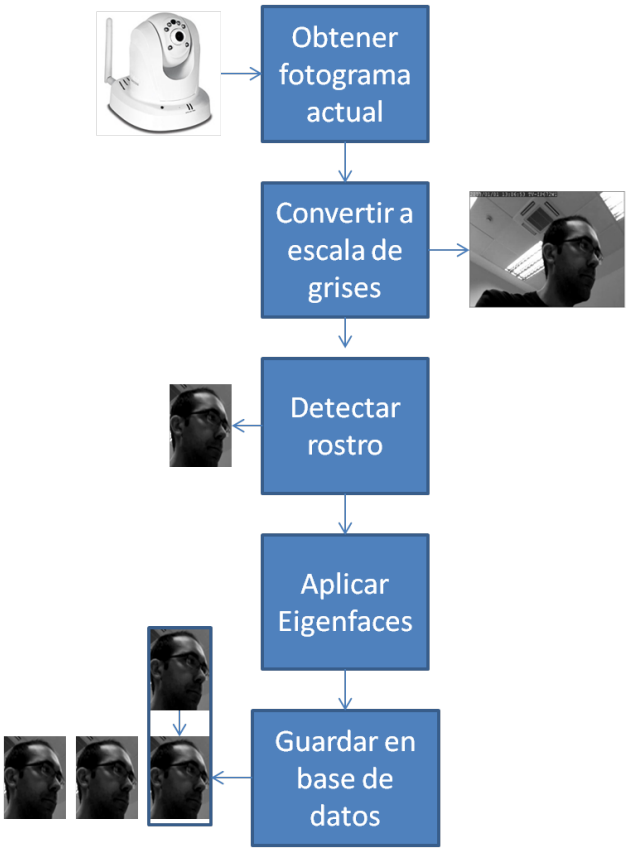

Figura 4.5: Acciones para entrenar Eigenfaces.

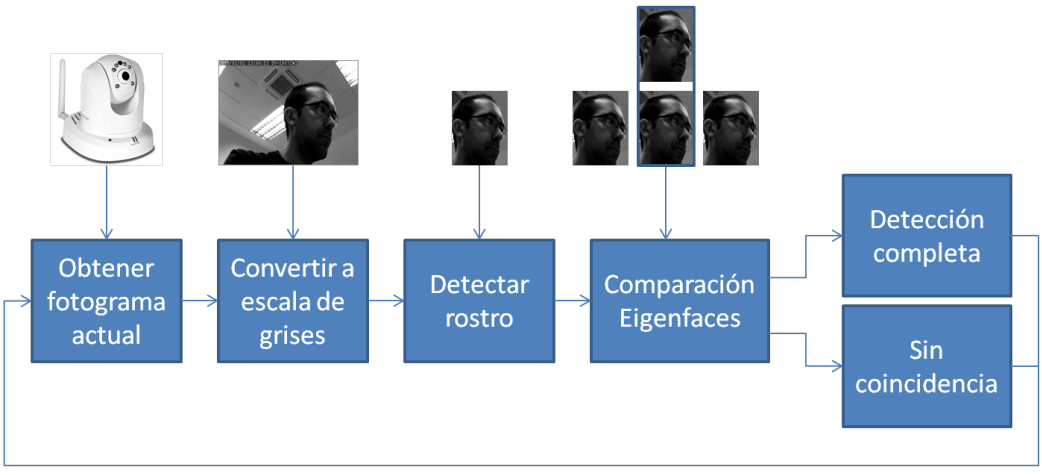

Figura 4.6: Proceso de reconocimiento.

4.2. Implementación

4.2.1. Módulo interfaz de usuario.

Se ha implementado una interfaz de usuario para que resulte intuitiva y agradable al usuario. La interfaz de usuario está programada en C# y funciona sobre el framework .NET 3.5.

La interfaz de usuario está dividida en dos partes, la parte de añadir usuarios a la base de datos, pestaña de reconocimiento (Ver Figura 4.7) y la parte de operar con el robot, pestaña de vigilancia (Ver Figura 4.8).

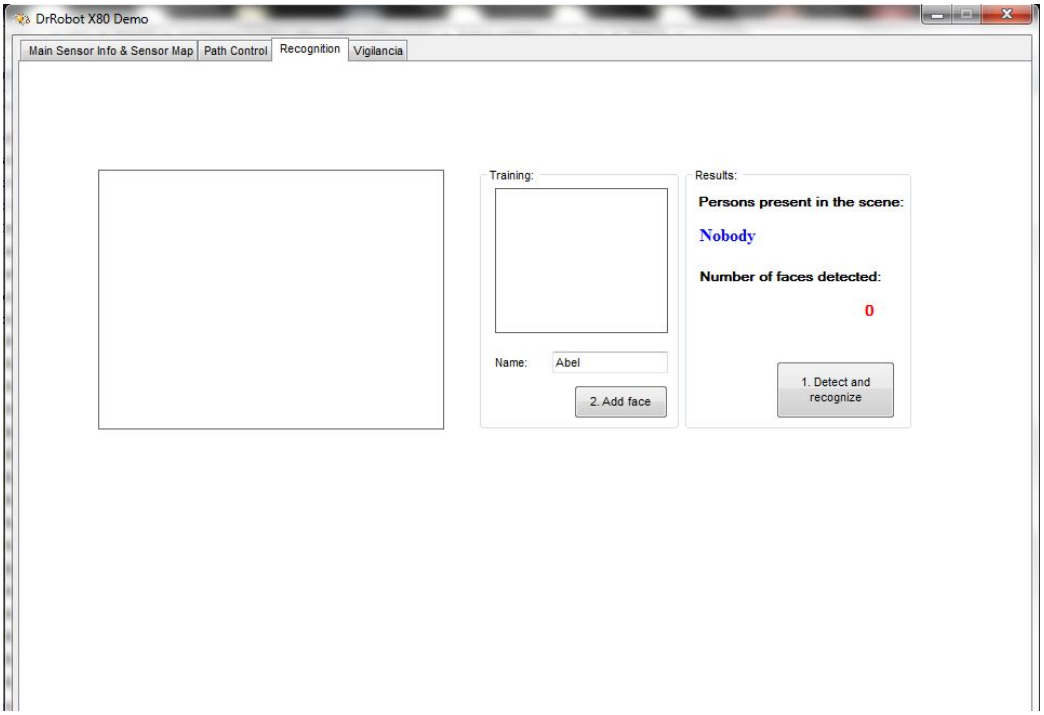

Figura 4.7: Sección añadir usuarios.

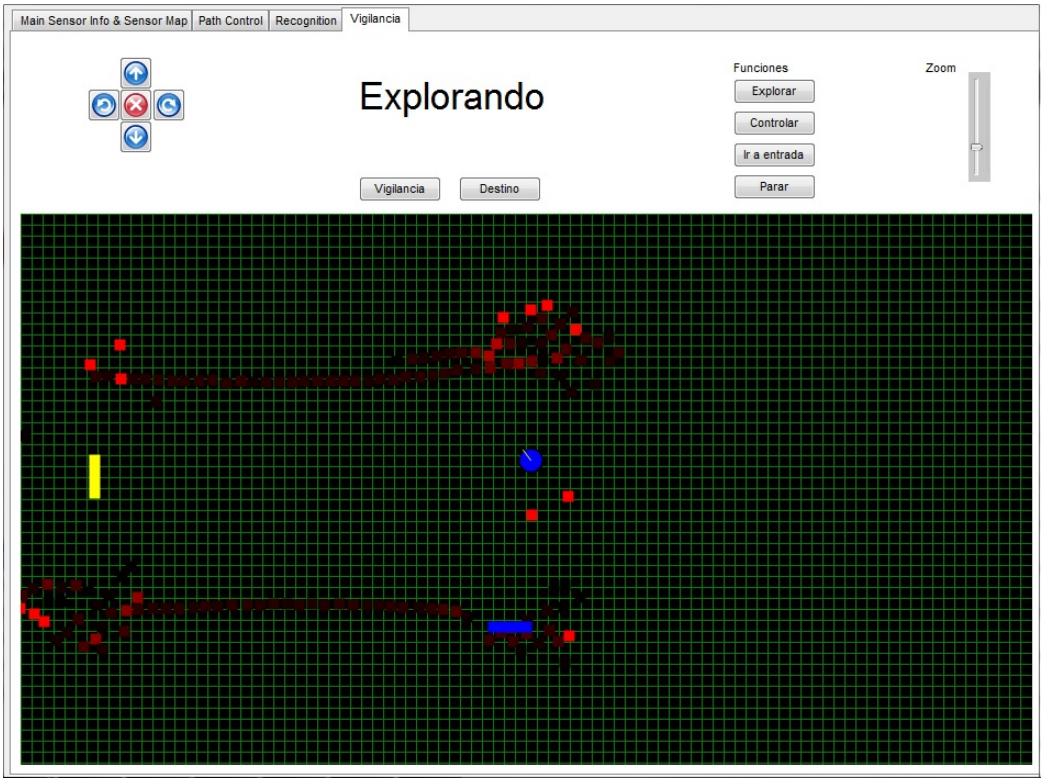

Figura 4.8: Sección controlar robot.

4.2.2. Módulo agente de vigilancia.

Para desarrollar el módulo agente de vigilancia se han añadido los objetos necesarios para su funcionamiento. Estos objetos son:

- Un botón para introducir el punto de vigilancia.
- Un botón para introducir el punto de entrada.
- Un botón para iniciar la labor de vigilancia.
- Una máquina de estados que controla al robot en las diferentes tareas de la vigilancia.

Cuando se pulsa el botón introducir punto de vigilancia, el programa queda preparado para introducir el punto. La forma más natural que se ha considerado de introducir el punto a sido a partir del mapa, donde el punto quedará colocado al hacer click. Para introducir el punto de entrada tenemos

que seguir los mismos pasos, pulsar el botón introducir punto de entrada y pulsar en el mapa en el lugar donde queremos introducir el punto de entrada.

Cuando se pulsa el botón *Controlar*, se inicializa a cero el valor de la máquina de estados y se pone en marcha el temporizador que se encarga de comprobar las condiciones de transición de la máquina de estados.

Se pueden ver las etapas de la máquina de estados en el Código 4.1.

```
switch (controlando)
{
    case 0://Ir a punto de control
      IrPuntoControl();
    break;
    case 1://De camino a punto vigilancia
      if (PuntoControl()==Alcanzado)
      {
        controlando = 2;
      }
    break;
    case 2://Buscar personas con los sensores
      if(DetectarPersonas()==True)
      {
        controlando = 3;
        IniciarReconocimientoFacial();
      {
    break;
    case 3://Buscar personas con la cámara
      if(DeteccionUsuario()==True)
      {
        controlando = 4;
      }
    break;
    case 4://Si detecta a un conocido, acompañar al punto de
   entrada
      IrPuntoEntrada();
      controlando = 5;
    break;
    case 5://Ir a puerta de entrada
      if (PuntoEntrada()==Alcanzado)
      {
         controlando = 0;
      }
    break;
}
```

Código 4.1: Máquina de estado vigilancia.

4.2.3. Módulo localizador en el entorno.

El funcionamiento del módulo localizador en el entorno funciona gracias a cuatro apartados que lo componen. Los apartados son:

- Base de datos localizador.
- Control periódico de los sensores.
- Algoritmo de adición de puntos a la base de datos.
- Búsqueda de puntos de interés de la base de datos.

Cuando se desea implementar un programa informático con memoria, para que tenga acceso a información guardada en ejecuciones anteriores, es necesario guardar la información de interés en archivos que posteriormente podrán ser leídos por el programa. Esta tarea puede resultar tediosa cuando se desea guardar un número indefinido de datos, o cada uno de los datos contiene diferentes valores.

Existe la posibilidad de guardar los puntos de interés en formato .XML. El formato XML ofrece diferentes opciones de lectura y escritura, la diferencia principal entre los diferentes formatos estriba en la eficiencia que se hace de los recursos a la hora de leer y escribir.

En referencia a la base de datos del localizador, se ha programado la lectura e incorporación a la memoria RAM de la aplicación al inicio de la aplicación. Al iniciar el programa se ejecuta el Código 4.2 para cargar los datos del archivo .XML en la memoria RAM. Por cada punto guardado en el archivo .XML, se crea un nuevo punto de tipo 'Shadow' y se guarda en una lista llamada 'Shadows'.

```
1 //Cargar sombras
  // cargamos el XML, que se parsea directamente a XDocument
3 XDocument myxml = XDocument.Load("shadows_serial.xml");
  // armamos la lista con Linq
5 shadows = (from xml in myxml.Descendants("Shadow")
  select new Shadow((double)xml.Element("PosX"),
7                   (double)xml.Element("PosY"),
                    (int)xml.Element("Confidence"),
9                    (int)xml.Element("Time"))
                          ).ToList<Shadow>();
```

Código 4.2: Inicialización base de datos.

El control periódico de los sensores se consigue gracias a un temporizador, se ha programado para que genere un evento cada 100ms. En el Código 4.3 se pueden ver las acciones generadas por el temporizador.

```
sensorData.UsDis[0] = (double)motionControl.GetSensorSonar1() /
    100;
sensorData.UsDis[1] = (double)motionControl.GetSensorSonar2() /
    100;
sensorData.UsDis[2] = (double)motionControl.GetSensorSonar3() /
    100;
sensorData.UsDis[3] = (double)motionControl.GetSensorSonar4() /
    100;
sensorData.UsDis[4] = (double)motionControl.GetSensorSonar5() /
    100;
sensorData.UsDis[5] = (double)motionControl.GetSensorSonar6() /
    100;

sensorData.IrDis[1] = AD2Dis(motionControl.GetCustomAD3());
sensorData.IrDis[2] = AD2Dis(motionControl.GetCustomAD4());
sensorData.IrDis[3] = AD2Dis(motionControl.GetCustomAD5());
sensorData.IrDis[4] = AD2Dis(motionControl.GetCustomAD6());
sensorData.IrDis[5] = AD2Dis(motionControl.GetCustomAD7());
sensorData.IrDis[6] = AD2Dis(motionControl.GetCustomAD8());
```

Código 4.3: Evento temporizador sensores.

Puesto que los recursos de memoria son limitados, y además un número muy elevado de obstáculos puede comprometer la velocidad de ejecución del programa, resulta conveniente guardar un número limitado de obstáculos. Con el fin de guardar los obstáculos que resulten de mayor interés, se guardan los obstáculos siguiendo los siguientes dos criterios.

- Se guardan los obstáculos detectados por el robot en el rango que más fiabilidad aportan los sensores. Se ha detectado experimentalmente que los sensores de ultrasonidos tienen su rango efectivo entre 0.1m y 0.8m. En cambio los sensores de infrarrojos tienen su rango efectivo entre 0.1 y 0.4m. Por tanto se pueden añadir los obstáculos detectados por el sensor de ultrasonidos que se encuentren entre 0.1 y 0.8m y los obstáculos detectados por los sensores de infrarrojos entre 0.1 y 0.4m.

- La segunda limitación a la hora de añadir un obstáculo a la lista se encuentra en una discretización de los puntos que se pueden añadir a la lista. De este modo antes de añadir un nuevo obstáculo a la lista, se hace una búsqueda para ver si existe algún obstáculo a una distancia menor a la del umbral, este umbral se introduce en función del tamaño del entorno a vigilar y de la memoria que se quiere destinar al programa.

De este modo cuando los sensores detecten un nuevo obstáculo dentro de su rango efectivo, se podrá crear un nuevo obstáculo en la lista si no existe

ninguno cercano, o podrá aumentar la confianza en puntos anteriores. El Código 4.4 muestra cómo se evalúa la incorporación de nuevos obstáculos.

```
para cada Nuevo_Obstaculo en sensores
    si(Nuevo_Obstaculo en Rango_Efectivo)
        si Nuevo_Obstaculo cerca Obstaculos_Lista
            Aumentar_Confianza
        si no
            Añadir_Nuevo_Obstaculo
```

Código 4.4: Tratamiento nuevos obstáculos.

La lectura de los obstáculos de la lista resulta de utilidad para diferentes aplicaciones, como por ejemplo comprobar si existen puntos cercanos al nuevo obstáculo detectado o para guardar los datos de los obstáculos en un archivo antes de apagar el control. El Código 4.5 guarda la lista de obstáculos antes de cerrar.

```
// armamos el XDocument
// los XElement pueden ir anidados
XDocument xdoc = new XDocument(new XElement("Shadows",
from s in shadows
    select new XElement("Shadow",
    new XElement("PosX", s.getPositionX().ToString()),
    new XElement("PosY", s.getPositionY().ToString()),
    new XElement("Confidence", s.getConfidence().ToString()),
    new XElement("Time", s.getTime().ToString()
                        )
     ));
            xdoc.Save("shadows_serial.xml");
```

Código 4.5: Tratamiento nuevos obstáculos.

4.2.4. Módulo generador de trayectorias.

El módulo generador de trayectorias que se ha utilizado es el mismo que el fabricante ha implementado en su versión DrRobotX80Demo. Lo que se ha realizado ha sido analizar cómo se podía utilizar el generador de trayectorias propio del robot para su utilización en este proyecto.

Analizando el código que suministra el fabricante se puede ver que la función se utiliza para generar las trayectorias es *SendP2PGoCmd(int index)*. Cuando se llama a la función SendP2PGoCmd, el control lee los datos de la fila *index*, que se encuentra en otra pestaña de la aplicación (Ver Figura 4.9). Lo que se ha programado para poder utilizar la función SendP2PGoCmd ha sido escribir en una fila de la tabla los valores del punto al que se quiere ir,

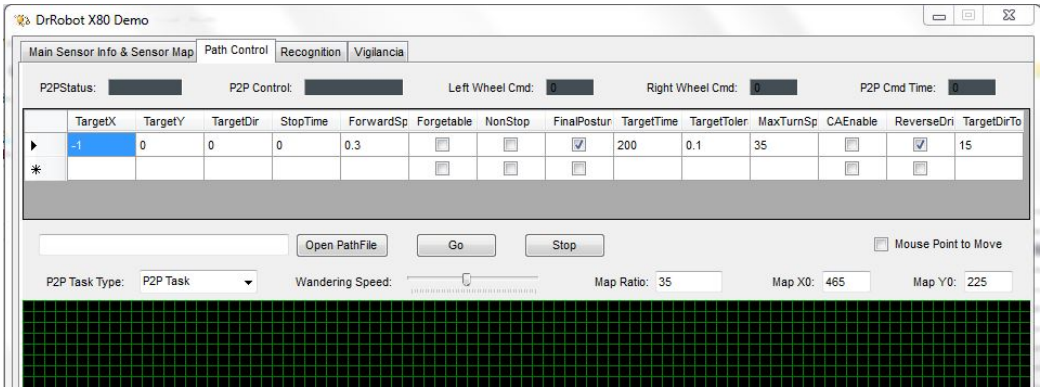

Figura 4.9: Tabla de introducción de coordenadas.

y posteriormente llamar a la función con el número de indice de la fila en la que se ha escrito.

Para escribir en la tabla se ha utilizado el Código 4.6.

```
                    //Posición X
dataGridViewPointConfig.Rows[0].Cells[0].Value = PositionX();
                    //Posición Y
dataGridViewPointConfig.Rows[0].Cells[1].Value = PositionY();
                    //Orientación
dataGridViewPointConfig.Rows[0].Cells[2].Value = Orientation();
```

Código 4.6: Introducir datos en tabla.

4.2.5. Módulo de reconocimiento facial.

Durante la programación del detector de caras se modificó la captura de imágenes de la cámara web por la captura de imágenes a través del router de la cámara inalámbrica [5][6].
La función que se utilizaba originalmente para capturar las imágenes tomadas por la cámara web se puede ver en el Código 4.7.

```
//Initialize the capture device
grabber = new Capture();
grabber.QueryFrame();
//Initialize the FrameGraber event
Application.Idle += new EventHandler(FrameGrabber);
```

Código 4.7: Función para capturar imágenes de la cámara web.

Para poder iniciar la recepción de imágenes tomadas por la cámara inalámbrica se introduce como parámetro una cadena que tiene el siguiente formato[3]:

nombre de usuario:*contraseña*@*dirección IP de la cámara*/*canal de reproducción de imágenes.*
El Código 4.8 muestra la programación que se ha utilizado para tomar las imágenes de la cámara IP.

Para poder procesar las imágenes tomadas por la cámara IP es preciso configurar la salida de vídeo. Para configurar la salida de vídeo hay que entrar en la configuración de la cámara. A la configuración de la cámara se accede desde el explorador web introduciendo la IP de la página de configuración. Para configurar la transmisión de imágenes hay que conectar la cámara y el ordenador a la misma red inalámbrica. En la Figura 4.12 se muestra el acceso a la configuración de la cámara. En la ventana emergente se introduce el usuario y la contraseña que proporciona el fabricante. Una vez dentro de la configuración de la cámara, en la sección de configuración, se pueden configurar hasta 4 salidas de vídeo y audio diferentes. Se puede ver una vista de la sección de configuración en la Figura 4.13. Para esta aplicación se ha configurado la salida número 1 con la calidad y fps adecuados para la aplicación. *play1.sdp* está vinculado con la configuración de la cámara número 1.

```
string videoStreamAddress = "rtsp://admin:admin@192.168.0.101/
    play1.sdp";
//Initialize the capture device
grabber = new Capture(videoStreamAddress);
grabber.QueryFrame();
//Initialize the FrameGraber event
Application.Idle += new EventHandler(FrameGrabber);
```

Código 4.8: Función para capturar imágenes de la cámara web.

También se detectaron problemas en la toma de imágenes a 30 fps. Pese a que la cámara es capaz de capturar y transmitir imágenes a 30 fps, el ordenador no es capaz de procesar toda esta información. Esta situación genera problemas de tipo sobrecarga de buffer de entrada, etc. Para solucionar este problema se optó por bajar la velocidad de transmisión hasta que el ordenador fuera capaz de procesar la información recibida. De este modo, fue necesario bajar la velocidad de transmisión de imágenes hasta 6 fps. En la Figura 4.10 se puede ver el tipo de problemas y en la Figura 4.11 como ha quedado la recepción de imágenes.

Figura 4.10: Recepción de imágenes a 10 FPS

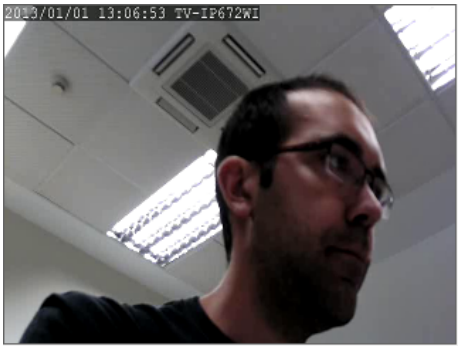

Figura 4.11: Recepción de imágenes a 6 FPS

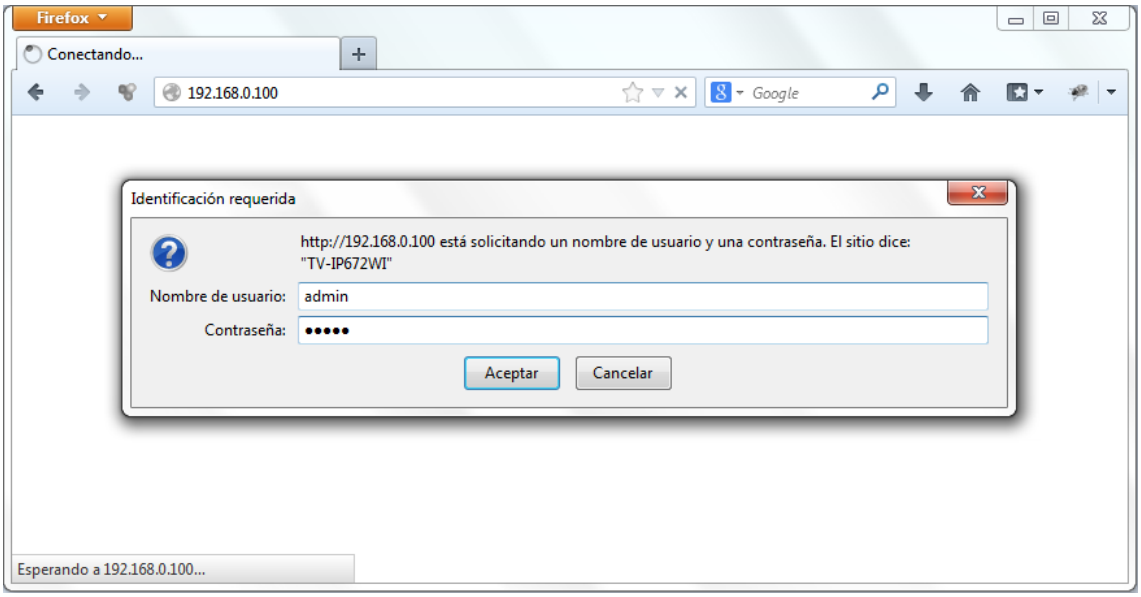

Figura 4.12: Acceso a la configuración de la cámara.

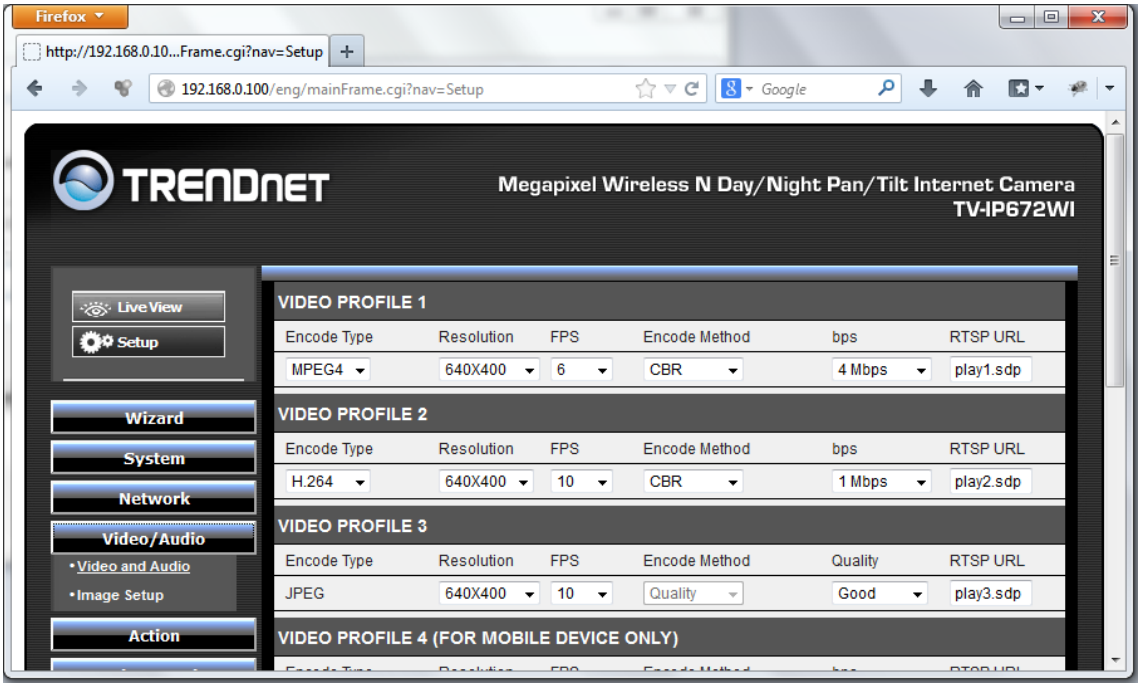

Figura 4.13: Configuración de transmisión de imágenes.

Capítulo 5

Evaluación

En las pruebas que se han realizado sobre el robot de vigilancia se han evaluado dos módulos. En primer lugar se ha analizado el comportamiento del módulo de odometría. Por otra parte se ha probado la fiabilidad del módulo de reconocimiento facial.

También se han evaluado las aplicaciones de robot de vigilancia y robot de vigilancia para aplicación en hospitales.

A continuación se explican los detalles de los diferentes experimentos.

5.1. Experimento 1: Módulo de odometría

Se ha estudiado el funcionamiento de la odometría del robot. El experimento se ha diseñado según las siguientes premisas.

- Se mandan al robot una serie de órdenes para alcanzar unos puntos previamente definidos.
- Se detiene el robot en cada uno de los puntos, se toma nota de la posición que el robot ha calculado, y se mide la ubicación real del robot con ayuda de un flexómetro.
- Se dibujan en un gráfico las tres medidas de cada punto para visualizar como se va acumulando el error.

En el gráfico de la Figura 5.1 se muestra la medida de referencia que se ha ordenado al robot en color azul, en color verde se muestra la posición en la que el robot ha calculado que está y en color rojo la posición en la que se encuentra el robot.

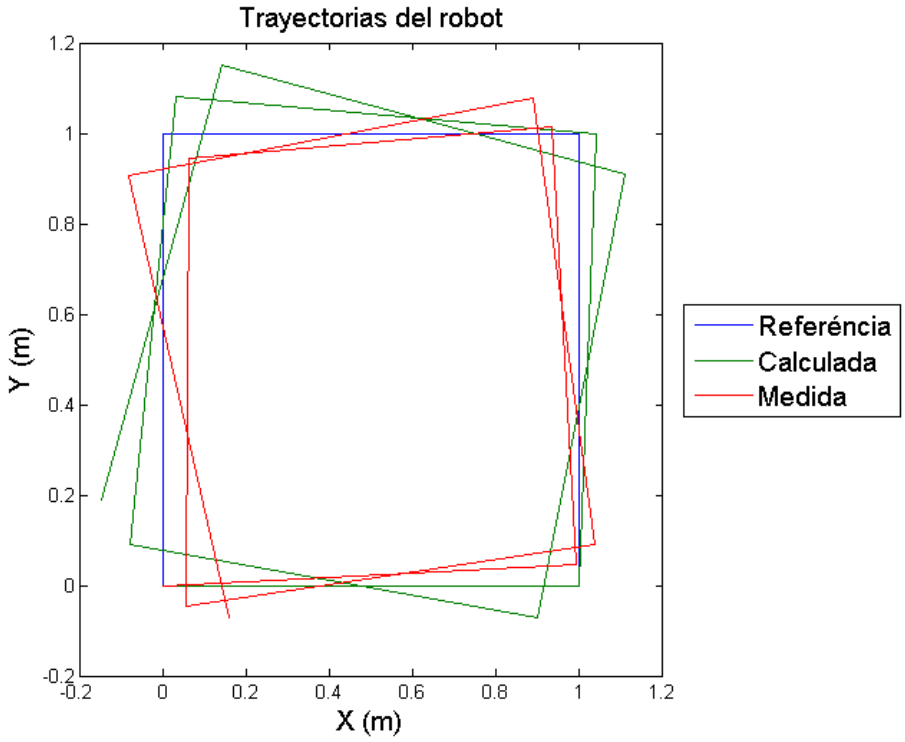

Figura 5.1: Huella del robot.

Con el fin de observar cuál es el error acumulado del robot, se ha calculado el error que existe entre la posición medida del robot y la posición calculada del robot. En la Figura 5.2 se presenta el error acumulado en el robot. Se observa un error aproximado del 5 % en el módulo de odometría.

Este error puede comprometer aplicaciones de mayor nivel, como cargar las baterías o recibir usuarios. Con el fin de reducir al máximo el error de posicionamiento nos planteamos como trabajo futuro la utilización de un filtro de Kalman.

5.2. Experimento 2: Módulo reconocimiento facial

Para evaluar el funcionamiento del módulo de reconocimiento facial se ha contabilizado el número de no-detecciones que cometía el sistema ante un

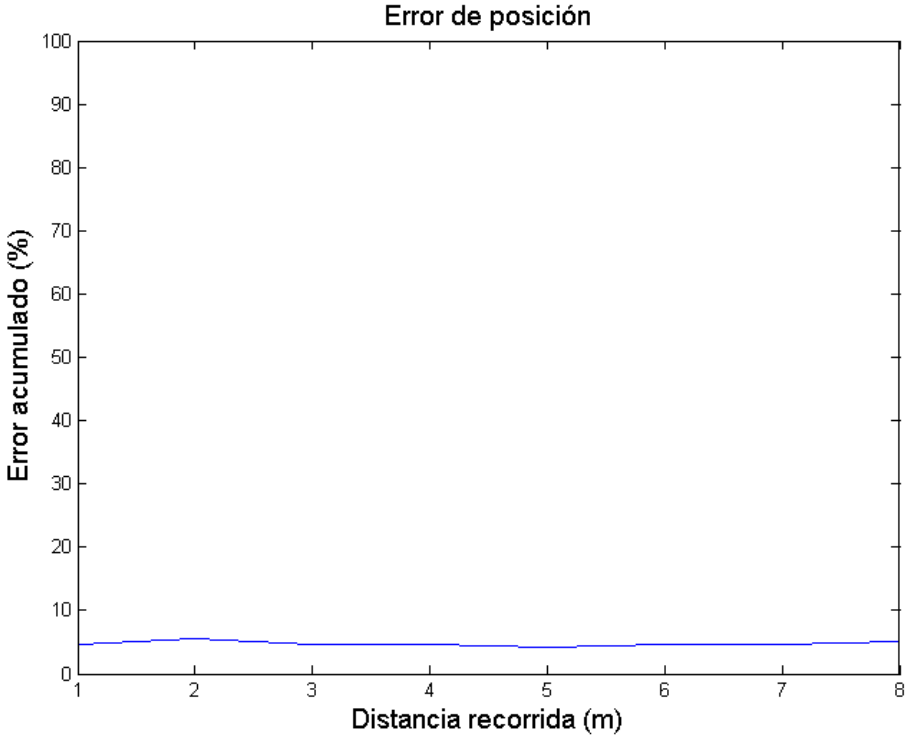

Figura 5.2: Errores acumulados en el robot.

usuario que sí estaba en la base de datos. Se ha evaluado el funcionamiento del sistema en función de las imágenes guardadas en la base de datos. Se ha diseñado el siguiente procedimiento para realizar la evaluación.

- Se añaden las fotografías del usuario a la base de datos.
- El usuario se posiciona frente al detector en diferentes posiciones y orientaciones.
- Se toma nota de las detecciones y no-detecciones.

Se pueden ver los resultados en la Figura 5.3.

A partir de los resultados presentados se descarta por el momento el uso de una única imagen para detectar a los usuarios, los resultados con dos imágenes en la base de datos son bastante mejores, pero para el desarrollo de los siguientes experimentos se tomarán 4 imágenes de cada usuario.

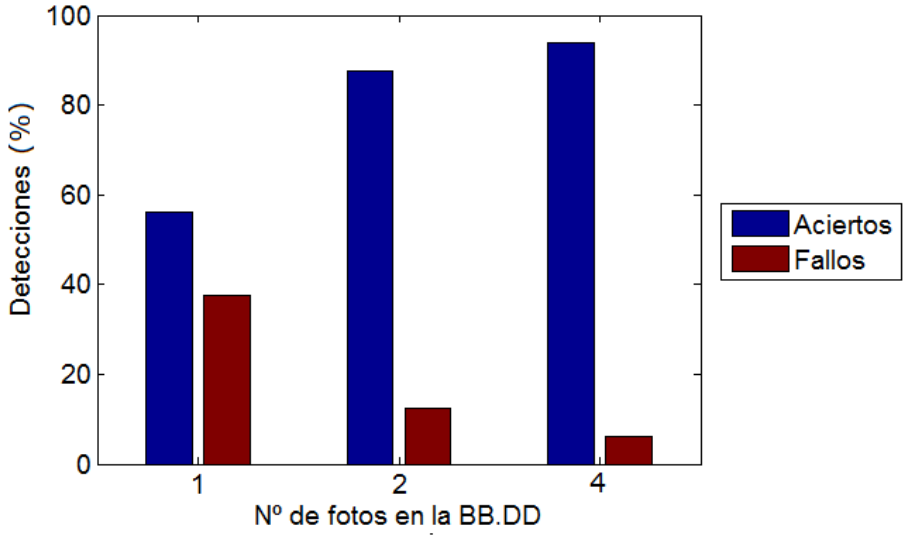

Figura 5.3: Resultado falsos negativos.

5.3. Experimento 3: Robot de vigilancia

A continuación se diseña una aplicación en la que el robot vigilante debe acompañar a usuarios del laboratorio a la entrada. Participan un total de 15 personas en este experimento. Se incluyen 4 imágenes de 10 usuarios en la base de datos. Participan también 5 personas que no tienen permiso para entrar en el departamento, y por tanto no han añadido fotografías a la base de datos.

La evaluación consiste en que el robot acompañará a los 10 usuarios que tienen permiso para entrar al laboratorio y denegará el acceso a las 5 personas que no disponen del mismo. El robot dispone de 10 segundos para detectar a los usuarios. Por último, se reiniciará la posición del robot cada dos usuarios detectados para evitar que el error acumulado impida la realización de la prueba. En la Figura 5.4 se muestran las detecciones y no-detecciones de los usuarios que ha hecho el robot. Así como las no-detecciones y detecciones falsas que ha hecho el robot de usuarios que no estaban en la base de datos. En la prueba realizada surgen errores de dos tipos. El primer error surge cuando el robot no reconoce la cara de un usuario que está en la base de datos. El segundo tipo de error ocurre cuando el robot permite el acceso al laboratorio a personas que no tienen permiso. Esto ocurre porque el control reconoce por error a otras personas que están en la base de datos.

Analizando los datos recogidos en la prueba, se verifica que el sistema

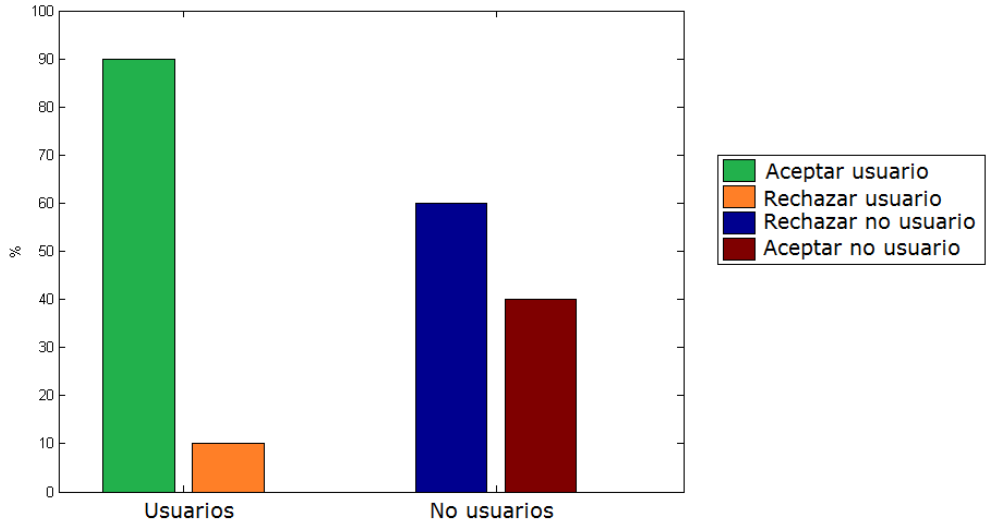

Figura 5.4: Resultado aplicación reconocimiento.

reconoce correctamente al 90 % de los usuarios del laboratorio, mientras que no reconoce al 10 % restante. En el caso de reconocimientos de personas que no tienen acceso, el sistema restringe el acceso al 60 % de las personas sin autorización, mientras que tiene un error del 40 % permitiendo el acceso a personas no usuarias.

5.4. Experimento 4: Robot de vigilancia en hospitales

En este experimento se ha probado la fiabilidad del robot en la tarea de acompañar pacientes a las consultas donde deben recibir el tratamiento. Para ello se plantea un escenario en el que hay 4 consultas donde acompañar a los pacientes, cada paciente tiene asignado una consulta y cuenta con una base de datos de 4 imagenes. La cola de pacientes que el robot debe acompañar a los respectivas consultas es de 50 pacientes. En la Figura 5.5 se muestra un esquema del escenario de la aplicación. Se controla en primer lugar para cada paciente si el robot ha sido capaz de detectarlo, en el caso de que el robot haya sido capaz de detectarlo, se controla si el robot guía correctamente al paciente la consulta adecuada. En la Figura 5.6 se muestran los resultados de la prueba con los pacientes acompañados correctamente (en azul), los pacientes acompañados a una consulta incorrecta (en verde) y los

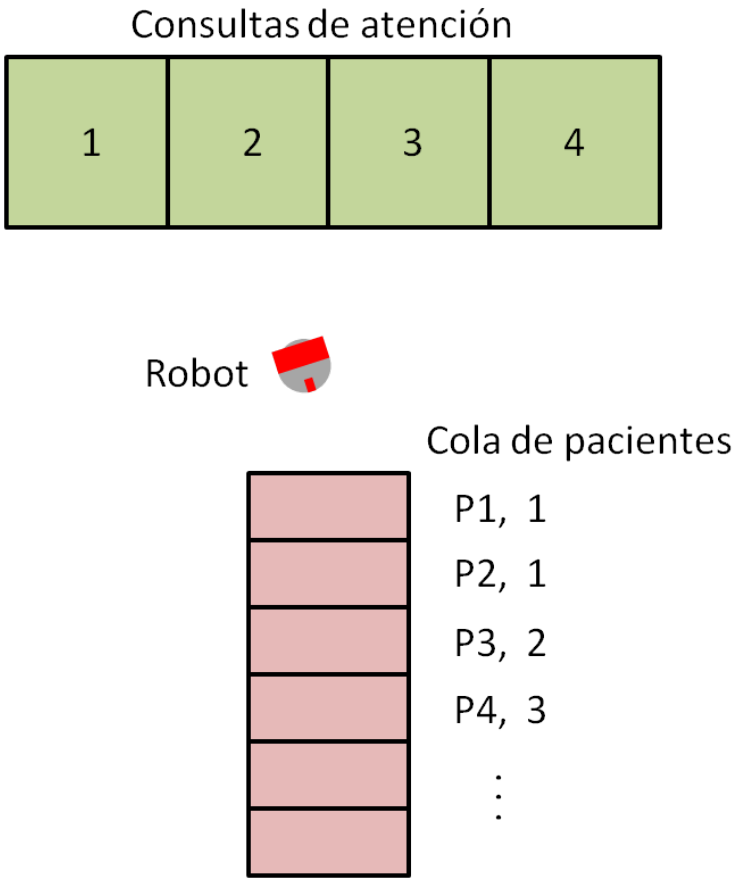

Figura 5.5: Escenario experimento 4.

pacientes no detectados (en rojo). En la gráfica se puede ver como el robot es capaz de reconocer y acompañar con éxito al 84 % de los pacientes, tiene problemas para reconocer al 10 % de los pacientes y es incapaz de completar el acompañamiento en el 6 % de las ocasiones. Analizando los resultados de

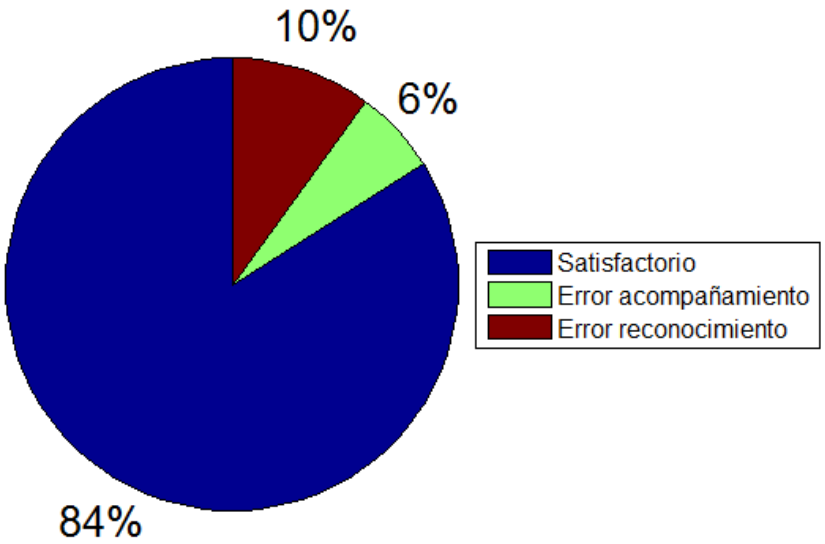

Figura 5.6: Resultado acompañamiento.

este nuevo experimento se observa que ha aumentado ligeramente el número de pacientes no detectados. Además el robot no completa con éxito el acompañamiento de otro 6 % de los pacientes. A pesar de ello el robot sigue acompañando hasta la consulta adecuada al 84 % de los pacientes. Estos son los datos obtenidos de la prueba técnica en el laboratorio. No obstante en una posterior prueba piloto podrían aparecer problemas sociales, por ejemplo personas que no desean acompañar al robot, o que lo abandonan cuando se ha iniciado el servicio.

Capítulo 6

Presupuesto

Se ha realizado un presupuesto previo a la realización del trabajo, en el que se consideran todos los recursos humanos y materiales necesarios para el desarrollo del mismo. Al finalizar el trabajo se ha realizado un recuento de recursos utilizados, así como una comparación entre los diferentes presupuestos.

6.1. Presupuesto inicial

En el presupuesto inicial se ha tenido en cuenta los costes de los recursos humanos y de los materiales utilizados.

6.1.1. Coste personal

Para el cálculo del coste personal, se ha establecido una categoría profesional para el desarrollador del trabajo de Ingeniero en prácticas. El coste medio de la hora de trabajo por los ingenieros en prácticas es de 20€/hora en España. El segundo precio de mano de obra es el del tutor y cotutor, a los que se les ha asignado una categoría de ingenieros senior. Su trabajo tiene un coste de 50€/hora. En la Tabla 6.1 se muestran los recursos humanos que se estimaron para el desarrollo del trabajo.

	Precio/hora	Nº horas	Subtotal
Ingeniero en prácticas	20	300	6,000.00 €
Ingeniero sénior	50	15	750.00 €
		Total	6,750.00 €

Tabla 6.1: Tabla recursos humanos inicial.

6.1.2. Coste software y hardware

En esta sección se presenta el presupuesto de los recursos materiales y de software. Se indica el precio del producto, así como el tiempo de amortización y el tiempo estimado de uso. En la Tabla 6.2 se muestran los recursos materiales que se estimaron para la realización del trabajo.

Software y hardware	Precio	Tiempo amortización	Uso del producto	Subtotal
Robot	2352	24	3	294.00 €
HP G61	499	24	5	103.96 €
Visual Studio	5504	18	3	917.33 €
TexnicCenter	0	6	5	0.00 €
			Total	1,315.29 €

Tabla 6.2: Tabla recursos materiales inicial.

6.1.3. Coste total del proyecto

Para calcular el presupuesto se suma el coste de mano de obra y el de hardware y software. En la Tabla 6.3 se muestra el total del presupuesto. Por

Descripción	Coste total
Personal	6,750.00 €
Software y hardware	1,315.29 €
Coste total aplicación	8,065.29 €

Tabla 6.3: Tabla presupuesto inicial.

lo tanto el presupuesto para el trabajo es de 8065.29€.

6.2. Presupuesto final

Al finalizar el trabajo se ha realizado un presupuesto para comprobar si se han mantenido los costes establecidos en el presupuesto inicial. Al igual que en el presupuesto inicial se ha dividido el coste del proyecto en coste de personal y coste de software y hardware. A continuación se explican los costes de mano de obra.

6.2.1. Coste personal

En el desarrollo del trabajo, se ha excedido las horas planificadas para el desarrollador principal. Por ello se ha incrementado el coste de mano de obra. En el punto de ingeniero senior también ha aumentado el número de horas. Ello se debe a que en parte de las reuniones estaban el tutor y el cotutor, aumentando de este modo las horas en este concepto. En la Tabla 6.4 se muestra el coste asociado a la mano de obra.

	Precio/hora	Nº horas	Subtotal
Ingeniero en prácticas	20	324	6,480.00 €
Ingeniero sénior	50	22.5	1,125.00 €
		Total	7,605.00 €

Tabla 6.4: Tabla recursos humanos final.

6.2.2. Coste software y hardware

Para el desarrollo del proyecto, se ha necesitado una cámara inalámbrica que no se había previsto inicialmente. Por ello se ha incrementado el coste. Por otra parte al prolongarse la duración del trabajo, el uso de los programas también se ha ampliado en unos meses. La Tabla 6.5 muestra los recursos software y hardware utilizado para el desarrollo del trabajo.

Software y hardware	Precio	Tiempo amortización	Uso del producto	Subtotal
Robot	2352	24	3	294.00 €
HP G61	499	24	6	124.75 €
Cámara TrendNET	153.13	24	3	19.14 €
Visual Studio	5504	18	3	917.33 €
TexnicCenter	0	6	6	0.00 €
			Total	1,355.22 €

Tabla 6.5: Tabla recursos materiales final.

6.2.3. Coste total del proyecto

Por último se muestra el coste que ha tenido el proyecto, que se calcula sumando los costes de mano de obra y los costes de software y hardware. En la Tabla 6.6 se muestra el coste que tendría que pagar el cliente. El coste

Descripción	Coste total
Personal	7,605.00 €
Software y hardware	1,355.22 €
Coste total aplicación	8,960.22 €

Tabla 6.6: Tabla presupuesto final.

total del trabajo es de 8960.22€. El coste final se ha incrementado respecto al coste estimado inicialmente. Pero para tratarse de un trabajo tecnológico en el que se han tratado conceptos tan diversos el incremento del precio es aceptable.

Capítulo 7

Conclusiones

En este proyecto se ha presentado una aplicación de robot de vigilancia. Para ello se ha implementado el comportamiento de exploración y el comportamiento de vigilancia. También se ha programado el módulo de mapa de ocupación. Así como también se ha desarrollado la interfaz de usuario, que sirve para introducir los parámetros de control del robot y controlarlo. Se han diseñado las aplicaciones de robot de vigilancia y robot de vigilancia en hospitales. Para verificar el trabajo realizado se han realizado pruebas de los módulos de odometría y reconocimiento facial. Asimismo se han evaluado las aplicaciones de robot de vigilancia y robot de vigilancia en hospitales. Se considera que los resultados han sido satisfechos.

Capítulo 8

Anexos

1. Funciones más representativas del software implementado.
2. Planificación temporal de tareas.

1. Clase shadows

```
class Shadow
{
    private double positionX = 0, positionY = 0;
    private int confidence = 0, time = 0;
    public Shadow(double positionX, double positionY, int
    confidence, int time)
    {
        this.positionX = positionX;
        this.positionY = positionY;
        this.confidence = confidence;
        this.time = time;
    }
    //lecture functions
    public double getPositionX() {return this.positionX;}
    public double getPositionY() {return this.positionY;}
    public int getConfidence() {return this.confidence; }
    public int getTime() {return this.time;}

    //write functions
    public void setPositionX(double positionX)
    {
        this.positionX = positionX;
    }
    public void setPositionY(double positionY)
    {
        this.positionY = positionY;
    }
    public void setConfidence(int confidence)
    {
        this.confidence = confidence;
    }
    public void setTime(int time)
    {
        this.time = time;
    }
}
```

Código 1: Obstáculos mapa de ocupación.

2. Clase sensor

```
class Sensor
{
    private double positionX = 0, positionY = 0;
    private int sensorAngle;
    public Sensor(int numSensor, double distance, int type, int
   time)
    {
        if (type == 0)//type==0; IR sensor
        {
            switch (numSensor)
            {
                case 1: sensorAngle = 135; break;
                case 2: sensorAngle = 105; break;
                case 3: sensorAngle = 075; break;
                case 4: sensorAngle = 045; break;
                case 5: sensorAngle = 000; break;
                case 6: sensorAngle = 270; break;
                case 7: sensorAngle = 180; break;
            }
        }
        else if (type == 1)//type==1; sonar sensor
        {
            switch (numSensor)
            {
                case 1: sensorAngle = 150; break;
                case 2: sensorAngle = 090; break;
                case 3: sensorAngle = 030; break;
                case 4: sensorAngle = 330; break;
                case 5: sensorAngle = 270; break;
                case 6: sensorAngle = 210; break;
            }
        }
        this.positionX = distance * Math.Cos(sensorAngle * (2.0
   * Math.PI) / 360);
        this.positionY = distance * Math.Sin((sensorAngle + 180)
    * (2.0 * Math.PI) / 360);
    }
    //lecture functions
    public double getPositionX() { return this.positionX;}
    public double getPositionY() { return this.positionY;}
}
```

Código 2: Leer información sensores.

3. Clase Punto

```
class Punto
{
    private double positionX = 0, positionY = 0, orientation =
    0;
    private int enabled = 0;
    public Punto(double positionX, double positionY, double
    orientation, int enabled)
    {
        this.positionX = positionX;
        this.positionY = positionY;
        this.orientation = orientation;
        this.enabled = enabled;
    }
    //lecture functions
    public double getPositionX() {return this.positionX;}
    public double getPositionY() {return this.positionY;}
    public double getOrientation() {return this.orientation;}
    public int getEnabled() {return this.enabled;}
    //write functions
    public void setPositionX(double positionX)
    {
        this.positionX = positionX;
    }
    public void setPositionY(double positionY)
    {
        this.positionY = positionY;
    }
    public void setEnabled(int enabled)
    {
        this.enabled = enabled;
    }
}
```

Código 3: Guardar punto de entrada y punto de vigilancia.

4. Inicialización

```
public partial class DrRobotX80DemoForm : Form
{
    //Draw shadows
    //
    private List<Shadow> shadows = new List<Shadow>();
    //
    //Configurar vigilancia
    int setVigilancia = 0, setEntrada = 0;
    Punto puntoEntrada = new Punto(0, 0, 0, 0);
    Punto puntoVigilancia = new Punto(0, 0, 0, 0);

    //Variable para bucle de control
    int controlando = 3;
    int explorando = 0;

    #endregion
    #region face variables define
    //Declararation of all variables, vectors and haarcascades
    Image<Bgr, Byte> currentFrame;
    Capture grabber;
    HaarCascade face;
    MCvFont font = new MCvFont(FONT.CV_FONT_HERSHEY_TRIPLEX, 0.5
    d, 0.5d);
    Image<Gray, byte> result, TrainedFace = null;
    Image<Gray, byte> gray = null;
    List<Image<Gray, byte>> trainingImages = new List<Image<Gray
    , byte>>();
    List<string> labels = new List<string>();
    List<string> NamePersons = new List<string>();
    int ContTrain, NumLabels, t;
    string name, names = null;

    #endregion
    #region form event

    public DrRobotX80DemoForm()
    {
        InitializeComponent();

        //Face recognition
        //Load haarcascades for face detection
        face = new HaarCascade("haarcascade_frontalface_default.
    xml");
        //eye = new HaarCascade("haarcascade_eye.xml");
        try
        {
```

```
44              //Load of previus trainned faces and labels for each
     image
                string Labelsinfo = File.ReadAllText(Application.
    StartupPath + "/TrainedFaces/TrainedLabels.txt");
46              string[] Labels = Labelsinfo.Split('%');
                NumLabels = Convert.ToInt16(Labels[0]);
48              ContTrain = NumLabels;
                string LoadFaces;
50
                for (int tf = 1; tf < NumLabels + 1; tf++)
52              {
                    LoadFaces = "face" + tf + ".bmp";
54                  trainingImages.Add(new Image<Gray, byte>(
    Application.StartupPath + "/TrainedFaces/" + LoadFaces));
                    labels.Add(Labels[tf]);
56              }
            }
58          catch (Exception e)
            {
60              //MessageBox.Show(e.ToString());
                MessageBox.Show("Nothing in binary database, please
    add at least a face(Simply train the prototype with the Add
    Face Button).", "Triained faces load", MessageBoxButtons.OK,
    MessageBoxIcon.Exclamation);
62          }

64          //End Face recognition
            //set robot information here
66          robotInfo.MotorDir = 1;      //I90 series = -1, X80 and
    Sputnik = 1
            robotInfo.OneCircleCount = 1200; //I90 series = 800, X80
     and Sputnik = 1200
68          robotInfo.WheelRadius = 0.134;      //unit :m prev 0.0825
            robotInfo.WheelDis = 0.26;  ////I90 series = 0.30, X80
    and Sputnik = 0.26
70          p2pDrive.SetRobotInfo(robotInfo);

72          robotPosition.robotX = 0;
            robotPosition.robotY = 0;
74          robotPosition.robotDir = 0;

76          robotMapPosition.robotX = 0;
            robotMapPosition.robotY = 0;
78          robotMapPosition.robotDir = 0;

80          //read Ir and US sensor config file
            string directory = System.Environment.CurrentDirectory;
82          try
            {
```

```
84              irSensorConfigSet.ReadXml(directory + "\\
    IrSensorConfig.xml");
        }
86      catch
        {
88          MessageBox.Show("Unable to find IR Ranger Sensor
    configure file");

90      }
        // irSensorConfigSet.ReadXml("c:\\IrSensorConfig.xml");
92      irNum = irSensorConfigSet.DataTableIrSensor.Count;
        sensorMapConfig.IrNum = irNum;
94      if (irNum <= 20)  //must less than 20 Ir sensor
        {
96          for (int i = 0; i < irNum; i++)
            {
98              IrSensorConfigSet.DataTableIrSensorRow row = (
    IrSensorConfigSet.DataTableIrSensorRow)irSensorConfigSet.
    DataTableIrSensor.Rows[i];
                sensorMapConfig.IrConfigData[i].OffsetX = double
    .Parse(row.OffsetX);
100             sensorMapConfig.IrConfigData[i].OffsetY = double
    .Parse(row.OffsetY);
                sensorMapConfig.IrConfigData[i].Angle = double.
    Parse(row.Angle);
102             sensorMapConfig.IrConfigData[i].Weight = double.
    Parse(row.Weight);
                sensorMapConfig.IrConfigData[i].DisTag = int.
    Parse(row.DisTag);
104         }
        }
106     //Ultrasonic config file
        try
108     {
            usSensorConfigSet.ReadXml(directory + "\\
    UsSensorConfig.xml");
110     }
        catch
112     {
            MessageBox.Show("Unable to find Ultrasonic Sensor
    configure file");
114     }
        //  usSensorConfigSet.ReadXml("c:\\UsSensorConfig.xml");
116     usNum = usSensorConfigSet.DataTableUsSensor.Count;
        sensorMapConfig.UsNUm = usNum;
118
        if (usNum <= 6)             //not more than 6 Us sensor
120     {
            for (int i = 0; i < usNum; i++)
```

```
122             {
                    UsSensorConfigSet.DataTableUsSensorRow row = (
    UsSensorConfigSet.DataTableUsSensorRow)usSensorConfigSet.
    DataTableUsSensor.Rows[i];
124                 sensorMapConfig.UsConfigData[i].OffsetX = double
    .Parse(row.OffsetX);
                    sensorMapConfig.UsConfigData[i].OffsetY = double
    .Parse(row.OffsetY);
126                 sensorMapConfig.UsConfigData[i].Angle = double.
    Parse(row.Angle);
                    sensorMapConfig.UsConfigData[i].Weight = double.
    Parse(row.Weight);
128                 sensorMapConfig.UsConfigData[i].DisTag = int.
    Parse(row.DisTag);
            }
130     }
 }
132  private void DrRobotX80DemoForm_Load(object sender,
    EventArgs e)
 {
134     //connect to activeX control
        motionControl.connectRobot("drrobot1");
136     tmrImage.Enabled = true;

138     //register service call back function
        p2pDrive.RegisterP2PDriveCmdCallback(UpdateP2PCmd);
140     sensorMapBuilder.RegisterSensorMapCallback(
    UpdateSensorMap);

142     //config sensor
        sensorMapBuilder.ConfigSensor(sensorMapConfig);
144     //must preset positon for sensormapbuilder
        sensorMapBuilder.PreSetPosition(robotMapPosition);
146     //sart send sensor information to sensor map builder
    service
        tmrSensorMapBuilder.Enabled = true;
148
        //init pointset here
150     //PointSet pointSet = new PointSet();
        PointSet.PointConfigTableRow row = (PointSet.
    PointConfigTableRow)pointSet.PointConfigTable.NewRow();
152     //set target at the 1m away with the joystick direction
        row.TargetX = -1;
154     row.TargetY = 0;
        row.TargetDir = 0;
156
        row.ForwardSpeed = 0.3;
158     row.CAEnable = false;
        row.FinalPosture = true;
```

```
160         row.Forgetable = false;
            row.MaxTurnSpeed = 35;        //35 degree
162         row.NonStop = false;

164         row.ReverseDrive = true;

166         row.StopTime = 0;
            row.TargetTime = 200;
168         row.TargetTolerance = 0.1;
            row.TargetDirTolerance = 15;          //15 degree
170
            pointSet.PointConfigTable.AddPointConfigTableRow(row);
172         dataGridViewPointConfig.DataSource = pointSet.
    PointConfigTable;
            int columnCnt = dataGridViewPointConfig.ColumnCount;
174         for (int i = 0; i < columnCnt; i++)
            {
176             dataGridViewPointConfig.Columns[i].Width = (
    dataGridViewPointConfig.Width - dataGridViewPointConfig.
    RowHeadersWidth) / columnCnt;
            }
178
            txtMapRatio.Text = MapRatio.ToString();
180         MapX0 = pictureBoxMap.Width / 2;
            MapY0 = pictureBoxMap.Height / 2;
182         txtMapX0.Text = MapX0.ToString();
            txtMapY0.Text = MapY0.ToString();
184
            drawMap();
186         drawRobot();
            drawPath();
188


190         //to update sensor map and robot position to p2p drive
    service
            tmrP2PUpdate.Enabled = true;
192


194         //set servo config
            string directory = System.Environment.CurrentDirectory;
196         _servoConfig.ReadXml(directory + "\\servoconfig.xml");

198         int cnt = _servoConfig.ServoConfigDataTable.Count;
            if (cnt != ServoNum)
200         {
                return;
202         }
            else
204         {
```

```
            for (int j = 0; j < ServoNum; j++)
206         {
                DataSetServoConfig.ServoConfigDataTableRow
    rowHead = (DataSetServoConfig.ServoConfigDataTableRow)
    _servoConfig.ServoConfigDataTable.Rows[j];
208
                servoConfigSet[j] = new ServoConfigSet();
210             servoConfigSet[j].Ini = rowHead.Ini;
                servoConfigSet[j].Min = rowHead.Min - 1;
212             servoConfigSet[j].Max = rowHead.Max + 1;

214         }
            trackBarHeadPan.Maximum = servoConfigSet[1].Max;
216         trackBarHeadPan.Minimum = servoConfigSet[1].Min;
            trackBarHeadPan.Value = servoConfigSet[1].Ini;
218
            trackBarHeadTilt.Maximum = servoConfigSet[0].Max;
220         trackBarHeadTilt.Minimum = servoConfigSet[0].Min;
            trackBarHeadTilt.Value = servoConfigSet[0].Ini;
222
            motionControl.ServoTimeCtrAll((short)servoConfigSet
    [0].Ini, (short)servoConfigSet[1].Ini, NOCONTROL, NOCONTROL,
    NOCONTROL, NOCONTROL, 1000);
224     }
        //Cargar sombras
226     // cargamos el XML, que se parsea directamente a
    XDocument
        XDocument myxml = XDocument.Load("shadows_serial.xml");
228
        // armamos la lista con Linq
230     shadows = (from xml in myxml.Descendants("Shadow")
                   select new Shadow((double)xml.Element("PosX")
    , (double)xml.Element("PosY"), (int)xml.Element("Confidence")
    , (int)xml.Element("Time"))
232                                    ).ToList<Shadow>();
    }
234 }
```

Código 4: Inicialización de valores.

5. Aplicación explorar

```
private void timer_Explorar_Tick(object sender, EventArgs e)
{
    switch (explorando)
    {
        //Comprobar si está cerca de una pared
        case 0:
            double distancia_minima = 0.5;//1m
            for (int i = 0; i < 7; i++)
            {
                if (sensorData.IrDis[i] < 0.5)
                {
                    if (sensorData.IrDis[i] < distancia_minima)
                    {
                        distancia_minima = sensorData.IrDis[i];
                    }
                }
            }
            for (int i = 0; i < 6; i++)
            {
                if (sensorData.UsDis[i] < 0.5)
                {
                    if (sensorData.UsDis[i] < distancia_minima)
                    {
                        distancia_minima = sensorData.UsDis[i];
                    }
                }
            }
            if (distancia_minima == 0.5)
            {
                explorando = 1;//Buscar pared en linea recta
            }
            else if (distancia_minima < 0.5)
            {
                explorando = 2;//Posicionar al robot paralelo a
    la pared
            }
            break;
        case 1://Avanzar hasta encontrar pared
            double speed = 800.0;
            double dist_min = 0.5;
            if ((sensorData.UsDis[0] < 1.0) || (sensorData.UsDis
    [1] < 1.0) || (sensorData.UsDis[2] < 1.0))
            {
                //Calcular la distancia mínima entre los tres
    sensores
                dist_min = sensorData.UsDis[0];
```

```
44              if (sensorData.UsDis[1] < dist_min) dist_min =
    sensorData.UsDis[1];
                if (sensorData.UsDis[2] < dist_min) dist_min =
    sensorData.UsDis[2];
46          }
            speed = 300.0 + 500.0 * dist_min;//Frenar cuando se
    acerque a una pared
48          if (dist_min < 0.3)
            {
50              explorando = 2;//Posicionar al robot paralelo a
    la pared
            }
52          motionControl.DcMotorVelocityNonTimeCtrAll((short)(-
    speed), (short)(speed), NOCONTROL, NOCONTROL, NOCONTROL,
    NOCONTROL);

54          break;
        case 2://Posicionar al robot paralelo a la pared
56
                //Comprobar si el robot está paralelo a la pared
58              if ((sensorData.IrDis[4] > 0.15) && (sensorData.
    IrDis[4] < 0.35))
                {
60
                    explorando = 3;//Seguir pared
62              }
            //Girar lento hasta encontrar un mínimo en el sensor
     infrarrojo Ir X (X infrarrojo derecho)
64          motionControl.DcMotorVelocityNonTimeCtrAll(300, 300,
    NOCONTROL, NOCONTROL, NOCONTROL, NOCONTROL);
            break;
66      case 3://Seguir pared

68          speed = 400.0;
            double distancia;
70          double G45 = 45 * Math.PI / 180;
            double frontal = 1.0;
72          double medida_sensor = sensorData.IrDis[4];
            //Control proporcional k = 1000
74          if ((sensorData.UsDis[2] * Math.Sin(G45) < 0.4) && (
    medida_sensor < 0.5))
            {
76              distancia = ((sensorData.UsDis[2] * Math.Sin(G45
    )) + medida_sensor) / 2.0;
            }else if((sensorData.UsDis[2] * Math.Sin(G45) < 0.4)
     && (sensorData.UsDis[3] * Math.Sin(G45) < 0.4))
78          {
                distancia = ((sensorData.UsDis[2] * Math.Sin(G45
    )) + (sensorData.UsDis[3] * Math.Sin(G45))) / 2.0;
```

```
80              }
                else if ((sensorData.UsDis[3] * Math.Sin(G45) < 0.4)
    && (medida_sensor < 0.5))
82              {
                    distancia = 1.3*((sensorData.UsDis[3] * Math.Sin
    (G45)) + medida_sensor) / 2.0;
84              }
                else if (sensorData.UsDis[2] * Math.Sin(G45) < 0.4)
86              {
                    distancia = 1.5*(sensorData.UsDis[2] * Math.Sin(
    G45));
88              }
                else
90              {
                    distancia = 0.2;
92                  //explorando = 2;//Buscar pared girando
                }
94
                if ((sensorData.UsDis[0] < 0.5) || (sensorData.UsDis
    [1] < 0.5) || (sensorData.IrDis[2] < 0.5) || (sensorData.
    IrDis[3] < 0.5) || (sensorData.IrDis[1] < 0.5) || (sensorData
    .IrDis[0] < 0.5))
96              {
                    //explorando = 2;
98                  medida_sensor = sensorData.UsDis[0];
                    if (sensorData.UsDis[1] < medida_sensor)
    medida_sensor = sensorData.UsDis[0];
100                 if (sensorData.IrDis[0] < medida_sensor)
    medida_sensor = sensorData.IrDis[0];
                    if (sensorData.IrDis[1] < medida_sensor)
    medida_sensor = sensorData.IrDis[1];
102                 if (sensorData.IrDis[2] < medida_sensor)
    medida_sensor = sensorData.IrDis[2];
                    if (sensorData.IrDis[3] < medida_sensor)
    medida_sensor = sensorData.IrDis[3];
104                 frontal = (0.5-medida_sensor) * 2000;
                    speed = speed * medida_sensor * 2;
106             }
                double diferencial = 2000 * (0.2 - distancia);
108             motionControl.DcMotorVelocityNonTimeCtrAll((short)(-
    speed + diferencial + frontal), (short)(speed + diferencial +
     frontal), NOCONTROL, NOCONTROL, NOCONTROL, NOCONTROL);
                break;
110         }
}
```

Código 5: Temporizador tarea explorar.

6. Introducir punto vigilancia/entrada

```
1  private void pictureBoxShadows_MouseClick(object sender,
       MouseEventArgs e)
   {
3      //set the point by mouse
       if (setVigilancia == 1)
5      {
           double x = (double)(MapY0 - e.Y) / VRes * 0.0254 *
       MapRatio;
7          double y = (double)(MapX0 - e.X) / HRes * 0.0254 *
       MapRatio;

9          puntoVigilancia = new Punto(x, y, Math.PI/2, 1);
           setVigilancia = 0;
11     }
       if (setEntrada == 1)
13     {
           double x = (double)(MapY0 - e.Y) / VRes * 0.0254 *
       MapRatio;
15         double y = (double)(MapX0 - e.X) / HRes * 0.0254 *
       MapRatio;

17         puntoEntrada = new Punto(x, y, 0, 1);


19

           setEntrada = 0;
21     }
   }
```

Código 6: Introducir parámetros vigilancia.

7. Aplicación vigilar

```
private void timer_Controlar_Tick(object sender, EventArgs e)
{
  switch (controlando)
  {
    case 0://Ir a punto de control
        //Posición X
        dataGridViewPointConfig.Rows[0].Cells[0].Value =
  puntoVigilancia.getPositionX();
        //Posición Y
        dataGridViewPointConfig.Rows[0].Cells[1].Value =
  puntoVigilancia.getPositionY();
        //Orientación
        dataGridViewPointConfig.Rows[0].Cells[2].Value =
  puntoVigilancia.getOrientation() / Math.PI * 180.0; //A
  grados
        //Tiempo de transito
        dataGridViewPointConfig.Rows[0].Cells[3].Value = Math.
  Sqrt((puntoVigilancia.getPositionX() - robotPosition.robotX)
  * (puntoVigilancia.getPositionX() - robotPosition.robotX) + (
  puntoVigilancia.getPositionY() - robotPosition.robotY) * (
  puntoVigilancia.getPositionY() - robotPosition.robotY)) /
  0.1;
        //Velocidad avance
        dataGridViewPointConfig.Rows[0].Cells[4].Value = 0.3;
        //Punto olvidable
        dataGridViewPointConfig.Rows[0].Cells[5].Value = false;
        //Sin parada
        dataGridViewPointConfig.Rows[0].Cells[6].Value = false;
        //Postura final
        dataGridViewPointConfig.Rows[0].Cells[7].Value = false;
        //Tiempo de objetivo
        dataGridViewPointConfig.Rows[0].Cells[8].Value = 200;
        //Tolerancia de objetivo
        dataGridViewPointConfig.Rows[0].Cells[9].Value = 0.2;
        //Velocidad máxima de giro
        dataGridViewPointConfig.Rows[0].Cells[10].Value = 50;
        //CA habilitado
        dataGridViewPointConfig.Rows[0].Cells[11].Value = true;
        //Conducción inversa
        dataGridViewPointConfig.Rows[0].Cells[12].Value = false;
        //Tolerancia de giro en objetivo
        dataGridViewPointConfig.Rows[0].Cells[13].Value = 10;

        SendP2PGoCmd(0);
        controlando = 1;
        lbl_status.Text = "Ir a punto de vigilancia";
```

```
38          break;
     case 1://De camino a punto vigilancia
40          if ((robotP2PStatus == DrRobotP2PSpeedDrive.
    P2PServiceStatus.P2POver))
            {
42              controlando = 2;
            }
44
            break;
46      case 2://Buscar personas con los sensores
            lbl_status.Text = "Detectar personas";
48
            if(sensorData.UsDis[1]<0.8)
50          {
                controlando = 3;
52              string videoStreamAddress = "rtsp://admin:admin@192
    .168.0.100/play1.sdp";

54              //Initialize the capture device
                grabber = new Capture(videoStreamAddress);
56
                //Initialize the capture device
58  //              grabber = new Capture();
                grabber.QueryFrame();
60              //Initialize the FrameGraber event
                Application.Idle += new EventHandler(FrameGrabber);
62          }
            break;
64      case 3://Buscar personas con la cámara
            lbl_status.Text = label29.Text;
66          string usuario = "Abel,";
            if (string.Compare(label29.Text, usuario)==0)
68          {
                Application.Idle -= new EventHandler(FrameGrabber);
70              controlando = 4;
            }
72          break;
     case 4://Si detecta a un conocido, acompañar al punto de
    entrada
74          //Posición X
            dataGridViewPointConfig.Rows[0].Cells[0].Value =
    puntoEntrada.getPositionX();
76          //Posición Y
            dataGridViewPointConfig.Rows[0].Cells[1].Value =
    puntoEntrada.getPositionY();
78          //Orientación
            dataGridViewPointConfig.Rows[0].Cells[2].Value =
    puntoEntrada.getOrientation() / Math.PI * 180.0; //A grados
80          //Tiempo de transito
```

```
            dataGridViewPointConfig.Rows[0].Cells[3].Value = Math.
        Sqrt((puntoVigilancia.getPositionX() - robotPosition.robotX)
        * (puntoVigilancia.getPositionX() - robotPosition.robotX) + (
        puntoVigilancia.getPositionY() - robotPosition.robotY) * (
        puntoVigilancia.getPositionY() - robotPosition.robotY)) /
        0.1;
82          //Velocidad avance
            dataGridViewPointConfig.Rows[0].Cells[4].Value = 0.3;
84          //Punto olvidable
            dataGridViewPointConfig.Rows[0].Cells[5].Value = false;
86          //Sin parada
            dataGridViewPointConfig.Rows[0].Cells[6].Value = false;
88          //Postura final
            dataGridViewPointConfig.Rows[0].Cells[7].Value = false;
90          //Tiempo de objetivo
            dataGridViewPointConfig.Rows[0].Cells[8].Value = 200;
92          //Tolerancia de objetivo
            dataGridViewPointConfig.Rows[0].Cells[9].Value = 0.2;
94          //Velocidad máxima de giro
            dataGridViewPointConfig.Rows[0].Cells[10].Value = 60;
96          //CA habilitado
            dataGridViewPointConfig.Rows[0].Cells[11].Value = true;
98          //Conducción inversa
            dataGridViewPointConfig.Rows[0].Cells[12].Value = false;
100         //Tolerancia de giro en objetivo
            dataGridViewPointConfig.Rows[0].Cells[13].Value = 10;
102
            SendP2PGoCmd(0);
104         controlando = 5;
            break;
106     case 5:
            //Ir a puerta de entrada
108         lbl_status.Text = "Ir a puerta de entrada";
            if ((robotP2PStatus == DrRobotP2PSpeedDrive.
        P2PServiceStatus.P2POver))
110         {
                controlando = 0;
112         }
            break;
114     }
}
```

Código 7: Temporizador tarea vigilar.

8. Cerrar aplicación

```
private void DrRobotX80DemoForm_FormClosed(object sender,
    FormClosedEventArgs e)
{
    // armamos el XDocument
    // los XElement pueden ir anidados
    XDocument xdoc = new XDocument(new XElement("Shadows",
        from s in shadows
        select new XElement("Shadow",
                new XElement("PosX", s.getPositionX().ToString()
    ),
                new XElement("PosY", s.getPositionY().ToString()
    ),
                new XElement("Confidence", s.getConfidence().
    ToString()),
                new XElement("Time", s.getTime().ToString())
                )
            ));
    xdoc.Save("shadows_serial.xml");

    p2pDrive.DeregisterP2PDriveCmdCallback();
    sensorMapBuilder.DeregisterSensorMapCallback();
}
```

Código 8: Guardar mapa de ocupación al cerrar.

1. Planificación temporal

A continuación se presentan dos planificaciones temporales, la primera es la que se definió al inicio del proyecto y la segunda se ha ido adaptando a medida que avanzaba el trabajo. La planificación inicial se ha hecho de modo orientativo considerando las fases iniciales del proyecto. El la planificación final, se han ido modificando la planificación al término de cada fase para subsanar los problemas que han ido apareciendo.

1.1. Planificación inicial

En la planificación inicial se ha tenido en cuenta los siguientes trabajos a realizar.

- Descripción del problema y reuniones. Se realizarían 3 reuniones, en la primera se decidiría el tipo de robot a programar y algunos detalles de la aplicación. En las sucesivas reuniones se iría controlando el progreso del trabajo.
    - Tiempo estimado: 15 horas.
- Estudio lenguaje C#. Esta es etapa del aprendizaje del lenguaje C#, así como la instalación del software necesario para su programación.
    - Tiempo estimado: 45 horas.
- Estudio API robot. En esta etapa se estudiaría la librería de funciones del robot, así como sus dispositivos hardware.
    - Tiempo estimado: 30 horas.
- Investigación reconocimiento facial. En esta fase se investigaría sobre algoritmos y programas de reconocimiento facial.
    - Tiempo estimado: 45 horas.
- Desarrollo aplicación. Durante este periodo se desarrollaría la aplicación.
    - Tiempo estimado: 60 horas.
- Pruebas del sistema. Este sería el tiempo destinado a probar la funcionalidad del robot.
    - Tiempo estimado: 21 horas.

- Documentación. Aquí se documentaría todas las actividades que se realicen el TFG, planificación temporal, estudio del arte, definición de requisitos, etc.
    - Tiempo estimado: 72 horas.
- Presentación. Durante este tiempo se prepararía la presentación del TFG.
    - Tiempo estimado: 12 horas.

El total de horas de la planificación inicial suma 300 horas, que es el número de horas que debería durar un trabajo final de carrera. En la Figura 1.2 se muestra un Diagrama de Gantt en el que aparecen las horas dedicadas a cada apartado a lo largo del tiempo.

1.2. Planificación final

En este apartado se va a detallar el tiempo empleado a cada una de las secciones previstas.

- Descripción del problema y reuniones. En esta fase el tiempo estimado y el empleado coincidieron. Los motivos son porque no se hicieron más reuniones de las previstas, y su duración se ha controlado adecuadamente.
    - Tiempo estimado: 15 horas.
    - Tiempo empleado: 15 horas.
- Estudio lenguaje C#. En el estudio del nuevo lenguaje utilicé menos horas de las previstas.
    - Tiempo estimado: 45 horas.
    - Tiempo empleado: 42 horas.
- Estudio API robot. Para completar esta fase se ha utilizado el tiempo que se estimó.
    - Tiempo estimado: 30 horas.
    - Tiempo empleado: 30 horas.
- Investigación cámara inalámbrica. Como al inicio no se había contemplado la necesidad de utilizar una cámara inalámbrica, se ha tardado 33 horas más de lo previsto.

    - Tiempo estimado: 0 horas.
    - Tiempo empleado: 33 horas.
- Investigación reconocimiento facial. Para completar esta tarea se ha utilizado el tiempo previsto.
    - Tiempo estimado: 45 horas.
    - Tiempo empleado: 45 horas.
- Desarrollo aplicación. Esta etapa se ha tardado lo que se planteó inicialmente.
    - Tiempo estimado: 60 horas.
    - Tiempo empleado: 60 horas.
- Pruebas del sistema. En esta fase se ha tardado menos de los estimado, en parte gracias al conocimiento adquirido sobre la plataforma.
    - Tiempo estimado: 21 horas.
    - Tiempo empleado: 18 horas.
- Documentación. En esta etapa se ha tardado menos de lo estimado.
    - Tiempo estimado: 72 horas.
    - Tiempo empleado: 66 horas.
- Presentación. En la preparación de la presentación se ha tardado más de lo estimado inicialmente.
    - Tiempo estimado: 12 horas.
    - Tiempo empleado: 15 horas.

En total se ha tardado 324 horas en completar el proyecto. Son más horas de las estimadas para un TFG, pero teniendo en cuenta que han aparecido secciones que no se tenían en cuenta, un incremento del 8 % en un trabajo de estas características no se considera excesivo. En las Figuras 1.2 y 1.2 se muestran los Diagramas de Gantt con la programación temporal empleada. Se muestran en la misma hoja para poder comparar la planificación inicial con la planificación real.

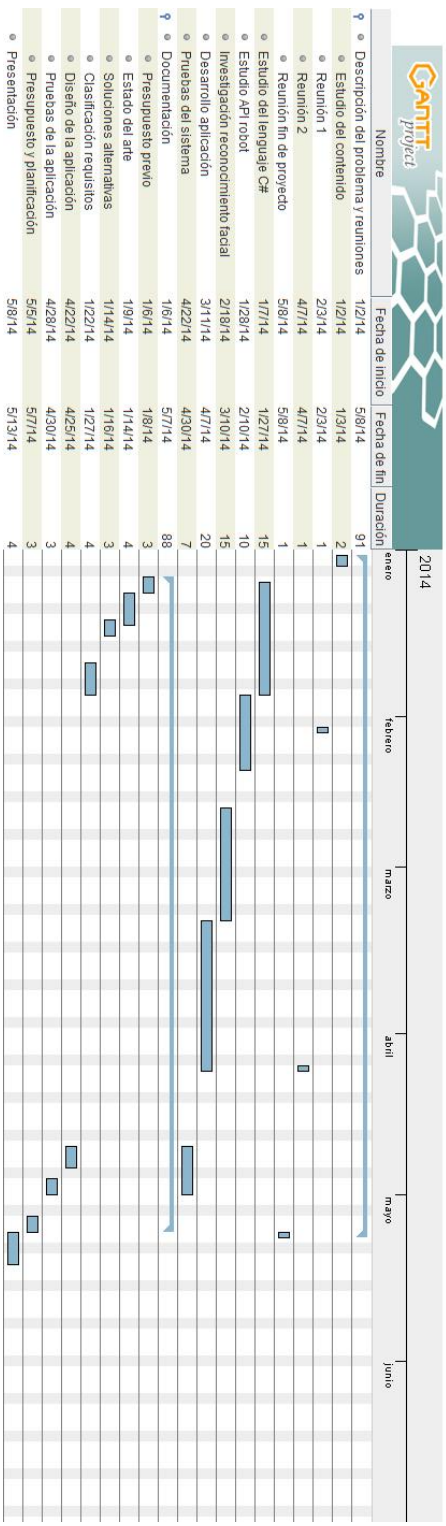

Figura 2: Diagrama de Gantt final.

Figura 1: Diagrama de Gantt inicial.

Bibliografía

[1] Biblioteca de funciones para visión artificial
openCV, 2014 (Última versión), www.opencv.org

[2] Wrapper para utilizar openCV en lenguajes .Net
Emgu CV, 2013 (Última versión), www.emgu.com

[3] Blog con información cámaras IP
Rolando Palermo, 2012, http://blog.rolandopalermo.com/2012/03/camara-ip-cam-java.html

[4] Tutorial configurar openCV en Visual StudioKike
Coronado, 2013,
http://mecatronicauaslp.wordpress.com/2013/07/31/configurar-un-proyecto-de-opencv-2-4-6-en-visual-studio-2010-x86x64

[5] Wiki con códigos para controlar cámaras Trendnet
Zoneminder, 2014, http://www.zoneminder.com/wiki/index.php/Trendnet

[6] Página oficial cámara Trendnet
Trendnet, 2014, http://www.trendnet.com/langsp/products/proddetail.asp?prod=210_TV-IP672WI#tabs-solution02

[7] Blog con códigos para controlar cámaras Trendnet
Ispyconnect, 2014, http://www.ispyconnect.com/man.aspx?n=TrendNet

[8] Web con códigos para movimiento de cámaras Trendnet
Jonathan McCurry, 2008, http://www.lavrsen.dk/foswiki/bin/view/Motion/TrendNet

[9] Página oficial plataforma robótica X80Pro
Drrobot, 2012, http://www.drrobot.com/products_item.asp?itemNumber=X80Pro

[10] Hoja de características robot Summit
Summit, 2013, http://www.robotnik.es/ilsupload/Summit.pdf

[11] International Federation of Robotics, World Robotics Report 2013.

[12] Página oficial avión UAV
MicroPilot, 2014, http://www.micropilot.com/products-mp-visione.htm

[13] Página oficial hexacóptero
Screen Copter, 2014, http://www.screencopter.com/index.php/es/hexacopter.html

[14] Página oficial robot bipedo
Pal robotics, 2014, http://reemc.pal-robotics.com/es

[15] Página oficial laser de rango
Hokuyo, 2008, https://www.hokuyo-aut.jp/02sensor/07scanner/ubg_04lx_f01.html

[16] Reconocimiento facial utilizando openCV y .NET
2012, Sergio Andrés Gutiérrez, http://www.codeproject.com/Articles/239849/Multiple-face-detection-and-recognition-in-real-ti

[17] Pulli, K., Baksheev, A., Kornyakov, K., & Eruhimov, V. (2012). Real-time computer vision with OpenCV. Communications of the ACM, 55(6), 61-69.

[18] Lorsakul, A., & Suthakorn, J. (2007). Traffic sign recognition for intelligent vehicle/driver assistance system using neural network on OpenCV. In Proceedings of the 4th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI 2007) (pp. 22-24).

[19] Pisarevsky, V. (2007). Opencv object detection: theory and practice. Intel Corporation, Software and Solutions Group, Slide presentation.