UNIVERSIDAD REY JUAN CARLOS

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Curso académico 2004-2005

Proyecto Fin de Carrera

Navegación visual del robot Pioneer.

Tutor: José M. Cañas Plaza

Autor: Pedro Miguel Díaz Peña

Para mi novia y mi familia.

Agradecimientos.

Quiero agradecer la gran ayuda prestada por mis compañeros del grupo de robótica de la URJC. Destacando de manera especial, el esfuerzo y trabajo de José Mª Cañas, tutor del proyecto.

También quiero resaltar el gran apoyo de mi novia y mi familia, sin los que no habría terminado nada de lo que me propuesto en la vida.

No olvidándome de mis compañeros de clase Manolo, Nieves , Antonio, Kike, Dani, Javi, con los que me une una gran amistad.

Gracias a todos.

Índice general

Índice de figuras

Resumen

En los últimos años, los enormes avances tecnológicos han permitido el desarrollo de nuevas disciplinas científicas como es el caso de la visión por ordenador. El uso de la visión computacional está creciendo en la robótica. Fundamentalmente porque es un sensor muy barato y puede aportar mucha información. El principal problema, es la complejidad de extraer, del flujo de imágenes, información útil para la tarea del robot. Dentro de la la visión computacional en robótica se realizan técnicas de reconocimiento de patrones, de reconstrucción 3D, de navegación reactiva, etc.

El objetivo de este proyecto es que un robot pueda navegar por un entorno semi-estructurado detectando los obstáculos cercanos únicamente con visión monocular. La dificultad reside en el uso exclusivo de una sola cámara en contraste con las técnicas clásicas de navegación con sensores de distancia (sonar-láser) y con las técnicas de visión estéreo, que usan triangulación desde 2 o más cámaras.

Utilizando un modelo de proyección para la cámara estimamos la distancia de los obstáculos al robot. Basándose en la *hipótesis de suelo*, por la que toda imagen esta compuesta por suelo y obstáculos, se determinan los puntos que corresponden a los *bordes* que delimitan el suelo de los objetos. Realizando un barrido, con el cuello mecánico, se fusionan los datos de profundidad en el espacio real que servirán como referencia para navegar. Éstos, se segmentan en unidades homogéneas en relación a su posición en el entorno. Una vez agrupados los datos, el algoritmo de navegación determina la velocidad y sentido del robot en cada momento.

La forma de implementar la funcionalidad que se busca ha sido a través de la plataforma *jde.c*. Desarrollada por el grupo de robótica de la URJC, resuelve los aspectos más generales de la programación de robots. El proyecto se ha implementado por medio de tres hebras iterativas o esquemas. Uno reactivo, encargándose de la navegación del robot. Dos perceptivos, el primero tratando los aspectos más cercanos a la obtención de información de las imágenes capturadas por la cámara y un segundo, encargado de abstraer los datos de profundidad para distinguir estructuras conocidas, como por ejemplo la forma de un pasillo .

Capítulo 1

Introducción

A grandes rasgos el objetivo de este proyecto es hacer que un robot se mueva en un entorno utilizando como única fuente de datos una cámara. Este tipo de sensor aporta mucha información con el inconveniente de la complejidad de extraer datos útiles para la tarea del robot.

En este capítulo se describe el contexto global en el que se desarrolla este proyecto fin de carrera: comienza con una reseña histórica de los inicios de la robótica, repasando sus orígenes industriales y recorriendo algunos ejemplos de sus aplicaciones en la actualidad. A continuación se introduce un pequeño resumen del termino *navegación local* y sus aplicaciones, así como los algoritmos más interesantes, y una breve descripción de la visión por ordenador, una de las técnicas en las que se basa este proyecto.

1.1. Robótica

Una obra checoslovaca publicada en 1917 por Karel Kapek, denominada Rossum's Universal Robots, dio lugar al término robot. La palabra checa *"Robota"* significa servidumbre o trabajador forzado, y cuando se tradujo al ingles se convirtió en el término robot.

Por siglos el ser humano ha construido máquinas que imiten las partes del cuerpo humano. Los antiguos egipcios unieron brazos mecánicos a las estatuas de sus dioses. Estos brazos fueron operados por sacerdotes, quienes clamaban que el movimiento de estos era inspiración de sus dioses. Los griegos construyeron estatuas que operaban con sistemas hidráulicas, los cuales se utilizaban para fascinar a los adoradores de los templos. Durante los siglos XVII y XVIII en Europa fueron construidos muñecos mecánicos muy ingeniosos que tenían algunas características de robots. Jacques de Vauncansos construyó varios músicos de tamaño humano a mediados del siglo XVIII. Esencialmente se trataba de robots mecánicos diseñados para un propósito específico: la diversión.

El desarrollo en la tecnología ha contribuido a flexibilizar los mecanismos autómatas para desempeñar tareas dentro de la industria. Son varios los factores que intervienen para que se desarrollaran los primeros robots en la década de los 50.

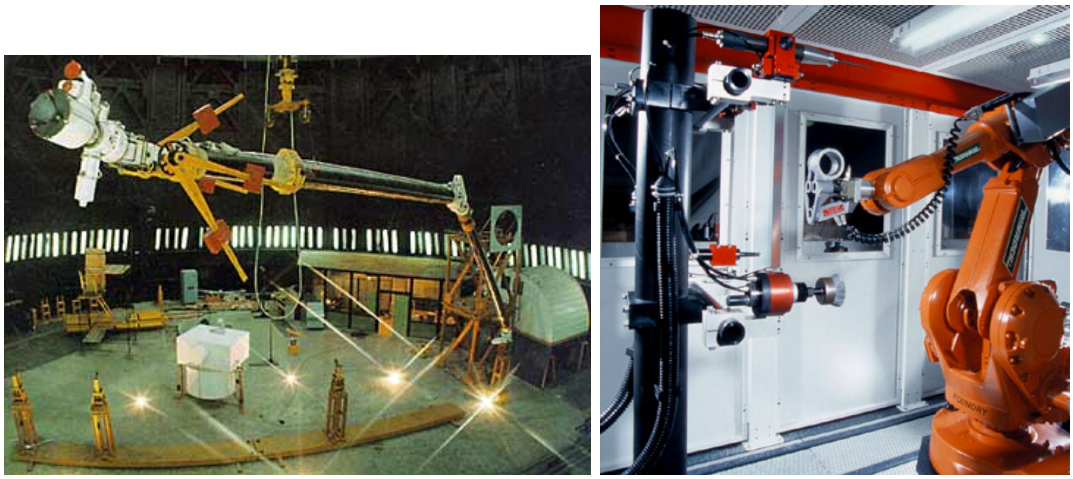


Figura 1.1: Brazo robótico transbordador espacial, brazo robótico industrial

Los robots son usados hoy en día para llevar a cabo tareas peligrosas, difíciles y repetitivas. La manufactura continúa siendo el principal mercado donde los robots son utilizados. En particular, robots articulados (similares en capacidad de movimiento a un brazo humano) son los más usados comúnmente. Estos son utilizados en una diversidad de aplicaciones, desde robots soldadores en la industria automotriz, hasta brazos teleoperados en el transbordador espacial. La industria automotriz ha tomado gran ventaja de esta nueva tecnología donde los robots han sido programados para reemplazar el trabajo de los humanos en muchas tareas repetitivas.

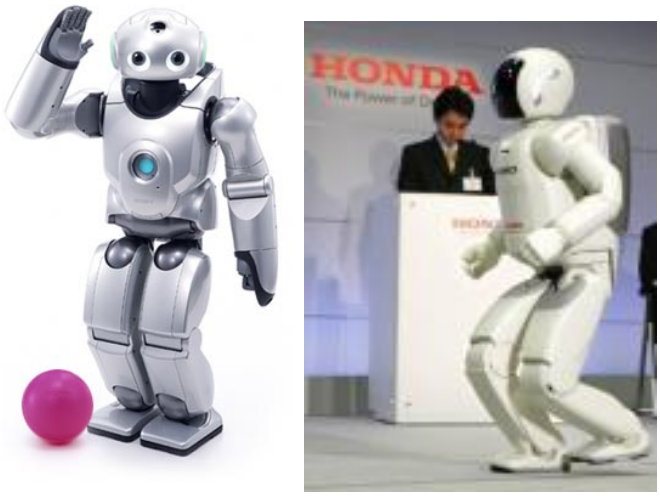


Figura 1.2: Robot QRIO de Sony, robot ASIMO de Honda

Asimismo, es necesario recalcar que en la actualidad se encuentran robots que cumplen sólo con la función de entretener, sin realizar trabajo útil, aún cuando pueden utilizar tecnología avanzada. Este es el caso del perro mascota *AIBO* de Sony también utilizado como jugador de fútbol en las competiciones mundiales de robots como la Robocup[1].

En estos momentos se está avanzando en los humanoides, robots que se limitan a imitar los actos y gestos humanos. Un robot humanoide puede tener enormes ventajas para cierta clase de funciones. Los más avanzados son el *ASIMO* de Honda y *QRIO* de Sony consiguiendo movimientos bípedos muy complicados como saltar y subir escaleras.

El futuro de los robots promete ser increíble, ya que el hombre acompaña sus avances y creaciones con todas las "herramienta" que tiene a su alcance, para realizar sus tareas, y sobre todo las que sean peligrosas o directamente imposibles para él.

1.2. Navegación de Robots

El primer problema de los robots móviles es la navegación. El campo de la robótica móvil se caracteriza por la gran cantidad de incertidumbre presente en los entornos reales. La autonomía de navegación es en un agente que requiere de capacidades complejas.

Se llama navegación al conjunto de métodos y técnicas usados para dirigir el curso de un robot móvil a medida que éste atraviesa su entorno. El robot tiene que llegar a algún destino sin chocar ni con obstáculos fijos, ni con otros móviles que eventualmente puedan aparecer el en camino. Con la aparición de la navegación de robots han surgido problemas específicos como la construcción de mapas, la localización, la planificación de caminos, etc. Para efectuar una navegación lo más común es disponer de un mapa del entorno, aunque no necesariamente. En esta diferencia radica la división de la navegación robótica en dos bloques, la navegación global y la navegación local.

La navegación global se basa en calcular de antemano la ruta a seguir por el robot. Se encarga de generar un plan de acción para mover el robot desde un punto origen a un punto destino. Para ello se ha de tener conocimiento a priori del entorno. Este conocimiento puede ser adquirido mediante un mapa del entorno o construido por medio de las medidas sensoriales del robot. Para calcular la ruta se usan técnicas de planificación o búsqueda que son bastante costosas en tiempo. Algunos ejemplos de

[1] http://www.robocup.org

estas técnicas pueden ser: grados de visibilidad (Nilsson 1969) que proporcionan un enfoque geométrico en el que cada nodo del grafo es cada uno de los vértices de los obstáculos que pertenecen al mapa, y planificación basada en gradiente [Konolige, 2000] en que se asigna a un valor numérico a cada punto del espacio, y se genera una onda desde el destino hasta encontrar el robot.

La navegación local por el contrario es reactiva apoyándose en medidas sensoriales obtenidas a cada momento y que le sirven al robot para tomar decisiones en ese instante. Esto implica que este tipo de navegación sea sensible, a muy corto plazo, de cambios en el entorno como son objetos dinámicos o en movimiento. Esta navegación es importante en entornos donde pueden aparecer obstáculos imprevistos en cualquier momento, como puede ser unas oficinas, puesto que las personas que trabajan en ellas se mueven constantemente. Algunos métodos de navegación son:

- Método de velocidad y curvatura(CVM)[Reid Simmons,1996]:Este método trabaja añadiendo restricciones al espacio de velocidad y escogiendo el punto en el espacio que satisface todas las restricciones y maximiza una función objetivo. Las restricciones derivan de las limitaciones físicas de las velocidad y aceleración propias del robot, y de los datos sensoriales que indican la presencia de obstáculos. Éstos últimos se usan para representar, para cada posible curvatura, cuan lejos puede viajar el robot antes de colisionar con un obstáculo. La distancia curva a los obstáculos se haya dividiendo el espacio de velocidad en un número discreto de regiones, cada una de los cuales tiene una distancia constante al punto de impacto.Este método encuentra el punto de cada región que maximiza la función objetivo. El punto máximo de entre todos, se usa para comandar el robot.

- Método de carriles y velocidad (VLM)[Reid Simmons,1996]:Este método combina el CVM con una nuevo método direccional llamado el método de carriles. El método de carriles divide en entorno en carriles, y luego elige el mejor a seguir para optimizar el viaje hacia la dirección deseada. Una dirección local es calculada para entrar y seguir el mejor carril, y CVM usa esta dirección para determinar las velocidades lineal y angular óptimas. Para ello considera la dirección a seguir, las limitaciones físicas y las restricciones del entorno. Combinando los métodos de dirección y velocidad, VLM consigue un movimiento libre de colisiones así como un movimiento suave teniendo en cuenta la dinámica del robot.

- Método de fuerzas virtuales (VFF)[J. Borenstein, 1989]: Esta técnica se basa en que el destino del robot ejerce una fuerza atractiva sobre él. Los obstáculos ejercen también una fuerza, en este caso repulsiva, sobre el robot. Cuanto más cerca estén estos menos atracción o repulsión ejercen. Así el movimiento del robot está go-

bernado por el sumatorio vectorial de dichas fuerzas. Esto supone que el robot cambiará su rumbo en el momento detectar un obstáculo y lo recuperará una vez superado éste. Es la técnica que usaremos sobre el robot para dotarlo de navegación local.

La navegación híbrida que combina la planificación de caminos con la ejecución reactiva de esos planes es uno de los enfoques usados en la navegación de robots actualmente. Un ejemplo de esta navegación es el robot *XAVIER*[2]

1.3. Visión en Robots Móviles

Visión es la ventana al mundo de muchos organismos. Su función principal es reconocer y localizar objetos en el ambiente mediante el procesamiento de las imágenes. La visión computacional es el estudio de estos procesos, para entenderlos y construir máquinas con capacidades similares. Un área muy ligada a la de la visión computacional es el procesamiento de imágenes. El objetivo del procesamiento de imágenes es mejorar la calidad de estas para su posterior utilización o interpretación.

Actualmente existen muchas aplicaciones prácticas de la visión computacional como por ejemplo reconocimiento de matrículas de vehículos, reconocimiento facial, reconocimiento de iris, reconocimiento de huellas dactilares, etc.

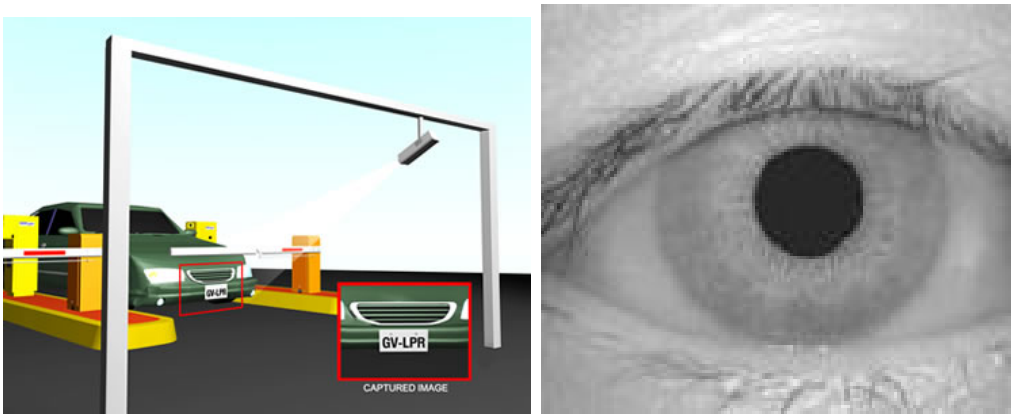


Figura 1.3: Simulación de reconocimiento de matrículas, Imagen de iris

En la robótica móvil se utilizan cámaras para localizar obstáculos, identificar objetos y personas, encontrar caminos, etc. La visión es un sensor más para el robot. La cámara es un sensor más como fuente de información del entorno. Rico en información pero difícil de extraer flujo útil directamente, para la tarea del robot.

[2]http://www.cs.cmu.edu/ Xavier/

La visión puede ser utilizada para generar comportamiento. No es necesario un procesamiento visual muy complicado para generar comportamiento complejo. En esta linea aparecen robots capaces de seguir objetos en movimiento o navegar por un entorno. Una de las técnicas en la que se puede basar un comportamiento de navegación visual es lo que llamamos *hipótesis de suelo*[Ian Horswill] en la que se asume que una cámara situada encima del robot está inclinada hacia abajo. La parte inferior de las imágenes es suelo, subiendo en ésta, cuando aparece una discontinuidad con el suelo ahí sabemos que hay un obstáculo.

La localización visual, basada en el reconocimiento de patrones comparando éstos con el mapa memorizado del entorno, es otra de las técnicas de la robótica basadas en visión.

1.4. Navegación visual del Robot Pioneer

El objetivo de este proyecto es que un robot pueda deambular por su entorno detectando los obstáculos cercanos exclusivamente con visión monocular.

En esta linea también se han desarrollado proyectos fin de carrera que utilizan como sensor una cámara, como el comportamiento sigue persona [Calvo04] [Herencia04], localización visual del robot Pioneer [Alberto05], representación visual de escena con una cámara direccional [Marta05], etc.

Figura 1.4: Robot perteneciente al grupo de Robótica de la URJC

Nuestro proyecto intenta resolver el problema de la navegación local. En contraposición de otras técnicas de navegación local, que se apoyan en sensores de distancia como el láser o el sonar, utilizamos la visión monocular como único sensor. Este nos ofrecerá un flujo de datos que procesaremos, apoyándonos en técnicas comentadas en

puntos anteriores para generar una navegación robusta en un entorno como el de la figura 1.4.

Estos comportamientos se programan sobre la plataforma desarrollada íntegramente en el grupo de robótica de la Universidad Rey Juan Carlos [3] llamada *JDE* (Jerarquía Dinámica de Esquemas). *JDE* proporciona un acceso sencillo a los sensores y actuadores del robot. Se basa en crear esquemas, o hebras iterativas, cada una de las cuales encargada de una función determinada; consiguiéndose generar comportamientos complejos con la ejecución simultanea de estas.

Para terminar este capítulo se indica la estructura propuesta para la memoria: en la que en el capitulo 3 se describe la plataforma de desarrollo que hemos usado para realizar este proyecto (*JDE* "Jerarquía Dinámica de Esquemas"), capítulo 4 donde se detalla el diseño general, las técnicas y los algoritmos utilizados para conseguir los objetivos marcados al comienzo del proyecto, capítulo 5 explicaremos las pruebas que se han realizado. Por último se exponen las conclusiones y las posibles lineas futuras que derivan tras la realización el proyecto.

[3] http://gsyc.escet.urjc.es/robotica

Capítulo 2

Objetivos

Una vez comentado en el capítulo de introducción el contexto de este proyecto, en este capítulo vamos a fijar los objetivos concretos, los requisitos de partida y la metodología utilizada.

2.1. Objetivos

El objetivo principal de este proyecto es que un robot pueda deambular en un entorno de oficinas usando como único sensor una cámara situada encima de este. Se plantean ciertos problemas a la hora de generar este comportamiento, que hay que superar para conseguir el objetivo marcado. Como objetivo global que se persigue es la navegación del robot Pioneer por un entorno semi-estructurado, como son los pasillos del Departamental de la ESCET, sin colisionar con ningún obstáculo. Este objetivo global desglosado seria:

- 1. Desarrollo de un algoritmo capaz obtener los bordes de una imagen de manera robusta.
- 2. Obtener una estimación de profundidad de los obstáculos cercanos fusionando, en una representación local de distancias más amplia que el campo visual de la cámara, y mostrar esta estimación en una representación segmentada.
- 3. Desarrollar un método de navegación local basado en VFF y realizar pruebas en el simulador.
- 4. Generar un algoritmo de navegación sobre el robot real, experimentar y ajustar.

Los subobjetivos 1-2 son la parte perceptiva del comportamiento del robot y los puntos 3-4 son la parte reactiva. Si conseguimos implementar una aplicación que cumpla estos 4 puntos habremos conseguido generar una navegación reactiva usando como única fuente de datos la visión monocular.

2.2. Requisitos

Una vez explicados los objetivos del proyecto, este se debe ajustar a los siguientes requisitos para asegurar su buen comportamiento:

- 1.El robot sobre el que esta pensado el proyecto es el Pioneer 3-DX, pudiéndose apoyar en herramientas de simulación como es *Player/Stage*. Pero el proyecto final deberá de funcionar en el robot real. Se ha de desarrollar el proyecto sobre la *plataforma JDE* que se explica detalladamente en el siguiente capitulo. En concreto, debe de funcionar sobre la versión *JDE (2.4) , oculo (3.8) y otos (5.2)*. Este entorno restringe la utilización de otro lenguaje de programación que no sea el lenguaje C, aunque en versiones futuras soportará C++ [Lobato05].

- 2.Deberá disponer de una cámara como única fuente de datos. También hará uso necesariamente de un cuello mecánico situado encima del robot.

- 3.El algoritmo de filtrado de bordes debe de ser muy robusto dado que todas las medidas se basan en estas discontinuidades.

- 4.La navegación debe ser reactiva, en todo momento. Los algoritmos deben ser lo más simplificado posible para que el robot tenga un movimiento continuo y progresivo. Cuanto menos CPU consumamos, mas hueco dejamos para otros algoritmos como por ejemplo planificadores deliberativos o algoritmos de localización que soy muy costos. Más iteraciones de nuestro algoritmo redundarán en una mayor calidad del movimiento.

- 6.Deberá tener un interface para interactuar con el usuario en el que se muestre en cada momento la situación del robot y los datos obtenidos.

Además de estos requisitos a lo largo del proyecto pueden surgir otros requisitos derivados de los posibles cambios en la forma de enfocar la resolución del proyecto.

2.3. Metodología

En esta sección se pasa a describir el método utilizado para la realización de este proyecto. Básicamente se han realizado iteraciones sobre el diseño e implementación, así como reuniones semanales con el tutor.

El desarrollo del proyecto se ha basado en un modelo de desarrollo en espiral. La elección de este modelo viene determinada, en cierto modo, por la estructura sobre la que se implementa el proyecto, dividiendo el problema en tareas que una vez agrupadas generan el comportamiento buscado.

En la figura 2.1 se observan las etapas que componen este modelo de desarrollo: Análisis de Requisitos, diseño e implementación, pruebas y planificación del siguiente ciclo.

La ventaja principal del modelo en espiral es que, al basarse en prototipos, en cada iteración existen puntos de verificación que ayudan afrontar el proyecto mediante etapas, además de ser flexible en cuanto a cambios en los requisitos iniciales. Los objetivos de cada etapa vienen determinados por un plan de trabajo inicial en el que se indican las tareas a llevar a cabo semanalmente.

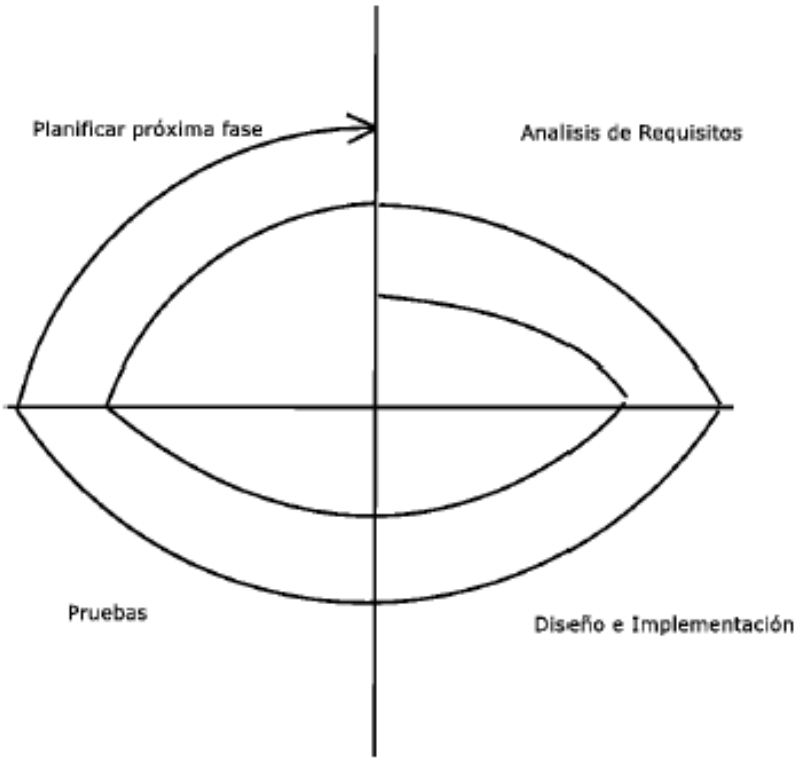


Figura 2.1: Modelo en Espiral

Las iteraciones generadas a lo largo de este proyecto son:

Prototipo 1: Filtro de bordes(primeras dos semanas).

Prototipo 2: Barrido del cuello mecánico(tercera semana).

Prototipo 3: Fusión de los datos de profundidad generados con el movimiento del cuello mecánico (las dos semanas posteriores).

Prototipo 4: Segmentación de los datos de profundidad (aproximadamente dos semanas).

Prototipo 5: Obtención de segmentos paralelos con unas determinadas cualidades (pasillo)(una semana).

Prototipo 6: VFF con destino "final del pasillo" y como repulsión los datos de profundidad (aproximadamente 3 semanas).

Estos prototipos han sido presentados en versiones numeradas al tutor del proyecto en las reuniones programadas semanalmente. Una vez finalizada toda la implementación del código se pasa a escribir la memoria del proyecto, usando para ello unas ocho semanas de trabajo.

Cabe destacar el trabajo que se realizo previo a la ejecución de este proyecto. En un periodo de formación en el que se aprendieron conocimientos básicos de la robótica y se describieron otras plataformas para la programación de robots autónomos.

Capítulo 3

Plataforma de desarrollo

En este capítulo se detallan las herramientas software y hardware sobre las que se ha desarrollado este proyecto. Comienza con una descripción del robot sobre el que se ejecuta la aplicación. Posteriormente profundiza en la arquitectura y modelo de programación utilizados. Y para terminar se realiza una descripción breve de la biblioteca de tratamiento de imágenes y del simulador *Player/Stage*.

3.1. Robot Pioneer

El hardware disponible es el robot Pioneer 3-DX(figura 3.1). Es una plataforma móvil ágil y versátil que comercializa la empresa ActivMedia[1]. Este modelo de la gama Pioneer viene equipado con un procesador de 18 MHz Hitachi H8S/2357 con 32Kb RAM y 128Kb Memoria flash. Tiene una autonomía de 5 horas y es capaz de adquirir una velocidad máxima de 1,6 m/s.

Figura 3.1: Robot Pioneer

Dispone de un equipo sensorial que le permite obtener información de su entorno. El robot Pioneer lleva una corona de 16 transductores de ultrasonido (8 en la parte

[1] http://www.activmedia.com/

delantera y 8 en la parte trasera) que pueden ser utilizados para calcular distancias o detectar obstáculos cercanos. También incorpora un sensor láser con 180 grados de amplitud que, a diferencia de la corona de ultrasonidos, es mucho más preciso. Lleva unos odómetros (encoders) asociados a las ruedas para saber cuanto han girado éstas. Por último, una cámara situada encima de todo el conjunto completa el grupo de sensores disponibles. Ésta, puede ser de varios tipos (firewire, webcam, etc), diferenciados estos por las velocidades de transmisión y el tipo de conexión con el ordenador. En este proyecto hemos probado varias cámaras, pero finalmente hemos elegido el modelo VIDERE DCAM, cámara firewire que ofrece imágenes a 30 fps(figura 3.2).

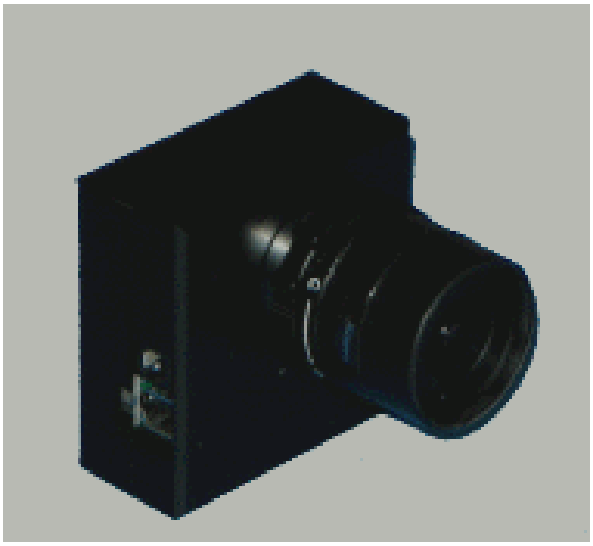


Figura 3.2: Cámara Videre DCAM

Respecto a los actuadores, el robot tiene tres ruedas, dos de las cuales son controladas por motores que dan la capacidad de poder navegar, y girar sobre si mismo. La tercera rueda situada en la parte posterior no tiene tracción y otorga la estabilidad al robot. Encima del láser se encuentra el cuello mecánico, con el que se puede, situando la cámara encima de este, visualizar el entorno sin necesidad de girar el robot [Cañas03b].

Encima de esta plataforma móvil, se incorpora un ordenador portátil. En esta máquina corre un sistema operativo GNU/Linux, en el cual se va a ejecutar la aplicación de este proyecto. Dispone de una tarjeta de red inalámbrica 802.11 que aporta, al conjunto, la comunicación del robot con otras máquinas.

3.2. Arquitectura y modelo de programación con *JDE*

JDE es un entorno que facilita la creación de programas para que el robot se comporte de determinada manera y exhiba conductas autónomas. La plataforma resuelve los aspectos más generales de la programación de robots, como el acceso a los sensores y actuadores, la multitarea, las interfaces gráficas y las comunicaciones entre programas.

En la práctica incluye tres servidores, varias librerías y un conjunto de ejemplos cuyo código puede servir de base para nuevas aplicaciones. Este entorno ha sido creado por el grupo de Robótica de la URJC [Cañas03]. *JDE* se encuentra a disposición de todo el mundo debido a su carácter de software libre, pues tiene licencia GPL [2].

JDE propone una organización para las aplicaciones robóticas como la de nuestro proyecto, basada en esquemas. La programación con esquemas *JDE* define la aplicación como un conjunto de esquemas que se ejecutan simultáneamente y en paralelo. Los esquemas no son más que procesos que funcionan concurrentemente y se encargan de una tarea sencilla y concreta. La ejecución simultánea de varios esquemas dan lugar a un comportamiento.
Existen dos tipos de esquemas en *JDE*: los esquemas perceptivos y los esquemas de actuación. Los esquemas perceptivos se encargan de producir y almacenar información que puede ser leída por otros esquemas. Esta información puede provenir de medidas sensoriales o de la información elaborada por otros esquemas. Los esquemas de actuación toman decisiones sobre los motores o activación de otros esquemas de nivel inferior a partir de la información que generan los esquemas perceptivos.
Los esquemas se pueden organizar en niveles de tal forma que los esquemas de un nivel inferior son activados o desactivados por los esquemas del nivel superior. De esta organización jerárquica podemos inferir la existencia de esquemas padre y esquemas hijo.

Para superar la limitación de que cualquier programa, que controle al robot, ha de ejecutarse en el portátil que lleva a bordo, nació la actual arquitectura de servidores y clientes *jde.c*. De un lado los servidores ofrecen acceso a los sensores y los comandos motrices a cualquier programa cliente, y del otro lado los distintos programas clientes solicitan esos servicios a través de la red. Cada servidor encapsula ciertos sensores y actuadores que hay en su máquina y ofrece su funcionalidad al resto de los programas a través de una interfaz de mensajes.

[2] http://gsyc.escet.urjc.es/jmplaza/software.html

Esta arquitectura permite aprovechar mejor la capacidad de cómputo disponible. Facilita el procesamiento distribuido y paralelo de la información que proviene de los sensores. No es necesario que un cliente se ejecute en la máquina donde residen los sensores, puede estar en cualquier ordenador a bordo del robot, e incluso en cualquiera exterior enganchado a la red. Este acceso remoto supone un valor añadido frente a entornos como Saphira o ARIA, que sólo ofrecen acceso local. Otra virtud de esta orientación a servidores y clientes es que hace más independiente a los programas del robot concreto.
Estas ventajas conllevan un coste en cuanto al aumento de los retardos entre que se lee el dato sensorial y este llega, atravesando los servidores y la red, al programa donde realmente se procesa.

Los clientes, con independencia de la máquina en que se ejecutan, solicitan medidas sensoriales y envían comandos intercambiando mensajes con los servidores a través de la red. Entre un cliente y un servidor se establece una conexión tcp, y a través de ella se establece entre ambos un diálogo regido por un protocolo. Este protocolo *JDE* entre servidores y clientes fija los mensajes posibles y las reglas o semántica que los acompañan. Está pensado para el envío continuo (streaming) y en él se ha establecido tres patrones de interacción: la suscripción a sensores cortos, el envío bajo demanda de imágenes y las órdenes de movimiento. Como ya hemos anticipado se define una interfaz de mensajes que ofrece el acceso remoto a los sensores y actuadores del robot. Así la llamada a función o la consulta a una variable que se utilizan en el acceso local se sustituyen en el acceso remoto por el envío de cierto mensaje. Si localmente las variables eran actualizadas por una tarea, ahora esas variables no residen en la máquina donde está el sensor y la tarea de actualización incluirá su petición al servidor correspondiente y la lectura desde la red de las respuestas oportunas. Del mismo modo la llamada a función con que se comanda un movimiento se sustituye por el envío del mensaje correspondiente al servidor que realmente materializa esas órdenes a los motores. Gracias a los servidores no es necesario que el cliente enlace ninguna librería de acceso local a sensores o motores, con lo que el programa puede compilarse en cualquier máquina fuera del robot.

La programación de aplicaciones directamente sobre los servidores *JDE* obliga a que la propia aplicación se encargue de enviar y recibir mensajes hacia/desde los servidores. Esto implica que el programador gaste cierto esfuerzo en crear los búfferes de recepción y de envío oportunos. Además debe tener en cuenta el tiempo de cómputo de su programa dedicado a las comunicaciones.
La programación en esquemas libera a la aplicación de esta tarea, ya que incorpora varios esquemas de servicio que se encargan de las comunicaciones: recoger de los ser-

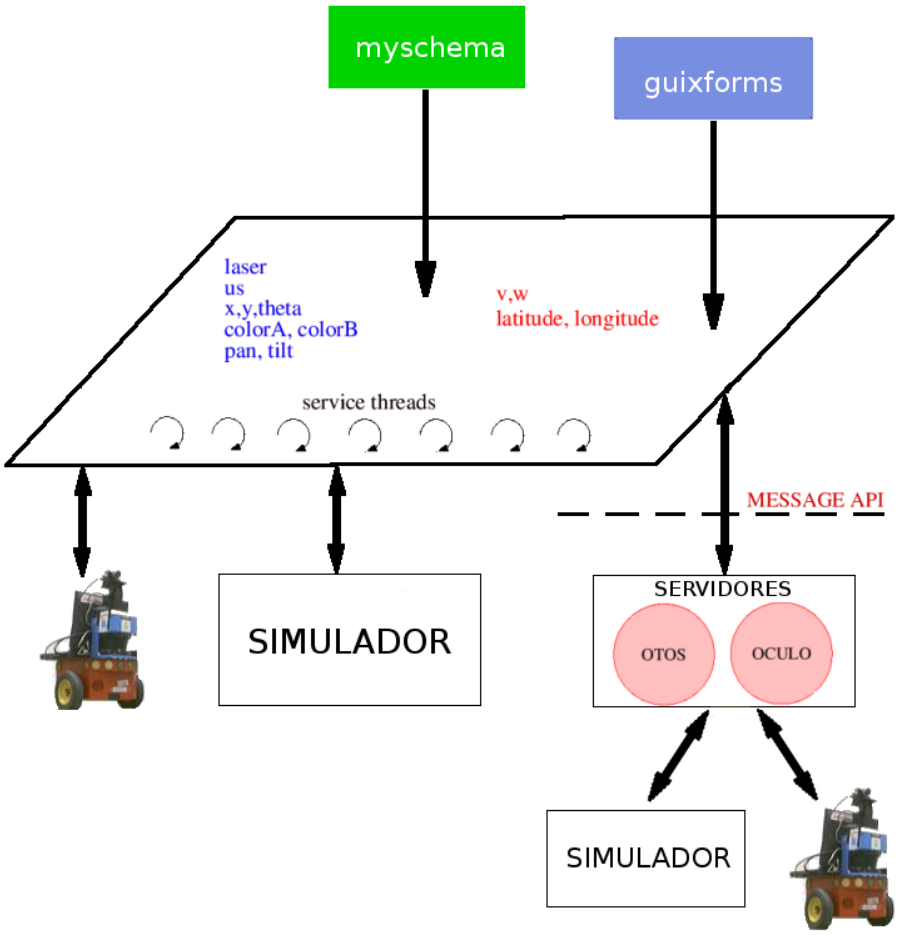


Figura 3.3: Plataforma *jde.c*

vidores la información de los sensores del robot, y enviar a los motores las órdenes de movimiento a través de los mismos servidores.

Este proyecto hará uso de la información ofrecida por los esquemas *Frontera* y *Pasillo*, que ponen a disposición del resto varias estructuras para poder generar una dinámica de movimiento del robot. Respecto a las variables de actuación, el esquema *Motors* pone a disposición del resto de esquemas las variables v, que comanda el valor de la velocidad lineal y w, que lo hace con la velocidad angular.

3.2.1. Servidor *Otos*

El servidor *otos* reúne los servicios de los sensores de proximidad como sónars, láser y táctiles. Además ofrece los datos de odometría que generan los encoders de los motores de tracción del robot, tanto la posición como la velocidad estimada a partir de ellos. También entrega las medidas del sensor de voltaje de la batería y permite enviar comandos de movimiento a la base.

Para todos esos sensores ofrece un servicio de suscripción directa, que consiste en que el cliente, una vez conectado, se suscribe a un determinado sensor y el servidor le

entrega datos de ese sensor cada vez que haya medidas nuevas. Los clientes pueden suscribirse exclusivamente a los sensores de su interés por separado, discrecionalmente, y de-suscribirse a voluntad en cualquier momento.
Más en concreto, a través del servidor *otos*, comandaremos los valores de velocidad lineal y de giro del robot para poder navegar por el entorno.

3.2.2. Servidor *Oculo*

El servidor *oculo* auna las funciones asociadas a la unidad pantilt y a la cámara. Con ello permite a los clientes mover la unidad pantilt a voluntad especificándole ángulos objetivo y pone a disposición de los clientes el flujo de imágenes obtenidas con la cámara. Estas se ofrecen a los clientes a través de un servicio de *imágenes bajo demanda*. Se da soporte a imágenes en niveles de gris e imágenes RGB.

3.3. Biblioteca SUSAN

En nuestro proyecto necesitamos extraer información del flujo de imágenes y para ello hemos utilizado la biblioteca SUSAN [3] y unos filtros de color ad-hoc.

SUSAN es el acrónimo de *Smallest Univalue Segment Assimilating Nucleu*. Esta biblioteca ofrece 6 funciones de las cuales hemos utilizado la búsqueda de bordes. Los bordes de una imagen se pueden definir como transiciones entre dos niveles de gris significativamente distintos. Éstos suministran una valiosa información sobre las fronteras de los objetos.

El algoritmo de detección de bordes que está implementado toma una imagen y, con una máscara predeterminada centrada en cada pixel, aplica un sistema de reglas. Si el brillo de cada pixel dentro de una máscara se compara con el brillo del núcleo de esa máscara entonces se puede definir que el área de ésta tiene el mismo o similar brillo que el núcleo.

El concepto de que cada punto de la imagen se asocia a él un área local de brillo similar es la base para el principio de SUSAN. El área SUSAN contiene mucha información sobre la estructura de la imagen. Esta técnica se diferencia de otros métodos en los que es necesario calcular derivadas de la imagen y realizar una reducción inicial del nivel de ruido.

El área SUSAN conlleva información muy importante sobre la estructura de la

[3] http://www.fmrib.ox.ac.uk/ steve/susan/index.html

imagen en la regiones contiguas de cualquier punto. Como se observa en la figura 3.4, el área SUSAN está en un máximo cuando el núcleo se sitúa en una región plana de la superficie de la imagen, ésta baja a la mitad de este máximo muy cerca de un borde, y baja incluso más cuando nos encontramos en una esquina. Esta es la característica del área SUSAN que se utiliza para determinar la presencia de bordes en la imagen.

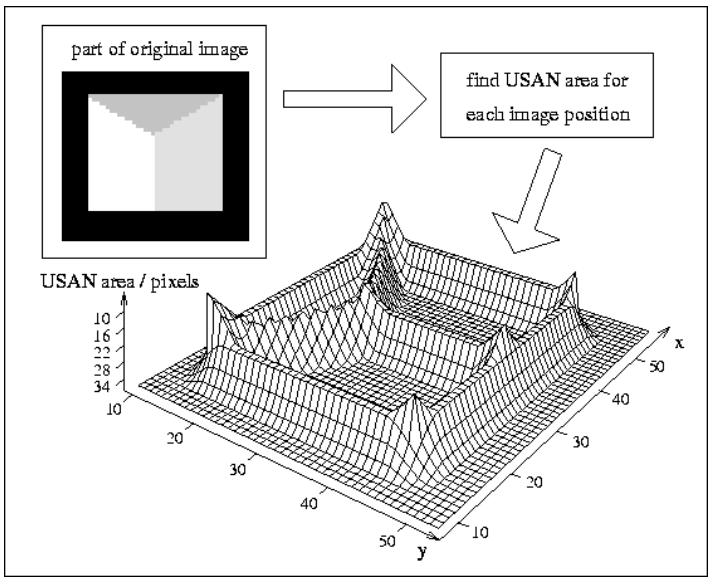


Figura 3.4: Diagrama tridimensional del área SUSAN dado una parte pequeña de una imagen de prueba, demostrando el borde y el realce de la esquina.

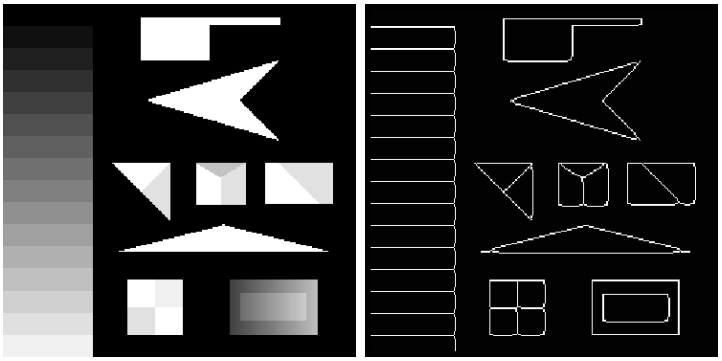


Figura 3.5: SUSAN Ejemplo de funcionamiento

En este proyecto se usan las funcionalidades de la biblioteca SUSAN para obtener datos significativos del entorno del robot. En el capitulo 4 se explica en profundidad el procedimiento que se ha seguido y como se utilizan estas funciones.

3.4. *Player/Stage*

Esta herramienta, en nuestro proyecto, la hemos utilizado para hacer pruebas con el algoritmo de navegación, utilizando medidas obtenidas por el láser, para posteriormente aplicarlo a los datos de profundidad desde visión.

La plataforma *Player/Stage* [Brian P. Gerkey, 2003] es un entorno de simulación de aplicaciones robóticas. Una herramienta de código libre que ayuda y simplifica al desarrollo de algoritmos de control y navegación de robots. El entorno proporciona, un servidor *Player* que ofrece a clientes el acceso a sensores y actuadores, y un simulador de robots *Stage.*

Stage es capaz de simular una población de robots móviles, sensores y objetos del entorno. Todos los sensores y actuadores son accesibles a través de la interfaz estándar de *Player*. Los clientes no notan ninguna diferencia entre los dispositivos reales y su equivalente simulado en *Stage*. El hecho de simular varios robots a la vez, nos va a permitir tener obstáculos dinámicos, que no aparecen en el mapa. Además puede simular cualquier entorno, ya que recibe este como una imagen en formato PNM. La novedad que supone representar el entorno con una imagen, en vez de como una colección de segmentos, es que tenemos libertad para representar entornos irregulares o curvos como se puede observar en la figura 3.6 .

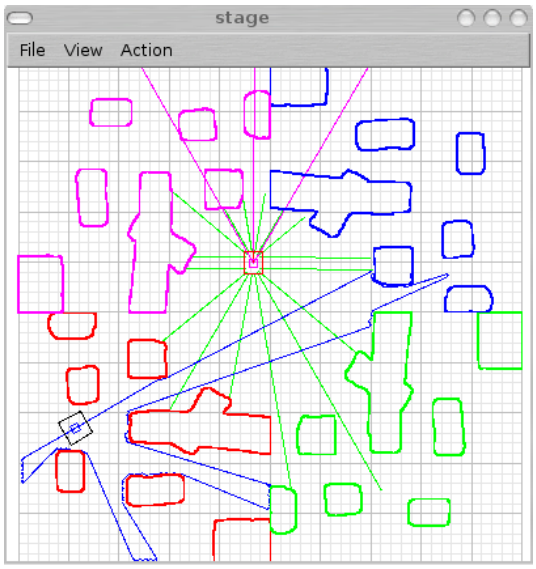


Figura 3.6: *Stage* simulando un mapa con dos robots

Player es un servidor de dispositivos basado en sockets que permite el control de una amplia variedad de sensores y actuadores robóticos. Los clientes se comunican con los dispositivos intercambiando mensajes sobre un socket TCP. Como resultado de esta arquitectura basada en red, *Player* permite a cualquier cliente, localizado en

cualquier lugar de la red, acceder a cualquier dispositivo. Al igual que UNIX, trata los dispositivos como ficheros. Así para leer de los sensores abriría el dispositivo en modo lectura y para comandar ordenes en modo escritura.

3.4.1. Soporte *JDE* para *Player/Stage*

Implementado dentro del servidor *Otos*, una hebra se conecta a *Player* para recibir las medidas sensoriales y comandar las órdenes de actuación. Esta hebra simplemente hace una traducción del formato de mensajes de *Otos*, al formato de mensajes entendido por *Player*. Con esta cualidad el servidor *Otos* se puede conectar indistintamente al robot real, o a un robot simulado.

Capítulo 4

Descripción Informática

En este capítulo se explica el diseño con esquemas que hemos realizado para solucionar el problema planteado, satisfaciendo los objetivos y requisitos propuestos en el capitulo 2 e implementando éste sobre la plataforma descrita en el capitulo anterior. Se comienza con una aproximación a la solución propuesta, para pasar a describir los esquemas que forman parte de la solución desarrollada.

4.1. Diseño General con esquemas

El objetivo de este proyecto es que un robot navegue sin chocar con ningún obstáculo utilizando exclusivamente visión monocular, combinando técnicas de visión computacional con algoritmos de navegación local.

La implementación de este proyecto se ha hecho sobre la plataforma *jde.c* y el lenguaje de programación C. Así, el diseño general para el comportamiento buscado se podría resumir en tres hebras o esquemas *JDE* que iterativamente procesan las imágenes de la cámara y comandan al robot la velocidad o sentido que debe de seguir.

Tres esquemas son los encargados el comportamiento buscado. El primero, esquema *frontera*, se encarga de procesar las imágenes recogidas por la cámara y calcular las estimaciones de distancia del robot a los objetos u obstáculos. Por lo tanto es un esquema perceptivo, que utiliza como entrada las imágenes y ofrece a los demás esquemas los datos de profundidad obtenidos. Cabe destacar que este esquema comanda ordenes al cuello mecánico para ampliar el ángulo de visión de la cámara y así aumentar las estimaciones de profundidad entorno al robot. El segundo esquema perceptivo *pasillo*, se encarga de procesar los datos de profundidad, generados por el esquema anterior, y encontrar entre éstos una estructura geométrica conocida, los pasillos. Partiendo de esta estructura calcula el punto central de ésta más alejado del robot. Utiliza como entrada las estimaciones de profundidad y pone a disposición de los demás esquemas el punto en el centro del pasillo percibido más lejano al robot. Por último, el esquema de

actuación *navegar*, donde está implementado el algoritmo de navegación, se encarga de comandar el movimiento del propio robot con un controlador reactivo en su interior. Utiliza como destino de la navegación el punto central del pasillo y las estimaciones de distancia a los obstáculos. Actualiza las velocidades de translación y de rotación del robot.

La activación de estos tres esquemas la realiza otro esquema jerárquicamente superior, esquema *padre*, que encapsula todo este comportamiento. También existe un esquema de servicio *guipolly* que interactúa con el usuario y desde el cual se activan todas las funcionalidades, además de visualizarse en cada momento la situación del robot y las estructuras de datos de los esquemas anteriores.

El diseño de la aplicación sobre la plataforma *jde.c* se observa en la figura 4.1. Hay tres partes bien definidas. Los esquemas perceptivos, dentro de cuadrados, que recogen, y procesan información de las variables sensoriales(imagen que pone a disposición *jde.c* en $ColorA$), el esquema reactivo, dibujados en círculos, que comanda las ordenes para generar el movimiento del robot,y la plataforma *jde.c* que actualiza las variables sensoriales y materializa las variables de actuación. Las variables que actualizan iterativamente los esquemas implementados son utilizadas por otros para generar el comportamiento deseado.

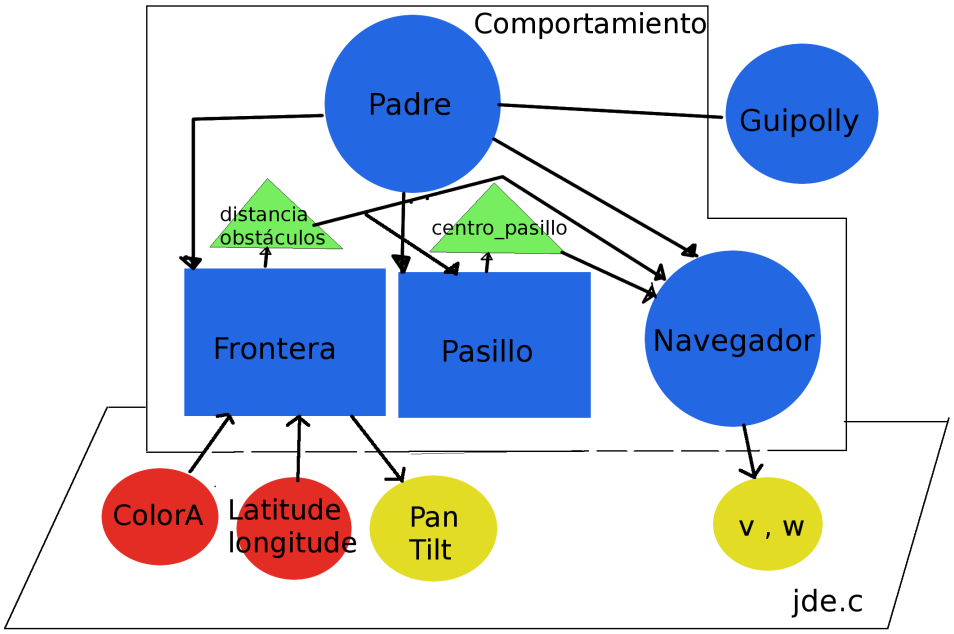


Figura 4.1: Diseño de la aplicación sobre *JDE*.

Una vez resumido el diseño general de la aplicación, se pasa a describir cada uno de los esquemas que componen la aplicación. Detalladamente, se exponen las técnicas utilizadas y su fundamento teórico.

4.2. Esquema perceptivo *Frontera*

Este esquema perceptivo se encarga de calcular las estimaciones de distancia a los obstáculos alrededor del robot. Para ello utiliza como entrada las imágenes captadas por la cámara, que pone a disposición la plataforma *jde.c*. Como salida, ofrece a los demás esquemas los datos de profundidad obtenidos.

El trabajo de este esquema se divide en dos partes: en una primera fase de procesamiento monocular, se haya la distancia a los objetos en el campo visual de la cámara, apoyándose en la *hipótesis de suelo* en la que se asume que la parte baja de la imagen es suelo. En la segunda fase, de fusión y composición, se rota la cámara con el cuello mecánico, y se van fusionando las estimaciones de distancia alrededor del robot.

Es un esquema en el que se realizan múltiples cálculos geométricos necesarios para la traducción de puntos 2D de la imagen a puntos 3D del entorno del robot. Realiza 5 iteraciones por segundo en las que se actualizan las mediciones. Dentro de los ficheros *frontera.c* y *frontera.h* se encuentra el código cuya funcionalidad está representada en la siguiente figura 4.2.

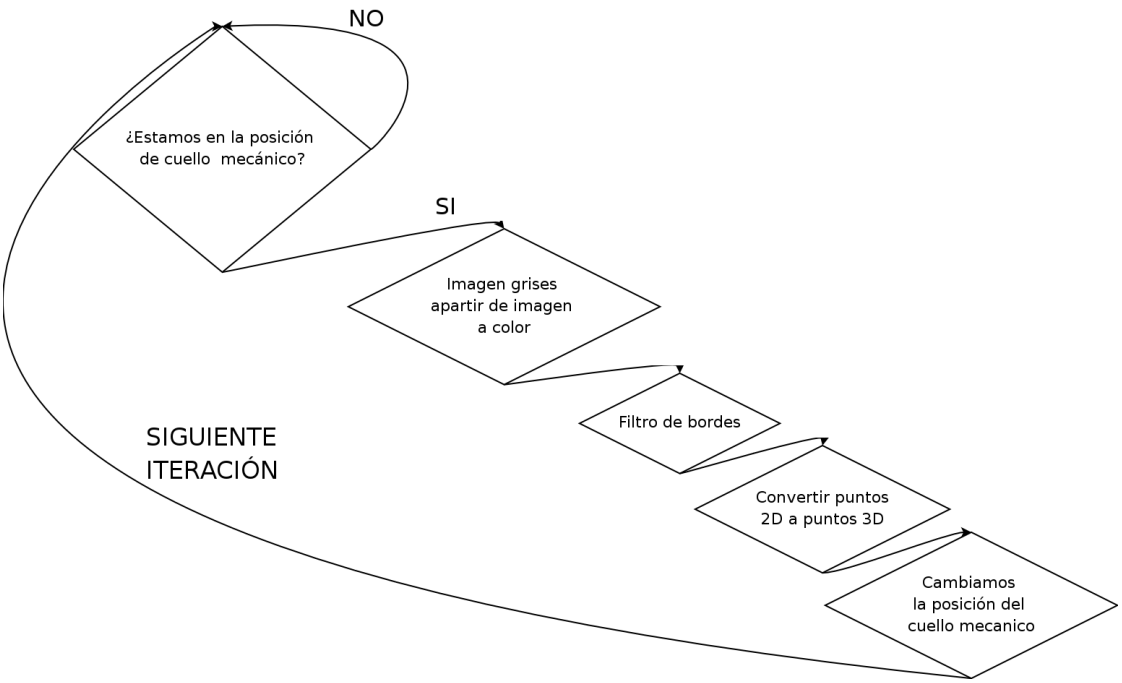


Figura 4.2: Pseudocódigo del esquema frontera.

4.2.1. Procesamiento monocular

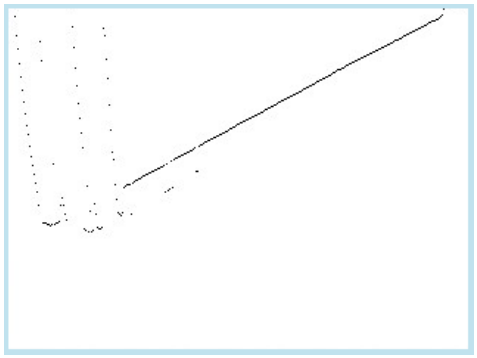
Se convierte la imagen RGB *colorA*, captada por la cámara, a escala de grises. Este proceso es necesario para operaciones posteriores sobre la imagen. Se realiza de la forma siguiente: recorriendo la imagen de arriba abajo y de izquierda a derecha, obteniendo de los tres valores RGB de cada pixel un único valor mediante la expresión 4.2:

$$n = Numero\_columnas * Numero\_filas * 3 \quad (4.1)$$

$$img\_grises[j]_{j=0}^{n/3} = (img\_color[i] + img\_color[i+1] + img\_color[i+2]/3)_{i=1}^{n} \quad (4.2)$$

y el resultado es el que se observa en la figura 4.3(b)

(a) (b)



(c)

Figura 4.3: a) Imagen en color b) Imagen en escala de grises. c) Imagen una vez filtrada

Una vez está la imagen en escala de grises se pasa ésta por un filtro de bordes. La librería SUSAN pone a nuestra disposición esta función para realizar el filtrado.

```
void i2onlyedges(unsigned char *image);
```

Pasándole la imagen en la escala de grises nos devuelve ésta binarizada, donde los bordes encontrados se pintan en color negro y los demás en color blanco. En la figura 4.3(c) se puede observar la imagen que se obtiene una vez filtrada.

Ahora bien, una vez detectada la discontinuidad del suelo con los objetos podemos calcular en dimensiones de la imagen estos puntos discontinuos. Pero estos datos, en sí, no nos sirven para estimar la distancia del robot a los obstáculos. Tenemos que apoyarnos en un modelo de cámara.

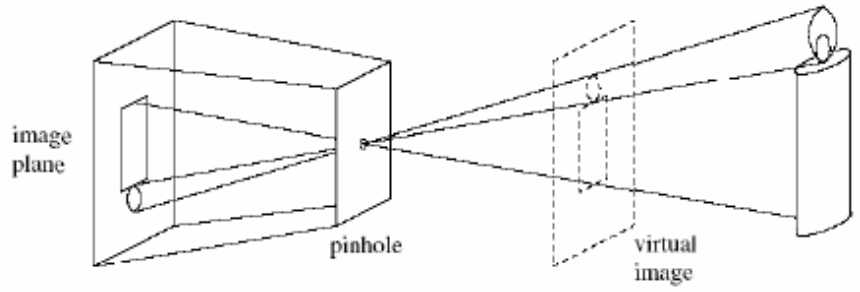


Figura 4.4: Modelo de cámara *pin-hole*.

El modelo de cámara *pin-hole* es uno de los más sencillos que podemos plantearnos, ya que modela razonablemente bien una cámara común con una formulación matemática muy simple.

El modelo de cámara *pin-hole* supone que para todo punto que impacta en el plano de proyección, el rayo de luz que sale rebotado de un cuerpo y llega a la cámara, atravesando el centro óptico, independientemente del punto de origen y del punto de impacto en el plano de proyección. Esto hace el modelo matemático de la cámara muy simple, ya que de una aplicación directa del teorema de Thales, podemos sacar las ecuaciones:

$$Zx = -fX \tag{4.3}$$

$$Zy = -fY \tag{4.4}$$

suponiendo que:

- Las coordenadas del punto en el espacio que se proyecta serán $(X, Y, Z)^T$, donde el eje de la coordenada Z es la línea que pasa por el centro óptico y es perpendicular al plano de proyección, y las coordenadas $(0, 0, 0)$ están situadas en el centro óptico.
- Las coordenadas del punto proyectado serán $(x, y)^T$.

- f es la distancia focal -es decir, la distancia que separa el centro óptico del plano de proyección.

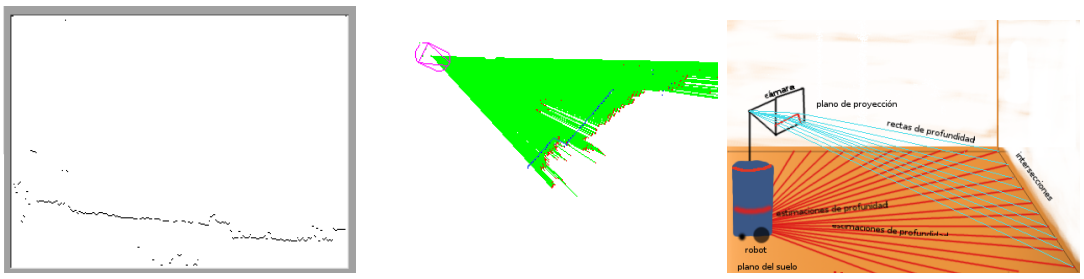
Tanto la apertura horizontal y vertical de la cámara, como la distancia focal son los datos que nos han sido necesarios para, basándonos en este modelo, calcular la posición3D de los puntos de la imagen sobre el plano de proyección, solidarios éstos al centro óptico de la cámara. Ésto se realiza para cada punto con la siguiente función

```
void pixel2D2punto3D(int fronterarefimg,tpunto3d *pto_planoproyeccion,int i)
```

Una vez tenemos calculados los puntos del plano de proyección tenemos que traducirlos a distancias referentes al robot. Es ahora el momento de utilizar la que hemos llamado *hipótesis de suelo*. Esta se basa en que la cámara, situada encima del robot, está inclinada enfocando al suelo. Ésto hace que la imagen, observándola de abajo arriba, se componga de suelo y demás objetos. La discontinuidad entre estos es la calculada por el filtro de bordes y la que vamos a traducir a coordinas 3d referentes al robot.

Definimos el plano que contiene el suelo sobre el que esta situado el robot. Los pixels frontera los situados en 3D se unen con el centro óptico. Los rayos que forman se intersectan con el plano del suelo. Es justo donde se acaba el suelo y comienzan los obstáculos. Con esto obtenemos la distancia de los obstáculos al robot, como se puede observar en la figuras 4.5(b) y 4.5(c) en la que de verde tenemos los rayos de profundidad y de rojo las estimaciones de distancia que obtienen. Esta es la función que realiza todo el proceso.

```
void interseccion(tpunto3d centro_optico, tpunto3d pto_planoproyeccion ,
                  tplano plano, tpunto3d *i)
```



(a) (b) (c)

Figura 4.5: Ejemplo de la técnica utilizada. a) imagen obtenida del filtro de bordes b) Vista de los rayos de profundidad y los puntos de discontinuidad en coordenadas 3D. c) Simulación de la técnica utilizada

4.2.2. Fusión y composición

Una vez que ya hemos conseguido la estimación de distancia de los obstáculos al robot queremos ampliar su campo visual. Es necesario para poder elaborar una buena composición de los obstáculos en derredor. Para ampliar el campo visual es necesario apoyarnos en el cuello mecánico. Moviendo éste, sobre el que se sitúa la cámara, realizamos un barrido de 90 grados en ambos lados (180 en total). Definiendo posiciones claves del cuello mecánico tomamos instantáneas de la imagen para obtener las distancias a los obstáculos alrededor del robot. Mientras se van fusionando las estimaciones para tener un conjunto de distancias a objetos del entorno del robot. En la figura 4.6 se observa el funcionamiento.

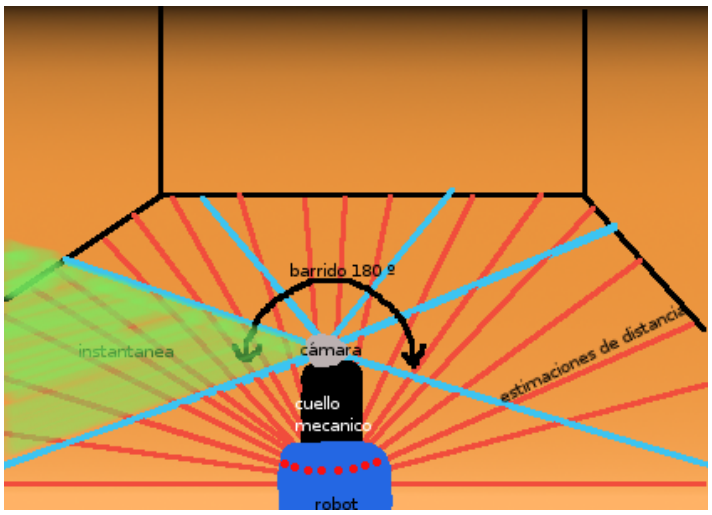


Figura 4.6: Rotación de la cámara sobre el cuello mecánico. El color verde refleja el ejemplo de una captura de la imagen en una posición determinada del cuello mecánico.

Estas estimaciones se ponen a disposición de los demás esquemas en la matriz $profundidad\_derredor[]$ definida por la estructura siguiente

```
typedef struct vision_points{
  float x;
  float y;
  struct timeval time; //marca de tiempo del punto
}Tvision_points;
extern Tvision_points  profundidad_derredor[];
```

4.3. Esquema perceptivo Pasillo

Conseguida la primera meta ahora pasamos a procesar los datos de las estimaciones de distancia.

Este esquema se encarga de segmentar los datos de profundidad recogidos por el esquema anterior. Esta segmentación se hace para determinar si un conjunto de distancias tienen una estructura conocida. Para nosotros esta estructura buscada será la forma rectangular de un pasillo. Delimitado por dos segmentos paralelos y con una distancia definida entre estos.

10 iteraciones por segundo de este esquema sirven para que este proceso sea fluido y rápido. El código del esquema está implementado dentro de los archivos pasillo.c y pasillo.h.

Dividiendo por la funcionalidad este esquema se puede dividir en 3 partes que son:

- 1. Segmentar los datos de profundidad.
- 2. Búsqueda de segmentos paralelos que cumplan ciertas reglas previas.
- 3. Calculo de segmento central o centro de pasillo.

Primero tenemos que segmentar los datos de profundidad. Para esto nos apoyamos en un algoritmo en el que se hipotetizan segmentos[Victor Gómez,2003]. Con esta técnica se calcula la pendiente entre el primer y el ultimo dato de la distancias de los obstáculos al robot, para posteriormente intentar verificar si todos las estimaciones están, con una determinada tolerancia, dentro esta recta hipotetizada. De este modo se calcula la posición hipotética sobre el eje de coordenadas Y para cada punto x del interior basándonos en la ecuación punto-pendiente de la recta mostrada a continuación 4.5. Por la cual podemos calcular el valor de y ($y\_hipotetico$) para un punto si conocemos el valor x para ese punto (x), otro punto de la recta ($x1$,$y1$) donde se encuentra el punto a analizar y el valor de la pendiente ($pendiente\_recta$) para la recta.

$$y\_hipotetico = pendiente\_recta * (x - x1) + y1 \tag{4.5}$$

Una vez calculado este valor se compara con el valor real que tiene ese punto en el eje Y. Si la diferencia entre estos está fuera de un rango predefinido puede asumirse como punto inicial de un posible segmento. Se hace valida la hipótesis de que es un segmento correcto, si todos los puntos que deberían pertenecer a este segmento

están dentro de una franja definida para ese segmento. Entonces el siguiente paso sería analizar el resto de las estimaciones de distancia. Hipotetizando un nuevo segmento que tiene ahora como punto inicial la primera estimación y como punto final el punto inicial del segmento encontrado. Si por el contrario, no fuera un segmento correcto, el algoritmo se repite pero ahora calculando la pendiente entre la estimación que se salió del rango predefinido y el ultimo dato, sabiendo que las estimaciones se guardan en *profundidad_derredor* de izquierda a derecha. Utilizando esta técnica este algoritmo extrae los segmentos de derecha a izquierda. Estas operaciones se repetirán hasta que se han analizado todas las estimaciones de distancia como se muestra en la figura 4.7.

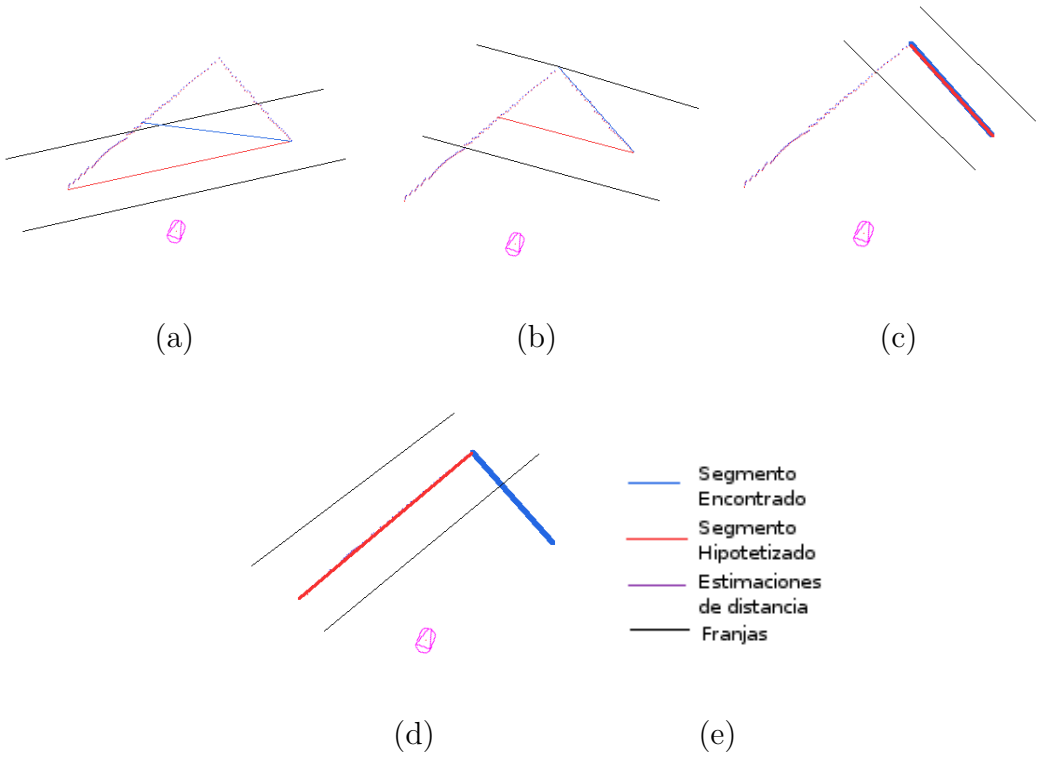


Figura 4.7: a) Comienzo de la segmentación b) Segmento encontrado c) Verificación de hipótesis d) Total de segmentos encontrados e) Leyenda.

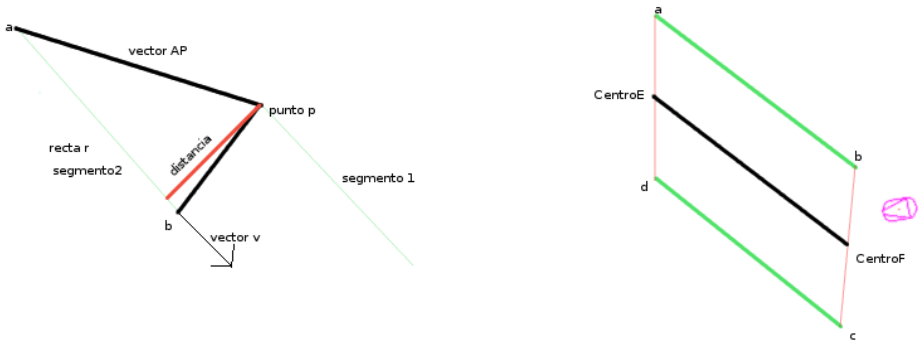
Esta solución para la segmentación es muy rápida y robusta frente a ruidos y pixels aislados en parte porque no calcula pendientes gracias a su banda de tolerancia.

Una vez agrupados los datos de profundidad tenemos que determinar si estos segmentos tienen una distribución adecuada, en concreto en la forma de un pasillo. Asumimos que estos se definen por tener dos largos segmentos paralelos y la distancia entre estos está acotada entre 1 y 3 metros.

Para detectar si los segmentos son paralelos calculamos las pendientes de éstos y comparamos éstas entre si. Si las pendientes son semejantes dentro de un rango definido

previamente hemos encontrado los segmentos buscados. Ahora bien, necesitamos que estos segmentos cumplan que, al ser segmentos que definen un pasillo, la distancia entre éstos esté entre 1 y 3 metros de longitud. Esta distancia se calcula mediante de la proyección ortogonal 4.8(a) del extremo de uno de los segmentos con la recta que contiene al otro segmento.

$$d(P,r) = \frac{|\vec{AP}\, x\, \vec{v}|}{|\vec{v}|} \tag{4.6}$$



(a) (b)

Figura 4.8: a) Calculo de la proyección ortogonal de un punto sobre una recta, b)Calculo del segmento central del pasillo

Cuando se encuentran varios pasillos se selecciona el par de segmentos mas largo, porque así a lo largo de ese pasillo se tiene mas camino libre.

Encontrado el pasillo, sólo nos queda determinar el segmento central paralelo a las paredes de éste.La figura 4.8(b) y la expresión 4.8 muestran el calculo de este segmento.

$$centroE(x,y) = (\frac{(ax-dx)}{2}, \frac{(ay-dy)}{2}) \tag{4.7}$$

$$centroF(x,y) = (\frac{(bx-cx)}{2}, \frac{(by-cy)}{2}) \tag{4.8}$$

Terminado el trabajo que realiza este esquema sólo le queda poner a disposición de los demás, en *centro_pasillo*, la coordenada *centroE* que siempre será el punto más distante al robot por como está implementado el algoritmo de segmentación. Además, podemos asegurar que, al ser el par de segmentos de mayor longitud de los encontrados con los que calculamos el segmento central, la distancia entre *centroE* y *centroF* será la mayor de las posibles estimadas con todos los segmentos paralelos encontrados.

Si se da el caso que no se encuentra pasillo alguno entonces no actualizará la variable *centro_pasillo*, teniendo ésta el valor de la última iteración. No es problema ya que siempre que se determina el centro de un pasillo este es válido en cualquier caso.

4.4. Esquema de actuación Navegador

Este es el esquema que genera el movimiento del robot. Materializa el control reactivo basado en la técnica de campos de fuerzas virtuales o *virtual Force Fields*(VFF)[J. Borenstein, 1989]. Percibe obstáculos con la profundidad calculada por el esquema *frontera* y utiliza como objetivo de navegación el punto central del pasillo calculado por el esquema *pasillo*. Es el encargado de comandar los valores de velocidad de translación y de rotación del robot.

Este algoritmo se basa en fuerzas ficticias entorno al robot. El destino ejerce una fuerza de atracción sobre el éste y los obstáculos también ejercen una fuerza pero repulsiva, que aumenta cuanto más cerca esté del objeto. El robot se orientará con la suma vectorial de la fuerza atractiva con las fuerzas repulsivas, moviéndose en esa dirección. Cuando el robot, viajando hacia el destino, se encuentra un obstáculo lo evita retomando el rumbo al rebasar éste. Es una solución de compromiso entre avanzar hacia el destino y evitar obstáculos.

El algoritmo toma como entrada las coordenadas de *centro_pasillo*, y las estimaciones de distancia $profundidad\_derredor[]$, que serán la fuerza atractiva y las fuerzas repulsivas respectivamente. Como resultado de este esquema se actualizan las variables v velocidad de translación y w velocidad de rotación del robot, que la plataforma *jde.c* lleva a los motores.

El destino del robot será el punto central del pasillo mas alejado de él. A diferencia de otras implementaciones de éste algoritmo, el objetivo será dinámico, cambiando a medida que el robot se mueva, ya en cada iteración del esquema *pasillo* se calcula este punto central. Haciendo ésto que el robot deambule a lo largo del pasillo siguiendo *la zanahoria*. Si en algún momento el robot llega al destino, una vez allí, comienza a rotar.

En cada iteración de este esquema, se calculan las fuerzas repulsivas de los obstáculos percibidos y se suman vectorialmente. Estas fuerzas se calculan como el cociente de una constante de repulsión k entre la distancia del robot al objeto, siendo esta distancia siempre superior a cero.

$$Xrepulsiva = \sum_{i=0}^{i=180} cos(i) * \frac{k}{distancia[i]} \tag{4.9}$$

$$Yrepulsiva = \sum_{i=0}^{i=180} sin(i) * \frac{k}{distancia[i]} \tag{4.10}$$

$$Xresultante = Xatractiva + Xrepulsiva; \tag{4.11}$$

$$Yresultante = Yatractiva + Yrepulsiva; \tag{4.12}$$

La fuerza atractiva viene dada por la coordenada de destino. La suma vectorial de todas las fuerzas proporciona al esquema la dirección a seguir para evitar el obstáculo, y a la vez, seguir avanzando hacia el objetivo.

En el momento que nos encontremos cerca de un obstáculo el sumatorio de las fuerzas repulsivas influye más en la resultante total que la fuerza atractiva. Esto hace que el robot se aparte del obstáculo. Podemos ver un ejemplo en la figura 4.9, donde la linea color rojo es la fuerza atractiva, generada por el destino, la línea amarilla es la fuerza repulsiva, generada por la suma vectorial de la fuerza ejercida por cada obstáculo detectado, y la línea azul que es la resultante de la suma vectorial de las dos anteriores, y hacia donde debe dirigirse el robot.

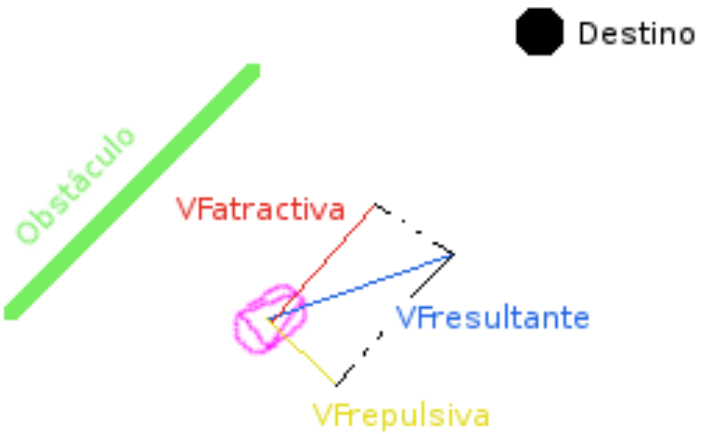


Figura 4.9: Ejemplo de fuerzas virtuales sobre el robot

Una vez se ha obtenido la dirección deseada de avance, se compara el ángulo a viajar y el ángulo actual del robot. La expresión para calcular el ángulo a viajar es la siguiente

$$\theta = atan(Xresulante, Yresultante) \tag{4.13}$$

Si estos anguloso son similares el robot estaría orientado hacia la resultante, y por lo tanto podemos hacer que se mueva linealmente aumentando la velocidad de translación. Si ésto no es así debemos rotar el robot hacia el ángulo a viajar. Dependiendo del cuadrante en el que éste se sitúa, el robot, asignando una velocidad de rotación positiva, gira a la derecha,y negativa a la izquierda.

4.5. Esquema de servicio Guipolly

Programado sobre la biblioteca Xforms la misión de este esquema es ayudar a la depuración de los otros esquemas, para eso muestra por pantalla los datos de las estructuras más relevantes. Permite interactuar con ellos , para activarlos, desactivarlos cambiar algún parámetro, etc.

Se encarga de dibujar la interface gráfica de la aplicación. Funciona como un esquema más de *JDE*, es decir, es una hebra iterativa que se ejecuta 10 veces por segundo, que comprueba el estado de los elementos de la aplicación, activando o desactivando su funcionalidad según se encuentren. La interface de la aplicación esta basada en la proporcionada por la plataforma *JDE*, de la que se ha sustituido algunas funciones innecesarias y se ha dotado de distintos elementos para mostrar toda la funcionalidad de este proyecto.

Este esquema se encarga de dibujar además de la imagen captada por la cámara, la imagen filtro de bordes, las estimaciones de profundidad, las fuerzas generadas por el VFF, los segmentos encontrados y el segmento central del pasillo.

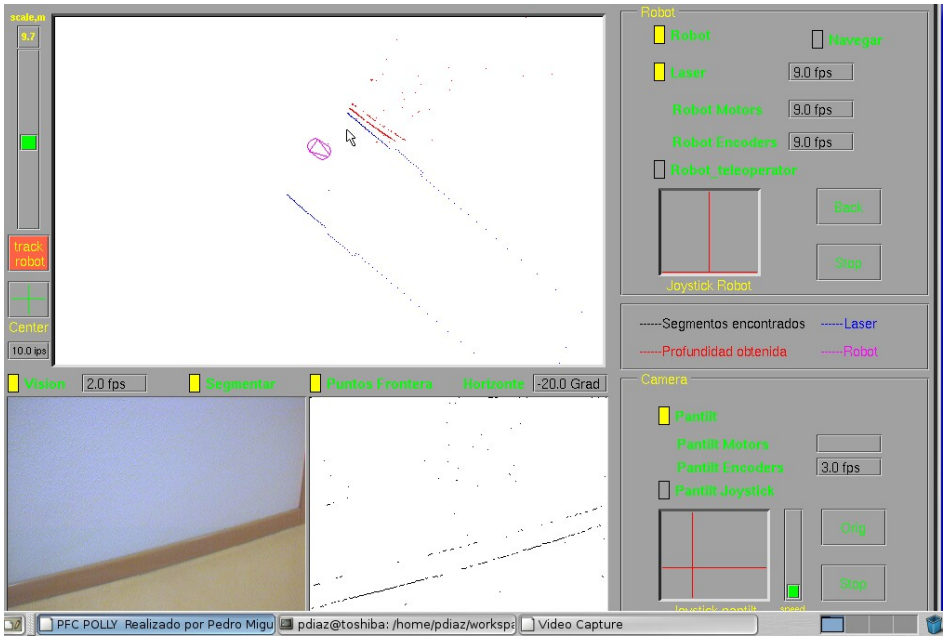


Figura 4.10: Interface de la aplicación

Pulsando el botón adecuado podemos visualizar las imágenes que son recogidas por la cámara, las medidas sensoriales o activar o desactivar el comportamiento programado. Además podemos teleoperar al robot y mover la cámara comandando ordenes al cuello mecánico.

En la figura 4.10 se muestra el aspecto del interface de la aplicación, que podemos dividir por su funcionalidad en:

- Para el procesamiento monocular, en la parte baja del interface se encuentra la zona de visualización de imagen. Se compone de dos *canvas* en los que se plasma la imagen en tiempo real y el filtro de bordes de ésta. Podemos activar o desactivar esta visualización presionando los botones situados encima de las imágenes. Unido a todo ésto varias cajas de texto en las que se muestran datos de velocidad de transmisión de la cámara y ángulo *tilt* en el que esta situada ésta, engloban componen esta zona perceptiva del interface de la aplicación.

- Fusión y composición, utilizando para la visualización de las estimaciones de distancia alrededor del robot una ventana regulable en escala en la que se dibujan tanto el robot como todas las estructuras importantes de los esquemas que estén activados.

- Teleoperación, con el teleoperador para manejar el cuello mecánico y teleoperador del comportamiento del robot.

- Activación o desactivación del comportamiento, mediante el botón de navegación total se activa la navegación visual. Podemos, si se desea, activar y desactivar los esquemas uno por uno utilizando los botones cuyo nombre es igual al esquema que deseamos ejecutar.

Capítulo 5

Resultados Experimentales

En el capítulo anterior hemos descrito la solución al problema de la navegación visual. En éste explicaremos qué pruebas y experimentos hemos realizado para validar y mejorar la implementación en el camino hacia esta solución, según los distintos prototipos desarrollados iterativamente expuestos en el capítulo 2. Comenzamos explicando los experimentos realizados con cada uno de los elementos que entran en juego para la consecución del proyecto para luego detallar una ejecución típica del comportamiento que hemos generado.

5.1. Pruebas de procesamiento monocular

En la primera implementación del algoritmo para el filtro de bordes éste trabajaba en paralelo a un filtro de color. La idea era una vez procesada la imagen y encontrados las discontinuidades asegurar que éstas eran discontinuidades entre el suelo y otros objetos, ya que si la cámara no está lo suficientemente inclinada la *hipótesis de suelo* se rompe. La imagen no estaría compuesta por suelo y demás objetos, que es la primera premisa que se debe cumplir.

El calculo de frontera entre el suelo y los objetos solo con bordes falla en ese caso. Un ejemplo de esta situación anómala podemos verlo en la figura 5.1(dcha) donde la imagen captada por la cámara muestra la pared de un pasillo.

Este filtro de color se basaba en los valores RGB de los pixels de la imagen. Se encargaba de pintar de negro los pixels que corresponden al suelo y de blanco los demás. Para descartar si un pixel de la imagen corresponde al suelo se define un rango de valores para la componente R(nivel de rojo en ese pixel), la componente G (nivel de verde) y la componente B (nivel de azul) respectivamente. Si los niveles del pixel que estamos comprobando están dentro de ese rango podemos decir que es parte del suelo.

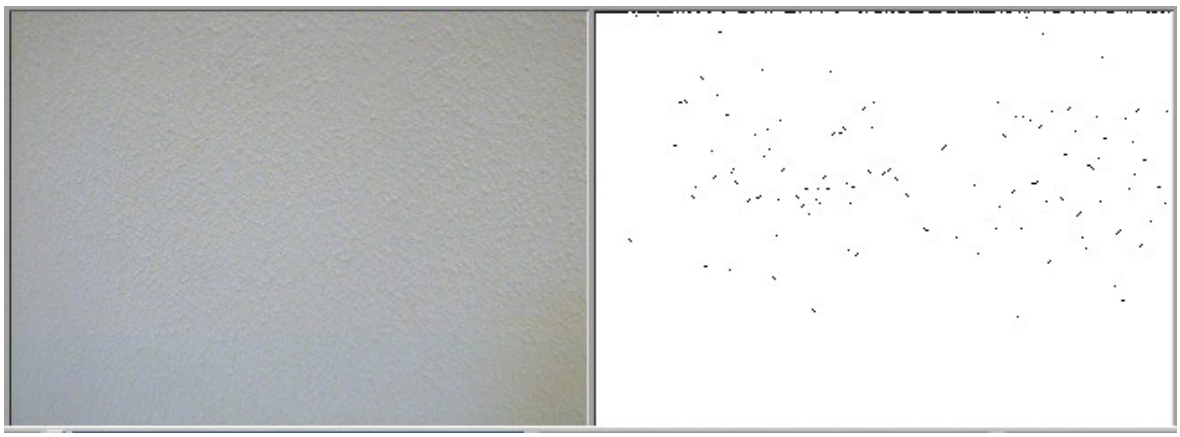


Figura 5.1: Resultado del filtro de bordes sin filtro de color

El problema que surge de este tipo de filtros de color es que para ser robustos a cambios de luminosidad en rgb el filtro debe abrirse y entonces deja de ser discriminante con el suelo. Así sucede que zonas de la imagen cumplen las necesidades de filtrado sin ser éstas correspondientes a pixels que muestran el suelo.

Descartado este tipo de filtro color se realiza una nueva implementación ahora en el espacio HSI 5.2. Donde la componente H, que es el matiz del pixel, o lo que es lo mismo, el color en sí; la componente S, que es la Saturación, es decir, identifica la claridad del color; y la componente I, que es la Intensidad del color o luminosidad. Es más invariante, más robusto a cambios de luminosidad que quedan absorbidos fundamentalmente por la componente I, que se ignora en el filtro, que es exclusivamente HS.

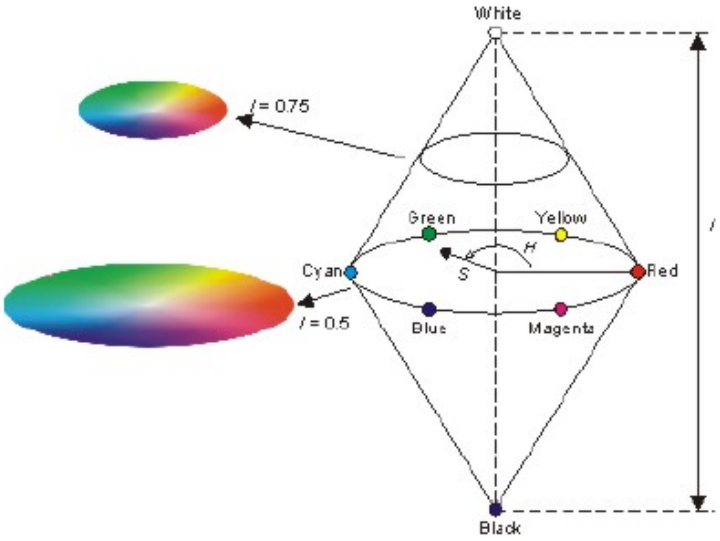
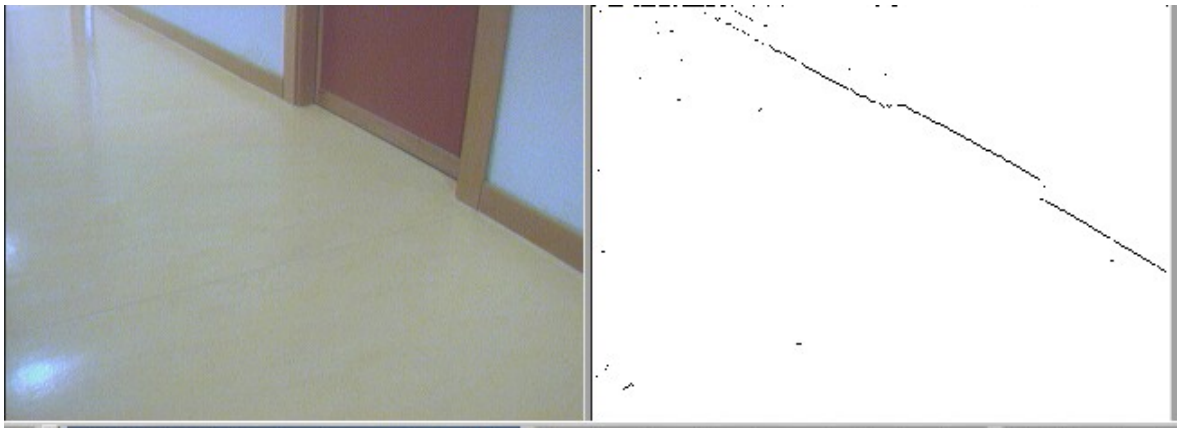


Figura 5.2: Representación del espectro HSI.

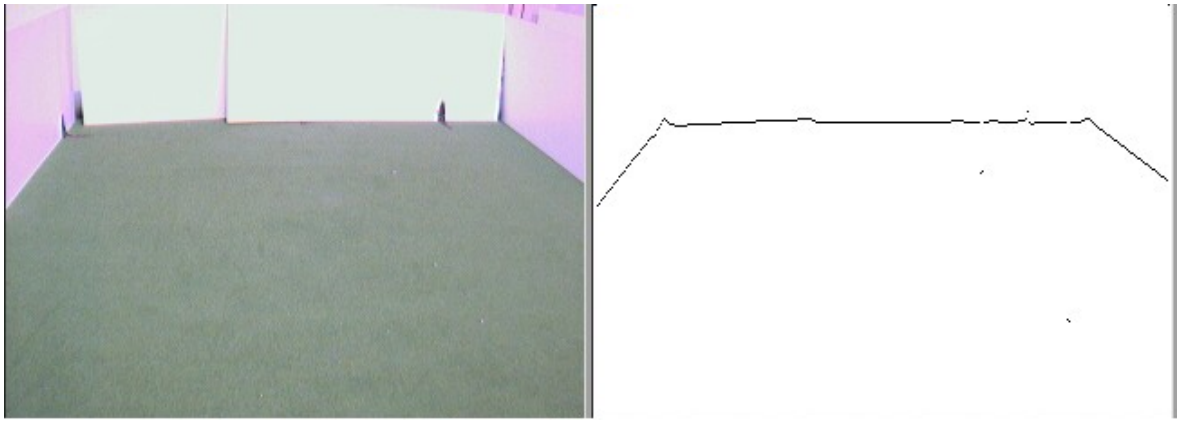
Se observa un mejor funcionamiento que el filtro RGB pero ahora surge un problema

añadido. El tipo de cámara que estamos utilizando posee un *iris* por el cual regula automáticamente el color de la imagen que está captando. Esto hace que para un mismo color del suelo en diferentes posiciones de la cámara las componentes S y H varíen ostensiblemente. Generando ésto mucho ruido en la imagen una vez filtrada.

Como solución a estos problemas se barajaron múltiples ideas pero al final se optó por inclinar la cámara mucho más para que nos asegurásemos que siempre estén compuestas las imágenes por parte de suelo y parte de obstáculos. Con esta inclinación de la cámara se pueden calcular obstáculos en unos metros, más que suficiente para nuestra navegación local. Podemos ver que el filtro funciona correctamente tanto en el pasillo real, en la figura 5.3(b), como en un espacio creado para los experimentos, figura 5.3(a).



(a)



(b)

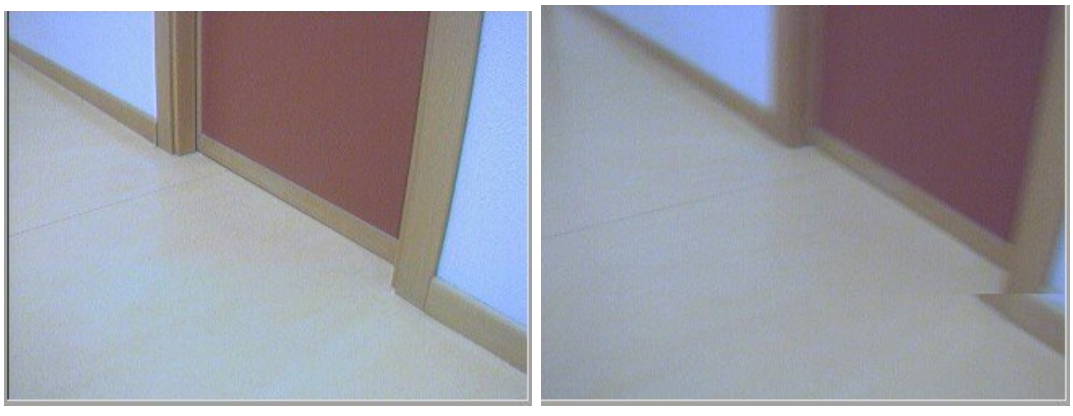
Figura 5.3: a)Filtro de bordes en el pasillo b) filtro de bordes en el *corralillo* .

5.2. Pruebas de fusión y composición

A partir de una imagen captada por la cámara en un momento determinado podemos calcular las estimaciones de profundidad de los obstáculos al robot que caen dentro del campo visual de la cámara. El campo visual de la cámara no es lo suficiente amplio como para determinar mediante una imagen los objetos situados en derredor del robot. Una primera forma de ampliar el campo de visión podría haber sido rotar el robot sobre si mismo para que la cámara tome imágenes del entorno del robot. Esta solución no es la más idónea, ya que el robot necesita estar parado para calcular las estimaciones de distancia a los obstáculos. Por esto se hace más conveniente una segunda opción, situar la cámara encima de un cuello mecánico.

Determinando la posición pan(movimiento horizontal) y tilt(movimiento vertical), y teniendo en cuenta la amplitud horizontal de la cámara, determinamos la posición inicial y final del movimiento del cuello mecánico.

Mediante un barrido continuo en horizontal, que va desde -66 grados a 66 grados, se van fusionando las estimaciones de distancia de los obstáculos entorno al robot. Consiguiendo una "imagen de los alrededores" de 180 grados contando con la amplitud horizontal de la cámara en ambas posiciones extremas.



(a) (b)

Figura 5.4: a) Imagen con barrido no continuo b) Imagen con barrido continuo.

El problema del barrido continuo es que las imágenes tomadas salen borrosas como se observa en la figura 5.4(b), no siendo éstas validas para el calculo de las estimaciones de distancia. Es necesario parar el cuello mecánico un tiempo determinado en unas posiciones definidas previamente en las que se toman instantáneas. Realizamos un

serie de pruebas que determinaron que el tiempo necesario era medio segundo para cada posición. Esto provoca que el barrido completo sea más lento pero asegurándonos que las imágenes que utilizamos para calcular las estimaciones sean correctas y que la imagen de alrededores que se compone se aproxime a realidad, como se puede observar en la figura 5.5.

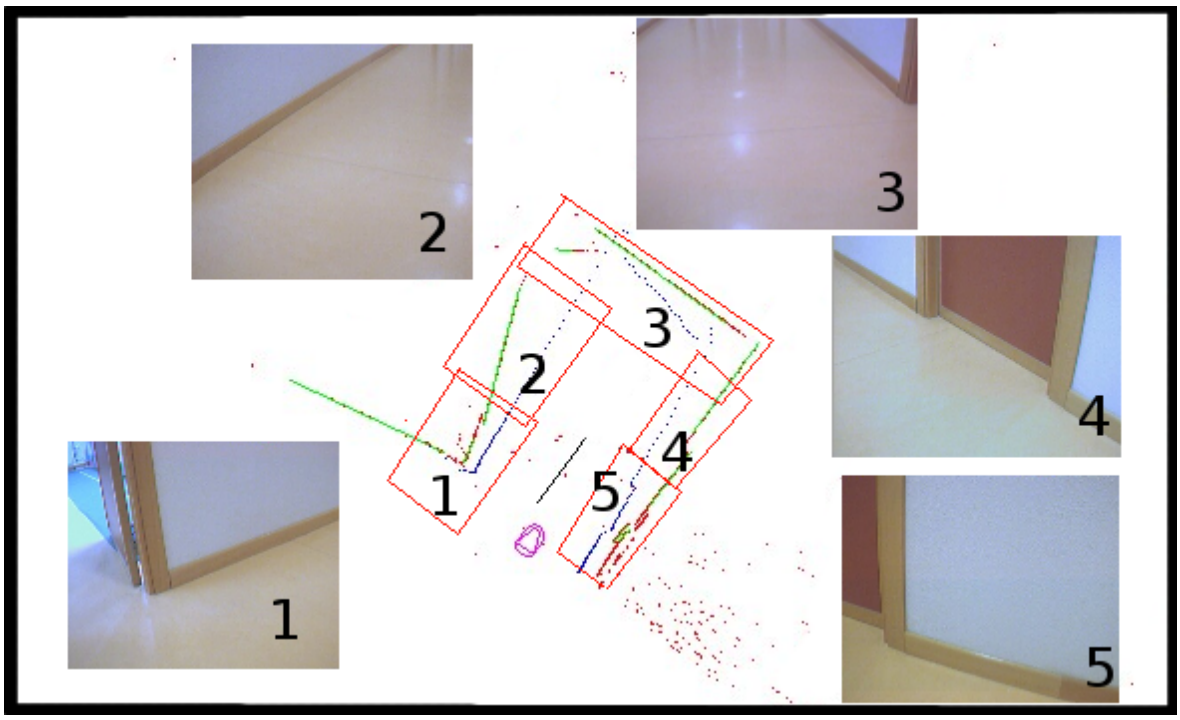


Figura 5.5: Composición de alrededores a partir de las imágenes.

5.3. Experimentos de navegación sobre láser

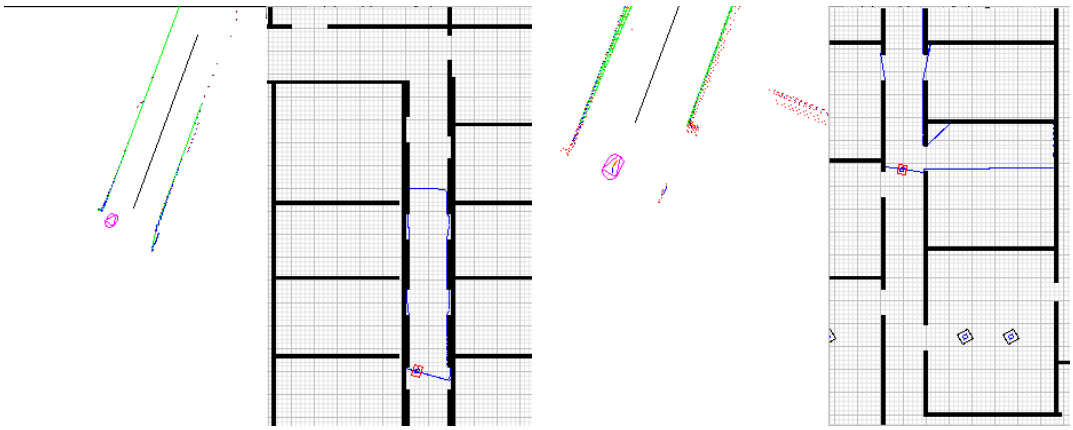
Estos experimentos son para depurar el algoritmo de navegación, el esquema motor y la segmentación de las estimaciones de distancia calculadas. Para ello nos apoyaremos en un principio en el simulador *Player/Stage*. Una vez comprobado que funciona el algoritmo de navegación pasamos a realizar nuevos experimentos ahora en el robot real.

Al no poder utilizar las estimaciones de distancia que hemos calculado anteriormente en el simulador utilizamos para la depuración, tanto en el simulador como en el robot real, los datos recogidos por el sensor láser. Homólogos éstos a los que hemos elaborado desde visión. Si se realiza un algoritmo robusto sobre las distancias que recoge el láser el paso a las estimaciones de distancia recogidas por visión es inmediato.

Nuevamente recordamos que la técnica que utilizamos para la navegación local es la

de campos de fuerzas virtuales VFF. En la que tendremos como destino el punto central mas alejado del pasillo y como elementos repulsivos las estimaciones de distancia a los objetos.

Utilizamos un mapa del departamental en el que simulamos varias situaciones en las que se podría encontrar el robot. Este mapa tiene dos pasillo similares en el que uno de ellos se diferencia del otro en que todas las puertas de éste están cerradas. Observamos que en este experimento el funcionamiento es correcto, el robot se orientaba y mueve hacia el destino. En las figuras 5.6(a) y 5.6(b) dividimos cada subimagen en dos trozos, "representación interna de nuestro robot" en la parte izquierda y simulador en la parte derecha. En el pasillo 5.6(a) el robot deambula a lo largo de toda su extensión por su parte central, girando al llegar a sus extremos. Por el contrario, en el pasillo 5.6(b) al tener todas sus puertas abiertas en algunos casos el robot entra por éstas o giraba a mitad del pasillo. Ésto es debido a la dificultad para el esquema *pasillo* de calcular el punto central de un pasillo en el que hay múltiples segmentos paralelos generados por la apertura de las puertas. La linea de color negro es el segmento que marca el centro del pasillo, y el punto de este segmento más alejado del robot es el destino de la navegación.



(a) (b)

Figura 5.6: a) Navegación con puertas cerradas b) Navegación con puertas abiertas.

Otra de las pruebas que simulamos fue como respondería el robot en una esquina entre dos pasillos. Como podemos observar en la figura 5.7 a media que el robot sale de un pasillo detecta el otro pasillo, orientándose hacia éste y navegando por él.

Como prueba definitiva de la calidad de la navegación implementada situamos 3 cajas en el camino del robot para ver cómo solventaba esta situación. Como se observa

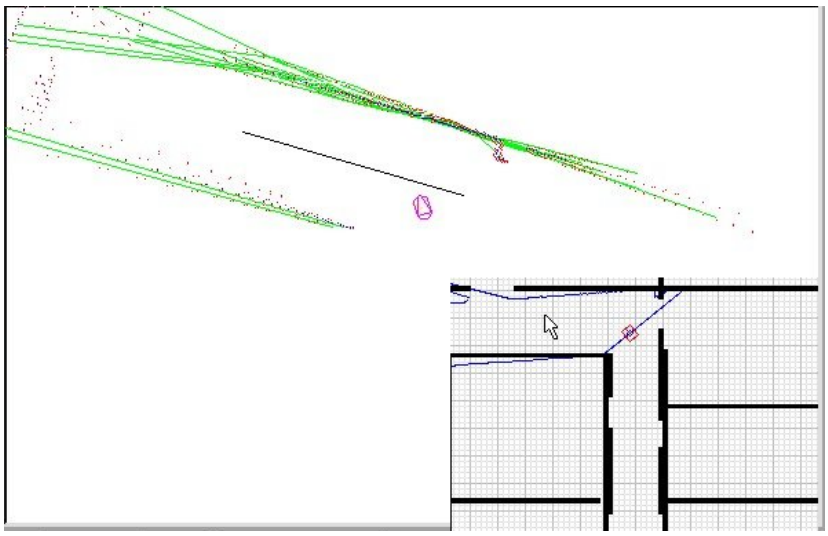


Figura 5.7: Comportamiento del algoritmo en una esquina del pasillo.

en la figura 5.8 estas se sitúan cerca del centro del pasillo a ambos lados. En este punto se visualiza perfectamente el funcionamiento de la técnica de campos de fuerzas virtuales. A media que el robot se acerca al obstáculo la resultante entre el destino y las fuerzas repulsivas hacen que el robot se limite a rebasar el objeto pero a la vez acercase a su destino. La cuestión de poner varias cajas en ambos lados es para demostrar que el algoritmo funciona para objetos situados en cualquier punto delante del robot.

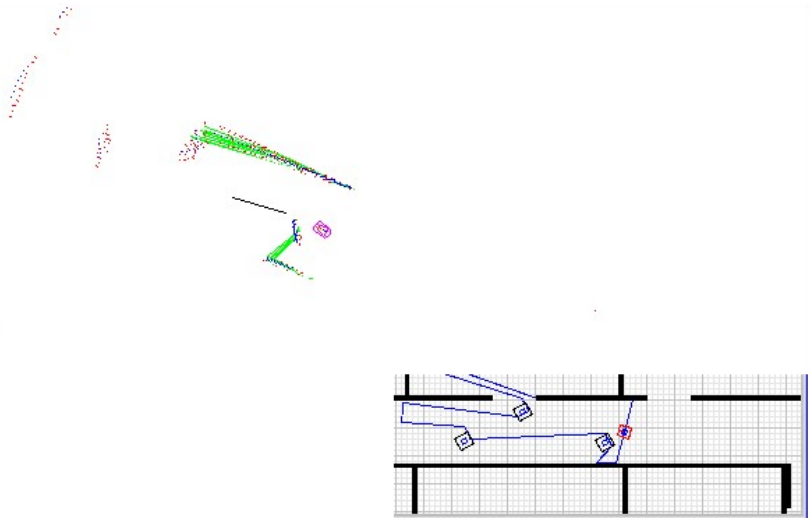


Figura 5.8: Situación en la que tres objetos se interponen en el camino del robot.

Con todos estos experimentos sobre el simulador llegamos a la conclusión de que ya se podían realizar las pruebas en el robot real utilizando el láser como sensor de distancias.

La primera prueba sobre el robot real consistía en situar éste en una pequeña habitación que simulábamos situando tableros que formaban un rectángulo de 4 metros

de largo por 2 de ancho como el que se observa en la figura 5.9. Colocamos el robot en una esquina mirando hacia uno de los tableros para ver como reaccionaba al iniciar el comportamiento. El robot comienza a girar en el sitio esperando a que se encuentre un destino de la navegación. En el momento en el que tiene un objetivo comienza a moverse hacia éste. Al llegar a la pared del rectángulo las fuerzas resultantes hacen que el robot gire, generándose un nuevo destino. Deambulando por lo tanto a lo largo del rectángulo por su parte central, como lo haría si fuera un pasillo de las mismas dimensiones.

Figura 5.9: Simulación de un pasillo de pequeñas dimensiones.

Una vez comprobado que el comportamiento era correcto, cambiamos las condiciones del rectángulo. Pero no antes de iniciarse el comportamiento si no a lo largo de éste. Cabe esperar que el robot que antes recorría el rectángulo por el centro ahora al aumentar el ancho éste, lo siga haciendo. Efectivamente, así es, demostrando que el comportamiento generado es sensible a cambios dinámicos no solo de objetos en movimiento si no en la anchura del pasillo en el que se encuentra.

Figura 5.10: Navegación por el pasillo sin obstáculos.

En una nueva prueba para el robot real colocamos éste en el pasillo del departa-

mental. Libre de obstáculos tantos dinámicos como estáticos. Observamos que el robot deambulaba a lo largo del pasillo y se orientaba perfectamente con el centro de éste, como se pude observar en la figura 5.10.

Aumentando la dificultad, como último experimento con estimaciones de distancia láser colocamos un obstáculo, una papelera. Situada ésta próxima al centro del pasillo observamos que el robot no choca con la papelera y además, si hay hueco entre ésta y la pared, la rebasa como podemos ver en la figura 5.11 tomada en el momento de realizar esta prueba.

Figura 5.11: Navegación por el pasillo con obstáculos.

Para todos los experimentos hemos visto que la velocidad de muestro del láser es muy rápida respecto de la del movimiento del robot. La "imagen de obstáculos" generada por el láser no tiene discontinuidades significativas aunque el robot se esté moviendo, por eso no afecta a la segmentación de las estimaciones de distancia.

Estos experimentos nos han demostrado que el algoritmo de navegación funciona correctamente. Es ahora el momento de sustituir las estimaciones láser por las estimaciones de distancia a los objetos calculadas desde visión.

5.4. Ejecución típica del comportamiento

Una vez realizadas todas las pruebas de los componentes ahora pasamos a probar el sistema completo y sobre el robot real.

Para comenzar los esquemas comandan al cuello mecánico la posición exacta en la cual se va a tomar la primera imagen. Una vez en esa posición, en la que la cámara está inclinada y mirando hacia su izquierda se recoge la imagen y se procesa. Al procesar la imagen obtenemos las primeras estimaciones de distancia de los obstáculos que se encuentran a la izquierda del robot. En la figura 5.12 vemos como corresponden las discontinuidades del suelo recogidas en la imagen, dentro del cuadrado rojo, con las distancias a los obstáculos calculadas. Una vez procesada la imagen pasamos a tomar la siguiente.

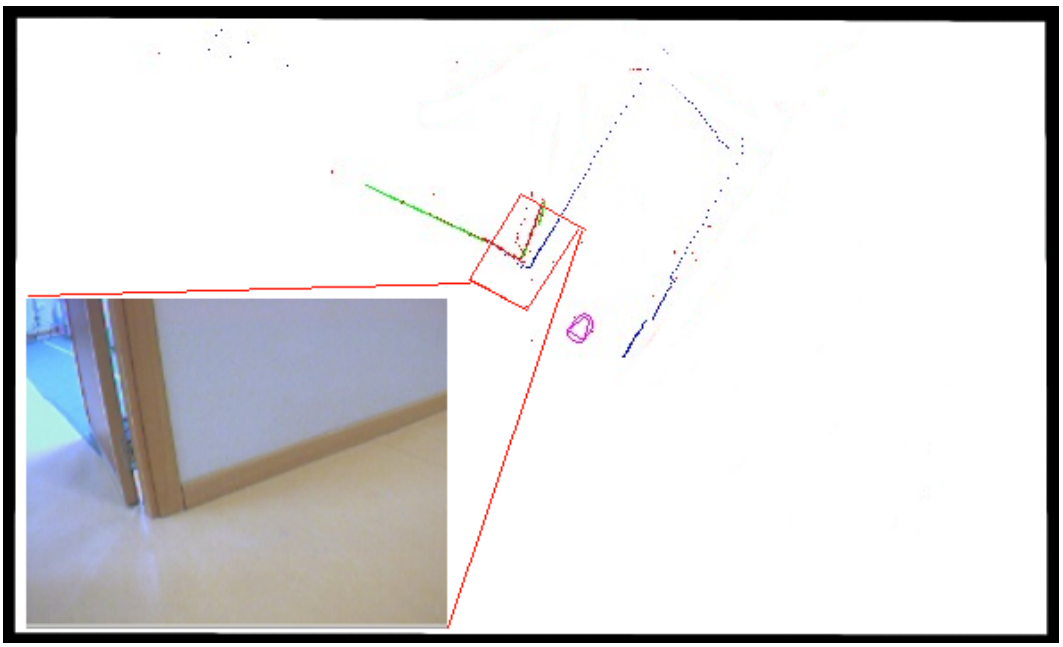


Figura 5.12:

El cuello mecánico es situado situado en el siguiente posición de barrido en la que nuevamente se toma la instantánea de la imagen. Esta nueva imagen recoge los obstáculos que están delante en la parte izquierda del robot, como podemos observar en la figura 5.13. Calculadas las nuevas estimaciones se fusionan con las anteriores. Recordamos que para que la imagen no esté borrosa en cada posición del cuello mecánico esperamos un tiempo determinado.

Cabe destacar que como la cámara está inclinada mirando hacia el suelo no podemos determinar si hay obstáculos delante del robot que queden por encima del campo visual de la cámara. Damos entonces por hecho que más allá de lo que ve la cámara todo es

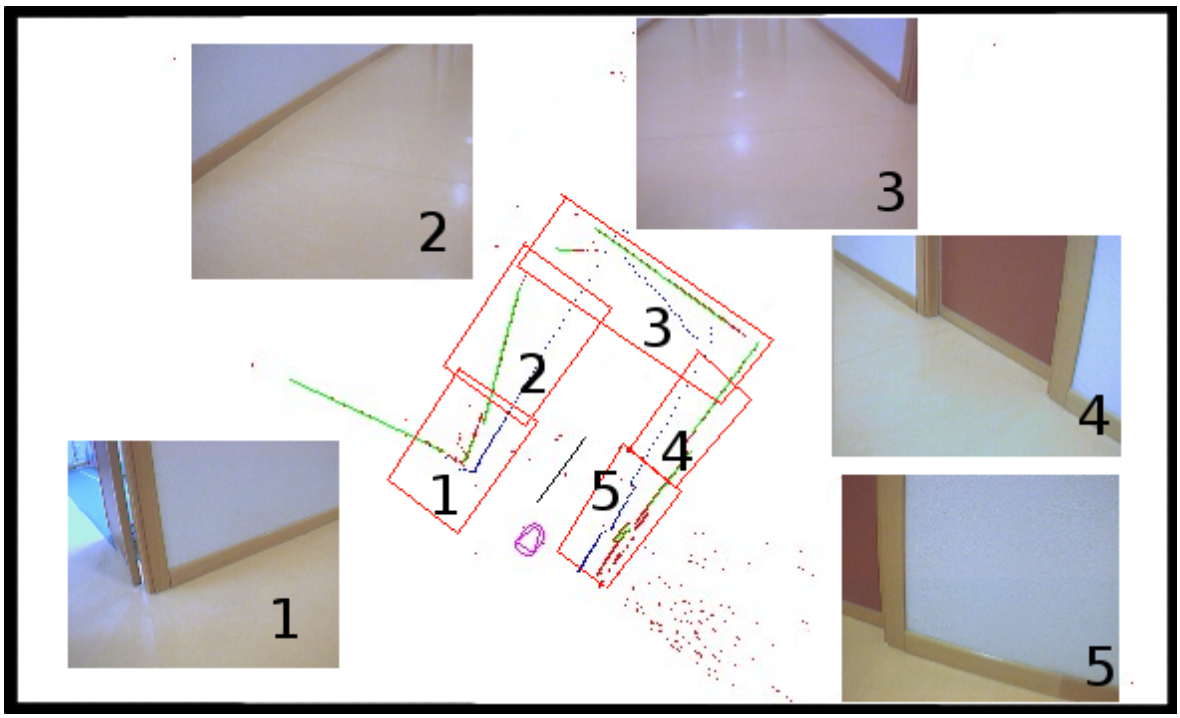


Figura 5.13: Composición de alrededores a partir de las imágenes.

un obstáculo. Por eso se observa en la figura 5.13 que la estimación número 3 el robot ha detectado delante de él, a lo lejos, una pared. A medida que se vaya acercando a ésta posible pared y se vayan tomando nuevas estimaciones se verá si puede ser o no un obstáculo.

Una a una se van calculando las estimaciones de distancia en derredor gracias al barrido del cuello mecánico. Concurrentemente a este barrido el algoritmo de segmentación agrupa las estimaciones para buscar segmentos paralelos que asemejen a la forma de un pasillo. En la figura 5.5 dibujada con una linea negra vemos perfectamente como se ha encontrada el segmento central del pasillo. En este momento el robot comienza a moverse en dirección al extremo del segmento central que está mas alejado del robot.

Las nuevas estimaciones de profundidad se realizan concurrentemente a la navegación del robot. Esta navegación no es muy rápida para que las imágenes tomadas no salgan borrosas.

El robot navega por el pasillo por su parte central. Nuevas estimaciones de profundidad hacen que el destino de la navegación cambie a medida que nos vamos acercando. Si nos encontramos un obstáculo el algoritmo de navegación determina la orientación que tendrá el robot para no chocar con éste o rebasarlo si fuera el caso. Si el robot llega al destino gira poco a poco en el sitio esperando que con nuevas estimaciones se encuentre de nuevo otro destino para viajar hacia él.

Capítulo 6

Conclusiones y Trabajos Futuros

Una vez descrita la solución propuesta para este comportamiento y comentados algunos de los experimentos más relevantes, terminamos esta memoria exponiendo las conclusiones sacadas de este proyecto y las posibles lineas futuras en las que se puede investigar.

6.1. Conclusiones

Haciendo un repaso de los objetivos marcados en el capítulo 2, vamos a describir en que medida se han cumplido éstos satisfaciendo los requisitos también expuestos en ese capítulo. El objetivo principal se ha cumplido con éxito, el robot es capaz de deambular por los pasillos del departamental de la U.R.J.C. utilizando únicamente visión monocular.

El objetivo global estaba articulado en 3 etapas principales:(1)diseño e implementación de un filtro de bordes para discriminar en la imagen entre suelo y pared,(2)la obtención de estimación de profundidad de los obstáculos cercanos fusionando en una representación más amplia que el campo visual de la cámara y (3) finalmente, un algoritmo de navegación local para el robot real que use estas estimaciones. Podemos decir para cada etapa los conclusiones que hemos obtenido.

El primer subobjetivo era el desarrollo de un algoritmo capaz de obtener los bordes de una imagen de manera robusta. Esto se ha descrito con detalle en el capítulo 4. Los experimentos nos han demostrado lo complicado que es ajustar este algoritmo. Viéndose afectado por las condiciones de luz y la variedad cromática del entorno. Incluso la idea de fusionar el algoritmo de bordes con un filtro de color, queda descartada ya que las variaciones de luz generan múltiples pixels *borde* erróneos. Como solución a estos problemas ajustamos el algoritmo a unas condiciones de luz relativamente con-

troladas reduciendo el número de escenarios en los que el comportamiento funciona correctamente.

El segundo subobjetivo era el de obtener las estimaciones de distancia a los obstáculos alrededor del robot y segmentar estos datos. Esta parte es el núcleo fundamental entorno al cual gira toda problemática del proyecto. De nuevo en el capítulo 4 explicamos como se ha conseguido , que no ha sido más que apoyándonos en la que hemos llamado *hipótesis suelo*. Utilizando el modelo de cámara *pin-hole* se intersectan los rayos de profundidad de la frontera suelo-pared en la imagen con el plano del suelo. Siendo definidos estos rayos como la rectas que pasan por el centro óptico y atraviesan los pixels frontera.Con eso conseguimos una estimación de profundidad para los obstáculos en el campo visual de la cámara. Gracias a que ésta estaba situada encima del cuello mecánico hemos conseguido ampliar el campo visual fusionando las estimaciones de profundidad al rededor del robot. El algoritmo para la segmentación de estos datos se fundamenta en hipotetizar segmentos para luego comprobar si estos son correctos. Una vez agrupados los datos se busca en el conjunto de segmentos una estructura conocida, la del pasillo.

El tercer subobjetivo era realizar un algoritmo de navegación local que relaciona la información anterior con las ordenes continuas a los motores para materializar el movimiento adecuados, y probar éste sobre un simulador. Para ello se ha implementado el algoritmo de navegación local VFF [J. Borenstein, 1989]. Tal y como se vio en el capítulo 4 de la memoria, se ha programado este algoritmo en un esquema, que se encarga de llevar al robot lejos de los obstáculos a la par que avanza hacia el destino. Para poder hacer pruebas sobre el simulador se utilizaron como estimaciones de distancia las obtenidas por el láser, ya que *Player/Stage* no simula todavía imágenes de cámaras. Se comprobó que el algoritmo funciona correctamente y que es susceptible a objetos dinámicos que aparezcan en el campo visual del robot.

El último subobjetivo era llevar el algoritmo de navegación local al robot real. Es aquí donde entra la dificultad de trabajar con el robot en un entorno real. Primero se comprobó que el algoritmo de navegación local que se había probado en el simulador funcionaba en el mundo real con la misma fiabilidad, utilizando, eso si, las estimaciones de distancia tomadas por el láser. En el momento que se sustituyeron éstas por las estimaciones calculadas por visión el resultado no era el mismo, no era tan robusto y fiable. No por que este algoritmo de navegación estuviera mal implementado si no porque las condiciones de un entorno real afectan en gran modo al comportamiento del robot. Al ser estas estimaciones calculadas desde visión, los puntos débiles de esta técnica se derivan en el comportamiento final del robot. Como anteriormente dijimos

controlando las condiciones del entorno el comportamiento se acerca en gran medida a lo que se intentaba conseguir. Como conclusion global, destacamos lo complicado que es trabajar sobre el robot real. Es muy costoso de ajustar un algoritmo para un entorno relativamente grande ya que las condiciones de este varian susceptiblemente. Una muy buena implementación teórica del comportamiento puede no funcionar correctamente sobre el robot real. Esto hace que las pruebas y experimentos conlleven mucho tiempo. Incluso ajustes en un mismo entorno de un día para otro pueden no ser los correctos. Al ser un comportamiento basado en visión las condiciones de luz afectan ostensiblemente. En un entorno real son poco predecibles, pudiendo estas cambiar totalmente el funcionamiento del robot.

Ha sido generada la navegación local para el robot Pioneer y se ha desarrollado el comportamiento íntegramente en la plataforma *jde.c*, en el que como requisitos importantes debía de implementarse el proyecto mediante esquemas, y ser éstos programados en el lenguaje C.

6.2. Lineas Futuras

En esta sección se detallan algunas posibles mejoras que se podrían realizar sobre este proyecto y que sirven como nuevas lineas de investigación para otros proyectos fin de carrera.

- Utilizar ésta técnica de navegación local unida a una navegación global [Raúl Isado,2005]. Generando un camino a priori con un mapa del entorno y utilizando la cámara como sensor de objetos no contenidos en el mapa. El robot realizaría su recorrido predeterminado y en el momento que en el ángulo de visión de la cámara entrara un objeto que no estaba definido en el mapa, se recalcule dinámicamente el camino a seguir hasta rebasar este.

- Generación de mapas del entorno utilizando visión. Basándonos en la *hipótesis de suelo* que hemos utilizado para este proyecto y utilizando como apoyo un filtro de color. Ampliando el barrido del cuello mecánico al máximo posible para, estando parado el robot, obtener un mapa del lugar en que se encuentra. Desplazándose el robot posteriormente a zonas seguras dentro del mapa calculado para, tomando nuevas estimaciones ir aumentando el área de conocimiento del entorno.

- Navegar usando dos cámaras encima del cuello mecánico. Pudiendo estimar distancias a objetos que no se encuentren situados en el suelo. Detectar obstáculos

que se mueven por encima, delante del robot. Incluso generar un mapa 3D del entorno.

Bibliografía

[Cañas03] CAÑAS, J.M.: *"Jerarquía Dinámica de Esquemas para la generación de comportamiento autónomo."*, Tesis Doctoral, Universidad Politécnica de Madrid, 2003.

[Cañas03b] CAÑAS, J.M., MATELLÁN, V.: *"Manual de programación para el robot Pioneer."*, Informe Técnico del Gsyc, 2003.

[Cañas04] CAÑAS, J.M.: *"Manual de programación de robots con JDE."*, Informe Técnico del Gsyc, 2004.

[Brian P. Gerkey, 2003] Andrew Howard Brian P. Gerkey, Richard T. Vaughan. The player/stage project: Tools for multi-robot and distributed sensor systems. Proceedings of the international conference on Advanced Robotics., pages 317-323, 2003.

[Calvo04] CALVO, R.: *"Comportamiento sigue persona con visión direccional."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2004.

[Lobato03] LOBATO, D.: *"Evitación de obstáculos basada en ventana dinámica."*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2003.

[Herencia04] ORTIZ HERENCIA R.: *"Comportamiento sigue persona con visión"*, Proyecto Fin de Carrera URJC,2004.

[Crespo03] CRESPO M.A..: *"Localización probabilística en un robot con visión local"*, Proyecto Fin de Carrera UPM,2003.

[Lopez05] LOPEZ A..: *"Navegación local utilizando grafo de visibilidad"*, Proyecto Fin de Carrera URJC,2004.

[Redo05] KACHACH R..: *"Localización del robot Pioneer basada en láser"*, Proyecto Fin de Carrera URJC,2005.

[Alberto05] LÓPEZ FERNÁNDEZ A.: *"Localización visual del robot Pioneer"*, Proyecto Fin de Carrera URJC,2005.

[Marta05] MARTÍNEZ DE LA CASA PUEBLA M.: *"Representación visual de escena con una cámara direccional"*, Proyecto Fin de Carrera URJC,2005.

[Lobato05] LOBATO, D.: *"Arquitectura JDE+ para el control de robots"*, Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2005.

[Ian Horswill] HORSWILL, I.: *"Polly: A Vision-Based Artificial Agent."* AAAI 1993.

[J. Borenstein, 1989] Y. Koren J. Borenstein.: *"Real-time obstacle avoidance for fast mobile robots."* IEEE Journal of Robotics and Automation, 1989.

[Reid Simmons,1996] Simmons R.: *"The curvature-velocity method for local obstacle avoidance"* Proceedings of the 1996 International Conference on Robotics and Automation.

[Victor Gómez,2003] Victor M. Gómez, José M. Cañas, Félix San Martín, Vicente Matellán: *"Vision based schemas for an autonomous robotic soccer player."* Proceedings of IV Workshop de Agentes Físicos WAF-2003, pp 109-120, Universidad de Alicante (Spain), March 2003. ISBN: 84-607-7171-7

[Raúl Isado,2005] José Raúl Isado: *"Navegacíon global de un robot utilizando metodo de gradiente"*Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2005.

[Konolige, 2000] Kurt Konolige.: *"A gradient method for realtime robot control."* IROS,2000.