ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

Ingeniería de Telecomunicación

PROYECTO FIN DE CARRERA

Autolocalización en tiempo real mediante seguimiento visual monocular

Autor: Luis Miguel López Ramos

Tutor: José María Cañas Plaza

Curso académico 2009/2010

a la memoria de mi hermano Eduardo.

Te echo de menos.

Agradecimientos

Quiero dar las gracias a todas las personas que me han apoyado mientras hacía este Proyecto, empezando por mis padres, que se han esforzado durante toda mi vida por darme lo mejor, y si hoy soy quien soy sin duda es gracias a ellos y a toda mi familia.

También quiero agradecérselo a José María Cañas, mi tutor en este gran trabajo. Lo elegí porque creo que disfruta mucho enseñando, y lo hace realmente bien, como he podido comprobar estos meses. Gracias, JoseMaria, por tu confianza y tu cercanía, por pensar siempre en positivo, y por contagiarnos a todos tu espíritu colaborativo.

También les debo este proyecto a todos los profesores que se esfuerzan cada día para que esta Escuela de Teleco tan joven salga adelante y gane prestigio día a día.

A todos mis compañeros del laboratorio de Robótica, en especial a Julio, Edu, Sara y Pablo, siempre dispuestos a echar una mano cuando los necesitara, y con los que he pasado tantos buenos ratos de duro trabajo pero también de absoluto relax.

A Chema, RaúlPatricia, *Peperra*, Fer, Ángel y muchos otros que me han acompañado durante este largo viaje de 5 años de clases.

A Lucía y a todos los amigos: *Escu*, Miguel, María, Jorge, Sergio, Chechu, Ian, Kevin y David... porque sois esenciales.

Resumen

Este proyecto se encuadra en el contexto del procesamiento de imagen y la visión artificial, con aplicaciones en la autolocalización de robots y la creación de sistemas de realidad aumentada.

El objetivo de este proyecto es el diseño y programación un algoritmo que estima en tiempo real la posición y orientación de una cámara móvil autónoma en un entorno estático, utilizando exclusivamente las imágenes obtenidas por la cámara. El algoritmo se ha validado experimentalmente haciendo uso de una cámara de videoconferencia real y en condiciones de laboratorio. El algoritmo calcula la trayectoria realizada y la muestra en una ventana de gráficos OpenGL, junto con un modelo de la cámara y las regiones de confianza de los puntos de referencia.

Las técnicas más importanes utilizadas en este proyecto son el filtro de Kalman, el filtro de Kalman extendido y el modelo *pinhole* de cámara. Se ofrece una perspectiva sistémica del algoritmo, dividida en dos partes diferenciadas: una de seguimiento 2D de puntos de interés basada en procesamiento de imagen y filtrado de Kalman, y otra de localización 3D de la cámara y establecimiento de puntos de referencia, basada en un filtro de Kalman extendido.

Para el desarrollo de la aplicación se han utilizado los lenguajes de programación C y C++, sobre la plataforma sofware JdeRobot. Esta plataforma nos ha permitido reutilizar componentes desarrollados en otros proyectos de una forma sencilla. Además, a lo largo del proyecto se han usado bibliotecas como OpenGL, OpenCV, GTK y GSL.

Índice general

Índice de figuras

Capítulo 1

Introducción

La vista es el sentido que más utilizamos para captar información de nuestro entorno y consiste en la habilidad de detectar la luz y de interpretarla. Tanto los humanos como los animales poseemos un sistema visual que nos permite crear un esquema de nuestro entorno y conocer detalles de los objetos que nos rodean. Gracias a estos detalles somos capaces de reconocer dichos objetos por su color, por su forma, detectar movimiento en ellos o incluso estimar aproximadamente la distancia que nos separa.

En los últimos años, un área activa de investigación es la reproducción de estas habilidades mediante sistemas informáticos. Las cámaras son sensores de bajo coste cada vez más comunes en nuestra vida cotidiana. Esta proliferación junto con el aumento de la capacidad de cómputo de los ordenadores de sobremesa han reavivado el interés en la visión por computador.

Este proyecto fin de carrera es una aplicación directa de la autolocalización visual, consiste en un sistema automático que permite seguir puntos fijos en el interior de una habitación y estimar la posición 3D de la cámara gracias a la información de las imágenes que se reciben.

En este primer capítulo se describe el contexto sobre el que se apoya el sistema: la visión artificial de modo genérico, la autolocalización y la realidad aumentada.

1.1. Visión artificial

La visión artificial o visión computacional es el área de la inteligencia artificial que se dedica a extraer información de las imágenes con el fin de comprender y asimilar lo que en ellas está sucediendo. La visión artificial tiene como objetivo por tanto interpretar

las imágenes analizadas para obtener información relevante sobre elementos que en ellas aparecen.

Los comienzos de la visión artificial se remontan a mediados del siglo XX, cuando aparecen los primeros programas para la detección de tanques u obstáculos. Sin embargo, el procesamiento ágil de las imágenes requiere una gran velocidad de cómputo que hasta la década de 1990 no se comenzó a lograr. A partir de entonces la visión computacional comenzó a emplearse para múltiples tareas y su desarrollo fue concentrándose en problemas de tratamiento de las imágenes como la segmentación, el reconocimiento de formas o el filtrado de bordes.

La utilización de estas técnicas posibilita el desarrollo de sistemas que persiguen objetivos concretos tales como *reconocimientos faciales o de iris* (figura 1.1 (a)), *reconocimiento de objetos* o *seguimiento 2D de objetos*. En robótica la visión puede utilizarse para la navegación del robot así como para la autolocalización o reconstrucción de lo que el robot ve.

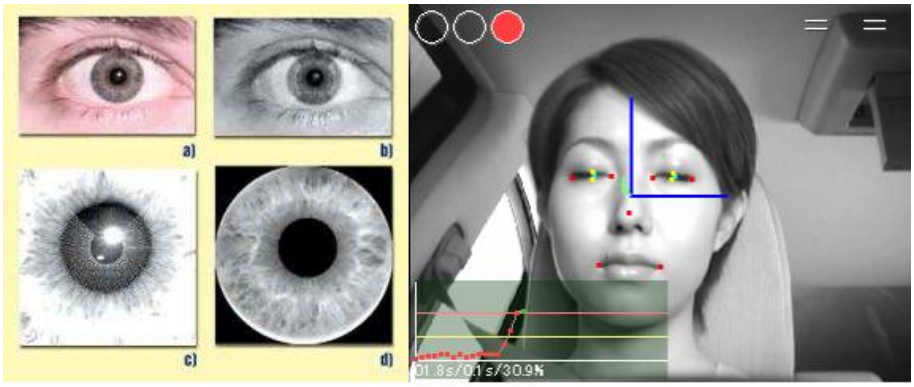


Figura 1.1: Reconocimiento con visión artificial - Iris (a) y Estado conductor (b).

Una reciente aplicación utilizada por la conocida marca de coches Nissan (figura 1.1 (b)) es el análisis de la cara del conductor mediante las imágenes que proporciona una pequeña cámara situada cerca del volante. Centrándose más concretamente en los ojos del conductor, permite extraer información acerca de la frecuencia de parpadeo y si los ojos están cerrados o abiertos, este análisis constituye una de las pruebas que realiza el vehículo para detectar si el conductor está en estado de embriaguez.

Otra aplicación de gran utilidad es la identificación automática de matrículas (figura 1.2). Situando cámaras en las carreteras o en aparcamientos el sistema permite localizar e

Figura 1.2: Identificación de matrículas.

identificar la matrícula en el vídeo y mediante un módulo de reconocimiento de caracteres se extrae el número de matrícula. Este sistema combinado con una base de datos que almacene información respecto a los conductores y vehículos es suficiente para gestionar y controlar el acceso a aparcamientos o incluso al casco urbano de una ciudad. Dos buenos ejemplos de ello son los aparcamientos del aeropuerto de Barajas y la zona centro de Londres.

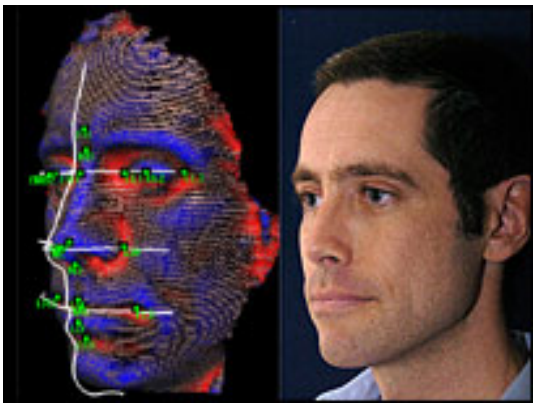


Figura 1.3: Modelo rostro tridimensional.

En la actualidad, los avances en geometría proyectiva, pares estéreo y autocalibración han abierto la puerta a la extracción de *información tridimensional* de las imágenes. El uso de estas nuevas técnicas permite alcanzar nuevas cotas en el área de la visión artificial, pues la información tridimensional permite conocer mejor y con una mayor precisión el entorno. Por ejemplo es posible hoy día realizar *reconocimiento 3D* de rostros u objetos (figura 1.3), que pueden ser utilizados en distintos ámbitos como la representación de piezas en entornos industriales, el reconocimiento de personas mediante el estudio biométrico en

áreas de seguridad, diseño infográfico para la industria del cine o de videojuegos, etc.

Otro ejemplo es el sistema *EyeVision*[1], que permite modelar en tiempo real una escena de manera fotorrealista con el fin de obtener nuevas imágenes virtualizadas de la misma. Este sistema ha sido empleado en la Superbowl para la repetición de jugadas desde cualquier ángulo. A través de la colocación estratégica de un determinado número de cámaras alrededor de un estadio es posible obtener la realidad virtualizada de lo sucedido segundos antes. La escena tridimensional se obtiene de la composición de las imágenes 2D que consigue cada una de las cámaras.

Figura 1.4: Sistema de repetición tridimensional Eye-Vision.

1.2. Autolocalización visual

Los seres humanos y muchos otros animales dotados de visión tenemos la habilidad de reconocer el mundo que nos rodea y hacernos una idea de dónde estamos a partir de lo que vemos. Además, la vista supone un importante apoyo al sentido del equilibrio. La evolución de la visión no se debe sólo a la mejora de los órganos sensores, los ojos, sino también a un cerebro preparado para procesar la información que le llega. Este procesamiento incluye un proceso de selección, ya que no toda la información que llega al ojo es útil (es redundante). La memoria desempeña un papel importante en el proceso de la visión, ayudándonos a clasificar sin darnos cuenta gran parte de lo que vemos.

Por lo tanto, un ordenador con un sensor óptico y un procesador lo suficientemente potentes y suficiente memoria debería ser capaz de imitar esta habilidad. Actualmente, la robótica se enfrenta al problema de la autolocalización apoyándose en visión binocular y

[1] http://www.ri.cmu.edu/events/sb35/tksuperbowl.html

localización probabilística, entre otros procedimientos. Dentro del grupo de Robótica de la URJC se han desarrollado varios proyectos de autolocalización visual (Alberto López, José Manuel Domínguez), cuya estrategia consiste en sintetizar una imagen a partir de un mapa, y compararla con la imagen obtenida por la cámara del robot. Esta comparación, junto con las medidas de odometría, da una medida de verosimilitud que se introduce en un filtro de partículas que calcula la posición más probable. Estos métodos necesitan de un mapa establecido a priori y del apoyo de la odometría[2], que es relativamente (aunque poco) fiable en el caso de robots con ruedas, pero no en robots con patas como los Nao. Se hace necesario para estos el desarrollo de técnicas de autolocalización que hagan la función de *odometría visual*.

En este proyecto nos planteamos como objetivo el saber dónde está la cámara y hacia dónde mira, a partir de las imágenes que recibe esta y sin ninguna información adicional. Este problema se conoce técnicamente con el nombre de monoSLAM.

1.2.1. MonoSLAM

MonoSLAM son las siglas de *Monocular Simultaneous Location And Mapping*, es decir, localización y construcción de mapas simultáneos basados en visión monocular. Se basa en imitar la capacidad innata de humanos y animales de reconocer el entorno sin tener ninguna información a priori. A diferencia de otras técnicas como visión binocular o sensado mediante láser, el monoSLAM necesita que el sensor óptico se mueva por el entorno para poder percibir la profundidad.

Además del monoSLAM y del SLAM óptico existen técnicas de SLAM no basadas en visión como la que utiliza el modelo de helicóptero autónomo de interiores del proyecto RANGE del MIT, que utiliza un sensor láser para averiguar dónde está a la vez que confecciona un mapa de su entorno. La utilidad del SLAM queda más que demostrada ya que es capaz de detectar huecos como la ventana de la figura y pasar a través de ellos.

Otro antecesor del monoSLAM es SFM, Structure from Motion, que se refiere al proceso de analizar la estructura tridimensional analizando la secuencia de imágenes de un objeto en movimiento. La idea es la misma que en monoSLAM, pero en las técnicas que se agrupan bajo este nombre no se contempla el requisito de funcionar en tiempo real, por lo que las imágenes se procesan en bloque (y no on-line) para calcular la estructura de un objeto que encaja mejor con la secuencia analizada.

[2]Estimación de la posición a partir del movimiento de los motores del robot

1.3. Algunas aplicaciones de la autolocalización

Desarrollar un algoritmo que estime la posición del sensor con respecto a puntos del entorno con una sola cámara tiene utilidad ya que cada vez son más los dispositivos portátiles con cámara (ordenadores portátiles, teléfonos móviles, consolas de videojuegos, sensores inalámbricos) cuyo tamaño se tiende a minimizar (orden de centímetros), y la visión binocular necesita de 2 cámaras separadas una distancia conocida y del orden de decenas de centímetros. Esto hace los dispositivos más caros y más grandes, que es lo que muchas veces se pretende evitar.

1.3.1. Realidad aumentada

Durante el último año ha ido ganando relevancia la realidad aumentada. Entendemos un sistema de realidad aumentada como aquél que presenta al usuario una imagen real en la que se superpone, de forma automática, información gráfica extra, que tiene una relación estrecha con la imagen real y depende de ésta. En particular, actualmente podemos encontrar sistemas que capturan una imagen real por medio de una cámara y la presentan al usuario, transformada, en una pantalla. Esto es sólo el principio ya que esta información puede presentarse en unas gafas semitransparentes o en un sistema de realidad virtual, dando la sensación de inmersión total en el sistema. Se especula con que en el futuro la realidad aumentada podría venir equipada en lentes de contacto [3].

La primera aplicación comercial de la realidad aumentada es como reclamo publicitario, ya que es muy vistosa. Películas como Transformers (2009) contrataron algunos chirimbolos publicitarios a pie de calle en los que el que se pusiera delante podía ver su propia imagen caracterizada como uno de los personajes de la película. Algunas revistas de actualidad incluyen aplicaciones que permiten que el lector apunte a la página impresa con su webcam para que parezca que las imágenes de la revista cobraban vida. Estas mismas tecnologías pueden servir para la posproducción de los spots, campo en el que muchísimas veces se recurre a efectos visuales para reforzar la ficción publicitaria.

1.3.2. Videojuegos

Otra aplicación comercial que se está popularizando actualmente, son los videojuegos. El juego *invizimals*, estrenado recientemente, nos permite jugar como si la consola PSP,

[3] http://spectrum.ieee.org/biomedical/bionics/augmented-reality-in-a-contact-lens/0

dotada de una webcam, fuese un "visor" de seres virtuales, que podremos ver moviéndose por nuestro entorno. Dado que para representar correctamente la parte virtual es necesario un patrón de referencia fácilmente reconocible por el software, este patrón se introduce en el argumento del juego como una *trampa* que sirve para atraer a los animales virtuales.

La presencia de la *trampa* hace que la experiencia de juego sea limitada, ya que en cuanto salga del campo de visión de la cámara no se "ve" a los personajes del juego, y si movemos la *trampa* de su sitio desaparece la sensación de realidad aumentada (si la arrastramos por una superficie, los personajes parecerán deslizarse por ella, y si la levantamos parecerán levitar por el aire). Esto se debe a que la aplicación no es capaz de autolocalizar la cámara si no es en presencia del patrón. Si este videojuego contase con un algoritmo de autolocalización visual en tiempo real como el descrito en este PFC no estaríamos sujetos a esta limitación.

(a)

(b)

Figura 1.5: Imágenes del videojuego Invizimals.

1.3.3. Ayuda a personas con discapacidades

Dejando un momento de lado las aplicaciones comerciales "masivas", la realidad aumentada puede servir para mejorar la percepción de personas con alguna discapacidad. Las personas con visión "túnel" sufren una reducción de su campo de visión. Este defecto se puede mitigar con la ayuda de unas gafas en las que se superponen imágenes generadas por ordenador[4], que "comprimen" la información visual en un rango más pequeño, de manera que el paciente con visión túnel puede hacerse una idea de qué tiene delante de un sólo golpe de vista, aproximadamente como lo haría una persona sin este problema. Los participantes del experimento conseguían encontrar objetos de su entorno en un tiempo típicamente 4 veces inferior al que tardarían sin la ayuda de este sistema. A este campo de investigación también se le conoce como Visión aumentada. En la figura 1.6(b) se puede ver cómo, a pesar del campo visual reducido, la visión aumentada nos permitiría saber que a la izquierda de la chica hay otra persona y hacernos una idea de a qué distancia están.

(a) (b)

Figura 1.6: (a) Gafas de visión aumentada para personas con visión túnel. (b) Imagenes que vería el paciente.

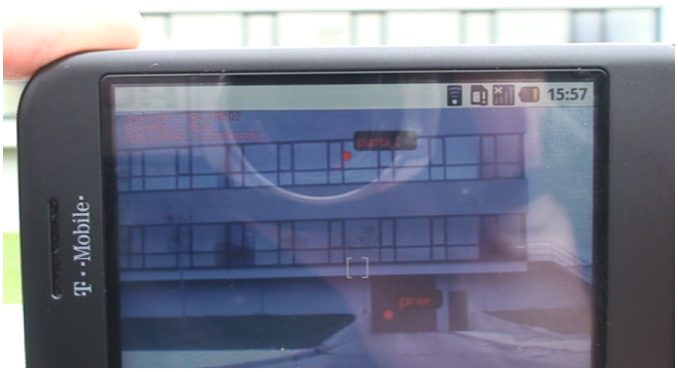
[4]http://www.newscientist.com/article/dn9886-augmented-reality-glasses-tackle-tunnel-vision-.html

1.3.4. Redes sociales móviles

Un escenario en el que la autolocalización y la realidad aumentada ganan relevancia cada día es el de las redes sociales móviles. En el nuevo horizonte que se vislumbra, los usuarios ya no sólo querrán acceder a información de personas o recursos como lugares, notas, fotografías, eventos, etc. sino ver de forma interactiva si se encuentran cerca de ellos o hacia dónde moverse para encontrarlos. Para ello los recursos deben estar georreferenciados, es decir, debe existir una base de datos que almacene (y actualice si es necesario) su posición. Además, esta información debe presentarse rápidamente y de forma resumida, ya que inevitablemente la tendencia será georreferenciarlo todo y si no se seleccionase la información llegaríamos a un punto en que la información mostrada saturaría la pantalla y la capacidad del usuario.

Tener datos sobre la posición y orientación de la cámara del teléfono móvil da a estas aplicaciones una ventaja decisiva: saber al instante dónde se centra la atención del usuario sin que éste tenga que hacer ningún esfuerzo. Además esto permite al usuario interactuar con la aplicación sin más que mover su terminal, por ejemplo acercándose a un objeto para obtener más datos sobre él.

Con esta intención y gracias a la potencia de los terminales actuales, existen ya prototipos de redes sociales móviles en las cuales se puede ver información sobre un lugar georreferenciado en la pantalla (figura 1.7(a)) o el seguimiento de una persona. La intención en versiones más avanzadas es que al enfocar por ejemplo un bar, la aplicación debería ser capaz de darnos la lista de precios, decirnos quiénes de entre nuestros contactos de la red están dentro, etc.



(a)

Figura 1.7: Funcionamiento de la red social LibreGeo.

Actualmente, la estimación de la posición de dispositivos móviles se fundamenta

([Román López, 2009]) en tecnología GPS (que generalmente no funciona en interiores) y en el conocimiento a priori de la infraestructura de la red, apoyándose en GSM, WiFi e IP. Estas técnicas no siempre tienen una exactitud suficiente para las aplicaciones de georreferenciación, por lo que la autolocalización visual podría contribuir a mejorar su eficiencia.

Capítulo 2

Objetivos

Una vez presentado el contexto en el que se desarrolla nuestro trabajo, en este capítulo vamos a detallar los objetivos concretos que pretendemos alcanzar con nuestro proyecto, así como los requisitos que debe cumplir nuestra solución.

2.1. Descripción del problema

Se diseñará y programará un algoritmo que estime en tiempo real la posición y orientación de una única cámara móvil autónoma en un entorno estático, utilizando exclusivamente las imágenes obtenidas por la cámara.

El algoritmo se validará experimentalmente haciendo uso de una webcam real y en condiciones de laboratorio, es decir, se sostendrá la cámara en la mano y se realizará un movimiento suave (sin acelerones ni giros bruscos) ante un escenario favorable y el algoritmo calculará la trayectoria realizada y la mostrará en una ventana de gráficos OpenGL.

Además, el proyecto se articulará en varios subobjetivos que nos permitirán alcanzar el objetivo principal y que se describen a continuación:

1. Se programará un algoritmo de seguimiento 2D de puntos de interés en la imagen capturada por la cámara. La aplicación de seguimiento 2D será de carácter general y servirá de base tanto para este como para otros proyectos de visión del grupo. Esta aplicación permitirá el ajuste de los parámetros del algoritmo mediante ensayo/error.

2. Se diseñará un algoritmo que estime la posición y la orientación de una cámara dadas las proyecciones de algunos puntos fijos y conocidos, basándonos en el diseño de [Davison, 2007]. Para ello se empleará un Filtro Extendido de Kalman (EKF).

3. Se combinarán la aplicación de seguimiento 2D y el algoritmo de localización 3D en una aplicación prototipo que permitirá validar experimentalmente la efectividad de estos sistemas y mostrar los resultados en pantalla. La aplicación deberá crear nuevos puntos de referencia automáticamente.

2.2. Requisitos

Teniendo en cuenta los objetivos de la sección anterior, el proyecto deberá satisfacer una serie de requisitos descritos a continuación:

- Software modular: El software programado en este proyecto será un componente de JdeRobot, la plataforma de desarrollo de software del grupo de Robótica de la URJC. Este componente hará uso de componentes ya existentes y proveerá de datos a otros componentes.
- Plataforma: La arquitectura de desarrollo JdeRobot está desarrollada en lenguaje C/C++ sobre el sistema operativo GNU/Linux, por lo tanto se establece como requisito de partida el desarrollo del proyecto en estos lenguajes y sobre este sistema operativo.
- Visión artificial: Para la autolocalización, deberemos hacer uso únicamente de una cámara, no pudiendo utilizar otro tipo de sensores que nos ayuden en esta tarea, y obligando a hacer uso de la visión artificial.
- Interfaz de imágenes: Las imágenes procesadas se obtienen de la interfaz Varcolor de JdeRobot, por lo que podrán tener distinto origen: una cámara local, una grabación en archivo, un programa de simulación o incluso la red. Las características como el tamaño y la frecuencia de refresco también podrán variar. Los algoritmos deberán funcionar independientemente del tipo de imágenes utilizadas.
- Cámara: El sistema deberá funcionar con todo tipo de cámaras, a pesar de que tengan distintas distancias focales. Cada cámara real se calibrará con el software incluido en la suite JdeRobot [Kachach, 2008]
- Tiempo real: La utilidad del algoritmo reside en que se conozca la posición de la cámara móvil en el instante actual, por lo que el procesamiento de la imagen deberá hacerse on-line, intercalándose con los procesos de adquisición de imagen y presentación de resultados. Para aprovechar al máximo la información que provee la

cámara, el tiempo empleado en procesar un frame deberá ser menor que el periodo de adquisición de la cámara, alcanzando una tasa de 30 IPS.

- Precisión: Una condición de éxito de este proyecto es que alcance una precisión mayor que los utilizados hasta ahora por el grupo en alguno de los escenarios de trabajo de robótica.

- Robustez: Un efecto que se desea evitar en lo posible es la pérdida de los puntos de referencia, ya que si perdemos demasiados no dispondremos de datos para mantener la cámara localizada, viéndonos obligados a apoyarnos otros algoritmos para relocalizar la cámara.

- Licencia: El código del proyecto es software libre, por lo que será liberado bajo la licencia **GPLv3**[1].

2.3. Metodología de desarrollo

En el desarrollo del sistema descrito, el modelo de ciclo de vida utilizado ha sido el modelo en espiral basado en prototipos, ya que permite desarrollar el proyecto de forma incremental, aumentando la complejidad progresivamente y haciendo posible la generación de prototipos funcionales.

Este tipo de modelo de ciclo de vida nos permite obtener productos parciales que puedan ser evaluados, ya sea total o parcialmente, y facilita la adaptación a los cambios en los requisitos, algo que sucede muy habitualmente en los proyectos de investigación.

El modelo en espiral se realiza por ciclos, donde cada ciclo representa una fase del proyecto. Dentro de cada ciclo del modelo en espiral se pueden diferenciar 4 partes principales que pueden verse en la figura 2.1, y donde cada una de las partes tiene un objetivo distinto:

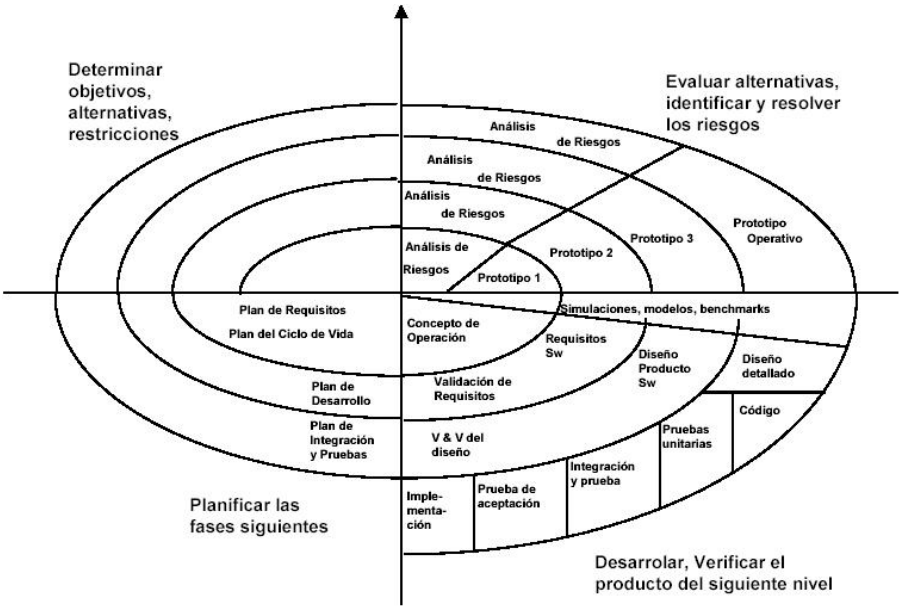
- Determinar objetivos: Se establecen las necesidades que debe cumplir el sistema en cada iteración teniendo en cuenta los objetivos finales, por lo que según avancen las iteraciones aumentará el coste del ciclo y su complejidad.

- Evaluar alternativas: Determina las diferentes formas de alcanzar los objetivos que se han establecido en la fase anterior, utilizando distintos puntos de vista, como el

[1] http://www.gnu.org/licenses/gpl-3.0-standalone.html

rendimiento que pueda tener en espacio y tiempo, las formas de gestionar el sistema, etc. Además se consideran explícitamente los riesgos, intentando mitigarlos lo máximo posible.

- Desarrollar y verificar: Desarrollamos el producto siguiendo la mejor alternativa para poder alcanzar los objetivos del ciclo. Una vez diseñado e implementado el producto, se realizan las pruebas necesarias para comprobar su funcionamiento.
- Planificar: Teniendo en cuenta el funcionamiento conseguido por medio de las pruebas realizadas, se planifica la siguiente iteración revisando posibles errores cometidos a lo largo del ciclo y se comienza un nuevo ciclo de la espiral.



(a)

Figura 2.1: Modelo en espiral.

Los ciclos que se han seguido en nuestro proyecto están relacionados con cada una de las etapas que se describirán en la siguiente sección. A lo largo de estas etapas, se han realizado reuniones semanales con el tutor del proyecto para negociar los objetivos que se pretendían alcanzar y para evaluar las alternativas de desarrollo.

2.3.1. Plan de trabajo

Durante el transcurso del proyecto, se han marcado progresivamente una serie de requisitos a cumplir dividiendo el trabajo a realizar en distintas etapas:

1. Familiarización con JdeRobot: el objetivo de esta fase es aprender a utilizar el software JdeRobot y las aplicaciones y *drivers* que contiene, ya que algunos de estos formarán parte del proyecto final. Se hace uso del componente Opencvdemo [**?**] para aprender los procedimientos propios de la arquitectura y cómo usar los componentes de las interfaces GTK.

2. Introducción a la visión artificial: Una vez hecha la toma de contacto con la arquitectura software, se procede a ampliar Opencvdemo con funciones de procesamiento de imagen que más tarde emplearemos en nuestro proyecto, tales como detección de esquinas, filtros morfológicos, segmentación mediante blobs, y cálculo del histograma; así como otras utilidades de la biblioteca openCV: uso de regiones de interés para disminuir la carga computacional y funciones de trazado que permiten ver sobre la propia imagen los resultados del procesamiento.

3. Seguimiento 2D: Llegados a este punto se lleva a cabo el trabajo para cumplir el primer subobjetivo:

   Seguimiento 2D de un objeto: A partir de una característica extraída de forma simple (filtro de color + media), se procede a aplicar un filtro de Kalman que debe reforzar la medida frente a ruido y oclusiones momentáneas.

   Seguimiento 2D de varios objetos (de idéntica apariencia): Primero para un número fijo de objetos (segmentación mediante el algoritmo K-medias), después para un número variable de objetos (segmentación mediante blobs y filtros morfológicos), se intenta que el sistema distinga cada objeto a partir sólo de su posición, a pesar de que pueda haber cruces y oclusiones.

   Aplicar seguimiento 2D a detección de esquinas: Para hacerlo más robusto, se tendrán en cuenta también características basadas en apariencia.

4. Salto de 2D a 3D: basándonos en [Davison, 2003], diseñamos el EKF que estima la posición y orientación de la cámara a partir de unos puntos de referencia conocidos. Dada la complejidad del sistema, se procede a diseñar primero el modelo de predicción y después el de observación. Posteriormente utilizamos los datos de salida del seguimiento 2D como entrada para la localización 3D.

5. Interfaz de usuario: Dado que en esta fase la posición real de los puntos de referencia se establece de forma manual en esta fase.Para facilitar la depuración, se crea un GUI en el que podamos Reutilizando código de otros proyectos del grupo ([Miangolarra,

2010], [Barrera, 2008]) incluimos en la aplicación un visor que representa en un escenario virtual los puntos de referencia y la cámara.

6. Fase de depuración y optimización: En esta fase se presta especial atención a que el algoritmo ejecute en un tiempo razonable para alcanzar una tasa de 30 IPS, para ello analizamos los puntos críticos del procesamiento y reducimos al máximo el tiempo de ejecución, con el objetivo de que el algoritmo escale al mayor número de puntos de referencia posibles.

7. Fase de pruebas: Las pruebas se realizan en un escenario sencillo en el que el seguimiento 2D es muy robusto para encontrar los parámetros óptimos del EKF. Se elabora un informe de cuántos puntos somos capaces de seguir a 30 IPS, con qué resolución, y se estimará el sesgo y el error cuadrático medio del estimador de posición y orientación.

8. Inicialización automática de nuevos puntos de referencia: Llegados a este punto, se evalúan dos alternativas: utilizar un filtro de partículas [Davison, 2003] o ampliar el EKF según el artículo [Montiel *et al.*, 2006].

Se puede ver un seguimiento detallado de este plan de trabajo en el mediawiki[2] del grupo de Robótica junto con imágenes y vídeos que muestran los resultados alcanzados.

[2]http://jderobot.org/index.php/Lm.lopez-pfc-teleco

Capítulo 3

Fundamentos Teóricos

En este capítulo describiremos algunos de los procedimientos y técnicas matemáticas más importantes que se han utilizado en el proyecto. En concreto, se han utilizado algunos conceptos de geometría proyectiva, rotaciones en espacio tridimensional, el filtro de Kalman y el filtro de Kalman extendido.

3.1. Geometría Proyectiva y modelo matemático de cámara

El problema que queremos resolver implica proyecciones de un espacio tridimensional (el mundo externo a la cámara) en un espacio bidimensional (ya que las imágenes son planas). Por lo tanto, es fundamental conocer la manera en que un punto 3D se convierte en un punto 2D, y esto conlleva modelarlo matemáticamente.

3.1.1. Parámetros intrínsecos: Modelo Pinhole

El modelo de cámara pinhole recoge la idea de una proyección cónica, es decir, asume que todos los rayos de luz pasan por el mismo punto (el foco de la cámara). Se basa en el modelo de cámara oscura, en el que los rayos de luz entran en una caja por un agujero minúsculo e impactan en el lado contrario, formando una imagen del objeto que la caja tiene enfrente.

En el caso de las cámaras actuales, el modelo pinhole funciona porque las lentes delgadas ideales son capaces de concentrar los rayos de luz en un solo punto. Se puede ver un dibujo simplificado en la figura 3.1, en la que se muestra el plano imagen frente al foco de la cámara

y no detrás, como lo estaría en una cámara real. Este modelo es útil por su sencillez y tiene buena precisión a la hora de modelar las cámaras utilizadas en este proyecto.

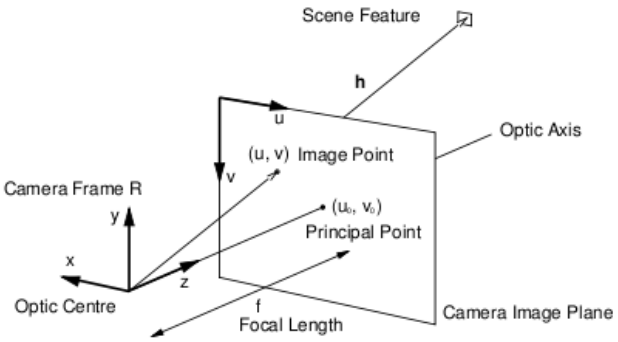


Figura 3.1: Modelo pinhole de cámara.

Una cámara ideal con el foco situado en el origen de coordenadas, apuntando en la dirección positiva del eje z, proyecta los puntos 3D en el plano imagen según la siguiente ecuación:

$$\begin{pmatrix} u \\ v \end{pmatrix} = F \cdot \begin{pmatrix} -\frac{x}{z} \\ -\frac{y}{z} \end{pmatrix}$$

donde F es la distancia focal, es decir, la distancia del foco al plano imagen.

Por razones de índole informática, el origen de coordenadas de las imágenes almacenadas en un sistema informático suele encontrarse en la esquina superior izquierda de la imagen, por lo que, si esta tiene dimensiones $m \times n$, la ecuación queda:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} m/2 \\ n/2 \end{pmatrix} + F \cdot \begin{pmatrix} -\frac{x}{z} \\ -\frac{y}{z} \end{pmatrix}$$

El punto $\binom{m/2}{n/2}$ sería el centro óptico de nuestro modelo.

Aun suponiendo que la lente fuera ideal, si esta no está perfectamente alineada con el plano de proyección (físicamente, la película en caso de las cámaras tradicionales, o el CCD en el caso de las cámaras digitales) esto da lugar a que el centro óptico no se encuentre exactamente en $\binom{m/2}{n/2}$ sino en un punto genérico $\binom{u_0}{v_0}$ llamado punto principal. Además la imagen podría estar ligeramente achatada, lo que se modela como una distancia focal distinta en el eje x que en el eje z. Finalmente, el modelo queda:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} + \begin{pmatrix} f_x & 0 \\ 0 & f_y \end{pmatrix} \begin{pmatrix} -\frac{x}{z} \\ -\frac{y}{z} \end{pmatrix} \tag{3.1}$$

Otras imperfecciones pueden hacer que la matriz F no sea diagonal, pero el sistema seguiría siendo lineal. Este modelo, ampliado, puede encontrarse en [**?**], en el que se utilizan coordenadas homogéneas (que se van a evitar en este proyecto por no resultar de utilidad).

La matriz de proyección y las coordenadas del centro óptico se conocen como parámetros intrínsecos de la cámara. En este proyecto necesitaremos cámaras calibradas, es decir, de parámetros intrínsecos conocidos.

3.1.2. Parámetros extrínsecos

Por otro lado, especialmente en el caso de cámaras móviles, no podemos asumir el foco está en el origen de coordenadas ni que la cámara mira en la dirección positiva del eje z. Para ello emplearemos dos espacios ortonormales de coordenadas, uno haciendo referencia al mundo real y otro solidario con la cámara.

Siguiendo la formulación de [Davison, 2002], asignaremos el superíndice R a las coordenadas solidarias con la cámara y el superíndice W a las coordenadas del mundo real. En el espacio solidario con la cámara, ésta siempre apunta en la dirección del eje z y su foco se encuentra siempre en el cero. En el espacio del mundo real, la cámara se encuentra en el punto t^{WR} y su orientación viene dada por la matriz R^{WR} (ver sección 3.2).

Para traducir un punto de un sistema de referencia a otro, se utiliza la siguiente ecuación lineal:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}^W = t^W + R^{WR} \begin{pmatrix} x \\ y \\ z \end{pmatrix}^R \tag{3.2}$$

El vector t^{WR} y la matriz R^{WR} constituyen los parámetros extrínsecos de la cámara. Precisamente el objetivo de este proyecto se define como estimar en tiempo real los parámetros extrínsecos de una cámara móvil.

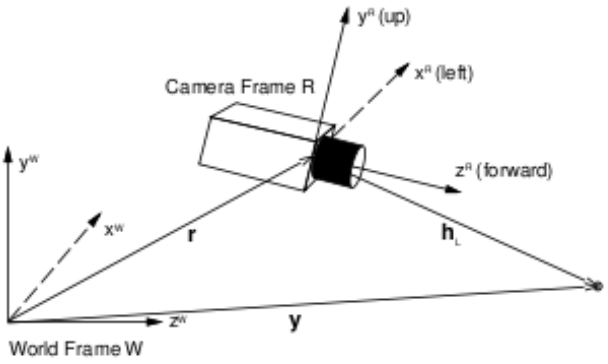


Figura 3.2: Parámetros extrínsecos y relación entre sistemas de referencia.

3.2. Cuaterniones y Rotación Espacial

Hay varias maneras de expresar la rotación. Para empezar, indicaremos que una rotación tiene 3 grados de libertad, ya que no sólo nos interesa la dirección en la que apunta nuestra cámara sino cuánto ha girado sobre su eje óptico.

La rotación se puede expresar mediante una matriz de rotación 3×3 que tiene 9 valores. Estos 9 valores no son independientes ya que las matrices de rotación son unitarias (formadas por vectores unitarios ortogonales entre sí). Por razones de coste computacional no es conveniente estimar directamente estos 9 valores.

En aviónica se emplean los ángulos de Tait-Bryan, conocidos como pitch-yaw-roll: *pitch* es el ángulo que forma la dirección de avance del avión con el plano horizontal, *yaw* es el rumbo de navegación, y *roll* es el ángulo que indica el horizonte artificial. Es muy común la extensión de este sistema al control de brazos robóticos.

Para el caso de las cámaras, una forma muy extendida es partir del punto en que se encuentra la cámara y definir un segundo punto llamado foco de atención (FOA), cuya proyección recae en el centro óptico, y añadir el parámetro *roll*.

Estos dos sistemas tienen un problema, y es que cuando se apunta al cenit el parámetro *roll* pierde su significado. Este efecto se conoce como *gimbal lock* y puede provocar que los sistemas de control generen movimientos bruscos y velocidades impredecibles.

Una representación que evita este efecto nocivo es la que utiliza cuaterniones. Un cuaternión es, básicamente, un vector de 4 componentes. Los cuaterniones se pueden ver también como entidades matemáticas análogas a los números complejos formadas por una

parte real escalar q_0 y una parte imaginaria que es un vector de 3 dimensiones $\begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix}$.

Dejando un poco de lado la teoría de los cuaterniones, la parte que nos interesa es que un cuaternión unitario (tal que la suma de los cuadrados de sus componentes es igual a 1) representa una rotación en el espacio. La parte vectorial del cuaternión es el eje sobre el cual se ha girado. La parte real del cuaternión indica qué ángulo α se ha girado según la fórmula:

$$\alpha = 2 \arccos q_0 \tag{3.3}$$

Una ventaja de esta representación es que la matriz de rotación se obtiene a partir del cuaternión de forma sencilla:

$$R(q) = \begin{pmatrix} a^2+b^2-c^2-d^2 & 2bc-2ad & 2bd+2ac \\ 2bc+2ad & a^2-b^2+c^2-d^2 & 2cd-2ab \\ 2bd-2ac & 2cd+2ab & a^2-b^2-c^2+d^2 \end{pmatrix} \tag{3.4}$$

donde $\begin{pmatrix} a & b & c & d \end{pmatrix}$ son las componentes $\begin{pmatrix} q_0 & q_1 & q_2 & q_3 \end{pmatrix}$ del cuaternión q. Un cuaternión no unitario daría como resultado una rotación añadida a un cambio de escala. Otra propiedad interesante es que la distancia euclídea entre 2 cuaterniones (tomados como vectores de 4 componentes) da una métrica de cómo de parecidas son dos transformaciones. La ventaja que podemos obtener de esto es que, si obtenemos un estimador de un cuaternión que no es unitario, basta con normalizarlo (dividir entre su módulo, lo que equivale a buscar el cuaternión unitario más cercano) para obtener un estimador mejor. Además los cuaterniones son cómodos para trabajar con distribuciones gaussianas 4-dimensionales.

Otras propiedades:

- El cuaternión identidad es $\begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}$. El eje de giro es un vector nulo, pero esto no provoca ningún problema ya que si $q_0 = 1$ entonces $\alpha = 0$.
- Un cuaternión $q = \begin{pmatrix} q_0 & q_1 & q_2 & q_3 \end{pmatrix}$ y su opuesto $-q = \begin{pmatrix} -q_0 & -q_1 & -q_2 & -q_3 \end{pmatrix}$ representan la misma rotación. Hay que prestar especial atención a esta propiedad cuando se resten cuaterniones.
- El conjugado de un cuaternión q, $q^* = \begin{pmatrix} q0 & -q1 & -q2 & -q3 \end{pmatrix}$ representa la rotación contraria a q, igual que ocurre con la inversa de una matriz de rotación. Como las matrices de rotación son unitarias y reales, su inversa es igual a su hermítica y a su traspuesta.

$$R(q^*) = R^{-1}(q) = R^H(q) = R^T(q) \tag{3.5}$$

- La composición de dos rotaciones se expresa mediante el producto de las matrices de rotación (ordenando las matrices de derecha a izquierda), y mediante el producto de Hamilton[1] en el caso de los cuaterniones (en el mismo orden que las matrices).

3.3. Modelos dinámicos y filtrado de Kalman

El filtro de Kalman es un caso de estimador bayesiano muy utilizado en Ingeniería, que permite estimar on-line el estado de un sistema dinámico en tiempo discreto a partir de observaciones en presencia de ruido. Bajo determinadas condiciones (sistemas lineales con ruido gaussiano, como se verá más adelante), el filtro de Kalman es un estimador de mínimo error cuadrático medio y, además de ofrecer una estimación del estado del sistema, también ofrece una medida de la incertidumbre con que se ha hecho la estimación.

Bajo unas condiciones menos estrictas y a costa de perder la optimalidad, el filtro de Kalman puede emplearse en su versión extendida para una familia más amplia de sistemas dinámicos.

A continuación se incluye una breve introducción al filtrado de Kalman y bajo qué condiciones se puede aplicar. La formulación completa se puede encontrar en [Haykin, 1986].

3.3.1. Modelo probabilístico

Un sistema dinámico se puede definir de manera probabilística con estas 3 fdps:

- El estado del sistema en cada instante es una variable aleatoria n-dimensional $X_{|t}$, y a lo largo del tiempo conforma un proceso aleatorio $\left\{X_{|t}\right\}_{t\geq 0}$ markoviano (es decir, si conozco $x_{|t-1}$, los anteriores no me dan ninguna información adicional acerca de $x_{|t}$). Este proceso está caracterizado por una fdp de transición de estado $p\left(x_{|t} \mid x_{|t-1}\right)$.

- La observación del sistema en cada instante es una variable aleatoria l-dimensional $Y_{|t}$ que sólo depende de $X_{|t}$, caracterizada por una fdp condicional $p\left(y_{|t} \mid x_{|t-1}\right)$. Las observaciones son condicionalmente independientes: $p\left(y_{|1:t} \mid x_{|1:t-1}\right) = \prod(p\left(y_{|k} \mid x_{|k}\right))$. Si fijamos $Y_{|t} = y_{|t}$, entonces $p\left(y_{|t} \mid x_{|t-1}\right)$ es la verosimilitud de $x_{|t}$.

- La fdp a priori para el estado $x_{|0}$: $p\left(x_{|t}\right)$

[1] http://en.wikipedia.org/wiki/Quaternions#Hamilton_product

3.3.2. Predicción y filtrado

Estamos interesados en conocer la esperanza matemática de un vector que es función de nuestro estado. Para ello se definen dos fdp auxiliares:

$$p\left(x_{|t} \mid y_{|1:t}\right) = \text{densidad de filtrado} = p\left(y_{|t} \mid x_{|t}\right) \cdot \frac{p\left(x_{|t} \mid y_{|1:t-1}\right)}{p\left(y_{|t} \mid y_{|1:t-1}\right)} (\text{regla de Bayes})$$

$$p\left(x_{|t} \mid y_{|1:t-1}\right) = \text{densidad de predicción} = \int \left(p\left(x_{|t} \mid x_{|t-1}\right) \cdot p\left(x_{|t-1} \mid y_{|1:t-1}\right) dx_{|t-1}\right.$$

y el algoritmo de filtrado recursivo óptimo es:

1. predicción: $p\left(x_{|t} \mid y_{|1:t-1}\right) = \int p\left(x_{|t} \mid x_{|t-1}\right) \cdot p\left(x_{|t-1} \mid y_{|1:t-1}\right) dx_{t-1}$
2. actualizacion: $p\left(x_{|t} \mid y_{|1:t}\right) \propto p\left(y_{|t} \mid x_{|t}\right) \cdot p\left(x_{|t} \mid y_{|1:t-1}\right)$

Pero $p\left(x_{|t} \mid y_{|1:t-1}\right)$ no tiene forma analítica cerrada salvo cuando:

a. el espacio de estados es discreto y finito.

b. trabajamos con un sistema lineal y gaussiano, en cuyo caso las densidades de filtrado y de predicción son gaussianas. Precisamente el filtro de Kalman utiliza la solución analítica de estas fdp.

3.3.3. Modelos dinámicos

El modelo estadístico es muy rico pero es poco manejable a la hora de modelar sistemas reales. Para ello suele utilizarse el modelo de sistemas dinámicos estocásticos en formato de espacio de estados (*state-space models*):

- Estado del sistema: $x_{|t} \in \mathbb{R}^n$. Generalmente, el estado no es observable directamente.
- Observaciones del sistema: $y_{|t} \in \mathbb{R}^l$. Son observables.
- Ecuación de estado: $x_{|t} = f(x_{|t-1}) + u_{|t}$.

  $f : \mathbb{R}^n \to \mathbb{R}^n$ posiblemente no lineal.

  $u_{|t} \in \mathbb{R}^n$ tiene carácter de perturbación, es ruidoso.
- Ecuación de observación: $y_{|t} = h(x_t) + v_{|t}$.

  $h : \mathbb{R}^n \to \mathbb{R}^l$ posiblemente no lineal.

  $v_{|t} \in \mathbb{R}^l$ es ruidoso, generalmente el error sistemático de medida.

3.3.4. Filtro de Kalman

Vamos a particularizar nuestro modelo dinámico para funciones lineales y perturbaciones gaussianas:

- El vector de estado x_t sigue una distribucción gaussiana n-dimensional de media xgorro sub t y tiene una matriz de covarianza P. Por lo tanto, el estimador MMSE del vector de estado es precisamente xgorrot, la media de la distribución.
- El vector de observación sigue una distribución gaussiana l-dimensional de media ysubt y tiene una matriz de covarianza S.
- las funciones f y h (de estado y de observación) son lineales y quedan definidas por las matrices $F_{n\times n}$ y $H_{l\times n}$ respectivamente.
- Las perturbaciones u_t y v_t son gaussianas, ambas con media 0, y con matrices de covarianza $Q_{t,n\times n}$ y $R_{t,l\times l}$ respectivamente.
- La fdp a priori del sistema es también gaussiana, de media $\hat{x}_{|0}$ y matriz de covarianza $P_{|0}$.

Dadas estas propiedades, se puede demostrar que la estimación MMSE del vector de estado y de las matrices de covarianza se pueden obtener de forma incremental con el siguiente algoritmo:

Predicción

Predicción del estado (pronóstico)	$\hat{x}_{t\|t-1} = F_t\hat{x}_{t-1}$
Predicción de la covarianza	$P_{t\|t-1} = F_tP_{t-1}F_t^T + Q_t$

Actualización

Innovación o residuo	$\tilde{y}_t = z_t - H_t\hat{x}_{t\|t-1}$
Covarianza del residuo	$S_t = H_tP_{t\|t-1}H_t^T + R_t$
Ganancia de Kalman	$K_t = P_{t\|t-1}H_t^TS_t^{-1}$
Estimación actualizada del estado	$\hat{x}_t = \hat{x}_{t\|t-1} + K_t + \tilde{y}_t$
Covarianza actualizada del estado	$P_t = (I - K_tH_t)P_{t\|t-1}$

3.3.5. Filtro de Kalman Extendido

Cuando las ecuaciones de estado u observación, o ambas, son no lineales, se puede hacer una aproximación que consiste en linealizar las funciones f y h en torno al vector de

estado actual. Para ello, se sustituyen las matrices F y H por sus jacobianas JF_t y JH_t. La aproximación mejora cuanto mayor es la frecuencia de muestreo del sistema.

Predicción	
Predicción del estado (pronóstico)	$\hat{x}_{t\|t-1} = f(\hat{x}_{t-1})$
Predicción de la covarianza	$P_{t\|t-1} = JF_t P_{t-1} JF_t^T + Q_t$
Actualización	
Innovación o residuo	$\tilde{y}_t = z_t - h(\hat{x}_{t\|t-1})$
Covarianza del residuo	$S_t = JH_t P_{t\|t-1} JH_t^T + R_t$
Ganancia de Kalman	$K_t = P_{t\|t-1} JH_t^T S_t^{-1}$
Estimación actualizada del estado	$\hat{x}_t = \hat{x}_{t\|t-1} + K_t + \tilde{y}_t$
Covarianza actualizada del estado	$P_t = (I - K_t JH_t) P_{t\|t-1}$

Obsérvese que las funciones f y h sólo se sustituyen por las matrices JF_t y JH_t cuando éstas afectan a matrices. Cuando afectan a vectores (en el caso de la predicción de x_t y el cálculo del residuo) se prefiere conservar la función original ya que la aproximación conlleva una pérdida de exactitud.

La versión extendida del filtro de Kalman ya no es un estimador óptimo, y será menos óptimo cuanto más no-lineales sean las funciones de estado y observación. Además, el EKF es potencialmente catastrófico, es decir, si la hipótesis de partida es incorrecta, el filtro podría inestabilizarse y no llegar nunca a aproximar razonablemente el vector de estado. Otra desventaja es que tiende a subestimar la matriz de covarianza.

Capítulo 4

Materiales y métodos

En este capítulo vamos a describir los elementos empleados en el desarrollo del proyecto, tanto hardware como software. Además describiremos varias aplicaciones que hemos desarrollado y que nos han sido de ayuda para alcanzar los objetivos del proyecto.

4.1. Elementos hardware

Para el funcionamiento de este sistema se necesita una única cámara conectada a un ordenador. El software incluído está preparado para ser compilado y ejecutado en cualquiera de los PC's disponibles actualmente en el mercado, siendo el principal elemento limitador la capacidad de procesamiento, como se verá en el apartado de experimentos. Las pruebas se realizaron en un PC equipado con un procesador Intel(R) Core(TM)2 Quad CPU Q9300 @ 2.50GHz.

El sistema operativo utilizado para la implementación del sistema ha sido Ubuntu 8.04, una de las distribuciones de GNU/Linux más importantes actualmente, y que está basada en Debian. Este sistema operativo es de libre distribución y soporta oficialmente las arquitecturas hardware Intel x86 y AMD64.

Para el desarrollo y las pruebas del algoritmo se han empleado dos tipos de cámaras:

- Logitech® Webcam Pro 9000 para las pruebas más básicas. Se comunica con el PC mediante el puerto USB por lo que su rendimiento típico es con una tasa de refresco de 30 fps con una resolución de 320x240 px.
- Apple iSight, que utiliza el puerto FireWire. Se usa para pruebas más avanzadas por poder alcanzar una tasa de 30 fps con una resolución de 640x480 px.

(a) (b)

Figura 4.1: (a) Cámara Logitech Quickcam pro 9000. (b) Cámara Apple iSight

4.2. JdeRobot

La implementación se ha realizado sobre la plataforma JdeRobot [1]. Esta plataforma ha sido creada por el grupo de robótica de la URJC, se trata de una plataforma de desarrollo de software para aplicaciones con robots y visión artificial.

La principal razón por la que se ha elegido JdeRobot es que ofrece una interfaz sencilla para la programación de sistemas de tiempo real y resuelve problemas relacionados con la sincronización de los procesos y la adquisición de datos. Además está preparada para aprovechar la potencia de los procesadores multinúcleo, evitando por ejemplo que la interfaz de usuario ralentice el procesamiento de la información. Otras razones son la facilidad con la que se pueden utilizar drivers y aplicaciones ya creados para esta plataforma de desarrollo y la posibilidad de reutilizar código de proyectos con partes similares.

JdeRobot está programado fundamentalmente en C/C++, aunque componentes de las últimas versiones hacen uso de Python (jdeRobot 4.4) y Java (jdeRobot 5.0) , y proporciona un entorno de programación donde la aplicación se compone de distintos hilos de ejecución asíncronos llamados esquemas. Cada esquema es un *plugin* que se carga automáticamente en la aplicación y que realiza una funcionalidad específica que podrá ser reutilizada.

Los componentes de JdeRobot que realizan funciones específicas de procesamiento y toma de decisiones se llaman **esquemas**. Existen dos tipos de esquemas en JdeRobot, los perceptivos, que son los encargados de realizar el procesamiento de datos para proporcionar información sobre el robot y el mundo en el que opera, y los esquemas de actuación, que se encargan de tomar decisiones para alcanzar una meta, como lanzar órdenes a los motores

[1] http://jderobot.org/

o lanzar nuevos esquemas, ya que pueden combinarse formando jerarquías. El componente que se desarrollará en este proyecto es exclusivamente perceptivo, no utilizaremos ningún actuador.

JdeRobot simplifica el acceso a los dispositivos hardware desde el programa, de modo que para obtener la imagen actual de la cámara sólo hay que leer una variable local. La interfaz de imagen es la misma para distintas fuentes de vídeo, lo que nos da una gran flexibilidad. Operar con actuadores en el caso de esquemas de actuación es tan simple como escribir en otra variable local. Para generar este comportamiento existe una serie de **drivers**, que actúan como *plugins* y son los encargados de dialogar con los dispositivos hardware concretos.

En particular, para la realización de nuestro proyecto, hemos utilizado algunos de los drivers que proporciona la plataforma, que son:

- Video4linux: Driver encargado de la obtención de imágenes desde las cámaras USB (como la Logitech quickcam) que realiza la actualización de la imagen obtenida de la cámara automáticamente.
- Firewire: Driver equivalente a Video4linux pero preparado para obtener imágenes de las cámaras mediante el puerto IEEE 1392 (FireWire) como es el caso de la cámara Apple iSight.
- Mplayer: Este driver hace uso de la aplicación mplayer y de los pipelines del sistema operativo para decodificar un archivo de vídeo o una secuencia de *broadcast* en cualquier formato compatible con mplayer y actualizar con ella las imágenes con que trabajarán nuestros equemas.
- Imagefile: Que facilita la lectura de imágenes almacenadas en el disco duro.

La versión utilizada de JdeRobot ha sido la 4.3.0, lanzada en abril de 2009, y que se compone de 17 esquemas y 12 drivers.

4.2.1. Calibrador

Un esquema de JdeRobot que ha servido como apoyo a este proyecto es Calibrador, documentada en [Kachach, 2008]. La utilidad de Calibrador es calcular el modelo de proyección de la cámara utilizada. Calibrar las cámaras es importante ya que a la hora de

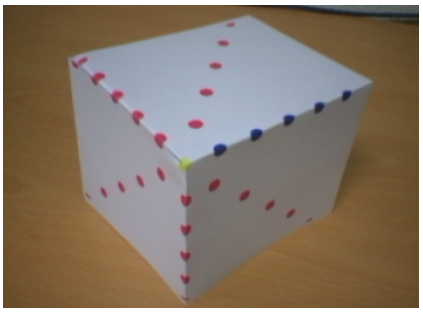
procesar las imágenes con información 3D, pequeñas variaciones en la función de proyección de la cámara pueden acarrear importantes errores.

Las cámaras del mercado vienen de fábrica con unos parámetros ligeramente distintos de los nominales. Las cámaras de foco fijo utilizadas en este proyecto son el caso más simple ya que estos parámetros no varían con el tiempo, así que basta con calibrar la cámara una vez y guardar en un archivo los parámetros.

Sin embargo, cuando se utilizan cámaras de foco variable, la cámara debe recalibrarse cada vez que se mueva el foco, debido a los errores de posición de los motores que mueven las lentes.

La aplicación Calibrador permite calcular los parámetros lineales de la cámara según el modelo pinhole descrito en el capítulo anterior. El procedimiento de calibración consiste en apuntar a un patrón conocido (ver figura) y asegurarse de que la correspondencia entre los puntos del espacio y los puntos de la imagen es correcta (bien porque el esquema los detecte automáticamente o porque los introduzcamos manualmente). Una vez hecho esto, el programa calcula la función de proyección y la representa de forma matricial. Una vez estimada la matriz de proyección M, se descompone en producto de 3 matrices: K (intrínseca), R (rotación) y T (traslación).

En este proyecto haremos uso de la matriz K, que es inmune a los cambios de posición y orientación de la cámara. Las funciones de rotación y traslación son las que estimará nuestro esquema sin necesidad de tener el patrón de calibración delante.



(a)

Figura 4.2: Patrón de calibración.

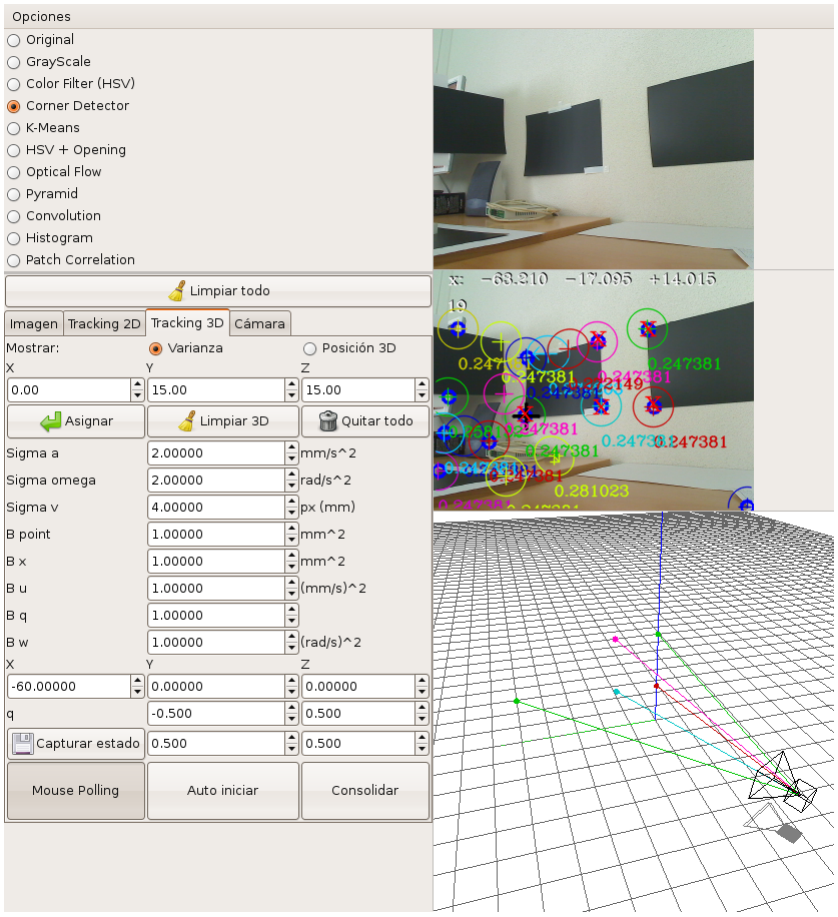
4.3. Biblioteca GTK para interfaces gráficas

La función principal de la interfaz gráfica de nuestro proyecto es la visualización de los datos de salida del algoritmo de autolocalización así como de sus pasos intermedios, indispensables para detectar fallos y asegurar la robustez del sistema. También ha servido para realizar el ajuste de los parámetros de los algoritmos utilizados, ya que no se disponía de una herramienta previa para analizar las propiedades estadísiticas de las imágenes utilizadas.

GTK+ es un conjunto de bibliotecas multiplataforma para crear interfaces gráficas de usuario en múltiples lenguajes de programación como C, C++, Java, Python, etc. GTK+ es software libre bajo licencia LGPL y es parte del proyecto GNU. Entre las bibliotecas que componen a GTK+, destaca GTK, que es la que realmente contiene los objetos y funciones para la creación de la interfaz de usuario.

Son numerosas las aplicaciones desarrolladas con esta librería, algunas muy conocidas como el navegador web Firefox o el editor gráfico GIMP, por lo que puede comprobarse su gran potencia y estabilidad.

En nuestro proyecto, hemos utilizado GTK+ para la realización de las interfaces gráficas, ayudándonos del programa de diseño de interfaces Glade para simplificar el desarrollo, y cuyo aspecto puede verse en la figura 4.3.



(a)

Figura 4.3: Interfaz gráfica de nuestro proyecto.

4.4. OpenGL

OpenGL es una especificación estándar que define una API multiplataforma e independiente del lenguaje de programación para desarrollar aplicaciones con gráficos en 2D y 3D. A partir de primitivas geométricas simples, como puntos o rectas, permite generar escenas tridimensionales complejas. Actualmente es ampliamente utilizado en realidad virtual, desarrollo de videojuegos (figura 4.4) y en multitud de representaciones científicas.

Hemos utilizado esta API en nuestras aplicaciones para mostrar una representación en 3D simplificada de la cámara utilizada y su campo de visión (representado como una pirámide) para demostrar que la autolocalización es correcta.

(a)

Figura 4.4: Captura del videojuego Counter Strike, desarrollado en OpenGL.

4.5. OpenCV

OpenCV es una librería de visión artificial desarrollada inicialmente por Intel, que actualmente es código abierto y está publicada sobre la licencia BSD, lo que permite su libre utilización en propósitos comerciales o de investigación.

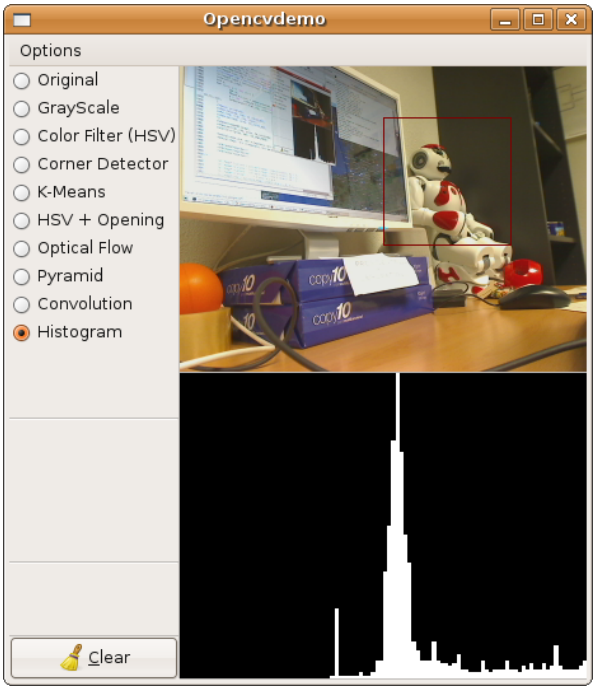
Esta librería nos proporciona un extenso conjunto de funciones para trabajar con visión artificial cuyo desarrollo se ha realizado primando la eficiencia, para lo que se ha programado utilizando C y C++ optimizados, pudiendo además hacer uso del sistema de primitivas de rendimiento integradas en los procesadores Intel (IPP), que son un conjunto de rutinas de bajo nivel específicas en estos procesadores y que cuentan con una gran eficiencia.

La documentación de esta librería puede encontrarse en la red ampliamente detallada y continuamente actualizada en forma de wiki [2].

Para tomar un primer contacto con el procesamiento de imágenes y la visión artificial, realizamos prácticas con Opencvdemo (parte del proyecto [?]), un esquema didáctico de JdeRobot que utiliza algunas funciones de la librería OpenCV. La finalidad del esquema es mostrar el potencial de la librería OpenCV y sus casos típicos de uso, utilizando las funciones que incorpora para tareas de visión artificial: transformación y análisis de la imagen y representación de la información sobre la imagen original.

Sobre este esquema hemos puesto en práctica las siguientes técnicas(ver figura): Detector de esquinas, clustering k-medias, segmentación mediante blobs, filtros morfológicos, uso de regiones de interés (ROI), cálculo del histograma y correlación entre parches de imagen.

Dentro de estas funcionalidades, algunas de ellas han sido utilizadas en la aplicación final, por lo que vamos a detallar en las siguientes secciones las más importantes.



(a)

Figura 4.5: Esquema Opencvdemo.

[2]http://opencv.willowgarage.com/documentation/index.html

4.5.1. Regiones de interés

La función `cvSetImageROI` permite seleccionar una subregión rectangular de la imagen que se está procesando. El rectángulo seleccionado se trata como una imagen independiente. La mayoría de las funciones de openCV soportan el uso de ROIs y eso se traduce en una disminución drástica de la carga computacional puesto que el procesamiento no se realiza sobre toda la imagen, sino sólo sobre este rectángulo. Por ejemplo, para el caso de la correlación de parches, típicamente se computa sobre unos pocos cuadrados de 15x15 dispersos a lo largo de la imagen. Basta cambiar el tamaño del rectángulo definido en la llamada a `cvSetImageROI` para pasar a procesar parches de distinto tamaño.

4.5.2. Detector de esquinas

Esta función de OpenCV realiza la extracción de características de la imagen. Este proceso ofrece a su salida las coordenadas gráficas de las "esquinas fuertes" de la imagen, aquellas que se espera correspondan a puntos de interés de la imagen. Estas esquinas son puntos de alta derivada espacial en la componente de luminancia de la imagen. El algoritmo calcula para cada punto de la imagen la matriz de gradiente de un parche cuadrado centrado en dicho punto y calcula los autovalores de esta matriz. El parámetro que determina la calidad de una esquina es el mínimo de los autovalores.

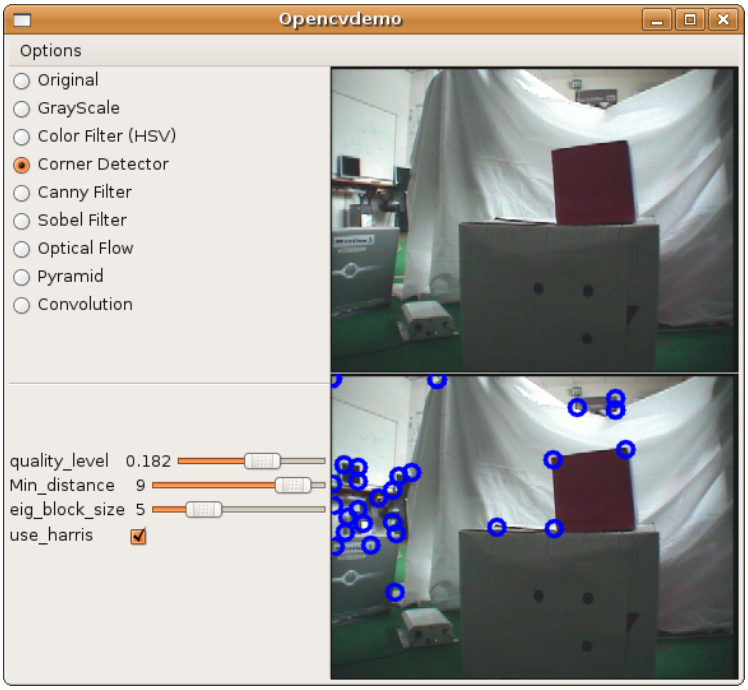
```
void cvGoodFeaturesToTrack(const CvArr* image, CvArr* eigImage,
CvArr* tempImage, CvPoint2D32f* corners, int* cornerCount,
double qualityLevel, double minDistance, const CvArr* mask=NULL,
int blockSize=3, int useHarris=0, double k=0.04);
```

La función devuelve, como máximo, tantos puntos como indique el valor `cornerCount`, garantizando una calidad y una distancia mínima entre ellos (dadas por los parámetros de entrada `qualityLevel` y `minDistance`).

4.5.3. Correlación de parches

Esta función realiza el cálculo de la característica basada en apariencia utilizada por A. Davison en su tesis doctoral (davison-phd) que constituye una medida de la divergencia entre dos parches monocromáticos de igual tamaño.

$$C = \sum_{parche} \frac{\left[\frac{g_1-\bar{g_1}}{\sigma_1} - \frac{g_0-\bar{g_0}}{\sigma_0}\right]^2}{n_{pixels}} \tag{4.1}$$

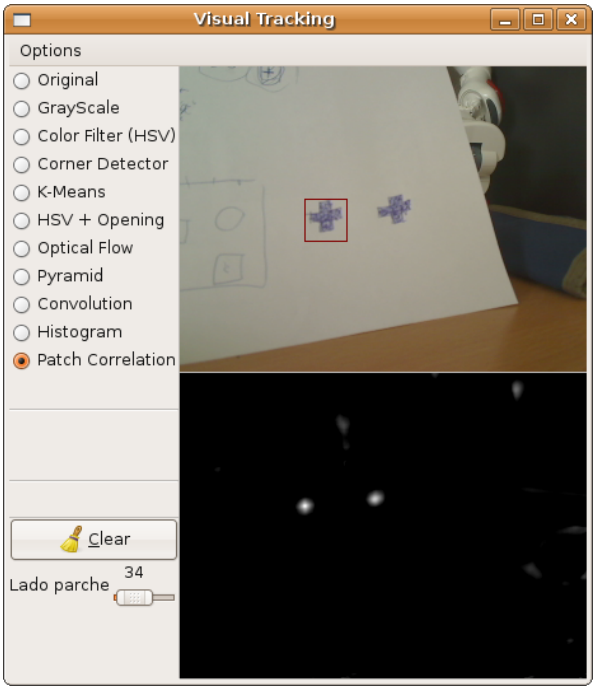


(a)

Figura 4.6: Detector de esquinas de OpenCV.

donde g_1, g_0 son valores de luminancia en posiciones homólogas de los dos parches; $\bar{g}_1, \bar{g}_0$ son los valores medios, y σ_1, σ_0 las desviaciones típicas de la luminancia a lo largo de los parches.

Parches idénticos producen una salida de valor 0, este valor aumenta cuanta mayor sea la diferencia entre los parches. Parches totalmente distintos producen salidas por encima de 1. Una ventaja de esta medida es que está normalizada respecto a cambios generales de intensidad, dado que estima mejor la correspondencia a lo largo de períodos largos de operación del sistema, en los que la luminosidad del escenario puede cambiar. Es más, la intensidad de la imagen percibida puede variar incluso dependiendo del ángulo con que miremos a una marca de posición.



(a)

Figura 4.7: Función de correlación de la imagen con el parche seleccionado.

Capítulo 5

Seguimiento 2D de puntos de interés

La estrategia elegida para llegar a un algoritmo de localización 3D robusto consiste en dividir el problema en 2 partes: primero, conseguiremos una estimación de la proyección de los puntos de referencia en el plano imagen, libre de ruido y que empareje correctamente entre fotogramas, haciendo uso de filtros de Kalman. Después, apoyándonos en el seguimiento 2D, aplicaremos el algoritmo de autolocalización 3D.

En este capítulo se describe el algoritmo de seguimiento 2D, estableciendo el modelo dinámico usado para seguir un punto, a continuación se explica cómo se extiende a varios puntos y después se describe cómo se incorporan nuevos puntos al seguimiento y cómo se detectan las pérdidas. Por último, se explicarán algunos de los experimentos que se han realizado en torno a este sistema.

5.1. Seguimiento 2D de un objeto

Comenzaremos describiendo cómo se usa un filtro de Kalman para seguir un objeto en movimiento. Para este primer paso, asumiremos lo siguiente: que disponemos de una medida ruidosa de las coordenadas del objeto que queremos seguir, y que además se nos garantiza que la medida corresponde al objeto en cuestión. Es importante saber si la medida realizada corresponde o no al objeto de interés ya que muchas veces podemos estar ante falsos positivos o, como se verá más adelante, tener varios datos disponibles y no saber cuál de ellos es la medida que nos interesa. Si no tuviésemos cierta seguridad de que la medida está bien asociada, no podríamos asumir que el ruido es gaussiano con media cero y el filtro de Kalman no sería de utilidad.

5.1.1. Modelo de Velocidad Constante

Para modelar un objeto moviéndose en la imagen hemos utilizado el modelo conocido como de *Velocidad constante*. Esto significa que el paso de predicción del filtro de Kalman no introduce cambios en la velocidad, y que éstos se tratan como ruido del proceso. En otras palabras, se considera que la aceleración tiene distribución gaussiana (en este caso bidimensional, $\binom{ax}{ay}$) de media 0 y de matriz de covarianza conocida. En este modelo la aceleración, la velocidad y la posición en cada instante se relacionan mediante el siguiente diagrama:

$$\begin{array}{ccccc} a & \rightarrow & v & & x \quad \text{instante t-1} \\ & & \downarrow & \searrow & \downarrow \\ a & \rightarrow & v & & x \quad \text{instante t} \\ & & \downarrow & \searrow & \downarrow \\ a & \rightarrow & v & & x \quad \text{instante t+1} \end{array}$$

El vector de estado comprende la posición y la velocidad estimada: $\begin{pmatrix} x \\ y \\ v_x \\ v_y \end{pmatrix}$

La posición se medirá en píxeles y la velocidad en píxeles por segundo. La aceleración no forma parte del vector de estado porque no obtenemos ningún beneficio al estimarla (es un proceso blanco, es decir, su valor en un instante no da ninguna información sobre el valor que tomará en el instante futuro).

Matriz de transición de estado corresponde a la función $x_{|t} = x_{|t-1} + v_{|t-1}\Delta t$:

$$F = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5.1}$$

Matriz de covarianza del ruido del proceso:

$$\begin{aligned} Q &= \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_a^2(\Delta t)^2 & 0 \\ 0 & 0 & 0 & \sigma_a^2(\Delta t)^2 \end{pmatrix} = G \cdot E\left[aa^T\right] \cdot G^T = \\ &= \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ \Delta t & 0 \\ 0 & \Delta t \end{pmatrix} \begin{pmatrix} \sigma_a^2 & 0 \\ 0 & \sigma_a^2 \end{pmatrix} \begin{pmatrix} 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & \Delta t \end{pmatrix} \end{aligned} \tag{5.2}$$

El vector de observación consta sólo de la posición del punto en la imagen:

$$z = \begin{pmatrix} x \\ y \end{pmatrix}$$

y la matriz de observación es de la forma

$$H = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{5.3}$$

El ruido de observación tiene una componente inevitable de ruido de cuantificación debido al carácter discreto de las coordenadas de la imagen, que se modela como una uniforme entre -0,5 y 0,5 , cuya desviación típica es de 0,29 píxeles. Además existen otras fuentes de ruido como distorsiones e interferencias en la imagen, y las debidas al desenfoque y la codificación de la imagen. También se desea filtrar el ligero temblor que introduce la persona que sujeta la cámara. En la práctica se modela la varianza del ruido de observación σ_v^2 con un valor más grande (del orden de 1 píxel).

Matriz de covarianza del ruido de observación:

$$R = \begin{pmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_v^2 \end{pmatrix} \tag{5.4}$$

Rechazos

Si la medida no está disponible o sabemos que es errónea, no podemos calcular el residuo ni ejecutar el paso de actualización del filtro de Kalman. Sin embargo, sí que podemos ejecutar el paso de predicción del filtro, con el consiguiente aumento de la incertidumbre estimada. En nuestro modelo de *velocidad constante*, un rechazo equivale a suponer que

la aceleración en este instante ha sido 0, y que el objeto en seguimiento ha continuado su trayectoria con la misma velocidad que tenía cuando fue observado correctamente por última vez. Esto nos permitirá defendernos frente a oclusiones.

5.2. Seguimiento 2D de múltiples puntos

Para afrontar el seguimiento de múltiples puntos de interés, usaremos varios filtros de Kalman independientes. En cada iteración se procederá a aplicar el detector de esquinas de OpenCV, a continuación se hará el paso de predicción de los filtros de Kalman, después un proceso de asociación esquina-filtro, y por último se ejecutará el paso de actualización de los filtros (si procede).

Al comienzo de la ejecución del esquema se crea un *pool* en el que reservaremos los recursos necesarios para la ejecución en paralelo de tantos filtros de Kalman como puntos queramos estar siguiendo, como máximo, en un momento determinado.

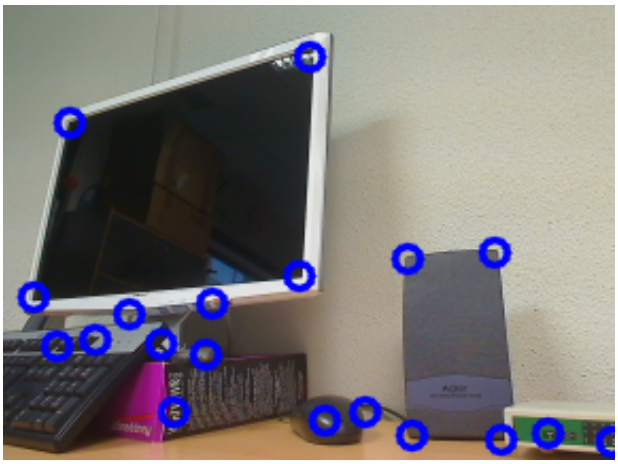
5.2.1. Extracción de características

En cada iteración de la aplicación, se realiza una extracción de características de la imagen por medio de la función de OpenCV `cvGoodFeaturesToTrack` descrita en la sección 4.5.2. El resultado de esta función es un conjunto desordenado de puntos $\binom{x}{y}$ de la imagen que consideraremos candidatos a ser el vector de observación del punto que es objetivo del seguimiento.

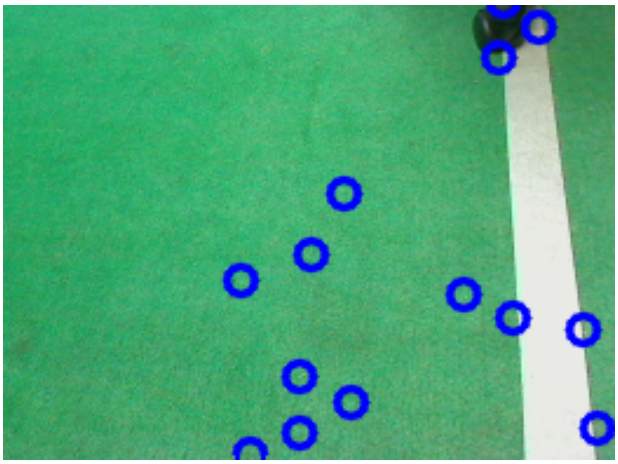
En la figura se puede ver el resultado de la extracción. La imagen 5.1(a) corresponde a un entorno fuertemente estructurado, en el que la detección de esquinas tiene éxito en la mayoría de los casos. Aún así se detectan falsas esquinas (por ejemplo, en el brillo del ratón o en las letras) que tendremos que ser capaces de descartar. En cambio, la imagen 5.1(b) es de un entorno hostil en el que no existen puntos de apoyo fiables. El algoritmo devuelve muchas falsas esquinas que no son persistentes a lo largo del tiempo ya que la imagen de esa superficie es ruidosa.

5.2.2. Asociación de las medidas a los filtros de seguimiento

Los puntos que hemos obtenido en la fase de extracción de características no podemos utilizarlos directamente como observaciones para los filtros de Kalman, ya que no sabemos



(a)



(b)

Figura 5.1: (a) Detección de esquinas exitosa. (b) Detección de esquinas fallida.

qué candidato corresponde a qué filtro. Es necesario un paso intermedio entre la extracción de características y la ejecución de los filtros de Kalman, que hemos llamado etapa de asociación o *emparejamiento*. El procedimiento establecido para la búsqueda del mejor candidato es el siguiente:

1. Se establece un área de búsqueda de forma elíptica alrededor del punto obtenido del paso de predicción del filtro de Kalman (ver sección *Regiones de confianza*).
2. Para cada punto que esté dentro del área de búsqueda, se calcula la distancia de Mahalanobis[1] al centro del área de búsqueda según la matriz de covarianza P del filtro de Kalman.
3. Se extrae un parche cuadrado de imagen en torno a dicho punto y se calcula la función de divergencia con el parche que se guardó la última vez asociado al punto objetivo.
4. Se ignoran los candidatos cuya divergencia exceda un determinado umbral.
5. Si no queda ningún candidato, se considera un rechazo. No se hará el paso de corrección del filtro de Kalman.
6. Se calcula el coste asociado a cada candidato como

   $(1 - P) * distancia^2 + P * divergencia^2$
7. Se establece el vector de observación para el paso de corrección como el del candidato cuyo coste sea mínimo.
8. Se extrae un nuevo parche de imagen en torno a la posición corregida y se guarda en memoria asociado al filtro.

Modelo cinético avanzado

Para evitar la confusión entre objetos que se mueven a cierta velocidad por el plano imagen, hemos introducido una modificación en el modelo cinético cuyo principal efecto es deformar las elipses de error en la dirección en la que se mueve el objeto. Para ello, nos mantenemos en el modelo de velocidad constante, pero ahora la aceleración pasa a ser una variable dependiente de la velocidad. En concreto, se espera que la componente principal de la aceleración tenga la misma dirección que la velocidad.

[1] Comparar entre distancias Mahalanobis equivale a comparar verosimilitudes sin tener que evaluar la función densidad de probabilidad de la densidad de predicción.

Este modelo es de utilidad en aquellas secuencias de vídeo en las que los objetos que se quiere seguir tienen un movimiento aproximadamente rectilíneo. En el experimento de la sección 5.4.1 en el que se aplica el seguimiento a una escena con movimientos rápidos pero rectilíneos, se redujo drásticamente el número de pérdidas al aplicar este modelo.

Regiones de Confianza

A la hora de representar por pantalla el estado del seguimiento, los datos que más nos interesa conocer son la posición estimada y la incertidumbre con que se ha hecho esa estimación. La incertidumbre viene dada por la matriz de covarianza P del filtro de Kalman y una manera cómoda de representarla es dibujar la elipse de incertidumbre.

Def. Elipse de incertidumbre (o elipse de error): Dada una distribución gaussiana bidimensional, es la región de $\mathbb{R}^2$ en la que una muestra de la distribución cae con probabilidad 0,394. También se define como el lugar geométrico de los puntos cuya distancia de Mahalanobis a la media de la distribución es igual a 1. Si se busca una región de aceptación del 95 % basta con multiplicar los ejes de la elipse por un factor de 2,447.

Para representar la elipse con funciones de trazado de Opencv necesitamos conocer sus parámetros: dirección del eje mayor y longitudes de los ejes. Para ello se hace una descomposición espectral de la matriz de covarianza, de la forma $P = U\,\Lambda\,U^T$, donde U es una matriz unitaria de autovectores de P y Λ es una matriz diagonal que contiene los autovalores de P.

Los autovectores, que forman una base ortonormal en $\mathbb{R}^2$, corresponden a direcciones de los ejes de la elipse. La raíz cuadrada de cada autovalor asociado a cada autovector es la longitud de cada eje respectivo de la elipse.

5.3. Base dinámica de filtros de seguimiento

Nuestra aplicación necesita tener continuamente varios puntos localizados y en seguimiento. Además los puntos entrarán y saldrán del marco de la imagen de forma impredecible así que debemos modelar la aparición y desaparición de las marcas como un proceso aleatorio de nacimiento y muerte.

Situaciones típicas que se dan en el seguimiento de múltiples puntos

Como si hiciéramos un análisis de riesgos, vamos a exponer los eventos típicos que pueden darse en un caso de seguimiento. Partimos del caso trivial, en el que tenemos n objetos en seguimiento, y en cada *frame* obtenemos n medidas fiables. Iremos paso a paso aumentando la complejidad del sistema.

- Aparición: Bien porque el campo de visión de la cámara abarque un nuevo objeto de interés, o porque ese objeto aparezca en escena (deja de estar ocluido, el entorno es cambiante, etc), pasaremos de tener n objetos a tener n+1. El nuevo objeto es persistente (no desaparece de inmediato).

- Desaparición: El campo de visión deja de abarcar un objeto existente, o éste desaparece (debido a una oclusión o porque deja de dar positivo en la función de interés), pasamos de tener n objetos a tener n-1. El objeto perdido no reaparece de inmediato.

- Parpadeo: Un objeto existente no tiene observación en un instante determinado, pero reaparece de inmediato. Esto puede deberse a un falso rechazo del emparejamiento o a un fallo de la función de detección de esquinas. Los parpadeos deben ignorarse, deseamos que el seguimiento tenga persistencia.

- Espurio o artefacto: Parece que ha aparecido un nuevo objeto pero en seguida se pierde, ya sea por una falsa medida o por un objeto que existe pero es fugaz. En ambos casos nos interesa ignorar estos espurios, en el primer caso son dañinos y en el segundo se consideran inútiles para el algoritmo de localización.

- Separación: Un objeto se divide en dos o más. Equivale a una aparición sólo que esta se produce muy cerca de un objeto existente, y durante un breve período de tiempo puede dar lugar a falsos positivos.

- Unión: Un objeto se "come" a otro (generalmente porque lo ocluye). Debemos evitar que dos filtros de seguimiento se refieran al mismo objeto.

- Cruce: Parece que se ha producido una unión y después una separación, cuando lo que ha pasado es que dos objetos se han cruzado o han pasado muy cerca el uno del otro, resultando que durante el cruce parecía que había uno solo. Lo ideal es que no se destruyera ningún filtro de seguimiento durante este instante.

- Salto: Una nueva observación de un objeto en seguimiento se produce lejos de la posición pronosticada. Este efecto no debería confundirse con una desaparición y una aparición simultáneas ya que el seguimiento se perdería.
- *Cambio de pareja*: dos objetos se han cruzado pero no ha dejado de haber dos observaciones, sin embargo los filtros de seguimiento se han intercambiado, dando lugar a dos medidas falsas y persistentes, y por lo tanto dañinas.

Una vez analizados los potenciales problemas que surgen en el seguimiento, empezamos eligiendo una política sencilla de creación y destrucción, y después iremos añadiendo heurísticos para corregir los problemas particulares que siga habiendo. La política elegida tiene las siguientes características:

1. Crear un nuevo filtro de seguimiento cada vez que se encuentre un nuevo objeto. Buscamos los nuevos puntos de interés en el resultado de ejecutar `cvGoodFeaturesToTrack` con un umbral de calidad alto.
2. Destruir un filtro cuando su valor de incertidumbre (la suma de los autovalores de la elipse de error) exceda un cierto umbral, ya que en ese momento consideramos que se ha perdido el rastro del objeto en seguimiento.

   Por el momento, con estas dos medidas tenemos cubiertos los casos de Aparición, Desaparición y Parpadeo.
3. Ignorar los filtros recién creados durante un breve período de tiempo (del orden de 10 iteraciones, 0,3 segundos). Esto nos inmuniza frente a los espurios.
4. Si dos filtros tienen su estimación de posición muy cerca (por debajo del umbral de distancia que se pasó a `cvGoodFeaturesToTrack`), se destruye el más nuevo. Esto nos permite manejar los casos de unión, que serán muy frecuentes cuando movamos la cámara en escenarios en los que se produzcan oclusiones.

Una vez corregidos estos aspectos, problemas derivados de casos como separación, salto y cambio de pareja se resuelven gracias al uso de características basadas en apariencia, en concreto la comparación entre parches.

5.4. Experimentos

5.4.1. Montaña Rusa

En este experimento se utilizó un vídeo procedente de Youtube en el que se muestran las imágenes de un recorrido en una montaña rusa. Este vídeo se ha elegido porque presenta muchas y diversas dificultades al contener trayectos de la cámara a distintas velocidades y por distintos escenarios, y nos permite poner a nuestro sistema al límite de su capacidad. Las conclusiones del experimento son:

- El modelo cinético avanzado resulta de mucha utilidad en este escenario ya que tenemos muchos puntos de interés que se desplazan por la imagen a gran velocidad con trayectorias paralelas. Cuanto más se exagera la excentricidad de las elipses de error, menor es el número de pérdidas y *cambios de pareja*. La excentricidad no se puede incrementar demasiado ya que eso provocaría confusión entre las esquinas de la barandilla (ver foto 5.2(d))

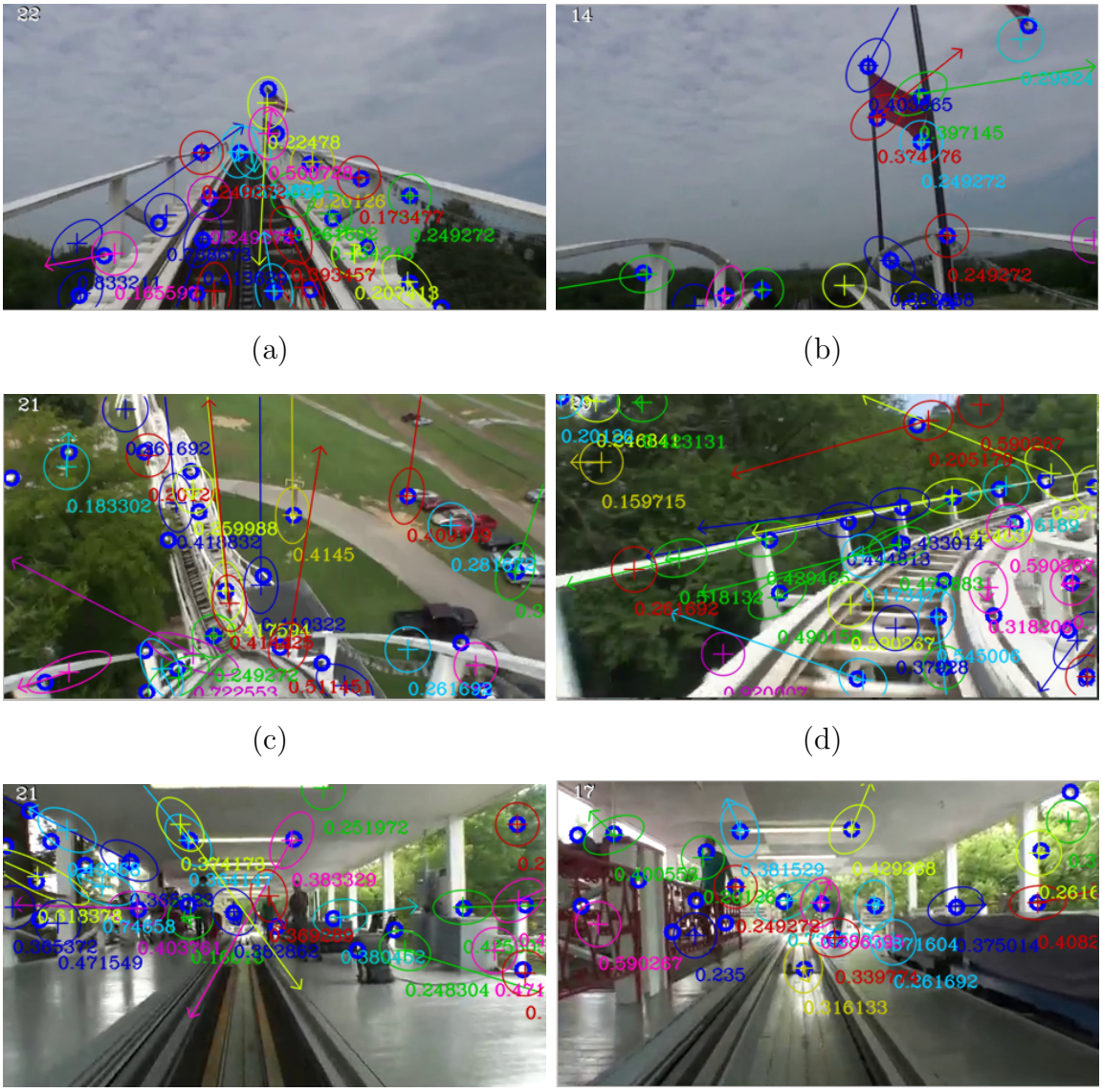
- Aún así, los puntos en seguimiento tienen en general un tiempo de vida muy corto y no tendrían utildad en el algoritmo de localización 3D para el que se quiere utilizar el seguimiento. De este experimento surgió la idea de ignorar los filtros de reciente creación.

- El seguimiento es mejor cuando la cámara pasa por la cima y por el punto de destino, llama la atención la calidad con que se siguen las banderas (5.2(a) y 5.2(b)) y los tubos fluorescentes del techo (fotos 5.2(e) y 5.2(f))).

- A pesar de la gran velocidad con que se movía la cámara, también tuvieron un seguimiento de calidad los puntos que estaban muy alejados de la cámara, como los coches de la fotografía 5.2(c).

- Las imágenes de árboles son especialmente ruidosas desde el punto de vista del seguimiento.

- Las nubes, a pesar de ser una parte de la imagen muy útil para la estimación de la orientación de la cámara, no se consideran puntos de interés por no ser esquinas.



(a) (b) (c) (d) (e) (f)

Figura 5.2: Prueba del seguimiento 2D sobre la secuencia de vídeo de la montaña rusa.

5.4.2. Entorno del laboratorio

En esta serie de experimentos se utilizó una cámara real, en concreto la *Quickcam*, conectada al mismo ordenador en el que se hizo el desarrollo software para comprobar la eficacia del algoritmo en el laboratorio de Robótica. Las conclusiones del experimento son:

- Cuanto más fuertes son las esquinas detectadas, mayor es la persistencia de los filtros y más robusto es el seguimiento. En este caso, las pantallas de PC y los marcos de las puertas y ventanas. Es, por tanto, un acierto dar prioridad a las esquinas que han obtenido mayor puntuación en el preprocesado, a la hora de crear nuevos filtros; ya que darán lugar a filtros de seguimiento más fiables.

- En los puntos anteriormente mencionados, consideramos que el seguimiento es lo bastante fiable como para realizar en él las pruebas del algoritmo de localización. Este resultado da luz verde a la segunda parte del proyecto.

- Las condiciones de iluminación afectan directamente a la calidad del seguimiento ya que la cámara que hemos utilizado (*Quickcam*) varía automáticamente su tiempo de exposición en función de la luminosidad percibida. Esto se traduce en que, cuando la iluminación es escasa y movemos la cámara rápidamente, la imagen *sale movida*, es decir, sufre una distorsión lineal que provoca fallos en el detector de esquinas y en la comparación de parches.

- Patrones como el de la figura 5.4.2 corren el riesgo de confundir al algoritmo, haciendo que detecte un objeto en movimiento cuando lo que hay en realidad son varios objetos idénticos colocados en línea. Este efecto negativo se desencadena cuando el detector de esquinas no capta el objeto en seguimiento pero sí el que está a su lado. Como son idénticos, la divergencia entre los parches es cercana a 0 y el sistema *cree* que el objeto se ha movido y actualiza la estimación de la velocidad. Esto hace que la predicción de la posición del objeto en el siguiente instante corresponda con el siguiente de la hilera, y el siguiente paso de actualización se hace con las coordenadas de este tercer punto, resultando un filtro de seguimiento deslizándose rápidamente a lo largo de la hilera de objetos.

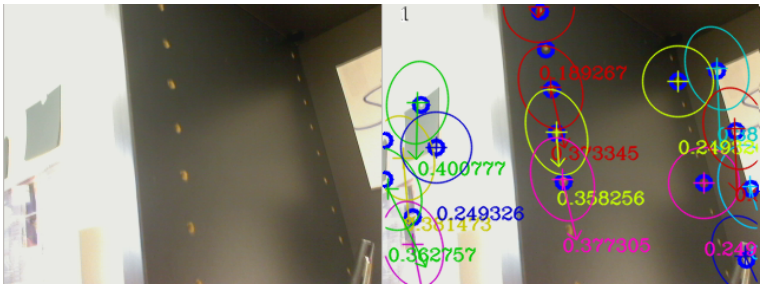
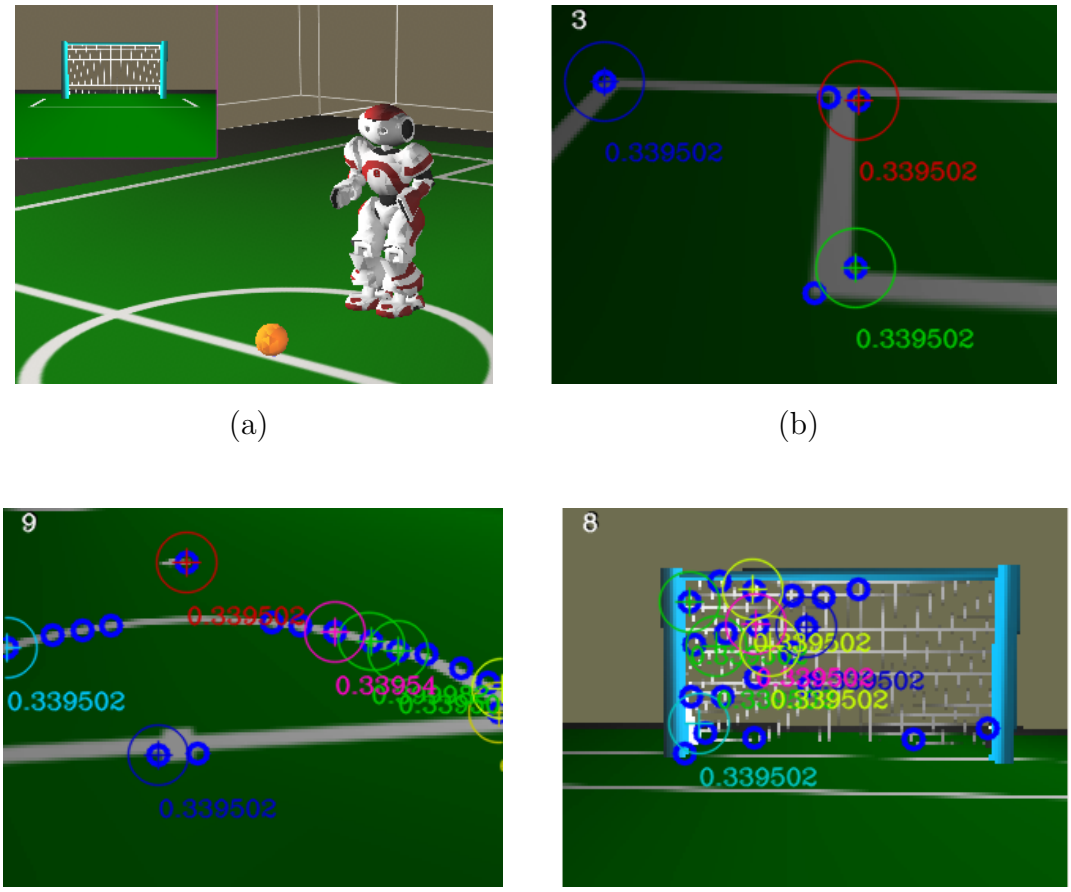


Figura 5.3: Patrón que confunde al algoritmo de seguimiento 2D

5.4.3. Robot Nao simulado

Dado que el algoritmo de seguimiento se ha introducido en un proyecto de visión sobre un robot Nao simulado en la aplicación Webots, vamos a explicar aquí los resultados obtenidos:

- Debido a la baja tasa de refresco de la interfaz de imagen del robot Nao, se tuvo que ampliar artificialmente el tamaño del área de búsqueda para obtener resultados positivos. Esto reduce drásticamente la precisión del seguimiento, y los filtros saltaban frecuentemente entre esquinas cercanas (ver figura 5.4(b)).
- Las imágenes del simulador son de baja calidad y las líneas del campo aparecen pixeladas. Como se puede ver en la imagen, se detectan como esquinas muchos puntos de la línea circular (ver figura 5.4(c)).
- La portería del campo virtual es una imagen hostil para nuestro algoritmo ya que tiene muchas esquinas muy juntas(ver figura 5.4(d)).



(a) (b)

(c) (d)

Figura 5.4: Evaluación del seguimiento 2D sobre el robot Nao en el simulador webots.

Capítulo 6

Localización 3D

Una vez descrito el algoritmo que realiza el seguimiento 2D de los puntos de interés en la imagen, pasamos a describir el algoritmo que estima la posición y orientación de la cámara en 3D y la posición en el espacio de los puntos de apoyo[1] que se obtienen de la imagen.

6.1. Diseño

Con la experiencia adquirida en el trabajo descrito en el capítulo 5, estamos en condiciones de refinar las especificaciones del sistema: buscamos una estimación ágil, es decir, flexible y en tiempo real, exclusivamente desde las imágenes de un mundo desconocido no ingenierizado: una vez programado y optimizado, el sistema no necesitará de ningún tipo de marcador para encontrar las coordenadas 3D de la cámara y de los puntos de apoyo, que seleccionará de forma *oportunista*, es decir, apoyándose en lo mejor entre lo que encuentre.

Conviene resaltar que pretendemos resolver un problema inverso en el que los datos disponibles son una función no invertible de las incógnitas (la función de proyección de una cámara es una aplicación de un espacio de dimensión 3 en un espacio de dimensión 2) por lo que en general nuestro problema puede tener infinitas soluciones. El uso del EKF nos permite escoger entre las soluciones posibles la más probable dada una serie de observaciones, constituyendo un enfoque bayesiano de la solución. Una vez entendido esto,

[1] A lo largo del capítulo nos referiremos a estos puntos como puntos de apoyo, puntos de referencia o *balizas*. En los artículos en inglés es común referirse a estos puntos como *landmarks*.

nos marcamos como objetivo llegar a una estimación razonable de la posición y orientación de la cámara a partir de un conocimiento a priori mínimo.

En este proyecto asumiremos que las cámaras utilizadas están perfectamente calibradas, aunque existe la posibilidad de incluir los parámetros intrínsecos de la cámara en el EKF para que se ajusten de manera automática al mismo tiempo que se hace la estimación de los parámetros extrínsecos. Dado que conocemos los intrínsecos, basta con darle al sistema las coordenadas exactas de 4 puntos para que fije la posición de la cámara. La práctica habitual es utilizar las esquinas de una hoja de papel DIN-A4 ya que se puede encontrar en cualquier parte y sus medidas son estándar (210 $\times$ 297 mm). El conjunto de observaciones de estos 4 puntos coplanarios y simétricos es compatible con varias posiciones y orientaciones de la cámara, por lo que al inicializar el sistema conviene dar también una estimación de la posición y orientación inicial para que las coordenadas de los nuevos puntos de apoyo sean coherentes con el escenario que se va a mapear.

6.2. EKF para estimación de posición 3D de la camara

En esta sección explicaremos cómo se usa el Filtro Extendido de Kalman para estimar la posición 3D de la cámara a partir de un conjunto de puntos cuya posición en el mundo 3D conocemos de antemano. Por ahora asumiremos que los puntos de apoyo son conocidos (con un nivel pequeño de incertidumbre), que contamos en todo momento con observaciones correctamente asociadas a cada uno y que por tanto ninguno escapa del marco de la imagen.

6.2.1. Modelo dinámico

En primer lugar describiremos el nuevo modelo utilizado, que incluye el modelo de movimiento de la cámara y el modelo de observación basado en el modelo pinhole de cámara descrito en la sección 3.1.1.

Los vectores de estado y observación tienen dimensiones variables, dado que ya no utilizamos N filtros para seguir N puntos, sino que todos están relacionados entre sí dentro de un EKF que los engloba a todos.

$$\text{Vector de estado: } \hat{x} = \begin{pmatrix} x_v \\ y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} \tag{6.1}$$

donde el sub-vector x_v es el vector de estado de la cámara (explicado a fondo en la sección 6.2.2) y cada vector y_i es el vector de 3 coordenadas del punto i.

$$\text{Vector de observación: } \hat{z} = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_N \end{pmatrix} \tag{6.2}$$

donde cada sub-vector z_i es la observación proporcionada por el algoritmo de seguimiento 2D, es decir, la posición del píxel en que se observa.

Mapa de covarianza

En implementaciones avanzadas del algoritmo monoSLAM, en las que se podría estar apoyando la localizacón en más de 100 puntos, la matriz de covarianza P adquiere dimensiones titánicas y no siempre se utiliza entera sino que se ejecuta el filtro extendido de Kalman sobre determinados subconjuntos de puntos, elegidos de tal manera que se maximice la información que contiene la nueva estimación. Por ello es frecuente hablar de *mapas de covarianza* en lugar de matrices de covarianza.

$$P = \begin{pmatrix} P_{x_v x_v} & P_{x_v y_1} & P_{x_v y_2} & \dots & P_{x_v y_N} \\ P_{y_1 x_v} & P_{y_1 y_1} & P_{y_1 y_2} & \dots & P_{y_1 y_N} \\ P_{y_2 x_v} & P_{y_2 y_1} & P_{y_2 y_2} & \dots & P_{y_2 y_N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ P_{y_N x_v} & P_{y_N y_1} & P_{y_N y_2} & \dots & P_{y_N y_N} \end{pmatrix} \tag{6.3}$$

6.2.2. Modelo de movimiento de la cámara

La cámara se modela como un cuerpo rígido cuya posición viene descrita por los parámetros de traslación y rotación. Mantenemos también estimaciones de su velocidad

lineal y velocidad angular, aplicando a ambas un modelo de *velocidad constante* similar al del capítulo anterior pero en 3D, para modelar un movimiento suave (sin acelerones ni giros bruscos).

x_v es el vector de estado de la cámara. Este vector es un sub-vector del vector de estado del sistema (ignoramos las posiciones de los puntos de apoyo):

$$x_v = \begin{pmatrix} r^W \\ v^W \\ q^{WR} \\ \omega^R \end{pmatrix} \tag{6.4}$$

donde r^W es la posición de la cámara en el mundo (dimensión 3), v^W es la velocidad con que se mueve (dimensión 3), el cuaternión q^{WR} (dimensión 4) expresa su orientación con respecto al sistema de coordenadas absolutas (ver sección3.2), y ω^R es un vector que indica la velocidad angular de la cámara. La dimensión total de x_v es 13.

El vector ω^R se interpreta de la siguiente manera: se considera que el centro de la rotación es el foco de la cámara; el eje de rotación viene dado por la dirección de ω^R y su módulo, $|\omega^R|$ es la velocidad angular en rad/s. La dirección de rotación sigue la regla de la mano derecha: si nuestro pulgar derecho apunta en la dirección de ω^R el resto de los dedos de la mano indican la dirección de giro. Si la cámara permanece un intervalo Δt rotando con velocidad ω^R, habrá girado un ángulo $\alpha = |\omega^R|\Delta t$ rad en torno al vector unitario $\frac{\omega^R}{|\omega^R|}$, y el cuaternión correspondiente será

$$q_g(\omega^R \Delta t) = \left(\cos\frac{\alpha}{2} \quad \frac{\omega_x^R}{|\omega^R|}\sin\frac{\alpha}{2} \quad \frac{\omega_y^R}{|\omega^R|}\sin\frac{\alpha}{2} \quad \frac{\omega_z^R}{|\omega^R|}\sin\frac{\alpha}{2} \right) \tag{6.5}$$

Consideramos que la aceleración angular $\Omega^R = \frac{d\omega^R}{dt}$ tiene distribución gaussiana de media 0 y matriz de covarianza conocida, acorde con el modelo de velocidad constante.

La función de predicción del modelo de la cámara queda de la siguiente manera:

$$x_{v|t} = f(x_{v|t-1}) = \begin{pmatrix} r^W_{|t-1} + v^W_{|t-1}\Delta t \\ v^W_{|t-1} \\ q_g(\omega^R_{|t-1}\Delta t) \times q^{WR}_{|t-1} \\ \omega^R_{|t-1} \end{pmatrix} \tag{6.6}$$

El símbolo $\times$ denota la conjugación de dos cuaterniones (ver sección 3.2) mediante el producto de Hamilton. El cuaternión $q(\omega^W \Delta t)$ es el definido por el vector de giro en el intervalo de tiempo Δt.

El ruido del proceso, la aceleración, es gaussiano blanco aditivo, y afecta sólo a las velocidades (lineal y angular), y además es isotrópica (los cambios de velocidad son equiprobables en todas las direcciones) por lo que la matriz Q queda:

$$Q = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \sigma_a^2 I_3 (\Delta t)^2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_\omega^2 I_3 (\Delta t)^2 \end{pmatrix} \tag{6.7}$$

donde I_3 es la matriz identidad 3×3 y $\sigma_a^2, \sigma_\omega^2$, las varianzas de la aceleración lineal y angular en cualquier dirección son parámetros del algoritmo que se han determinado mediante ensayo y error.

La función de predicción, f, es lineal en la parte de las traslaciones, pero es no lineal en la parte de las rotaciones, debido a la operación de conjugación de cuaterniones. Esta es una de las razones por las cuales nos vemos obligados a utilizar el filtro extendido de Kalman. En los cálculos se sustituirá la matriz F por la matriz jacobiana de f, JF:

Derivación de JF

Sea g la aplicación no lineal sobre $\mathbb{R}^4$ que consiste en hacer el producto de Hamilton de un cuaternión q con otro q_d por la derecha

$$\begin{array}{rccl} g\colon & \mathbb{R}^4 & \to & \mathbb{R}^4 \\ & q_{t-1} & \rightsquigarrow & q_{|t} = q_{|t-1} \times q_d \end{array} \tag{6.8}$$

Su matriz jacobiana depende de q_d y es

$$Jg(q_d) = \begin{pmatrix} a & -b & -c & -d \\ b & a & d & -c \\ c & -d & a & b \\ d & c & -b & a \end{pmatrix} \tag{6.9}$$

donde a b c d son las componentes del cuaternión q_d.

Si la multiplicación se hace por la izquierda por el cuaternión q_i, la jacobiana de la aplicación es $Jg^H(q_i)$. (esto se demuestra mediante derivación directa, ver producto de hamilton)

$$JF_{|t} = \begin{pmatrix} I_3 & \Delta t I_3 & 0 & 0 \\ 0 & I_3 & 0 & 0 \\ 0 & 0 & \frac{\partial q_{|t}}{\partial q_{|t-1}} & \frac{\partial q_{|t}}{\partial \omega} \\ 0 & 0 & 0 & I_3 \end{pmatrix} \tag{6.10}$$

$$\frac{\partial q_{|t}}{\partial q_{|t-1}} = Jg(q_g(\omega\Delta t)) \tag{6.11}$$

$$\frac{\partial q_{|t}}{\partial \omega} = \frac{\partial q_{|t}}{\partial q_g}\frac{\partial q_g}{\partial \omega} = Jg^H(q_{|t-1})\frac{\partial q_g}{\partial \omega} \tag{6.12}$$

$$\frac{\partial q_g}{\partial \omega} = \begin{pmatrix} \frac{-\Delta t \sin\beta}{2|\omega|}\omega^T \\ I_3\frac{\sin\beta}{|\omega|} + \frac{\frac{\Delta t}{2}\cos\beta - \frac{\sin\beta}{|\omega|}}{|\omega|^2}\omega\omega^T \end{pmatrix} \tag{6.13}$$

donde $\beta = \frac{|\omega|\Delta t}{2}$.

6.2.3. Modelo de observación

La función de observación es dependiente del número de puntos que se estén observando. Mostraremos en esta sección la jacobiana de la función de observación[2] para un solo punto, y su extensión a varios puntos se puede obtiener de inmediato por ser sencilla e intuitiva.

Derivación de JH

La función h se puede expresar como la composición de las funciones de proyección, rotación y traslación: $x = \begin{pmatrix} x_v \\ y \end{pmatrix}$

$$\begin{pmatrix} u \\ v \end{pmatrix} = h\begin{pmatrix} x_v \\ y \end{pmatrix} = K\left(R\left(T\begin{pmatrix} x_v \\ y \end{pmatrix}\right)\right) \tag{6.14}$$

y, por la regla de la cadena, su jacobiana es:

$$JH = J_{KRT}(x) = JK(R(T(x))\ JR(T(x))\ JT((x)) \tag{6.15}$$

$$\begin{array}{rccc} K: & \mathbb{R}^3 & \to & \mathbb{R}^2 \\ & \begin{pmatrix} x^{CR} \\ y^{CR} \\ z^{CR} \end{pmatrix} & \rightsquigarrow & \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} u_0 - \frac{f_x x}{z} \\ v_0 - \frac{f_y y}{z} \end{pmatrix} \end{array} \tag{6.16}$$

[2]Nota: ya que todos los miembros de la función de observación se refieren al instante t, a lo largo de este apartado evitaremos escribir $|_t$ en todas las variables

$$JK = \begin{pmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} & \frac{\partial u}{\partial z} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} & \frac{\partial v}{\partial z} \end{pmatrix} = \begin{pmatrix} -\frac{f_x}{z} & 0 & \frac{f_x}{z^2} \\ 0 & -\frac{f_y}{z} & \frac{f_y}{z^2} \end{pmatrix} \tag{6.17}$$

$$\begin{array}{rccl} R\colon & \mathbb{R}^7 & \to & \mathbb{R}^3 \\ & \begin{pmatrix} x^C \\ q^{WR} \end{pmatrix} & \rightsquigarrow & x^{CR} = R_q(q^{WR})x^{CR} \end{array} \tag{6.18}$$

$$JR = \begin{pmatrix} \frac{\partial x^{CR}}{\partial x^C} & \frac{\partial x^{CR}}{\partial q^{WR}} \end{pmatrix} = \begin{pmatrix} R_q(q^{WR}) & x^C\nabla_q x + y^C\nabla_q y + z^C\nabla_q z \end{pmatrix} \tag{6.19}$$

$$\begin{array}{rccl} R\colon & \mathbb{R}^{10} & \to & \mathbb{R}^7 \\ & \begin{pmatrix} x^V \\ q^{WR} \\ y^W \end{pmatrix} & \rightsquigarrow & \begin{pmatrix} x^C \\ q^{WR} \end{pmatrix} = \begin{pmatrix} y^W - x^V \\ q^{WR} \end{pmatrix} \end{array} \tag{6.20}$$

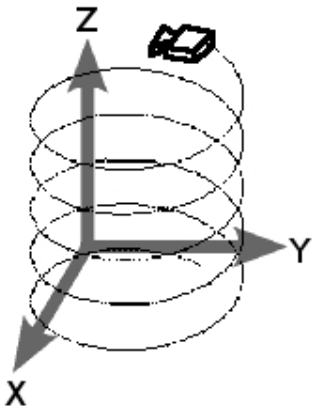
$$JT = \begin{pmatrix} -I_{3\times3} & 0_{3\times4} & I_{3\times3} \\ 0_{4\times3} & I_{4\times4} & 0_{4\times3} \end{pmatrix} \tag{6.21}$$

6.3. Base dinámica de puntos de apoyo

Durante la ejecución típica de nuestra aplicación, la cámara se mueve continuamente, perdiendo de vista algunos puntos y entrando otros nuevos en escena, llegando al extremo de giros de 180 grados (ver figura 6.1(a)) que tienen como consecuencia que damos la espalda a todos los puntos que veíamos anteriormente. El algoritmo debe ser capaz de apoyarse en cualquier conjunto de puntos, no siendo ninguno indispensable para la correcta estimación de la posición y orientación de la cámara.

Sea cual sea el número de puntos de apoyo presentes, la matemática sigue siendo la misma, sólo que el tamaño de las matrices involucradas aumenta y disminuye en función de la información disponible. La complejidad computacional crece con el tamaño de las matrices, pero a cambio la precisión del sistema aumenta. El tratamiento de la base dinámica de puntos de apoyo de esta parte se hace más sencilla que en el capítulo anterior, precisamente gracias al trabajo realizado en el bloque de procesamiento 2D, que nos provee de un conjunto ordenado de puntos en seguimiento, con lo que el emparejamiento punto-observación ya está hecho de antemano.

Cuando se produce una pérdida en el seguimiento 2D, se notifica al algoritmo de localización para que la elimine de su vector de estado y de su mapa de covarianza. Ya



(a)

Figura 6.1: Trayectoria de la cámara que hace que todos los puntos de referencia se pierdan y se sustituyan por otros nuevos.

que la base de puntos de apoyo está implementada sobre una lista dinámica de la librería estándar de C++, basta con ejecutar la función que elimina un elemento de una lista, y ésta mantiene su orden. En el mapa de covarianzas, al ser una matriz, se deben eliminar la fila y la columna correspondientes a las covarianzas cruzadas de dicho punto con la cámara y todos los demás. Como las matrices utilizadas son estáticas, se debe recomponer la matriz del siguiente modo:

$$\begin{pmatrix} P_{x_v x_v} & \dots & P_{x_v y_{i-1}} & P_{x_v y_{i+1}} & \dots \\ \vdots & \ddots & \vdots & \vdots & \dots \\ P_{y_{i-1} x_v} & \dots & P_{y_{i-1} y_{i-1}} & P_{y_{i-1} y_{i+1}} & \dots \\ P_{y_{i+1} x_v} & \dots & P_{y_{i+1} y_{i-1}} & P_{y_{i+1} y_{i+1}} & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

El subíndice i es el lugar que ocupaba en la lista el punto de referencia eliminado.

Cuando se añade un nuevo punto a la base, se inserta siempre al final de la lista. Basta con yuxtaponer la posición estimada del punto por debajo en el vector de estado, y ampliar el mapa de covarianza de la siguiente manera:

$$P = \begin{pmatrix} P_{x_v x_v} & P_{x_v y_1} & P_{x_v y_2} & \dots & P_{x_v y_N} & 0_{13\times 3} \\ P_{y_1 x_v} & P_{y_1 y_1} & P_{y_1 y_2} & \dots & P_{y_1 y_N} & 0_{3\times 3} \\ P_{y_2 x_v} & P_{y_2 y_1} & P_{y_2 y_2} & \dots & P_{y_2 y_N} & 0_{3\times 3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & 0_{3\times 3} \\ P_{y_N x_v} & P_{y_N y_1} & P_{y_N y_2} & \dots & P_{y_N y_N} & 0_{3\times 3} \\ 0_{3\times 13} & 0_{3\times 3} & 0_{3\times 3} & 0_{3\times 3} & 0_{3\times 3} & P_{**} \end{pmatrix} \tag{6.22}$$

donde P_{**} es la covarianza estimada del punto recién insertado. El filtro extendido de Kalman calculará automáticamente las covarianzas cruzadas del nuevo punto con el vector de la cámara y el resto de puntos.

6.3.1. Inicialización de nuevos puntos de referencia

Una vez caracterizado el problema de la autolocalización a partir de un cierto número de puntos conocidos, inmediatamente surge la necesidad de conocer la situación en el espacio de los puntos que entran en el campo de visión de la cámara según se mueve ésta. Desde el establecimiento de la línea de investigación del SLAM basado en EKF se han manejado diversas estrategias para el cálculo de nuevas balizas o puntos de referencia:

- La primera aproximación es modelar la distribución del punto buscado como una gaussiana 3-dimensional cuya dirección de máxima variación es la del rayo que parte de la cámara en la dirección de la observación retroproyectada[3]. Pronto se demuestra que una distribución gaussiana no es adecuada para esta medida. Intuitivamente: modelarlo como una gaussiana equivale a suponer que es igual de probable que el punto esté detrás de la cámara que esté *tan solo* más allá del doble de distancia de la cámara de lo que lo estaba la hipótesis inicial; y evidentemente esa asunción no es correcta.

- Una segunda aproximación, empleada por Davison en su investigación a partir de [Davison, 2003] está basada en un filtro de partículas, que es lo que típicamente se utiliza en lugar del filtro de Kalman si la componente de ruido de un sistema dinámico sigue una distribución no gaussiana. Las partículas se reparten uniformemente por el rayo de proyección, a lo largo de un rango de distancias entre 0,5 y 5 metros. Este

[3]La retroproyección de un punto de una imagen es el conjunto de todos los puntos del mundo cuya proyección es el punto dado. En el contexto en el que trabajamos, la retroproyección de un punto es siempre una recta.

sistema tiene éxito en SLAM en interiores, pero no es válido para exteriores en los que incluso podríamos encontrar puntos en el infinito (e.g. la luna y las estrellas). Un problema de la aproximación basada en filtros de partículas es que hasta que el punto no está debidamente localizado no contribuye a la estimación del estado de la cámara.

- [Montiel *et al.*, 2006] sostiene que si parametrizamos la profundidad d(la distancia del punto a la cámara) en el EKF como $\rho = \frac{1}{d}$, la incertidumbre de este parámetro ρ sí es gaussiana. Esto permite incluir la estimación del punto de referencia en el mapa desde la primera medida obtenida, así como considerar los puntos en el infinito.

- En este PFC se propone una versión de la parametrización $\rho = \frac{1}{d}$ que simplifica los cálculos, con la diferencia de que lo que se incluye en el EKF no es la inversa de la profundidad sino la del alejamiento z: $\tau = \frac{1}{z}$ (la proyección de d sobre el eje óptico de la cámara). Se plantea la hipótesis de que este sea más preciso.

Parametrización de la profundidad inversa

Primero explicaremos esta técnica de [Montiel *et al.*, 2006] para después proponer la versión propia del PFC, de parametrización del alejamiento inverso.

Redefinimos el vector de estado de cada baliza como el vector de dimensión 6

$$y_i = \begin{pmatrix} x_i \\ y_i \\ z_i \\ \theta_i \\ \phi_i \\ \rho_i \end{pmatrix}$$

que modela un punto 3D localizado en

$$\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} + \frac{1}{\rho_i} m(\theta_i, \phi_i)$$

Dejando libre el parámetro ρ_i, este vector de estado codifica el rayo que parte del centro óptico de la cámara en el instante en que fue avistado $\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}$, cuya dirección viene dada por el vector unitario $m(\theta_i, \phi_i)$, donde θ_i, ϕ_i son los ángulos de azimut y elevación. La profundidad del punto buscado a lo largo de este rayo es $d = \frac{1}{\rho_i}$.

Nuestra alternativa: parametrización del alejamiento inverso

El método anterior presenta algunas desventajas:

- Trabajar con ángulos implica usar funciones trigonométricas. Además del engorro que supone trabajar con ellas, por su alta no-linealidad, no son la mejor alternativa para meterlas en un EKF dado que la no-linealidad disminuye la optimalidad del EKF.
- Vuelve a existir el riesgo de *gimbal lock* cuando la elevación sea $\pm\frac{\pi}{2}$.
- La incertidumbre de los ángulos, aunque es pequeña, no tiene por qué ser gaussiana. De hecho, si asumimos un modelo pinhole en el que el ruido de la imagen es gaussiano, al pasar a ángulos desparece la gaussianidad.

Podemos evitar etos 3 problemas si redefinimos el vector de estado de la siguiente manera:

$$y_i = \begin{pmatrix} x_i \\ y_i \\ z_i \\ \alpha_i \\ \beta_i \\ \tau_i \end{pmatrix}$$

que modela un punto 3D localizado en

$$\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} + \frac{1}{\tau_i} R_q(q_i^*) \begin{pmatrix} \alpha_i \\ \beta_i \\ 1 \end{pmatrix}$$

La matriz de rotación es constante y permanece al margen del EKF. q_i es el cuaternión de la orientación de la cámara en el instante en el que el punto fue avistado. En esta ecuación aparece conjugado ya que la matriz de rotación que se calcula es la que pasa de un sistema de referencia solidario con la cámara al sistema de referencia absoluto, y el cuaternión que estima el EKF es precisamente el correspondiente a la rotación inversa (ver propiedad 3.5).

Igual que antes, dejando libre el parámetro τ_i, este vector de estado codifica el rayo que parte del centro óptico de la cámara en el instante en que fue avistado $\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}$; pero ahora la dirección viene dada por el vector $\begin{pmatrix} \alpha \\ \beta \\ 1 \end{pmatrix}$.

Esta formulación surge de despejar x e y en 3.1,

$$\begin{aligned} x &= \frac{(u-u_0)}{-f_x} z = \alpha z \\ y &= \frac{(v-v_0)}{-f_y} z = \beta z \end{aligned} \tag{6.23}$$

y nos facilita inmediatamente datos para inicializar el vector de estado y su covarianza:

$$\hat{y}_{i|t} = \begin{pmatrix} \hat{x}_{v|t} \\ \hat{r}^{WC}_{|t} \\ \frac{(u-u_0)}{-f_x} \\ \frac{(v-v_0)}{-f_y} \\ \frac{1}{2\,z_{min}} \end{pmatrix} \tag{6.24}$$

$$P_{|t} = \begin{pmatrix} Px_{v|t} & & 0_{13\times 6} & \\ & \begin{pmatrix} Pr_{x|t} & & 0_{3\times 3} \\ & \begin{pmatrix} \frac{\sigma_u^2}{f_x^2} & 0 & 0 \\ 0 & \frac{\sigma_v^2}{f_y^2} & 0 \\ 0 & 0 & \frac{1}{(4\,z_{min})^2} \end{pmatrix} \end{pmatrix} \\ 0_{6\times 13} & \\ & \begin{pmatrix} 0_{3\times 3} & \end{pmatrix} \end{pmatrix} \tag{6.25}$$

El valor inicial de $\hat{\tau}$, $\frac{1}{2\,z_{min}}$ y su varianza σ_τ^2 se han elegido de manera que la región de aceptación del 95 % cubra desde un alejamiento prudencial z_{min} hasta infinito.

Para obtener la jacobiana de la función de observación, JH, sólo hay que multiplicar la obtenida en la ecuación 6.15 por JD por la izquierda , cuya forma queda relativamente sencilla:

$$JD = \begin{pmatrix} I_{13\times 13} & 0_{13\times 3} & 0_{13\times 3} \\ 0_{3\times 13} & I_{3\times 3} & R_q(q_i^*) \cdot \begin{pmatrix} \frac{1}{\tau} & 0 & \frac{-\alpha}{\tau^2} \\ 0 & \frac{1}{\tau} & \frac{-\beta}{\tau^2} \\ 0 & 0 & \frac{-1}{\tau^2} \end{pmatrix} \end{pmatrix} \tag{6.26}$$

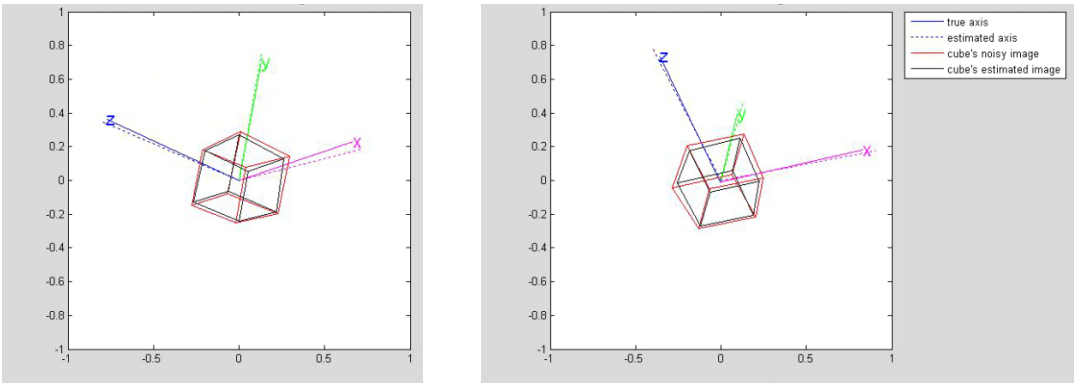
6.4. Experimentos

En esta sección se pretende validar experimentalmente el sistema diseñado y desarrollado. Además de su función demostrativa, los experimentos permiten ajustar los parámetros de los modelos y son una ayuda indispensable al proceso de depuración y corrección de errores.

6.4.1. Simulación Matlab

En primer lugar se materializó el algoritmo simplificado en MATLAB, poniéndolo a funcionar con datos simulados del movimiento de una cámara alrededor de un cubo durante 2 segundos.

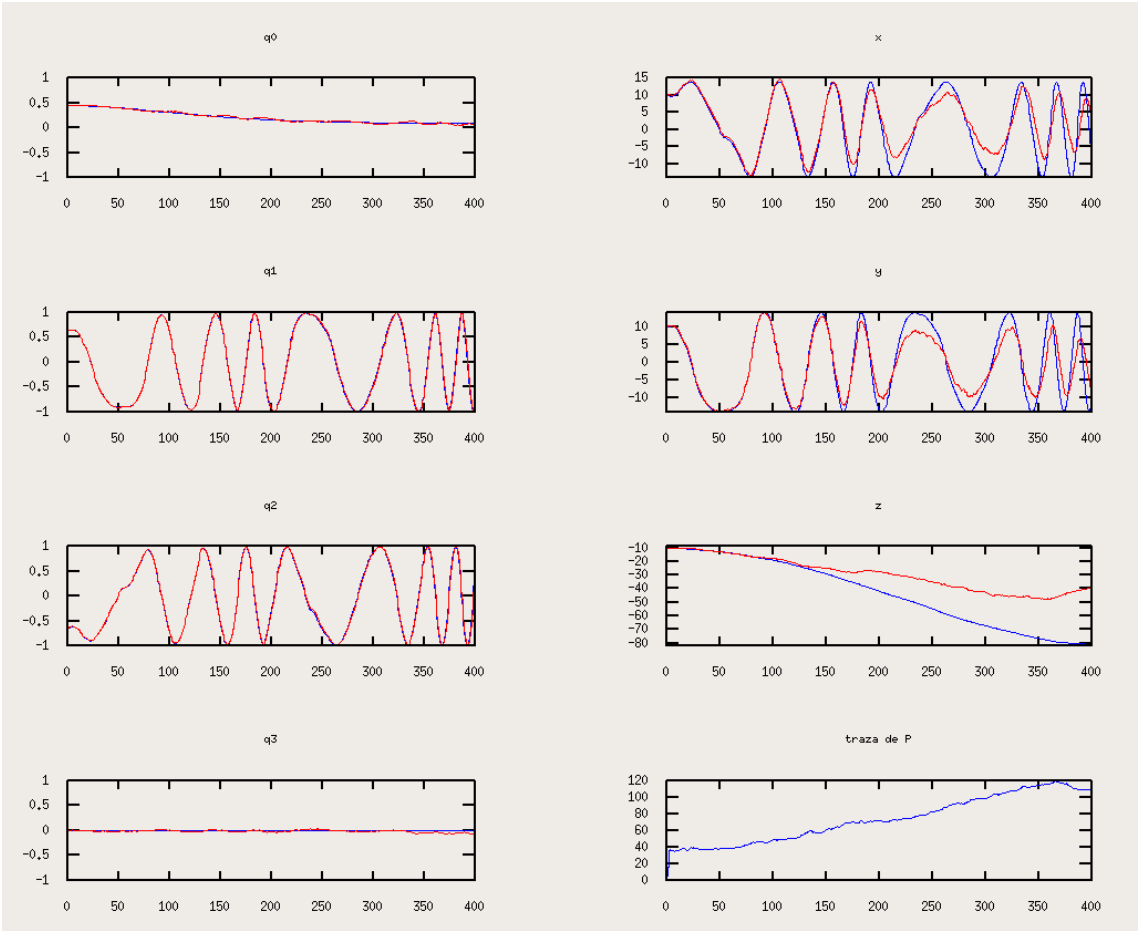
Los puntos de referencia establecidos eran los 8 vértices del cubo centrado en el origen y la cámara (pinhole ideal) realizaba un recorrido aleatorio por la superficie de un cilindro imaginario situado alrededor del cubo y sin perder éste de vista (ya que en la simulación el marco de la imagen era infinito y las proyecciones de los puntos que se encontraban en el plano focal desestabilizaban el sistema porque su función de observación devolvía infinito). Se representan las observaciones ruidosas del cubo, los ejes teóricos y los ejes estimados. Los resultados gráficos se trazaron fácilmente con la función plot de Matlab ya que las funciones de proyección necesarias para trazar gráficos en 3D están implementadas en el modelo del EKF.



(a) (b)

Figura 6.2: Resultados gráficos de la simulación en MATLAB en dos instantes distintos.

En este experimento se comprobó la viabilidad del sistema, pero se puso de manifiesto la necesidad de optimizar el código ya que el tiempo teórico de simulación era de 2 segundos y el tiempo de ejecución fue sensiblemente mayor, lo que significaba que estábamos lejos de satisfacer el requisito de tiempo real. Además se puede constatar que el error cometido en la estimación es mayor cuanto mayor es la distancia de la cámara a las balizas (debido al crecimiento del eje z en la figura 6.3(a)) ya que la potencia de la señal de las proyecciones 2D es más pequeña y el ruido modelado tiene una potencia independiente del tiempo, y por lo tanto disminuye la relación señal a ruido.



(a)

Figura 6.3: Resultados de la simulación en MATLAB en el dominio del tiempo.

6.4.2. Ejecución típica

El experimento principal de este proyecto, que constituye la prueba de concepto de los algoritmos diseñados, consiste en arrancar el algoritmo de localización 3D inicializándolo con la observación de una hoja de papel DIN-A4, con la posición de sus esquinas conocida.

El primer reto era que el algoritmo arrancase correctamente con la cámara quieta, con la hipótesis de partida en la posición exacta de la cámara, a 60 cm de distancia del folio. Esto no es trivial ya que cuando los parámetros del modelo estaban mal elegidos el filtro se desestabilizaba y se podía ver en el visor openGL cómo la posición estimada de la cámara hacía un recorrido sin sentido por el escenario.

Una vez ajustados los parámetros para conseguir esto, el siguiente objetivo fue afinar los parámetros para que la estimación fuese estable al mover la cámara.

Se observó que la estimación lleva un ligero retardo con respecto a la posición real de la cámara, que en el peor caso es del orden de 1 s (perceptible a simple vista) cuando se mueve la cámara rápidamente.

El error medio de la estimación de la posición con la cámara se ha medido con un resultado de unos 5 cm en la posición de inicio. El error de posición aumenta cuanto más nos alejemos del folio, ya que al reducirse la distancia entre las observaciones aumenta la relación señal a ruido de cuantificación de las medidas.

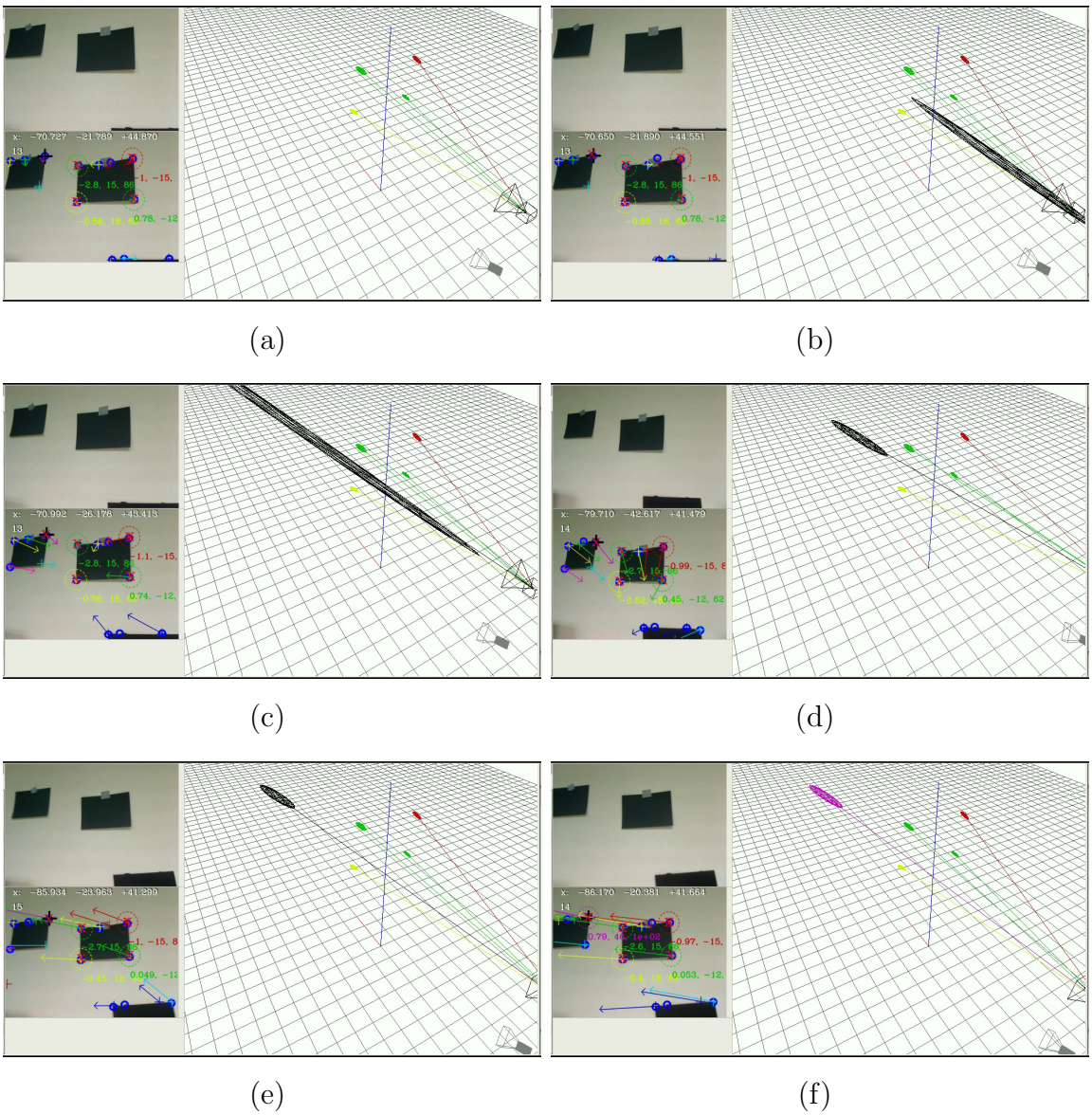
El rendimiento temporal del esquema permite su ejecución con más de 14 puntos de apoyo simultáneos con una tasa de 30 IPS.

6.4.3. Incorporación de nuevos puntos desconocidos a priori

Una vez constatado el éxito de este primer experimento, procedemos a probar la inicialización de nuevos puntos de referencia mediante el método del alejamiento inverso descrito en la sección 6.3.1. En las imágenes se puede ver cómo en un principio las regiones de confianza de la estimación del nuevo punto (coloreadas en negro) se alargan notablemente en la dirección del rayo en el que se las espera encontrar. Cuando la cámara se ha movido lo suficiente, el elipsoide se hace más pequeño y menos excéntrico, y se coloca en la posición correcta.

En el prototipo actual de la aplicación, los puntos que se deseen añadir al mapa 3D son seleccionados por el usuario de uno en uno haciendo uso del ratón. Una vez el punto se ha seleccionado, el Cuando el eje mayor del elipsoide de error está por debajo de un

cierto umbral (en el experimento se estableció este umbral en 5 cm) el punto estimado se incorpora al vector de estado y al mapa de covarianzas. Repitiendo el proceso varias veces, se consigue ir estableciendo puntos de referencia en un conjunto de cuadros de color negro colocados en la pared hasta llegar a tener localizadas las 4 esquinas del monitor.



(a) (b)

(c) (d)

(e) (f)

Figura 6.4: Secuencia de consolidación de un nuevo punto de apoyo.

Capítulo 7

Conclusiones y Trabajos futuros

En los capítulos anteriores hemos explicado los algoritmos diseñados para resolver el problema de la autolocalización 3D con una sola cámara. En este capítulo vamos a resumir las conclusiones más importantes que hemos sacado con la realización de este proyecto y repasaremos los objetivos planteados para saber en qué grado se han satisfecho. Por último, indicaremos cuáles pueden ser los trabajos futuros abiertos en relación con el proyecto realizado.

7.1. Conclusiones

Se ha diseñado y programado un algoritmo que estima en tiempo real la posición y orientación de una única cámara móvil autónoma en un entorno estático, utilizando exclusivamente las imágenes obtenidas por la cámara. Este algoritmo se ha validado experimentalmente haciendo uso de una webcam real. Si se sostenie la cámara en la mano y se realiza un movimiento suficientemente amplio y suave, una vez inicializado este algoritmo es capaz de establecer nuevos puntos de referencia en el entorno, gestionando cuándo se pierden de vista algunos de ellos.

A continuación vamos a repasar los subobjetivos que se plantearon en el capítulo 2 para conocer la solución adoptada en cada uno de ellos:

1. El primero de los subobjetivos era el seguimiento 2D de puntos de interés en la imagen (descrito en el capítulo 5), que se ha resuelto mediante filtrado de Kalman, con el apoyo de funciones de la librería OpenCV para el tratamiento de imagen. Este algoritmo por sí solo tiene entidad propia y está siendo utilizado actualmente en aplicaciones de visión computacional del grupo de Robótica de la URJC.

2. El siguiente subobjetivo era lograr localizar la cámara en 3D a partir de observaciones de un número fijo de puntos conocidos. Concretamente se ha visto que basta con 4 puntos para obtener una estimación razonable y estable. Para cumplir este objetivo se ha seguido la estrategia basada en EKF descrita en el capítulo 6, basada en el trabajo de [Davison, 2007]. En el proceso de modelado del sistema y diseño de los algoritmos se hizo un uso intensivo de la geometría proyectiva para calcular las funciones de proyección y sus matrices jacobianas. Durante la simulación del algoritmo en MATLAB se hizo patente que el rendimiento temporal no era un tema trivial, dado que el tiempo de ejecución era sensiblemente mayor al tiempo simulado.

3. Los algoritmos anteriores se combinaron para hacer la localización a partir de imágenes capturadas con una cámara real, consiguiendo así la validación experimental. El algoritmo de localización maneja una base dinámica de puntos de apoyo o *balizas* en la que entran nuevos puntos de referencia y se descartan aquellos de los que se pierde el seguimiento. Los resultados se muestran en un GUI que permite ajustar los parámetros, indicando en todo momento cuáles son los puntos que están en seguimiento y permitiendo al usuario ver un mapa tridimensional en un visor openGL el que se traza un modelo de la cámara, su campo de visión y la posición y la incertidumbre de los puntos de apoyo.

Estos objetivos se han satisfecho tras un período de trabajo que ha comprendido el aprendizaje de nuevas técnicas de procesamiento de señal, conceptos de geometría y desarrollo de software, así como un repaso de Álgebra Lineal.

Las aplicaciones realizadas para JdeRobot han sido el esquema *VisualTracking*, que permite ejecutar sólo el seguimiento 2D y jugar con los parámetros del algoritmo para adaptarlo a propósitos específicos; y el esquema *monoSLAM*, en el que se condensa toda la funcionalidad desarrollada en el proyecto y que realiza la autolocalización de la cámara.

Además, a pesar de no ser un objetivo del proyecto, se ha ampliado el esquema de JdeRobot Opencvdemo, verificando su utilidad como elemento de formación para el desarrollo de aplicaciones de procesamiento de imagen y visión artificial. También ha sido mérito de este proyecto la incorporación de nuevas técnicas a *Opencvdemo*, desarrolladas durante la fase de aprendizaje de la plataforma y de visión artificial, como ya se describió en la sección 4.5.

La plataforma JdeRobot se ha enriquecido gracias a nuestro proyecto, puesto que el esquema *monoSLAM* y las nuevas funciones de *Opencvdemo* ya forman parte de la versión oficial de la distribución.

Nuestro proyecto ha sido el primero en abordar la autolocalización visual y la construcción de mapas de forma simultánea, y se caracteriza por su flexibilidad, sentando las bases para que los robots del grupo puedan explorar entornos desconocidos. Otro factor importante es que se ha utilizado una única cámara como sensor para conocer la localización, recibiendo de este sensor imágenes de tamaño variable. Esta única cámara se ha usado para extraer información 3D novedosa del entorno, algo que se hacía típicamente con pares estéreo. Esto nos permite ver los pares estéreo desde un nuevo punto de vista, utilizándolos como una fuente de diversidad para robustecer el SLAM. Nuestro algoritmo será aplicable a diversos tipos de cámaras, siempre que estas se calibren correctamente.

Este algoritmo es útil para la implementación de sistemas avanzados de realidad aumentada en los que no es necesaria la presencia de patrones gráficos de referencia ni un proceso de postproducción.

Requisitos

Los requisitos que exigimos a las aplicaciones desarrolladas en este proyecto han estado marcados por lo especificado en la sección 2.2. Como ya establecimos allí, se ha utilizado la arquitectura de desarrollo JdeRobot para generar nuestras aplicaciones, siendo finalmente el lenguaje de programación C++ el utilizado. El hecho de utilizar esta plataforma de desarrollo nos ha facilitado enormemente el desarrollo, al disponer de esquemas y drivers ya existentes que han sido de gran utilidad y proporcionar la infraestructura necesaria para programar aplicaciones de procesamiento en tiempo real. En concreto, han sido cruciales para el éxito de este proyecto el esquema Calibrador [Kachach, 2008] y el driver Video4linux2.

Para el desarrollo de estos esquemas hemos manejado múltiples bibliotecas en el entorno del sistema Linux[1], como OpenCV para el procesamiento de imagen, OpenGL para la representación en 3D, GSL para las funciones algebraicas y GTK para el desarrollo de interfaces gráficas.

El requisito de tiempo real se ha cumplido ya que durante todo el desarrollo del proyecto se han realizado pruebas con el sistema funcionando a 30 IPS, optimizando cuando fue necesario las partes más críticas del código y haciendo una gestión proactiva de la memoria.

La precisión de la localización depende directamente del número de puntos de apoyo y de la distancia de estos a la cámara. presentando un error de entre 5 y 10 cm cuando

[1]Esto ha supuesto un importante complemento en la formación de Ingeniero de Telecomunicación, dotándome de habilidades necesarias para el desarrollo de software.

se tienen 4 puntos de apoyo a una distancia del orden de medio metro de la cámara. La precisión de los algoritmos de autolocalización y navegación utilizados en el grupo basados en sensores láser presentan un error medio de 20 cm. La condición de éxito comentada en la sección 2.2 (superar a algún algoritmo existente en algún escenario) se cumple en cuanto probamos nuestro sistema en un escenario no modelado a priori, gracias a que se va construyendo un mapa de éste a medida que la cámara se mueve, mientras que los esquemas de autolocalización visual y láser desarrollados hasta la fecha necesitaban conocer a priori el mapa del entorno.

La robustez del algoritmo depende directamente de cómo de estructurado esté el entorno en el que trabajamos. Escenas como la mostrada en la figura 5.1(a) son favorables y es poco probable que se pierda la localización. Sin embargo, entornos parecidos al de la figura 5.1(b) probablemente necesitarían del apoyo de técnicas de visión basadas en memoria visual para funcionar de forma fiable.

En lo referente a flujo de trabajo excesivo, la aplicación no presenta problemas en los PCs utilizados ya que el máximo de puntos de apoyo que puede procesar está muy por encima del número necesario para proporcionar una buena estimación de la localización 3D, y la aplicación permanece por debajo de ese límite para que se cumpla siempre el *timing* propuesto de 30 IPS sin descuidar el resto de requisitos.

7.2. Trabajos futuros

El trabajo realizado en el proyecto puede ser continuado más adelante por otras vías que describiremos a continuación.

Si se pierden demasiados puntos de referencia, el algoritmo pierde la localización y no se puede restablecer a no ser que se vuelva a inicializar. Este problema se puede paliar incorporando técnicas de relocalización como las descritas en [Williams, 2007].

Actualmente el funcionamiento de la aplicación está dividido en dos bloques independientes: seguimiento 2D y localización 3D. La ventaja de esta arquitectura es la facilidad para el diseño y depuración del prototipo, pero presenta un problema fundamental y es que el primer bloque manipula los datos que va a utilizar el segundo, introduciendo inevitablemente un error que reducirá la precisión de la localización. Los trabajos de Andrew Davison y George Klein demuestran que eligiendo bien los parámetros el procesamiento se puede hacer en un solo paso, evitando ejecutar el banco de filtros 2D, con la consiguiente ganancia en velocidad de cómputo, estabilidad y precisión. Las áreas de búsqueda se obtendrían de la matriz de covarianza estimada del residuo S del EKF 3D.

Las principales desventajas del EKF son el coste computacional de calcular las matrices jacobianas de los modelos de estado y observación y el hecho de que subestima la incertidumbre de las medidas debido al uso de una aproximación de primer orden. El *unscented Kalman filter* (UKF)(ref) emplea una técnica de muestreo determinista conocido como *unscented transform* para elegir un conjunto mínimo de puntos *sigma* de los cuales se puede obtener la media y covarianza de la estimación con más precisión que el EKF, y con un menor coste computacional.

El algoritmo desarrollado funciona bien en escenarios en los que los puntos de referencia no se mueven de su posición real, y esto limita mucho las posibilidades de aplicación del monoSLAM. Sin alterar significativamente la naturaleza del algoritmo, el uso de modelos dinámicos ligeramente más complejos podría permitir la inmersión en entornos dinámicos, contemplando la posibilidad de que distintas partes del entorno se muevan a distintas velocidades. Un entorno dinámico en el que podríamos aplicar una variante de este algoritmo es la percepción 3D de automóviles desde un automóvil en marcha, donde cada coche lleva una velocidad distinta relativa a la cámara. Esto podría ser de gran utilidad en sistemas de seguridad *precrash* al poder predecir los movimientos de los coches de alrededor.

Una vía por la que se puede mejorar el rendimiento del monoSLAM es el pretratamiento de la imagen. Por ejemplo, en nuestro algoritmo las características basadas en apariencia se extraen sólo de la componente de intensidad o luminancia. Técnicas como la extracción

de componentes principales de color nos permitirían aprovechar la información disponible en las demás componentes de la imagen. Como se vio en los experimentos descritos en la sección 5.4.2, puede existir desenfoque debido al movimiento de la cámara, que es un tipo de distorsión lineal. El grado de desenfoque depende de la distancia focal, la apertura de las lentes, la distancia entre cámara y objeto y la velocidad lineal y angular de la cámara. Esta distorsión puede modelarse y atacarse con técnicas de deconvolución ciega ¡ya que el monoSLAM proporciona continuamente una buena estimación de estos parámetros!. El resultado final es una situación de beneficio mutuo en la que el procesamiento de imagen contribuye a robustecer las estimaciones del monoSLAM, y éste ayuda a construir la base de conocimiento necesaria para llevar a cabo un buen procesamiento de imagen.

Bibliografía

[Agüero Durán, 2010] Carlos E. Agüero Durán. Técnicas de percepción compartida aplicadas a la asignación dinámica de roles en equipos de robots móviles. *Tesis doctoral. Universidad Rey Juan Carlos*, 2010.

[Barrera, 2008] Pablo Barrera. Aplicación de los métodos secuenciales de monte carlo al seguimiento visual 3d de múltiples objetos. *Tesis doctoral. Universidad Rey Juan Carlos*, 2008.

[Davison, 1998] Andrew J. Davison. Mobile robot navigation using active vision. *Phd Thesis. University of Oxford*, 1998.

[Davison, 2002] Andrew J. Davison. Slam with a single camera. *SLAM/CML Workshop at ICRA*, 2002.

[Davison, 2003] Andrew J. Davison. Real-time simultaneous localisation and mapping with a single camera. *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, 2003.

[Davison, 2004] Andrew J. Davison. Real-time 3d slam with wide-angle vision. *5th IFACE/EURON Symposium on Intelligent Autonomous Vehicles*, 2004.

[Davison, 2007] Andrew J. Davison. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence, vol 29, no. 6*, 2007.

[García, 2009] Eduardo Perdices García. Autolocalización visual en la robocup basada en detección de porterías 3d. *Proyecto Fin de carrera. Ing. Informática - Universidad Rey Juan Carlos*, 2009.

[Hartley and Zisserman, 2004] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[Haykin, 1986] Simon Haykin. *Adaptive Filter Theory*, chapter 7. Pearson Education, 1986.

[Kachach, 2008] Redouane Kachach. Calibración automática de cámaras en la plataforma jdec. *Proyecto Fin de Carrera. Ing. Informática - Universidad Rey Juan Carlos*, 2008.

[Martín Rico, 2008] Francisco Martín Rico. Aportaciones a la auto-localización visual de robots autónomos con patas. *Tesis doctoral.Universidad Rey Juan Carlos*, 2008.

[Marugán, 2010] Sara Marugán. Seguimiento visual de personas mediante evolución de primitivas volumétricas. *Proyecto Fin de Carrera. Ing. Informática - Universidad Rey Juan Carlos*, 2010.

[Miangolarra, 2010] Pablo Miangolarra. Memoria visual 3d para un robot móvil con par estéreo. *Proyecto Fin de Carrera. Ing. Informática - Universidad Rey Juan Carlos*, 2010.

[Montiel *et al.*, 2006] J. Montiel, J. Civera, and A. Davison. Unified inverse depth parametrization for monocular slam. In *Proceedings of Robotics: Science and Systems*, Philadelphia, USA, August 2006.

[Román López, 2009] Raúl Román López. Localización de dispositivos móviles para redes sociales dinámicas. *Proyecto Fin de Carrera. Ing. Telecomunicación - Universidad Rey Juan Carlos*, 2009.

[Williams, 2007] B. Williams. Real-time slam relocalisation. *Proc International Conference on Computer Vision, Rio de Janeiro*, 2007.