

# Proyecto Fin de Carrera

## Grado en Ingeniería Electrónica, Robótica y Mecatrónica

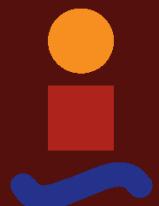
Localización y mapeo simultáneo en interiores  
(SLAM) basado en visión con cámara gran angular

Autor: Alberto González Isorna

Tutor: Manuel Ruiz Arahal

Dep. de Ingeniería de Sistemas y Automática  
Escuela Técnica Superior de Ingeniería  
Universidad de Sevilla

Sevilla, 2017





Proyecto Fin de Carrera  
Ingeniería Electrónica, Robótica y Mecatrónica

# Localización y mapeo simultáneo en interiores (SLAM) basado en visión con cámara gran angular

Autor:

Alberto González Isorna

Tutor:

Manuel Ruiz Arahal

Dep. de Sistemas y Automática  
Escuela Técnica Superior de Ingeniería  
Universidad de Sevilla  
Sevilla, 2017



Proyecto Fin de Carrera: Localización y mapeo simultáneo en interiores (SLAM) basado en visión con cámara gran angular

Autor: Alberto González Isorna

Tutor: Manuel Ruiz Arahal

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, Septiembre 2017

El Secretario del Tribunal

*A mi familia*

*A mis maestros*

*A P.*



# Agradecimientos

---

Después de un intenso período de varios meses, hoy es el día: escribo este apartado de agradecimientos para finalizar mi trabajo de fin de grado. Ha sido una gran experiencia, no solo en el campo científico, sino también a nivel personal. Escribir este trabajo ha tenido un gran impacto en mí y es por eso que me gustaría agradecer a todas aquellas personas que me han ayudado y apoyado durante este proceso.

En primer lugar, me gustaría agradecer a mi familia, por estar ahí en los momentos en los que llevar todo para delante parecía imposible, y por ser pacientes e intentar comprender qué estaba haciendo a pesar de que ninguno de ellos sea especialmente amante de la ingeniería.

En segundo lugar agradecer a mis amigos, en especial a mi amigo Andrés, el cual me ha ayudado aportándome material, y asesorándome en algunos temas en los que me encontraba más perdido.

En tercer lugar, a mi tutor Manuel Ruiz, por conducir como debía enfocar el trabajo, aportarme ideas, y guiarme en la realización del proyecto.

Por último a Patricia, por dejarme el material para poder realizar el trabajo, apoyarme en todo momento y creer siempre en mí.

*Alberto González Isorna  
Sevilla, septiembre 2017*



# Resumen

---

Para nosotros, los humanos, descubrir nuevos lugares, y pasear libremente por ellos, sin perdernos y sabiendo en todo momento como poder salir y como hemos entrado, resulta algo trivial. En cambio, para los robots es algo realmente complejo. Cuando un robot entra un lugar desconocido, debe crear un mapa y saber ubicarse en él. Esto en la literatura se conoce como SLAM (*Simultaneous Location And Mapping*). Hay numerosos métodos y formas de aplicar el SLAM. En este proyecto se pretende llegar a una solución de bajo coste, utilizando un número pequeño de sensores, haciendo un uso notable de la visión, que es la forma natural en la que nosotros los humanos resolvemos el problema. Tradicionalmente, los robots que utilizan visión se localizan y mapean el entorno basándose en observaciones de objetos o puntos de referencia (landmarks) que van encontrando a medida que avanzan. Como elemento diferenciador, nuestro robot irá haciendo observaciones al techo, ya que al ser este un elemento invariable y muy observable, le aporta unas interesantes propiedades. Por otro lado, para observar la mayor parte del techo y reducir en costes, usaremos una cámara gran angular.

# **Abstract**

---

For us, the human beings, to discover new places, and to walk freely along them, without getting lost and being able to go out like at all time and since we have entered, it turns out to be slightly trivial. On the other hand, for the robots it is something really complex. When the robot enters an unknown place, it must create a map and be able to locate in him. This in literature is known as SLAM (simultaneous location and mapping). There are numerous methods and ways of applying the SLAM. In this project one tries to come to a solution of low cost, using a small number of sensors, doing a notable use of the vision, which is the natural form in which we solve the problem.

In element differentiator of other types of SLAM based on vision, we will base on observations of the ceiling, to this being a considered invariable element throughout the time. On the other hand to observe most of the ceiling and to reduce in costs, we will use a chamber that has great angular as a chamber with a fisheye lens.

# Índice

---

<i>Agradecimientos</i>	<i>ix</i>
<i>Resumen</i>	<i>xi</i>
<i>Abstract</i>	<i>xii</i>
<i>Índice</i>	<i>xiii</i>
<i>Índice de Ecuaciones</i>	<i>xv</i>
<i>Índice de Tablas</i>	<i>xv</i>
<i>Índice de Figuras</i>	<i>xvi</i>
<i>Notación</i>	<i>xix</i>
<b>1 INTRODUCCION</b>	<b>20</b>
1.1 Presentación	20
1.2 Estado del arte	21
1.3 Objetivos	23
1.4 Aplicaciones	23
1.5 Planificación	24
1.6 Estructura del proyecto	25
<b>2 HITO A: VISIÓN</b>	<b>26</b>
2.1 Introducción a los sistemas de imágenes digitales	26
2.1.1 Adquisición de imágenes - Cámaras digitales	26
2.1.2 Color	27
2.1.3 Conceptos fundamentales de las imágenes digitales	27
2.1.4 Modelo de la cámara estenopeica	28
2.2 Calibración de la cámara	29
2.2.1 Modelo de la cámara ojo de pez	29
2.2.2 Cámara Qumox SJCAM4000	30
2.2.3 Calibración de la SJCAM4000	30
2.3 Transformaciones (I): del mundo a la cámara	33
2.4 Transformaciones (II): de la cámara al mundo	34
2.4.1 Introducción	34
2.4.2 Midiendo la profundidad: el láser	35
2.5 Cobertura, conclusiones y áreas de mejora: HITO A	39

<b>3 HITO B: CARACTERÍSTICAS</b>	<b>40</b>
3.1 Introducción al reconocimiento de formas	40
3.2 Filtrado mediante Transformada de Fourier	42
3.3 Método del rectángulo creciente	47
3.3.1 Desdistorsión de la imagen	47
3.3.2 Imagen binaria de contornos óptima	48
3.3.3 Encontrando las líneas	49
3.3.4 Redistorsión de los puntos obtenidos	50
3.3.5 Resultados	50
3.4 Mediando el mundo real	51
3.5 Cobertura, conclusiones y áreas de mejora: HITO B	52
<b>4 HITO C: SLAM</b>	<b>53</b>
4.1 Introducción	53
4.2 El filtro de Kalman (KF) y sus versiones no lineales (EKF, UKF)	53
4.2.1 KF	53
4.2.2 EKF	54
4.2.3 UKF	55
4.3 Ceilmarks	56
4.4 Localización y mapeo simultáneo (SLAM)	56
4.4.1 El robot se mueve, modelo de movimiento	57
4.4.2 El robot mide, modelo de observación	58
4.4.3 Nuevas observaciones y reobservaciones	59
4.4.4 No linealidades	59
4.5 Implementando el SLAM	60
4.5.1 Métodos	60
4.5.2 Simulaciones	60
4.5.3 Implementación	62
4.5.4 Resultados	67
4.6 Cobertura, conclusiones y áreas de mejora: HITO C	72
<b>5 LÍNEAS FUTURAS</b>	<b>73</b>
<b>6 COSTES</b>	<b>74</b>
<b>7 CONCLUSIONES DEL PROYECTO</b>	<b>75</b>
<i>Referencias</i>	76
<i>Glosario</i>	78
<i>Anexos</i>	79

# ÍNDICE DE ECUACIONES

---

Ecuación 2-1: Modelo cámara estenopeica	28
Ecuación 2-2: Modelo cámara ojo de pez	29
Ecuación 2-3: Ecuación distancia cámara-láser	35
Ecuación 3-1: Transformada discreta de Fourier 2D	42
Ecuación 3-2: Transformada inversa discreta de Fourier 2D	42
Ecuación 4-1: estado y entradas del vehículo	57
Ecuación 4-2: posición del landmark	57
Ecuación 4-3: Modelo aumentado simplificado	57
Ecuación 4-4: Modelo aumentado completo	57
Ecuación 4-5: Modelo observación SLAM	58
Ecuación 4-6: Ecuación compacta de la matriz de covarianza	58

# ÍNDICE DE TABLAS

---

Tabla 1-1: Clasificaciones del SLAM	21
Tabla 2-1: Características SJCAM 4000	30
Tabla 2-2: Media de los errores reproyección de cada imagen	32
Tabla 3-1: Ventajas y desventajas de nuestro filtrado en frecuencia	46
Tabla 3-2: Ventajas y desventajas método rectángulo envolvente	50
Tabla 4-1: Cumplimiento de requisitos de nuestros ceilmarks	56
Tabla 4-2: Errores cometidos SLAM definitivo	71
Tabla 6-1: Costes de los robots 1-4	74
Tabla 0-1: Parámetros de retorno calibración	82

# ÍNDICE DE FIGURAS

---

Figura 1-1: Clasificación general reconocimiento de formas	22
Figura 1-2: Prioridad de tareas del presente proyecto	23
Figura 1-3: Diagrama de Gantt Inicial	24
Figura 1-4: Diagrama de Gantt Final	24
Figura 2-1: Proceso completo imagen digital	26
Figura 2-2: Respuesta de los conos a las distintas frecuencias	27
Figura 2-3: Filtro RGB	27
Figura 2-4: Modelo cámara estenopeica	28
Figura 2-5: Modelos de proyección perspectiva central (izquierda) y ojo de pez (derecha) en 2D	29
Figura 2-6: Modelo ojo de pez	29
Figura 2-7: Davide Scaramuzza	30
Figura 2-8: Fotografías 3 (izq.) y 10 (derecha) tomadas en el proceso de calibración	31
Figura 2-9: Extracción de esquinas imagen 1	31
Figura 2-10: Ecuación de proyección y relación ángulo de incidencia-distancia	32
Figura 2-11: Posición de las imágenes (izquierda) y fotografía 9 (derecha)	32
Figura 2-12: Desarrollo modelo completo directo	33
Figura 2-13: Plano epipolar	34
Figura 2-14: Relaciones láser-cámara	35
Figura 2-15: Apertura de una imagen.	35
Figura 2-16: Imagen del techo de la habitación en escala de grises	36
Figura 2-17: Imagen del techo tras aplicar métodos morfológicos	36
Figura 2-18: Imagen original tras encontrar el láser	37
Figura 2-19: Proceso seguido para hallar el modelo inverso	38
Figura 2-20: Experimento medición y verificación distancia	38
Figura 2-21: Porcentaje de cobertura hito A	39
Figura 3-1: Ejemplo clasificación de características	41
Figura 3-2: Radiografía y su respectiva transformada de Fourier	41
Figura 3-3: Imagen del pasillo y DFT	42
Figura 3-4: Filtrado de imágenes en frecuencia	43
Figura 3-5: Imagen original de la cocina	43
Figura 3-6: DFT Imagen desdistorsionada y filtro paso alto	44
Figura 3-7: Imagen filtrada paso alto	44
Figura 3-8: Imagen ideal filtrada	45
Figura 3-9: Vista inferior imagen ideal	45
Figura 3-10: DFT Imagen desdistorsionada y filtro específico	46
Figura 3-11: Imagen real tras la búsqueda de esquinas	46

Figura 3-12: Imagen original y desdistorsionada	47
Figura 3-13: Imagen binaria de contornos, sin ajuste y ajustando la imagen	48
Figura 3-14: Orden de búsqueda	49
Figura 3-15: Imagen original en escala de grises tras la búsqueda de esquinas	50
Figura 3-16: Posición de las esquinas respecto de la cámara en metros	51
Figura 3-17: Porcentaje de cobertura HITO B	52
Figura 4-1: Esquema KF	54
Figura 4-2: Esquema del EKF	54
Figura 4-3: Esquema UKF	55
Figura 4-4: Resumen ecuaciones SLAM	59
Figura 4-5: Esquema del triciclo	60
Figura 4-6: Estimación posición por odometría	61
Figura 4-7: SLAM con un landmark	61
Figura 4-8: SLAM con tres landmarks	62
Figura 4-9: Esquema proceso SLAM	63
Figura 4-10: Proceso de toma de fotografías	63
Figura 4-11: Ejes de la cámara y el mundo real	64
Figura 4-12: Estimación inicial SLAM	67
Figura 4-13: Estimación final SLAM	67
Figura 4-14: Determinantes matrices covarianzas (P)	68
Figura 4-15: Fotografía 14, falso ceilmark	69
Figura 4-16: Estimación sin corrección	69
Figura 4-17: Mapa del habitáculo	70
Figura 4-18: Errores cometidos en los diferentes casos	70
Figura 4-19: Gráfico superficie errores posición	71
Figura 4-20: Errores cometidos en el ángulo en radianes	71
Figura 4-21: Porcentaje cobertura hito C	72
Figura 5-1: Esquema completo robot complementario	73
Figura 7-1: Idea original	75



# Notación

---

$A^*$	Conjugado
c.t.p.	En casi todos los puntos
c.q.d.	Como queríamos demostrar
e.o.c.	En cualquier otro caso
IRe	Parte real
IIIm	Parte imaginaria
sen	Función seno
tg	Función tangente
arctg	Función arco tangente
sen	Función seno
sgn	Función signo
rect	Función rectángulo
Sinc	Función sinc
$\partial y / \partial x$	Derivada parcial de $y$ respecto
$x^\circ$	Notación de grado, $x$ grados.
$\Pr(A)$	Probabilidad del suceso $A$
SNR	Signal-to-noise ratio
MSE	Minimum square error
:	Tal que
<	Menor o igual
$\Leftrightarrow$	Si y sólo si
erf	Función erf

# 1 INTRODUCCION

---

*"La ciencia puede divertirnos y fascinarnos, pero es la ingeniería la que cambia el mundo"*

- Isaac Asimov -

## 1.1 Presentación

**E**l interés de este proyecto surge de una idea de hace aproximadamente dos años, en la que soñaba con un robot camarero, el cual era capaz de moverse libremente por interiores y servir a los clientes. Por aquel entonces, mis conocimientos en robótica no eran muy amplios, y no sabía muy bien cómo se podría proceder. El año pasado ya empecé a idear como podría llevarlo a cabo y a que problemas me tendría que enfrentar. Este año decidí afrontar el problema y pensé que este proyecto sería la oportunidad perfecta.

Teniendo en cuenta las limitaciones temporales del proyecto, y que ha sido un trabajo que he elegido libremente y por lo tanto no tenía un guion a seguir, no ha sido posible abarcar el proyecto completo, aunque si resolver el problema principal: la localización y mapeo en un habitáculo desconocido, de una forma asequible.

Para resolver este problema, existen actualmente varias soluciones que están comentadas en la siguiente sección. En este proyecto se trabajará principalmente con métodos basados en la visión, debido a que es el medio natural en el que nos desenvolvemos los humanos. Esto conlleva una serie de ventajas e inconvenientes explicadas en esta memoria y resumidas en las conclusiones de la misma.

El método de localización y mapeo simultáneo (SLAM) elegido está basado en el filtro de Kalman Extendido (EKF) explicado en la sección 4.2: El filtro de Kalman (KF) y sus versiones no lineales (EKF, UKF). Este método, pese a no ser de los más simples de implementar, tiene un sinfín de aplicaciones y está muy extendido en el mundo de la robótica.

La fuente principal de información de nuestro SLAM es la visión, lo que conlleva la principal ventaja de obtener mucha información del entorno, y la desventaja de tener un algoritmo lo suficientemente potente para distinguir información útil de aquella que no nos aporta nada. Los métodos elegidos para el tratamiento de imágenes y la extracción de características serán abordados más adelante.

## 1.2 Estado del arte

**E**n este apartado expondremos en primer lugar una breve historia del SLAM, seguido de una clasificación actual de los distintos procedimientos y posteriormente comentaremos los métodos de cómputo más utilizados. Además analizaremos una breve historia del arte de la visión por computador y los algoritmos de reconocimiento de formas.

El nacimiento del problema del SLAM ocurre durante la Conferencia sobre Robótica y Automática del IEEE, en la ciudad de San Francisco, en 1986 [1]. Durante el acto, se trataron diversos temas relacionados con el mapeo y la localización de robots, problemas que hasta entonces se consideraban separados, y se reconoció que era un problema fundamental de la robótica que merecía ser tratado.

Los años posteriores, mientras diversos estudios avanzaban en este campo [2], se estaban produciendo importantes mejoras en el campo de la visión por computador y los filtros probabilísticos usados para la navegación visual y la navegación por sónar, destacando el filtro de Kalman.

Paulatinamente, se llegó a la conclusión que el problema de mapeo y localización debía ser enfocado como un problema único, ya que la localización permite al robot saber encontrarse en el mapa, y el mapa le posibilita poder localizarse.

El descubrimiento más importante fue que el problema del mapeado y la localización, una vez formulado como un único problema de estimación, era de naturaleza convergente. Aún más, se reconoció que las correlaciones entre los objetos de referencia, que otros científicos intentaban minimizar o eliminar, eran la pieza clave del problema y que, cuantas más ricas fueran estas correlaciones, mejor sería la solución [1]

Hay diversas clasificaciones en cuanto a formas de realizar el SLAM, dependiendo de razones como la forma de extracción de datos, el entorno,... A continuación se muestra una tabla con las clasificaciones más habituales en la literatura, se marcará en verde las clasificaciones que corresponden con el actual proyecto.

---

### Clasificaciones del SLAM

- **Offline:** se realiza SLAM sobre un conjunto de datos que previamente fueron recuperados con algún robot.
  - **Online:** se realiza SLAM a la vez que el robot se mueve
  - **Topológico:** se basa en algunas las características del entorno.
  - **Métrico:** proveen información métrica de los lugares
- 
- **Pasivo:** otra entidad se encarga del control del robot, SLAM puramente observador
  - **Activo:** el robot explora de forma activa el entorno
  - **Híbridos:** combinación de los dos anteriores
- 
- **Basado en marcas:** se extraen características de las medidas de los sensores y se arma el mapa a partir de esa información
  - **Volumétrico:** el mapa es muestreado a una resolución que permite una reconstrucción fotográfica del entorno.
- 

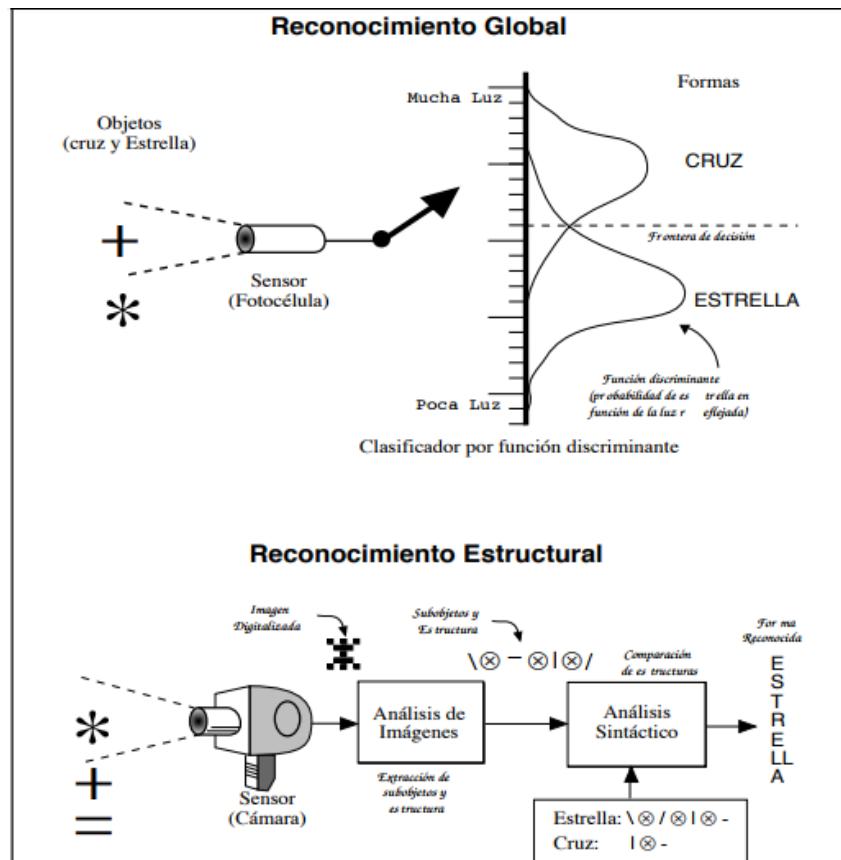
Tabla 1-1: Clasificaciones del SLAM

En cuanto a los principales algoritmos utilizados podemos destacar:

- a) **Filtro Extendido de Kalman:** es el utilizado en este proyecto. El filtro de Kalman es un algoritmo desarrollado por Rudolf E. Kalman en 1960 que sirve para poder identificar el estado oculto (no medible) de un sistema dinámico lineal. El filtro de Kalman (KF) y su versión extendida (EKF) será abordada en la sección 4.2. Se trata de la más extendida en la literatura y de las que mejores resultados se ha obtenido en la práctica.
- b) **Filtro de partículas:** propuesto en 1993 por N. Gordon, D. Salmond, y A. Smith se plantea como una implementación no paramétrica del filtro de Bayes [3]. La idea es representar la función de distribución posterior mediante un conjunto de muestras aleatorias (partículas) con pesos asociados y calcular estimaciones de acuerdo a dichas estimaciones y pesos.
- c) **Mapa de ocupación de celdillas:** fueron introducidos Hans Moravec y Alberto Elfes a mediados de 1980. Este método se basa en discretizar el espacio, dividiéndolo en celdillas, las cuales se clasifican como ocupadas o vacías según un determinado índice de confianza o probabilidad.

En lo que se refiere a la visión artificial o visión por computador, esta empezó a finales de los años 60 en universidades pioneras en inteligencia artificial. Posteriormente, los estudios realizados en los años 70 formaron los primeros cimientos de muchos de los algoritmos de visión computacional que existen actualmente, incluyendo la extracción de bordes de imágenes, etiquetado de líneas, modelado no poliédrico y poliédrico, representación de objetos como interconexiones de estructuras más pequeñas, flujo óptico y estimación de movimiento [4].

Dos son actualmente las maneras más usuales de representar las formas en su aspecto de conjunto de objetos: aquellas que recurren a una visión geométrica de lo que es un conjunto (Representación global), considerándolo como una (sub)región de un determinado espacio y aquellas que prefieren verlo como formado por elementos que cumplen ciertas reglas estructurales (Reconocimiento estructural). Nuestro caso de estudio es el reconocimiento estructural, ya que permite más flexibilidad y algoritmos de computación más avanzados. Dentro de éste encontramos diversos métodos:



- Discriminantes lineales: las funciones discriminantes son rectas que separan regiones.
- Discriminantes por distancia: los objetos se clasifican según la distancia mínima a cada prototipo ideal.
- Discriminante de bayes: la clasificación es probabilística. Se clasifica un objeto en la clase que más probabilidad tiene de haber producido dicho patrón.
- Redes neuronales: inspiradas en el comportamientos de las neuronas y conexiones del cerebro humano y tratan de crear un programa, sistema o máquina que sea capaz de solucionar problemas de reconocimiento de formas, de modo análogo a como lo haríamos los humanos.

Figura 1-1: Clasificación general reconocimiento de formas

## 1.3 Objetivos

El proyecto tiene como finalidad principal analizar una solución específica y de bajo coste, al siguiente problema: Un robot se encuentra en una habitación o recinto cerrado en el que debe saber ubicarse y poder construir un mapa básico del entorno, pudiendo si es posible identificar las salidas del mismo. Como ya mencionamos, debido a limitaciones estructurales del trabajo, no se ha abordado algunas partes. La decisión adoptada ha sido conseguir resolver el problema, ahorrando o eliminando alguna parte de la implementación tanto a nivel de hardware como software, como detallaré más adelante.

En la siguiente figura se muestran los objetivos del proyecto ordenados por prioridad descendente, según el porcentaje que deberían ocupar dentro del proyecto:

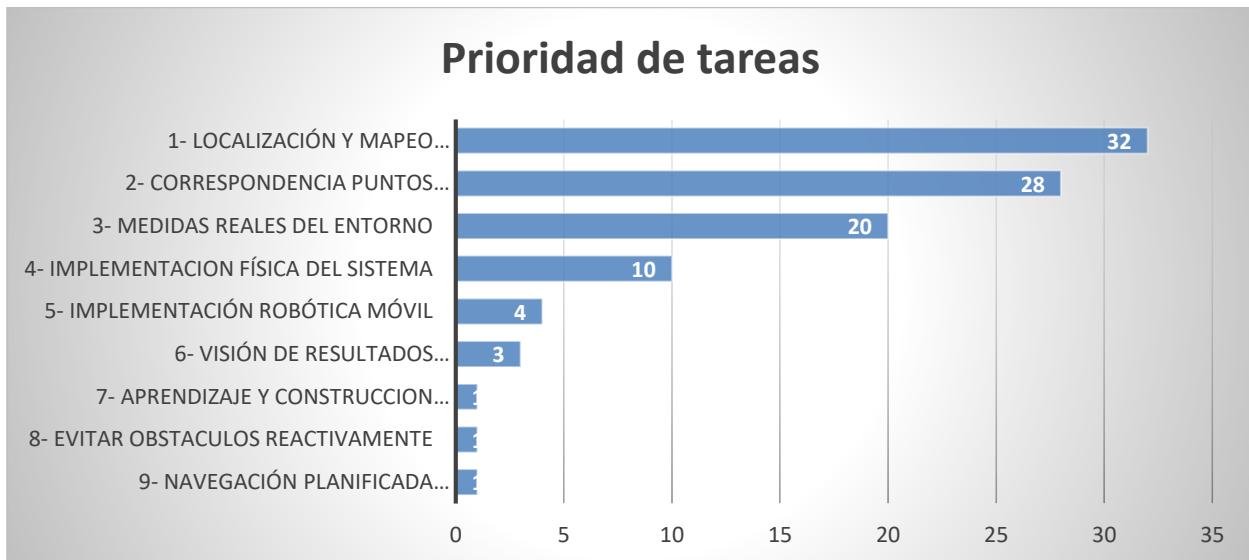


Figura 1-2: Prioridad de tareas del presente proyecto

## 1.4 Aplicaciones

Como se citó en la introducción, en un primer momento la idea originaria de la aplicación de este proyecto sería crear un robot camarero, lo cual significaría montar el robot de este proyecto a otra plataforma con brazos manipuladores, y abordar las tareas propias a la manipulación del camarero. Una vez centrados en el proyecto nos hemos dado cuenta que es trasladable a múltiples aplicaciones y perfiles.

Teniendo en cuenta que nuestro robot está en la categoría de robots de interiores, las aplicaciones serían las siguientes:

- Robot doméstico: robot que ayudaría a personas con movilidad reducida. Los robots podrían desempeñar tareas domésticas incorporando manipulación y software de reconocimiento de voz.
- Robot de emergencias: el hecho de que sea capaz de reconocer lugares en los que nunca antes ha estado, podría ser útil en escenarios en los que hay una situación donde el hombre no puede acceder o corre peligro (ya sea una bomba, elementos químicos, entornos de alta tensión,...).
- Robot clínico: una de las aplicaciones más inmediatas sería para el transporte de herramientas quirúrgicas de una sala a otra, de comida o incluso de pacientes. Para ello habría que modificar el SLAM para que fuera capaz también de hacer un mapeo a nivel de planta y en resumen del edificio entero.

## 1.5 Planificación

En cuanto a la planificación adoptada, en un primer momento se organizó para que el proyecto ocupase desde febrero a principios de junio, es decir, cuatro meses. Posteriormente, debido a razones externas, la prolongación de la implementación del EKF y los exámenes finales, se decidió presentarlo en septiembre, aprovechando julio para terminar la memoria. En las siguientes figuras podemos ver el diagrama de Gantt del proyecto inicial y final.

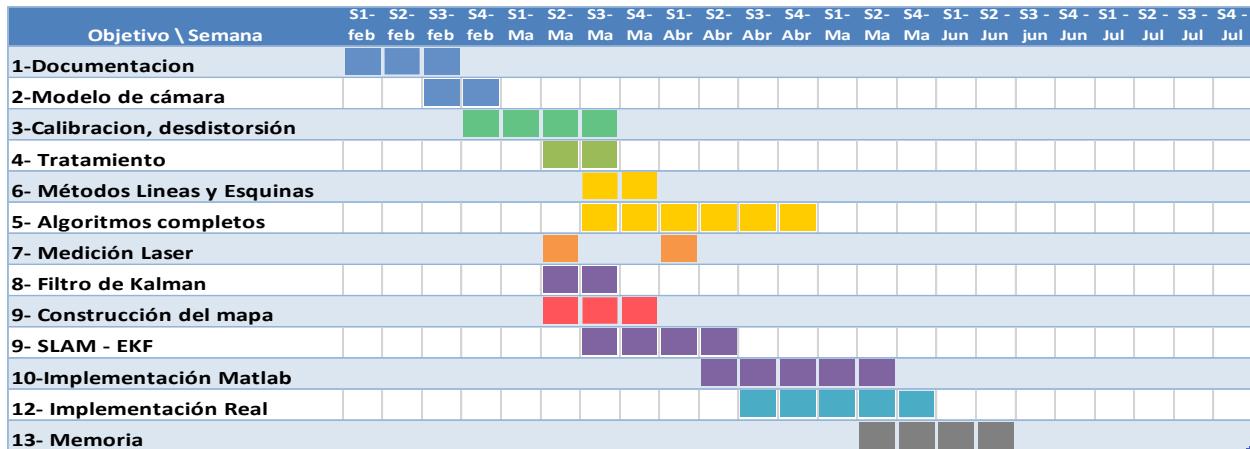


Figura 1-3: Diagrama de Gantt Inicial

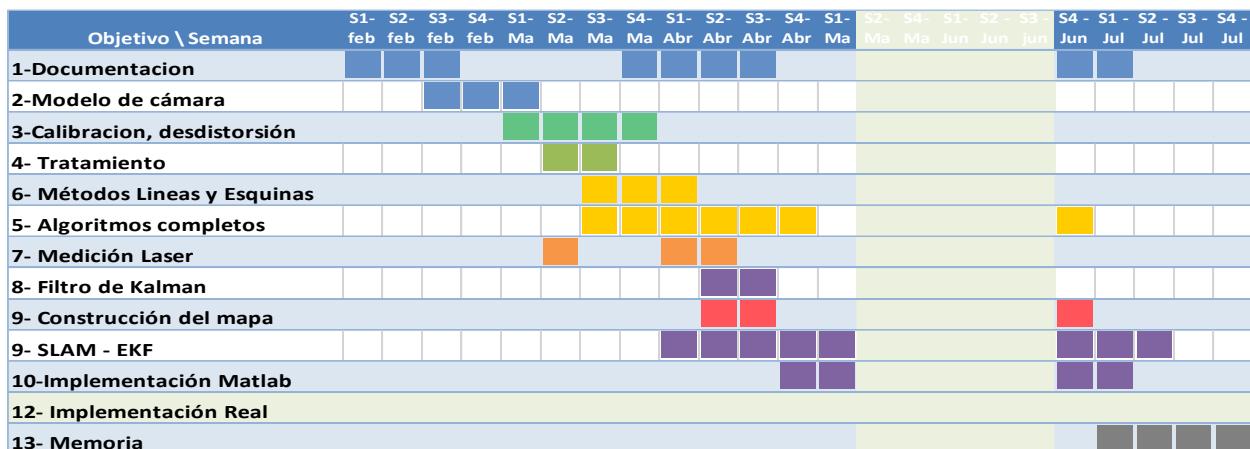


Figura 1-4: Diagrama de Gantt Final

Cabe mencionar:

- En un primer lugar se investigó como podría hacerse la implementación hardware del código. En primer lugar se barajó la idea de exportar a C/C++ el código Matlab, pero se desistió debido a incompatibilidades de librerías propias de Matlab. Si lográbamos escribir en C las librerías habríamos resuelto el problema. Pese a que los programas no hacen mucho uso de las librerías algunas si son fundamentales, como las que se usan para la visión por computador. La segunda opción que se barajó fue hacerlo todo en Open CV [5] o traducirlo a Open CV una vez finalizado, lo que era bastante asequible. Por limitaciones temporales no se hizo y se decidió no proceder con la implementación externa del sistema, quedando todos los programas en Matlab y teniendo que hacer uso del ordenador para el mapeo y la localización.
- Por otro lado debido a que mis nociones sobre filtro de Kalman eran escasas y no tenía la asignatura de Robótica avanzada, el ponerme al día con el SLAM-EKF y llegar a entenderlo me supuso más tiempo del previsto.

## 1.6 Estructura del proyecto

El proyecto se ha estructurado conforme a los objetivos del diagrama de Gantt y la necesidad lógica que ha seguido el mismo. Iniciaremos la memoria hablando de cómo extraer correctamente los datos de la cámara, posteriormente como filtrar y analizar esta información y de qué forma nos servimos de ella para implementar el SLAM junto a otros sensores.

Por ello el proyecto se divide en tres grandes bloques o hitos diferenciados. Dentro de cada uno hay unas metas específicas a lograr para llegar al final del proyecto en el que el robot sepa ubicarse y mapear el entorno

- Hito A: Visión
  - a. Calibración de la cámara ojo de pez
  - b. Extracción del Modelo de la cámara
  - c. Funciones de Transformación 2D|3D
- Hito B: Características
  - d. Métodos extracción características
  - e. Fourier
  - f. Rectángulo envolvente
  - g. Cálculo de distancias y superficies
- Hito C: Localización
  - h. KF, EKF, ejemplo simple
  - i. Introducción SLAM
  - j. ceilmarks
  - k. Implementación simulada
  - l. Implementación final

## 2 HITO A: VISIÓN

*"Como la vista es al cuerpo, la visión es al espíritu"*

- Aristóteles -

Este apartado ocupará desde una introducción de adquisición de imágenes digitales, una explicación del funcionamiento de la cámara, seguido del proceso de calibración, hasta como se ha conseguido obtener la relación de un punto del espacio en tres dimensiones al plano de la cámara que es nuestro principal objetivo. También hablaremos más delante como hemos hecho para conseguir la transformación inversa.

### 2.1 Introducción a los sistemas de imágenes digitales

Los sistemas de imágenes digitales ocupan desde que capturamos la foto hasta que la observamos por pantalla. Cabe recalcar, que lo que se obtiene de la adquisición bruta de los sensores, dista mucho de la imagen final. Esta tiene que ser procesada e interpretada para que al mostrarla por pantalla entendamos que hemos fotografiado y esta fotografía sea lo más parecida a la realidad. Por tanto los grandes bloques de estos sistemas serían: **adquisición, procesado y representación**. A modo ilustrativo se expone en la siguiente imagen cuales serían estos tres grandes bloques que permiten llevar a cabo esta *comunicación* entre imágenes, en este caso en un sistema médico:

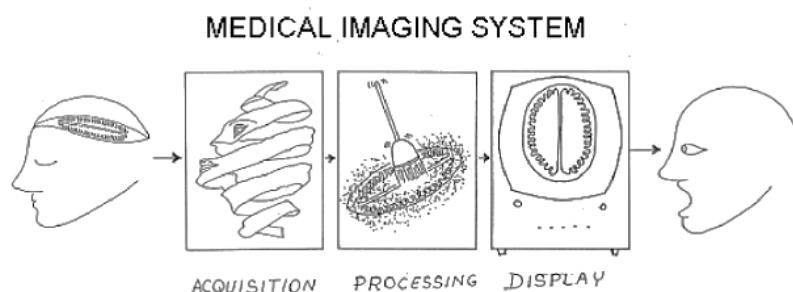


Figura 2-1: Proceso completo imagen digital

#### 2.1.1 Adquisición de imágenes - Cámaras digitales

La adquisición de la imagen está a cargo de algún transductor o conjunto de transductores que transforman la información de luz o alguna otra forma de radiación en una representación del objeto en cuestión.

En el caso de la visión humana, esta ocurre gracias a unos receptores que se encuentran en nuestros ojos. En concreto, los bastones se ocupan de captar intensidad luminosa y los conos discriminan la longitud de onda.

En el caso de las imágenes digitales, estos transductores son los fotoreceptores. Los sensores más comunes son los CCD (Charge coupled Device) y los CMOS (Complementay metal-oxide semiconductor). Sus diferencias vienen explicadas en el Anexo A: Sensores CCD vs CMOS.

Una cámara digital aloja un sensor y además varios elementos para conducir la luz (**lentes**), controlar la cantidad que pasa (**diafragma**) y **filtrarla**. El propósito de la lente es transmitir rayos de luz en la cámara y enfocarlos para formar una imagen intensa sobre el sensor. Se denomina **objetivo** al dispositivo que contiene el conjunto de lentes convergentes y divergentes.

## 2.1.2 Color

La sensación que captamos de color o tono se corresponde a la respuesta conjunta de los conos a las distintas componentes del espectro de la luz que se está captando.

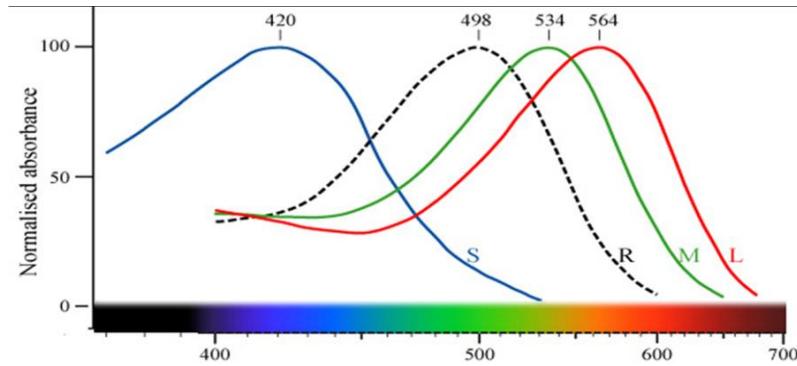


Figura 2-2: Respuesta de los conos a las distintas frecuencias

Como hemos visto, gracias a los conos podemos distinguir longitudes de onda, es decir, ver los distintos colores. En el caso de la cámara digital esto se realiza mediante un filtro RGB. Este filtro se corresponde con una matriz cuyas celdillas solo dejan pasar uno de los tres colores principales (rojo, verde o azul).

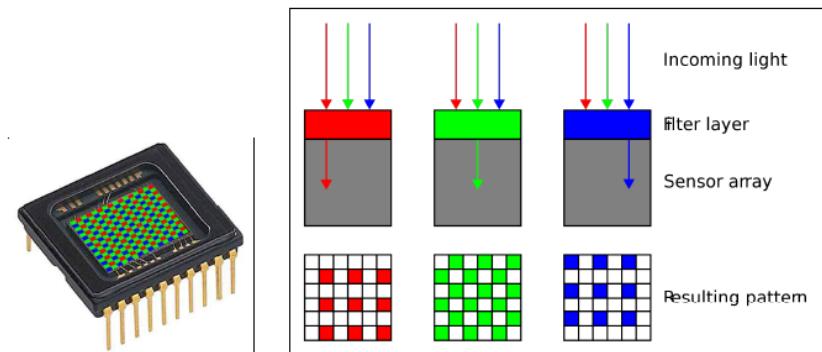


Figura 2-3: Filtro RGB

Hay varios modelos para expresar el conjunto de colores; RGB, CMYK, HSL,... en nuestro proyecto usaremos el RGB aunque no sea el más afín a nuestra percepción de los colores<sup>1</sup>.

## 2.1.3 Conceptos fundamentales de las imágenes digitales

Una vez entendido como funciona la cámara, expondremos una serie de conceptos fundamentales de las imágenes digitales necesarios a lo largo de esta memoria:

- Píxel: se correspondería a cada cuadrado de la matriz de bayes, se corresponde con la unidad en dos dimensiones de una imagen. Su nombre viene de picture element.
- Resolución: es el número de píxeles que hay por unidad de longitud de la imagen.
- Cuantización: es la discretización de los valores de la intensidad luminosa de cada pixel. Una imagen codificada en 4 bits tendrá 16 posibles valores de gris mientras que una imagen codificada en 8bits tendrá 255 valores de gris.
- Celdilla: se corresponde a la forma que toma cada píxel, usualmente cuadrada, aunque existen también hexagonales con interesantes propiedades.

<sup>1</sup> Las funciones de las respuestas de los conos no son lineales, ni iguales para cada color primario. Es por ello que el modelo aditivo lineal es una aproximación a como vemos los colores, hay modelos mas fieles como el HSV.

- Histograma: distribución del número de píxeles que hay en la imagen por cada tono de color.
- Contraste: diferencia relativa de intensidad entre un punto de una imagen. Cuanto más amplio sea el histograma más contraste tendremos.
- Saturación: intensidad de un matiz específico. En un sistema RGB la saturación puede ser descrita como la desviación estándar entre las coordenadas Roja, Verde y Azul.
- Distancia focal: distancia entre el centro óptico de la lente y el foco o punto focal.

### 2.1.4 Modelo de la cámara estenopeica

Obtener el modelo de la cámara nos permite corregir distorsiones y defectos producidos durante la toma de la imagen, medir el entorno a partir de la imagen (fotogrametría), y crear imágenes virtuales a partir de planos, modelos 3D, etc.

En esencia, el modelo de la cámara nos facilita la relación de un punto que está en el espacio 3D con la representación del mismo en nuestro plano de la imagen. El modelo más simple es el de la cámara estenopeica<sup>2</sup>, que nos servirá para entender cuál es el modelo de la cámara completa.

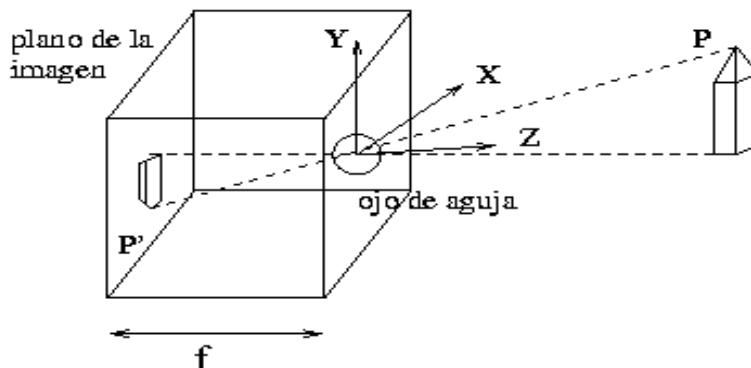


Figura 2-4: Modelo cámara estenopeica

En este modelo, por trigonometría podríamos sacar las ecuaciones de proyección:

$$x = -X \frac{f}{Z} ; \quad y = -Y \frac{f}{Z} ; \quad (I)$$

Ecuación 2-1: Modelo cámara estenopeica

A partir de ahora, las componentes que figuren en minúscula se entenderán como las del punto de la imagen 2D y en mayúscula las componentes del punto en el espacio 3D. Observamos que teniendo la distancia focal y la distancia del objeto a la cámara podríamos obtener esta transformación.

Este modelo sólo es válido para la cámara estenopeica, las lentes captan más luz que el estenope, pero introducen distorsiones. Además, siendo nuestra lente de ojo de pez, no es un efecto que podamos despreciar en absoluto. El efecto de la distorsión es que los puntos de la imagen se desplazan respecto lo que indica la ecuación de proyección. Habitualmente los elementos de distorsión se descomponen en componentes tangenciales y radiales.

Para averiguar cuál sería entonces el modelo de nuestra cámara, tendríamos que recurrir a un proceso de calibración, que es justamente lo que veremos en el siguiente apartado.

<sup>2</sup> Una cámara estenopeica (del griego στένω/steno estrecho y ὄπη/ope abertura, agujero) es una cámara sin lente, que consiste en; una caja donde se estanca a la luz, con sólo un pequeño orificio por donde entra la luz, el estenope y un material fotosensible.

## 2.2 Calibración de la cámara

Hemos visto en el apartado anterior que gracias al modelo de la cámara estenopeica descrito por la ecuación (I) podemos pasar de un punto en el espacio ( $X, Y, Z$ ) a su equivalente en el plano imagen ( $x, y$ ). Aparte de las distorsiones como ya comentamos, nuestra cámara sigue una geometría de proyección distinta al tener una lente de ojo de pez. A continuación comentaremos como es el modelo en este caso, describiremos la cámara que vamos a utilizar y el proceso de calibración que hemos seguido.

### 2.2.1 Modelo de la cámara ojo de pez

Las imágenes realizadas con una lente ojo de pez, no siguen el modelo de la proyección cónica que vimos en la cámara estenopeica, sino que siguen aproximadamente una relación lineal entre el ángulo de incidencia del haz de luz del punto en cuestión y la distancia al centro de la lente.

En la siguiente imagen se plasma el modelo de una cámara convencional (izquierda) y el de una con una lente ojo de pez (derecha). En dos dimensiones por simplicidad.

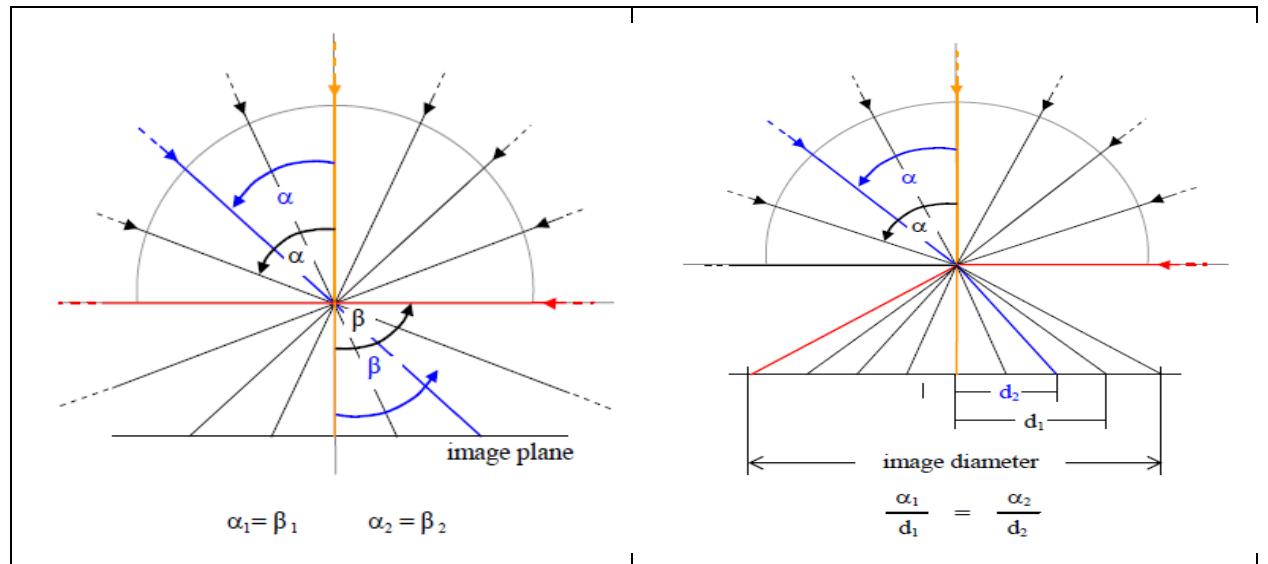


Figura 2-5: Modelos de proyección perspectiva central (izquierda) y ojo de pez (derecha) en 2D

La idea sería equivalente en el modelo de tres dimensiones, las ecuaciones del modelo completo serían las siguientes:

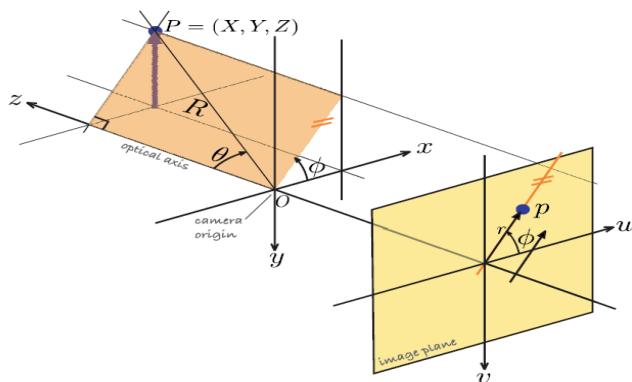


Figura 2-6: Modelo ojo de pez

$$\left\{ \begin{array}{l} R^2 = X^2 + Y^2 + Z^2 \quad (I) \\ \theta = \arccos \left( \frac{Z}{R} \right) \quad (II) \\ \phi = \arctan \left( \frac{Y}{X} \right) \quad (III) \\ u = r \cos \phi \quad (IV) \\ v = r \sin \phi \quad (V) \\ r = f(\theta) \quad (VI) \end{array} \right.$$

Ecuación 2-2: Modelo cámara ojo de pez

## 2.2.2 Cámara Qumox SJCAM4000

Sjcam Global, Inc. Es una empresa China dedicada a la producción y venta de cámaras deportivas de acción, empleadas para la captura de fotografía y vídeo en alta definición y en resoluciones hasta de 4k en deportes extremos, vídeo y fotografía aérea. Qumox es la distribuidora española.

El modelo utilizado en cuestión es el SJCAM 40000, es una cámara de acción de gama media cuyas características que más nos atañen son las siguientes:

<b>Chipset</b>	Novatek 96650 <sup>3</sup>
<b>Lente</b>	Objetivo gran angular 170º A + HD <sup>4</sup>
<b>Formato de vídeo</b>	MOV
<b>Resolución de video</b>	1080P 30fps , 720P 60fps, VGA (848 *480) 60fps, QVGA (640*480) 60fps
<b>Formato de imagen</b>	JPG
<b>Resolución de imagen</b>	12 MP   10 MP   8 MP   5MP   3 MP   2 MHD
<b>Frecuencia</b>	50Hz   60Hz
<b>Medidas</b>	29.8x59.2x41 mm

Tabla 2-1: Características SJCAM 4000

## 2.2.3 Calibración de la SJCAM4000

Existen diversas técnicas de calibración de cámara. De acuerdo a la literatura, estas pueden ser clasificadas en dos grandes categorías; calibración fotogramétrica y autocalibración. La calibración fotogramétrica se realiza mediante la observación de patrones cuya geometría en el espacio 3D es conocida con un buen nivel de precisión. La autocalibración se basa en el movimiento de la cámara observando una escena estática, a partir de sus desplazamientos y usando únicamente la información de la imagen.

Debido a que la autocalibración resultaba más compleja, y no había muchos procedimientos informáticos para llevarla a cabo, optamos por la calibración fotogramétrica. En primer lugar recurrimos a la librería de calibración de MATLAB, pero no ofrecía mucha exactitud para lentes con tanta distorsión como la nuestra. Por ello, gracias a la orientación del tutor, elegimos la librería de calibración de Davide Scaramuzza<sup>5</sup>.

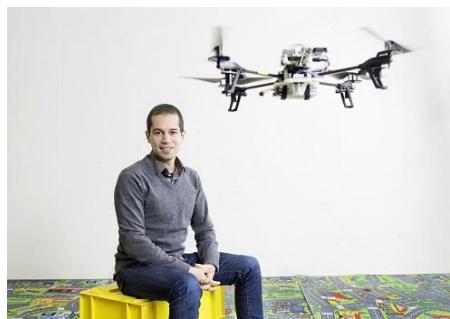


Figura 2-7: Davide Scaramuzza

<sup>3</sup> Es un híbrido de microcontrolador y procesador digital de señal, llamado DSC (digital signal controller), con un procesador de video; <https://dashcamtalk.com/cams/mobius/Novatek%20NT96650.pdf>

<sup>4</sup> Tras investigar bastante y hacer comprobaciones nos dimos cuenta que ofrece 170º pero de diagonal, por lo que en realidad limita bastante la adquisición como veremos más adelante

<sup>5</sup> Davide Scaramuzza (1980, italiano) es profesor de Robótica en la Universidad de Zurich. Es fundador y director del Grupo de Robótica y Percepción, donde realiza investigaciones de vanguardia sobre la visión por ordenador de baja latencia aplicada a la navegación autónoma de robots terrestres y microvoluntarios guiados visualmente. Recibió su PhD (2008) en Robótica y Visión por Computadora en ETH Zurich (con Roland Siegwart). Fue postdoctorado en ETH Zurich y la Universidad de Pensilvania (con Vijay Kumar y Kostas Daniilidis)

Los pasos seguidos para lograr una correcta calibración han sido los siguientes:

- Impresión del patrón:** una vez que tenemos la librería el primer paso es imprimir el patrón (Anexo B) y pegarlo a una superficie rígida de forma que no se curve, ya que podría estropear el proceso de calibración.
- Toma de fotografías:** el objetivo es cubrir el mayor volumen de fotogramas posible, cuanto más datos dispongamos, más precisa será la calibración. Hubo que repetir varias veces el proceso de calibración por razones de exceso o defecto de luminosidad, y poca variedad en el ángulo de las fotografías. Finalmente un conjunto de 18 fotografías fueron elegidas y nos aportaron los mejores resultados. A continuación hemos escogido a modo de ejemplo dos de ellas, las fotografías 9 (izquierda) y 10 (derecha). Nótese la posición del tablero en cada foto, nos referiremos en el apartado f) a estas dos imágenes. Cabe mencionar el efecto que produce la distorsión en ambas imágenes, las líneas se vuelven curvas y esta deformación se hace más notoria cuanto más alejados nos encontramos del centro de la lente.

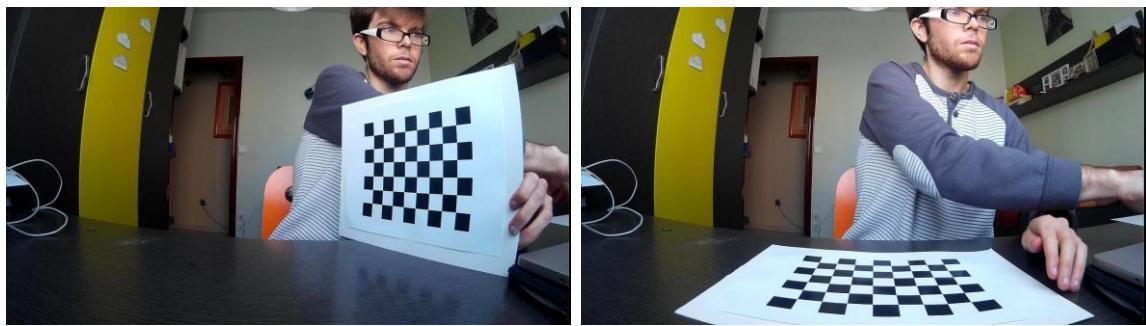


Figura 2-8: Fotografías 3 (izq.) y 10 (derecha) tomadas en el proceso de calibración

- Extraer las esquinas del tablero:** es un proceso que realiza automáticamente a través de la función *cornerfinder*. Esta función está basado en el algoritmo Harris. Este algoritmo a su vez, tiene como principio que una esquina tiene una baja autosimilitud con su vecindad<sup>6</sup>. En la imagen sucesiva, podemos ver un ejemplo de cómo sería la pantalla de MATLAB una vez que ha encontrado las esquinas del tablero. En este caso encuentra los 42 vértices.

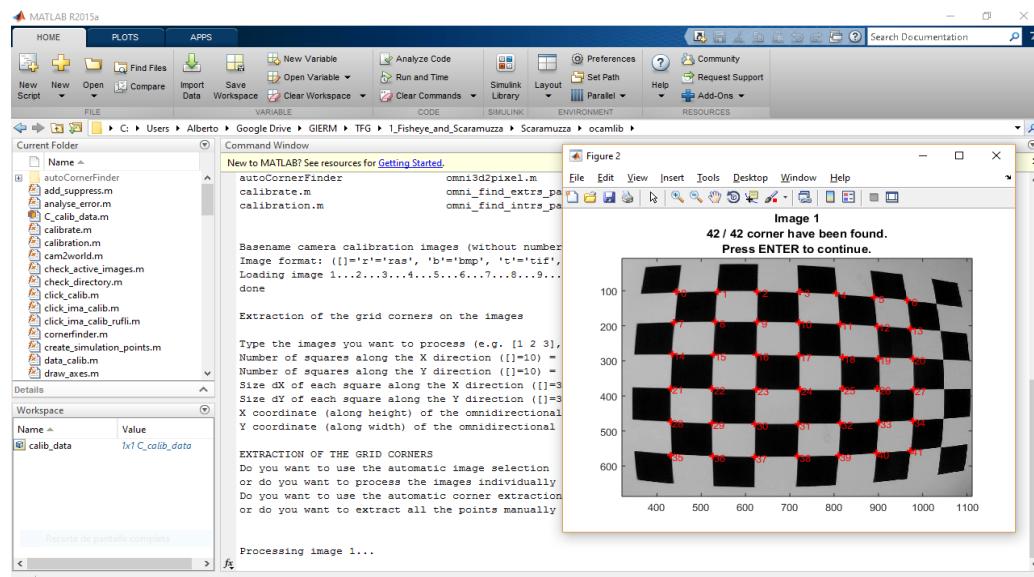


Figura 2-9: Extracción de esquinas imagen 1

- Ajuste:** en determinadas condiciones, debido al ángulo con el que se toma la fotografía o a factores

<sup>6</sup>O lo que es equivalente; se parece poco a su vecindad, la suma de diferencias al cuadrado (SDC) es alta, en todas las direcciones. La SDC se puede calcular mediante un tensor. Una SDC alta en todas las direcciones implica que los autovalores del tensor sean significativos. En esto último se basan las principales técnicas de búsqueda de esquinas.

lumínicos, el algoritmo no encuentra todas las esquinas o puede hallar una esquina en una posición inexacta. Para ello, hay una herramienta que permite añadir o ajustar los puntos manualmente.

- e) Calibración: elegimos el orden del polinomio. El de orden cuatro, como nos proponía inicialmente la interfaz, nos ofrece los mejores resultados. El modelo nos ofrece una relación entre el ángulo  $\theta$  y radio  $r$  (ver figura 2-6). Además, tal y como vimos en el apartado anterior, la relación entre la distancia desde el centro de la imagen en píxeles ( $r$ ) y los grados del haz que va del centro de la lente al objeto, es lineal.

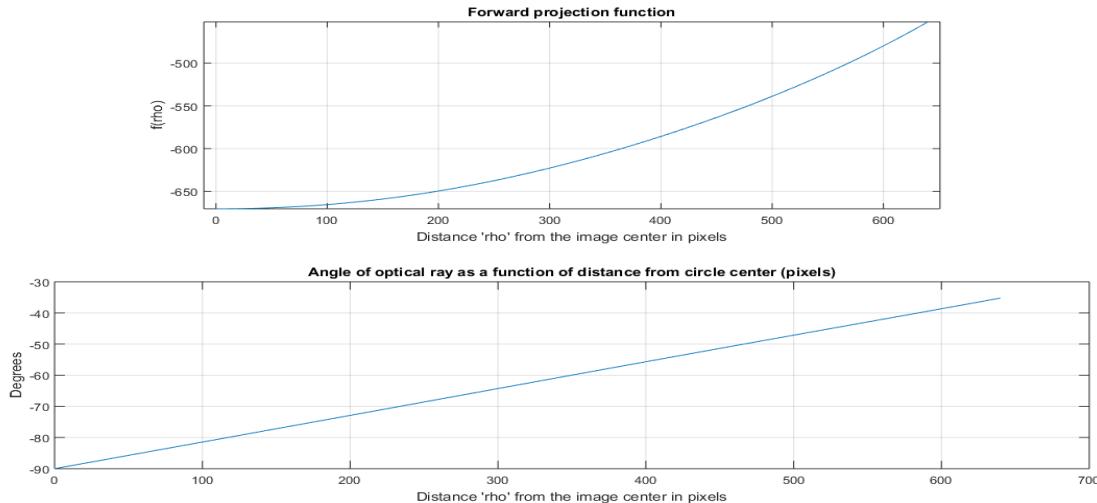


Figura 2-10: Ecuación de proyección y relación ángulo de incidencia-distancia

- f) Encontrar el centro: tras obtener el polinomio, utilizamos la herramienta *findcenter*, la cual inicia un algoritmo iterativo que va encontrando el centro que se ajusta mejor a la re-proyección de las esquinas de cada casilla de cada fotografía. Una vez tenemos el modelo se vuelven a proyectar los puntos obtenidos y se calcula el error (tabla 2-2). Podemos ver también la ubicación de las fotografías respecto del centro de la lente con la herramienta *show extrinsic* (figura 2-10).

Nº IM.	1	2	3	4	5	6	7	8	9
ERROR	$0.51 \pm 0.23$	$0.60 \pm 0.31$	$0.62 \pm 0.33$	$0.22 \pm 0.13$	$0.47 \pm 0.25$	$0.58 \pm 0.42$	$0.65 \pm 0.40$	$0.40 \pm 0.24$	$0.36 \pm 0.24$
Nº IM.	10	11	12	13	14	15	16	17	18
ERROR	$0.68 \pm 0.48$	$0.79 \pm 0.50$	$0.97 \pm 0.47$	$0.36 \pm 0.20$	$1.70 \pm 0.67$	$0.58 \pm 0.39$	$1.47 \pm 0.88$	$0.51 \pm 0.26$	$0.43 \pm 0.23$

Tabla 2-2: Media de los errores reproyección de cada imagen

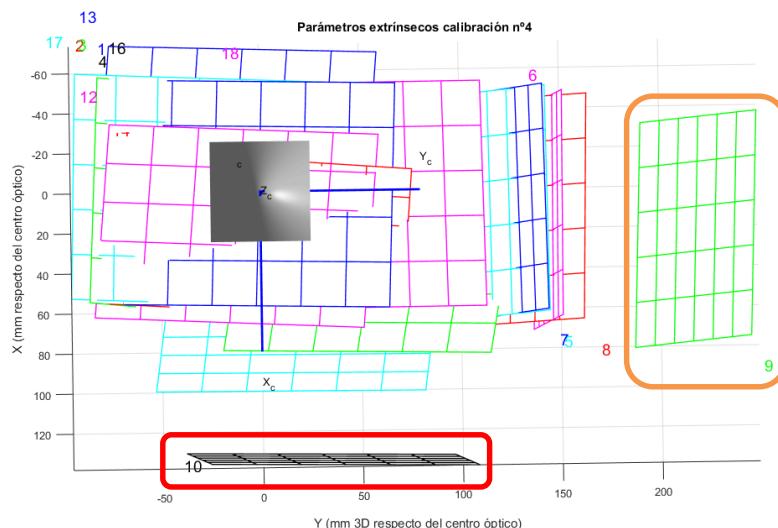


Figura 2-11: Posición de las imágenes (izquierda) y fotografía 9 (derecha)

Observando las figuras 2-10 y 2-7 vemos la correspondencia del experimento con los resultados obtenidos. En la figura 2-10 vemos como la fotografía 10 (encuadrada en rojo) es la que ocupa la posición más baja y, si lo recordamos, la fotografía 10 de la figura 2-7 estaba efectivamente apoyada en la mesa. De la misma forma, remarcamos como la fotografía 9, figura 2-10 (encuadrada en naranja), es la que ocupa la posición más a la derecha, y si lo recordamos, la fotografía 9 de la figura 2-7 vemos como se posiciona más a la derecha.

- g) Afinamiento: una vez encontrado el centro, es posible mejorar los resultados con una herramienta que incorpora la librería. Finalmente, conseguimos los siguientes resultados:

*Average error [pixels]: 0.661220*

*Sum of squared errors: 558.084823*

Teniendo en cuenta que cada fotografía tiene  $720 * 1980 = 1425600$  píxeles y tenemos un error de 0.66 píxeles, el error cometido en toda la fotografía es inapreciable. Sin embargo, estamos en dos dimensiones, por lo que sería más correcto hablar en términos de área. El área de incertidumbre sería de  $\pi * 0.66^2 = 1.34$  píxeles lo que continua siendo pequeño, en órdenes de magnitud para hacernos una idea sería como el de una pelota de tenis en un campo de fútbol.

La librería nos devuelve el modelo completo y otras variables interesantes, las cuales se adjuntan en; Anexo C: Parámetros de retorno – Occam Calib.

## 2.3 Transformaciones (I): del mundo a la cámara

Una vez que tenemos el modelo, calcular como sería la posición de un punto del espacio trasladado al plano de la cámara es algo poco complejo. Recordando la ecuación 2-2, los pasos a seguir serían:

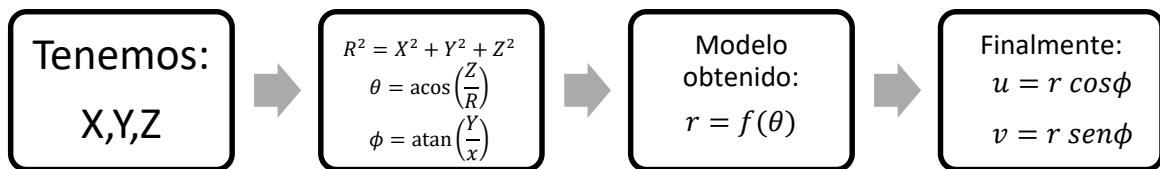


Figura 2-12: Desarrollo modelo completo directo

La librería incorpora una función para ello, llamada *world2cam*. Esta función recibe como parámetros de entrada una matriz de puntos del espacio y una estructura. Dicha estructura contiene el modelo obtenido en la calibración de la cámara.

Esta función la analizaremos más adelante ya que nos servirá para re-distorsionar la imagen.

## 2.4 Transformaciones (II): de la cámara al mundo

Esta transformación es más interesante ya que nos permite a partir de una fotografía de un objeto, estimar cual es la posición real que ocupa dicho objeto respecto a la cámara. El modelo inverso sería igual que el de la figura 2-11 pero recorrido en sentido inverso y utilizando el polinomio inverso en vez del directo ( $f^{-1}$  en vez de  $f$ ).

### 2.4.1 Introducción

La transformación inversa no es inmediata ya que tenemos un sistema indeterminado, existen más incógnitas que ecuaciones. El objetivo es pues añadir restricciones al problema o averiguar alguna de las tres componentes de forma que el problema quede determinado.

Hay varios métodos que nos permiten realizar la transformación inversa. En terrenos no uniformes o exteriores, donde no podemos tomar referencias fijas, un medio muy utilizado es la visión estéreo.

Al tomar una fotografía de un mismo objeto desde dos perspectivas distintas, conociendo las relaciones entre ambas cámaras, podemos añadir restricciones al problema y hallar la solución. Cabe mencionar el plano epipolar, que es aquel que pasa por el punto en cuestión, y los dos centros ópticos de las imágenes. Una vez obtenido, podemos obtener más relaciones interesantes conociendo las relaciones con otros puntos.

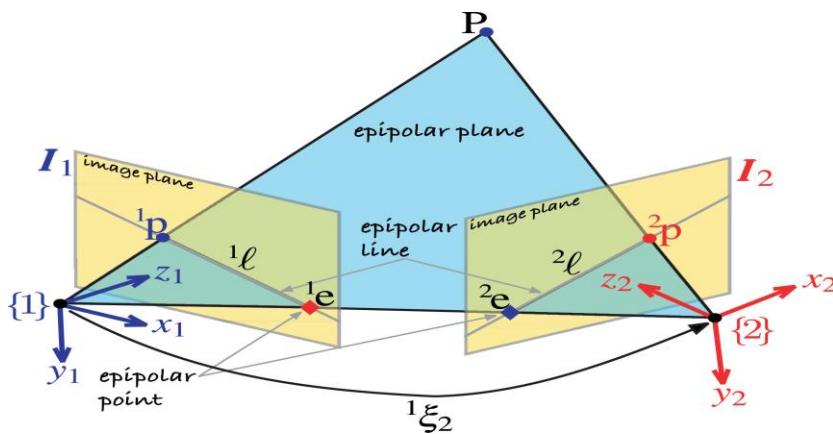


Figura 2-13: Plano epipolar

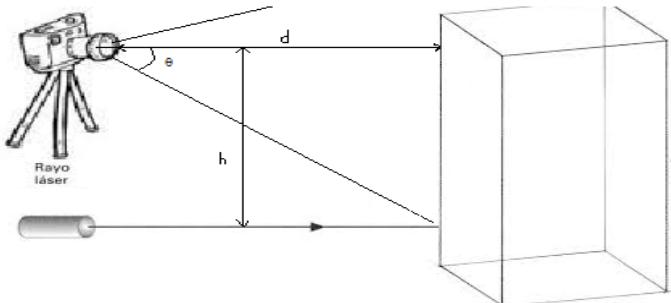
Sin embargo, uno de los principales inconvenientes de este método es que la asociación de imágenes resulta muy compleja; puede haber puntos de interés que se ocultan de una perspectiva a otra, condiciones lumínicas desfavorables que afecten sólo a una cámara, o simplemente puntos que resulte difícil asociarlos.

Debido a que los métodos de asociación resultan muy complejos y añadir una segunda cámara de las mismas características aumentaría significativamente el precio del robot, descartamos esta opción.

Dado que nuestro robot va a circular por interiores, la distancia del robot al techo va a permanecer constante a lo largo de todo el trayecto. En un primer momento pensamos en que podríamos usar un sensor de distancia para medir la altura del techo, pero para esa distancia no nos valdría un sensor de ultrasonidos de bajo coste. Por lo que barajamos la opción de un sensor de distancia por láser, el cual incrementaría también el precio del proyecto. Finalmente, nos dimos cuenta que mediante un láser cualquiera y la cámara en conjunto, podríamos medir la distancia al techo, aprovechándonos justamente de las propiedades del modelo inverso y de las restricciones que impondríamos entre el láser y la cámara.

## 2.4.2 Midiendo la profundidad: el láser

El principio que utilizamos consiste en aplicar relaciones trigonométricas entre la cámara, el láser y el techo para añadir una restricción al problema inverso y junto con el modelo de la cámara, poder obtener las coordenadas del punto en el espacio. La situación sería la siguiente:



$$\begin{aligned} \tan(\theta) &= \frac{h}{d} ; \quad d = \frac{h}{\tan(\theta)} \quad (\text{I}) \\ r &= f(\theta) \quad (\text{II}) \\ \theta &= f^{-1}(r) \quad (\text{III}) \end{aligned}$$

Ecuación 2-3: Ecuación distancia cámara-láser

Figura 2-14: Relaciones láser-cámara

Por lo tanto el problema se reduce a encontrar la distancia del centro óptico en la foto al láser, hallar el polinomio inverso (ec. 2-3 (III)) y realizar un simple cálculo (ec 2-3 (II)). Lo más costoso fue encontrar un método fiable que localizara el láser en la mayoría de los casos y con buena exactitud, ya que una pequeña desviación produciría cambios importantes en la distancia medida.

### 2.4.2.1 Encontrando el láser

Primeramente, decidimos aprovechar las propiedades del propio láser, es decir, que tiene una sola longitud de onda. Además como nuestro láser era rojo y filtrar por los colores primarios (RGB) es trivial, decidimos actuar por ahí. El algoritmo busca un color rojo superior a cierto umbral y coge el que encuentra más cerca del centro óptico de la imagen. Esto último es porque la cámara está relativamente cerca del láser, concretamente a 22 centímetros, que es lo que nos permitía la plataforma donde estaba apoyado. Para ver la función completa ver en: Anexo D: Algoritmo detección del láser (I).

Posteriormente, decidimos implementar otro método más fiable, basado en métodos morfológicos<sup>7</sup> y algunas de las propiedades utilizadas en el algoritmo anterior. Los métodos morfológicos utilizan operaciones no lineales en las imágenes, son muy robustos en determinadas condiciones, ofrecen gran flexibilidad y tienen una programación sencilla. Concretamente utilizamos la transformación *white top-hat*.

La transformación *white top-hat* se define como la imagen original menos la imagen tras la apertura:

$$I_{wth} = I - I \circ b ; \quad o \equiv \text{operador de apertura}$$

A su vez la apertura de una imagen consiste en una erosión que elimina los elementos pequeños, seguida de la dilatación que reconstituye la forma original. En la siguiente figura lo podemos ver más claro:



Figura 2-15: Apertura de una imagen.

<sup>7</sup> La idea detrás de los métodos morfológicos es simple: comparar localmente la imagen con una plantilla local llamada elemento estructural y realizar las operaciones correspondientes atendiendo a si cumple ciertas condiciones de contorno o no.

Por lo tanto si a la imagen original, le restamos la última obtenida, obtendríamos solamente los elementos pequeños, que es justamente lo que nos interesa ya que queremos recuperar un puntero láser en una fotografía de una habitación. Para ver el algoritmo completo ver Anexo E: Algoritmo detección del láser (II).

En las figuras siguientes podemos notar los resultados obtenidos:



Figura 2-16: Imagen del techo de la habitación en escala de grises



Figura 2-17: Imagen del techo tras aplicar métodos morfológicos

Vemos como en una situación desfavorable, es decir, con luz exterior y la luz encendida, mientras que el primer algoritmo fallaba, este nos ofrece buenos resultados. Fijándonos en la imagen 2-16 vemos cerca del centro solamente dos puntos, uno que se corresponde con el láser y otro provocado por ciertos reflejos del sol. Finalmente el algoritmo encuentra correctamente el láser al ser el más cercano al centro de la imagen.

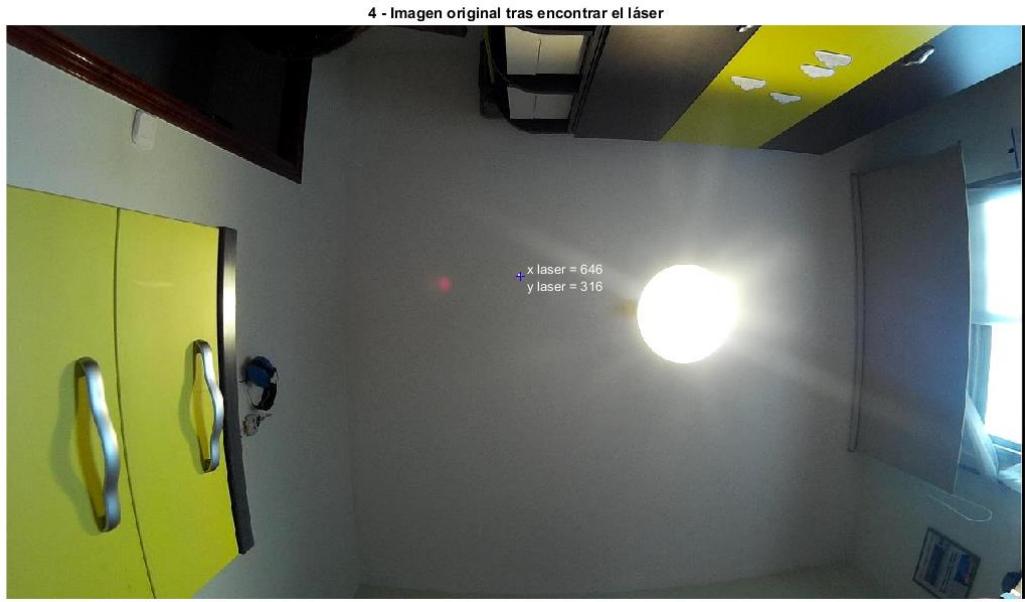


Figura 2-18: Imagen original tras encontrar el láser

#### 2.4.2.2 Midiendo distancias

Ya tenemos el primer paso, hemos conseguido la distancia del centro al laser ( $r$ ). Ahora, el objetivo es calcular el ángulo theta para poder aplicar las ecuaciones 2-3. Una opción es calcular el polinomio inverso como mencionamos y sustituir en dichas ecuaciones.

La librería nos ofrece una función muy potente, llamada *cam2world* la cual recibe un punto de la cámara y refleja cuál sería su ubicación real en una esfera de radio unidad. Como solo nos interesa el ángulo y la  $R$  sería uno, para la obtención de dicho ángulo bastaría con calcular el arcoseno a la coordenada Z (ecuación 2-2 (II)):

```
M1=cam2world([y_laser_f;x_laser_f],fisheye_model);
z=M1(3); % la tercera coordenada
theta=acos(z); % R=1           → ecuación 2-2 (II)
zp = regla/tan(theta);      → ecuación 2-3(I)
```

Obtenemos los siguientes resultados:

```
M1 =
-0.026743617931752
0.087962951745574
-0.995764680042492
```

```
zp =
-2.707690042179121
```

Obtenemos una distancia al techo de 2.7 metros, lo que concuerda con las medidas realizadas manualmente de la distancia de la cámara al techo.

Tras obtener la distancia a la pared basta con multiplicar las otras dos componentes del vector que obtenemos de cam2world ( $x, y$ ) por el factor de la distancia y obtendríamos la X, Y, Z del punto en cuestión, por lo que el problema inverso estaría finalizado, concluyendo así este primer hito. Podríamos obtener igualmente la ubicación real de cualquier punto de la imagen. A modo aclaratorio se muestra el proceso completo seguido, y por último una imagen de cómo se efectuaron las mediciones y comprobaciones.



Figura 2-19: Proceso seguido para hallar el modelo inverso

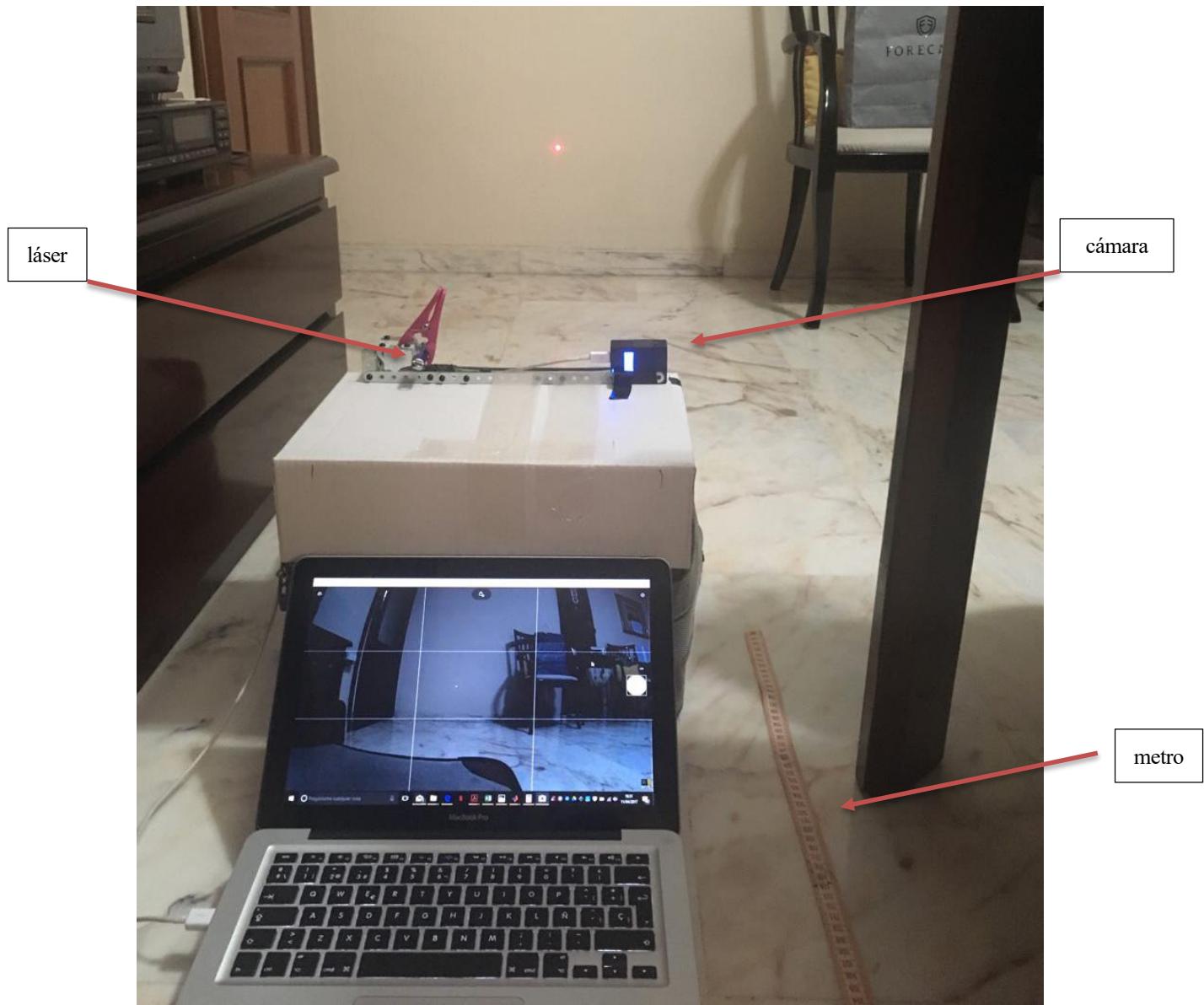


Figura 2-20: Experimento medición y verificación distancia

## 2.5 Cobertura, conclusiones y áreas de mejora: HITO A

Para finalizar esta sección nos gustaría resumir cual ha sido el porcentaje de cobertura de cada objetivo, para conocer en qué zonas hemos actuado bien y en cuales se podría haber mejorado. Viendo el siguiente gráfico:

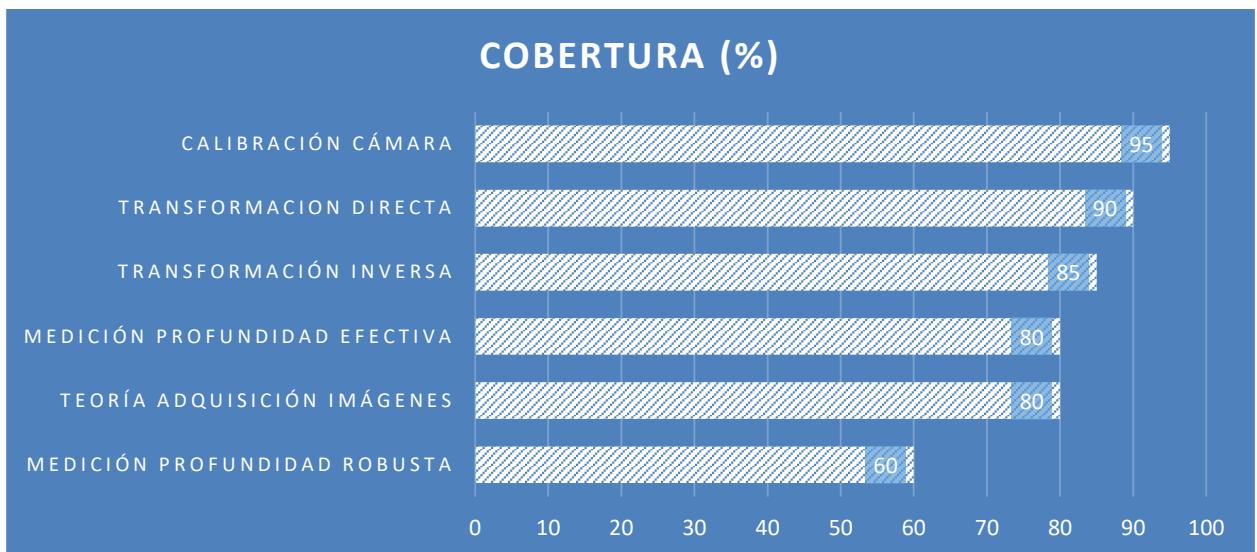


Figura 2-21: Porcentaje de cobertura hito A

Vemos como en general hemos cumplido satisfactoriamente todos los objetivos; desde la calibración de la cámara a la estimación de profundidad. Como punto negativo, los métodos de obtención del láser como comentamos anteriormente, no son tan robustos como nos hubiera gustado, ya que ante ciertas condiciones (láser cerca de una lámpara o una fuente de luz fuerte) puede dar lugar a error.

Las conclusiones de este hito son las siguientes:

- Al introducir una lente con un ángulo de visión elevado, la distorsión que se observa es también notable, por lo que prima la obtención de una calibración exhaustiva y un modelo preciso.
- La plataforma láser-cámara tiene que ser muy rígida y nivelada con respecto a la superficie a medir. Errores en algunas de estas precondiciones suponen errores significativos en las mediciones reales. A más distancia, mayor serán estos errores.
- En general el método láser-cámara para medir distancias es satisfactorio, y el precio es de los más bajos si ya contamos con una cámara. El precio de un puntero láser comercial es de 3 euros.
- La visión estéreo permite obtener distancias de cualquier punto que se encuentre en las dos imágenes, con la desventaja de que la asociación de dichos puntos es algo compleja y a veces no es posible.

En cuanto a áreas de mejora cabe destacar:

- Sería muy interesante trabajar con una cámara que disponga también de filtro infrarrojo, ya que permitiría paliar las condiciones lumínicas.
- Elegir un láser con un ángulo de dispersión menor y un color que no sea combinación lineal de ninguno de los colores propios de techos.

# 3 HITO B: CARACTERÍSTICAS

*“La belleza es el acuerdo entre el contenido y la forma”*  
-- Henrik Johan Ibsen --

Esta sección es crítica en el proyecto, ya que la información que obtengamos de las imágenes será la que introduzcamos en la estimación del mapa y la localización del robot. Una medida de una característica errónea podría provocar un falso positivo, que podría llegar a convertir inestable el sistema.

El hito se valida por tanto encontrando un método fiable que sea capaz de detectar los puntos característicos de la imagen en cuestión. Como dijimos en la introducción, nuestro robot va observando continuamente el techo, nuestros puntos de interés serán principalmente las cuatro esquinas de la habitación (a partir de ahora ceilmarks) y otros objetos igualmente invariantes como las lámparas.

En primer lugar haremos una introducción a la extracción de características, hablaremos de los distintos acercamientos que hemos realizado, y nos centraremos en el algoritmo elegido. Comentaremos las distintas ventajas e inconvenientes de cada uno y por último comentaremos técnicas más avanzadas que pese a ser valoradas, finalmente han sido descartadas del proyecto.

## 3.1 Introducción al reconocimiento de formas

El primer acercamiento fue usar las técnicas de reconocimientos de formas ya estudiadas. El procedimiento de un sistema automático de reconocimiento de formas (SARF), es siempre el mismo; separar las regiones, calcular las características de cada región y aplicar un clasificador. A la salida del clasificador nos dice que región es de cada clase y cuál coincide por tanto con la que estamos buscando.

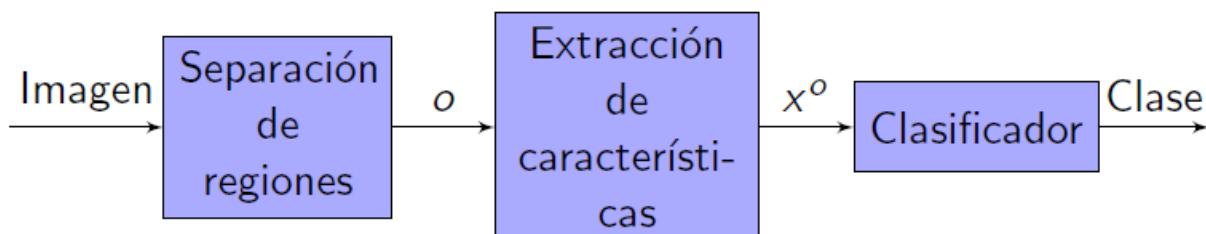


Ilustración 3-1: esquema sistema automático de reconocimiento de formas

En cuanto a la separación de regiones se refiere, existen diversos métodos; mediante umbral, derivadas espaciales, seguimiento de contornos, métodos morfológicos, etc. Más adelante abordaremos algunos de ellos, ya que lo emplearemos en nuestro algoritmo.

Una vez separada las regiones, debemos extraer las características de cada región. Si por ejemplo queremos separar manzanas y naranjas, una buena característica sería el color, mientras que la forma podría inducir a error. Podríamos elegir otras características como la homogeneidad de su superficie, o el tamaño.

Todas estas características irían a un clasificador, el cual pondera de alguna forma todas las características respecto de un ideal de clase y discrimina si la región en cuestión pertenece a una clase o a otra. Por ejemplo, si la región en cuestión es una manzana, extraeríamos el color, el tamaño y la forma y lo compararíamos con una manzana o una gama de manzanas “ideales” y, pese a que la manzana esté mordida por ejemplo, llegaríamos a la conclusión que es una manzana y no una naranja.

Antes de acudir al problema real, probamos algunos ejemplos. En la siguiente figura podemos ver el problema de clasificación de tres clases de frutas con dos características.

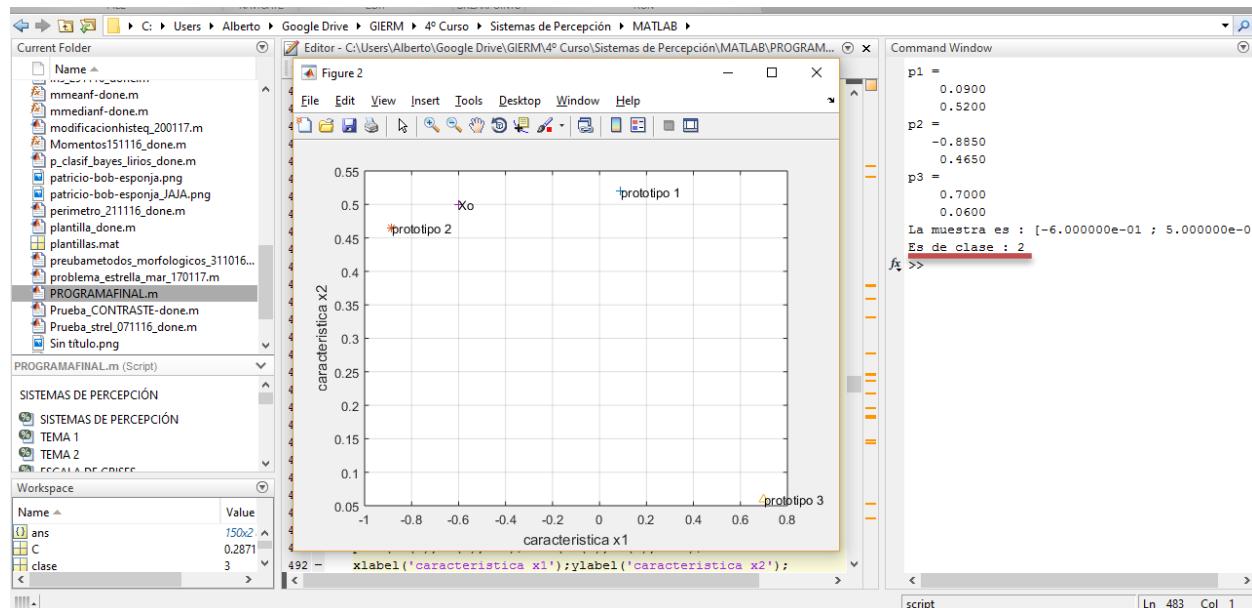


Figura 3-1: Ejemplo clasificación de características

Vemos como el objeto Xo, se clasifica finalmente como de clase 2. Podemos ver el código completo en el Anexo F: Ejemplo código clasificador características.

Al implementar este método con imágenes reales del techo, encontramos grandes dificultades al separar las regiones, ya que su entorno es complejo con múltiples líneas, formas y sombras distintas, con características similares.

Decidimos entonces no seguir con este método y buscar otras alternativas. Leyendo el libro “Digital Image Processing Using Matlab” [1] nos encontramos con una sección que nos llamó la atención: “Procesado de imágenes mediante la transformada de Fourier”. Esto junto a la frase que nos dijo nuestro profesor Alfredo “nosotros vemos en Fourier” nos alentó a coger ese camino.

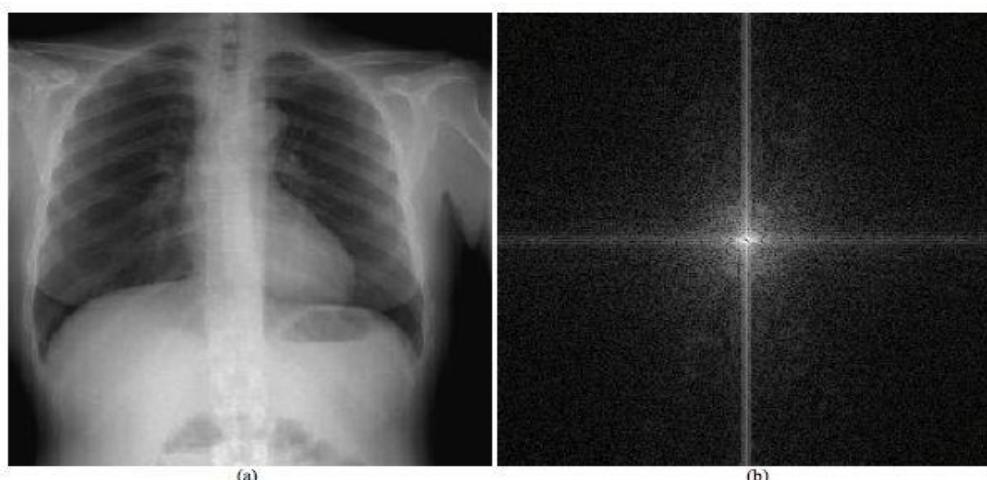


Figura 3-2: Radiografía y su respectiva transformada de Fourier

### 3.2 Filtrado mediante Transformada de Fourier

La Transformada de Fourier separa una función en sus componentes frecuenciales. Para imágenes en 2D, el resultado es el mismo; analiza la componente frecuencial de las imágenes. Todo lo estudiado en análisis y filtrado de señales en 1D es aplicable a 2D, sólo que hay que tener una visión espacial de la transformada. Las puertas y su transformada el seno cardinal de 1D pasarían a ser los prismas y senos cardinales de 2D.

Dado la naturaleza discreta de las imágenes, cuando nos refiramos a la transformada, hablaremos en términos de Transformada de Fourier en Discreta (DFT de sus siglas en inglés) en dos dimensiones. La DFT se define como:

$$\hat{I}[f_x, f_y] = \sum_{k_x=0}^{W-1} \sum_{k_y=0}^{H-1} I[k_x, k_y] e^{-j2\pi \left( \frac{f_x k_x}{W} + \frac{f_y k_y}{H} \right)}.$$

Ecuación 3-1: Transformada discreta de Fourier 2D

La transformación inversa (IDFT) sería la siguiente:

$$I[k_x, k_y] = \frac{1}{WH} \sum_{f_x=0}^{W-1} \sum_{f_y=0}^{H-1} \hat{I}[f_x, f_y] e^{j2\pi \left( \frac{f_x k_x}{W} + \frac{f_y k_y}{H} \right)}.$$

Ecuación 3-2: Transformada inversa discreta de Fourier 2D

A continuación para hacernos una idea más clara mostraremos una imagen de un pasillo a la izquierda y su correspondiente transformada a la derecha:

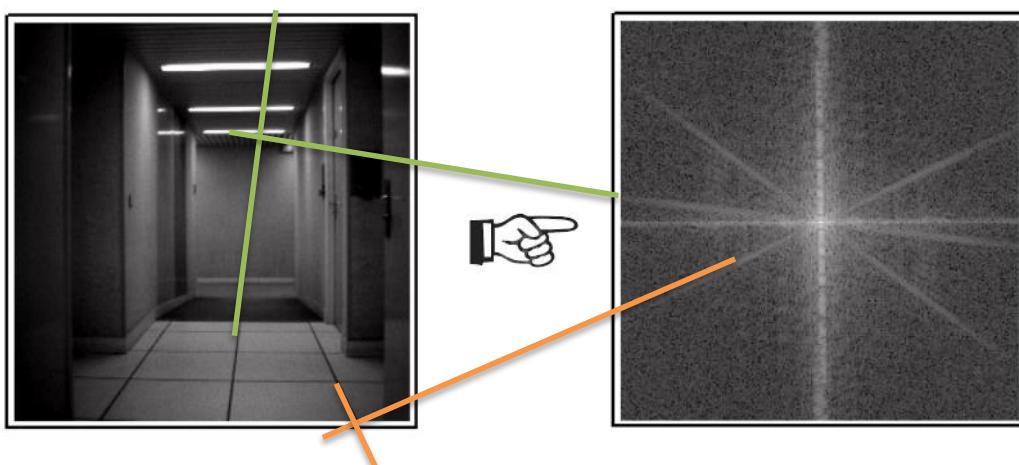


Figura 3-3: Imagen del pasillo y DFT

En la figura de la DFT (derecha) las bajas frecuencias se encuentran en la cruz principal, encontraremos siempre una cruz brillante ya que corresponderá con el marco de la imagen. Conforme más nos alejamos de la imagen, más altas son las frecuencias. Podemos ver la correspondencia de las líneas con la DFT. El motivo de que formen 90° entre sí es el siguiente; por ejemplo la línea verde es estrecha en el eje x y larga en el eje y. Esto se transforma en una alta frecuencia en el eje x (ancho en el eje x) y una baja frecuencia en el eje y (estrecho en el eje y). Esto provoca este aparente giro, que simplemente es provocado por el paso a frecuencia de la imagen.

La ventaja es que al saber que las líneas van a estar giradas en Fourier, podemos establecer la correspondencia, y saber qué líneas nos está mostrando la transformada, como hemos hecho con la línea verde o la naranja.

En la siguiente figura se explica gráficamente como sería proceso de filtrado en frecuencia:

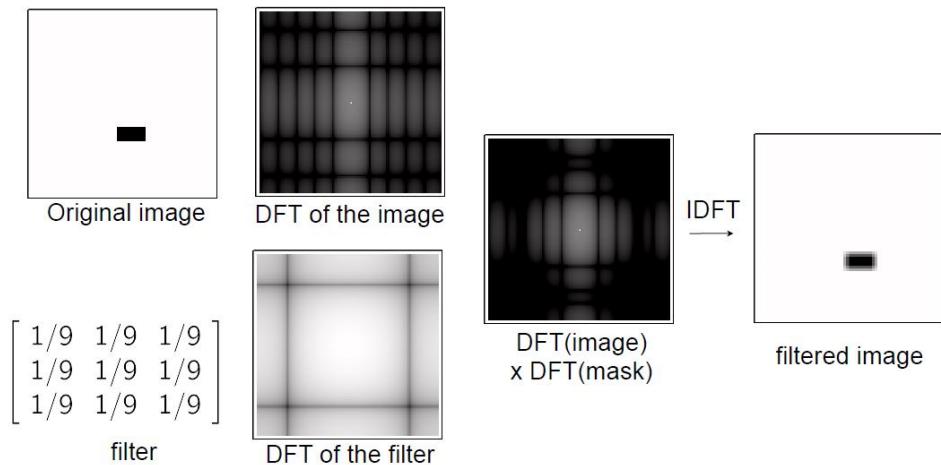


Figura 3-4: Filtrado de imágenes en frecuencia

Primeramente calculamos la DFT de la imagen y elegimos el filtro deseado en frecuencia ya sea paso alto, bajo o banda. Dado que la convolución espacial se transforma en una multiplicación en frecuencia, bastaría con multiplicar ambas DFT (término a término) y obtendríamos la imagen filtrada en frecuencia. Por último para volver al dominio temporal bastaría con aplicar la IDFT.

Partimos de las precondiciones que la habitación se ve completamente, sus cuatro esquinas son visibles. En primer lugar procedimos a usar un paso alto, para poder discriminar las cuatro líneas que separan el techo de las paredes, elegimos un Butterworth de orden tres. El código lo podemos ver en: Anexo G: Detección de esquinas DFT

La imagen original es una foto tomada desde la perspectiva del robot del techo de la cocina. Para hacernos una idea de las dimensiones, contamos con unos diez metros cuadrados.



Figura 3-5: Imagen original de la cocina

En primer lugar vemos como hay una esquina que permanece oculta y se puede confundir fácilmente con la línea del techo debido a la continuidad del mueble a lo largo de la parte superior. Por otro lado observamos la fuerte distorsión que sufren estas líneas, si nuestro cometido es filtrar por dichas líneas debemos re-distorsionar la imagen. Esto lo explicaremos más adelante.

Tras re-distorsionarla y convertirla a escala de grises, calculamos la DFT de cada imagen, como se muestra a continuación:

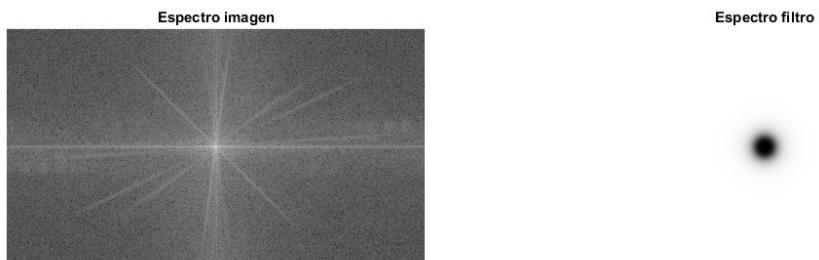


Figura 3-6: DFT Imagen desdistorsionada y filtro paso alto

Tras multiplicar ambas y volver al dominio temporal obtenemos la imagen filtrada:

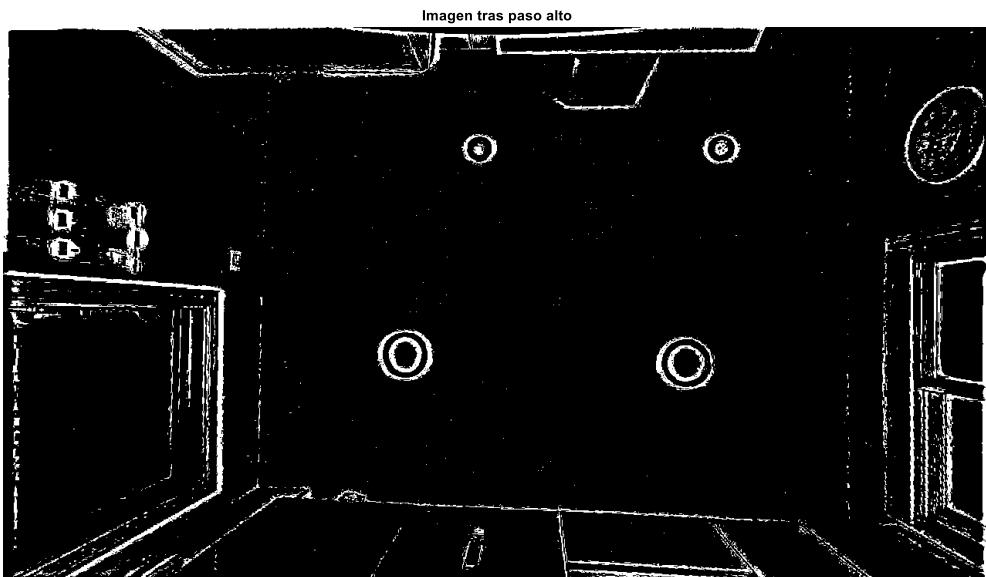


Figura 3-7: Imagen filtrada paso alto

Volvemos a caer en problemas similares a los que obtuvimos con el reconocimiento de formas; la imagen es compleja, tiene muchas más aristas que el techo. El paso alto no es por tanto el adecuado para nuestro objetivo.

Prestando atención a la figura, el techo está paralelo al plano imagen. Siempre que sepamos la localización del robot en el mapa y el ángulo inicial, podremos rotar la imagen para ponerla en esta pose. Esto nos da una propiedad interesante. Las líneas que buscamos serán paralelas a los bordes de la foto. Por tanto no sería nada descabellado recurrir a un filtro específico que filtrara en frecuencia las líneas coincidentes con el marco (la cruz principal). Dejaremos una tolerancia ya que estas líneas no serán siempre del todo paralelas como se ve en la figura anterior.

Probamos varios filtros específicos y la imagen filtrada mostraba efectivamente solamente las líneas paralelas al marco, pero el resultado seguía sin ser satisfactorio; el marco de la ventana era mucho más paralelo que las propias líneas en cuestión lo que hacía que elegir el máximo de esta imagen filtrada, nos indujera en numerosos casos a error.

Otra propiedad interesante de las líneas que andamos buscando es que ocupan la mayor parte de la pantalla, por lo que no sería de una frecuencia muy alta en Fourier. Ajustando el filtro, obtuvimos unos resultados distintos. Ahora se reconocía mejor las líneas objetivo, pero el marco seguía teniendo el gradiente más elevado, lo que impedía diferenciar las líneas en cuestión.

Por último se nos ocurrió aprovechar una propiedad interesante de la transformada; al hacer filtros con puertas obtendríamos senos cardinales, lo que nos introducirían negativos en las zonas de alto gradiente. Al no haber más imagen fuera del marco, el marco no tendría esta componente negativa.

Por lo tanto lo que hicimos fue encontrar en cada línea cuales eran los puntos más negativos, hacer un histograma para todas las filas y elegir la media. Hicimos lo mismo para las columnas y sorprendentemente el resultado fue satisfactorio.

Para entenderlo un poco mejor, hemos cogido una imagen de habitación ideal (Anexo H: Imagen habitación Ideal) y le hemos aplicado nuestro filtro. La representación 3D de la imagen filtrada sería:

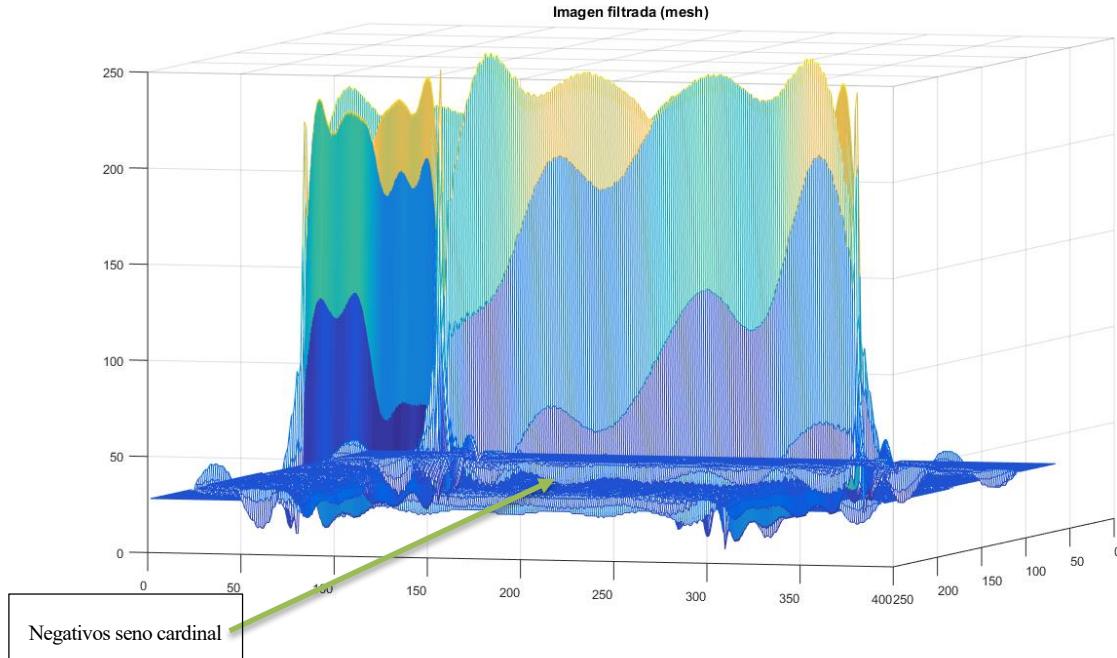


Figura 3-8: Imagen ideal filtrada

Vemos efectivamente como los máximos (al no haber otros elementos en la imagen ideal) coinciden con las líneas en cuestión. Observamos el efecto del seno cardinal que vuelve negativa los puntos justo al lado del máximo. De hecho si miramos la imagen desde abajo, veríamos justamente las cuatro líneas que andamos buscando:

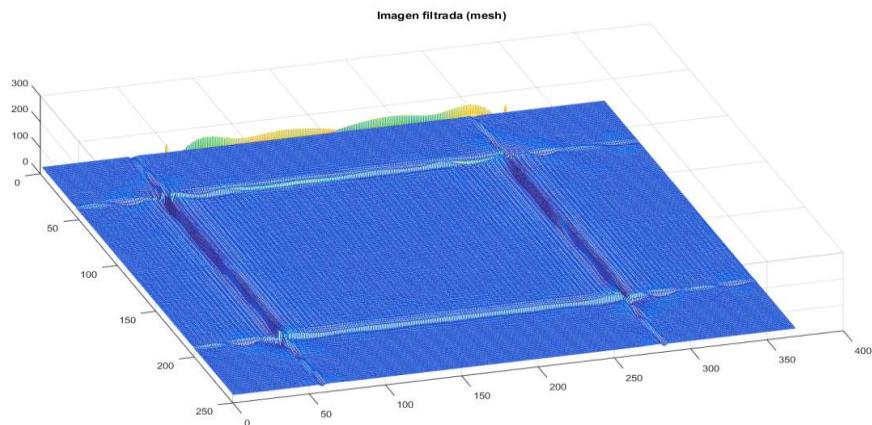


Figura 3-9: Vista inferior imagen ideal

Calculando el punto de corte de estas líneas obtendríamos directamente las cuatro esquinas, que es justamente lo que andábamos buscando.

En el caso real, las imágenes serían las siguientes:

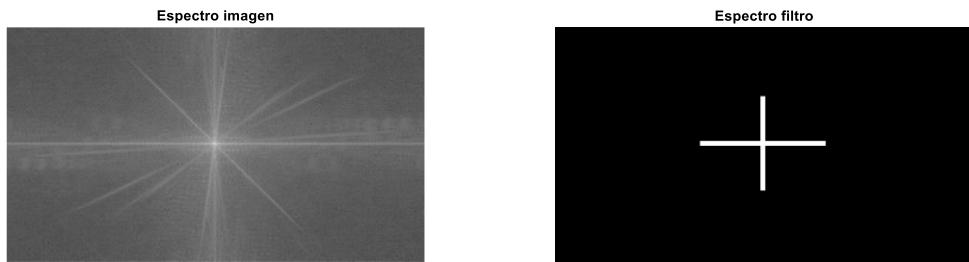


Figura 3-10: DFT Imagen desdistorsionada y filtro específico

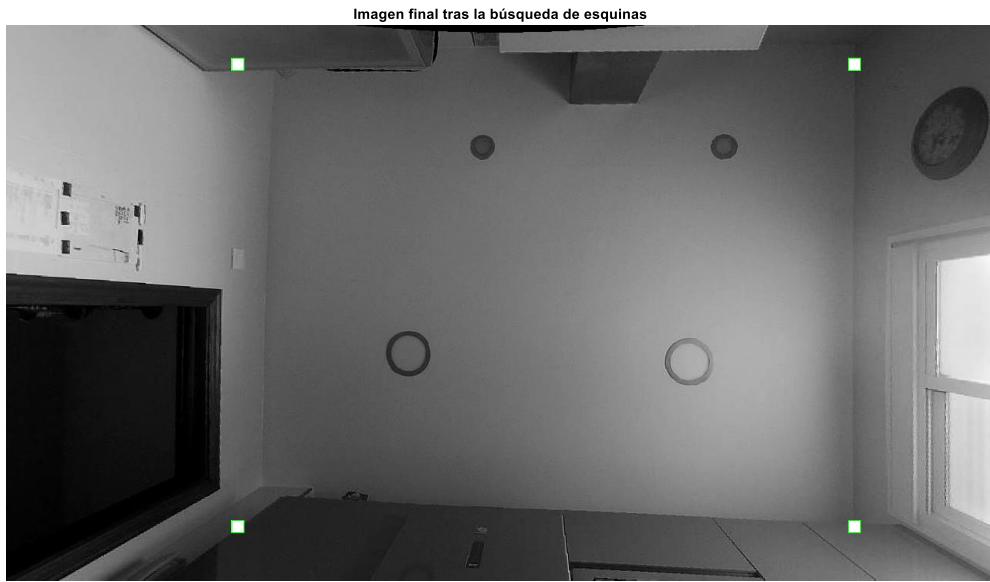


Figura 3-11: Imagen real tras la búsqueda de esquinas

Comprobamos que el algoritmo encuentra satisfactoriamente las cuatro esquinas. Tras probar con varias fotografías en condiciones distintas y sitios distintos dimos el método por válido, aunque no por definitivo. Las ventajas y desventajas de este algoritmo son las siguientes:

Ventajas	Desventajas
Es rápido, tarde de media 1,5 segundos, de los cuales 1,25 son solo de carga de la imagen, el modelo y la desdistorsión	Tienen que aparecer las cuatro esquinas, en un caso real esto no ocurrirá en la mayoría de casos
Es muy intuitivo, nos permite ajustar el filtro hasta obtener los resultados deseados, podríamos establecer un ajuste automático teniendo las esquinas bien localizadas una primera vez.	El hecho de tener las líneas paralelas al marco, por un lado no facilita para nada la resolución. Por otro lado, si se realiza en otra posición, el filtro no valdría. Habría que reajustarlo.
La implementación en cualquier código es muy sencilla, simplemente bastaría con definir funciones para la DFT y la IDFT y realizar multiplicaciones de matrices	Los marcos, muebles, u otros objetos presentan una gran dificultad para el filtro. Pudiendo dar esquinas que realmente no lo son. No hay corrección de errores.

Tabla 3-1: Ventajas y desventajas de nuestro filtrado en frecuencia

Debido a las desventajas nombradas, y que el resultado no era del todo satisfactorio, decidimos emplear un tercer método, que fuera capaz de detectar esquinas aunque no estuvieran las cuatro en la imagen y supiese interpretar lo que está pasando con el resto de ellas. El método consiste en una malgama de técnicas específicas para este tipo de casos. Lo hemos titulado el método del rectángulo creciente.

### 3.3 Método del rectángulo creciente

Este método fue evolucionando bastante a lo largo del proyecto a medida que encontrábamos problemas. Para no extendernos más de lo necesario, hablaremos en exclusiva de la última versión. El método parte de las siguientes precondiciones:

- El robot se encuentra inicialmente cerca del centro de la habitación, aunque no es imprescindible.
- El techo es liso, no dispone de elementos que puedan ser confundidos con líneas. Pueden tener lámparas y otros elementos.
- La iluminación debe ser adecuada, de lo contrario el algoritmo no funciona adecuadamente. En un caso real el robot podría tener un sensor LDR y proporcionar él mismo la iluminación si necesario.

Dado que los principales problemas que nos encontramos eran que el entorno del techo (muebles, marcos, puertas, ventanas,...) perturbaban las líneas del techo, decidimos hacer un algoritmo que partiera del centro y fuera buscando hacia afuera las posibles líneas de techo, siendo consciente que en algún momento toparía con ellas, antes de interferir con el resto de elementos.

Los pasos que sigue este algoritmo son:

1. Desdistorsionar la imagen, para poder trabajar con líneas.
2. Obtener la imagen binaria de contornos óptima, para evitar problemas de ruido.
3. Barrer rectangularmente el techo hasta encontrar una línea o el fin de la imagen e interpretar los resultados, calculando las esquinas que están disponibles.
4. Re-distorsionar la imagen y los puntos obtenidos.

Tras obtener las esquinas en la imagen distorsionada, aplicando modelado inverso podríamos calcular su ubicación en el espacio, e introducirlo como medidas válidas como elementos de localización.

#### 3.3.1 Desdistorsión de la imagen

Como ya mencionamos anteriormente, si vamos a trabajar con líneas, necesitamos aliviar los efectos de distorsión que produce la lente ojo de pez. Para ello, la librería nos proporciona una función (Anexo I: Función undistorts) que realiza la mayoría del proceso siendo necesario ajustar solo algunos parámetros. Si recordamos el apartado 2.2.3 (calibración de la cámara) recordaremos que obtuvimos un polinomio que nos relacionaba la distancia al centro de la imagen con el ángulo que formaba el haz de luz que va desde el objeto al centro de la lente. Si la cámara fuera estenopeica, esta relación sería lineal. Por tanto visto de otra forma tenemos la información de cuanto se distorsiona la imagen. La función encuentra el polinomio inverso (función findinvpoly) y realiza la función world2cam que ya vimos, pero esta vez con el polinomio inverso.



Figura 3-12: Imagen original y desdistorsionada

Parámetros que recibe:

- Occam model: modelo que ya hemos visto, resultado de la calibración. Anexo C: Parámetros de retorno – Occam Calib.
- A: Imagen original de la cámara
- Fc: Como el modelo inverso nos lo da con un radio unidad, podemos elegir el factor de escala. Sería el radio habitual que captaría. Si lo cambiamos simplemente observaremos un zoom en la imagen. Se ha elegido fc=2 ya que da los mejores resultados.
- Display: permite mostrar el resultado o no.

### 3.3.2 Imagen binaria de contornos óptima

Para trabajar con líneas utilizaremos la transformada de Hough como explicaremos más adelante, y para ello nos es necesario tener una imagen binaria de contornos lo más ajustada posible. El principal obstáculo que nos encontramos es que las condiciones de iluminación provocan que no podamos utilizar el mismo algoritmo para cualquier situación, si no debería adaptarse al contexto. El efecto es que aparece una imagen muy ruidosa.

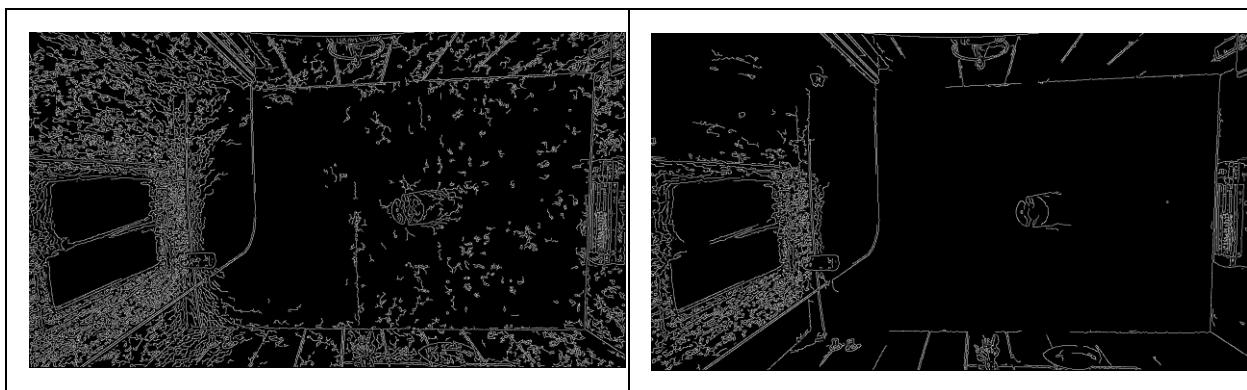


Figura 3-13: Imagen binaria de contornos, sin ajuste y ajustando la imagen

La función que realiza la imagen binaria de contornos se denomina Edge. Tiene como parámetros la imagen de origen, el método y el nivel de threshold. En cuanto al método, realizamos numerosas pruebas, el más satisfactorio fue el *Canny*. El nivel de threshold es el que se autoajusta según la foto.

La idea es que las líneas de interés de una imagen binaria sin ruido tienen un peso de alrededor del 10% de la imagen completamente llena. Si toda la imagen estuviera llena su suma tendría un valor de  $(M \times N)^8$ <sup>8</sup>. Inicialmente, el algoritmo calcula la imagen binaria con un nivel muy bajo (sería similar a la figura 3-11 izquierda), obtiene su suma y la compara con el 10% de  $M \times N$ . Si esta suma es mayor, seguimos subiendo el nivel hasta que la suma sea inferior a dicho valor (figura 3-11 derecha). Escrito en MATLAB sería:

```

while (found==0 && tr<1)
    BW = edge(img_u,'Canny',tr);
    tr=tr+inc;
    if (show==1), imshow(BW); shg; end;
    err=sum(sum(BW))/MN;
    if (err<=tol)
        found=1;
        break;
    end
end

```

<sup>8</sup> M es el número de filas de la imagen, mientras que N es el número de columnas. En nuestro caso  $M \times N$  sería igual a 921600

### 3.3.3 Encontrando las líneas

Una vez que tenemos la imagen binaria, vamos buscando las cuatro líneas o las que podamos encontrar. El eje principal de esta parte es la transformada de Hough, por lo que la explicaremos brevemente. La transformada de Hough es una técnica para la detección de figuras en imágenes digitales. Con la transformada de Hough es posible encontrar todo tipo de figuras que puedan ser expresadas matemáticamente, tales como rectas, circunferencias o elipses. El algoritmo de la transformada de Hough usa una matriz, llamada acumulador, cuya dimensión es igual al número de parámetros desconocidos del problema. Por cada punto en la imagen, se buscan todas las posibles figuras a las que puede pertenecer ese punto. Las figuras se pueden detectar buscando las posiciones del acumulador con mayor valor (máximos locales en el espacio del acumulador).

El pseudocódigo sería el siguiente:

- 1: cargar imagen
- 2: detectar los bordes en la imagen
- 3: por cada punto en la imagen:
  - 4: si el punto está en un borde:
  - 5: por todos los posibles ángulos:
    - 6: calcular para el punto con un ángulo
    - 7: incrementar la posición en el acumulador
  - 8: buscar las posiciones con los mayores valores en el acumulador
  - 9: devolver las rectas cuyos valores son los mayores en el acumulador.

Se ha establecido el siguiente convenio para el orden de búsqueda:

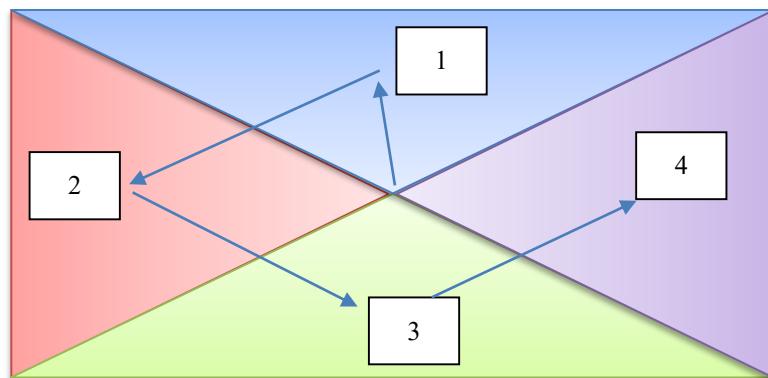


Figura 3-14: Orden de búsqueda

El rectángulo está inicialmente en el centro de la imagen. Según el orden que vemos en la figura se va expandiendo en la dirección de búsqueda y cuando encuentra una línea con las características correspondientes a esa región, la guarda y pasa a buscar la siguiente línea.

Tras encontrar las líneas acude a una función que hemos creado llamada *interpretline*, que se subdivide en dos funciones; *mexcludeline* y *mexcludeline\_nall*. Estas dos son muy parecidas salvo que para la primera intentamos encontrar siempre las cuatro esquinas ayudándonos de las anteriores<sup>9</sup> y la segunda sólo nos devuelve de las que tiene datos.

El programa emplea otras funciones auxiliares, como *calcuvH* que nos dice en qué región se encuentra la línea encontrada o la función *mpunto\_corte* que calcula el punto de corte de dos líneas dados dos pares de puntos. La función completa la podemos ver en:

<sup>9</sup> La suposición que se hace es que al final de la línea detectada está la esquina, lo que no ocurre siempre. Es válido incluso si solo hay tres líneas o dos. En el caso de haber solo una línea o ninguna devuelve error, ya que no habría ninguna esquina en la foto.

Anexo J: Función completa extracción de características.

### 3.3.4 Redistorsión de los puntos obtenidos

Una vez encontramos las esquinas deseadas, toca volver a calcular cuales serían estos puntos en la imagen original<sup>10</sup> para poder hacer la transformación inversa y obtener cuales serían los puntos en el mundo real. Realmente, podríamos aprovechar que ya hemos hecho la transformación inversa al re-distorsionar la imagen, e incluirlo en el algoritmo *undistorts*. Por motivos de claridad, lo hemos separado en otra función; ver Anexo K: Función de re-distorsión.

### 3.3.5 Resultados

Tras analizar varias fotografías en distintas condiciones, llegamos a la conclusión que es un buen método y funciona en un alto porcentaje de veces, aunque no es muy genérico y poco versátil. A continuación expondremos un ejemplo del cuarto de baño esta vez, con una iluminación adecuada.



Figura 3-15: Imagen original en escala de grises tras la búsqueda de esquinas

Las ventajas y desventajas de este método son:

Ventajas	Desventajas
Es más inmune a condiciones adversas de iluminación. Debido a la imagen binaria de contornos adaptada a la iluminación.	Es más lento que Fourier, tarda de media unos 2,20 segundos de los cuales 1,25 son solo de carga de la imagen, el modelo y la desdistorsión.
No tienen por qué aparecer las cuatro esquinas, además es capaz de interpretar los resultados y añadir alguna esquina si alguna línea no estuviera clara.	Es ineficiente en algunas ocasiones, ya que si sólo aparece la esquina número cuatro por ejemplo, pasaría antes por la uno, la dos y la tres aunque no haya nada.
Las líneas no tienen por qué ser paralelas al marco, simplemente tienen que estar dentro de una de las cuatro zonas.	La implementación no es tan inmediata, se mezclan funciones del autor, de la librería de Scaramuzza y propias de MATLAB. Además, ocupa 538 líneas en comparación con Fourier, que tenía unas 100.

Tabla 3-2: Ventajas y desventajas método rectángulo creciente

<sup>10</sup> Recordemos que la imagen sobre la que hemos trabajado es la que estaba desdistorsionada, por lo que hay que deshacer el cambio

### 3.4 Mediando el mundo real

Ya sabemos en qué píxeles están las esquinas, serán nuestras primeras y principales medidas sobre las que se apoyará el SLAM. Tras esto, intentamos añadir el reconocimiento de las lámparas, ya que, como mencionamos en la introducción, cuantos más puntos de referencia tenga el SLAM más efectivo será este. Debido a que encontramos dificultades en el reconocimiento y este hito se estaba alargando más de lo debido, decidimos abandonar temporalmente esta idea para ver si con lo que teníamos era suficiente.

El último paso que quedaría para obtener las medidas reales en metros, sería unir el proceso de medición de distancias del láser que vimos en: *Midiendo la profundidad: el láser, a la extracción de características que hemos visto en esta sección.*

Más concretamente, los pasos a seguir son los siguientes:

1. Medir la distancia al techo con el láser y la cámara (una vez).
2. Reconocer los elementos de referencia que tengamos (como vimos en más atrás en 3.3).
3. Calcular sus coordenadas en píxeles.
4. Pasar estos puntos del mundo 2D al mundo 3D con *cam2world*<sup>11</sup>.
5. Escalar el vector (X,Y,Z) por el factor de distancia al techo.

En la siguiente figura, podemos ver el análisis de la cocina, que es el habitáculo más pequeño y que mejor iluminación tenía.

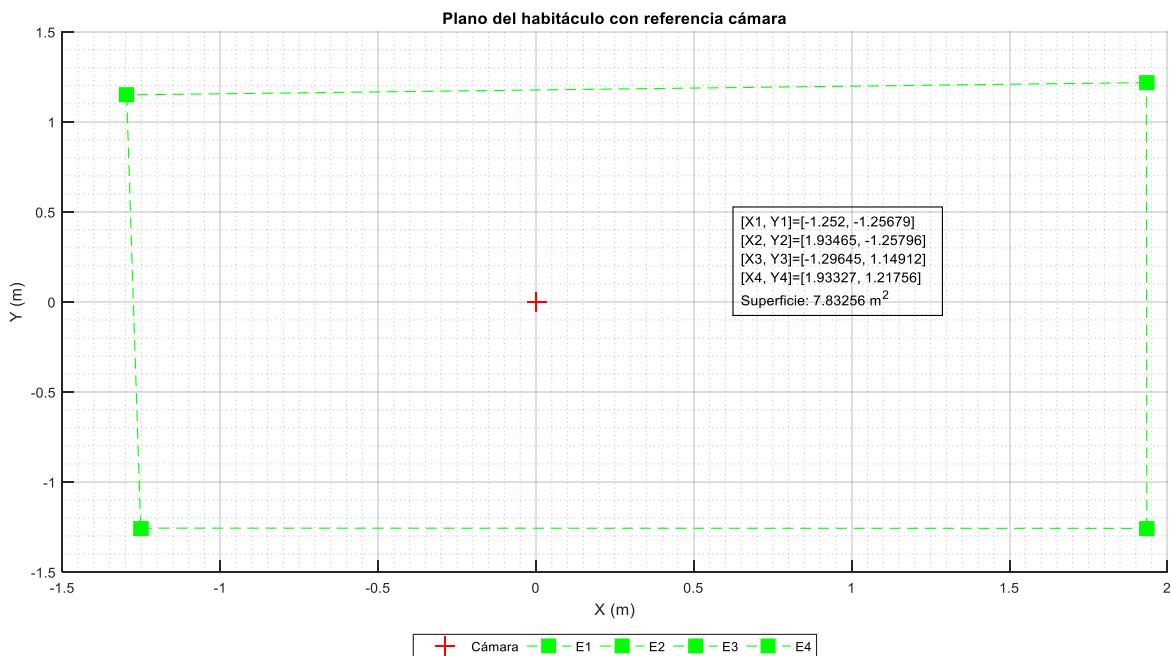


Figura 3-16: Posición de las esquinas respecto de la cámara en metros

Observamos la posición de las esquinas en metros respecto de la cámara, ubicada en el centro. Teniendo las estas medidas podemos calcular las dimensiones del habitáculo y su superficie como se muestra en la figura. Las medidas reales se acercan con bastante exactitud a los datos reales, tras varios experimentos medimos una desviación media de 8 cm.

Dado que esta sección ha ocupado gran parte del proyecto, el algoritmo funciona y los resultados son buenos siempre que la imagen siga unas determinadas condiciones, damos por válido este método y finalizamos así este hito.

<sup>11</sup> Si recordamos esta función, nos daba como resultado el vector que va desde el centro de la lente al objeto en el mundo real, escalado a factor unidad. Es por ello que para hacer el modelado inverso, es necesario tener la referencia de la distancia, como dijimos en el hito de visión.

### 3.5 Cobertura, conclusiones y áreas de mejora: HITO B

Para finalizar esta sección nos gustaría resumir cual ha sido el porcentaje de cobertura de cada objetivo, para conocer en qué zonas hemos actuado bien y en cuales se podría haber mejorado. Viendo el siguiente gráfico:

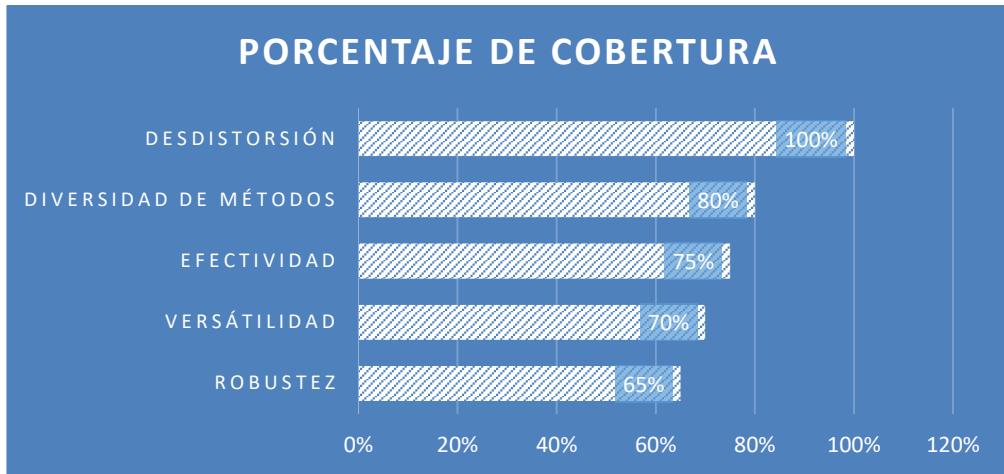


Figura 3-17: Porcentaje de cobertura HITO B

Observamos que en general los objetivos se han cumplido satisfactoriamente, resaltando como puntos débiles; la obtención de un método más universal a cualquier condición del techo y más robusto frente a perturbaciones que pueda tener la imagen.

Las conclusiones de este hito son las siguientes:

- Encontrar un método fiable, versátil y efectivo que detecte automáticamente las esquinas del techo es una tarea más compleja de lo que nos esperábamos. Hacen falta métodos más avanzados o hacer un híbrido de los métodos que ya hemos visto.
- Es importante conocer la orientación del robot, ya que eso nos da una información crucial de donde están las líneas principales respecto de la cámara y cuanto habría que rotar la imagen para que el algoritmo sea más efectivo.
- Sería conveniente revisar la desdistorsión de la imagen, debido a que es un poco lenta. Podríamos guardar directamente los parámetros del modelo en la función para evitar hacer la carga del archivo. Por otro lado podríamos intentar trabajar con las líneas distorsionadas, aunque sería algo más complejo.

Por último cabe mencionar algunas de las mejoras que podrían hacer mejorar las funcionalidades de la extracción de características:

- Redes neuronales: Son inspiradas en el comportamientos de las neuronas y conexiones del cerebro humano tratando de crear un programa, sistema o máquina que sea capaz de solucionar problemas difíciles, actuar de forma humana y realizar trabajos pesados mediante técnicas algorítmicas convencionales. Tiene aplicaciones en el reconocimiento de formas, pero es algo complejo y se escapa del alcance de este proyecto.
- RANSAC: Random Sample Consensus (RANSAC) es un método iterativo para calcular los parámetros de un modelo matemático de un conjunto de datos observados que contiene valores atípicos. Es una herramienta muy potente para el reconocimiento de líneas.

# 4 HITO C: SLAM

---

*“Los científicos estudian el mundo tal como es; los ingenieros crean el mundo que nunca ha sido”.*

- Theodore von Karman -

Afrontamos en este hito el fin último que da título a este presente proyecto; lograr la localización y el mapeado simultáneo con una cámara gran angular. En este capítulo, comenzaremos explicando conceptos sobre los que se apoya el método de mapeo y localización simultánea (SLAM) que será el siguiente apartado. Profundizaremos en nuestra implementación, como ha sido la toma de decisiones y como la hemos llevado a cabo. Por último como venimos haciendo en los dos hitos anteriores comentaremos la cobertura de objetivos, las conclusiones y las áreas de mejora.

## 4.1 Introducción

Como comentamos en la introducción, el problema de la localización y mapeado simultáneos consiste en descubrir si es posible para un robot móvil navegar a través de un entorno desconocido y construir, de manera incremental un mapa mientras que determina, al mismo tiempo, su posición dentro de este mapa.

Los objetivos de este hito serían por tanto:

- Entender, aplicar y dominar tanto el filtro de Kalman como su versión extendida.
- Lograr entender y desarrollar algún ejemplo de SLAM.
- Simular el SLAM para nuestro caso.
- Implementar el SLAM con los datos reales que vamos obteniendo.

En la sección estado del arte (sección 1.2), descubrimos las distintas tipologías de SLAM y cuál sería la nuestra, aparte de los distintos algoritmos. Entre ellos, el más extendido y el que hemos elegido; el Filtro de Kalman Extendido (EKF).

## 4.2 El filtro de Kalman (KF) y sus versiones no lineales (EKF, UKF)

### 4.2.1 KF

El filtro de Kalman es un algoritmo desarrollado por Rudolf E. Kalman en 1960 que consiste en un conjunto de ecuaciones matemáticas que proveen una solución recursiva óptima, por el método de mínimos cuadrados. La meta de esta solución consiste en calcular un estimador lineal, insesgado y óptimo del estado de un sistema en cada instante de tiempo, a partir de la información disponible en el instante anterior y la información adicional disponible en ese instante. El filtro se desempeña suponiendo que el sistema puede ser descrito a través de un modelo estocástico lineal<sup>12</sup>, en donde el error asociado tanto al sistema como a la información adicional que se incorpora en el mismo tiene una distribución normal con media cero y varianza determinada.

El filtro de Kalman tiene numerosas aplicaciones en tecnología. Una aplicación común es la guía, navegación y control de vehículos, especialmente naves espaciales. Además el filtro es ampliamente usado en campos como procesamiento de señales y econometría.

Una de las ventajas del filtro es al ser un algoritmo recursivo, este puede correr en tiempo real usando únicamente las mediciones de entrada actuales, el estado calculado previamente y su matriz de incertidumbre, no requiere ninguna otra información adicional.

El filtro se compone de una fase de predicción, en la que el modelo del sistema predice cual debería ser el nuevo

---

<sup>12</sup> Modelo cuyo estado es función lineal de sus entradas y el estado anterior y cuyo estado es una variable aleatoria que evoluciona a lo largo del tiempo.

estado y una fase de corrección, en la que se compara el estado estimado con las medidas observadas. Esta comparación se amplifica o se atenúa por la ganancia de Kalman ( $K$ ) que se va autorregulando y ponderando la importancia de las discrepancias de las medidas y las predicciones con el paso del tiempo. En la siguiente figura vemos un esquema del proceso completo.

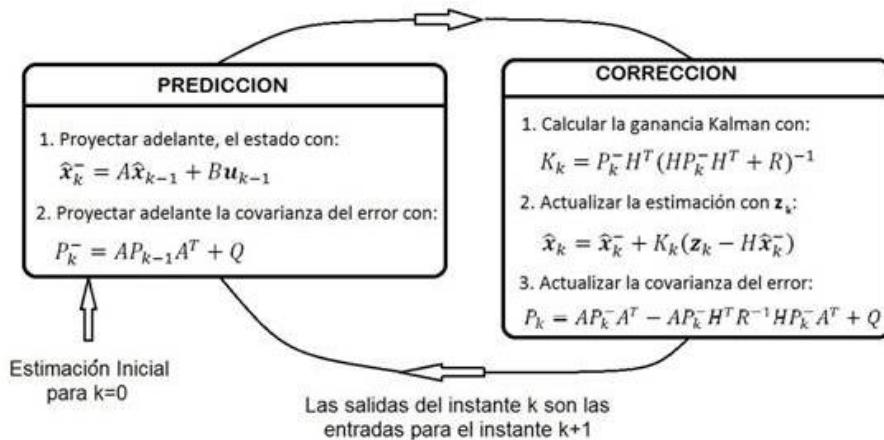


Figura 4-1: Esquema KF

Hicimos un algoritmo ejemplo y tras unos ajustes validamos el buen funcionamiento del filtro y nos percatamos la importancia del valor de dichos de parámetros ( $Q$ ,  $R$ ,  $P$ ). Debido a que nuestro sistema es no lineal, no podíamos recurrir a esta versión del filtro, necesitamos la versión para sistemas no lineales; el filtro de Kalman extendido.

#### 4.2.2 EKF

El filtro de Kalman extendido (EKF: Extended Kalman Filter) consiste en una variación del filtro de Kalman para abordar el problema de estimación del estado cuando el modelo es posiblemente no-lineal. Como veremos en la siguiente figura, el procedimiento es el mismo que en la figura 4-1, sólo que a cada paso es necesario linealizar el modelo.

Ahora, las densidades de distribución no son necesariamente gaussianas ya que el modelo no es lineal. No obstante, se aproximan estas densidades de distribución por densidades gaussianas. El EKF es entonces una forma de obtener aproximaciones de primer orden a los términos óptimos. Cuando el modelo es severamente no lineal, estas aproximaciones pueden generar esperanzas y covarianzas muy distintas a las esperanzas y covarianzas reales que pueden conducir a un desempeño muy pobre del filtro o incluso a su divergencia.

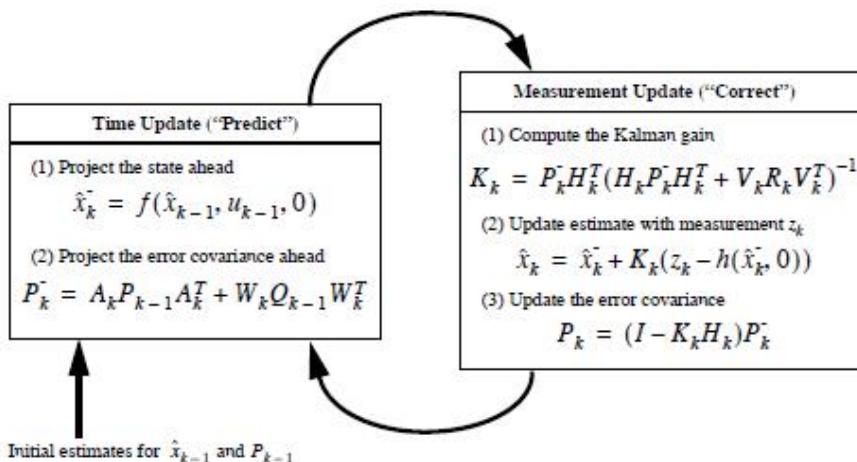


Figura 4-2: Esquema del EKF

Otro inconveniente del filtro es que requiere el cálculo de las matrices jacobianas, cuyo cálculo no es trivial en la mayoría de los casos y restringe la versatilidad del filtro. Para nuestro modelo del vehículo en primer lugar decidimos obtener los jacobianos a mano, de forma que solo era necesario sustituir con los valores necesarios. Posteriormente encontramos una función más potente que calculaba el EKF con unos algoritmos de computación más avanzados y tiene como ventaja que se puede pasar cualquier modelo como una función MATLAB y él mismo se encarga de calcular y evaluar los jacobianos, reduciendo enormemente el costo y el tiempo de computación. Ver Anexo L: Algoritmo EKF utilizado

#### 4.2.3 UKF

El filtro de Kalman extendido es el algoritmo de estimación más ampliamente aplicado en la estimación en sistemas no lineales. Sin embargo, a menudo proporciona estimaciones poco fiables si los sistemas no lineales son severos; para estos casos se han desarrollado otros algoritmos como, por ejemplo, el filtro de Kalman “unscented”.

La transformación “unscented” es un método propuesto por Julien y Uhlmann; posteriormente ha sido perfeccionado por Wan y van der Merwe. Una transformación “unscented” está basada en dos principios fundamentales. Primero, es fácil realizar una transformación no lineal en un punto singular. Segundo, no es muy difícil encontrar un conjunto de puntos individuales en un espacio de estados cuya función de densidad de probabilidad muestral se aproxime a la verdadera función de densidad de probabilidad.

Tomando estas dos ideas juntas, se supone que se conoce la media y la covarianza del vector  $(\bar{x}, P)$ , y se busca un conjunto de puntos determinísticos llamados sigma-puntos cuya media y covarianza son iguales a  $(\bar{x}, P)$  respectivamente. Después se aplica la función no lineal conocida  $f(x)$  a cada punto para obtener los transformados. La media y covarianza de los puntos transformados proporcionarán un buen estimador de la verdadera media y covarianza de  $f(x)$ .

La principal ventaja de este algoritmo es que no necesita de los jacobianos. Pese a ser computacionalmente más costoso, ofrece mejores resultados ante no linealidades, es por ello (y alguna ventaja que veremos más adelante) que vamos a elegir este método en lugar del EKF. El esquema del UKF sería el que sigue:

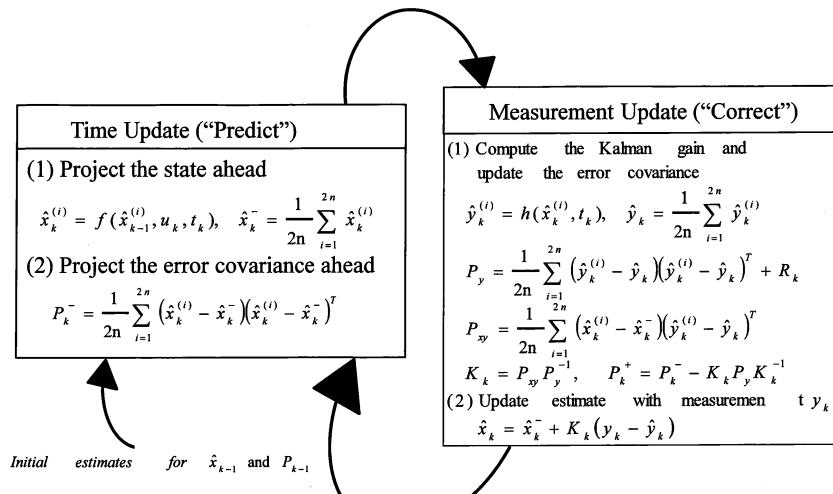


Figura 4-3: Esquema UKF

Al igual que para el EKF, elegiremos una función ya desarrollada que ofrece los mejores resultados debido a sus alto componente en algoritmos computación avanzados. Ver Anexo M: Algoritmo UKF utilizado.

### 4.3 Ceilmarks

Como veremos más adelante, para ubicarse el robot se basa en la odometría y en las observaciones que realiza a lo largo de su movimiento. A estas observaciones se les conoce como puntos de referencia o landmarks, las cuales deben cumplir ciertas propiedades que veremos a continuación. Tal y como mencionamos al principio de la memoria, nuestro robot tomará medidas del techo principalmente, por lo que llamaremos ceilmarks a estos puntos de referencia.

En la siguiente tabla comentaremos los requisitos necesarios para la elección de un buen punto de referencia [7] y argumentaremos por qué los ceilmarks elegidos son una buena opción o no:

REQUISITO	CEILMARK
Debe ser posible observarla desde distintas posiciones y ángulos.	Si la relación ángulo de visión-superficie a explorar es buena, podemos observar al menos un ceilmrk desde casi cualquier punto del habitáculo.
Debe ser lo suficientemente única como para poder ser distinguida de otras características. Es decir, si uno observa dos características y las re-observa en el futuro, debe ser posible identificar cuál es cuál.	Conociendo la localización del robot la posición de un ceilmrk es distinguible usando criterios simples como el de mínima distancia (lo veremos más adelante).
Si decidimos que algo debe ser un landmark, este debe permanecer estacionario. Una referencia en movimiento provocaría datos y estimaciones erróneas.	Nuestro landmarks (esquinas, lámparas y marcos de ventanas o puertas) son por definición estáticos.
Deben ser suficientes como para que el robot no circule durante largos períodos de tiempo sin observar ningún landmark	Como hemos comentado eligiendo un buen ángulo de visión-superficie (Anexo N: Relación Ángulo visión superficie) podemos observar un ceilmrk desde prácticamente cualquier punto de la habitación.

Tabla 4-1: Cumplimiento de requisitos de nuestros ceilmarks

### 4.4 Localización y mapeo simultáneo (SLAM)

El proceso del SLAM consiste en un cierto número de pasos: extracción de características, asociación de datos, estimación del estado y actualización de las características. La finalidad del proceso es usar el entorno para actualizar la posición del robot. Dado que la odometría<sup>13</sup> del robot no es enteramente fiable, no podemos depender directamente de ella. Debemos usar sensores de distancia, con sus errores también, para corregir esta posición. Aquí es donde entra el filtro ya que teniendo en cuenta los errores de odometría y los sensores es capaz de llegar a una solución óptima (en el caso lineal).

Los pasos del SLAM son los siguientes:

1. El robot se mueve, como tenemos el modelo del robot podemos estimar su posición (odometría). La incertidumbre aumenta.
2. Corregir la estimación tras la re-observación de los ceilmarks. Por un lado, re-observando los ceilmarks sabemos sus posiciones. Por otro, usando la estimación de la posición del paso previo es posible estimar dónde deberían estar ubicados. Esta diferencia de información es la que utiliza el filtro para estimar el estado. Se reduce la incertidumbre.
3. Finalmente, comenzaremos nuevamente el ciclo, añadiendo nuevos ceilmarks al mapa del robot. Utilizaremos la información actualizada sobre posición e incertidumbre obtenida en este paso.

<sup>13</sup> Odometría: La odometría es el estudio de la estimación de la posición de vehículos con ruedas durante la navegación. Para realizar esta estimación se usa información sobre la rotación de las ruedas para estimar cambios en la posición a lo largo del tiempo. La palabra "odometría" se compone por las palabras griegas hodos ("viajar", "trayecto") y metron ("medida").

#### 4.4.1 El robot se mueve, modelo de movimiento

En esta primera parte, mediante un algoritmo de planificación de trayectoria se le mandarían las acciones de control a los motores del robot, el cual comenzaría a moverse. Teniendo el modelo del robot, es decir, una serie de ecuaciones que nos relacionen el comportamiento del mismo, podríamos estimar su posición por odometría.

Como vimos en el apartado 4.2 las ecuaciones del filtro de Kalman parten de un modelo con una representación en el espacio de estados. Nuestro estado, al que ahora le llamaremos estado aumentado, contendrá la posición del robot, el ángulo y la posición de cada ceilmark.

- Modelo del vehículo: el estado del vehículo es igual a su posición y su ángulo. Su estado se puede escribir también como función del estado anterior y las entradas.

$$X_v(k) = [x(k), y(k), \varphi(k)]^T ; \quad u(k) = [\omega(k), \gamma(k)]^T$$

Ecuación 4-1: estado y entradas del vehículo

Las entradas son la velocidad angular y la curvatura. La configuración móvil elegida es triciclo debido principalmente a su simplicidad, tanto de construcción como de modelado. En contrapartida tenemos una mayor inestabilidad<sup>14</sup>.

- Modelo del ceilmark: el punto de referencia se caracteriza por su posición. Los landmarks se consideran estacionarios, por lo que el modelo es constante:

$$ci = [xi, yi]^T$$

Ecuación 4-2: posición del landmark

- Modelo de estado aumentado: como hemos mencionado se compone de los dos anteriores.

$$X(K) = \begin{bmatrix} X_v(k) \\ c1 \\ c2 \\ c3 \\ c4 \end{bmatrix} = \begin{bmatrix} F(X_v(k-1), u(k)) \\ c1 \\ c2 \\ c3 \\ c4 \end{bmatrix} + \begin{bmatrix} q_v(k) \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Ecuación 4-3: Modelo aumentado simplificado

$$X(K) = \begin{bmatrix} x_v(k) \\ y_v(k) \\ \theta(k) \\ x_{c1} \\ y_{c1} \\ x_{c2} \\ y_{c2} \\ x_{c3} \\ y_{c3} \\ x_{c4} \\ y_{c4} \end{bmatrix} = \begin{bmatrix} x_{v(k-1)} - Ts * vk * \sin(\theta_{k-1}) \\ y_{v(k-1)} + Ts * vk * \cos(\theta_{k-1}) \\ \theta_{k-1} + Ts * vk * \tan(ak)/b \\ x_{c1} \\ y_{c1} \\ x_{c2} \\ y_{c2} \\ x_{c3} \\ y_{c3} \\ x_{c4} \\ y_{c4} \end{bmatrix} + \begin{bmatrix} wx(k) \\ wy(k) \\ w\theta(k) \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Ecuación 4-4: Modelo aumentado completo

Siendo:

---

<sup>14</sup> Mayor inestabilidad que otros modelos como el Ackerman o los Diferenciales.

- $T_s$  : periodo de muestreo.
- $V_k$  : velocidad lineal de la plataforma.
- $x_v, y_v$  : coordenadas del robot.
- $\theta$  : ángulo de giro del móvil respecto la vertical.
- $xci, yci$  : coordenadas del ceilmark i.
- $ak$ : ángulo de la rueda respecto de la vertical
- $b$ : distancia entre las ruedas delanteras y traseras

Los valores de los parámetros se justificarán posteriormente en nuestra implementación. Como vemos en la ecuación 4-4, el estado es no lineal y por ello debemos aplicar el Filtro de Kalman Extendido como explicamos anteriormente. El resultado de esta primera parte sería la posición estimada del robot.

#### 4.4.2 El robot mide, modelo de observación

El robot se ha movido y hemos estimado su posición por odometría. Ahora cabe estimar cuales serían las posiciones que deberían tener los landmarks (modelo de observación); y compararlo con las medidas reales que obtengo del entorno.

- Modelo de observación: distancia y ángulo relativos al robot móvil.

$$z_i(k) = \begin{bmatrix} z_r^i(k) \\ z_\phi^i(k) \end{bmatrix} = \begin{bmatrix} \sqrt{(xi - x(k))^2 + (yi - y(k))^2} \\ \arctan\left(\frac{yi - y(k)}{xi - x(k)}\right) - \theta(k) \end{bmatrix} + \begin{bmatrix} r_r^i(k) \\ r_\phi^i(k) \end{bmatrix}$$

Ecuación 4-5: Modelo observación SLAM

Siendo:

- $z_r^i(k)$ : distancia del robot al ceilmark iésimo
- $z_\phi^i(k)$ : ángulo entre el robot y el ceilmark iésimo
- $xi, yi$ : estado del ceilmark iésimo
- $x(k), y(k)$ : estado del robot móvil
- $\theta(k)$ : ángulo del robot
- $r_r^i(k), r_\phi^i(k)$  : varianza del error cometido en radio y ángulo
- Matriz de covarianza: como vimos en el las ecuaciones del KF y EKF la matriz de covarianza es clave en el proceso, nos da una idea de las incertidumbres del sistema. En concreto nuestra matriz en el SLAM contiene la covarianza de posición del robot, la covarianza entre la posición del robot y los landmarks y la covarianza entre los landmarks.

$$P = \begin{bmatrix} P_{x,y,\theta} & \cdots & P_{mi} \\ \vdots & \ddots & \vdots \\ P_{im} & \cdots & P_{mm} \end{bmatrix}$$

Ecuación 4-6: Ecuación compacta de la matriz de covarianza

La importancia de la matriz de covarianza reside en que nos da una idea de convergencia del método y la incertidumbre asociada. En concreto, el determinante de la matriz de covarianzas nos da una medida de la incertidumbre global, el cual mediría el volumen del elipsoide que podría representar esta incertidumbre.

Según el teorema demostrado en [2]:

*El determinante de la matriz de covarianza de los objetos del mapa, excluido el robot (y cualquier submatriz principal, por ejemplo la varianza de cada objeto del mapa), es una función monótona no creciente del tiempo (decrece a medida que se incorporan observaciones al filtro).*

En el apartado de nuestra implementación del SLAM analizaremos como han sido los resultados de la incertidumbre y convergencia en nuestro caso.

Para no repetirnos, el proceso a seguir para el filtrado sería el mismo que en el EKF, cambiando las matrices por las respectivas tratadas en este apartado. Al final del apartado 4.4 veremos una figura en la que se resumen las ecuaciones que utiliza el SLAM.

#### 4.4.3 Nuevas observaciones y reobservaciones

Una vez que el robot ha estimado el nuevo estado y su incertidumbre el proceso vuelve a iterarse, teniendo en cuenta dos factores; la observación de nuevos ceilmarks y la asociación de datos de ceilmarks ya vistos.

- Adición de nuevos ceilmarks: el método tiene que ser lo bastante flexible como para ir añadiendo o quitando los estados necesarios en cada momento. A cada paso se calcularían las nuevas matrices y sus respectivos jacobianos.
- Asociación de datos: la formulación estándar del EKF-SLAM es especialmente sensible ante asociaciones incorrectas de marcas, como corroboraremos posteriormente. El problema en el que el robot vuelve a observar una marca tras un largo recorrido es especialmente difícil.

#### 4.4.4 No linealidades

El EKF-SLAM linealiza modelos no lineales de movimiento y observación, con lo que hereda varias limitaciones. En el caso no lineal el efecto más inmediato es que no nos aseguramos una solución óptima. Nuestro modelo presenta no linealidades en puntos específicos y en determinadas circunstancias, presentan un problema que puede llevar a la divergencia del método.

Para concluir, en la siguiente figura mostramos a modo de resumen una figura con todas las ecuaciones del algoritmo:

- **Predicción:** En este caso, predecimos el nuevo estado en base a la ecuación no lineal que modela el movimiento del vehículo

$$\hat{\mathbf{x}}(k+1|k) = \mathbf{f}(\hat{\mathbf{x}}(k|k), \mathbf{u}(k)) \quad (2.19)$$

$$\mathbf{P}(k+1|k) = \mathbf{F}(k)\mathbf{P}(k|k)\mathbf{F}^T(k) + \mathbf{Q}(k) \quad (2.20)$$

- **Observación:**

$$\hat{\mathbf{z}}_i(k+1|k) = \mathbf{h}_i(\hat{\mathbf{x}}(k+1|k), \mathbf{p}_i) \quad (2.21)$$

$$v_i(k+1) = \mathbf{z}_i(k+1) - \hat{\mathbf{z}}_i(k+1|k) \quad (2.22)$$

$$\mathbf{S}(k+1) = \mathbf{H}(k)\mathbf{P}(k+1|k)\mathbf{H}^T(k) + \mathbf{R}(k) \quad (2.23)$$

- **Actualización:**

$$\mathbf{K}_i(k+1) = \mathbf{P}(k+1|k)\mathbf{H}_i^T(k)\mathbf{S}_i^{-1}(k+1) \quad (2.24)$$

$$\hat{\mathbf{x}}(k+1|k+1) = \hat{\mathbf{x}}(k+1|k) + \mathbf{K}_i(k+1)v_i(k+1) \quad (2.25)$$

$$\mathbf{P}(k+1|k+1) = \mathbf{P}(k+1|k) - \mathbf{K}_i(k+1)\mathbf{S}_i(k+1)\mathbf{K}_i^T(k+1) \quad (2.26)$$

Figura 4-4: Resumen ecuaciones SLAM

## 4.5 Implementando el SLAM

En este apartado abordaremos los distintos métodos que barajamos a la hora de aplicar nuestro SLAM, posteriormente hablaremos de una simulación que realizamos a modo de paso previo a la implementación final.

### 4.5.1 Métodos

Dado que nuestro proyecto tiene una gran parte de visión por computador, no queríamos extendernos por exceso en este parte, pese a saber de antemano que sería las más difícil conceptualmente hablando. Las opciones que barajamos fueron las siguientes:

- Librerías: en un primer momento pensamos que podría suponer un gran ahorro de tiempo utilizar una librería que incorporara el algoritmo y las funciones de representación de los errores y los mapas. Tras analizar varias librerías, nos centramos en la librería de Joan Solà [8]. Ya que era la que más documentación nos aportaba. Finalmente nos dimos cuenta que poner en marcha algunos ejemplos no era difícil pero entender completamente todas las funciones de la librería para adaptarlo a nuestro método era una tarea algo compleja.
- Métodos propios: al ver que no avanzamos lo necesario, decidimos afrontar el problema por nuestra cuenta, y desarrollar nuestro propio SLAM paso a paso. Pese a saber que no sería tan eficaz como los ya implementados, nos permitiría adaptar como quisieramos la solución a nuestro problema y sobre todo entender cada línea de código escrita.

### 4.5.2 Simulaciones

Tuve la suerte de que un compañero de industriales, estaba dando justamente una asignatura en la que estaba tratando el SLAM, por lo que le pedí toda la documentación posible y las prácticas que fuera a hacer. Encontré una práctica en la que se pedía lo siguiente:

*“El objetivo es realizar un programa en Matlab que implemente el SLAM basado en filtro de Kalman Extendido (EKF) para estimar el estado de un robot móvil, y analizar su comportamiento en función de sus parámetros.”*

El robot móvil en cuestión era un triciclo, nos facilitaban los parámetros y sus errores (Anexo N: Parámetros y errores triciclo). El triciclo realizaba una trayectoria circular. En primer lugar analizaremos la localización obtenida únicamente a través de la odometría, posteriormente a través del SLAM con un landmark y posteriormente con tres landmarks.

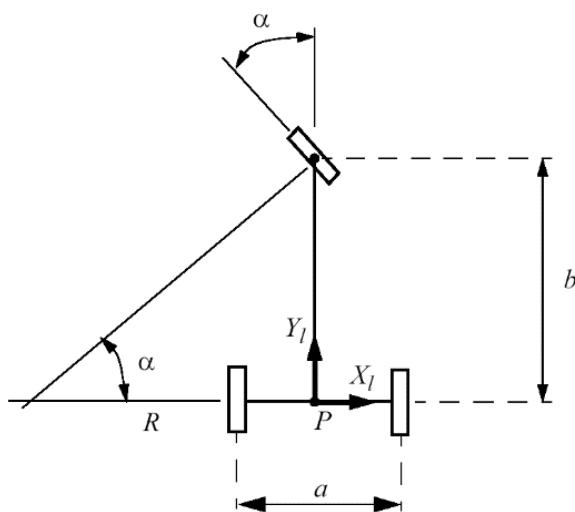


Figura 4-5: Esquema del triciclo

Odometría: En primer lugar se pedía cual sería la trayectoria del robot si solo se dispone de la odometría. Como

observamos en la siguiente figura, el error cometido va incrementándose a lo largo de la trayectoria como cabe esperar. En el punto final hay un error de unos 2.5 metros, inaceptable para localizar un robot en interiores.

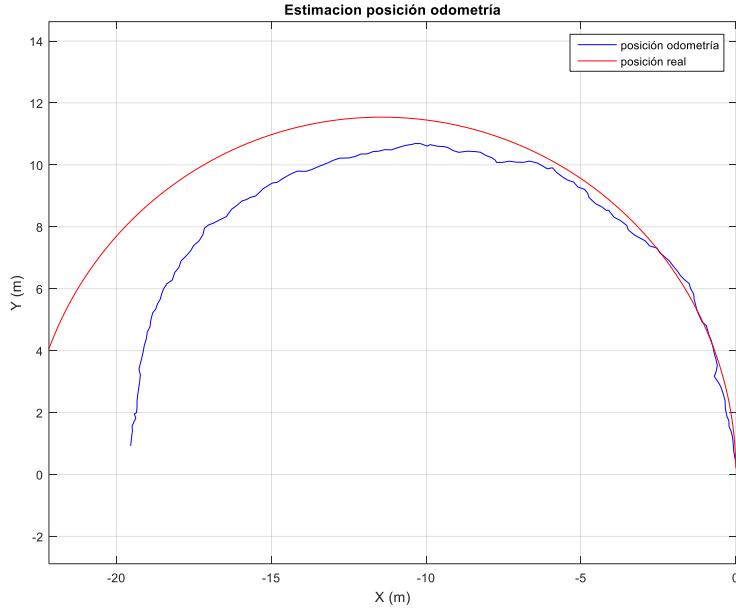


Figura 4-6: Estimación posición por odometría

SLAM 1 Landmark: En segundo lugar se pedía realizar el SLAM con un landmark, del cual suponíamos que conocíamos su posición en todo momento pero con errores. Debido a que teníamos el modelo decidimos en una primera instancia particularizar el SLAM para este triciclo con estas condiciones. Calculamos los jacobianos y aplicamos las ecuaciones de la Figura 4-4. Los resultados obtenidos son los siguientes:

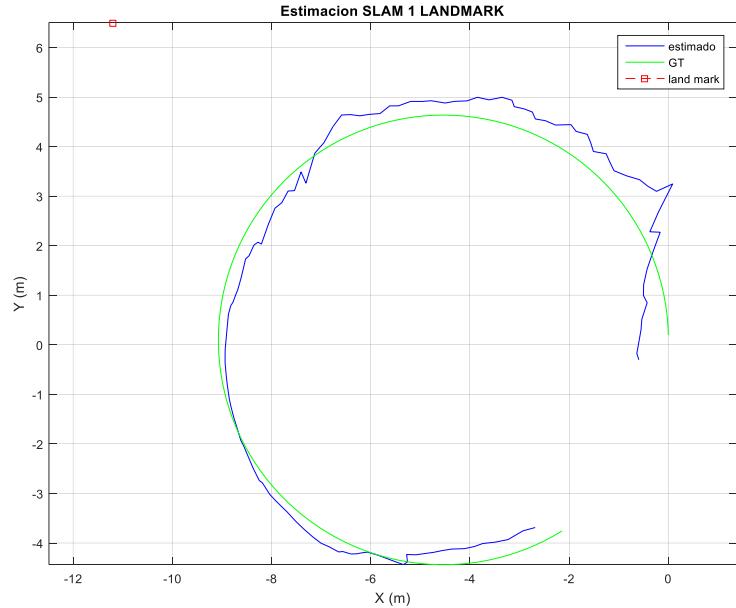


Figura 4-7: SLAM con un landmark

Vemos como al añadir un landmark la posición estimada empieza a converger con la real. Los errores son mucho menores como analizaremos a continuación.

SLAM 3 landmarks: Por último vemos los efectos de incluir más landmarks en diferentes posiciones. Cabe

mencionar que la posición de los landmarks afecta ligeramente a las no linealidades del sistema, siendo deseable no tener landmarks en puntos como el origen de coordenadas o puntos por los que pasa el móvil.

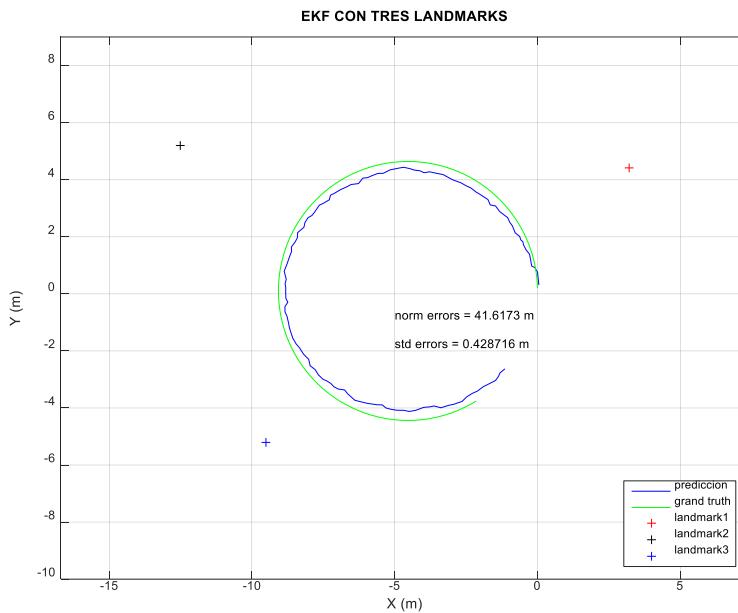


Figura 4-8: SLAM con tres landmarks

Observamos como las estimaciones son mucho mejores, tenemos ya un error aceptable para una localización de interiores. La posición de los landmarks tiene importancia, al colocarlos repartidos en la trayectoria cada uno puede aportar una información en cada tramo de trayectoria del robot. No interesa alejarlos demasiado ni acercarlos tanto como para que su información sea redundante. Como ya mencionamos hay que tener en cuenta también las no linealidades del sistema.

#### 4.5.3 Implementación

Dado que entender las librerías anteriormente mencionadas y adaptarlas a nuestro entorno iba a ser una tarea difícil, decidimos servirnos del ejemplo simulado e intentar avanzar en esa dirección. Había un problema fundamental; había que construir el robot, (o por lo menos la parte móvil con la cámara), para poder obtener las señales de actuación. Además habría que elegir una planificación de trayectoria adecuada y un seguimiento asociado que no tuviera errores de más de cinco grados.

Para eliminar estos problemas lo que hicimos fue: en vez de adaptar la simulación que teníamos al mundo real, adaptamos el mundo real a la simulación que teníamos. Es decir:

1. Pintamos la trayectoria circular que seguía el triciclo en el suelo, adaptando los parámetros (distancias y velocidades) del mismo para que fuera coherente con el habitáculo en cuestión.
2. Establecimos una serie de puntos equidistantes en los que el robot tomaría la foto. Calculamos las coordenadas de cada punto y las marcamos en el suelo. En concreto decidimos tomar 15 fotografías, es decir, una fotografía cada 24 grados. Ver Figura 4-10: Proceso de toma de fotografías.
3. Hicimos una plataforma para la cámara y el láser (fijando el láser para que se mantuviera encendido) y fuimos tomando fotografías punto por punto.
4. Adaptamos el SLAM para los cuatro landmarks, y lo ajustamos para que cogiera datos de las fotografías justamente en las posiciones en las que correspondía, es decir, cada 24 grados.
5. Realizamos el SLAM completo ajustando posteriormente los parámetros en un bucle y elegimos aquellos que minimizaran errores.

Este proceso nos permitía probar el algoritmo sin tener que construir físicamente el robot, lo que nos ahorró mucho tiempo y esfuerzo. Además la construcción de un robot móvil no era el objeto de este presente proyecto.

A continuación se muestra un esquema de cómo sería el proceso completo, desde que el robot<sup>15</sup> se mueve hasta que completa un bucle del SLAM:

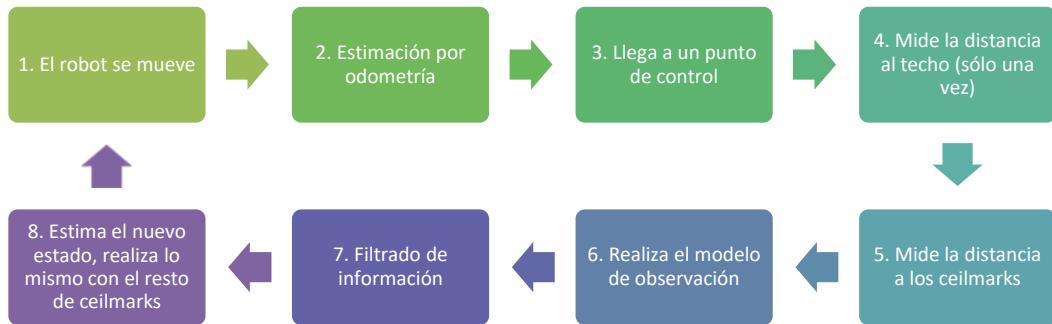


Figura 4-9: Esquema proceso SLAM

En la siguiente fotografía vemos el proceso de toma de fotografías:

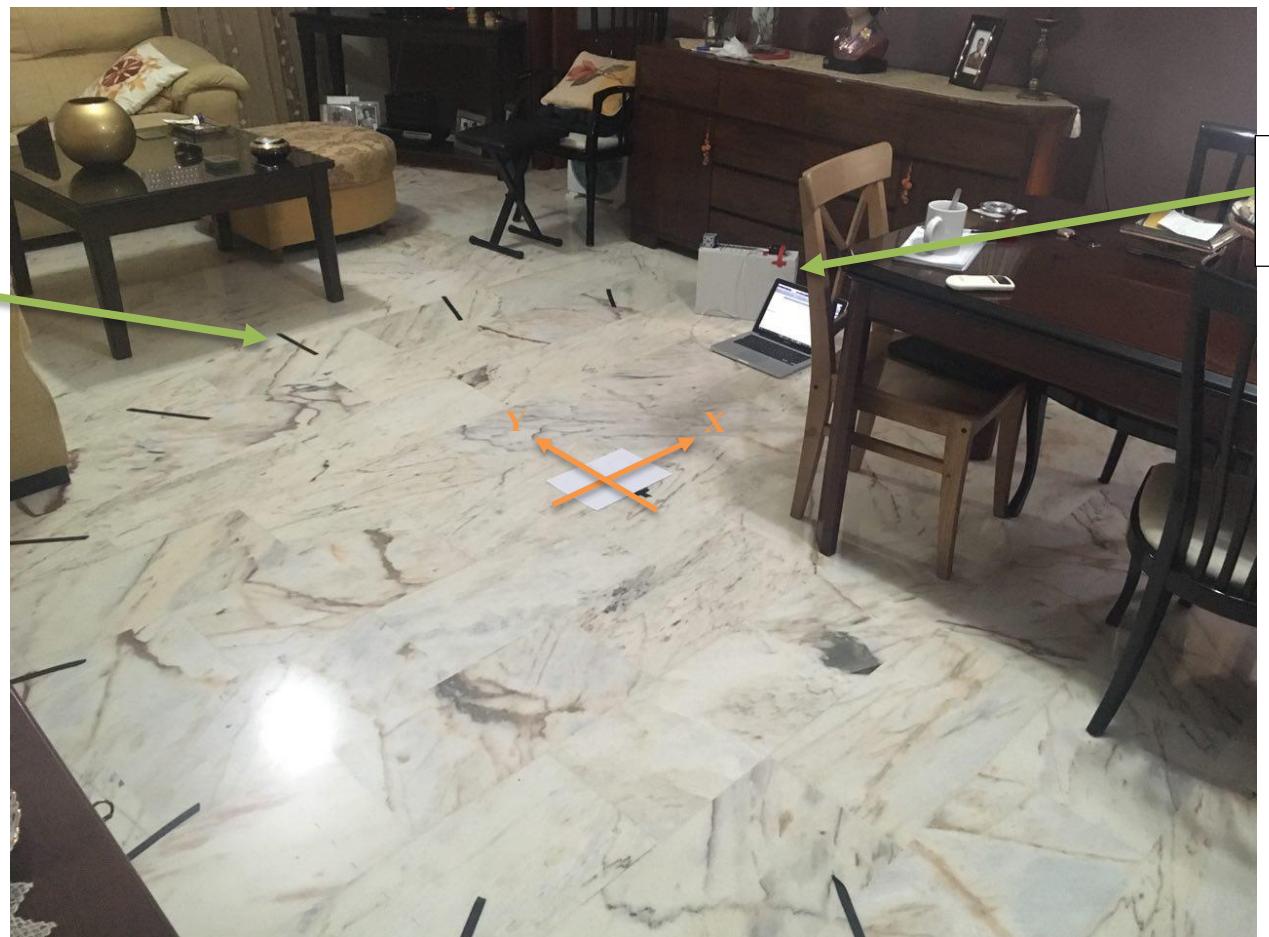


Figura 4-10: Proceso de toma de fotografías

<sup>15</sup> Nos referimos al robot aunque como sabemos en la práctica no se ha construido un robot físico, simplemente la plataforma donde va alojada la cámara y el láser. Toda la inteligencia va implementada en el ordenador y el SLAM es por lo tanto offline como dijimos en la introducción.

Por lo tanto lo único que nos quedaba era adaptar el código que teníamos de base a esta implementación. Tuvimos que solucionar algunos problemas que tuvimos con los ejes, ya que el sentido de giro de ejes de la cámara era en sentido negativo al seguido por el robot y los ejes del mundo real estaban cambiados respecto al de la cámara. En la siguiente figura se muestra cuáles son los ejes de la cámara respecto a los reales.

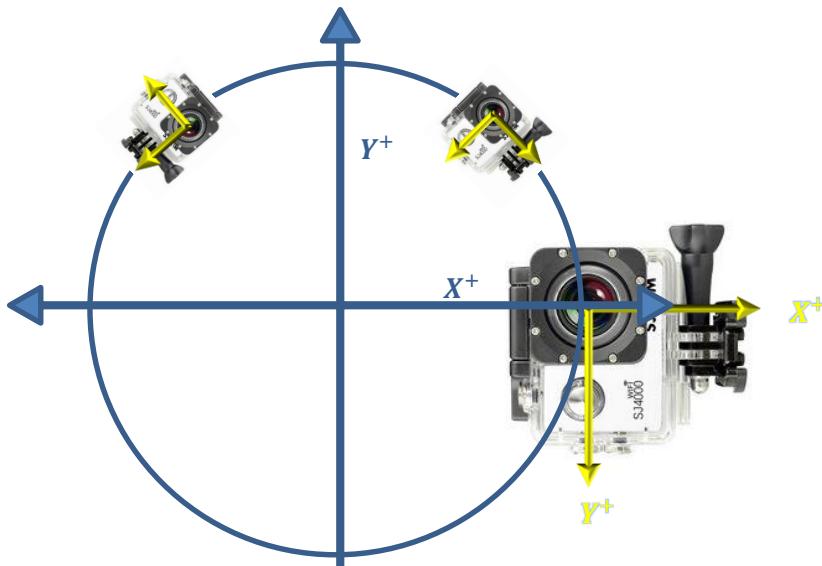


Figura 4-11: Ejes de la cámara y el mundo real

Iremos analizando y comentando las partes más importantes del código, omitiendo los procesos ya explicados en los hitos anteriores como la medición del techo y la extracción de características. Para estas partes omitidas se puede consultar el Anexo O: Código completo SLAM. Donde viene el código completo.

- Definición de parámetros:

```
close all;clc; clear;
nlandmark=4; % numero de landmarks
n=3+2*nlandmark; % orden del sistema
m=2; % orden de las medidas
vk=0.3; % velocidad (antendiendo a especif)
Ts=0.2093; % tiempo de muestreo Ts=s/vk=0.628/0.3
ak=deg2rad(15); % angulo de giro
b=0.4; % distancia entre ruedas
wx=0.05; % desviación posición x
wy=0.05; % desviación posición y
wp=deg2rad(1); % desviación ángulo de giro
red=0.4; % factor auxiliar
```

- Carga de los parámetros óptimos obtenidos; Q, R y P.

- Modelo del sistema y modelo de medición:

```
% MODELO DEL SISTEMA CON TODAS LAS VARIABLES POSIBLES
f=@(x) [x(1)-Ts*vk*sin(x(3)); % x
          x(2)+Ts*vk*cos(x(3)); % y
          x(3)+Ts*vk*tan(ak)/b; % angulo
          x(4); % 11
          x(5); % 12
          x(6); % 13
          x(7);
          x(8);
          x(9);
          x(10); % 14
```

```

x(11)] ;

% MODELO DE MEDIDA (DISTANCIA Y ANGULO AL LANDMARK)
p=1;
h=@(x) [sqrt((x(2+2*p)-x(1))^2+(x(3+2*p)-x(2))^2);
atan((x(3+2*p)-x(2))/(x(2+2*p)-x(1))+x(3))];

```

4. Carga de las fotografías y obtención de las coordenadas de los ceilmarks. Modelado inverso para la obtención de las medidas reales.
5. Comenzamos el bucle. Como N=150 y se toman 15 fotografías, se toma una fotografía cada diez pasos. Si coincide el paso actual con un marcador habilitamos la variable condición a uno. El vector vlandmark se ocupa de indicar que landmark tenemos que actualizar.<sup>16</sup>

```

For k=1:N
%% TOMAMOS LA IMAGEN Y EXTRAEMOS EL ROOFMARK
% si estamos en el punto correcto
if (k-1==cont*10), cont=cont+1; condicion=1;
else 65ptico65ón=0; end;
vlandmark=[0 0 0 0];

```

6. Si se cumple dicha condición, obtener las coordenadas del landmark, calcular el radio desde el centro de la cámara y el ángulo. Estas medidas son relativas al móvil, por lo que hay que pasarlas a absolutas. Recordamos que los ejes están cambiados en el eje Y y el ángulo de la foto va al contrario que el del robot, por lo que hay que desgirar el ángulo del robot.

```

If (65ptico65ón)
Prm=MF(cont, ⊕); % cogemos el punto de la matriz
if (Prm(1) ~= 0 && Prm(2) ~= 0) % si el punto es válido
    a=a+1;
    px=(Prm(1));py=(Prm(2));

    % Calculamos el punto en relativo
    r = norm([px,py]);
    theta = atan2(py,px);
    % desgirar t(3)
    alfa=-x(3); % pi-x(3)
    % Matriz de giro
    M=[cos(alfa) -sin(alfa); sin(alfa) cos(alfa)];
    P2=M*[px;-py]; % punto girado
    xr=P2(1)+x(1); % punto en coordenadas reales
    yr=-P2(2)+x(2);

```

7. En nuestro caso el ceilmark, al encontrarse uno en cada cuadrante basta con calcular en qué cuadrante se encuentra (en coordenadas absolutas) y obtendríamos que ceilmark estamos viendo. Pondríamos el valor del landmark correspondiente a uno.

```

% Calculamos el cuadrante
l = calcucuadrante(xr,yr);
% 65pti ya sabemos a que landmark se corresponde
vlandmark(l)=1; % lo ponemos a uno
disp('Nlandmark = '); disp(l);

```

---

<sup>16</sup> Esto no es muy eficiente si tenemos muchos landmarks, ya que estamos utilizando un vector de la misma dimensión, tengamos que actualizar todos los landmarks o sólo uno. Esto se debe a que la estructura de la matriz de estados es también fija. La librería anteriormente mencionada incorporaba una artimaña que le permitía trabajar sólo con lo necesario.

8. Para cada ceilmark estimamos la función de observación, en el caso de no haber landmarks mantenemos la misma posición añadiendo el ruido. Como ya hablamos usamos un algoritmo más avanzado, el ekf o ukf. La ventaja del ukf es que permite pasarle una función dentro de otra, lo que permite utilizar la atan2 y evitarnos algunos problemas<sup>17</sup>.

```

For p=1:nlandmark
    if(useukf==0) % este si empleamos ekf
        h=@(x) [sqrt((x(2+2*p)-x(1))^2+(x(3+2*p)-x(2))^2);
                  (atan((x(3+2*p)-x(2))/(x(2+2*p)-x(1)))+x(3))];
    else
        h=@(x) [sqrt((x(2+2*p)-x(1))^2+(x(3+2*p)-x(2))^2);
                  (atan2((x(3+2*p)-x(2)),(x(2+2*p)-x(1)))+x(3))];
    end

    if (vlandmark(p)) % si corresponde ese landmark
        % calculamos la mededida
        z(1,1) = r;
        z(2,1) = pi-theta;
    else
        z = h(s) + R*randn(m,1); %measurments
    end

```

9. Aplicamos el filtro de kalman correspondiente:

```

if (useukf==0)
    [x, P] = ekf(f,x,P,h,z,Q,R); % ekf
else
    [x, P] = ukf(f,x,P,h,z,Q,R); % ukf
end
end

```

10. Finalmente guardamos y pintamos las variables que nos interesen. Ver Anexo O: Código completo SLAM.

---

<sup>17</sup> La función atan2 devuelve la arctangente pero con la ventaja de que nos evitamos la dualidad posible de la arctangente, ya que verifica en qué cuadrante debería estar el ángulo y elige el adecuado.

#### 4.5.4 Resultados

Inicialmente los ceilmarks están en una posición aleatoria definida por el usuario, la incertidumbre es grande (figura 4-11). A medida que avanza el robot va tomando fotografías y va estimando el mapa y su localización satisfactoriamente, hasta que finalmente completa la vuelta (figura 4-12).

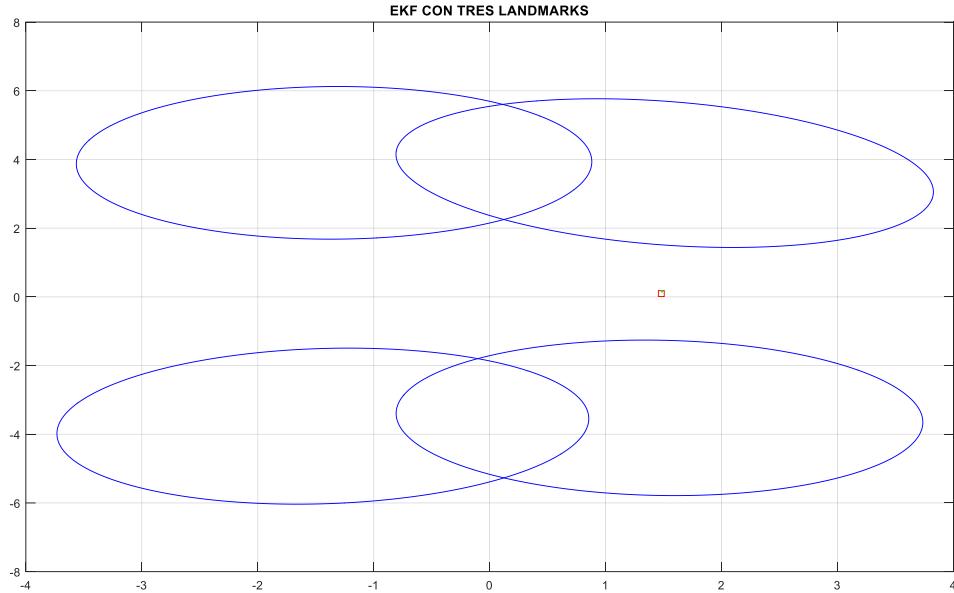


Figura 4-12: Estimación inicial SLAM

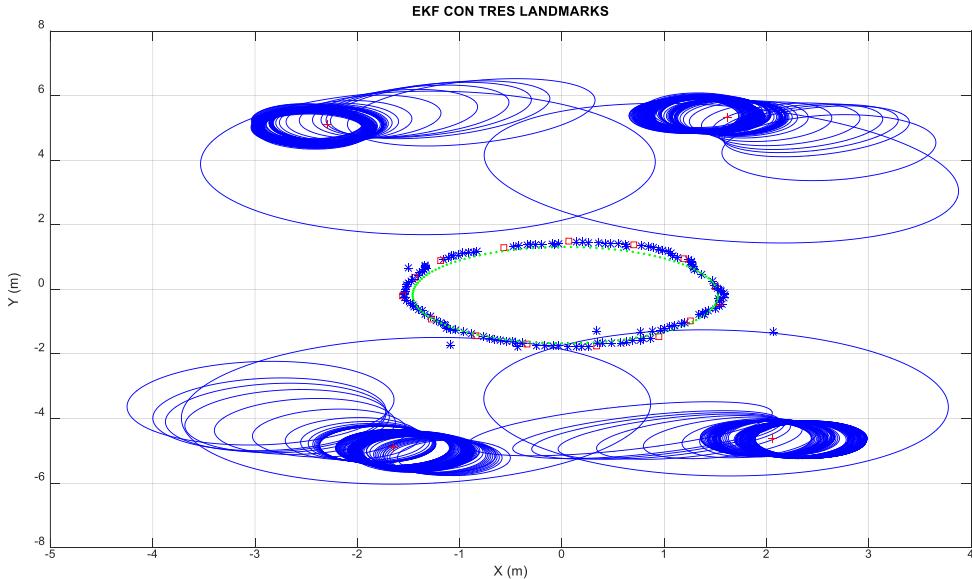


Figura 4-13: Estimación final SLAM

Siendo:

- Elipses azules: covarianza de los ceilmarks (ancho de la gaussiana).
- Cruces rojas: posición de los ceilmarks (media de la gaussiana).
- Cuadrados rojos: zonas de toma de fotografías.
- Recorrido azul: trayectoria estimada del robot.
- Recorrido verde: trayectoria real.

Vemos como a medida que el robot se desplaza, la incertidumbre de los ceilmarks se reduce, además se va ajustando la posición de los mismos. Vemos que el algoritmo converge y luego veremos más cuantitativamente como de buenos son estos resultados.

Cabe recordar que el algoritmo converge a la solución óptima en media no como valor constante. A continuación mostramos como son las matrices de covarianza tanto de los ceilmarks como de la posición del vehículo.

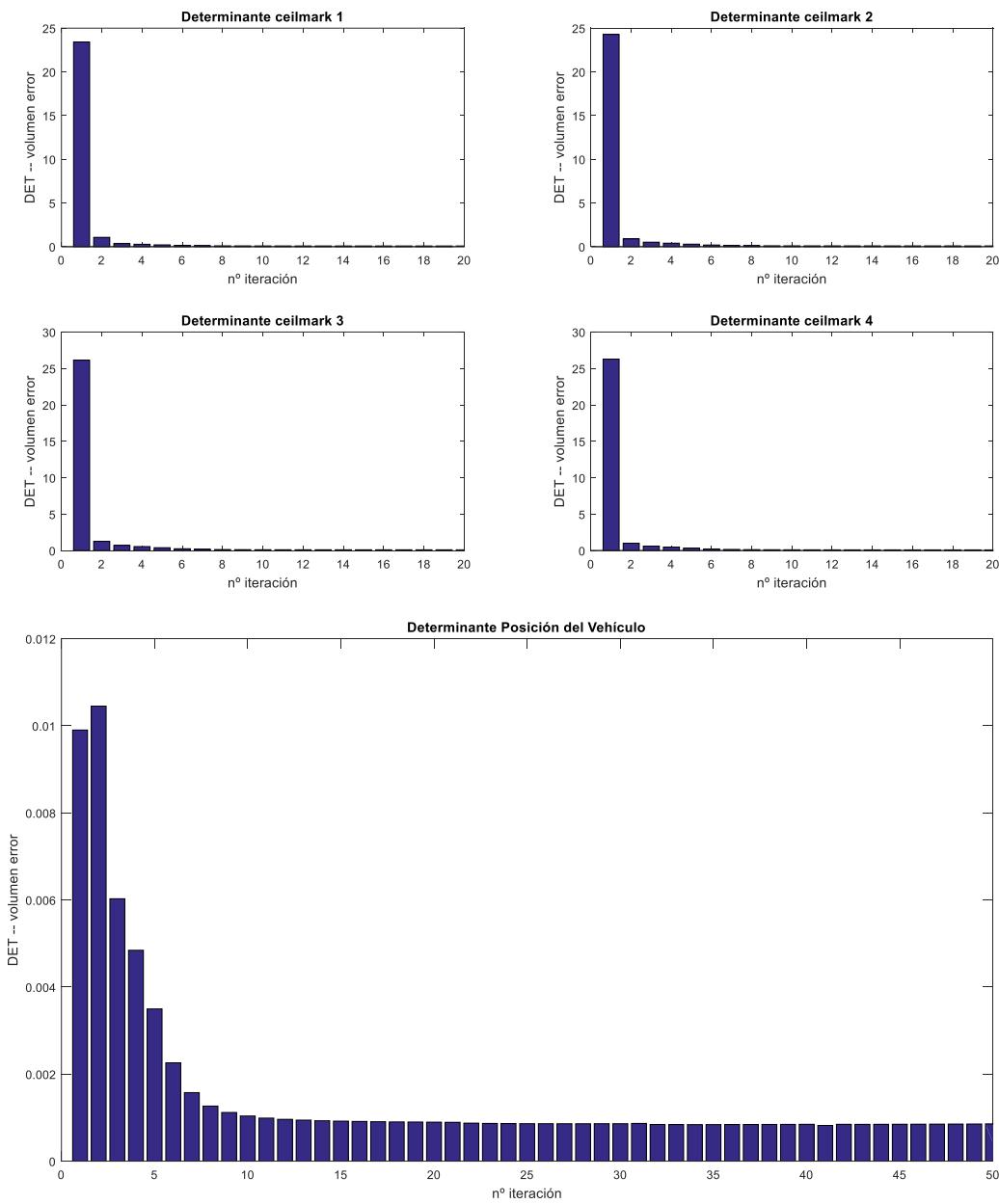


Figura 4-14: Determinantes matrices covarianzas (P)

Vemos como el determinante converge satisfactoriamente hacia la mínima varianza posible; para el vehículo, los errores propios de posición y ángulo y los ceilmarks sus errores de posición.

Para ver la robustez del algoritmo hemos introducido una imagen errónea en el proceso. Lo que ocurría es que el robot tomaba una foto debajo de la mesa e identificaba un falso ceilmark, en una posición errónea. Hasta ahora hemos hecho la corrección de esa foto manualmente, añadiendo una excepción a ese paso del filtro. A continuación mostraremos cual sería la estimación si esa fotografía falla.



Figura 4-15: Fotografía 14, falso ceilmark

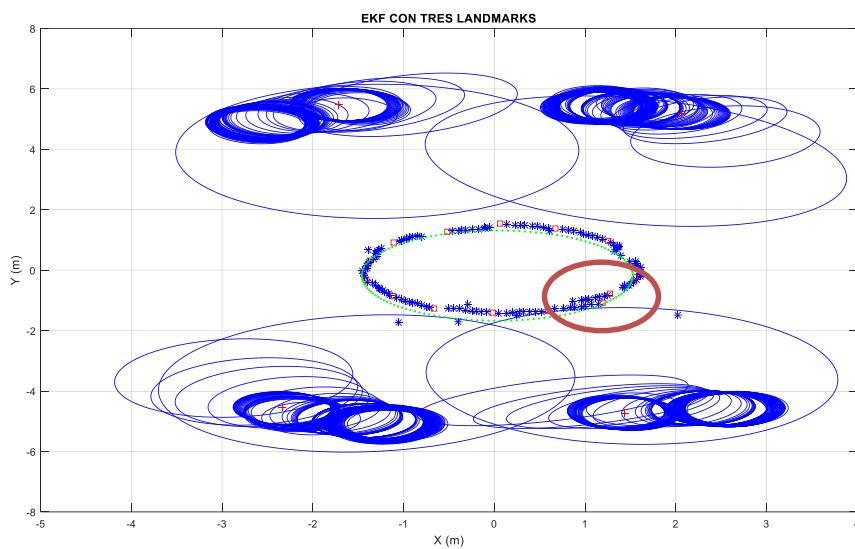


Figura 4-16: Estimación sin corrección

Vemos como efectos significativos un cambio del mapa en el último momento, el filtro estima mal la posición y parece que vuelve sobre el recorrido pisado como hemos señalado en rojo. Comprobamos efectivamente la importancia de una buena asociación de datos y un método con una visión más robusta. Una solución sería utilizar el láser en cada foto, si vemos que disminuye significativamente la altura del techo en alguna foto, ignoraríamos esa medición.

Construcción del mapa: Dado que nuestros puntos de referencia son las esquinas de la habitación, la construcción de las dimensiones del mapa es inmediata. Faltaría añadir como mencionamos más elementos al mapa, para tener una visión de los objetos internos que hay en el habitáculo. Para ello nos podríamos servir de una cámara frontal o un sensor de ultrasonidos por ejemplo.

En la siguiente figura tenemos el mapa construido tras la primera vuelta:

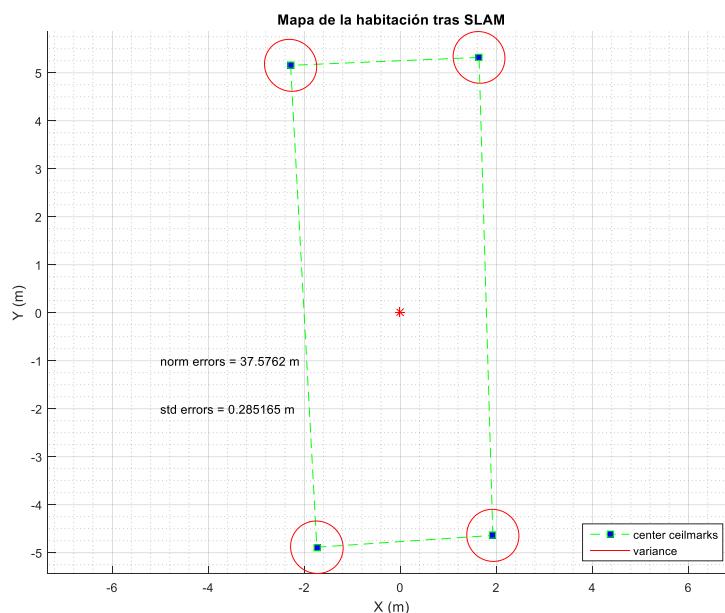


Figura 4-17: Mapa del habitáculo

Como cabía esperar, el mapa forma un rectángulo de  $40\text{ m}^2$  (4 m de ancho y 10 m de largo), tal y como es en la realidad. Observamos que los ángulos no forman  $90^\circ$  exactos, sobre todo el del landmark 4, ya que es del que menos datos tenemos.

Errores: A continuación analizaremos todos los errores cometidos de media en todos los casos (tras realizar la media de 15 experimentos por categoría).

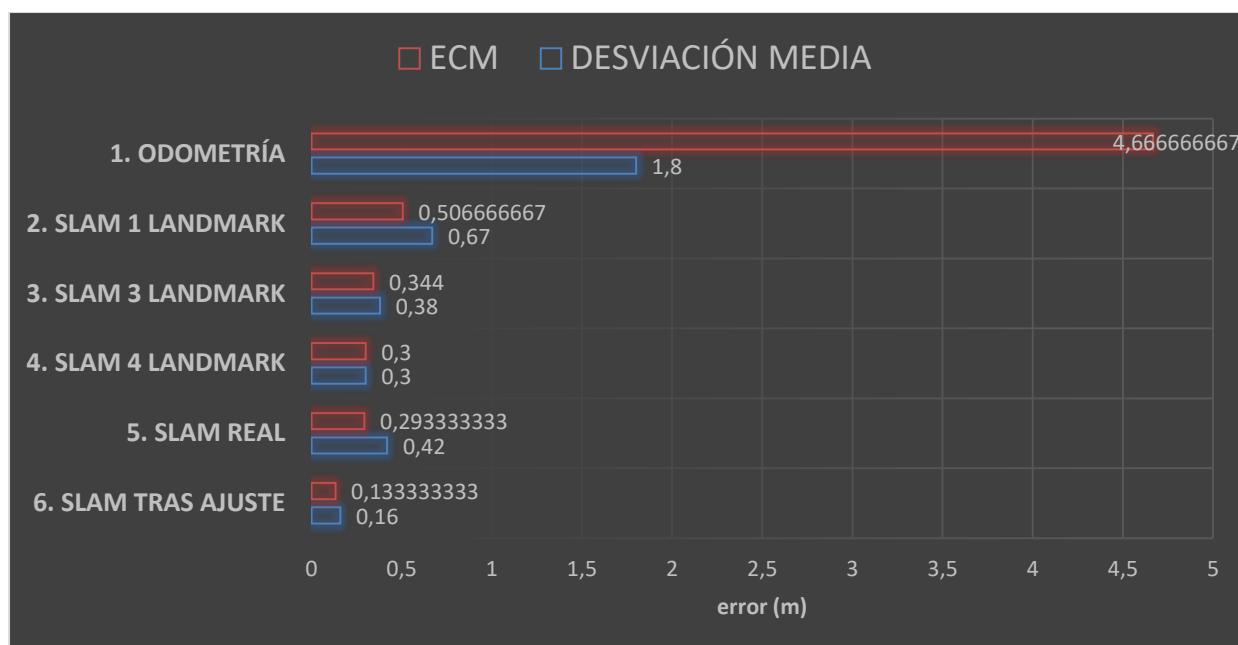


Figura 4-18: Errores cometidos en los diferentes casos

Los peores resultados se obtienen si utilizamos sólo la odometría; tenemos un error cuadrático medio (ECM)<sup>18</sup> de unos 4.6 metros y una desviación de 1,8 metros por lo que el 95% de los resultados están a  $\pm 3,6\text{m}$ <sup>19</sup> lo que resulta inaceptable. Medir utilizando sólo la odometría no es factible.

Dentro de los métodos de SLAM vemos que en las simulaciones (2, 3, 4 en el gráfico 4-18) los resultados mejoran a medida que añadimos landmarks, lo que reafirma el teorema de convergencia del filtro que dijimos anteriormente. Cabe destacar que el sesgo no mejora significativamente, lo que nos refleja que pese a mejorar la varianza el valor medio no mejora al mismo ritmo.

Si ahora pasamos al SLAM real sin ajuste, (5 en el gráfico) vemos una desviación de 0,42 metros, no podemos validarla ya que supondría que habría algunos resultados que estarían a un radio de 80 cm de distancia.

Tras ajustar los parámetros (6 en el gráfico), vemos como mejoran significativamente los resultados; la desviación desciende a 16 cm y el ECM es de 13 cm. Esto nos quiere decir que el 95% de los resultados obtenidos por el filtro se encuentran en un intervalo de 32 cm de confianza. Para hacernos una idea dimensional podemos calcular el área de incertidumbre frente al área del habitáculo.

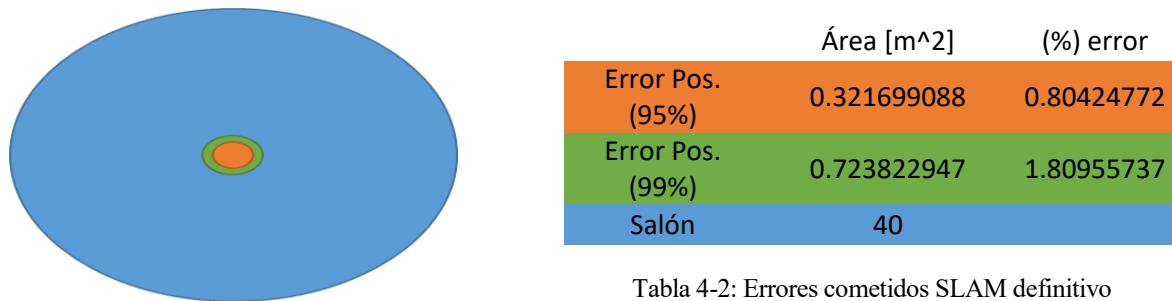


Tabla 4-2: Errores cometidos SLAM definitivo

Figura 4-19: Gráfico superficie errores posición

Vemos errores de posición del orden del 1% respecto a la superficie total, por lo que podemos considerar por válido este último método.

Por último cabe hablar de los errores cometidos en el ángulo, en la mayoría de los casos observamos errores de desviación de  $3,5^\circ$ . Debido a la fuerte curvatura del recorrido podemos dar por aceptables estos errores.

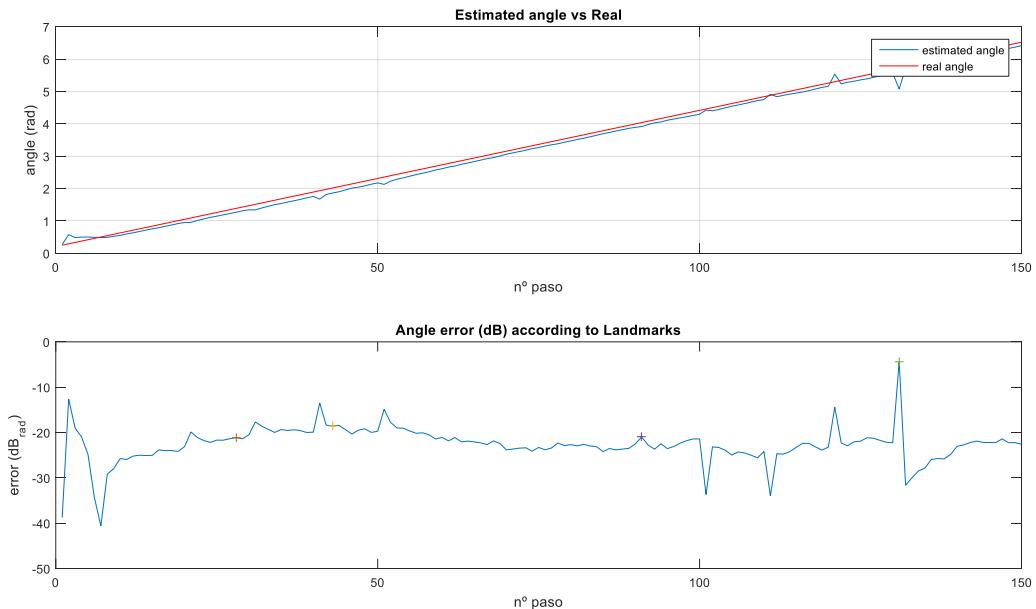


Figura 4-20: Errores cometidos en el ángulo en radianes

<sup>18</sup> En estadística, el error cuadrático medio (ECM) de un estimador mide el promedio de los errores al cuadrado. Es igual a la varianza más el sesgo. Por tanto si vemos en la gráfica un ECM mucho mayor que el cuadrado de las desviación quiere decir que tiene un gran sesgo.

<sup>19</sup> Dado que los experimentos realizados siguen una gaussiana, el intervalo de confianza está relacionado con la desviación a través de la función erf. El 68% de los resultados estarían a un radio de tamaño la desviación, el 95% a un radio de dos veces la desviación y el 99% a tres.

## 4.6 Cobertura, conclusiones y áreas de mejora: HITO C

Una vez encontrado el método último de nuestro SLAM quedaría validado este hito, para finalizar esta sección nos gustaría resumir cual ha sido el porcentaje de cobertura de cada objetivo, para conocer en qué zonas hemos actuado bien y en cuales se podría haber mejorado. Viendo el siguiente gráfico:

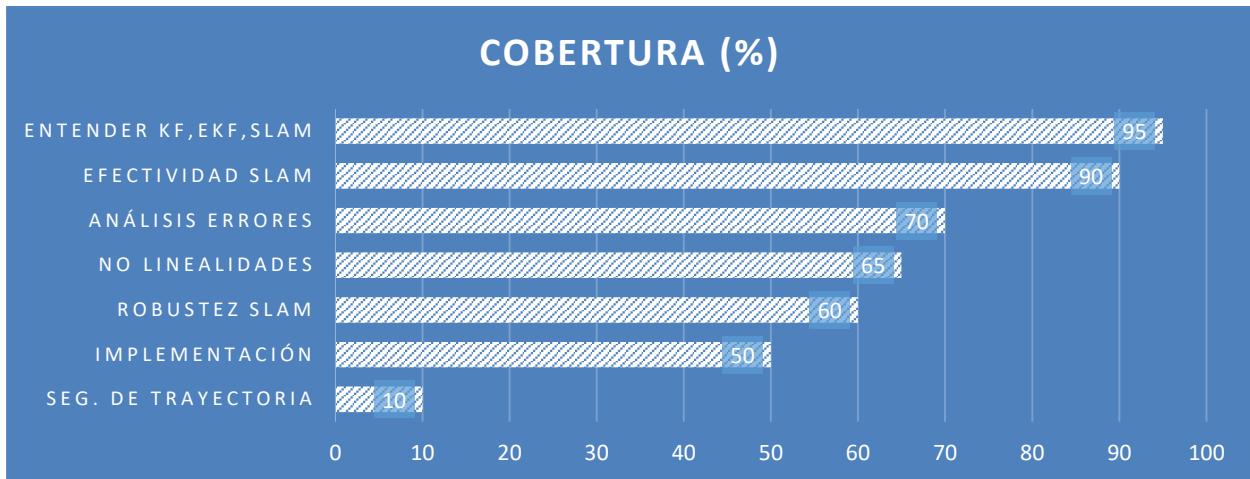


Figura 4-21: Porcentaje cobertura hito C

Vemos como nuestras dos prioridades básicas de este hito se han cumplido satisfactoriamente; entender cómo funciona el SLAM y aplicarlo junto a todo lo anterior de una forma efectiva. Vemos que los objetivos que menos hemos cumplido han sido la implementación, la cual prácticamente eliminamos toda la parte móvil y el seguimiento de trayectoria que estuvimos barajando algún método (como el pure pursuit) pero al no haber implementación real del robot no tenía sentido seguir investigando en esa vía.

Las conclusiones de este hito son las siguientes:

- Pese a que hay métodos más simples como el mapa de ocupación de celdillas, el EKF-SLAM es una herramienta muy potente y muy versátil a la hora de realizar SLAM, está muy extendida en la literatura, hay numerosas mejoras en los algoritmos de computación y se basa en una herramienta fundamental en la ingeniería como es el filtro de Kalman.
- Por un lado, hemos visto como los obstáculos visuales como mesas, muebles pronunciados, etc., pueden llegar a producir falsos puntos de referencia, por lo que habría que implementar soluciones para este tipo de problemas como ya mencionamos. Por otro lado, no hemos tenido en cuenta obstáculos físicos a nivel del suelo, podríamos por ejemplo poner un sensor ultrasonido de corta distancia y hacer una planificación con evitación de obstáculos.
- En definitiva el SLAM con visión utilizando una cámara con gran angular conlleva a un largo camino de procesado informático, métodos un tanto avanzados y una gran robustez (sobre todo a condiciones lumínicas), pero permitiendo una gran flexibilidad y asociación de lo que estamos midiendo con la realidad, como no ocurre con los sensores láser por ejemplo. Es por ello que en una aplicación real, nos serviríamos de al menos una cámara más (infrarroja por ejemplo) que, mediante fusión sensorial, ayuden a una mayor robustez y efectividad del filtro.

Las áreas de mejoras más importantes las podríamos resumir en las siguientes:

- El algoritmo funciona y los resultados son satisfactorios pero hubiera sido muy interesante obtener otros puntos de referencia para mejorar el filtro, por ejemplo, marcos de ventanas o puertas o lámparas.
- Trabajar con toda las matrices completas aunque sólo hayamos detectado un ceilmark es ineficiente, aunque adaptar las matrices al contexto actual requiere cierta complejidad en el algoritmo.
- Podríamos haber separado en dos procesos la representación gráfica en tiempo real y el bucle del filtro, dado que nuestros tiempos entre fotografías (unos dos segundos) y entre pasos (20 milisegundos) son muy lentos, no hemos tenido problemas.

# 5 LÍNEAS FUTURAS

En este apartado nos gustaría comentar cuales serían los desarrollos posibles posteriores a este proyecto hasta llegar a una solución para una situación real como las que planteamos al principio, como el robot clínico.

1. Importación de código: Uno de los primeros pasos sería la depuración de código y la exportación a otros lenguajes como C/C++ y openCV, para poder utilizarlo tanto en Windows como Linux bajo distintas plataformas. Para ello la mayoría de las funciones se podrían traducir casi literalmente, para algunas funciones complejas nos podríamos servir de funciones ya traducidas o podríamos ir traduciendo función a función.
2. Implementación física: A estos nos referimos con la construcción del robot triciclo. Podríamos dotar al robot de un acelerómetro de bajo coste y realizar una realimentación con un microcontrolador PIC<sup>20</sup> o un arduino, de forma que los errores de posición y ángulo tiendan a cero.
3. Hardware: Una vez tenemos el código traducido podríamos exportarlo a un procesador, PC, FPGA<sup>21</sup> o DSP. Si quisierámos hacer el SLAM online, deberíamos prestar atención a la interfaz de captura de imágenes y ajustar bien los tiempos de procesado y adquisición para no causar inestabilidades en el SLAM. Proponemos como opción más inmediata e intuitiva realizar la traducción a OpenCV y utilizar una raspberry que sea como el centro neurálgico tanto de entradas como salidas.
4. Fusión: Como ya hemos comentado, podríamos emplear la fusión sensorial para tener un robot más preciso y robusto. Podríamos incorporar ultrasonidos, ópticos, láser u otras cámaras. El procesador central, en el caso propuesto la raspberry sería la que se ocuparía de la adquisición y tratamiento de toda esta información.
5. Trayectoria: La trayectoria dependerá del comportamiento que le queremos dar al robot; si por ejemplo lo que deseamos es una exploración, la trayectoria y el algoritmo de seguimiento asociado serían, por ejemplo, bordear los límites hasta hacer el mapa y luego explorar la parte central. Si el comportamiento esperado es el desplazamiento de un punto a otro el SLAM se intentaría limitar a esa trayectoria.

Por último a modo de resumen de cómo sería este proyecto completo se muestra un esquema donde se expone el flujo completo de entradas y salidas:

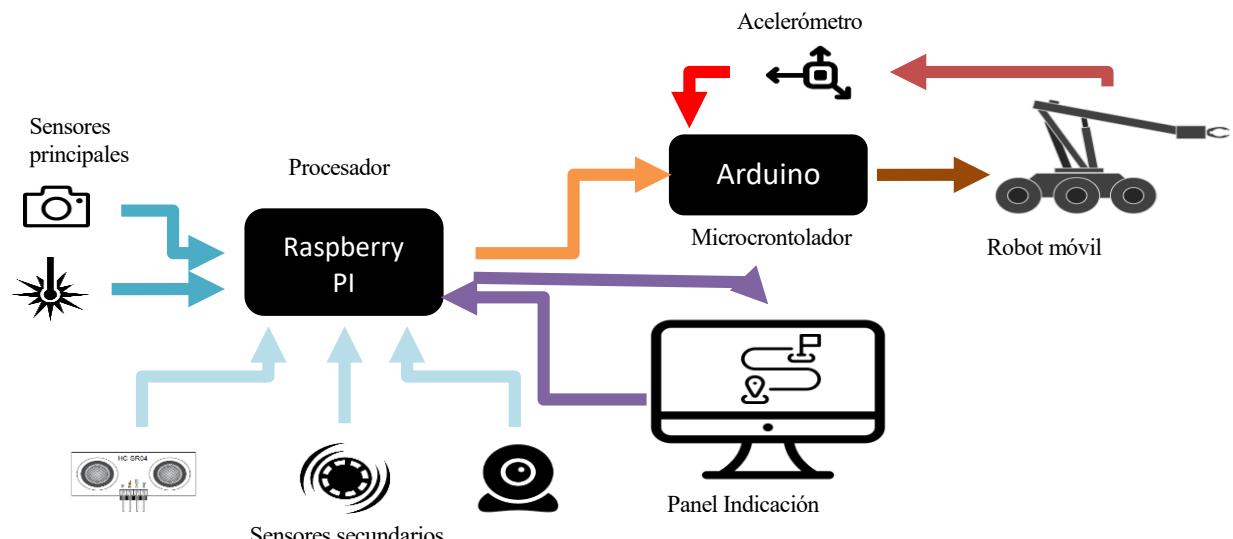


Figura 5-1: Esquema completo robot complementario

<sup>20</sup> Peripheral Interface Controller (PIC) son una familia de microcontroladores fabricados por Microchip Technology Inc. y derivados del PIC1650, originalmente desarrollado por la división de microelectrónica de General Instrument

<sup>21</sup> La traducción a FPGA no sería para nada inmediato como otras opciones. Valdría más la pena empezar de nuevo en este tipo de dispositivo.

# 6 COSTES

Ya que hemos mencionado en numerosas ocasiones el coste, no es para menos dedicarle al menos una página para hablar cuantitativamente de ello. Para analizar los costes que tendría el robot, diferenciaremos varios casos de robot y estudiaremos el coste de sus componentes. Se excluyen los costes de fabricación, empaquetado, ingeniería, logística, etc.

1. Robot 1: Es el “robot” de este proyecto, sería el precio base para poder realizar el SLAM. Contamos con la cámara, el láser y la plataforma.
2. Robot 2: Sería una versión realmente móvil y autónoma del proyecto (como vimos en el apartado anterior). Dispondría además del robot anterior, una plataforma móvil con sus respectivos sensores y control de movimiento.
3. Robot 3: Es la versión de venta al público del robot 2. Incluiría más sensores para asegurar su robustez, y una pantalla para indicación y mando del mismo.
4. Robot 4: Equivale al robot comercial actual con láser 2D o 3D. Es similar al robot 3, cambiando su fuente de información principal.

En la siguiente tabla se analizan detalladamente los costes de cada tipo de robot:

ROBOT 1	Precio 1	ROBOT 2	Precio 2	ROBOT 3	Precio 3	ROBOT 4	Precio 4
Procesador	40.00 €	Procesador	40.00 €	Procesador+	70.00 €	Procesador+	70.00 €
Cámara	59.99 €	Cámara+	72.99 €	Cámara+ 2x	90.00 €	Cámara -	40.00 €
Láser	2.00 €	Láser+	5.00 €	Láser+	7.00 €	Laser SICK	300.00 €
Plataforma	5.00 €	Plataforma móvil	10.00 €	Plataforma móvil	12.50 €	Plataforma móvil	12.50 €
Piezas de enlace	2.00 €	Microcontrolador	9.00 €	Microcontrolador	9.00 €	Microcontrolador	9.00 €
Alimentación	10.00 €	Motores	16.00 €	Motores+	24.00 €	Motores+	24.00 €
		Puente H	4.00 €	Puente H	4.00 €	Puente H	4.00 €
		Sensores	5.00 €	Sensores Movimiento	5.00 €	Sensores Movimiento	5.00 €
		Otros	5.00 €	Sensores distancia	15.00 €	Sensores distancia	15.00 €
		Alimentación	12.00 €	Pantalla	28.00 €	Pantalla	28.00 €
				Otros	20.00 €	Otros	20.00 €
<b>TOTAL</b>	<b>118.99 €</b>	<b>TOTAL</b>	<b>178.99 €</b>	<b>TOTAL</b>	<b>284.50 €</b>	<b>TOTAL</b>	<b>527.50 €</b>

Tabla 6-1: Costes de los robots 1-4

Los elementos que se indican con el signo “+” quieren decir que es una versión que ofrece más prestaciones que el modelo sin este signo.

Cabe considerar que estos costes son si compramos las piezas como venta al público por separado y lo montamos nosotros. El coste de cada robot si fuera producto de una cadena de producción y compráramos al por mayor disminuiría aproximadamente a la mitad. Además se ha incluido el precio de la cámara completa, mientras que sólo sería necesario el sensor y la lente (con su circuito de control).

Observando la tabla, vemos como el precio de implementar el SLAM con visión nos costaría de base unos 100 euros y la construcción del robot autónomo rondaría los 180 euros. Si cambiamos la visión por el láser, como vemos en la última columna, el precio incrementa considerablemente.

Por lo tanto podemos afirmar que un robot que implemente SLAM con visión será indudablemente más barato, aunque perderemos en términos de robustez y eficiencia, lo que puede solucionarse con algoritmos software más avanzados y unas cámaras más capaces de enfrentarse a condiciones lumínicas adversas, (como las cámaras infrarrojas).

# 7 CONCLUSIONES DEL PROYECTO

Los resultados específicos de cada parte ya han sido tratadas en los correspondientes apartados de cada hito, por lo que únicamente nos gustaría recalcar algunas conclusiones desde otras perspectivas, tales como la viabilidad, el aprendizaje, y la inclusión dentro de la carrera de Ingeniería Electrónica, Robótica y Mecatrónica.

- El hecho de cambiar la perspectiva tradicional del SLAM, la cual trabaja principalmente con sensores de distancia láser, nos ha aportado ciertas ventajas e inconvenientes. Como ventajas principales, la visión es el medio natural en que realizamos la localización los humanos y, a mi parecer, es el sentido en el que deberían evolucionar los métodos de localización y mapeo simultáneo.
- Los ceilmarks, introducidos en este proyecto, son como hemos visto una buena fuente de información. En primer lugar porque nos pueden proporcionar rápidamente el mapa, en segundo lugar es un medio que no suele tener muchos obstáculos y sobre todo es una fuente estática y presente en todos los espacios interiores.
- Este proyecto es muy multidisciplinar, ya que abarca numerosas asignaturas estudiadas en la carrera y otras no estudiadas. Hemos visto desde procesado de imagen, robótica, visión por computador, hasta elementos como modelado de sistemas y filtrado. A título personal ha sido una experiencia muy enriquecedora, tanto por los conceptos que hemos tenido que ampliar, los que hemos tenido que recordar y los que hemos aprendido desde cero.

Como conclusión última quiero hacer hincapié en la valía de este proyecto y proyectos como este que a partir de una idea (localizar a un robot en un entorno desconocido) y el ingenio y la planificación adecuada, son capaces de hacerlo realidad y materializarlo, que es en síntesis el trabajo de un ingeniero.

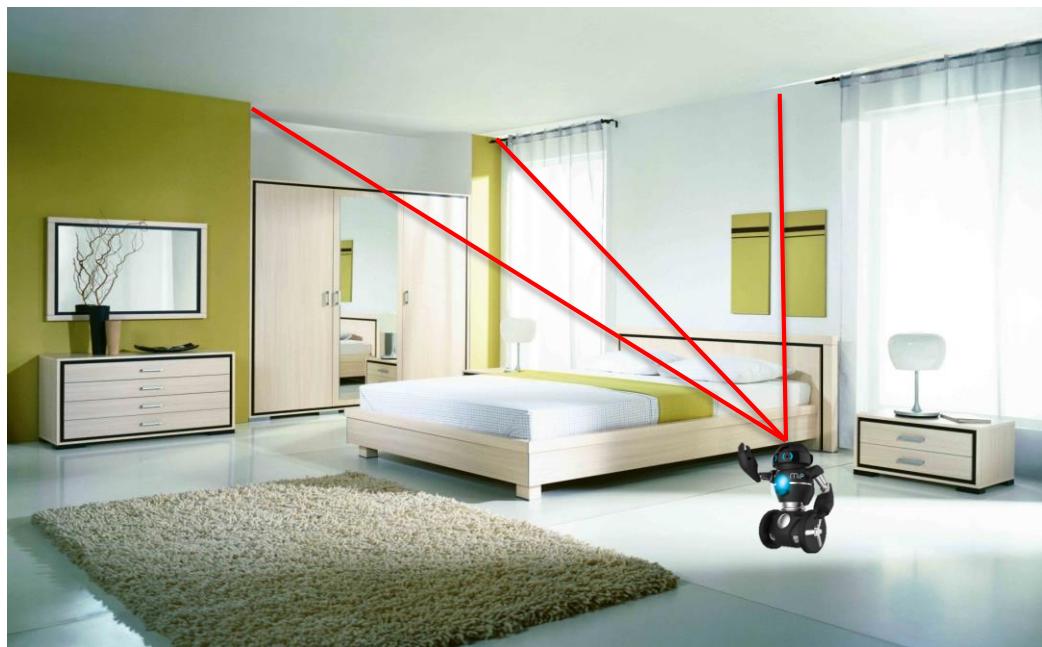


Figura 7-1: Idea original

# REFERENCIAS

---

- [1] J. Viñals Pons, «Localización y generación de mapas del entorno (SLAM) por medio de una Kinect,» Valencia, 2012.
- [2] D. J. Aguilar, «Construcción de mapas probabilísticos mediante técnicas de SLAM en entorno ROS,» Alcalá de Henares, 2014.
- [3] Federico Andrade y Martín Llofriu, «SLAM ; Estado del Arte,» Montevideo, Uruguay.
- [4] R. Szeliski, de *Computer Vision: Algorithms and Applications*, Springer Science & Business Media, 2010, pp. 10-11.
- [5] J. E. Solem, *Programming computer vision with Python*, O'Reilly, 2012.
- [6] Rafael C. Gonzalez, Richard Woods, Steven L. Eddins, *Digital Image Processing Using Matlab*, McGraw Hill, 2010.
- [7] Søren Riisgaard, Morten Rufus Blas, «SLAM for Dummies».
- [8] J. Solà, «An EKF-SLAM toolbox in Matlab,» 2013.
- [9] G. Zunino, *Simultaneous Localization and Mapping for Navigation in Realistic Environments*, Estocolmo, 2002.
- [10] José Antonio Puértolas Montañés, Adriana Mendoza Rodríguez, Iván Sanz Prieto, «Smart Indoor Positioning/Location and Navigation: A Lightweight Approach,» *International Journal of Artificial Intelligence and Interactive Multimedia*, vol. 2, nº 2.
- [11] Naveed Muhamad, David Sofi, Samia Ainouz, “Current state of art of vision based SLAM,” Bourgogne, France.
- [12] A. Pascual, «EKF y UKF: dos extensiones del filtro de Kalman para sistemas no lineales aplicadas al control de un péndulo invertido,» Uruguay, 2006.
- [13] J. M. Sebastián, «Detección de Esquinas y Vértices,» de *VISIÓN POR COMPUTADOR*, Madrid, 2010.
- [14] Luis Mejias, Peter Corke, Jonathan Roberts, «Field and Service Robotics,» de *Results of the 9th International Conference*, Switzerland, 2015.
- [15] J. A. C. Ibarrola, *El filtro de Kalman*.
- [16] P. M. Newman, *EKF Based Navigation and SLAM*, Oxford, 2006.
- [17] JOAQUIM R. R. A. MARTINS, PETER STURDZA, JUAN J. ALONSO, «The Complex-Step Derivative Approximation,» Stanford, Toronto.

- [18] H. Papasaika-Hanusch, *Digital Image Processing Using Matlab*, Institute of Geodesy and Photogrammetry, ETH Zurich.
- [19] L. J. M. Paniagua, *LOCALIZACIÓN DE ROBOTS MÓVILES DE RECURSOS LIMITADOS BASADA EN FUSIÓN SENSORIAL POR EVENTOS*, Vaencia, 2014.
- [20] D. Litwiller, «CCD vs CMOS: Facts and Fiction,» *PHOTONICS SPECTRA*, 2001.
- [21] P. Corke, *Robotics, Vision and Control; FUNDAMENTAL ALGORITHMS*, 2011.
- [22] E. A. S. MALPARTIDA, «SISTEMA DE VISIÓN ARTIFICIAL PARA EL RECONOCIMIENTO Y MANIPULACIÓN DE OBJETOS UTILIZANDO UN BRAZO ROBOT,» Lima, 2003.
- [23] P. López Escobés, «Summary of the State of the Art in indoor location systems,» Valladolid, 2009.
- [24] Francisco Bonin-Font, Alberto Ortiz and Gabriel Olivier, «Visual Navigation for Mobile Robots: a Survey,» Palma de Mallorca, 2005.
- [25] C. P. González, «Detección y seguimiento de objetos por colores en una plataforma raspberry PI,» Madrid, 2016.
- [26] J. L. G. Bruque, «Filtros de Kalman Extendido y “Unscented” en Sistemas Estocásticos no Lineales con Observaciones Inciertas,» Granada, 2011.
- [27] Héctor M. Becerra, Carlos Sagüés, «Visual Control of Wheeled mobile robots,» *Springer Tracts in Advanced Robotics*, nº 103, 2014.
- [28] Diego Aracena Pizarro, Pedro Campos, Clésio Luis Tozzi, «Comparación de técnicas de calibración de cámaras digitales,» *Fac. Ing. - Univ. Tarapacá*, vol. 13, nº 1, pp. 57-67, 2005.
- [29] E. Schwalbe, «GEOMETRIC MODELLING AND CALIBRATION OF FISHEYE LENS CAMERA SYSTEMS,» Dresden.
- [30] D. Scaramuzza, «scarabotix; Ocamlcalib-toolbox,» [En línea]. Available: <https://sites.google.com/site/scarabotix/ocamcalib-toolbox>.

# GLOSARIO

---

## Fundamentales

- EKF: filtro extendido de Kalman ..... xiv, xvii, 11, 15, 16, 44, 45, 46, 50, 52, 63  
KF: filtro de Kalman ..... xiv, xvii, 16, 44, 45, 50  
SLAM: localización y mapeo simultáneo iii, v, xi, xii, xiv, xv, xvi, xvii, 11, 12, 14, 15, 16, 42, 44, 47, 49, 50, 51, 52, 53, 54, 58, 62, 63, 64, 65

## Otros

- CCD ..... 71  
CMOS ..... 71  
CMYK ..... 18  
DFT ..... xvi, 33, 34, 35, 37, 70, 74  
DSP ..... 64  
FPGA ..... 64  
HSL ..... 18  
IDFT ..... 33, 34, 37  
LDR ..... 38  
MOSFET ..... 71  
MP ..... 21  
PC ..... 64  
PIC ..... 64  
RGB ..... 18  
UKF ..... xiv, xvii, 44, 46  
VGA ..... 21

# ANEXOS

---

<i>Anexo A: Sensores CCD vs CMOS</i>	80
<i>Anexo B: Patrón de calibración</i>	81
<i>Anexo C: Parámetros de retorno – Occam Calib</i>	82
<i>Anexo D: Algoritmo detección del láser (I)</i>	82
<i>Anexo E: Algoritmo detección del láser (II)</i>	83
<i>Anexo F: Ejemplo código clasificador características</i>	84
<i>Anexo G: Detección de esquinas DFT</i>	85
<i>Anexo H: Imagen habitación Ideal</i>	86
<i>Anexo I: Función undistorts</i>	88
<i>Anexo J: Función completa extracción de características</i>	89
<i>Anexo K: Función de re-distorsión</i>	90
<i>Anexo L: Algoritmo EKF utilizado</i>	90
<i>Anexo M: Algorimto UKF utilizado</i>	91
<i>Anexo N: Relación Ángulo visión superficie</i>	94
<i>Anexo Ñ: Parámetros y errores triciclo</i>	95
<i>Anexo O: Código completo SLAM</i>	95

### Anexo A: Sensores CCD vs CMOS

En un sensor CCD, los condensadores de la región fotosensible se encargan de acumular una carga eléctrica proporcional a los fotones incidentes; una vez finalizada la exposición propiamente dicha, la carga de los condensadores es transmitida a sus vecinos mediante un circuito de control (registro de desplazamiento). Así sucede sucesivamente hasta llegar al último condensador que transmite esta carga a un amplificador que a su vez la transforma en un voltaje proporcional. En un sensor digital, este voltaje será convertido a una señal digital mediante el correspondiente conversor A/D.

El CMOS (Complementary Metal-Oxid-Semiconductor) es una tecnología utilizada para la fabricación de circuitos integrados, basada en transistores MOS o MOSFET (Metal–Oxid–Semiconductor Field-Effect Transistor). El CMOS es una variante que integra típicamente dos transistores para obtener un diseño simétrico para generar todo tipo de funciones lógicas.

La principal diferencia con el diseño CCD es que en el CMOS los píxeles son tratados de forma individual; no existe transferencia de la carga eléctrica, como en el CCD. En cambio, en este último la lectura es simultánea en toda la matriz de píxeles durante la exposición.

El diseño de sensores con esta tecnología CMOS permite además combinar el propio sensor de imagen con las funciones de procesado de imagen en el mismo circuito integrado. En el diseño basado en la tecnología CCD, el sensor de imagen es un elemento totalmente diferenciado, y por tanto no puede ser integrado como el CMOS. Sin embargo, gracias a ello, los sensores CCD tienen menos ruido que los CMOS.

En resumen, los sensores CMOS tienen un menor coste de producción y un consumo más reducido que los CCD, pero estos últimos tienen menos ruido y una mayor eficiencia lumínica. Hoy en día es difícil determinar claramente cuál de las dos tecnologías ofrecer las mejores prestaciones en cuanto a calidad de imagen se refiere.

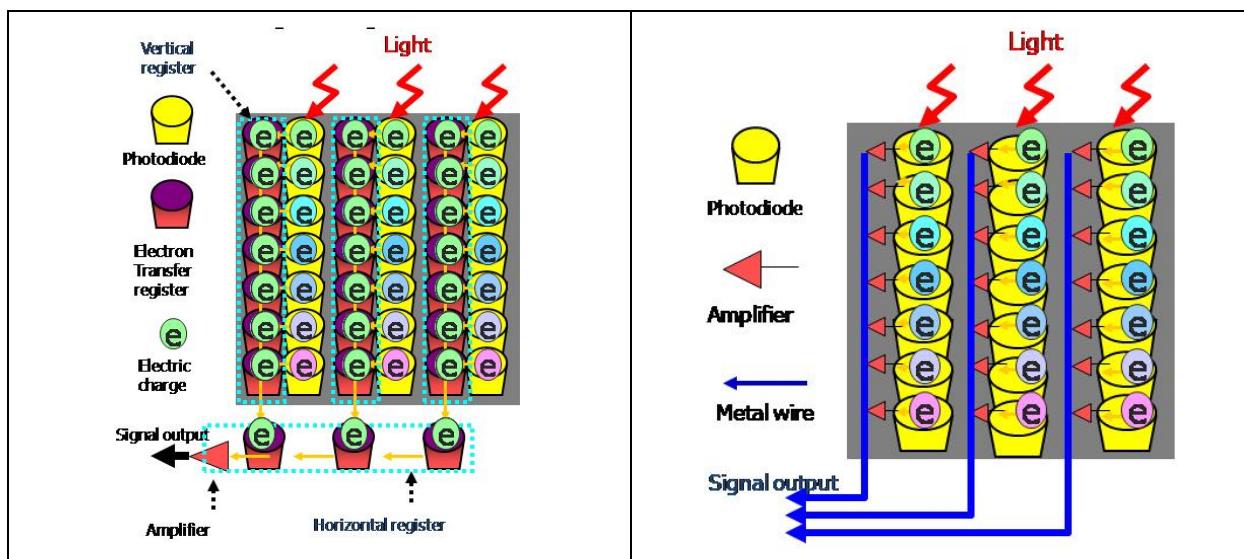


Figura Anexo 1: sensor CCD (izquierda) vs CMOS (derecha)

Anexo B: Patrón de calibración

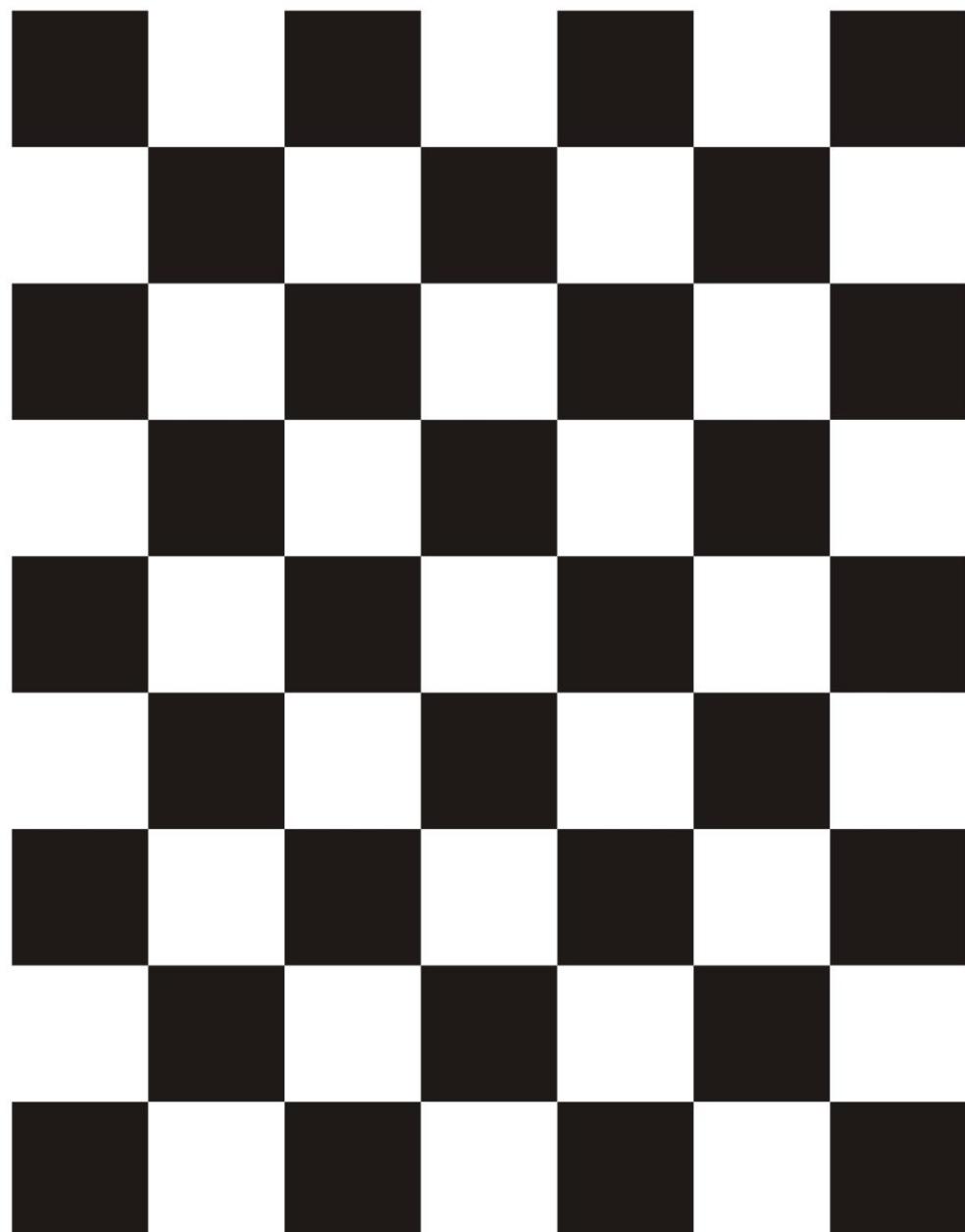


Figura Anexo 2 : Patrón de calibración a escala 1:1

### Anexo C: Parámetros de retorno – Occam Calib

La librería de Davide Scaramuzza la podemos descargar de <https://sites.google.com/site/scarabotix/ocamcalib-toolbox>. En esta sección expondremos que parámetros nos devuelve la función de calibración.

<b>Ocam_model = { ss, xc, yc, c, d, e, width, height }</b>	Estructura que contiene el modelo completo de la cámara, con elementos definidos aquí abajo.
<b>Ocam_model.ss</b>	Coeficientes del polinomio F
<b>ocam_model.xc</b>	Coordenadas del centro 82ptico de la imagen ojo de pez.
<b>ocam_model.yc</b>	
<b>Ocam_model.c</b>	
<b>ocam_model.d</b>	Parámetros de afinamiento de imágenes
<b>ocam_model.e</b>	
<b>ocam_model.width</b>	Ancho y alto de la imagen procesada
<b>ocam_model.height</b>	
<b>Xp_abs</b>	Fila y columna de las esquinas procesadas en absoluto respecto a la posición central
<b>Yp_abs</b>	
<b>Xt</b>	
<b>Yt</b>	Fila y columna de las esquinas en relativo
<b>ima_proc</b>	Número de imágenes usadas en el proceso de calibración
<b>Rrfin</b>	Parámetros extrínsecos de todos los tableros
<b>err</b>	Error de reporyección

Tabla 0-1: Parámetros de retorno calibración

### Anexo D: Algoritmo detección del láser (I)

El algoritmo se incluye también en la carpeta de archivos, se titula `laser_detect_figures_v2` el cual tiene una demo con una fotografía concreta.

```

clc;clear;
%% tratamiento imagen techo
% bueno si no hay luz
% se basa en que es puro rojo
% aparte lo cruza con la distancia
load Omni_Calib_Results.mat
load p_order4.mat
fisheye_model=calib_data.ocam_model;
%%
s0='laser150.jpg';s1='laser100.jpg';s2='laser80.jpg';
s3='laser60.jpg';s4='laser5520.jpg';s5='winp9.jpg';
s6='winpl1.jpg';
%%
f=imread(s5);
luz=0;
% Separacion de canales
fr = f( :,:,1);
fg = f( :,:,2);
fb = f( :,:,3);

%% resta de colores
frg=fr-fg;
% los colores que sean amarillos o blancos/grises desaparecen
% quedarian los rojos, si el techo es rojo no vale

% threshold
t=30;
frg_t=frg>t;

%% encontrar el laser
% en este caso buscamos un uno
esc=0.5;
Im_th_res = (imresize(frg_t, esc));
% tratamiento de la luz
[M,N]=size(Im_th_res);
yc=M/2;
xc=N/2;

xcc=(calib_data.ocam_model.xc); %xc e yc estan cambiados ya que ejex va hacia abajo
ycc=(calib_data.ocam_model.yc);
xrc=round(ycc/2);
yrc=round(xcc/2);

dmin=M^2+N^2; %es el maximo posible

for i=1:M
    for j=1:N
        if (Im_th_res(i,j)==1)
            d=(j-xc)^2+(i-yc)^2;
            if (d<dmin)
                dmin=d;
                y_laser=i;
                x_laser=j;
            end
        end
    end
end

%% En la imagen original
x_laser_f = round(x_laser / esc)
y_laser_f = round(y_laser / esc)
xc_f=round(xc / esc)
yc_f=round(yc/esc);

A=[y_laser_f;x_laser_f];
B=[yc_f;xc_f];
distp=norm(A-B);

```

#### Anexo E: Algoritmo detección del láser (II)

El algoritmo se incluye también en la carpeta de archivos, se titula `laser_detect_figures_v3` el cual tiene una demo con una fotografía concreta.

```

clc;clear all;
%% tratamiento imagen techo V2
% es el más rápido
% se supone que el laser es cercano al centro
% ademas supera un cierto treshold
addpath('C:\Users\Alberto\Google Drive\GIERM\TFG\1_Fisheye_and_Scaramuzza\Scaramuzza');
load Omni_Calib_Results.mat
fisheye_model=calib_data.ocam_model;
%f=imread('valida_conlaser_real_conluz.jpg');
f=imread('winp11.jpg');
luz=0;
% Separacion de canales
fr = f( :,:,1);
fg = f( :,:,2);
fb = f( :,:,3);
% Tonos de gris
suma = double(fr)+ double(fg) +double(fb);
fgris = uint8( suma/3 );

% reducimos la imagen a la mitad
esc=0.5;
fr_res =(imresize(fr, esc));
%%
% metodos morfológicos
r=5;
se = strel('disk',r);
fr_res_ad=imadjust(fr_res);
%fr_res_ad=adapthisteq(fr_res);
Im_th_res=imtophat(fr_res_ad,se);

%% Encontrar el laser
[M,N]=size(Im_th_res);
t=100; % como hemos hecho imadjust no hay problemas de poca luz
% no es el centro optico de verdad
xc0=N/2
yc0=M/2
xcc=(calib_data.ocam_model.xc); %xc e yc estan cambiados ya que ejex va hacia abajo
ycc=(calib_data.ocam_model.yc);
xc=round(ycc/2)
yc=round(xcc/2)
xrc=xc;
yrc=yc;

dmin=M^2+N^2; %es el maximo posible
maxth=0; % es el minimo posible

%% Recorrido pixel pixel
for i=1:M
    for j=1:N
        if (Im_th_res(i,j)>t) % buscamos el maximo top hat
            d=(j-xc)^2+(i-yc)^2;
            %criterio de distancia
            if (d<dmin)
                dmin=d;
                y_laser=i;
                x_laser=j;
            end
        end
    end
end

% en la imagen original
x_laser_f = round(x_laser / esc)
y_laser_f = round(y_laser / esc)
% en la imagen original
% x_laser_f2 = round(x_laser2 / esc)
% y_laser_f2 = round(y_laser2 / esc)
xc_f=round(xc / esc)
yc_f=round(yc/esc)

```

#### Anexo F: Ejemplo código clasificador características

```

%% Discriminante por distancia
% -----
% prueba clasificador min dist

% prototipos
p=[0.09 0.52;
   -0.885 0.465;
   0.7 0.06];

%Matriz de datos 2 caracteristicas, 3 clases
M =[ 0.10 0.08 -0.90 -0.87 0.60 0.80 ;
      0.50 0.54 0.45 0.48 0.05 0.07 ;
      1 1 2 2 3 3];

%Cálculo de prototipos
sump1=[0;0];sump2=sump1;sump3=sump1;
cnt1=0;cnt2=0;cnt3=0;
[nf,nc]=size(M);
for i=1:nc
    clase= M(3,i);
    if clase==1
        sump1=sump1+M(1:2,i);
        cnt1=cnt1+1;
    elseif clase==2
        sump2=sump2+M(1:2,i);
        cnt2=cnt2+1;
    elseif clase==3
        sump3=sump3+M(1:2,i);
        cnt3=cnt3+1;
    end
end
% hago la media
p1=sump1/cnt1
p2=sump2/cnt2
p3=sump3/cnt3

% prueba clasificacion
xo=[-0.60; 0.5];
fprintf('La muestra es : [%d ; %d];\n',xo(1),xo(2));
dif=xo-p1; d1=sqrt(dif'*dif);
dif=xo-p2; d2=sqrt(dif'*dif);
dif=xo-p3; d3=sqrt(dif'*dif);

v=[d1 d2 d3];
[C I]=min(v);

fprintf('Es de clase : %d\n',I);

figure;
plot(p1(1),p1(2),'+' );text(p1(1),p1(2),'prototipo 1');
hold on;grid on;
plot(p2(1),p2(2),'*' );text(p2(1),p2(2),'prototipo 2');
plot(p3(1),p3(2),'^' );text(p3(1),p3(2),'prototipo 3');
plot(xo(1),xo(2),'+' );text(xo(1),xo(2),'Xo');
xlabel('caracteristica x1');ylabel('caracteristica x2');
hold off;

```

## Anexo G: Detección de esquinas DFT

```

%% Fft FIND CORNERS
% se basa en que las lineas mas largas
horizontales y verticales
% son las que delimitan el techo y la pared

%% INICIO
addpath('C:\Users\Alberto\Google
Drive\GIERM\...
TFG\1_Fisheye_and_Scaramuzza\Scaramuzza');
close all;clc;
IM=imread('winp7.jpg');
%IM=imread('Imagenideal.png');
Img=rgb2gray(IM);
%cargamos parametros
loading_calib

%% desdistorsion
Img_u=undistorts(calib_data.ocam_model, Img
,2, 0);
[M,N]=size(Img_u);

%% imagen ideal
% Img_u=rgb2gray(imread('Imagenideal.png'));
% [M,N]=size(Img_u);
%% rotacion
% Img_u = imrotate(Img_u,-
1.5,'bilinear','crop');
%% inicializacion
filtersq=zeros(M,N);

incv=round(0.2*M);
% en fourier vertical y horizontal cambian
inch=round(0.15*N);
inc=7;
%% filtros
filtersq(M/2-inc:M/2+inc,N/2-inch:N/2+inch)=1;
filtersq(M/2-incv:M/2+incv,N/2-inc:N/2+inc)=1;

imfilsq=abs(mifilt2(Img_u,filtersq,2));
imfil2sq=uint8(round(255*mat2gray(imfilsq)));


%% imagenes filtradas
esc=0.5;
imesq_res=imresize(imfil2sq, esc);

%% buscamos los minimos
%M_points = findhistov2 (imfil2sq)
%% buscamos los minimos
[~,vposx]=min(imfil2sq,[],2);
[~,vposy]=min(imfil2sq);

edges=30;
[valx , xpos] = histcounts(vposx);
[valy , ypos] = histcounts(vposy);

% tolerancia
tol=10;
% buscamos en el histograma
[distx, x1,x2]=findinhisto(xpos,valx,N,tol)
[disty, y1,y2]=findinhisto(ypos,valy,M,tol)

M_corte(1,:)=[x1,y1];
M_corte(2,:)=[x1,y2];
M_corte(3,:)=[x2,y1];
M_corte(4,:)=[x2,y2];

%% imagen final
figure;
imshow(Img_u); hold on;
title('Imagen final tras la búsqueda de
esquinas');
xlabel('Procedimiento -> filtrado espacial
fft');
for i=1:4

% plot(M_points(i,1),M_points(i,2),'-
gs','MarkerSize',12,...
plot(M_corte(i,1),M_corte(i,2),'-
gs','MarkerSize',12, ...
'MarkerFaceColor','w');
end
hold off;

%% PLOTS
figure;
subplot(211);imshow(Img_u);title('Imagen
original');
subplot(212);imshow(imfil2sq); title('Imagen
filtrada');
figure;mesh(imesq_res);title('Imagen filtrada
(mesh)');

figure;
imshow(Img_u);title('Imagen original');
figure;
mesh(log(1+abs(fftshift(fft2(Img_u))));;
title('FFT2 IMAGEN ORIGINAL');

```

#### Anexo H: Imagen habitación Ideal

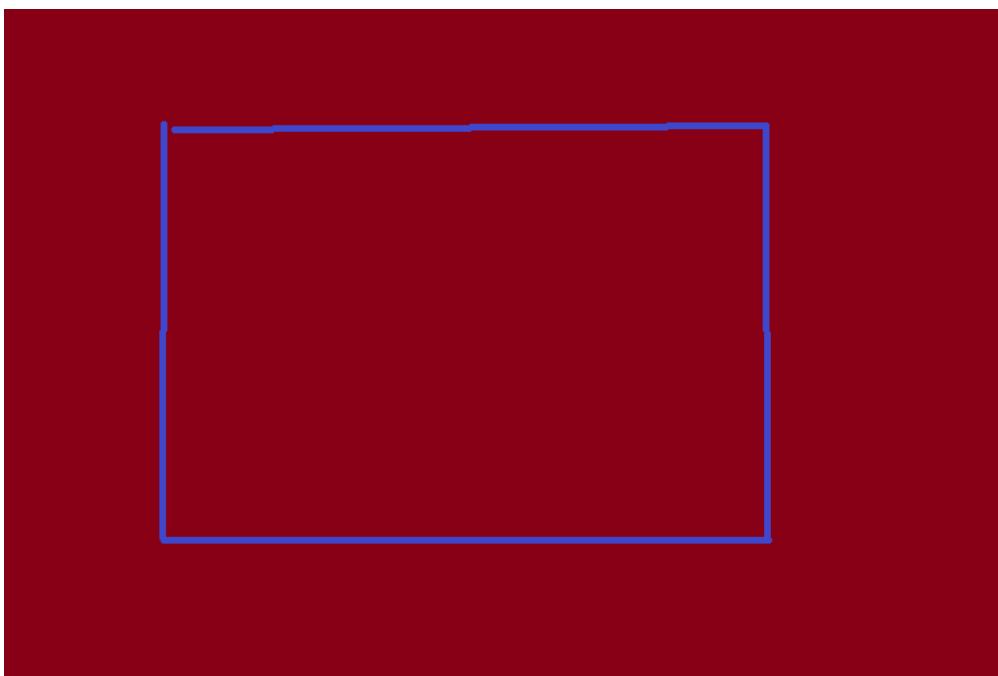
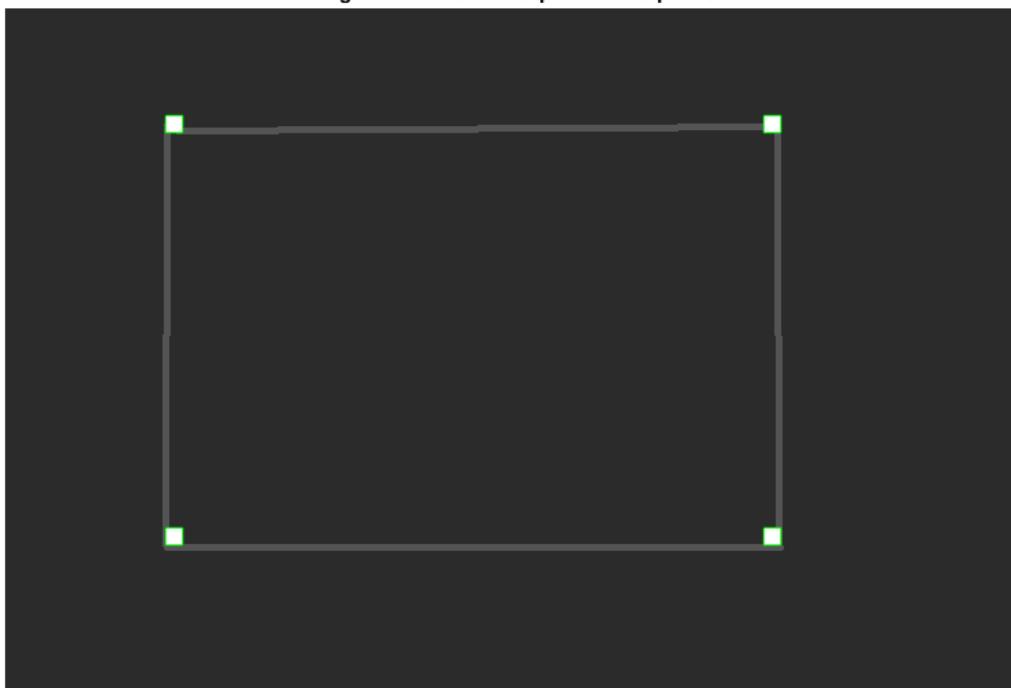


Imagen final tras la búsqueda de esquinas



Procedimiento -> filtrado espacial fft

Figura Anexo 3: Busqueda esquinas imagen ideal

### Anexo I: Función undistorts

```
%UNDISTORT unwrap part of the image onto a plane perpendicular to the
%camera axis
%   B = UNDISTORT(OCAM_MODEL, A, FC, DISPLAY)
%   A is the input image
%   FC is a factor proportional to the distance of the camera to the plane;
%   start with FC=5 and then tune the parameter to change the result.
%   DISPLAY visualizes the output image if set to 1; its default value is
%   0.
%   B is the final image
%   Note, this function uses nearest neighbour interpolation to unwrap the
%   image point. Better undistortion methods can be implemented using
%   bilinear or bicub interpolation.
%   Note, if you want to change the size of the final image, change Nwidth
%   and Nheight
% Author: Davide Scaramuzza, 2009

function Nimg = undistorts( ocam_model, img , fc, display)

if nargin < 3
    fc = 5;%distance of the plane from the camera, change this parameter to zoom-in or out
    display = 0;
end

% Parameters of the new image
Nwidth = 1280; %size of the final image
Nheight = 720;
Nx = Nheight/2;
Ny = Nwidth/2;
Nz = -Nwidth/fc;

if ~isfield(ocam_model,'pol')
    width = ocam_model.width;
    height = ocam_model.height;
    %The ocam_model does not contain the inverse polynomial pol
    ocam_model.pol = findinvpoly(ocam_model.ss,sqrt((width/2)^2+(height/2)^2));
end

if length(size(img)) == 3;
    Nimg = zeros(Nheight, Nwidth, 3);
else
    Nimg = zeros(Nheight, Nwidth);
end

[i,j] = meshgrid(1:Nheight,1:Nwidth);
Nx = i-Nx;
Ny = j-Ny;
Nz = ones(size(Nx))*Nz;
M = [Nx(:);Ny(:);Nz(:)];
m = world2cam_fast( M , ocam_model );
% del centro
% 269
% 450
if length(size(img)) == 2
    I(:,:,1) = img;
    I(:,:,2) = img;
    I(:,:,3) = img;
    % añadido por alberto
else
    I=img;
end

[r,~,~] = get_color_from_imagepoints( I, m' );
Nimg = reshape(r,Nwidth,Nheight');

if display
    figure; imagesc(Nimg); colormap(gray);
end
```

## Anexo J: Función completa extracción de características

Esta función es muy extensa, ocupa completamente desplegada más de 500 líneas, por lo que pondremos solamente el código principal con las llamadas a funciones.

```
function data=A_findcorners2 (sim,fc,tol2,suposeall,graph,mplots)
%% ALBERTO GONZALEZ ISORNA
% sim      -->     nombre de la imagen en formato string
% fc       -->     factor de cercanía normalmente 2 desdistorsion
% tol2    -->     porcentaje de linea/imagen (usual 0.65)
% supposeall -->   1 si suponemos que vamos a encontrar todas o 0 si no
% graph    -->     si queremos ver el proceso de las esquinas
% mplots   -->     para ver el proceso final

%% diferencias con v1
% la diferencia con la primera version es que siempre devuelve una matriz
% con las cuatro esquinas aunque no haya nada
tic;
% INICIO Y DESDISTORSION
M_corte=zeros(4,2);
calib_data = load_myfishhmodel ();
[img,img_u] = inicio (sim,fc,calib_data);
[M,N]=size(img_u);
MN=M*N;
% IMAGEN BINARIA OPTIMA
BWO = findbinopt (sim,img_u,MN);
% ENCONTRAMOS LAS 4 lineas
[line1,line2,line3,line4,fc1,fc2,fc3,fc4]= findalldesired (BWO,M,N,graph,tol2);
% ORGANIZAMOS LOS CASOS
disp('terminando....');
%% tenemos las cuatro lineas
if (isa(line1,'struct') && isa(line2,'struct') &&...
    isa(line3,'struct') && isa(line4,'struct')) )
num=0;cont=0;
M_corte = interpret_line (line1,line2,line3,line4,fc1,fc2,fc3,fc4,num,suposeall);
else
    disp('alguna linea no encontrada');
    %% otros casos
    [cont, n] = cuenta (line1,line2,line3,line4); % contamos que ocurre
    %% tenemos 3
    if (cont==1)
        num=find(n==1); %falta la uno
        M_corte = interpret_line (line1,line2,line3,line4,fc1,fc2,fc3,fc4,num,suposeall);
    end
    %% tenemos 2
    if (cont==2)
        n1=find(n==0);
        num=10*n1(1)+n1(2); % intrudicmos las lineas
        M_corte = interpret_line (line1,line2,line3,line4,fc1,fc2,fc3,fc4,num,suposeall);
    end
    %% tenemos 1 o ninguna
    if (cont>2)
        disp('no puedo encontrar nada...');
    end
end
%% FINALIZACION
t2=toc;
s1=sprintf('\n-> Finished in %g seconds',t2);
s2=sprintf('-> Numero de esquinas encontradas : %d\n',(4-cont));
disp(s1);
disp(s2);
%% DESDISTORSION
M_corte=round(M_corte);
M_cortedis=inverse_undistort(calib_data.ocam_model,M_corte,fc);
M_cortedis=round(M_cortedis);
%% FIGURAS
if (mplots)
    final_plots (img_u,img,M_corte,M_cortedis);
end;
% Atencion que el formato de M_corte y M_cortedis no es igual
% M_cortedis [row;col];
% M_corte [col row];
%% DATOS
data.BWO      = BWO;
data.M        = M;
data.N        = N;
data.M_corte = M_corte;
```

```

data.M_cortedis = M_cortedis;
data.lineas     = (4-cont);
data.model      = calib_data.ocam_model;
data.img_u      = img_u;
data.runtime    = t2;
end

```

### Anexo K: Función de re-distorsión

```

function m = inverse_undistort( ocam_model, points , fc)

% INVERSE_UNDISTORT
%camera axis

%points
% xu1 yu1
% xu2 yu2
% xu3 yu3
% ... ...

[mm,~]=size(points);
M=zeros(3,mm);

ti=points(:,2); % la x e y estan cambiadas
tj=points(:,1); % la x e y estan cambiadas

Nwidth = 1280; %size of the final image
Nheight = 720;
Nx = Nheight/2;
Ny = Nwidth/2;
Nz = -Nwidth/fc;

M(1,:)=ti - Nx ;
M(2,:)=tj - Ny ;
M(3,:)=Nz*ones(1,mm);

norm=sqrt(M(1,:).^2+M(2,:).^2+M(3,:).^2);

% normalizamos el vector
for i=1:mm
M(:,i)=M(:,i)./norm(i);
end

m = world2cam( M , ocam_model );
end

```

### Anexo L: Algoritmo EKF utilizado

```

function [x,P]=ekf(fstate,x,P,hmeas,z,Q,R)
% EKF Extended Kalman Filter for nonlinear dynamic systems
% [x, P] = ekf(f,x,P,h,z,Q,R) returns state estimate, x and state covariance,
P
% for nonlinear dynamic system:
%           x_k+1 = f(x_k) + w_k
%           z_k   = h(x_k) + v_k
% where w ~ N(0,Q) meaning w is gaussian noise with covariance Q
%           v ~ N(0,R) meaning v is gaussian noise with covariance R
% Inputs:   f: function handle for f(x)
%           x: "a priori" state estimate
%           P: "a priori" estimated state covariance
%           h: function handle for h(x)
%           z: current measurement
%           Q: process noise covariance
%           R: measurement noise covariance
% Output:   x: "a posteriori" state estimate
%           P: "a posteriori" state covariance

[x1,A]=jaccsd(fstate,x);          %nonlinear update and linearization at current
state
P=A*P*A'+Q;                      %partial update
[z1,H]=jaccsd(hmeas,x1);         %nonlinear measurement and linearization
P12=P*H';                          %cross covariance

%% FORM 1
% K=P12/(H*P12+R);              %Kalman filter gain
% x=x1+K*(z-z1);                %state estimate
% P=P-K*P12';                    %state covariance matrix
%% FORM 2
R=chol(H*P12+R);                %Cholesky factorization
U=P12/R;                         %K=U/R'; Faster because of back substitution
x=x1+U*(R'\(z-z1));            %Back substitution to get state update
P=P-U*U';                        %Covariance update, U*U'=P12/R/R'*P12'=K*P12.

function [z,A]=jaccsd(fun,x)
% JACCS D Jacobian through complex step differentiation
% [z J] = jaccsd(f,x)
% z = f(x)
% J = f'(x)

z=fun(x);
n=numel(x);
m=numel(z);
A=zeros(m,n);
h=n*eps;
for k=1:n
    x1=x;
    x1(k)=x1(k)+h*1i;
    A(:,k)=imag(fun(x1))/h;
end

```

## Anexo M: Algorimto UKF utilizado

```

function [x,P]=ukf(fstate,x,P,hmeas,z,Q,R)
% UKF    Unscented Kalman Filter for nonlinear dynamic systems
% [x, P] = ukf(f,x,P,h,z,Q,R) returns state estimate, x and state covariance,
P
% for nonlinear dynamic system (for simplicity, noises are assumed as
additive):
%
%      x_k+1 = f(x_k) + w_k
%      z_k   = h(x_k) + v_k
% where w ~ N(0,Q) meaning w is gaussian noise with covariance Q
%      v ~ N(0,R) meaning v is gaussian noise with covariance R
% Inputs:  f: function handle for f(x)
%          x: "a priori" state estimate
%          P: "a priori" estimated state covariance
%          h: function handle for h(x)
%          z: current measurement
%          Q: process noise covariance
%          R: measurement noise covariance
% Output: x: "a posteriori" state estimate
%          P: "a posteriori" state covariance
%
% Example:
%{
n=3;           %number of state
q=0.1;         %std of process
r=0.1;         %std of measurement
Q=q^2*eye(n); % covariance of process
R=r^2;         % covariance of measurement
f=@(x)[x(2);x(3);0.05*x(1)*(x(2)+x(3))]; % nonlinear state equations
h=@(x)x(1); % measurement equation
s=[0;0;1]; % initial state
x=s+q*randn(3,1); %initial state with noise
P = eye(n); % initial state covariance
N=20; % total dynamic steps
xV = zeros(n,N); %estmate
sV = zeros(n,N); %actual
zV = zeros(1,N);
for k=1:N
    z = h(s) + r*randn; % measurments
    sV(:,k)= s; % save actual state
    zV(k) = z; % save measurment
    [x, P] = ukf(f,x,P,h,z,Q,R); % ekf
    xV(:,k) = x; % save estimate
    s = f(s) + q*randn(3,1); % update process
end
for k=1:3 % plot results
    subplot(3,1,k)
    plot(1:N, sV(k,:), '-', 1:N, xV(k,:), '--')
end
%}
% Reference: Julier, S.J. and Uhlmann, J.K., Unscented Filtering and
% Nonlinear Estimation, Proceedings of the IEEE, Vol. 92, No. 3,
% pp.401-422, 2004.
%
% By Yi Cao at Cranfield University, 04/01/2008
%
L=numel(x); %numer of states
m=numel(z); %numer of measurements
alpha=1e-3; %default, tunable
ki=0; %default, tunable
beta=2; %default, tunable
lambda=alpha^2*(L+ki)-L; %scaling factor
c=L+lambda; %scaling factor

```

```

Wm=[lambda/c 0.5/c+zeros(1,2*L)]; %weights for means
Wc=Wm;
Wc(1)=Wc(1)+(1-alpha^2+beta);
c=sqrt(c);
X=sigmas(x,P,c);
[x1,X1,P1,X2]=ut(fstate,X,Wm,Wc,L,Q); %weights for covariance
process %sigma points around x
%sigma points around x1
%unscented transformation of
[z1,Z1,P2,Z2]=ut(hmeas,X1,Wm,Wc,m,R); %deviation of X1
%unscented transformation of
measurements %transformed cross-covariance
P12=X2*diag(Wc)*Z2'; %state update
K=P12*inv(P2); %covariance update
x=x1+K*(z-z1);
P=P1-K*P12';

function [y,Y,P,Y1]=ut(f,X,Wm,Wc,n,R)
%Unscented Transformation
%Input:
%      f: nonlinear map
%      X: sigma points
%      Wm: weights for mean
%      Wc: weights for covariance
%      n: numer of outputs of f
%      R: additive covariance
%Output:
%      y: transformed mean
%      Y: transformed smapling points
%      P: transformed covariance
%      Y1: transformed deviations

L=size(X,2);
y=zeros(n,1);
Y=zeros(n,L);
for k=1:L
    Y(:,k)=f(X(:,k));
    y=y+Wm(k)*Y(:,k);
end
Y1=Y-y(:,ones(1,L));
P=Y1*diag(Wc)*Y1'+R;

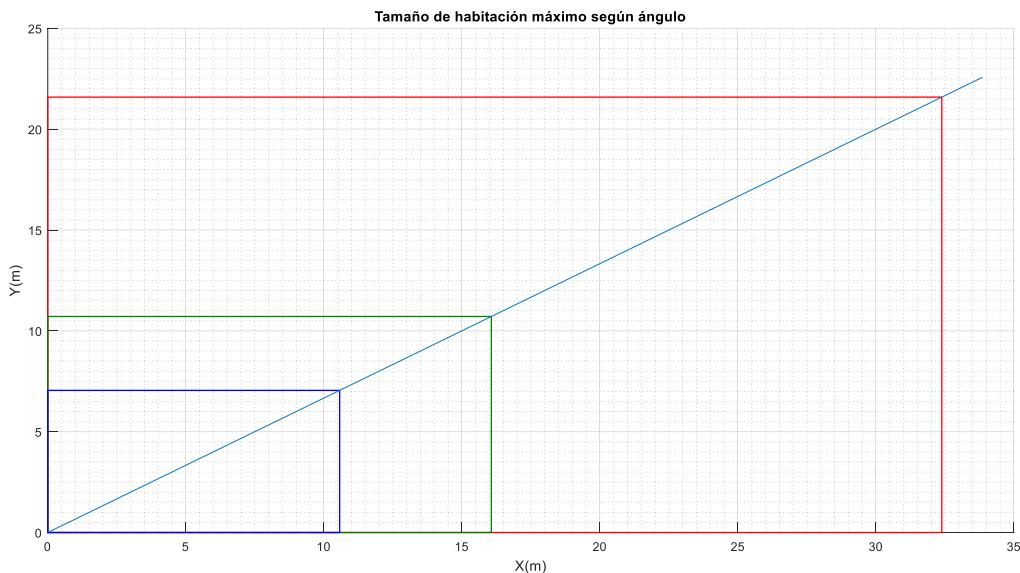
function X=sigmas(x,P,c)
%Sigma points around reference point
%Inputs:
%      x: reference point
%      P: covariance
%      c: coefficient
%Output:
%      X: Sigma points

A = c*chol(P)';
Y = x(:,ones(1,numel(x)));
X = [x Y+A Y-A];

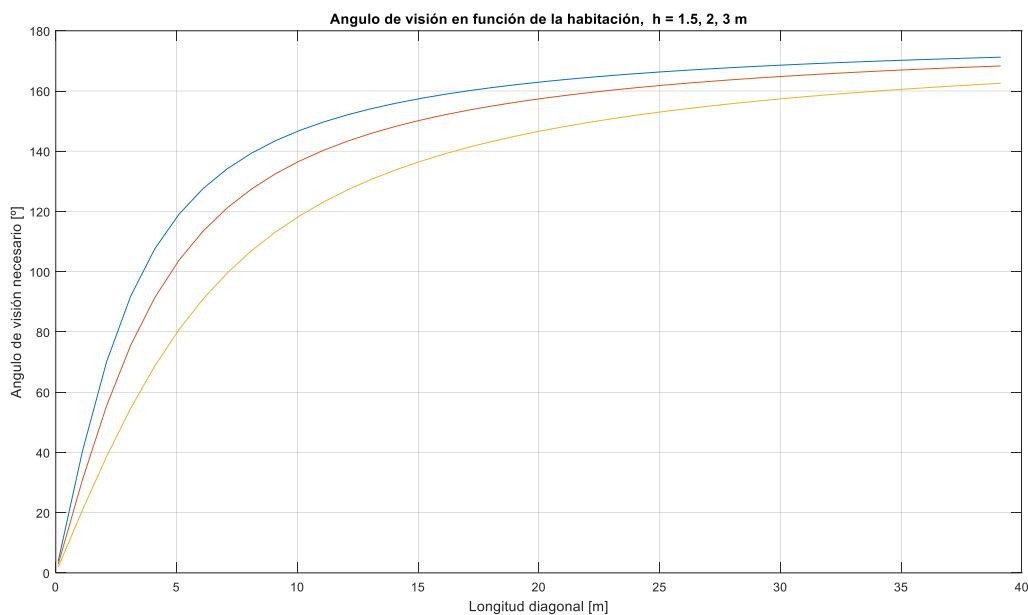
```

### Anexo N: Relación Ángulo visión superficie

Hemos elegido una altura de techo considerabelmente pequeña (1.5 metros). Y unos ángulos de cámara de  $170^\circ$  (en rojo),  $160^\circ$  (en verde) y  $150^\circ$  (en azul). Vemos como si verdaderamente tenemos unos  $170^\circ$  e diagonal podemos capturar con suerte un habitáculo bastante grande, aunque no lo bastante grande como un almacén.



Vemos como aumentar unos grados más supone aumentar bastante las dimensiones de la habitación, por lo que es recomendable que el ángulo de diagonal sea verdaderamente mayor a  $160^\circ$  por lo menos. En la siguiente figura, vemos la longitud mayor en función del ángulo.



Si aumentamos unos grados el ángulo de visión al principio el aumento de la diagonal es lineal, posteriormente pasa a exponencial, hasta los deseables  $180^\circ$ , en el que podemos ver todo el techo siempre que no haya interferencias.

## Anexo N: Parámetros y errores triciclo

- Velocidad constante:  $V = 2 \text{ m/s}$ .
- La orientación de la rueda delantera es constante:  $\theta = 10^\circ$ .
- La distancia  $b = 0,8 \text{ m}$ .
- La desviación típica de la posición es  $0,05 \text{ m}$ .
- La desviación típica de la orientación es  $1^\circ$ .
- El periodo de muestreo es:  $\Delta T = 0,1 \text{ s}$ .
- Posición inicial:  $x=0, y=0, \theta=0$ .

## Anexo O: Código completo SLAM

Omitiremos la parte final en la que se extraen mas gráficos.

```
%% SLAM - DEFINITIVO
%
% BASADO EN LA PRACTICA 3 - ROBÓTICA AVANZADA
% METODO SLAM: EKF - UKF
% MODELO DEL VEHÍCULO: TRICICLO
% AUTOR: ALBERTO GONZALEZ ISORNA
% FECHA: 05/07/17
% PARAMETROS DEL ROBOT TRICICLO
% DISTANCIA ENTRE RUEDAS Y DIRECCION: 0.4      m
% ANGULO DE GIRO: 15      °
% VELOCIDAD ANGULAR: 2*pi/30 rad/s
% TASA DE MUESTREO (experimental): 0.5 Hz (sería mucho mas)
% PERIODO: 30s por vuelta
%% COMENTARIOS
% el periodo de muestreo experimental
% es de 2.0933 s pero la estimacion
% a esa velocidad es muy mala, por lo
% que la solucion que se ha adoptado
% es mientras no haya mas informacion
% suponer que los landmarks permanecen
% donde estan con un ruido de medicion

%% INICIALIZACION
close all;clc; clear;
nlandmark=4; % numero de landmarks
n=3+2*nlandmark; % orden del sistema
m=2; % orden de las medidas
vk=0.3; % velocidad (antendiendo a especif)
Ts=0.2093; % tiempo de muestreo Ts=s/vk=0.628/0.3
ak=deg2rad(15); % angulo de giro
b=0.4; % distancia entre ruedas
wx=0.05; % desviacion posicion x
wy=0.05; % desviacion posicion y
wp=degtorad(2); % desviacion angulo de giro
red=0.4; % factor auxiliar

Rg=b/tan(ak); % radio de giro

load datos_QRP_optimos_def1.mat;
Q=miQ;
R=miR;
Q(1,1)=Q(1,1);
Q(2,2)=Q(2,2);

P=miP;
P(1,1)=0.1*P(1,1);
P(2,2)=0.1*P(2,2);
P(3,3)=P(3,3);
```

```
% MODELO DEL SISTEMA CON TODAS LAS VARIABLES POSIBLES
f=@(x) [x(1)-Ts*vk*sin(x(3)); % x
          x(2)+Ts*vk*cos(x(3)); % y
          x(3)+Ts*vk*tan(ak)/b; % angulo
          x(4); % 11
          x(5);
          x(6); % 12
          x(7);
          x(8); % 13
          x(9);
          x(10); % 14
          x(11)];
```

```
% MODELO DE MEDIDA (DISTANCIA Y ANGULO AL LANDMARK)
p=1;
h=@(x) [sqrt((x(2+2*p)-x(1))^2+(x(3+2*p)-x(2))^2);
          (atan((x(3+2*p)-x(2))/(x(2+2*p)-x(1)))+x(3))];
```

```
%% INICIALMENTE NO TENEMOS NI IDEA DE DONDE ESTÁN LOS LANDMARKS
```

```
%% CALCULO INICIAL DE LOS LANDMARKS
yc = 334.0033;
xc = 586.9512;
Zp=3;
rm= load('roofmarks_090717.mat');
data= load('data.mat'); %% hay info que sobra, en verdad solo el modelo
[numf, ~] = size(rm.rmarks);
Mrm=rm.rmarks;
Mrm=[Mrm(:,2) Mrm(:,1)]; % para cam2world
M=cam2world(Mrm,data.data.model);
M2=cam2world([yc;xc],data.data.model);

MR=zeros(3,15);

for i=1:numf
    if(rm.rmarks(i,1)~= 0)
        t=-Zp/M(3,i);
        MR(:,i)=t*M(:,i);
    else
        M(:,i)=0;
        MR(:,i)=0;
    end
end

t=-Zp/M2(3,1);
MRC(:,1)=t*M2(:,1);

MF =[MR(2,:) ' MR(1,:)' ];

%
%%
x0=1.5;      y0=0.1;      ang0=0.8*ak;
x11=2;        y11=5;
x12=-2;       y12=5;
x13=-2;       y13=-5;
x14=2;        y14=-5;

xl=[x11 x12 x13 x14];
yl=[y11 y12 y13 y14];
M=[xl;yl];
M=M(:);
s=[x0;y0;ang0;M]; % initial state
x=s+Q*randn(n,1); % initial state with noise
```

```
N=150;
xV = zeros(n,N); % estimate
sV = zeros(n,N); % actual
sF = zeros(n,N); % actual
sZ = zeros(nlandmark,N); % total dynamic steps
% allocate memory
```

```

zV = zeros(m,N);t=s;

u=s;sU=sF;

useukf=1;

hp=figure;
plot(0,0);hold on; grid on;

maximize(hp);

tpause=0.003;
% tpause=0.55;
g1=zeros(1,4);
ge=g1;

title('EKF CON TRES LANDMARKS');

%% VECTOR DE LANDMARKS
% 2 1
% 3 4
% mirando al salón

vlandmark=[0 0 0 0];
minrm=10000;
l=-1;
a=0;
cont=0;

%% BUCLE DEL ALGORITMO EKF - SLAM
% -----
for k=1:N
    %% TOMAMOS LA IMAGEN Y EXTRAEMOS EL ROOFMARK
    % si estamos en el punto correcto
    if (k-1==cont*10), cont=cont+1; condicion=1;
    else condicion=0; end;

    vlandmark=[0 0 0 0];

    if (condicion)
        Prm=MF(cont,:); % cogemos el punto de la matriz
        if (Prm(1)~=0 & Prm(2)~=0) % si el punto es válido
            a=a+1;
            px=(Prm(1));py=(Prm(2));

            % Calculamos el punto en relativo
            r = norm([px,py]);
            theta = atan2(py,px);
            % desgirar t(3)
            alfa=-x(3); % pi-x(3)
            % Matriz de giro
            M=[cos(alfa) -sin(alfa); sin(alfa) cos(alfa)];
            P2=M*[px;-py]; % punto girado
            xr=P2(1)+x(1); % punto en coordenadas reales
            yr=-P2(2)+x(2);

            % Calculamos el cuadrante
            l = calcucuadrante(xr,yr);

            if (a==8), l=3; end; % corrección
            % explicar las diferencia entre corregirlo o no

            % aqui ya sabemos a que landmark se corresponde
            vlandmark(l)=1; % lo ponemos a uno
            disp('Nlandmark = ');
            disp(l);
        end
    end

    %%
```

```

t= f(t);
sF(:,k)= t;

%% PLOTS
if (condicion), plot(x(1),x(2), '--rs'); % pintamos en rojo las fotos
else plot(x(1),x(2), '*b'); end; % pintamos en azul

plot(t(1),t(2), '.g');

for p=1:nlandmark
    if(useukf==0) % este si empleamos ekf
        h=@(x) [sqrt((x(2+2*p)-x(1))^2+(x(3+2*p)-x(2))^2);
                  atan((x(3+2*p)-x(2))/(x(2+2*p)-x(1))+x(3))];
    else
        h=@(x) [sqrt((x(2+2*p)-x(1))^2+(x(3+2*p)-x(2))^2);
                  atan2((x(3+2*p)-x(2)),(x(2+2*p)-x(1))+x(3))];
    end

    if (vlandmark(p)) % si corresponde ese landmark
        % calculamos la mededad
        z(1,1) = r;
        z(2,1) = pi-theta;
    else
        z = h(s) + R*randn(m,1); %measurments
    end

    sZ(p,k)=z(1); %distancia
    %z(2)=AngleWrapping(z(2));

    if (useukf==0)
        [x, P] = ekf(f,x,P,h,z,Q,R); % ekf
    else
        [x, P] = ukf(f,x,P,h,z,Q,R); % ukf
    end
end

s = f(s) + Q*randn(n,1); % update process

xV(:,k) = x; % save estimate
sV(:,k) = s; % save actual state
zV(:,k) = z; % save measurment
pV(:,:,k) = P; % save variance

% pintamos las elipses
for i=1:landmark
    j=2+2*i; % empezamos en 4
    sx=sqrt(P(j,j));
    sy=sqrt(P(j+1,j+1));
    ang=atan(sy/sx);
    xcp=x(j);
    ycp=x(j+1);
    g1(i)=plot(xcp,ycp,'+r');
    %g2(i)=ellipse(sx,sy,ang,xcp,ycp);
    ge(i)=ellipse(sx,sy,ang,xcp,ycp,'r');
end
shg;
pause(tpause);

% borramos el centro anterior
for i=1:nlandmark
    %set(hg(i),'Visible','off');
    set(g1(i),'Visible','off');
    set(ge(i),'Color','blue');
end
end

xlabel('X (m)');
ylabel('Y (m)');
% -----
%% FIN EKF - SLAM

```