Universidad de La Laguna

ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA

Grado en Ingeniería Electrónica Industrial y Automática

Trabajo Fin de Grado

Implementación y evaluación de un entorno experimental para localización de robots móviles usando filtros de Kalman

Autor : Iván Rodríguez Méndez

Tutor : Leopoldo Acosta Sánchez

Co–tutor : Antonio Luis Morell González

Junio, 2016

D. Leopoldo Acosta Sánchez, con N.I.F. 42165911B , catedrático adscrito al Departamento de Ingeniería de Sistemas y Automática de la Universidad de La Laguna.

CERTIFICA:

Que la presente memoria titulada: "Implementación y evaluación de un entorno experimental para localización de robots móviles usando filtros de Kalman" ha sido realizada bajo su dirección por D. Iván Rodríguez Méndez, con N.I.F. 43380782E.

Con esta fecha, autorizo la presentación de la misma.

En La Laguna, a 8 de junio de 2016.

El Director,

Leopoldo Acosta Sánchez.

Agradecimientos

A las primeras personas a las que se lo quiero agradecer son mis dos tutores Antonio Morell y Leopoldo Acosta, que sin su ayuda y sus conocimientos nada de esto hubiera sido posible. Agradezco mucho a Antonio el apoyo que me ha dado durante todo el tiempo en el que he estado compartiendo con él mi estancia en el laboratorio. Gracias por haberme enseñado que nunca es suficiente y que siempre se puede mejorar aunque sea en un punto o una coma, porque nunca hay que conformarse si aún está en nuestra mano realizar un mejor trabajo, gracias por hacerme entender esto aún restando tiempo de tus propias labores. Doy gracias también a Leopoldo ya que me ha enseñado que todo se puede lograr y que sólo depende de la actitud con la que afrontemos los problemas para poder superarlos y que formen parte del recuerdo, estoy seguro de que él sabrá entenderme. Gracias por hacer que surgiera en mi la inquietud de querer saber más sobre robótica y por poco a poco introducirme en este mundo. Muchas gracias por ofrecerme este proyecto cuando acudí a ti. Doy gracias también a mis compañeros de laboratorio con los que tanto tiempo he compartido gracias a una beca de colaboración. Gracias en especial a Fran, Yeray y Pablo por sus consejos y los momentos que hemos pasado juntos entre esas cuatro paredes.

Doy las gracias a mis padres también por haberme proporcionado la mejor educación y procurar siempre que mis valores fueran la humildad y el trabajo duro. Gracias por todas esas lecciones de vida que me han dado a lo largo de estos 21 años. Doy Gracias a mi padre, por haberme enseñado que no importa de donde vengas ya que con esfuerzo, trabajo y constancia todo se consigue, que en esta vida nadie te regala nada pero que tienes la posibilidad de conseguir lo que te propongas. Gracias por enseñarme que nunca hay dar nada por hecho y que siempre hay que probar cualquier solución a un problema por absurda que pueda parecer. Muchísimas gracias a mi madre, por aguantarme cada día y siempre procurar hacernos la vida más fácil a mi hermana y a mi. Gracias por enseñarme que hay que afrontar las situaciones difícil con calma, por hacerme confiar en las decisiones que tomo y sobre todo por procurar que siempre haga lo que haga sea con humildad y buena educación. Gracias también a mi hermana por su apoyo, ella es otro ejemplo de que con trabajo y esfuerzo se puede llegar a donde se quiera, estoy seguro de que llegará lejos. Gracias también al resto de mi familia, por estar siempre unidos. Muchas gracias a mi pareja, gracias por aguantar mi carácter en los malos momentos y aún así estar ahí para ser mi apoyo incondicional. Gracias por soportar esas tardes de sábado en las que no podíamos hacer nada interesante debido a que tenía que seguir trabajando en el proyecto. Agradezco que me hayas enseñado

que el mundo es de un color diferente si lo afrontas con una sonrisa, gracias por hacerme mejor persona. Te debo todos esos ánimos que me dabas para empezar cada día con energías renovadas y con ganas de comerme el mundo. Muchísimas gracias a mis amigos y compañeros. Gracias Nico, Carlos, Antonio y Aythami por estar ahí siempre, he aprendido grandes lecciones gracias a ustedes.

Finalmente, gracias a ti abuela, o mejor dicho *Mamá* como te gustaba que te llamaran. Gracias porque allá donde estés me has dado la fuerza para seguir, gracias por esas lecciones que me diste en tu último año con nosotros. Gracias a ti he aprendido a no darme por vencido.

Resumen

Uno de los problemas que más esfuerzos en investigación han generado en el ámbito de la robótica móvil, es sin duda la estimación de estados, aplicada en concreto a la localización. En este proyecto se pretenden estudiar una serie de herramientas con una sólida base estadística, llamadas filtros Bayesianos. Dentro de este tipo de filtros se estudiarán lo que se conocen como filtros de Kalman. En concreto estudiaremos cuatro tipos de filtros de Kalman, formulados para aplicarse a diferentes situaciones. Los filtros que estudiaremos a lo largo del trabajo serán: el filtro de Kalman clásico, el filtro de Kalman extendido, el filtro de Kalman *unscented* y el filtro de Kalman de Cubatura. Además, estudiaremos la implementación de estos en MATLAB para comprobar cual es el que presenta una mejor estimación para la localización de un robot móvil. Para determinar cual es el mejor de los filtros diseñaremos una serie de experimentos que nos ayudarán a detectar los puntos débiles y el que presente un mejor funcionamiento. Para la implementación de los experimentos utilizaremos V-REP, un programa que nos permite simular dinámicas reales para múltiples robots, sensores, actuadores e interacciones con diferentes entornos. Finalmente, con ayuda de esta herramienta obtendremos conclusiones sobre cuál es el filtro que presenta un mejor rendimiento, los puntos débiles, y evaluaremos si esto guarda relación con lo estudiado en los capítulos en los que hemos realizado la explicación teórica de cada uno.

Abstract

Dynamic state estimation is one of the problem on mobile robotics that has received a lot of research efforts lately, especially on applications related with localization. The aim of this project is to study a series of methods with solid foundations on statistics, called Bayesian filters. Within this type of filters, we will study those known as Kalman filters. In particular we will consider four types of filters, each of them designed to deal with particular situations. The algorithms assessed in this project will be: the classic Kalman filter, the extended Kalman filter, the unscented Kalman filter and the cubature Kalman filter. In addition, we will study a MATLAB implementation to find which one yields better estimates on the localization of a mobile robot. To determine which is the best filter we will design a series of experiments that will help us to discover the weak points of each one and which gives the best performance. For the implementation of the experiments we will use V-REP, a program that allows us to simulate real robots and their dynamics, sensors, actuators and their interactions with different environments. Finally, with the help of this tool, we will obtain important conclusions about which is the best filter and assess if this is related with the theoretical presentation made in the following chapters.

Índice

Capítulo 1. Introducción **1**
1.1. Objetivos del proyecto 1
1.2. ¿Qué es la robótica? 1
1.3. Incertidumbre en sistemas reales 2
1.4. Robótica probabilística 5
1.5. Ventajas y desventajas del enfoque probabilístico 8
1.6. Organización del documento 9

Capítulo 2. Introducción al filtro de Kalman **11**
2.1. ¿Qué es el filtro de Kalman? 11
2.2. Robot, entorno e interacción 13
2.3. Filtros Bayesianos 19
2.4. Modelo de espacio-estado 22

Capítulo 3. Filtros de Kalman **25**
3.1. Filtro clásico de Kalman (KF) 26
3.2. Filtro extendido de Kalman (EKF) 30
3.3. Filtro *unscented* de Kalman (UKF) 33
3.4. Filtro de Kalman de Cubatura (CKF) 38

Capítulo 4. Framework experimental **43**
4.1. Plataforma experimental 43
4.2. Modelado 3D usando Blender 45
4.3. Simulación dinámica con V-REP 46
4.4. Control de simulación mediante MATLAB 48

Capítulo 5. Experimentación y discusión **51**
5.1. Equipos de trabajo y sistemas 51
5.2. Estructura básica de los experimentos. 52

5.3. Descripción de los experimentos propuestos 65
5.4. Experimento 1: Odometría 68
5.5. Experimento 2: Ruido en las medidas 75
5.6. Experimentos 3: Variación del número de balizas 80
5.7. Experimento 4: Rendimiento ante una trayectoria general 85
5.8. Resultados generales 87

Conclusions and future work **89**
Future work 90

Apéndice A. Conceptos matemáticos para comprender el filtro de Kalman. **93**
A.1. Probabilidad de un evento. 93
A.2. Variables Aleatorias. 95
A.3. Media y varianza. 96
A.4. Distribución de probabilidad Normal o Gaussiana 98
A.5. Independencia y Probabilidad condicional 100

Apéndice B. Toolbox utilizada en Matlab. **103**
B.1. Filtro clásico de Kalman 103
B.2. Filtro extendido de Kalman de primer orden 106
B.3. Filtro *unscented* de Kalman *NonAugmented* 110
B.4. Filtro de Kalman de Cubatura 113

Apéndice C. Funciones implementadas para la API con V-REP. **117**
C.1. Abrir y cerrar conexión con V-REP 117
C.2. Estado de la simulación 118
C.3. Creación de objetos,modelos y escenas 119
C.4. Mensajes por pantalla e interfaz 120
C.5. Propiedades de los objetos 122
C.6. Funciones para el Robot 124
C.7. Funciones experimentales de los filtros 137

Bibliografía **175**

Índice de tablas

5.1. Experimentos : Odometría 66
5.2. Experimentos : Ruido sensores 66
5.3. Experimento: Variar el número de balizas 67
5.4. Experimento: Mejores condiciones de funcionamiento 68
5.5. Experimentos trayectoria general 85
5.6. Puntuaciones obtenidas por los filtros 88

Índice de figuras

1.1. Diagrama de localización de Markov [1]. 6

3.1. Iteración del KF [2][3]. 28
3.2. Funcionamiento del KF [1]. 29
3.3. Iteración del EKF [2] [3]. 33
3.4. Comparativa linealización y transformada UT [2] [4] 34
3.5. Iteración UKF [5] [6] 38
3.6. Iteración CKF [7]. 42

4.1. Pioneer P3dx 44
4.2. Pioneer P3dx disponible en el GRULL. 44
4.3. Modelo de ruido sensor. 45
4.4. Pioneer P3dx en Blender. 46
4.5. Simulación en V-REP. 47
4.6. Interfaz de Matlab. 48

5.1. Escena trayectoria recta V-REP 53
5.2. Escena trayectoria cuadrada V-REP 54
5.3. Escena trayectoria senoidal V-REP 55
5.4. Escena trayectoria arbitraria V-REP 55
5.5. Modelo del robot real en V-REP 56
5.6. Modelo del robot estimado en V-REP 56
5.7. Experimento recta con odometría 69
5.8. Experimento recta sin odometría 70
5.9. Trayectoria recta con odometría 70
5.10. Trayectoria recta sin odometría 71
5.11. Experimento cuadrado con odometría 71
5.12. Experimento cuadrado sin odometría 72
5.13. Trayectoria cuadrada con odometría 72

5.14. Trayectoria cuadrada sin odometría 73
5.15. Experimento seno con odometría 73
5.16. Experimento seno sin odometría 74
5.17. Trayectoria senoidal con odometría 74
5.18. Trayectoria senoidal sin odometría 75
5.19. Experimento recta función de medida de distancia 76
5.20. Experimento recta función de medida de ángulos 76
5.21. Trayectoria recta función de medida de distancia 77
5.22. Trayectoria recta función de medida de ángulos 77
5.23. Experimento cuadrado función de medida de distancia 78
5.24. Experimento cuadrado función de medida de ángulos 78
5.25. Trayectoria cuadrada función de medida de distancia 79
5.26. Trayectoria cuadrada función de medida de ángulos 79
5.27. Experimento seno función de medida de distancia 80
5.28. Experimento seno función de medida de ángulos 80
5.29. Trayectoria senoidal función de medida de distancia 81
5.30. Trayectoria senoidal función de medida de ángulos 81
5.31. Experimento recta variando número de balizas 82
5.32. Trayectoria recta 3 balizas 82
5.33. Experimento cuadrado variando número de balizas 83
5.34. Trayectoria cuadrada 3 balizas 83
5.35. Experimento seno variando número de balizas 84
5.36. Trayectoria senoidal 3 balizas 84
5.37. Experimento trayectoria general EKF 85
5.38. Experimento trayectoria general UKF 86
5.39. Experimento trayectoria general CKF 86

A.1. Diagrama de Venn de A unión B. 94
A.2. Diagrama de Venn de A intersección B. 94
A.3. Función de distribución de probabilidad Normal o Gaussiana. 99

Índice de algoritmos

2.1. Algoritmo filtro de Kalman 13
2.2. Algoritmo de filtros Bayesianos [1] $(bel(x_{t-1}), u_t, z_t)$ 19

3.1. Algoritmo KF [1] $(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$ 28
3.2. Algoritmo EKF [1] $(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$ 33
3.3. Algoritmo UKF [5] [6] 39
3.4. Algoritmo CKF [7] 41

5.1. Algoritmo experimentos 58

Índice de abreviaturas

KF Filtro de Kalman clásico 1
UKF Filtro de Kalman unscented 1
EKF Filtro de Kalman extendido 1
CKF Filtro de Kalman de cubatura 1

Símbolos y notación

a	escalar
$\lvert a \rvert$	valor absoluto de a
$\boldsymbol{a}$	vector columna
a_i	i-ésimo elemento del vector $\boldsymbol{a}$ o del conjunto de escalares $\{a_n\}$
$\mathbf{1}$	vector columna $[1, 1, \ldots, 1]^T$
$\lVert \boldsymbol{a} \rVert$	norma euclídea de $\boldsymbol{a}$
$\boldsymbol{a}^T$	traspuesta de $\boldsymbol{a}$
$\mathbf{A}$	matriz
$\boldsymbol{a}_i$	i-ésimo elemento de conjunto de vectores $\{\boldsymbol{a}_n\}$ o i-ésima columna de la matriz $\mathbf{A}$
a_{ij}	elemento en la fila i-ésima y columna j-ésima de la matriz $\mathbf{A}$
$\mathbf{I}$	matriz identidad
$\mathbf{A}^{-1}$	inversa de la matriz $\mathbf{A}$
$\lVert \mathbf{A} \rVert_F$	norma de Frobenius de la matriz $\mathbf{A}$
$\text{diag}(\boldsymbol{a})$	matriz diagonal cuyo i-ésimo elemento de la diagonal es a_i
$\det(\mathbf{A})$	determinante de la matriz $\mathbf{A}$
$\text{tr}(\mathbf{A})$	traza de la matriz $\mathbf{A}$
$[\ldots]$	vector o matriz
$\{\ldots\}$	conjunto o lista de elementos
X, Y, Z	ejes de coordenadas en $\mathbb{R}^3$
x, y	ejes de coordenadas en $\mathbb{R}^2$
$\mathbb{F}$	espacio de características
$f(x)$	función f en x
$f(x; p)$	función f en x con parámetro p
$\hat{a}$	estimación de a

$\langle a \rangle$	media del conjunto $\{a_n\}$
$\tilde{a}_i$	i-ésimo elemento del conjunto $\{a_n\}$ al que se le ha restado la media de dicho conjunto
$a^{(t)}$	valor de a en la iteración t
$\mathrm{var}(a)$	varianza de a
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	distribución normal multivariable de media $\boldsymbol{\mu}$ y varianza $\boldsymbol{\Sigma}$
$D_{KL}(\mathcal{P} \| \mathcal{Q})$	divergencia de Kullback-Liebler entre las distribuciones de probabilidad $\mathcal{P}$ y $\mathcal{Q}$

Capítulo 1

Introducción

1.1. Objetivos del proyecto

Los objetivos principales de este proyecto son los siguientes:

- Estudio de los distintos filtros de Kalman y los principios básicos de funcionamiento de cada uno de estos para la localización de un robot móvil en suelo deslizante. Los filtros a estudiar serán los siguientes:
  - Filtro de Kalman clásico (KF)
  - Filtro de Kalman extendido (EKF)
  - Filtro de Kalman unscented (UKF)
  - Filtro de Kalman de cubatura (CKF)
- Implementación en *MATLAB* de simulaciones para comparar el desempeño de cada uno de los filtros en distintas trayectorias y condiciones. Para la implementación de las simulaciones usaremos una *toolbox* [8] que implementa todos los filtros necesarios dentro de *MATLAB*.
- Simulación en un entorno 3D por medio de una API con el software *V-REP* de la implementación de los filtros en un robot móvil diferencial.
- Obtención de conclusiones usando el marco experimental desarrollado por medio de experimentos en las distintas plataformas sobre qué filtro presenta una mejor estimación de la localización del robot.

1.2. ¿Qué es la robótica?

El término robot fue inventado en 1921 por el novelísta Checo Karel Capek [9], para describir unas máquinas inteligentes parecidas a los humanos que realizaban trabajos

que estos no podían hacer o que bien no les gustaba hacer. En los años 40, Isaac Asimov acuñó el término robótica y enunció las famosas 3 leyes de la robótica, concretamente en su obra *I, Robot* [10]. Las tres leyes son las siguientes [11]:

1. Un robot no hará daño a un ser humano o, por inacción, permitir que un ser humano sufra daño.

2. Un robot debe obedecer las órdenes dadas por los seres humanos, excepto si estas órdenes entrasen en conflicto con la 1ª ley.

3. Un robot debe proteger su propia existencia en la medida en que esta protección no entre en conflicto con la 1ª o la 2ª ley.

Desde entonces la robótica se ha convertido en una importante disciplina científica sobre la que se ha investigado durante décadas. Aunque los mayores avances en este campo se han dado a partir de los años 70 en adelante, al igual que muchas otras ramas de la ciencia, la robótica ha evolucionado casi al mismo tiempo que lo han hecho los sistemas computacionales.

1.3. Incertidumbre en sistemas reales

Podemos definir la robótica como la ciencia relacionada con la percepción e interacción con el entorno por medio del control de dispositivos mecánicos gracias a un ordenador o algún tipo de sistema computacional [1]. Algunos ejemplos de éxito relacionado con los sistemas robóticos podrían ser las plataformas de exploración planetaria, brazos robóticos en cadenas de montaje, navegación autónoma de coches en autopistas o por ejemplo drones de rescate para facilitar a los equipos de búsqueda el desempeño de su labor. La realidad es que la mayoría de sistemas robóticos se encuentran aún en su *infancia*, sin embargo la idea de dotar de "inteligencia" a los sistemas robóticos tiene un enorme potencial para cambiar la sociedad. Ideas como evitar los accidentes de tráfico haciendo que los vehículos se comuniquen entre ellos o que incluso lleguen a llevar a las personas de un lugar a otro sin más intervención del factor humano más que para definir el destino al que queremos llegar. Por otro lado también podríamos disponer de sistemas domésticos que se encargaran de las arduas tareas del hogar. Algunos de estos sistemas están bastante extendidos en el mercado, como en el caso del Roomba , un robot capaz de aspirar nuestro hogar localizándose de manera autónoma dentro de este.

Las aplicaciones de los robots actuales son distintas a los robots del pasado, como por ejemplo, los manipuladores disponibles en las cadenas de montaje que resuelven el mismo problema de forma repetitiva día tras día (típicos de las primeras aplicaciones de la robótica) o en la actualidad aquellos que son capaces de actuar según la variación del propio entorno. La característica más relevante de los nuevos sistemas robóticos es que cada vez son más capaces de trabajar en entornos no estructurados, es decir entornos impredecibles donde no sabemos con qué puede encontrarse el robot. Para ilustrar lo anterior con un ejemplo, podríamos decir que un entorno industrial como una cadena de montaje es bastante más predecible y controlable que un entorno en el interior de una casa. Como resultado, debemos dotar a nuestro robot de un sistema sensorial que sea capaz de interaccionar de alguna manera con el entorno en el que se encuentre y además debe contar con un software lo suficientemente robusto para tomar decisiones de mayor o menor complejidad con diferentes situaciones que se le puedan presentar a este, para que sea capaz de anticiparse a situaciones cambiantes en entornos dinámicos. Por esta razón con el paso de los años la robótica se ha ido transformando y el software se ha convertido en el aspecto más importante de esta rama de la ingeniería. Por ello desarrollar un software robusto y que permita a un robot enfrentarse a entornos dinámicos es un tópico donde actualmente se están invirtiendo grandes esfuerzos en investigación.

El punto central de este trabajo no es otro que la **estimación de estados con incertidumbre**. Este será un punto clave ya que necesitamos saber donde se encuentra el robot en cada instante y para nosotros no será lo mismo que el robot se encuentre a 1 metro del objetivo que a 5 metros. Es obvio que cuanta más certeza tengamos en las estimaciones, menos incertidumbre encontraremos sobre las variables que queremos conocer y mejor será el funcionamiento de nuestro sistema. Para tratar este problema tenemos una serie de aspectos que influyen en el rendimiento de un estimador de estados, son los siguientes:

1. **Entorno**: Es lógico pensar que el mundo, es decir el entorno, es un lugar impredecible donde no podemos saber que se encontrará el robot. A pesar de lo anterior pueden llegar a existir entornos donde esa incertidumbre esté controlada, como los entornos industriales, o por el contrario donde muchos factores son impredecibles como podrían ser las carreteras o los hogares.

2. **Sensores:** Cuentan con la limitación con respecto a la información que pueden percibir de su entorno. Las limitaciones derivan de dos factores principales, el primero de ellos está relacionado con el rango y la resolución de estos lo cual está

sujeto a variables físicas. Un ejemplo claro es que las cámaras no pueden ver a través de una pared, incluso cuando no tienen ningún elemento que dificulte su percepción, la propia resolución espacial de la cámara está limitada. En segundo lugar, los sensores están siempre sujetos a ruidos que perturban las medidas de forma aleatoria (más adelante veremos como podemos modelar el ruido para su compensación) lo cual limita en gran medida la información que somos capaces de obtener de ellos.

3. **Actuadores**: Las acciones desarrolladas por los robots se dan en su mayoría por medio de motores y la gran parte de ellos pueden introducir efectos impredecibles como el control del ruido introducido en la consiga o incluso el desgaste de rodamientos y uniones (lo que provoca un comportamiento no deseado). Sin embargo en el mercado existen muchos tipos de actuadores, algunos como los de los robots industriales para trabajos pesados son precisos en cuanto a sus movimientos, por otra parte los actuadores con los que cuentan los robots de bajo coste son bastante imprecisos. De esta manera la construcción y las características de nuestro robot también podrían influir en la incertidumbre que tengamos.

4. **Modelos**: Los modelos son imprecisos, este hecho es inevitable. No debemos pasar por alto que los modelos no son más que meras abstracciones del mundo real. Como tal estos solo caracterizan el proceso físico que tiene lugar entre el robot y su entorno. Los errores en el modelo son otra fuente de incertidumbre, y además es una de las cuestiones que más se ha pasado por alto en robótica a pesar de que la mayoría de modelos existentes en el estado del arte son en su mayoría muy simplificados.

5. **Computación**: Los robots son sistemas de tiempo real y por lo tanto están limitados con respecto a la cantidad de recursos computacionales disponibles para ejecutar órdenes en un instante de tiempo (siempre y cuando nos estemos refiriendo a robots que interactúan en tiempo real con el entorno para adaptarse a los cambios ocurridos en él). La mayoría de algoritmos disponibles con mejor rendimiento (incluido el filtro de Kalman, que estudiaremos más adelante) parten de ciertas simplificaciones para ahorrar recursos computacionales, lo cual empeora la precisión de nuestro sistema.

Todos estos factores pueden hacer que la incertidumbre aumente. Tradicionalmente no se tenían en cuenta y por lo tanto la incertidumbre era un tema que se solía ignorar ya que el robot actuaba en espacios controlados. La tendencia de cambio en el estudio

de la incertidumbre se debe a que cada vez más se necesita que los robots actúen en entornos no estructurados y por lo tanto poder lidiar con ella hará que nuestro robot pueda funcionar de forma correcta.

1.4. Robótica probabilística

Durante el desarrollo de este trabajo trataremos de resolver la localización de nuestro robot por medio de algoritmos que en su fundamentación tienen una base probabilística. La probabilidad en la robótica es un enfoque que nos permite saber cuantificar de alguna forma el grado de incertidumbre tenemos en la percepción de nuestro robot y en sus acciones. Por lo tanto la idea principal de la aplicación de la probabilidad es tener explícitamente datos que nos indiquen cual es la incertidumbre, para ello usamos los cálculos de probabilidades. Otra diferencia la encontramos a la hora de realizar medidas, en lugar de confiar totalmente en la medida que nos devuelve nuestro sensor, lo que harían los algoritmos probabilísticos sería representar esa información por medio de una distribución de probabilidad lo cual nos daría una idea de la incertidumbre sobre nuestra medida. Lo anterior se traduce en que tendremos mayor conocimiento sobre la verosimilitud de nuestras medidas aplicando estos algoritmos. Como ya veremos más adelante la aplicación de algoritmos basados en probabilidades solucionan muchos problemas relacionados con las fuentes de incertidumbre que enumerábamos en el apartado anterior.

Para ilustrar mejor el enfoque probabilístico usaremos un sencillo y visual ejemplo: la localización de un robot móvil [1], que además es el tema en el que basamos este trabajo. La localización es el problema de estimar las coordenadas de un robot con respecto a un sistema de coordenadas de referencia con ayuda de los sensores y usando un mapa del entorno en el que se encuentra. Las coordenadas de las que hablamos en nuestro caso corresponden al vector de estados del robot que queremos estimar, las variables especificadas dentro del vector siguiente:

$$p = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{1.1}$$

Donde, x corresponde a la localización en dicho eje con respecto al sistema de referencia en metros o la unidad de medida longitudinal que decidamos usar; y sería lo mismo

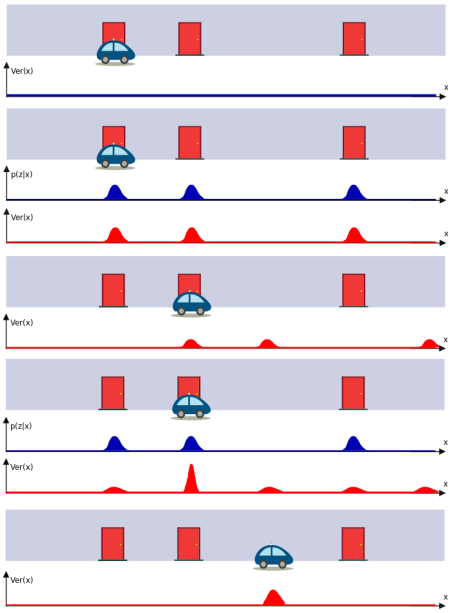

Figura 1.1. Diagrama de localización de Markov [1].

que x pero refiriéndose a su propio eje. Por último θ se correspondería a la rotación de nuestro robot con respecto a una referencia donde fijaríamos el cero. La unidades son radianes.

En la figura 1.1 podemos ver una ilustración que plasma el enfoque de la localización probabilística para un robot móvil. El problema de la localización para este ejemplo se conoce como localización global donde el robot se encuentra en cualquier punto del entorno y debe localizarse a si mismo desde cero (sin ningún conocimiento previo de donde puede estar). Como ya hemos comentado en este tipo de localización la estimación en un instante de tiempo (también conocida como certidumbre) se representa por medio de una función de densidad de probabilidad que se encuentra distribuida en el espacio en el que realizamos la localización (un entorno tridimensional o bidimensional). Lo anterior se ilustra en el primer fragmento de la figura 1.1 donde se muestra una distribución uniforme (a priori) lo cual corresponde a la máxima incertidumbre (coincidiendo con la localización inicial). Suponemos entonces que el robot toma una primera medida y el resultado de esta nos dice que el robot se encuentra cerca de una puerta. La verosimilitud resultante se muestra en el segundo fragmento de la figura 1.1 donde podemos observar que los puntos de máxima probabilidad se colocan donde se encuentran las puertas (ya que nuestra medida nos indica que es más probable que estemos en alguno de estos puntos) y en cualquier otro punto tenemos una probabi-

lidad más baja. Como podemos ver esta distribución presenta tres picos, cada uno de estos picos corresponden a cada puerta de manera indistinguible por lo que se asigna la misma probabilidad de encontrarse en cualquiera de estos tres lugares. Además, la distribución de probabilidad asigna alta probabilidad a tres diferentes localizaciones, ilustrando de esta manera que el marco de referencia probabilístico puede tener en cuenta distintas hipótesis que podrían llegar a entrar en conflicto y generar situaciones ambiguas. Finalmente, incluso en los lugares donde no está localizada una puerta tenemos una probabilidad distinta de cero. Lo anterior es debido a la incertidumbre que tenemos al realizar las medidas, por lo tanto si tenemos una probabilidad pequeña pero distinta de cero tenemos en cuenta la posibilidad de que el robot esté errando y por lo tanto no esté cercano a una puerta.

Ahora supongamos que el robot se mueve hacia la derecha. El tercer fragmento de la figura 1.1 muestra lo que le ocurre a la verosimilitud del robot cuando este se mueve, siempre y cuando el movimiento del robot se de tal y como asumimos. Vemos que la verosimilitud se desplaza en la misma medida que el robot, es decir, que se desplaza en la dirección de movimiento de este. También podemos observar como los picos se han suavizado debido a la incertidumbre existente en relación al movimiento del robot, ya que a medida que nos movemos sin volver a medir más estamos aumentando la incertidumbre sobre donde nos encontramos. Para finalizar, en el cuarto fragmento de la figura 1.1 está presentada la verosimilitud después de realizar la medida y darnos cuenta de que el robot se ha encontrado cerca de otra puerta. Esta observación permite a nuestro algoritmo asignar altas probabilidades de que el robot se encuentre cerca de una de las puertas por lo tanto las probabilidades para cada una de ellas no serán las mismas. Debido a esta segunda medida el robot se ha reducido la incertidumbre en su localización con respecto al momento antes de realizar la medida.

Como hemos visto este ejemplo ilustra el paradigma probabilístico en el contexto de aplicación en un problema concreto. El problema de la percepción del robot es un problema en cuanto a la estimación de los estados, y en el ejemplo que hemos mostrado se usa una herramienta conocida en probabilidad como **filtro bayesiano** o **teorema de bayes** para la estimación posterior del estado de la localización del robot en el espacio en el que se encuentra. Hablaremos sobre este teorema en posteriores apartados (Capítulo 2 y Apéndice A).

1.5. Ventajas y desventajas del enfoque probabilístico

Llegados a este punto la pregunta más importante probablemente sea cuales son las ventajas de un enfoque probabilístico en comparación a otras aproximaciones que no representan la incertidumbre de forma explícita. La respuesta a esta pregunta es una premisa general que sigue el enfoque que trataremos en este trabajo, y es la siguiente: *Un robot que tiene conocimiento de su propia incertidumbre y actúa en consonancia con esto es superior a otro que no lo hace.*

En general los enfoques probabilísticos son más robustos de cara a las limitaciones que presentan los sensores, es decir, con el ruido que introducen a las medidas, cómo les afecta la dinámica del entorno y demás efectos que se pueden producir sobre estos. Su gran ventaja sobre otras aproximaciones se aprecia cuando se aplica en entornos no estructurados donde la robustez ante la incertidumbre es tremendamente importante. La realidad es que los algoritmos basados en probabilidades son los únicos que han conseguido trabajar de forma correcta en los problemas más complejos de estimación de estados en robótica. Un ejemplo muy típico para ilustrar lo anterior es el del *robot secuestrado* donde el robot por sus propios medios debe recuperarse de un fallo de localización. Por contra, los algoritmos basados en probabilidad tienen sus puntos débiles en la precisión de los modelos que utilizan muchos algoritmos clásicos y como ya dijimos esto puede llegar a ser una fuente de incertidumbre.

Sin embargo, parece que las ventajas aún son mayores que las desventajas que se presentan. Por lo general, las dos limitaciones más frecuentes citadas de estos algoritmos son la **ineficiencia computacional**, y que **necesitan basarse en aproximaciones**. Los algoritmos probabilísticos son en esencia menos eficientes que los que no lo son, debido a que en ellos se consideran densidades de probabilidad completas. Por lo tanto estos modelos necesitan ser aproximados ya que el robot se mueve en un espacio continuo y tratar con los datos de esta manera para un sistema computacional sería casi inviable. Muchas veces la solución se basa en que la incertidumbre puede ser aproximada por medio de un modelo paramétrico compacto (que en nuestro caso será una distribución de probabilidad Gaussiana) por norma general la solución analítica de los modelos suele ser bastante inmanejable o costosa computacionalmente, por lo que se utilizan métodos aproximados que generalmente resuelven un problema de optimización . La utilización de los modelos compactos y las aproximaciones permitirán utilizar estos algoritmos con una eficiencia computacional muy superior.

1.6. Organización del documento

En este trabajo trataremos de dar una idea general sobre la estimación probabilística y sobre los filtros de Kalman como ejemplo de aplicación. Iremos evolucionando desde una introducción matemática básica para comprender mejor las herramientas de las que disponemos hasta una serie de experimentos donde intentaremos ver que filtro presenta un mejor desempeño cuando lo aplicamos a un robot móvil con deslizamiento. La organización del documento es la siguiente:

- En el capítulo 2, haremos una introducción a los filtros de Kalman y a todos los conceptos básicos.
- En el capítulo 3, introduciremos cada uno de los filtros que estudiaremos en este trabajo además de explicar sus algoritmos de funcionamiento.
- En el capítulo 4, describiremos todas las herramientas y equipos utilizadas para la realización de este trabajo.
- En el capítulo 5, realizaremos los experimentos sobre los filtros y determinaremos cual es el que presenta el mejor comportamiento ante diferentes situaciones de experimentación.
- Finalizaremos con una conclusión global sobre el trabajo realizado y plantearemos posibles trabajos futuros en la misma línea.

Capítulo 2

Introducción al filtro de Kalman

2.1. ¿Qué es el filtro de Kalman?

Dentro de las herramientas matemáticas existentes para la estimación estocástica de medidas de sensores ruidosas, se encuentra la herramienta que estudiaremos a lo largo de este trabajo, la denominada filtro de Kalman. La importancia de utilizar este filtro radica en que se postula como el principal método para estimar los estados de sistemas dinámicos lineales representados de la forma espacio-estado. Los sistemas representados de esta manera tienen muchas aplicaciones de interés, como veremos más adelante nos ayudarán en gran medida para la realización de simulaciones por ordenador. Aunque esta herramienta tiene un desarrollo matemático complejo, llegar a entender su funcionamiento de forma conceptual no es difícil y de hecho es lo que plantearemos en este trabajo. El KF fue presentado en el año 1960, por el matemático Rudolf E.Kalman, en un documento que describía una solución recursiva para el problema del filtrado lineal de datos discretos, por el método de mínimos cuadrados. La definición clásica del filtro supone que el sistema debe ser lineal y además estar afectado por ruidos Gaussianos, es decir ruidos o perturbaciones que siguen una distribución Gaussiana de media cero (ruido blanco), para que pueda realizarse la estimación de una forma óptima. Desde el momento de la invención del algoritmo, debido en gran parte al avance de la informática, el filtro ha sido objeto de una extensa investigación y aplicación. A lo largo de los años han surgido algunas modificaciones que permiten aplicar el filtro en condiciones para las que originalmente no estaba diseñado, las más populares son el EKF (que puede aplicarse a sistemas no lineales), el UKF y por último el CKF. Como vemos el hecho de que el algoritmo original fuera presentado hace más de 50 años no ha evitado que actualmente esta herramienta siga usándose para una gran variedad de aplicaciones. Los sistemas para los que se aplican este tipo de filtros van desde la simulación de instrumentos musicales en un entorno de realidad virtual, hasta la extracción de secuencias de movimiento en vídeos o incluso su uso en sistemas de navegación

autónoma y asistida, el guiado de misiles, etc.

Con respecto al principio de funcionamiento del filtro, este se presenta como el mejor estimador posible para una amplia clase de problemas y un muy efectivo estimador para una cantidad aún mayor de aplicaciones como ya decíamos anteriormente. Es un algoritmo fácil de entender conceptualmente y con pocos requerimientos computacionales (algoritmo 2.1). Como decíamos se trata de un método iterativo que estima el estado no observable (aquel estado que no es posible medir por medio de sensores, lo cual es una situación muy común) de un sistema dinámico lineal dadas una serie de medidas que proporcionan información acerca de dicho estado en cada instante de tiempo. Estas ideas las introduciremos con más profundidad más adelante (sección 2.3). Por lo tanto el filtro funciona suponiendo que el sistema puede ser descrito a través de un modelo estocástico lineal, 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 una varianza determinada. Es muy importante para el correcto funcionamiento de la herramienta caracterizar bien el ruido que presenta nuestro sistema ya que mejorará notablemente el desempeño de este, lo que se traduce en una buena estimación. Otro aspecto relevante a tener en cuenta para comprender el funcionamiento del filtro es la representación del sistema en la forma espacio-estado. Esta representación es esencialmente una notación conveniente para la estimación de modelos estocásticos donde se asumen errores en la medición del sistema , lo que permite abordar el manejo de un amplio rango de modelos de series temporales. Representando el sistema de esta manera podemos modelar los ruidos como una simple suma dentro de la ecuación, aunque esto lo detallaremos en posteriores secciones. El ciclo de funcionamiento del filtro es un procedimiento que opera por medio de un mecanismo de dos partes fundamentales, un ciclo de predicción y un ciclo de corrección, también conocido como ciclo de actualización. El modo de operación lo podemos ver en el algoritmo 2.1 el cual se ejecutaría de forma iterativa.

Básicamente este algoritmo lo que realiza es pronosticar el nuevo estado a partir de su estimación previa añadiendo un término de corrección proporcional al error de predicción, de tal forma que este último es minimizado estadísticamente. De esta manera es posible calcular la función de verosimilitud sobre el error de predicción del modelo . El funcionamiento como podemos ver es muy simple pero intuitivo y de lo que se trata es de predecir cual será el estado de nuestro sistema en el ciclo de predicción y posteriormente ajustar esa predicción en el ciclo de actualización.

En resumen, el procedimiento completo de estimación es el siguiente: el modelo es descrito en forma de espacio-estado por su potencia descriptiva y para un conjunto

Algoritmo 2.1. Algoritmo filtro de Kalman

1: **Ciclo de predicción:**
2: Predecir la media del estado posterior
3: Predecir la covarianza del estado posterior
4: **Ciclo de actualización:**
5: Calcular la ganancia de Kalman
6: Actualizar la predicción realizada con la medida z_k
7: Actualizar la covarianza

inicial de parámetros dados, los errores de predicción del modelo son generados por el filtro gracias a como lo hemos modelado y a la información que nos proporcionan los sensores [12] [13]. Los errores en el modelo son utilizados para evaluar iterativamente la función de verosimilitud hasta maximizarla para obtener la estimación que maximiza la verosimilitud según las asunciones sobre los modelos de ruido.

Se invita al lector interesado en un mayor nivel de profundidad sobre la fundamentación matemática del filtro de Kalman a que consulte el anexo A.

2.2. Robot, entorno e interacción

El entorno, o el mundo para un robot es un sistema dinámico que posee estados tanto observables como ocultos o internos. El robot puede adquirir información del entorno por medio de los sensores. Sin embargo, la medidas por lo general suelen estar afectadas por ruido, y por lo general hay muchas características de nuestro entorno que no pueden ser medidas de forma directa por parte de los sensores. Como consecuencia, el robot mantiene una verosimilitud que tiene relación con su entorno y los estados relacionados. El robot también es capaz de influir sobre el entorno con el uso de actuadores. Hay que tener en cuenta que el efecto de interactuar con el entorno por medio de los actuadores es algo impredecible.

2.2.1. Concepto de estado

Los entornos se pueden definir por medio de *estados*. En nuestro caso, podemos ver los estados como una colección de todos los aspectos del robot y de su entorno que pueden tener impacto en un futuro inmediato, un ejemplo muy claro sería la pose de nuestro robot. Por otra parte podemos definir estados que simplifiquen la realidad, por ejemplo asumiendo un mundo 2D en lugar de un entorno tridimensional. Los estados

pueden cambiar con el tiempo, tal y como lo hace un robot cuando se mueve, también existe la posibilidad de que dicho estado permanezca inalterable durante la operación del robot (por ejemplo, la localización de los obstáculos en el mundo). Los estados que sufren cambios pueden ser llamados *estados dinámicos*, y por otra parte los que no cambian los podemos denominar *estados estáticos*. Los estados también incluyen variables propias de los robots tales como la pose, la velocidad o cualquier tipo de información que nos pueda resultar de interés. En el desarrollo de este trabajo denotaremos los estados con su notación más habitual x, este vector contendrá los estados específicos que necesitemos. El estado en el tiempo t lo denotaremos como x_t. Los estados más utilizados en el campo de la robótica móvil y por lo tanto en este trabajo son los siguientes:

- **La pose del robot**, que daría información sobre la localización y orientación de este relativo a sistema de coordenadas global. Los robots móviles poseen seis grados de libertad, que se traducen en seis variables de estado. Tres variables se utilizarían para la posición en coordenadas de un espacio Cartesiano de tres dimensiones y las otras tres para dar información sobre las orientaciones angulares, también conocidas como ángulos de Euler (*pitch, roll y yaw*). Para los robots, que como en nuestro caso, se mueven en un espacio planar, la pose se reduciría a únicamente tres variables, dos de ellas para la localización espacial y la última para dar cuenta de la orientación (concretamente se usaría el *yaw*)

- **La configuración de actuadores**, como podría ser la posición de las uniones en un manipulador o la distancia entre ruedas en un robot móvil.

- **La velocidad del robot**, un robot que se mueve por el espacio está caracterizado por 6 parámetros de velocidad, uno para cada uno de las variables que nombramos anteriormente en la pose. Las velocidades son denominadas como *estados dinámicos* por su propia naturaleza cambiante.

- **La localización y los objetos o características del entorno**. Un objeto en el entorno podría ser cualquier cosa, es decir, un muro, un árbol, etc. Las características de estos objetos podrían ser sus propiedades físicas (color, textura, tamaño, etc). Dependiendo del grado de interacción de nuestro robot con el entorno podemos pasar de captar media docena de características de nuestro entorno a captar millones de variables de estado. Para los problemas tratados en este trabajo las características del entorno son estáticas como ya veremos es secciones posteriores.

- **Localización y velocidades de objetos del entorno**. Muchas veces el robot no es el único objeto en movimiento dentro del entorno por lo tanto también debe tener nociones sobre lo que pasa a su alrededor con otros objetos que se mueven.

- **Información interna**. Podría tratarse tanto del estado de las baterías como del estado de los sensores que posee nuestro robot.

Un estado x_t puede ser llamado **óptimo** si es el mejor predictor de las situaciones futuras que puedan darse. Estos estados se estiman por medio de lo que se conoce como **estimación estocástica**.

2.2.2. Estimación estocástica

Un proceso estocástico es aquel cuyo comportamiento es no determinista, en la medida que el siguiente estado del sistema está determinado tanto por las acciones predecibles del proceso como por elementos aleatorios. Actualmente existen muchos avances tecnológicos, concretamente computacionales (mayor velocidad de procesadores y sistemas informáticos en general), para estimar o calcular un estado desconocido desde un conjunto de medidas tomadas del proceso, la gran mayoría de los métodos existentes no tienen en cuenta la naturaleza ruidosa de las medidas. Para ilustrar mejor la naturaleza ruidosa del proceso pensemos en la localización de un robot. Mientras que el robot se va moviendo la información que captamos del entorno varía y nuestra fuente fundamental de información son los mismos sensores. La pose se estima derivada de la información recibida por medio de medidas ruidosas procedentes de sensores eléctricos, mecánicos, inerciales, ópticos, acústicos, magnéticos, etc. Este ruido es típicamente estadístico por naturaleza (o puede ser modelado como tal sin generar un error muy grande), lo cual nos conduce a métodos estocásticos para tratar los problemas.

2.2.3. El problema de diseño del observador.

Existe un problema muy usual y extendido con respecto a la teoría de los sistemas lineales conocido como el *problema de diseño del observador*. La base de este problema radica en la dificultad existente en determinar, o mejor dicho estimar, los estados internos de un sistema lineal únicamente con la información obtenida de las salidas del sistema, como ya hemos explicado anteriormente. Hay que destacar por otro lado, que también tenemos acceso a las entradas de control de nuestro sistema (normalmente modeladas con el vector u_t) pero obviamente estas no entran en juego a la hora de tratar con este

problema ya que es algo que presuponemos que conocemos. Una manera muy típica de enfocar este problema para poder entenderlo, es la comparación con una caja negra donde podemos observar las señales que salen de esta pero no sabemos nada sobre lo que ocurre en su interior. Si llevamos el ejemplo de la caja a la ecuación de estados rápidamente veremos que existen estados no medibles directamente y es lo que conocemos por estados no observables.

La mayoría de aproximaciones para este problema se basan en modelos de espacio-estado. Existe un modelo de proceso que es capaz de modelar la transformación de los estados de dicho proceso. Este modelo puede ser representado como una ecuación en diferencias de la siguiente forma:

$$\bar{x}_k = A\bar{x}_{k-1} + Bu_k + w_{k-1} \tag{2.1}$$

Por otra parte también tenemos una ecuación conocida como el modelo de medida. Esta ecuación describe la relación existente entre los estados del proceso y las medidas. Puede representarse de la siguiente manera:

$$\bar{z}_k = H_k\bar{x}_i + v_k \tag{2.2}$$

Los términos w_k y v_k son variables aleatorias que representan el ruido de proceso y de medida respectivamente.

En secciones posteriores hablaremos más en profundad sobre las ecuaciones (2.1) y (2.2) donde explicaremos de donde derivan exactamente.

2.2.4. Interacción con el entorno

Fundamentalmente existen dos tipos básicos de interacción entre el robot y el entorno en el que se encuentra. En una de ellas el robot puede influir en los estados del entorno por medio de sus actuadores y en el otro tipo de interacción el robot puede recoger información del entorno por medio de sensores. Ambos tipos de interacción pueden ocurrir simultáneamente, aunque las trataremos por separado y las definiremos de la siguiente forma:

- **Medidas de sensores**: La percepción es el proceso por medio del que el robot obtiene información de los sensores acerca de los estados del entorno. Por ejemplo, el robot puede obtener imágenes de una cámara, o consultar a sus otros sensores para recibir información acerca de los estados del entorno. El resultado de

esa información captada es lo que conocemos como *medida*. Normalmente esta información llega con cierto retardo con respecto al momento en que se realizó la medición, por lo tanto es como si tuviéramos información acerca de algunos estados anteriores.

- **Acciones de control**. Las acciones de control cambian los estados del entorno. Algunos ejemplos podrían ser el control del movimiento de un robot (envío de comandos a las ruedas para alcanzar una cierta consigna). Incluso cuando el robot no desarrolla ninguna acción los estados suelen cambiar. Así, para tener una mayor consistencia se considera que el robot siempre ejecutará acciones de control incluso cuando decida no mover sus motores. En la práctica, el robot ejecuta continuamente acciones de control y realiza medidas de manera concurrente.

La diferencia entre *medida* y *acción de control* es crucial. Como ya hemos dicho la percepción proporciona información acerca de las variables de estado del entorno y por lo tanto intenta aumentar el conocimiento del entorno. El movimiento, por otra parte lo que produce es una pérdida de conocimiento sobre los objetos que tenemos a nuestro alrededor debido al ruido asociado a los actuadores aunque a veces una acción de control apropiada puede llegar a lograr un movimiento bastante preciso. Esto no significa que se tengan que dar por separado la medida y la acción de control, ya hemos dicho que pueden darse simultáneamente.

2.2.5. Evolución de los estados

La evolución de los estados y las medidas se considera que siguen el modelo probabilista de inferencia Bayesiana [1]. En general, el estado x_t está generado de forma estocástica (sección 2.2.2). Así, tendría sentido especificar la distribución de probabilidad con la que se ha generado dicho estado x_t. En primera aproximación, la aparición del estado x_t podría estar condicionada por todos los estados anteriores, mediciones y acciones de control. Por lo tanto, las reglas de probabilidad caracterizan la evolución de los estados como una distribución de probabilidad que puede ser dada de la siguiente forma [1]:

$$p(x_t \mid x_{0:t-1}, z_{1:t-1}, u_{1:t}) \tag{2.3}$$

Donde $u_t o u_k$ es la acción de control y $z_k o z_t$ es la información sensorial, ambos en el instante t o del instante k si hablamos de tiempo discreto.

El robot ejecuta la acción de control u_1 primero, y luego realiza las medidas Sin embargo, si el estado x es completo será suficiente la información que contiene como

para saber lo que ha ocurrido con estados previos. En particular, x_{t-1} sería suficiente información de todos los estados previos de control y medida hasta ese instante de tiempo, serían $u_{1:t-1}$ y $z_{1:t-1}$, por lo que no necesitaríamos almacenar información más allá del estado anterior. Por lo tanto, de todas las variables anteriores en la expresión (2.3), el único estado de control que importaría sería el del instante actual u_t si conocemos el estado anterior del sistema x_{t-1}. En términos de probabilidad la ecuación (2.3) quedaría de la siguiente manera:

$$p(x_t \mid x_{t-1}, u_t) \tag{2.4}$$

Esta propiedad es lo que conocemos como *independencia condicional*, que es una de las reglas de Markov que detallaremos en la sección 2.3.2 . Por otro lado algo muy similar ocurre con las medidas que generamos a partir de nuestro modelo. De nuevo, si x_t es completo tenemos:

$$p(z_t \mid x_{0:t}, z_{1:t-1}, u_{1:t}) = p(z_t \mid x_t) \tag{2.5}$$

Lo que significa que el estado x_t es suficiente para predecir las medidas z_t (estando estas afectadas por ruido). El conocimiento de cualquier otra variable, tales como las medidas pasadas, acciones de control o incluso estados anteriores, es totalmente irrelevante si x_t cumple con lo que se llama propiedad o principio de Markov (sección 2.3.2).

2.2.6. Modelado de la certidumbre

Otro concepto muy importante en la robótica probabilística es la verosimilitud. La verosimilitud nos da una idea del conocimiento interno del robot acerca de los estados del entorno, es decir, no aporta conocimiento sobre como de precisa es la estimación de un estado. Como sabemos hay estados que no pueden ser medidos de forma directa, como por ejemplo la pose de un robot. En su lugar para conocer la pose del robot lo que habría que hacer es calcularla partiendo de otros datos disponibles. Por lo tanto, podemos distinguir el verdadero estado de su certidumbre interna , o el estado de los conocimientos con respecto a ese estado partiendo de la información relacionada disponible. En la robótica probabilistica la certidumbre se representa por medio de la *probabilidad condicional*. Una distribución de certidumbre asignaría una probabilidad a cada posible hipótesis con respecto al estado verdadero. La certidumbre de un estado x_t se representaría de la siguiente manera:

$$bel(x_t) = p(x_t \mid z_{1:t}, u_{1:t}) \tag{2.6}$$

Algoritmo 2.2. Algoritmo de filtros Bayesianos [1] $(bel(x_{t-1}), u_t, z_t)$

1: **for** all x_t **do**
2: $\quad \bar{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1}) bel(x_{t-1}) dx$
3: $\quad bel(x_t) = \eta p(z_t \mid x_t) \bar{bel}(x_t)$
4: **end for**
5: **return** $bel(x_t)$

Lo anterior se traduce como la distribución de probabilidad de x_t condicionada por las medidas anteriores $z_{1:t}$ y todas las acciones de control anteriores $u_{1:t}$. Es importante notar la distribución ha sido calculada después de tomar la medida en el instante actual z_t. También podríamos calcular una distribución previa a incorporar la medida, y justo después de aplicar la acción de control u_t. Todo lo anterior se calcularía de la siguiente manera:

$$\bar{bel}(x_t) = p(x_t \mid z_{1:t-1}, u_{1:t}) \tag{2.7}$$

En el contexto del filtro de Kalman esta distribución de probabilidad será lo que conozcamos como **predicción**. Y por lo tanto calcular $bel(x_t)$ a partir de $\bar{bel}(x_t)$ es lo que conocemos como **corrección** o **actualización**.

2.3. Filtros Bayesianos

2.3.1. Algoritmo de los filtros Bayesianos

El algoritmo más extendido para calcular la certidumbre viene dado por lo que se conocen como *filtros Bayesianos*. Estos filtros calculan una distribución de certidumbre $bel(\cdot)$ a partir de las medidas y las acciones de control.

En el algoritmo 2.2 podemos ver la manera de proceder a la hora de realizar los cálculos. Podemos apreciar que este algoritmo es iterativo, esto quiere decir que la certidumbre $bel(x_t)$ en un tiempo t se calcula a partir de la certidumbre anterior $bel(x_{t-1})$ en el tiempo $t-1$. Otro detalle muy importante es que hay una dependencia temporal de forma secuencial para x. Los datos de partida para la realización del cálculo son la certidumbre en el tiempo $t-1$, el estado de la acción de control más reciente u_t y la medida más reciente z_t. Como resultado el algoritmo nos devuelve la confianza $bel(x_t)$ en el tiempo t. Como también se habrá podido apreciar, en este algortimo solo contempla uno de los dos pasos existentes en los filtros Bayesianos, el ciclo de **actualización**.

Como sabemos este tipo de algoritmos tienen dos pasos fundamentales y diferenciados. En la línea 2, lo primero que haremos será procesar la certidumbre en función del estado de control u_t. Calculamos la certidumbre del estado x_t basándonos en la certidumbre del estado x_{t-1} y de la acción de control actual u_t. En concreto, como podemos ver la certidumbre $\bar{bel}(x_t)$ que el robot asigna al estado x_t es obtenida por medio de la integral del producto de ambas distribuciones. Todo este proceso es lo que conocemos como ciclo de **predicción**.

En la línea 3 ocurre la etapa de corrección o actualización. En esta línea, vemos como multiplicamos la certidumbre $\bar{bel}(x_t)$ por la probabilidad de las medidas z_t que hemos tomado. Una vez hecho esto y normalizado el resultado obtendremos una certidumbre final $bel(x_t)$, que devolvemos en la línea 5.

Es importante tener en cuenta en qué situación nos encontraríamos si estuviéramos en el estado inicial donde no existe información previa. El algoritmo necesitaría una certidumbre inicial en el tiempo inicial, es decir $bel(x_0)$ en el tiempo $t = 0$ como condición de contorno. Si se conoce el valor de x_0 con precisión, la certidumbre $bel(x_0)$ podrá ser inicializada con una distribución de probabilidad cuya media sea dicho valor x_0, lo que se traduce en que asignamos probabilidad cero a cualquier otro valor que no sea el de x_0. Por el contrario, si no tenemos conocimiento sobre el valor inicial x_0, deberemos inicializar la certidumbre siguiendo una función de probabilidad definida en el dominio de x_0.

2.3.2. La propiedad de Markov

En la teoría de la probabilidad y en estadística, un proceso de Markov, es un fenómeno aleatorio dependiente del tiempo para el cual se cumple una propiedad específica: **la propiedad de Markov** [14][15]. En una descripción común, un proceso estocástico con la propiedad de Markov, o sin memoria, es uno para el cual la probabilidad condicional sobre el estado presente x_t, futuro y pasado del sistema son independientes. En la localización de un robot móvil, x_t es la pose del robot, y se aplican filtros Bayesianos para intentar calcular la pose relativa a un mapa del entorno en el que se encuentra el robot. Los siguientes factores pueden tener un efecto sistemático de error sobre las lecturas de los sensores por lo que pueden inducir a violaciones de los principios de la propiedad de Markov [1]:

- Dinámicas no modeladas del entorno que además no están incluidas en el estado x_t (por ejemplo, gente moviéndose alrededor del robot y los efectos que estos producen sobre los sensores y nuestra localización)

- Inexactitudes en los modelos de probabilidad utilizados en el algoritmo ($p(z_t \mid x_t)$ y $p(x_t \mid u_t, x_{t-1})$)
- Errores en las aproximaciones realizadas sobre todo cuando aproximamos la representación de la certidumbre (por ejemplo, cuando la aproximamos a una distribución Gaussiana).
- Variables del software de control del robot que puedan influir en la selección del modo de control (por ejemplo, la variable que localiza los objetivos tendrá una influencia directa sobre los comandos de control que deban aplicarse).

Cualquiera de las variables anteriores puede ser incluida en la representación de estados. Sin embargo, muchas veces es preferible una representación incompleta de los estados a otra que genere mucha complejidad en nuestro modelo y por lo tanto mucha mayor necesidad computacional. En la práctica los filtros Bayesianos se muestran bastante robustos a la hora de tratar con este tipo de problemas.

2.3.3. Eficiencia computacional

Los filtros Bayesianos son implementados de muchas maneras, como veremos en el capítulo 3 y en nuestra propia implementación, existen un gran variedad de técnicas y algoritmos que estás derivadas de este. Cada una de estas implementaciones se diferencian principalmente en la manera de tener en cuenta las medidas tomadas, la transición entre estados y la certidumbre inicial. Estas suposiciones dan paso a diferentes tipos de distribuciones de certidumbre posteriores, y por lo tanto existen varias implementaciones que tendrán a su vez distintas características. Aunque existen muchas técnicas para el cálculo de la certidumbre, por lo general en robótica la certidumbre tendrá que calcularse por medio de una aproximación. Encontrar la aproximación adecuada para realizar el cálculo será el punto clave para la correcta estimación. No existe una aproximación que sea la mejor sobre las demás y mucho menos que se pueda aplicar sobre todos los campos de la robótica con los mismos resultados. A la hora de elegir una aproximación debemos tener en cuenta los siguientes aspectos:

1. **Eficiencia computacional**: Alguna aproximaciones, como las aproximaciones Gaussianas que se tratarán en secciones posteriores, hacen posible calcular la certidumbre en el tiempo usando polinomios en la dimensión del estado, es decir que el grado coincide con el número de estados. Otras podrían requerir funciones exponenciales. Por lo tanto en función de la aproximación matemática tendremos una

aproximación que será más o menos rigurosa según el grado de complejidad del modelo.

2. **Precisión de la aproximación**: Algunas aproximaciones pueden ser mucho más certeras que otras. Por ejemplo, las aproximaciones lineales Gaussianas están limitadas por una distribución unimodal, por otra parte la representación por medio del histograma permite una aproximación por distribuciones multimodales, aunque perderíamos precisión.

3. **Dificultad en la implementación**: La dificultad de implementar algoritmos probabilisticos depende de la variedad de factores, uno de ellos será la probabilidad de la medida $p(z_t \mid x_t)$ y la probabilidad en la transición de los estados $p(x_t \mid u_t, x_{t-1})$. Por lo tanto la dificultad de la implementación radica en las técnicas que utilicemos.

En el siguiente capítulo trataremos con algunas aplicaciones de algoritmos Bayesianos en concreto con las distintas variantes de los filtros de Kalman.

2.4. Modelo de espacio-estado

Los modelos espacio-estado son en esencia una notación usada, y por lo tanto conveniente, para los problemas de estimación y control. Estos modelos fueron desarrollados con el objetivo para hacer manejable lo que de otra manera sería un análisis demasiado complicado de asumir con respecto a la notación del problema. Por lo tanto esta notación nos simplificará mucho la notación para trabajar con todas las variables que nombramos anteriormente

Si consideramos un proceso dinámico descrito por una ecuación en diferencias de orden enésimo (similar a una ecuación diferencial) descrita de la siguiente forma:

$$y_{i+1} = a_{0,i} yi + ... + a_{n-1,i} y_{i-n+1} + u_i, i \geq 0 \tag{2.8}$$

donde u_i es ruido de proceso blanco de media cero (el ruido blanco es una señal aleatoria que se caracteriza por el hecho de que sus valores de señal en dos tiempos diferentes no guardan correlación estadística) con la siguiente autocorrelación:

$$E(u_i, u_j) = R_u = Q_i \delta_{ij} \tag{2.9}$$

y los valores iniciales $(y_0, y_{-1}, ..., y_{-n+1})$ son variables aleatorias con media-cero, con una matriz de covarianzas conocida de dimensiones nxn definida:

$$P_0 = E(y_{-j}, y_{-k}), j, k \, \epsilon \, [0, n-1] \tag{2.10}$$

También se asume que la correlación es cero para el siguiente caso:

$$E(u_i, y_i) = 0 \; para - n + 1 \leq j \leq 0 \; e \; i \geq 0 \tag{2.11}$$

lo cual asegura que [3]:

$$E(u_i, y_i) = 0, i \geq j \geq 0 \tag{2.12}$$

En resumen, el ruido es estadísticamente independiente del proceso a ser estimado. La ecuación en diferencia puede ser escrita de la siguiente forma:

$$\bar{x}_{i+1} \equiv \begin{bmatrix} y_{i+1} \\ y_i \\ y_{i-1} \\ \vdots \\ y_{i-n+2} \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & \cdots & a_{n-2} & a_{n-1} \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} y_i \\ y_{i-1} \\ y_{i-2} \\ \vdots \\ y_{i-n+1} \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} u_i$$

esta ecuación nos conduce al modelo de espacio-estado que se escribe de la siguiente forma y también es conocida como ecuación de estados (que toma la forma como la ecuación (2.1)):

$$\bar{x}_{i+1} = A\bar{x}_i + Bu_i \tag{2.13}$$

la ecuación de medida se escribe de la siguiente forma (como ya habíamos visto en la ecuación (2.2)):

$$\bar{y}_i = H_i\bar{x}_i \tag{2.14}$$

La ecuación (2.13) también conocida como modelo de proceso, representa la manera de calcular el nuevo estado en función de un estado inmediatamente anterior, es decir el estado actual ayuda a calcular el siguiente instante. La forma exacta para realizar este cálculo no es otra que una combinación lineal entre el estado previo $\bar{x}_i$ también conocido como x_t y el vector u_i conocido como u_{t+1}. La ecuación (2.14) también conocida como modelo de medida, calcula las medidas del proceso o mejor dicho las observaciones que en el estado actual se denotan como $\bar{y}_i$ aunque en nuestra notación también las podemos escribir como z_t. La medidas se derivan directamente del estado interno

$\bar{x}_i$ conocido como x_t.

Capítulo 3

Filtros de Kalman

Una vez hemos comprendido todos los conceptos relacionados con la estimación iterativa de estados pasaremos a describir una importante parte de los estimadores iterativos, los filtros Gaussianos. Este tipo de filtro en esencia no son más que una aplicación en sistemas continuos de los filtros Bayesianos que vimos en el capítulo 2. Las técnicas utilizadas en los filtros Gaussianos comparten la idea básica de que la certidumbre es representada por distribuciones normales o Gaussianas multivariables. Recordemos que la expresión para una distribución Gaussiana es la siguiente [1]:

$$p(x) = det(2\pi\Sigma)^{-\frac{1}{2}} exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)) \tag{3.1}$$

Esta densidad de probabilidad de la variable x está caracterizada por dos parámetros: La media μ y la covarianza Σ. La media típicamente será un vector que tendrá la misma dimensión que el vector de estados x. Por otra parte la covarianza será una matriz cuadrada, simétrica y positiva. La dimensión de la matriz de covarianzas será la será NxN si N es la dimensión del vector de estados, por lo tanto el número de elementos presentes en la matriz de covarianzas depende del número de elementos en el vector de estados.

El compromiso para representar la probabilidad a posteriori por medio de una Gaussiana tiene varios enfoques. Uno de los aspectos más importantes es que la representaciones Gaussianas son unimodales, lo que quiere decir que presentan un único máximo. Una probabilidad a posteriori de este tipo es característica de muchos problemas de seguimiento en robótica, en la que el probabilidad se centra en el estado verdadero con un pequeño margen de incertidumbre. La probabilidad a posteriori representada por una Gaussiana es una aproximación muy pobre para la mayoría de problemas de estimación global donde pueden existir infinidad de hipótesis, y cada una de ellas puede generar su propia probabilidad a posteriori (problema que resuelven los filtros de partículas).

La presentación de una Gaussiana por medio de su media y su covarianza se conoce como *representación de momentos*. Esto es así a causa de que la media y la covarianza son el primer y el segundo momento respectivamente de una distribución de probabilidad (los restantes momentos son cero en una distribución normal). Más adelante hablaremos sobre la *representación canónica* y la *representación natural* de momentos.

3.1. Filtro clásico de Kalman (KF)

Esta herramienta es probablemente la más estudiada en relación a la implementación de filtros Bayesianos. Como ya explicamos en el capítulo 2, el filtro de Kalman fue inventado en la década de los 50 por el matemático Rudolph Emil Kalman. Inicialmente se postulaba como una técnica para filtrar y predecir el comportamiento de sistemas lineales. El filtro de Kalman implementa la certidumbre computacional para estados continuos. Por lo tanto no sería aplicable a espacios de estados discretos o híbridos.

El KF representa en un tiempo t la certidumbre por medio de una media μ_t y una covarianza Σ_t. La probabilidad a posteriori también será una Gaussiana siempre y cuando las siguientes propiedades se cumplan [1]:

1. La probabilidad del siguiente estado $p(x_t \mid u_t, x_{t-1})$ debe ser una función lineal afectada con ruido Gaussiano. Lo cual se expresa según la siguiente ecuación (que ya vimos en el capítulo 2) que tiene la misma forma que la ecuación (2.1):

$$\bar{x}_t = A\bar{x}_{t-1} + Bu_t + w_{t-1} \tag{3.2}$$

   Donde x_t y x_{t-1} son vectores de estado y u_t es el vector de control en el tiempo t. En la notación que estamos utilizando estos dos vectores son verticales, es decir, están formados de la siguiente manera:

$$x_t = \begin{pmatrix} x_{1,t} \\ x_{2,t} \\ \vdots \\ x_{n,t} \end{pmatrix} \tag{3.3}$$

$$u_t = \begin{pmatrix} u_{1,t} \\ u_{2,t} \\ \vdots \\ u_{n,t} \end{pmatrix} \tag{3.4}$$

Los elementos A y B de la ecuación (3.2) son matrices,A es cuadrada NxN, donde N es la dimensión del vector de estados x_t. Por otra parte B tiene dimensiones NxM, donde M corresponde a la dimensión del vector de control. Multiplicando el vector de estados y el de control por las matrices A y B respectivamente la transición de estados se daría de forma **lineal**. Por la razón anterior se asume que el KF trabaja con sistemas de **dinámicas lineales**.

Por otra parte tenemos el factor ω_{t-1} que como ya comentamos en el capítulo anterior, hace referencia al vector de ruidos Gaussianos que pretende modelar la evolución aleatoria de los estados. Este vector deberá tener las mismas dimensiones que el vector de estados. Se caracterizan también por tener media cero y una covarianza no nula denominada R_t. Una transición de estados como la de la ecuación (3.2) se denomina *lineal-Gaussiana* para resaltar el hecho de que sus argumentos son lineales y está afectada por ruido Gaussiano aditivo. En la ecuación (3.2) se define la transición de estados en base a la probabilidad $p(x_t \mid u_t, X_{t-1})$, además esta ecuación calcula la media del siguiente estado. Para obtener la covarianza R_t tenemos que:

$$p(x_t \mid u_t, x_{t-1}) = det(2\pi R_t)^{-\frac{1}{2}} exp(-\frac{1}{2}(x_t - A_t x_{t-1} - B_t u_t)^T R_t^{-1}(x_t - A_t x_{t-1} - B_t u_t) \tag{3.5}$$

2. La probabilidad de las medidas $p(z_t \mid x_t)$ también debe tener argumentos lineales, con ruido Gaussiano aditivo. La ecuación de medida tiene la siguiente forma:

$$z_t = C_t x_t + \delta_t \tag{3.6}$$

Como vemos tiene la misma forma que la ecuación (2.2). Aquí C_t es una matriz de dimensiones KxN, donde K es la dimensión del vector de medidas z_t. Por otro lado el vector δ_t describe el ruido de medida. La distribución de probabilidad de δ_t es multivariable Gaussiana con media cero y covarianza Q_t. La probabilidad en

Algoritmo 3.1. Algoritmo KF [1] $(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$

1: $\bar{\mu}_t = A_t\mu_{t-1} + B_t u_t$
2: $\bar{\Sigma}_t = A_t\Sigma_{t-1}A_t^T + R_t$
3: $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_T C_t^T + Q_t)^{-1}$
4: $\mu_t = \bar{\mu}_t + K_t(z_t - C_t\bar{\mu}_t)$
5: $\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t$
6: **return** μ_t, Σ_t

la realización de la medida viene dada por:

$$p(z_t \mid x_t) = det(2\pi Q_t)^{-\frac{1}{2}} exp(-\frac{1}{2}(z_t - C_t x_t)^T Q_t^{-1}(z_t - C_t x_t)) \tag{3.7}$$

3. Por último, la certidumbre inicial $bel(x_0)$ seguirá una distribución normal. La media de la certidumbre la denominaremos μ_0 y la covarianza como Σ_0. La probabilidad la expresamos de la siguiente manera:

$$bel(x_0) = p(x_0) = det(2\pi\Sigma_0)^{-\frac{1}{2}} exp(-\frac{1}{2}(x_0 - \mu_0)^T \Sigma_0^{-1}(x_0 - \mu_0)) \tag{3.8}$$

3.1.1. Algoritmo del KF

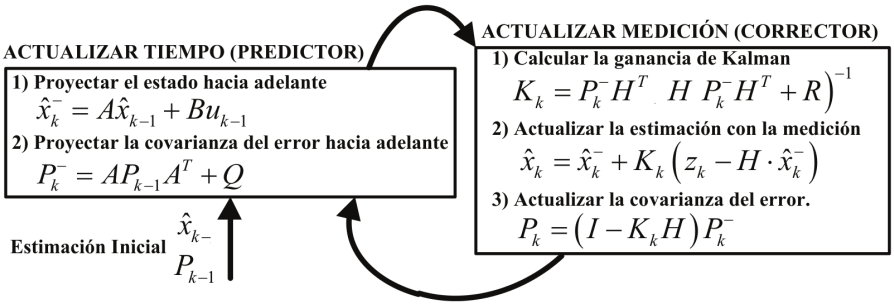

Figura 3.1. Iteración del KF [2][3].

Una vez ya tenemos claros todos los principios sobre los que se sostiene el filtro de Kalman clásico, podemos pasar a su implementación en un algoritmo. Podemos ver el planteamiento en el algoritmo 3.1. Las entradas al algoritmo son la certidumbre en un tiempo $t-1$, representada por su media μ_{t-1} y su covarianza Σ_{t-1}. Para la actualización de estos parámetros necesitaremos el vector de control u_t y las medidas tomadas en el

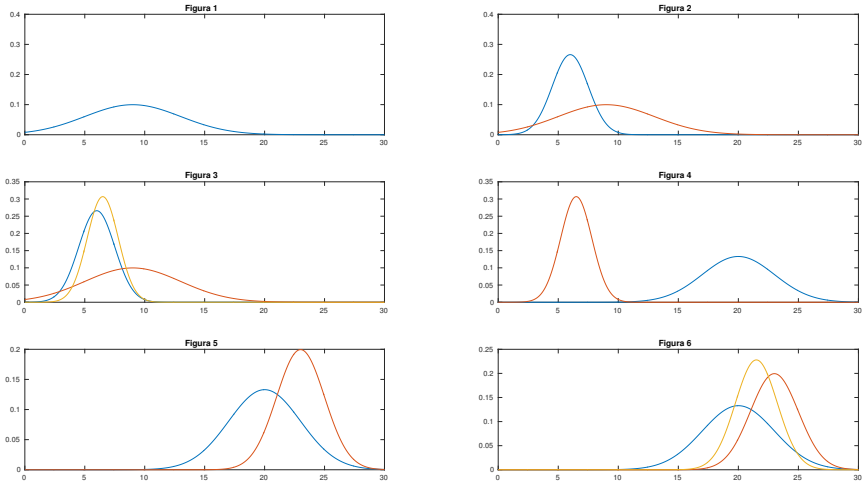

Figura 3.2. Funcionamiento del KF [1].

instante actual z_t. Podemos ver que la salida del algoritmo es la nueva media u_t y la nueva covarianza Σ_t de la certidumbre en el instante t. El KF es una herramienta muy eficiente a nivel computacional, ya que como podemos apreciar el cálculo matricial que realiza es bastante asequible para cualquier computador.

En la figura 3.2 muestra un ejemplo simple sobre el funcionamiento del filtro en un espacio unidimensional. Suponemos que el robot realizará movimientos hacia la derecha en el eje x. La probabilidad a priori de la localización del robot se muestra en la subfigura 1, donde podemos apreciar que la hemos caracterizado siguiendo una distribución Gaussiana. El robot realizará las medidas del entorno por medio de sus sensores (telémetro, ultrasonidos, etc), y estos devolverán una medida representada mediante otra Gaussiana cuya media será el valor de medida como podemos ver en la subfigura 2. La Gaussiana de la medida tiene el valor de pico centrado en el devuelto por los sensores, en cuanto a la varianza, esta corresponde a la incertidumbre existente en dicha medida. Combinando la probabilidad a priori con la medida obtenemos la probabilidad a posteriori que podemos ver en la subfigura 3. Podemos ver que la certidumbre resultante se encuentra entre las dos medias originales, y la incertidumbre que tiene la nueva Gaussiana es menor que el de las dos Gaussianas anteriores. El hecho de que las incertidumbre de la Gaussiana resultante sea menor que el de las anteriores puede parecer poco intuitivo. Este hecho es característico de la integración de la información en el filtro de Kalman donde en general cuanta más información tenemos disponible menos incertidumbre tendremos.

Siguiendo con el ejemplo, ahora suponemos que el robot realiza un movimiento hacia la derecha. En la subfigura 4 podemos ver como la incertidumbre ha aumentado además de que se ha desplazado en la misma medida que el robot lo ha hecho en el eje hacia la derecha. En la subfigura 5 el robot vuelve a realizar una medida, que combinada con la probabilidad a priori da como resultado una nueva certidumbre que se muestra en la subfigura 6. El filtro de Kalman alterna entre el *ciclo de actualización* donde integra las medidas tomadas por lo sensores con el *ciclo de predicción* que modifica la certidumbre conforme a las acciones que ha realizado el robot. El *ciclo de actualización* reduce la incertidumbre y el *ciclo de predicción* por lo general la aumenta.

3.2. Filtro extendido de Kalman (EKF)

La suposición de que los sistemas tienen transiciones lineales entre estados y que las medidas también son lineales afectadas con ruido Gaussiano aditivo raramente se cumple en la práctica [1]. Un claro ejemplo de la afirmación anterior sería un robot que se mueve con velocidad lineal y angular constante describiendo una trayectoria circular, esta transición de estados no podría ser descrita por ningún tipo de función lineal. Esta suposición, junto con la suposición de que la certidumbre sigue una distribución unimodal , hace que el KF sea una aproximación demasiado simple, por lo que sería aplicable solamente a problemas triviales de robótica siendo inaplicable en el resto de casos.

El EKF supera al KF en cuanto a uno de sus supuestos básicos, la linealidad. En el EKF la transición entre estados y las medidas están descritas por funciones no lineales, las funciones g y h respectivamente [1]:

$$x_t = g(u_t, x_{t-1}) + \varepsilon_t \tag{3.9}$$

$$z_t = h(x_t) + \delta_t \tag{3.10}$$

Este modelo estrictamente generaliza el modelo lineal Gaussiano en el que se basa el KF, este modelo lo podemos ver en las ecuaciones (3.2) y (3.6). La función g remplaza las matrices A_t y B_t de la ecuación (3.2), y h reemplaza la matriz C_t en la ecuación (3.6). La evolución de los estados al usar matrices g y h seleccionadas arbitrariamente no presentan una certidumbre distribuida siguiendo una Normal como si ocurría en el KF. De hecho, la realización de la actualización de la certidumbre es normalmente

imposible de desarrollar para funciones no lineales g y h, ya que los filtros Bayesianos no tienen un método para efectuar este cálculo.

El EKF calcula una aproximación a la certidumbre real por medio de una distribución Gaussiana. En particular, la certidumbre $bel(x_t)$ en un tiempo t es representada por una media μ_t y una covarianza Σ_t como ya explicamos en el capítulo 2. Así el EKF hereda del KF la forma de representar la certidumbre, pero se diferencia en que esta será una mera aproximación y no es calculada como en el KF.

3.2.1. Linealización mediante la expansión de Taylor

Como ya hemos dicho uno de los puntos básicos del EKF es la representación de la certidumbre mediante una aproximación de esta. Por lo tanto la idea principal es la *linealización*. Si suponemos que tenemos una función no lineal g una función Gaussiana proyectada a través de esta función se convertirá en no Gaussiana. Esto es porque existen no linealidades en g que perturban la certidumbre de tal manera que hacen que no puede ajustarse a una Gaussiana. Por lo tanto con la linealización lo que buscamos es aproximar la función g con una función lineal que sea tangente a la función g en la media de la Gaussiana. Proyectando la Gaussiana por medio de esta aproximación lineal, la distribución de probabilidad a posteriori si será una Gaussiana. De hecho, una vez que g ha sido linealizada el método para la propagación de la certidumbre es igual que para el KF. La misma explicación se aplica a la multiplicación de Gaussianas cuando se utiliza la función h. Para este caso el EKF vuelve a aproximar la función h por medio de una función lineal que es tangente a h, manteniendo así la naturaleza Gaussiana de la probabilidad a posteriori.

A la hora de linealizar funciones no lineales pueden utilizarse una gran variedad de técnicas. En concreto el EKF utiliza un método llamado **expansión de Taylor** [16]. En el EKF se suelen trabajar con expansiones de primer y segundo orden. La expansión de Taylor constituye una aproximación lineal a una función g por medio del valor de sus derivadas y su pendiente. La pendiente viene dada por la derivada parcial siguiente:

$$g'(u_t, x_{t-1}) := \frac{\partial g(u_t, x_{t-1})}{\partial x_{t-1}} \tag{3.11}$$

Está claro que el valor de g y su pendiente dependerá de los argumentos de g (vector de estados y de control). Una elección lógica para seleccionar los argumentos es elegir el estado que se considera más probable en el instante en el que se realiza la linealización. Para las distribuciones Gaussianas, el estado más probable es la media del posterior

estado de control u_{t-1}. En otras palabras, g es aproximada por su valor en el instante en que se da u_{t-1} (y u_t). La linealización por tanto se conseguiría de la siguiente forma:

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + g'(u_t, \mu_{t-1})(x_{t-1} - \mu_{t-1}) = g(u_t, \mu_{t-1}) + G_t(x_{t-1} - \mu_{t-1}) \tag{3.12}$$

Si lo escribimos utilizando la expresión de una Gaussiana tendríamos:

$$\begin{aligned} p(x_t \mid u_t, x_{t-1}) \approx \\ det(2\pi R_t)^{-\frac{1}{2}} exp(-\frac{1}{2}[x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})]^T \\ R_t^{-1}[x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})]) \end{aligned} \tag{3.13}$$

La matriz G_t es una matriz de dimensiones nxn, donde n es la dimensión del vector de estados. Esta matriz se conoce con el nombre de **Jacobiano**. El valor del Jacobiano depende de u_t y u_{t-1}, por lo tanto se diferencia para distintos instantes de tiempo.

La misma linealización es implementada para la función de medida h. En esta expresión la serie de Taylor se desarrolla alrededor de $\bar{\mu}_t$, que es el estado que se considera más probable para linealizar la función de la siguiente manera:

$$\begin{aligned} h(x_t) &\approx h(\bar{\mu}_t) + h'(\bar{\mu}_t)(x_t - \bar{\mu}_t) \\ &= h(\bar{\mu}_t) + H_t(x_t - \bar{\mu}_t) \end{aligned} \tag{3.14}$$

Sabiendo que $h'(x_t) = \frac{\partial h(x_t}{\partial x_t}$ podemos escribir la expresión en forma de Gaussiana:

$$\begin{aligned} p(z_t \mid x_t) = det(2\pi Q_t)^{\frac{-1}{2}} exp(-\frac{-1}{2}[z_t - h(\bar{\mu}_t) - H_t(x_t - \bar{\mu}_t]^T \\ Q_T^{-1}[z_t - h(\bar{\mu}_t) - H_t(x_t - \bar{\mu}_t)]) \end{aligned} \tag{3.15}$$

De esta manera aplicariamos los desarrollos de Taylor en el EKF y podríamos seguir usando los supuestos de que las certidumbres vienen dadas por distribuciones Gaussianas.

3.2.2. Algortimo del EKF

En el algoritmo 3.2 y la figura 3.3 podemos ver las ecuaciones que usa el EKF para funcionar. Como podemos apreciar este algoritmo es muy parecido al algortimo 3.1 donde se aplicaba el KF. Las diferencias más importantes las podemos apreciar en las

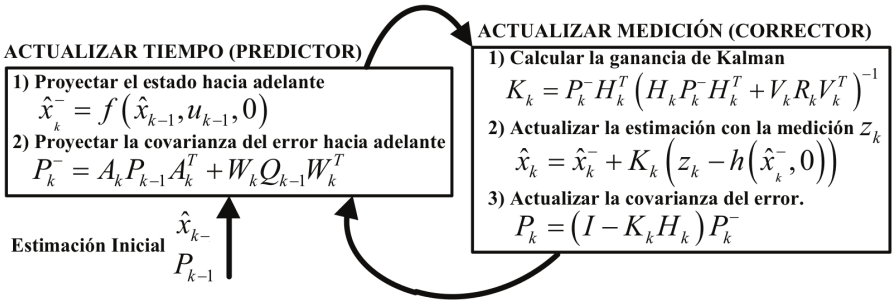

Figura 3.3. Iteración del EKF [2] [3].

Algoritmo 3.2. Algoritmo EKF [1] $(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$

$\bar{\mu}_t = g(u_t, \mu_{t-1})$
$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$
$\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
$\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$
return μ_t, Σ_t

líneas 1 y 4 en ambos algoritmos. La predicción del estado (línea 1) en el KF se realiza de la forma $A_t \mu_{t-1} + B_t \mu_t$ mientras que en el EKF se realiza de la forma $g(u_t, \mu_{t-1})$ por lo tanto la predicción de los estados se calculan de forma diferente. Por otra parte también se diferencian en la predicción de la medida (línea 4) en el KF se expresa como $C_t \bar{\mu}_t$ mientras que en el EKF se define como $h(\bar{\mu}_t)$.

Podemos ver que la predicción lineal usada en el KF ha sido reemplazada por expresiones no lineales en el EKF. Además, el filtro extendido de Kalman usa los Jacobianos G_t y H_t en lugar de las matrices lineales A_t,B_t y C_t presentes en el filtro clásico de Kalman. El Jacobiano G_t en el EKF corresponde a las matrices A_t y B_t del KF, por otro lado el Hessiano H_t corresponde a la matriz C_t.

3.3. Filtro *unscented* de Kalman (UKF)

El filtro UKF es una formulación para sistemas no lineales [17] al igual que el EKF pero sin utilizar la forma de linealizar el sistema de este último. La diferencia básica está en la manera en la que las variables aleatorias Gaussianas que modelan el ruido son representadas y propagadas a través de la dinámica del sistema. El UKF resuelve este problema por medio de un muestreo determinista. El UKF mantiene la estructura

de predicción-corrección tanto del KF como del EKF que podemos ver en las figuras 3.1 y 3.3 pero la forma de realizar el cálculo es diferente. En lugar de estimar la covarianza del error Σ_k^- utilizando el procedimiento del EKF, se hace uso de lo que se conoce como transformación Unscented (UT) [17] que calcula un conjunto de puntos de muestra, conocidos como **puntos Sigma**, para representar de forma más precisa la media y la varianza del estado (suponiendo que esta tiene una distribución Gaussiana). Si propagamos estos puntos a través del sistema no lineal de las ecuaciones (3.9) y (3.10) la media y la covarianza posterior se obtiene para cualquier no-linealidad presente en el sistema de forma precisa, dando un mejor resultado si lo comparamos con la linealización realizada en el EKF tal y como podemos ver en la figura 3.4 [18] [19] [20]. Los valores de los puntos sigma (que son escogidos cuidadosamente) se utilizan para obtener la predicción del estado y de su covarianza en una estimación *a priori*. El UKF se postula como un filtro muy preciso en cuanto a la estimación de estados (se trataría de una aproximación de segundo orden), sin embargo, es muy costoso en términos de potencia de cálculo necesaria ya que el algortimo de este filtro necesita calcular la raíz cuadrada de la matriz Σ_k^- aunque por otra parte no es necesario el cálculo del Jacobiano ni del Hessiano. El UKF no podría ser implementado en robots con recursos limitados ya que para estos llevaría mucho tiempo tratar de calcular la operación anterior.

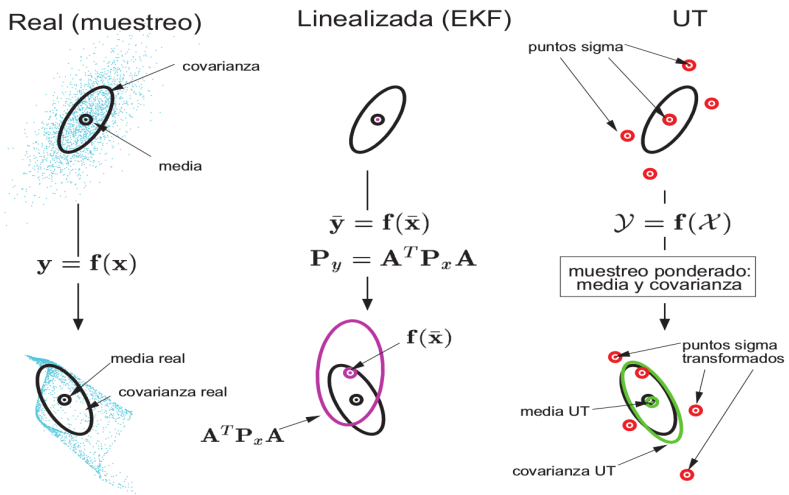

Figura 3.4. Comparativa linealización y transformada UT [2] [4] .

3.3.1. La transformación *unscented* (UT)

La predicción de los estados futuros de un sistema puede calcularse de distintas formas, la transformada UT es uno de ellas. Suponiendo que x es un variable aleatoria con media μ_x y covarianza Σ_x una segunda variable aleatoria y se podría relacionar con x por medio de la función no lineal:

$$y = f(x) \tag{3.16}$$

A la función anterior le podemos calcular a su vez la media (μ_y) y la covarianza (Σ_y). Los distribución de probabilidad de la variable transformada es consistente si se satisface la desigualdad [5] [20] en términos de la covarianza y la media de la expresión siguiente:

$$\Sigma_y - E[(y - \mu_y)(y - \mu_y)^T] \geq 0 \tag{3.17}$$

Si no se cumple la condición (3.17) se podrá considerar a Σ_y subestimada. Sin embargo, la consistencia no necesariamente implica que vayamos a obtener buenos resultados ya que los cálculos finales están sujetos a la minimización del error cuadrático medio.

La transformada *unscented* es un novedoso método utilizado para calcular la distribución de probabilidad de una variable aleatoria que ha sido sometida a una transformación no lineal. Este método se basa en la idea de que es más fácil aproximar una distribución Gaussiana que aproximar una función no lineal arbitraria [19]. El método consiste en elegir un conjunto de puntos llamados **puntos sigma**, con la condición de que su media y covarianza coincida con las de la variable aleatoria x del proceso. A continuación, la transformación no-lineal f se aplica a cada punto sigma y a estos puntos ($y = f(x)$) se les calcula la esperanza $E[y]$ y la covarianza Σ_y.

Si x es una variable aleatoria de dimensión L con media $E[x] = \mu_x$ y covarianza Σ_x, entonces para calcular la media $E[y] = \mu_y$ y la covarianza $P(y) = \Sigma_y$ se construye la matriz de puntos sigma χ_0 con $(2L + 1)$ vectores columna [21] [22] resultando:

$$\begin{aligned} \chi_0 &= \mu_x \\ \chi_i &= \mu_x + (\sqrt{(L+\lambda)\Sigma_x})_i; i = 1, \cdots, L \\ \chi_i &= \mu_x - (\sqrt{(L+\lambda)\Sigma_x})_{i-L}; i = L+1, \cdots, 2L \end{aligned} \tag{3.18}$$

donde $\delta = \alpha^2(L + k) - L$ y α (generalmente $1x10^{-4} \leq \alpha \leq 1$) determinan la dispersión de los puntos alrededor de μ_x, k es una constante relacionada con el parámetro de

escala (generalmente $0 \leq k \leq 3 - L$) y $(\sqrt{(L+\delta)\Sigma_x})$ es la i-ésima columna de la matriz cuadrada $(L+\delta)\Sigma_x$.

El cálculo de la media y la covarianza de la variable aleatoria también se puede lograr mediante la utilización de una descomposición de Cholesky [23] [5] y será en definitiva la que se utilizará para la implementación que se realiza en la toolbox [8] del UKF.

Se puede comprobar que los puntos sigma contienen la media y la covarianza de x en las siguientes ecuaciones que describen en forma simplificada la idea final del método propuesto para el UKF [20]:

$$\mu^{(UT)} \equiv \sum_{i=0}^{2L} W_i^{(m)} \chi_i = \mu_x \tag{3.19}$$

$$\Sigma_x^{(UT)} \equiv \sum_{i=0}^{2L} W_i^{(c)} (\chi_i - \mu_x^{(UT)})(\chi_i - \mu_x^{(UT)})^T = \Sigma_x \tag{3.20}$$

Donde los términos $W_i^{(c)}$ y $W_i^{(m)}$ son escalares que asignan un peso estadístico a la medida y dependen de los parámetros $\alpha, k y \beta$ como se plantea en las ecuaciones (3.21). Los valores de los parámetros son determinados según el tipo de dispersión y la distribución de probabilidad propuesta en el desarrollo de cada modelo. Los pesos estadísticos $W_i^{(c)}$ y $W_i^{(m)}$ se definen de la siguiente manera:

$$\begin{aligned} W_0^{(m)} &= \frac{\lambda}{L+\lambda} \\ W_0^{(c)} &= \frac{\lambda}{L+\lambda} + 1 - \alpha^2 + \beta \\ W_i^{(m)} = W_i^{(c)} &= \frac{1}{2(L+\lambda)}; i = 1, \cdots, 2L \end{aligned} \tag{3.21}$$

donde β es el parámetro que incorpora el conocimiento que se tiene de antemano acerca de la distribución x(generalmente $\beta = 2$ cuando se trata de distribuciones Gaussianas). Ahora, la transformación no-lineal f se aplica a cada conjunto de puntos sigma con el fin de generar una nube de puntos transformados (ecuación (3.22)) y de su distribución de probabilidad estimaremos la media:

$$Y_i = f(\chi_i); i = 0, \cdots, 2L \tag{3.22}$$

$$\mu_y \approx \mu_y^{(UT)} = \sum_{i=0}^{2L} W_i^{(m)} Y_i \tag{3.23}$$

La covarianza, se determina de la siguiente manera:

$$\Sigma_y \approx \Sigma_y^{(UT)} = \sum_{i=0}^{2L} W_i^{(c)} (Y_i - \mu_y^{(UT)})(Y_i - \mu_y^{(UT)})^T \tag{3.24}$$

3.3.2. Algoritmo del UKF

Una vez entendemos el principio funcionamiento de la transformada UT podemos pasar a explicar el algoritmo completo del UKF. El UKF puede considerarse el resultado de de incorporar esta transformación al EKF [5]. Lo que intenta el UKF es mejorar las aproximaciones que se hacen de los dos primeros momentos de una variable aleatoria que resulta de propagar otra variable aleatoria tomada como Gaussiana a través de la transformada *unscented*.

En el algortimo 3.3 podemos ver como funciona este filtro, además en la figura 3.5 podemos ver el ciclo realizado durante el funcionamiento del filtro. El algoritmo 3.3 corresponde a lo que se conoce como la versión *NonAugmented* del filtro UKF que es aplicada a sistemas afectados con ruido aditivo de media cero, además es una de las versiones con la implementación más sencilla.

Para la aplicación del algortimo hay que tener en cuenta que debemos establecer unos valores iniciales para $\hat{x}_0$ y Σ_0, los valores son los siguientes:

$$\hat{x}_0 = E[x_0] \tag{3.25}$$

$$\Sigma_0 = E[(x_0 - \hat{x}_0)(x_0 - \hat{x}_0)^T] \tag{3.26}$$

La idea de este algoritmo es muy parecida a la presentada para el EKF con la única diferencia de cómo se propaga la media y la covarianza a través del sistema. La figura 3.4 ilustra con un ejemplo sencillo las diferentes formas de propagación de la media y la covarianza para un sistema de dos dimensiones. En el lado izquierdo se observan la media y la covarianza verdaderas usando el muestreo de Monte Carlo. En el centro de la figura se muestra el resultado si usaramos la linealización del EKF. Por último, podemos ver el resultado de la transformada *unscented* y como vemos únicamente nos hacen falta 5 puntos para tener una propagación con una media y covarianza representativa. Si tenemos en cuenta las ideas anteriores podemos ver las similitudes en los algortimos 3.2 y 3.3 [24].

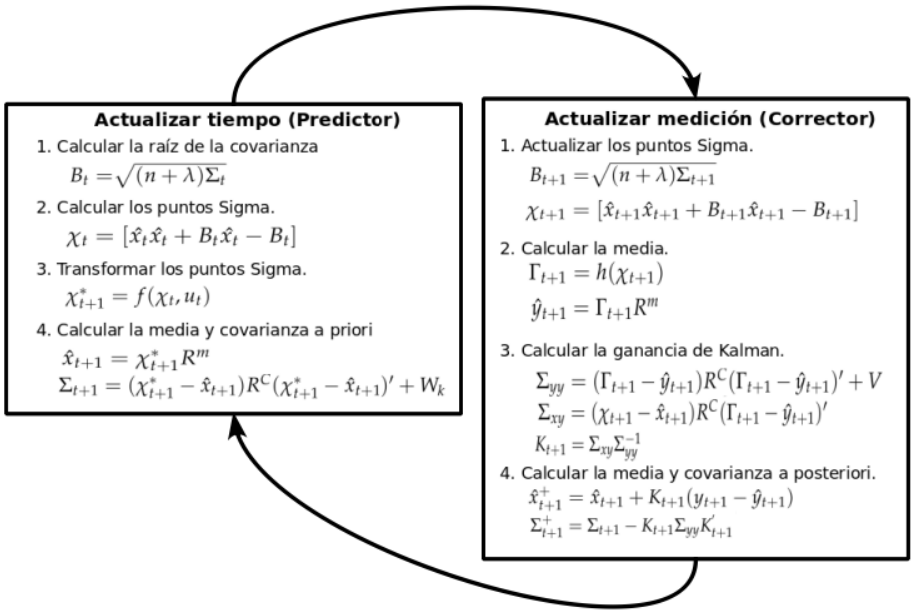

Figura 3.5. Iteración UKF [5] [6] .

3.4. Filtro de Kalman de Cubatura (CKF)

Cuando afrontamos la estimación de estados en sistemas no lineales, tenemos que dejar de lado la idea de buscar una solución óptima o analítica y debemos conformarnos con una solución subóptima dada por los filtros Bayesianos [25] [7] [26]. En términos computacionales una solución subóptima para la densidad de probabilidad a posteriori puede ser obtenida usando una de las siguientes aproximaciones:

- **Aproximación local**: En este tipo de aproximación derivamos los filtros no lineales para que la distribución de probabilidad a posteriori tome la forma de la distribución a priori. Por ejemplo, podemos asumir que se sigue una distribución Gaussiana como hemos hecho en los anteriores filtros (EKF y UKF). Este tipo de aproximaciones generan filtros más simples y de una ejecución más rápida.

- **Aproximación global**: Aquí no realizamos ningún tipo de suposición específica acerca de la densidad de probabilidad a posteriori. Por ejemplo, los filtros de partículas usando integraciones de Monte Carlo con el *sampling importance resampling (SIR)* entra dentro de esta aproximación. Normalmente lo métodos globales requieren una gran cantidad de recursos computacionales.

Desafortunadamente, los filtros para sistemas no lineales que hemos visto sufren el pro-

Algoritmo 3.3. Algoritmo UKF [5] [6]

$B_t = \sqrt{(n+\lambda)\Sigma_t}$
$\chi_t = [\hat{x}_t \hat{x}_t + B_t \hat{x}_t - B_t]$
$\chi^*_{t+1} = f(\chi_t, u_t)$
$\hat{x}_{t+1} = \chi^*_{t+1} R^m$
$\Sigma_{t+1} = (\chi^*_{t+1} - \hat{x}_{t+1}) R^C (\chi^*_{t+1} - \hat{x}_{t+1})' + W_k$
$B_{t+1} = \sqrt{(n+\lambda)\Sigma_{t+1}}$
$\chi_{t+1} = [\hat{x}_{t+1} \hat{x}_{t+1} + B_{t+1} \hat{x}_{t+1} - B_{t+1}]$
$\Gamma_{t+1} = h(\chi_{t+1})$
$\hat{y}_{t+1} = \Gamma_{t+1} R^m$
$\Sigma_{yy} = (\Gamma_{t+1} - \hat{y}_{t+1}) R^C (\Gamma_{t+1} - \hat{y}_{t+1})' + V$
$\Sigma_{xy} = (\chi_{t+1} - \hat{x}_{t+1}) R^C (\Gamma_{t+1} - \hat{y}_{t+1})'$
$K_{t+1} = \Sigma_{xy} \Sigma_{yy}^{-1}$
$\hat{x}^+_{t+1} = \hat{x}_{t+1} + K_{t+1}(y_{t+1} - \hat{y}_{t+1})$
$\Sigma^+_{t+1} = \Sigma_{t+1} - K_{t+1} \Sigma_{yy} K'_{t+1}$
return $\hat{x}^+_{t+1}, \Sigma^+_{t+1}$

blema de lo que se conoce como *maldición de la dimensionalidad* además de la *divergencia* [7]. Este efecto puede causar que el filtro vea degradado su rendimiento computacional cuando trabajamos con modelos con una alta dimensión de estados, normalmente con vectores con dimensión superior a 20 estados. La divergencia puede ocurrir por muchas razones, como podrían ser la imprecisión, un modelo incompleto de nuestro sistema físico o incluso la pérdida de información al capturar la verdadera distribución de probabilidad a posteriori. Por ejemplo, el EKF, que es uno de los filtros más utilizados para sistemas no lineales desde hace décadas trabaja bien en algunas aplicaciones de filtrado que no presenten grandes no linealidades, en caso contrario el sistema presentaría divergencias y el filtro no funcionaría correctamente.

Con la motivación de solucionar el problema anterior surgió lo que se conoce como Filtro de Kalman de Cubatura (CKF). Como sabemos los filtros Bayesianos son muy fáciles de manejar cuando se asume que todas las densidades de probabilidad son Gaussianas. De ser así el problema se reduce a calcular integrales multidimensionales cuyos integrandos son de la forma $funcinnolinealxGaussiana$. El CKF aprovecha lo que se conoce como las *reglas de cubatura* que sirven para realizar integrales multidimensionales con alta eficiencia de computo. Con las reglas de cubatura a nuestra disposición podemos describir el funcionamiento del filtro como el filtrado no lineal a través de la teoría estimación lineal, gracias a esta regla de integración el filtro posee su nombre. El CKF

se presenta como un filtro numéricamente muy preciso y además fácilmente extensible a problemas de estimación de gran número de dimensiones.

En cuanto a la base matemática el CKF presenta muchas similitudes con el UKF aunque su ámbito de aplicación es ligeramente distinto. Ambos métodos usan puntos seleccionados de forma determinista para caracterizar las distribuciones de probabilidad aunque como podremos imaginar estos se hallan de distinta manera para cada filtro. Recordemos que para el UKF seleccionábamos $(2n+1)$ puntos de muestreo con pesos $[\chi_i, \omega_i]_{i=0}^{2n}$ para caracterizar nuestra distribución de probabilidad, siendo n la dimensión del vector de estados. Para el CKF utilizaremos $2n$ puntos para caracterizar nuestra distribución de probabilidad.

3.4.1. Transformación esférica-radial

Como hemos dicho en la sección anterior, el filtrado de sistemas no lineales usando la suposición de distribuciones Gaussianas reduce el problema a cómo calcular las integrales cuyos integrandos son de la forma $funcinnolinealxGaussiana$. Específicamente consideraremos la integral de la siguiente forma:

$$I(f) = \int_{R^n} f(x) exp(-x^T x) dx \tag{3.27}$$

Definida en el sistema cartesiano de coordenadas. Para calcular el valor numérico de dicha integral lo que haremos será transformarla en la forma esférica-radial, y posteriormente se usará una regla esférica-radial de tercer grado.

En la transformación esférica-radial, el paso clave es cambiar el sistema de coordenadas cartesiano (vector x) a un sistema con radio r y vector director y. La transformación sería $x = ry$ donde $y^T y = 1$, por lo tanto $x^T x = r^2$. De esta manera la integral (3.27) puede ser reescrita en el sistema de coordenadas esférico-radial como:

$$I(f) = \int_0^\infty \int_{U_n} f(ry) r^{n-1} exp(-r^2) d\sigma(y) dr \tag{3.28}$$

Donde U_n es la superficie de la esfera definida por $U_n = [y \epsilon R^n \mid y^T y = 1]$ y σ es la superficie esférica medida o lo que es lo mismo, el área de U_n. Podemos escribir la integral radial como:

$$I = \int_0^\infty S(r) r^{n-1} exp(-r^2) dr \approx w_1 f(x_1) \tag{3.29}$$

Algoritmo 3.4. Algoritmo CKF [7]

$P_{k-1|k-1} = S_{k-1|k-1} S_{k-1|k-1}^T$
$X_{i,k-1|k-1} = S_{k-1|k-1}\xi_i + \hat{x}_{k-1|k-1}$
$X^*_{i,k|k-1} = f(X_{i,k-1|k-1}, u_{k-1})$
$\hat{x}_{k|k-1} = \frac{1}{m}\sum_{i=1}^{m} X^*_{i,k|k-1}$
$P_{k|k-1} = \frac{1}{m}\sum_{i=1}^{m} X^*_{i,k|k-1} X^{*T}_{i,k|k-1} - \hat{x}_{k|k-1}\hat{x}^T_{k|k-1} + Q_{k-1}$
$P_{k|k-1} = S_{k|k-1} S^T_{k|k-1}$
$X_{i,k|k-1} = S_{k|k-1}\xi_i + \hat{x}_{k|k-1}$
$Z_{i,k|k-1} = h(X_{i,k|k-1}, u_k)$
$\hat{z}_{k|k-1} = \frac{1}{m}\sum_{i=1}^{m} Z_{i,k|k-1}$
$P_{zz,k|k-1} = \frac{1}{m}\sum_{i=1}^{m} Z_{i,k|k-1} Z^T_{i,k|k-1} - \hat{z}_{k|k-1}\hat{z}^T_{k|k-1} + R_k$
$P_{xz,k|k-1} = \sum_{i=1}^{m} w_i X_{i,k|k-1} Z^T_{i,k|k-1} - \hat{x}_{k|k-1}\hat{z}^T_{k|k-1}$
$W_k = P_{xz,k|k-1} P^{-1}_{zz,k|k-1}$
$\hat{x}_{k|k} = \hat{x}_{k|k-1} + W_k(z_k - \hat{z}_{k|k-1})$
$P_{k|k} = P_{k|k-1} - W_k P_{zz,k|k-1} W_k^T$
return $\hat{x}_{k|k}, P_{k|k}$

donde $S(r)$ es definido por la integral esférica con la función de unidad de peso $w(y) = 1$ que puede ser aproximada por $2n$ puntos [26]:

$$S(r) = \int_{U_n} f(ry)d\sigma(y) \approx w\sum_{i=1}^{2n} f[u]_i \tag{3.30}$$

Donde $w > 0$ es el peso correspondiente para el generador $[u]$, por lo tanto finalmente tendríamos:

$$I(f) = \int_{R^n} f(x)exp(-x^T x)dx = \frac{\sqrt{\pi^n}}{2n}\sum_{i=1}^{2n} f(\sqrt{\frac{n}{2}}[1]_i) \tag{3.31}$$

Estas integrales esféricas y radiales son normalmente calculadas siguiendo lo que se conoce como **regla esférica de cubatura** y la **regla Gaussiana de cuadratura** que pueden ser consultadas en [26] [7] [25].

3.4.2. Algoritmo del CKF

En el algoritmo 3.4 y la figura 3.6 podemos ver el funcionamiento básico del CKF. Como en todos los filtros que hemos visto se mantiene la estructura predicción-actualización

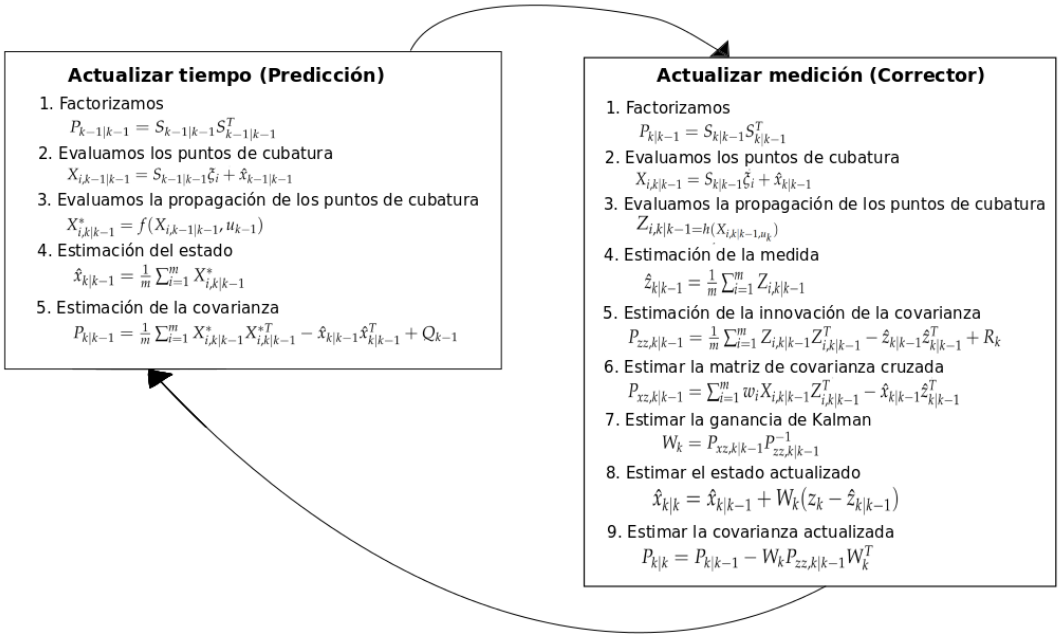

Figura 3.6. Iteración CKF [7].

que permite al filtro funcionar de forma iterativa. También hay que recordar que para la primera iteración del filtro suponemos que conocemos la densidad de probabilidad del instante anterior, es decir, $p(x_{k+1} \mid D_{k+1}) = \mathcal{N}(\hat{x}_{k-1|k-1}, P_{k-1|k-1})$. Una vez vemos el algoritmo 3.4 podríamos sacar las siguientes conclusiones acerca del CKF:

- **La regla de cubatura no utiliza derivadas**. Esta es una propiedad muy útil ya que resalta la necesidad de usar el CKF cuando resulta difícil calcular el Jacobiano o el Hessiano de un sistema. Esto es un punto a favor frente al EKF.

- **La regla de cubatura requiere 2n puntos**. Necesitamos 2n puntos de evaluación en cada ciclo del filtro (n es la dimensión del vector de estados). La complejidad computacional escala linealmente conforme aumenta el número de estados. Estos puntos son usados en los pasos 2 y 3 de ambos ciclos en la figura 3.6. En estos pasos se evalúan todos los puntos de cubatura hasta el número 2n.

Como vemos el filtro de Cubatura pretende dar un enfoque muy distinto a la estimación de estados en sistemas no lineales, por esta razón se presenta como una posible alternativa cuando la estimación usando el EKF o el UKF no es posible por tener problemas con la dimensionalidad o las divergencias del sistema.

Capítulo 4

Framework experimental

En esta sección hablaremos de todas las plataformas con las que hemos desarrollado este trabajo. Entre estas plataformas se encuentran el modelo del robot, el simulador utilizado, programas de diseño y herramientas de cálculo.

4.1. Plataforma experimental

El robot móvil que hemos decidido utilizar tiene configuración diferencial, es decir, lo que se conoce como robot de interiores. Este tipo de modelos son más fáciles de guiar a través de la mayoría de trayectorias ya que tiene la capacidad de girar sobre sí mismo. Por otro lado, la cinemática de este robot es bastante más sencilla que la de un robot que la de un robot en configuración tipo Ackermann (o modelo del triciclo), que permite simplificar la navegación del robot y presentar un caso algo más general. Para llegar a conclusiones más precisas hemos decidido usar un modelo de un robot real, para que así exista la posibilidad de implementar estos algoritmos sobre esta plataforma en un futuro. Además el Grupo de Robótica de la Univerdad de La Laguna dispone de una unidad (figura 4.2), por lo que será factible implementar esto en el futuro sobre el robot real en una continuación de este proyecto o en otro diferente.

El modelo en concreto que hemos seleccionado es el **Pioneer P3-DX** [27] ya que es una de las plataformas más usadas a nivel mundial, es fiable y compacto. Además tenemos la ventaja de que en el simulador que introduciremos más adelante tiene preinstalado el modelo. El sistema sensorial de este robot se compone de encoders ópticos situados en cada eje de motriz, con los que se puede calcular el movimiento mediante *Dead Reckoing* (aplicando el modelo cinemático diferencial) y 16 sensores de ultrasonidos para medir distancias desde el perímetro del robot. Se ha decidido dotar a este modelo de un sensor de profundidad tipo telémetro laser (Sick LMS-100), disponible también físicamente. Este sensor será considerado en las simulaciones como una fuente fiable para medir las balizas del entorno, asumiendo un modelo de ruido para el

Figura 4.1. Pioneer P3dx

Figura 4.2. Pioneer P3dx disponible en el GRULL.

mismo, que se detalla a continuación. El modelo de ruido de este tipo de sensores se distribuye de la misma forma que la Gaussiana que podemos ver en la figura 4.3. Lo que nos viene a decir esto es que un pulso láser devuelto por los sensores del robot seguirá esta distribución ya que como explicamos en el capítulo 1 esta es una forma de modelar la certidumbre en los sensores. Por lo tanto debemos tener en cuenta que la medida

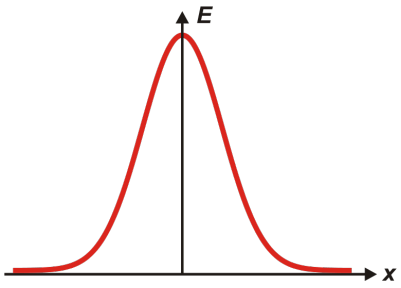

Figura 4.3. Modelo de ruido sensor.

devuelta por el sensor tiene un grado de incertidumbre con respecto al valor real de la magnitud que estamos intentando medir. Cuanto mejor caracterizado tengamos nuestro modelo de ruido mayores nociones tendremos sobre las medidas devueltas por el sensor y su fiabilidad.

En nuestro caso para la implementación del modelo de medida hemos tenido en cuenta este hecho y para ponerlo de manifiesto en nuestros scripts hemos implementado el modelo de medida de tal manera que aumenta la incertidumbre según nos alejamos.

Este robot puede alcanzar velocidades de hasta $1.6\frac{m}{s}$ y es capaz de soportar una carga de hasta 17 kilos. Por último, el robot dispone de su propia interfaz de programación, lo que facilita mucho la tarea de pasar el código desde el simulador a la plataforma real. En la figura 4.1 podemos ver el aspecto que tendría nuestro modelo según la implementación que hemos comentado y en la figura 4.2 la unidad de la que disponemos.

4.2. Modelado 3D usando Blender

Hemos utilizado un programa de diseño tridimensional, en este caso Blender, para realizar los diseños previos de nuestro Robot y así poder modificar cualquier aspecto constructivo por si fuera necesario en el modelo usado por el simulador. La idea es realizar el modelo 3D de nuestro robot para disponer de él en el caso de querer extender el diseño constructivo del mismo en el futuro, es decir, nos da la posibilidad de añadir más características constructivas y modificaciones sobre el modelo comercial. Por otra parte disponer del modelo parametrizado también nos permite poder fabricar recambios para nuestro robot con ayuda de una impresora 3D.Blender es un programa informático multi-plataforma (además de ser software libre), dedicado especialmente al modelado, iluminación, renderizado, animación y creación de modelos tridimensionales. También puede ser utilizado para la composición digital utilizando la técnica

procesal de nodos, edición de vídeo, escultura (incluye topología dinámica) y pintura digital. En Blender, además, se puede desarrollar vídeojuegos ya que posee un motor de juegos interno [28].

Por otra parte el realizar el diseño de nuestro modelo en este programa nos permite presentar el robot de una forma más precisa pudiendo prestar atención a cada detalle de la construcción de este. El modelo que podemos ver en la figura 4.4 es el que

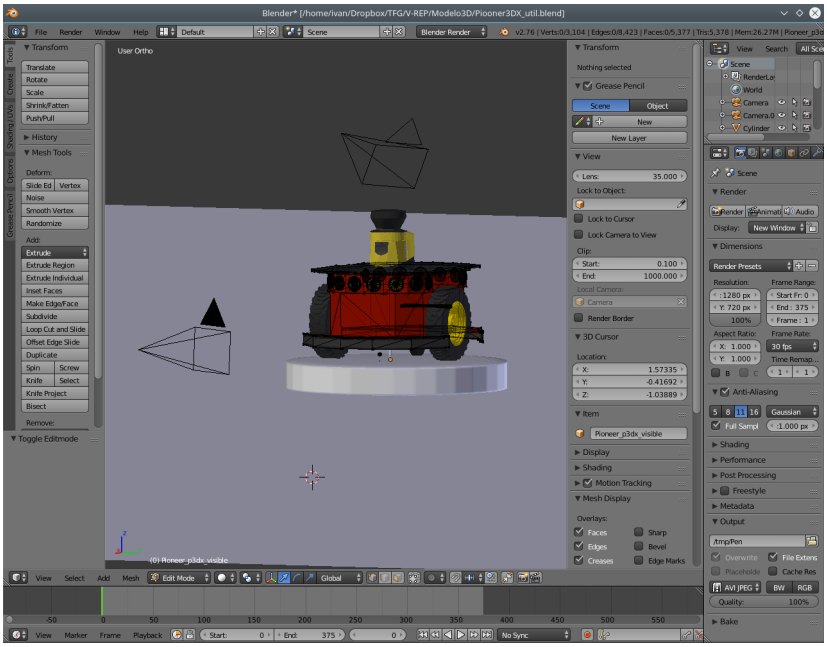

Figura 4.4. Pioneer P3dx en Blender.

posteriormente hemos implementado en V-REP para realizar las simulaciones que estimemos oportunas. Por último para realizar una presentación más vistosa de nuestro trabajo este programa nos permite realizar animaciones 3D con muy buenos resultados y exportables en cualquier formato de vídeo.

4.3. Simulación dinámica con V-REP

V-REP es programa multiplataforma y además libre que sirve para simular todo tipo de sistemas robóticos [29]. Los sistemas que pueden ser estudiados en esta plataforma van desde manipuladores antropomórficos, cadenas de montaje y también disponen de una gran cantidad de modelos de robótica móvil, podemos ver el simulador en la figura 4.5. Este programa dispone de un motor para interpretar y simular condiciones reales incluyendo físicas complejas. También dispone de funcionalidades para hacer cálculos

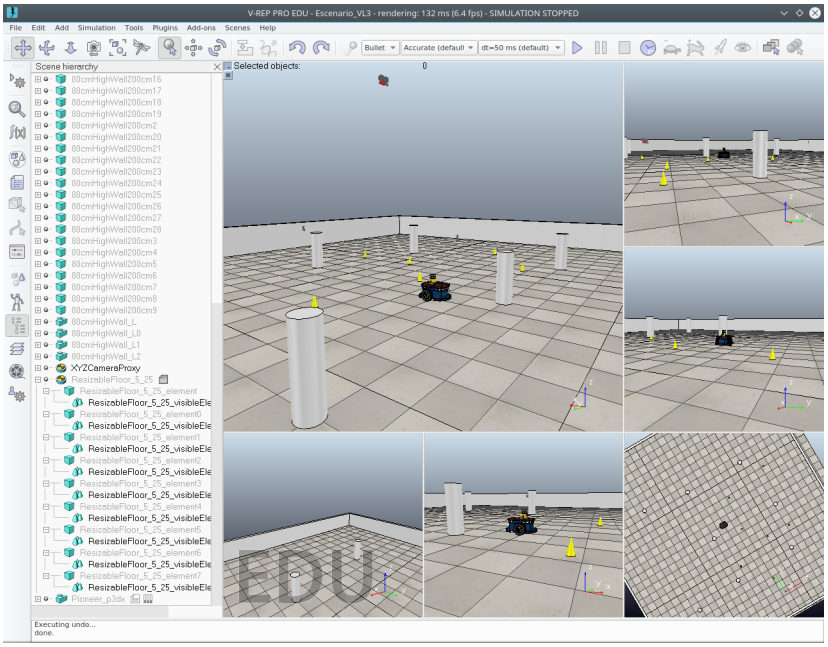

Figura 4.5. Simulación en V-REP.

de cinemática tanto directa como inversa. Ofrece un motor para gestionar las colisiones y simulaciones de partículas, como por ejemplo, robots que pintan superficies. Es capaz de simular una gran cantidad de sensores, y en especial destaca por su capacidad de simular sensores de visión, como cámaras. Además dispone con un motor de generación de rutas y planificación de movimientos, aunque no usaremos ninguno de estos para la realización de este trabajo. Permite importar una gran cantidad de modelos propios, como por ejemplo los generados en Blender además de permitir modificaciones sobre la escena cuando la simulación está en marcha.

En cuanto a la simulación, V-REP es un simulador que permite un alto nivel de personalización , de hecho cada parámetro de la simulación y del propio entorno es configurable lo que nos permite adaptar las condiciones de simulación como deseemos en cada momento. Esto es posible gracias a lo que se conoce como una API (Application Programming Interface) que nos permite controlar el simulador desde un programa externo como puede ser MATLAB/Octave, o incluso desde un código escrito en C++, Python, Java y LUA, entre otras. Por otra parte también puede ser controlado a través de una interfaz para ROS (Robot Operating System) [30]. El método que utilizaremos para trabajar con el simulador será programarlo en MATLAB y controlar la simulación desde este último. V-REP también permite trabajar con una estructura cliente-servidor lo cual nos da la posibilidad de realizar la simulación en un equipo mientras otro se encarga de realizar los cálculos, siempre y cuando los dos se encuentren conectados a

la misma red.

4.4. Control de simulación mediante MATLAB

La plataforma de MATLAB está optimizada para resolver problemas en ciencia e ingeniería. El lenguaje de MATLAB, basado en matrices, es una forma intuitiva de trabajar con datos y operar con ellos de manera rápida. Los gráficos integrados facilitan la visualización de los datos y la obtención de información a partir de ellos. Una vasta librería de toolboxes preinstaladas permiten empezar a trabajar inmediatamente con algoritmos esenciales para su dominio. Aunque en nuestro caso trabajaremos con una toolbox elaborada específicamente para trabajar con filtros de Kalman en sistemas discretos[8] y que hemos tenido que instalar de forma manual. Esta toolbox ha sido desarrollada por Simö Särkkä para la implementación de una serie de filtros, sobre todo para trabajar con sistemas discretizados como es nuestro caso con los filtros de Kalman. Hemos utilizado MATLAB para realizar todas las simulaciones pertinentes acerca de

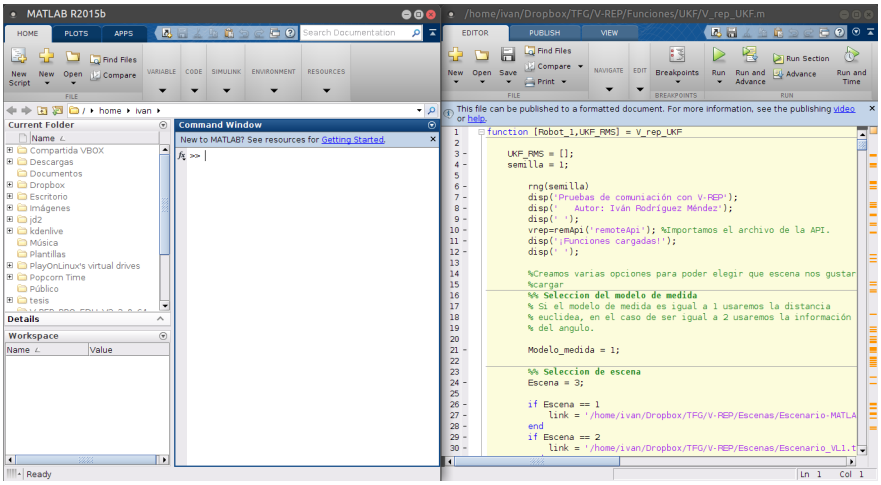

Figura 4.6. Interfaz de Matlab.

nuestra implementación del filtro de Kalman en un robot móvil. Nuestro trabajo con la toolbox no se ha limitado solamente a ejecutar los códigos que se proporcionan en esta, en el apéndice B se especifica las funciones que hemos seleccionado para implementar en nuestros scripts . Nos hemos servido de estas funciones para implementar la localización de nuestro robot, pero antes de realizarla también hemos tenido que desarrollar un entorno en el que poder trabajar en dicha localización. Hemos creado scripts en los que establecemos una serie de parámetros de simulación para nuestro robot, como pueden ser el ruido del sensor y el deslizamiento de las superficies del entorno y

el propio entorno en el que moveremos nuestro robot en V-REP. Además hemos desarrollado funciones de comunicación, modificación de propiedades de la escena y para la implementación del robot en V-REP (podemos ver todas ellas en el apéndice C). Por lo tanto desde MATLAB hemos implementado el control necesario para nuestros experimentos y el robot, la toolbox [8] ha sido una ayuda para no tener que implementar los filtros manualmente en nuestras funciones experimentales y por lo tanto no partir desde cero.

Como un primer paso, hemos realizado simulaciones en dos dimensiones sin conectarnos a V-REP, es decir, realizando todos los cálculos de la cinemática, el trazado de la ruta y la aplicación de los filtros en el código de MATLAB y representándolo por medio de gráficos en 2 dimensiones . Una vez ya teníamos algunas implementaciones bien configuradas pasamos a escribir el código para una API con V-REP. Desde MATLAB controlamos todos los parámetros relevantes de la simulación mientras V-REP se encarga de resolver la cinemática de nuestro modelo y los envía a MATLAB para su posterior tratamiento. Algunos de los parámetros que pasamos entre MATLAB y V-REP son:

- Velocidad de cada una de las ruedas.
- Posición del robot real.
- Posición del robot estimado.
- Lecturas de los sensores.
- Posición de los objetivos que el robot debe alcanzar en la escena.
- Posición de los puntos de referencia o balizas

Como vemos hay una gran cantidad de parámetros que se están enviando constantemente entre MATLAB y V-REP. Sobre estos datos MATLAB nos permite tratarlos y guardarlos con gran facilidad lo cual es una gran ventaja si queremos trabajar con todos los datos obtenidos a posteriori para estudiarlos y llegar a conclusiones sobre ellos. En el capítulo 5 especificaremos en mayor profundidad las funciones implementadas en nuestros scripts y el objetivo de estas.

Para más información acerca de las funciones utilizadas al completo dentro de la toolbox lea el apéndice B. Si quiere saber más acerca de las funciones que hemos implementado para trabajar con el simulador y la API de V-REP lea el apéndice C.

Capítulo 5

Experimentación y discusión

5.1. Equipos de trabajo y sistemas

En esta sección especificaremos los equipos con los que hemos trabajado para realizar las simulaciones y los experimentos de los que hablaremos en este capítulo.

Para la realización de los experimentos hemos trabajado principalmente con dos equipos, uno haciendo de cliente y el otro de servidor. Las características de los equipos son las siguientes:

- **EQUIPO 1 (SERVIDOR)**:
  - **Procesador**: 4x Intel(R) Core(TM) i7-3517U CPU @ 1.90 Ghz
  - **Memoria RAM**: 8 Gb
  - **Disco duro**: SSD
  - **Tarjeta Gráfica**: Nvidia GeForce GT 635M 2 Gb
  - **Sistema operativo**: Ubuntu 16.04 LTS 64 bits Kernel 4.4.0-22 generic
- **EQUIPO 2 (CLIENTE)**:
  - **Procesador**: 2x Intel(R) Core(TM)2 Duo CPU E7500 @ 2.93Ghz
  - **Memoria RAM**: 4 Gb
  - **Disco duro**: HDD
  - **Tarjeta Gráfica**: Tarjeta integrada Intel 256 MB
  - **Sistema operativo**: Kubuntu 16.04 LTS 64 bits Kernel 4.4.0-22 generic

El equipo que hace de cliente es el encargado de ejecutar todos los códigos en MATLAB para gestionar el correcto funcionamiento de la simulación y además almacena todos sus parámetros. Por otra parte el equipo que hace de servidor es el encargado de ejecutar V-REP y por lo tanto de hacer todos los cálculos relacionados con el *engine* de

físicas (tipo *bullet*). Este motor es el que más recursos computacionales necesita y por eso es más adecuado usar un equipo con mejor procesador para ello. Hemos elegido que el equipo más potente sea el que ejecute V-REP ya que la representación 3D requiere bastante potencia de cómputo y el motor de físicas del *engine* también, por otra parte gracias a los *trigger* de sincronización el equipo más lento no tiene problemas para ejecutar el código ya que el servidor espera a que acabe cada ciclo de simulación.

También tenemos la posibilidad de ejecutar el sistema cliente-servidor en un solo equipo. Para este tipo de simulaciones elegiremos el equipo 1 por ser el más potente de los dos.

5.2. Estructura básica de los experimentos.

Como comentamos en el capítulo 4, realizaremos los experimentos usando el software V-REP conectado a MATLAB por medio de una API. Hemos creado una serie de funciones (consultar apéndice C) para realizar la comunicación entre MATLAB y V-REP, y además algunas otras para modificar propiedades del simulador, nuestro modelo, etc .

Para la experimentación además hemos creado unos códigos de función principales que se encargarán de simular cada uno de los filtros estudiados en el capítulo 3. Estos códigos pueden ser consultados en el apéndice C en la séptima sección del mismo.

Para explicar el procedimiento seguido en los scripts de los experimentos sobre la localización del Pioneer P3-DX lo mejor es enseñar el procedimiento a través de un algoritmo. En total en la capítulo 3 estudiamos cuatro filtros diferentes por lo que tendríamos cuatro métodos con los que realizar pruebas y extraer diferentes datos, sin embargo la estructura organizativa de los scripts es la misma y por lo tanto aunque los códigos están diseñados para distintas herramientas el procedimiento seguido se ha mantenido en todos los experimentos.

En el algoritmo 5.1 podemos ver la estructura básica que presentan los códigos que implementan los experimentos en V-REP. Antes de pasar a comentar algunas cuestiones acerca del algoritmo debemos saber que para que nuestro código sea funcional debemos tener creadas algunas escenas y modelos para poder cargarlos posteriormente desde MATLAB en V-REP. En nuestro caso disponemos de 19 escenas diferentes ya que como explicaremos en la siguiente sección hay que testear el robot en diferentes situaciones para poder llegar a conclusiones lo más fehacientes posibles. Para lograr los diferentes coeficientes de rozamiento en nuestras escenas y por lo tanto los diferentes porcentajes de deslizamiento del robot lo que hemos hecho es utilizar el motor de

físicas tipo *bullet* (uno de los disponibles dentro de la simulación en V-REP) y hemos variado el tipo de física utilizada por el suelo, es decir, la fricción que presenta dicho material. Para conseguir estos coeficientes de rozamiento lo que hemos realizado es una interpolación lineal del parámetro adimensional que controla dicha característica en el motor de físicas, es decir, hemos visto cual es el caso de fricción baja y cual es el de fricción alta y hemos interpolado una serie de valores centrales. Como resultado de esta interpolación hemos obtenido los distintos porcentajes de deslizamiento. Por otro lado el diferencial de tiempo utilizado por este motor de físicas es de 50 *ms* con lo que nos hemos asegurado de usar los mismos para nuestros scripts. Las escenas disponibles para cargar desde nuestra API son las siguientes:

- **Escena con trayectoria recta:**En estas escenas colocamos todos los objetivos (conos) siguiendo una trayectoria recta y además disponemos de 11 balizas (cilindros) distribuidas de forma uniforme por el entorno. Podemos ver la escena en la figura 5.1. Los porcentajes de deslizamiento para el robot disponibles en las escenas son los siguientes:
  - Sin deslizamiento.
  - Deslizamiento del 3 %.
  - Deslizamiento del 5 %.
  - Deslizamiento del 7 %.

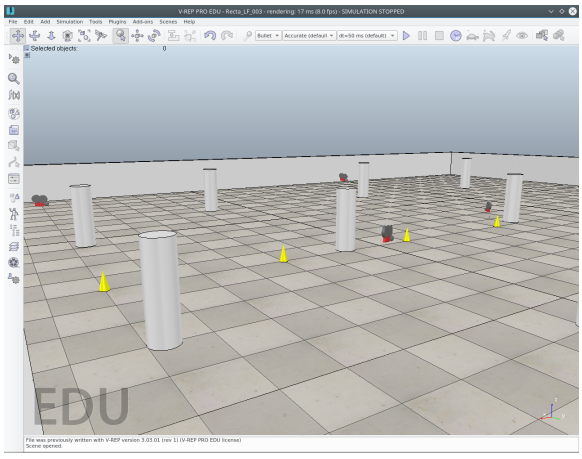

Figura 5.1. Escena trayectoria recta V-REP

Además para esta trayectoria también disponemos con variantes en las que modificamos el número de balizas, en concreto disponemos de una escena con tres balizas y otra con cinco.

- **Escena con trayectoria cuadrada:** En estas escenas colocamos 4 objetivos formando un cuadrado y además disponemos de 9 balizas que también están distribuidas de forma uniforme por el entorno pero pueden reducirse según el experimento. Podemos ver esta trayectoria en la figura 5.2. Los porcentajes de deslizamiento disponibles para esta escena son:
  - Sin deslizamiento.
  - Deslizamiento del 3 %.
  - Deslizamiento del 5 %.
  - Deslizamiento del 7 %.

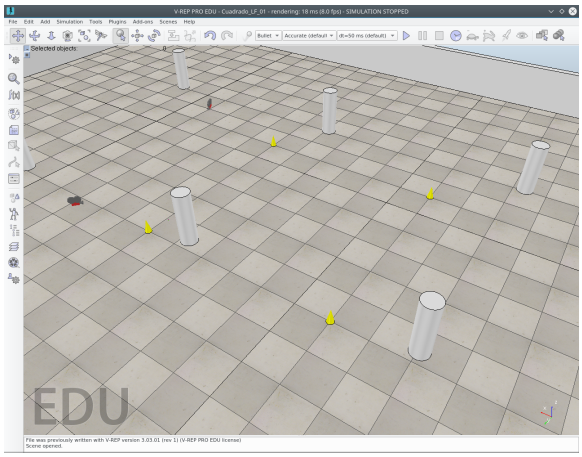

Figura 5.2. Escena trayectoria cuadrada V-REP

Al igual que para la recta, para esta escena también disponemos de unas modificaciones en la que variamos el número de balizas concretamente serán escenas con tres y cinco balizas.

- **Escena con trayectoria senoidal:** En estas escenas colocamos 7 objetivos para que el robot describa una trayectoria senoidal, además disponemos de 11 balizas en la escena aunque para algunos experimentos modificaremos el número de estas reduciendo el número a cinco y posteriormente a tres. Podemos ver la trayectoria senoidal en la figura 5.3. Los porcentajes de deslizamiento son los mismos que para las dos escenas anteriores.
- **Trayectoria arbitraria:** En esta escena realizaremos un experimento final (que podemos ver en la figura 5.4) para determinar cual es el filtro que presenta el mejor comportamiento siguiendo una trayectoria con varias curvas, rectas y diferentes características que pongan en conjunto las de las tres escenas anteriores.

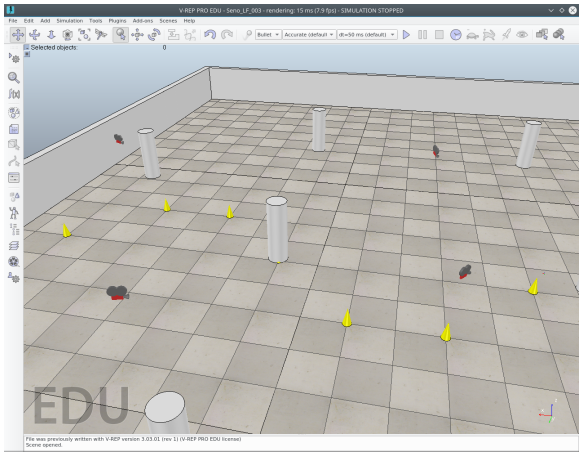

Figura 5.3. Escena trayectoria senoidal V-REP

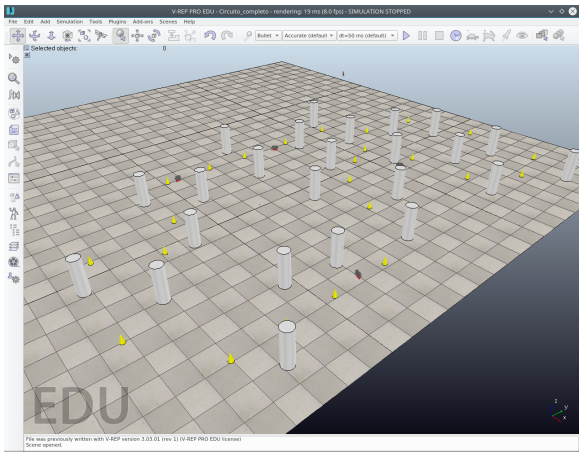

Figura 5.4. Escena trayectoria arbitraria V-REP

Con la variedad de escenas conseguimos que las ruedas deslicen en mayor o menor medida y por lo tanto el robot sufra deslizamientos dentro de la escena.

Por otra parte disponemos de dos modelos del robot, uno que hará el papel de robot principal (figura 5.5) y por lo tanto será el encargado de ejecutar los movimientos, y otro que será el robot estimado (figura 5.6) y por lo tanto mostrará la estimación de la posición que Kalman ha realizado. Los modelos utilizados como dijimos en el capítulo 4 serán los del **Pioneer P3-DX** ya que están pre-instalados en V-REP y su implementación es muy sencilla. Para la visualización de la trayectoria el robot real tiene implementado un marcador (una rotulador de un color específico) en su parte inferior, de esta manera podremos ver la trayectoria que ha seguido con mayor facilidad.

Una vez ya hemos hecho la introducción los modelos y las escenas implementadas podemos pasar a comentar el algoritmo seguido en la implementación de los expe-

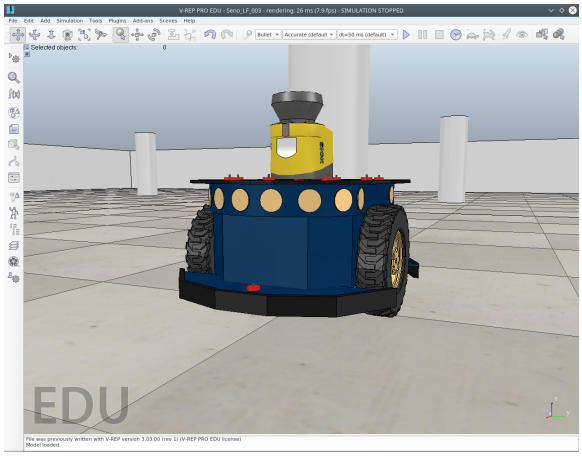

Figura 5.5. Modelo del robot real en V-REP

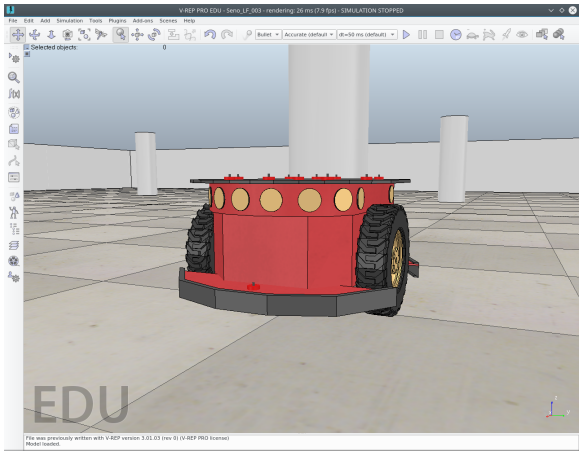

Figura 5.6. Modelo del robot estimado en V-REP

rimentos (algoritmo 5.1). En primer lugar, debemos introducir tres parámetros como entrada a las funciones experimentales. Estos parámetros son, la escena seleccionada para ejecutar la simulación, el modelo de medida que usará el robot en sus sensores y por último la afectación de ruido en los sensores. Con respecto al modelo de medida debemos añadir que esto es así ya que hemos implementado la posibilidad de tomar medidas de distancia usando el telémetro y por otra parte para añadir otras funciones de medida hemos configurado la posibilidad de medir diferencias angulares con respecto a los objetos de la escena tal y como se hace en la toolbox original [8]. Aunque el segundo modelo de medida no tenga una aplicación tan realista nos sirve para utilizar un modelo no lineal como parámetro en los filtros y así estudiar el comportamiento de estos. Las expresiones de los modelos de medida son las siguientes:

- **Modelo de medida de distancia:**

$$Distancia = \sqrt{(x_{Robot} - x_{baliza})^2 + (y_{Robot} - y_{baliza})^2} \tag{5.1}$$

- **Modelo de medida de ángulos:**

$$Angulo = \arctan \frac{y_{Robot} - y_{Baliza}}{x_{Robot} - x_{Baliza}} \tag{5.2}$$

Para estos dos modelos también hemos implementado sus derivadas ya que algunos filtros lo requieren como parámetro de entrada, por ejemplo el EKF.

Por otra parte en el algoritmo 5.1 usaremos una estructura de datos para guardar todos los datos relacionados con el robot y así poder analizarlos posteriormente, estos datos son los siguientes:

- $pose_{actual}$
- $Pose_{anterior}$
- Dimensiones del robot.
- Client ID.
- Handles del robot.
- Odometría.
- Últimas rotaciones de las ruedas.
- Medidas tomadas.
- Trayectoria real.
- Trayectoria estimada.
- Ruido del robot.

Una vez hemos pasados los parámetros de entrada a nuestra función, las primeras 12 líneas de nuestro algoritmo se refieren a la configuración de los parámetros numéricos y las propiedades de las escenas. En la línea 10 inicializamos los parámetros del filtro

Algoritmo 5.1. Algoritmo experimentos

```
Cargamos el objeto de la API remota de V-REP
Seleccionamos la escena que queremos cargar
Nos conectamos a la IP deseada (local o de un equipo remoto)
Abrimos la conexión con V-REP
Enviamos mensajes de verificación, cargamos la escena y los modelos
Definimos la posición de los objetos dentro de la escena y las guardamos en un vector
Establecemos la configuración del robot: Velocidad lineal y angular máximas, distancia de medida del telémetro, etc
Seleccionamos el modo de medida solicitado (medida de distancia o de diferencia angular)
Leemos la posición de las balizas y los objetivos dentro de la escena, guardamos esta información en matrices
Inicializamos los parámetros del filtro de Kalman
Establecemos las variables de ruido del robot
Iniciamos la simulación
for 1:NúmeroDeObjetivos do
  while Posición Actual != Posición Objetivo do
    Obtenemos la pose y la guardamos
    Calculamos el vector de control u_t
    Etapa de predicción (Filtro de Kalman)
    Realizamos las medidas con los sensores y actualizamos la odometría
    Etapa de actualización (Filtro de Kalman)
    Colocamos el robot estimado en la posición que Kalman devuelve y guardamos esta información
    Aplicamos el algoritmo de seguimiento de objetivos
    Movemos el robot según los comandos que devuelve el algoritmo
    Guardamos la pose después de movernos
    Mandamos el trigger de sincronización a V-REP
  end while
end for
Comprobamos que hemos alcanzado todos los objetivos
Calculamos el RMS (Error cuadrático medio) entre la trayectoria real y la estimada
Guardamos todos los datos de la simulación en el Struct de datos del Robot
Cerramos la escena y la conexión con V-REP
return RMS,Struct-Robot
```

suponiendo que estamos totalmente deslocalizados, es decir, media cero y una covarianza elevada. La información sobre las matrices necesarias para la inicialización de los filtros la hemos implementado de la siguiente forma:

- **Matriz A:** La matriz A debe tener la siguiente forma, siendo dt igual a 0.05 segundos coincidiendo con el diferencial de tiempo de simulación [8].

$$\begin{bmatrix} 1 & dt & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.3}$$

- **Matriz Q:** La matriz Q tomará la siguiente forma [8]:

$$\begin{bmatrix} \frac{1}{3}dt^3 q_x & \frac{1}{2}dt^2 q_x & 0 \\ \frac{1}{2}dt^2 q_x & dt * q_x & 0 \\ 0 & 0 & dt * q_y \end{bmatrix} \tag{5.4}$$

  Donde $q_x = q_y = 0{,}1$ para todos los filtros.

Desde la línea 13 hasta la 26 tenemos el algoritmo de seguimiento de rutas y aplicación del filtro de Kalman. La idea de este algoritmo es ir alcanzando uno por uno los objetivos dispuestos en la escena. Para ello implementamos dos bucles anidados que comprueban constantemente la posición del robot y envían comandos a V-REP (que a su vez los envía a las motores del robot) para que poco a poco pueda acercarse a su objetivo. Una vez entendida la idea del bucle podemos pasar a entender más en profundidad que es lo que pretendemos hacer en cada paso. En la línea 16 vemos que necesitamos calcular el vector de control u_t ya que es uno de los parámetros que necesita el filtro de Kalman para su ciclo de predicción. El código para calcular el vector de control es el siguiente [1]:

```
1 function [ u_t ] = odometry_motion(pose_ant,pose_act)
2         Rotacion_1 = atan2(pose_act(2,1) - pose_ant(2,1),pose_act(1,1) -
              pose_ant(1,1)) - pose_act(3,1);
3         Traslacion = sqrt((pose_ant(1,1) - pose_act(1,1))^2 + (pose_ant(2,1)
              - pose_act(2,1))^2 );
4         Rotacion_2 = pose_act(3,1) - pose_ant(3,1) - Rotacion_1;
5         u_t = [Traslacion*cos(pose_ant(3,1));Traslacion*sin(pose_ant(3,1));
              Rotacion_1 + Rotacion_2];
6     end
```

Como vemos calculamos el vector a partir de dos posiciones y lo hacemos convirtiendo el trayecto para alcanzarlas en una rotación, una traslación y de nuevo otra rotación. Este vector lo utilizaremos después de haber hecho el primero movimiento ya que necesitamos dos poses para calcularlo.

Las implementaciones realizadas en las líneas 17 y 19 de los filtros de Kalman pueden consultarse en el Apéndice B.

Siguiendo con el algoritmo 5.1 podemos ver en la línea 18 que realizamos las medidas con los sensores y además actualizamos la odometría. Como hemos dicho existen dos modelos para realizar las medidas, que vimos en las ecuaciones 5.2 y (5.2), y están implementados de la siguiente forma:

Modelo de medida de distancia

```
1 function [y_bal,y_real,y_adap,Pos_bal_adap,Lecturas] = Tomar_medidas_dist(
    Numero_bal,Pos_bal,pose,sd_baliza,dist_max)
2       %Funcion que realiza las medidas alrededor del robot, conforme a
3       %las balizas que tiene dentro del rango circular especificado en la
4       %de distancia.
5
6       for l=1:Numero_bal
7           y = sqrt((pose(1,1)-Pos_bal(1,1))^2 + (pose(2,1)-Pos_bal(2,1))^2
                ) + sd_baliza*randn;
8           y_bal(l,1) = y;
9           if (y <= dist_max ) %Establecemos el limite de metros que el
               telemetro es capaz de medir
10              y_real(l,1) = y_bal(l,1);
11          else
12              y_real(l,1) = 0;
13          end
14      end
15
16      Lecturas = 1;
17      y_adap = [];
18      Pos_bal_adap = [];
19
20      for h=1:size(y_real,1)
21          if (y_real(h,1) ~= 0)
22              y_adap(Lecturas,1) = y_real(h,1);
23              Pos_bal_adap(:,Lecturas) = Pos_bal(:,h);
24              Lecturas = Lecturas +1;
25          end
26      end
```

```
27
28    end
```

Modelo de medida de ángulos

```
1 function [y_bal,y_real,y_adap,Pos_bal_adap,Lecturas] = Tomar_medidas_angle(
    Numero_bal,Pos_bal,pose,sd_baliza,dist_max)
2       %Funcion que realiza las medidas alrededor del robot, conforme a
3       %las balizas que tiene dentro del rango circular especificado en la
4       %distancia maxima, aunque para este caso lo que medimos es el angulo
            .
5
6       for l=1:Numero_bal
7           y_dist = sqrt((pose(1,1)-Pos_bal(1,l))^2 + (pose(2,1)-Pos_bal(2,
                l))^2 );
8           y = atan2(pose(2,1)-Pos_bal(2,l),pose(1,1)-Pos_bal(1,l)) +
                sd_baliza*randn;
9           y_bal(l,1) = y;
10          if (y_dist <= dist_max ) %Establecemos el limite de metros que
                el telemetro es capaz de medir
11              y_real(l,1) = y_bal(l,1);
12          else
13              y_real(l,1) = 0;
14          end
15      end
16
17      Lecturas = 1;
18      y_adap = [];
19      Pos_bal_adap = [];
20
21      for h=1:size(y_real,1)
22          if (y_real(h,1) ~= 0)
23              y_adap(Lecturas,1) = y_real(h,1);
24              Pos_bal_adap(:,Lecturas) = Pos_bal(:,h);
25              Lecturas = Lecturas +1;
26          end
27      end
28
29  end
```

Podemos observar que en ambos códigos procedemos de manera muy parecida por lo que la única diferencia está en el modelo de medida. Además hemos implementado una distancia máxima de detección de las balizas para que sea lo más parecido posible a un

sensor real, de esta manera solo detectaríamos la balizas dentro del rango de medida del telémetro.

La actualización de la odometría la hacemos de la siguiente forma:

```
1function [Mov_centro , robot] = Actualizar_odom(vrep,clientID,robot) %
    Funcion para calcular al odometria de nuestro robot, es decir, sacar la
    posicion en funcion de lo que nos hemos movido.
2       robot.Pose_Anterior = robot.Odometria ;
3       [inc_izq,inc_der,robot] = Inc_encoder(vrep,clientID,robot);
4       Mov_centro = (inc_izq + inc_der)/2;
5       inc_yaw = ((inc_der - inc_izq) / robot.Radio_Robot);
6       a_norm = norm_angulo(inc_yaw + robot.Pose_Anterior(3,1));
7       robot.Odometria(3,1) = a_norm;
8       factor = 0;
9       if (abs(inc_yaw) < 0.00001)
10          robot.Odometria(1,1) = robot.Pose_Anterior(1,1) + Mov_centro*cos
                (robot.Pose_Anterior(3,1));
11          robot.Odometria(2,1) = robot.Pose_Anterior(2,1) + Mov_centro*sin
                (robot.Pose_Anterior(3,1));
12      else
13          factor = Mov_centro / norm_angulo(inc_yaw);
14          robot.Odometria(1,1) = robot.Pose_Anterior(1,1) + (sin(inc_yaw)*
                cos(robot.Pose_Anterior(3,1))- sin(robot.Pose_Anterior(3,1))
                *(1 -cos(inc_yaw)))*factor ;
15          robot.Odometria(2,1) = robot.Pose_Anterior(2,1) + (sin(inc_yaw)*
                sin(robot.Pose_Anterior(3,1))+ cos(robot.Pose_Anterior(3,1))
                *(1 -cos(inc_yaw)))*factor ;
16
17      end
18   end
```

Donde lo que hacemos es realizar una propagación, según lo que se han movido las ruedas de nuestro robot gracias a los *encoder*, de la posición del robot.

Por último dentro del bucle de seguimiento de la trayectoria y aplicación del filtro de Kalman en la línea 21 ejecutamos el algoritmo de seguimiento de rutas. Este algoritmo lo único que hace es plantear una trayectoria curva entre dos puntos, al igual que un interpolador. Si dos objetivos están muy cerca entre sí el robot girará sobre si mismo ya que no sería capaz de trazar una trayectoria curvilínea entre la posición actual y el objetivo en el que quiere posicionarse. Hemos decidido que el robot realice trayectorias curvilíneas ya que si le añadimos deslizamiento junto con este tipo de movimiento será más difícil de estimar la pose para el filtro de Kalman, es decir, lo hemos diseñado para

las peores condiciones de funcionamiento en lo que a la estimación se refiere. El código de seguimiento de rutas es el siguiente:

```
function [w,v] = Seguir_objetivos(w,v,W,V) %Funcion de seguimiento de
    objetivos, el robot tratara de describir una curva suavizada.
    %El metodo consiste en realizar una interpolacion con el suavizado
    %entre tramos de la trayectoria.
        dist = v;
        if w > W
            w = W;
        end
        if w < -W
            w = -W;
        end

        if (v > V)
            v = V;
        end

        if (v < 0)
            v = 0;
        end

        if  abs(w) > 0.05 && abs(w) < 0.15 && dist > V
            v = V/4;
        else
            if abs(w) > 0.01 && abs(w) < 0.05 && dist > V
                v = V/2;
            else
                if abs(w) > 0.15
                    v = 0;
                end
            end
        end
    end
```

En este código pasamos como parámetros las velocidades angulares y lineales máximas de nuestro robot. Posteriormente el algoritmo se encarga de calcular la velocidad angular y lineal necesarias para posicionarse en el objetivo deseado y enviamos esos parámetros a una función de movimiento que veremos a continuación.

Recordemos que las ecuaciones de la cinemática de nuestro robot son de la siguiente

forma como veremos en la función *mover robot*:

$$v = \frac{(v_r + v_l)}{2} = \frac{(w + w_r)r}{2} \tag{5.5}$$

$$w = \frac{(v_r - v_l)}{L} = \frac{(w - w_r)r}{L} \tag{5.6}$$

Los estados de nuestro robot estarían definidos como:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} -r \cdot sen(\theta)/2 & -r \cdot sen(\theta)/2 \\ r \cdot cos(\theta)/2 & r \cdot cos(\theta)/2 \\ -r/L & r/L \end{bmatrix} \begin{bmatrix} w_l \\ w_r \end{bmatrix} \tag{5.7}$$

Además del algoritmo de seguimiento de trayectorias necesitaremos calcular la cinemática (ecuaciones (5.5), (5.6) y (5.7)) de nuestro robot para saber cual es el comando que debemos enviar a cada motor, el código para ello es el siguiente:

```
1function [vl,vr] = Mover_robot(vrep,clientID,robot,v,w) %Funcion que nos
    permite que robot siga unos objetivos prestablecidos
2       if abs(v) < 0.0001 && abs(w) < 0.0001
3           vl = 0; %Detenemos la rueda izquierda
4           vr = 0; %Detenemos la rueda derecha
5       else
6           v_hat = v ; %Aqui podemos poner un termino de ruido. Hemos
                optado por considerar ese ruido como parte de la escena (
                friccion del suelo)
7           w_hat = w ; %Aqui tambien podemos poner un termino de ruido.
8           vl = (v_hat - robot.Radio_Robot * w_hat) / robot.
                Radio_Rueda_Izquierda; %calculamos los comandos que
                tendriamos que mandar a cada una de las ruedas.
9           vr = (v_hat + robot.Radio_Robot * w_hat) / robot.
                Radio_Rueda_Derecha;
10      end
11      Enviar_signal(vrep,clientID,[vr vl]); %Enviamos el string de
            movimiento al robot en V-REP.
12   end
13    function Parar_robot(clientID) %Funcion que hace que el robot pare sus
        motores.
14      Enviar_signal(clientID,[0 0]);
15    end
```

Una vez terminamos el bucle en la línea 26 únicamente debemos calcular el error

cuadrático medio cometido en la estimación (línea 28), guardar estos datos (línea 29) y cerrar la escena (línea 30). Los parámetros devueltos por el algoritmo son el error cometido en la estimación y una estructura de datos que contiene toda la información de lo acontecido dentro del experimento (trayectoria seguida, medidas realizas, trayectoria estimada, dimensiones del robot, etc).

5.3. Descripción de los experimentos propuestos

Una vez tenemos clara la estructura de nuestro códigos y su funcionamiento podemos pasar a introducir la serie de experimentos que pretendemos realizar para analizar cual es el filtro que presenta un mejor desempeño. Para la evaluación de los filtros hemos diseñado cuatro series de experimentos, con un total de 2000 simulaciones que se distribuyen como veremos en los siguientes apartados. Además en los experimentos que hemos planteado solamente variaremos uno de los parámetros para así ver como afecta al rendimiento global del filtro.

5.3.1. Impacto al añadir un sensor de odometría

Con este experimento se pretende determinar que filtros se ven afectados de manera negativa por la falta de la odometría o por contra se ven beneficiados por falta de esta información, por ejemplo cuando existe mucho deslizamiento y esta información no es válida. En la tabla 5.1 podemos ver un resumen de la serie de experimentos a realizar.

La idea es ejecutar 10 simulaciones sobre cada uno de los supuestos expuestos en la tabla para cada uno de los filtros, por ejemplo, ejecutar para el KF, EKF, UKF y CKF la escena de la recta sin afectación de ruido en los sensores y con un deslizamiento del 3 % 10 veces para cada uno y luego calcular la media del error obtenido para guardar ese dato. Además utilizaremos el modelo de medida de distancia (ecuación 5.2) para tomar la información de los sensores, en el caso del filtro de Kalman clásico ante la imposibilidad de implementar el modelo de medida realizaremos todos los experimentos con la información de la odometría (su única información sensorial) y comprobaremos que aunque no es perfecta la estimación no es tan errónea como se podría pensar.

5.3.2. Sensibilidad ante ruido de medida de las balizas

En esta serie de experimentos lo que pretendemos observar es como se comportan los filtros cuando existen distintas afectaciones de ruido en los sensores del robot y que

Tabla 5.1. Experimentos : Odometría

Experimento 1		
Recorrido	Ruido Baliza	Porcentaje de deslizamiento
Recta	0	0,00 %
		3,00 %
		5,00 %
		7,00 %
Cuadrado	0	0,00 %
		3,00 %
		5,00 %
		7,00 %
Seno	0	0,00 %
		3,00 %
		5,00 %
		7,00 %

efecto produce este hecho a la estimación realizada. Para esta serie de experimentos podemos ver la estructura en la tabla 5.2 .

Tabla 5.2. Experimentos : Ruido sensores

Experimento 2		
Recorrido	Porcentaje de deslizamiento	Ruido sensores (m)
Recta	5 %	0
		0.02
		0.05
		0.1
Cuadrado	5 %	0
		0.02
		0.05
		0.1
Seno	5 %	0
		0.02
		0.05
		0.1

Para esta serie de experimentos también ejecutaremos 10 iteraciones de cada uno de los casos presentados para cada filtro y posteriormente calcularemos la media de

los errores obtenidos. Por otra parte repetiremos la serie de experimentos usando dos modelos de medida distintos, medida de distancia (ecuación 5.2) y medida de ángulos (ecuación (5.2)).

5.3.3. Variación del número de balizas

En este experimento queremos determinar que es lo que ocurre con cada filtro cuando la información sensorial es muy pobre o muy abundante , es decir, cuando hay pocas balizas o muchas respectivamente. Consideraremos que una información suficiente es proporcionada por 5 balizas alrededor de la trayectoria y que una información pobre es proporcionada por 3 balizas. Estas pruebas se realizarán sobre las mismas 3 trayectorias de las que hemos hablado hasta el momento (una recta, una trayectoria cuadrada y una senoidal). Además ajustaremos los parámetros de deslizamiento y de ruido a sus valores medios, 5 % y 0.05 metros respectivamente. El modelo de medida utilizado será el modelo de distancia (ecuación 5.2) ya que el modelo de medida de ángulos (ecuación (5.2)) puede generar singularidades cuando existe poca información sensorial, lo cual haría que la estimación fuera errónea. Podemos ver en la tabla 5.3 un resumen de la serie de experimentos a realizar.

Tabla 5.3. Experimento: Variar el número de balizas

Experimento 3			
Recorrido	Porcentaje deslizamiento	Ruido Balizas (m)	Balizas
Recta	5,00 %	0,05	3
			5
Cuadrado	5,00 %	0,05	3
			5
Seno	5,00 %	0,05	3
			5

5.3.4. Experimentos para un caso general

Para esta serie de experimentos final trataremos de ejecutar una simulación en una trayectoria compleja. Además pondremos balizas suficientes para que todos los filtros tengan la información sensorial necesaria para su correcto funcionamiento. Por último, sintonizaremos todos los filtros (al igual que en los experimentos anteriores) de tal manera que intentemos obtener un rendimiento estándar por parte de cada uno, es decir,

testearlos en condiciones por defecto. Los parámetros por defecto ya vienen implementados en la toolbox [8] y para no añadir más complejidad a la hora de seleccionar *los puntos Sigma o los de Cubatura* hemos decidido usar los parámetros por defecto ya que aunque no es la solución óptima , es bastante funcional. Para estas pruebas estableceremos los parámetros de deslizamiento y de ruido a sus valores medios, 5 % y 0.05 metros respectivamente, podemos ver el resumen del experimento en la tabla 5.4. Además utilizaremos el modelo de medida de distancia (ecuación 5.2), por ser el más robusto, para realizar la estimación.

Tabla 5.4. Experimento: Mejores condiciones de funcionamiento

Experimento 4		
Recorrido	Porcentaje deslizamiento	Ruido sensores (m)
Trayectoria general	5,00 %	0,05

5.4. Experimento 1: Odometría

Para la sintonización que hemos realizado de los filtros hemos dejado la mayor parte de los parámetros por defecto, incluidos los *puntos sigma* y los *puntos de cubatura* que podrían ser añadidos como parámetros de entrada a las funciones del UKF y CKF respectivamente. Estos parámetros están implementados originalmente desde la toolbox [8] por lo que hemos implementado los filtros con dichos valores para que los experimentos sean repetibles, contando siempre con el error numérico del motor físico. Por otra parte, en el filtro de Kalman clásico hemos incorporado la información recogida de la odometría para la ecuación de medida lo que quiere decir que la única información para localización con este filtro se son los datos obtenidos de la odometría. Para incorporar la odometría en el EKF simplemente hemos introducido los datos necesarios como parámetro de entrada al ciclo de actualización, es decir la posición estimada según la odometría. Para integrar la odometría en el UKF y el CKF hemos realizado una ponderación entre lo que nos devuelve el ciclo de actualización y lo que hemos estimado con la odometría, de esta manera antes de entrar al ciclo de actualización estaríamos teniendo en cuenta la posición en la que nos encontramos según la odometría. La ponderación para el UKF es la siguiente:

$$M = \frac{(10 * odom + 90 * M_{predicha})}{100} \tag{5.8}$$

Como vemos al aplicar la ponderación estamos dándole más importancia a lo que nos devuelve el ciclo de predicción (90 %) que a la información que tenemos de la odometría (10 %). Estos valores se han establecido de esta manera ya que para la sintonización del filtro hemos ejecutado varias pruebas previas a los experimentos y hemos determinado que esta es la ponderación que generalmente funciona mejor. Para el caso del CKF la ponderación ha sido:

$$M = \frac{12 * odom + 3 * M_{predicha}}{15} \tag{5.9}$$

Las razones para elegir esta ponderación son las mismas que para el UKF. En esta sección de experimentos veremos como afecta implementar o no la odometría a nuestro modelo. Hemos dividido la sección en apartados que contienen los resultados para cada una de las trayectorias en función de las diferentes condiciones de experimentación. Finalmente, al final de cada apartado, representaremos la trayectoria del peor caso en cuanto a la estimación se refiere para hacernos una idea de la trayectoria seguida por el robot.

5.4.1. Resultados para la trayectoria recta

Los resultados para este tipo de trayectoria son los que se pueden ver en las figuras 5.7,5.9,5.8 y 5.10.

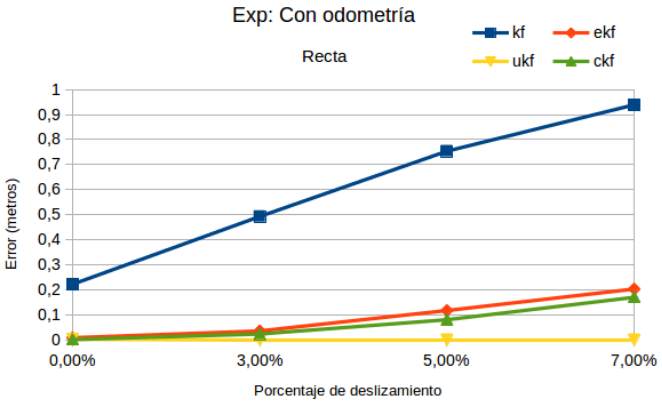

Figura 5.7. Experimento recta con odometría

En las figuras 5.7, 5.9, 5.8 y 5.10, podemos ver los resultados obtenidos para las dos tandas de experimentos, es decir, con y sin odometría. Vemos que en ambos casos el UKF es el que presenta el menor error, seguido del CKF y posteriormente por el EKF.

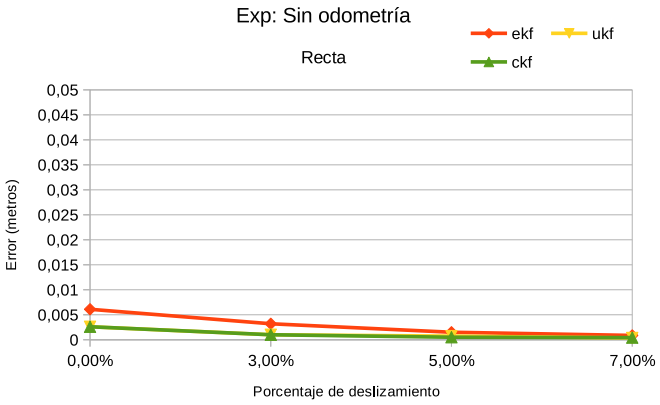

Figura 5.8. Experimento recta sin odometría

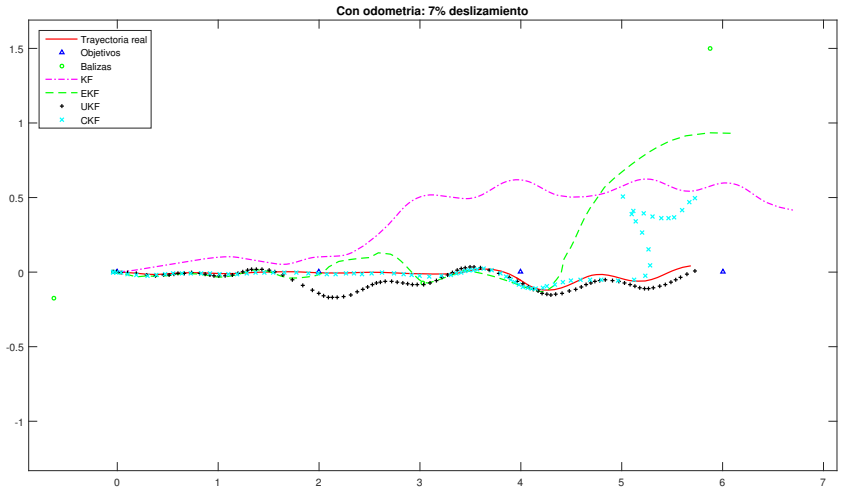

Figura 5.9. Trayectoria recta con odometría

Con respecto al filtro clásico de Kalman vemos que su estimación no es del todo correcta, aunque el error para el 7 % de deslizamiento es de 0.93 metros, y si pensamos que solamente hemos tenido en cuenta la odometría para la estimación este valor no es tan malo. Otra observación interesante entre las figuras 5.7 y 5.8 es que cuando no implementamos la odometría la estimación es mejor. Esto es debido a que los deslizamientos hacen que la odometría no sea una información válida y por lo tanto en ese caso sería mejor no implementarla como demuestra la figura 5.8 donde el máximo error obtenido es de 0.006 metros, lo que significa que hemos tenido una estimación casi perfecta. Es muy importante tener en cuenta que las trayectorias reales mostradas en las figuras 5.9 y 5.10 no son las trayectorias que han seguido todos los filtros ya que al trabajar

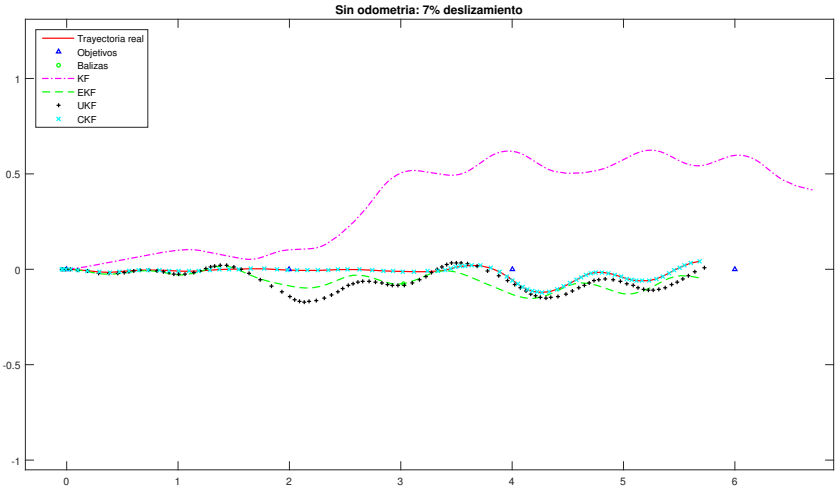

Figura 5.10. Trayectoria recta sin odometría

con deslizamientos para cada filtro la trayectoria se ha podido ver afectada de distinta manera (por ejemplo si el robot ha deslizado en otra dirección), por esta razón hemos realizado baterías de experimentos y luego hemos obtenido la media del error, con lo cual la trayectoria mostrada es meramente orientativa para saber la forma de la ruta tomada por nuestro robot y no como referencia comparativa.

5.4.2. Resultados para la trayectoria cuadrada

Para las trayectorias cuadradas la configuración ha sido la misma que para las rectas, los resultados se muestran en las figuras 5.11, 5.13, 5.12 y 5.14. En este experimento

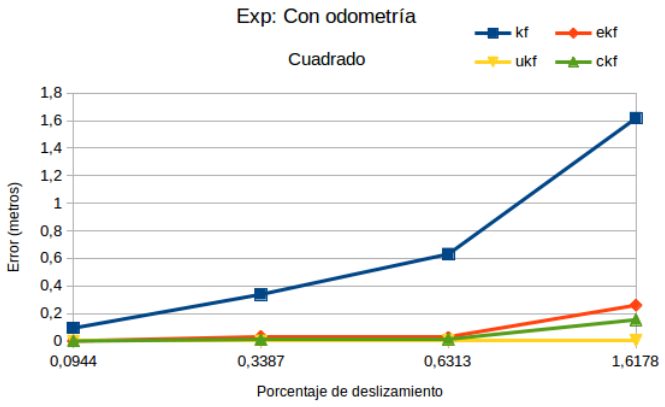

Figura 5.11. Experimento cuadrado con odometría

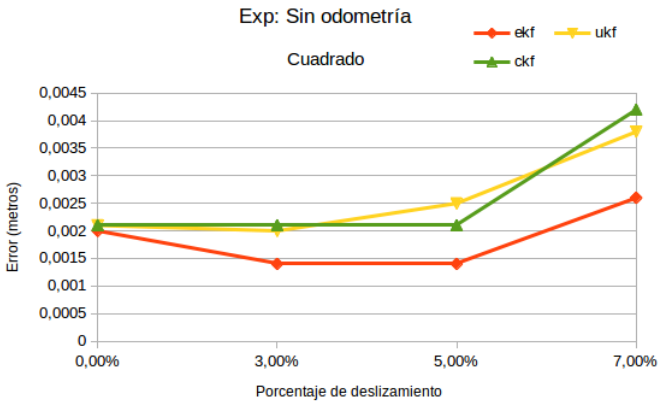

Figura 5.12. Experimento cuadrado sin odometría

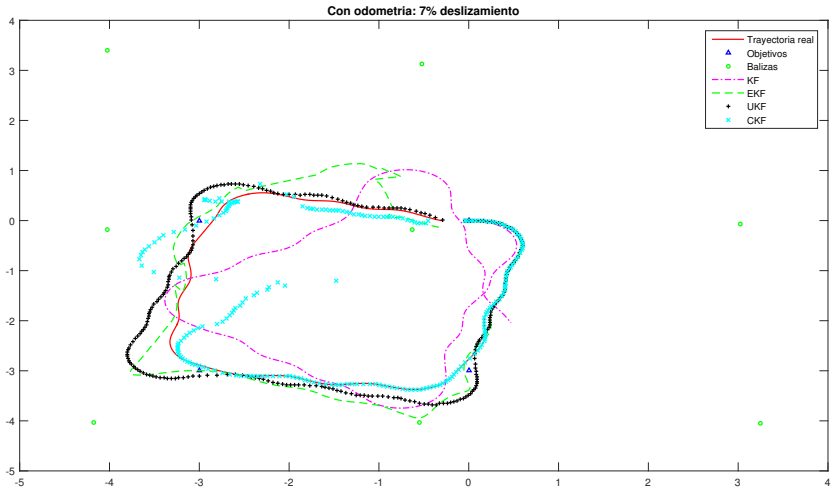

Figura 5.13. Trayectoria cuadrada con odometría

llegamos a unas conclusiones muy parecidas a las del experimento de las rectas. Vemos que los resultados en la implementación de la odometría 5.11 resulta más errónea a medida que el deslizamiento aumenta. En cambio cuando no implementamos la odometría los errores obtenidos son muy ínfimos como vemos en la figura 5.12. En este experimento el filtro que ha presentado el error más bajo ha sido el EKF en el experimento sin la implementación de la odometría como vemos en la figura 5.12.

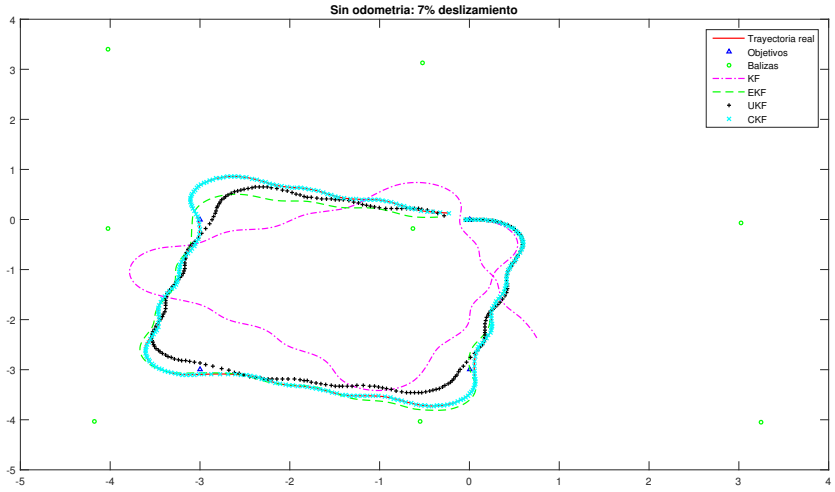

Figura 5.14. Trayectoria cuadrada sin odometría

5.4.3. Resultados para la trayectoria senoidal

Los resultados de este experimento son mostrados en las figuras 5.15, 5.17, 5.16 y 5.18.

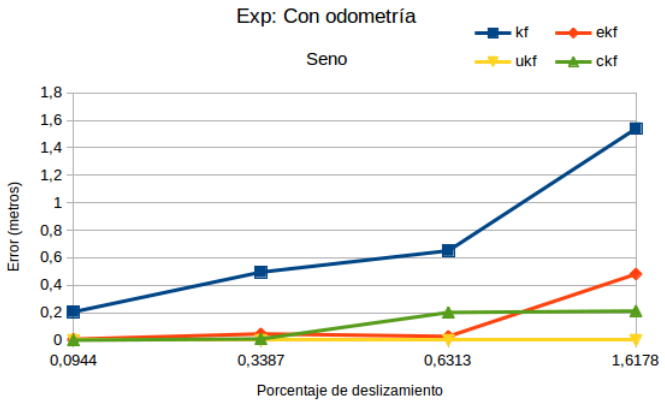

Figura 5.15. Experimento seno con odometría

Para esta serie de pruebas llegamos a las mismas conclusiones que para todas las anteriores, por lo que podemos concluir que la odometría puede ser una información peligrosa si trabajamos con deslizamientos muy grandes. Por otra parte hay que notar que en las figuras 5.17 y 5.18 el filtro clasico de Kalman tiene un comportamiento muy diferente e igualmente erróneo, esto es debido a que en cada simulación el motor de físicas del simulador hace que el robot pueda deslizar más o menos en distintas

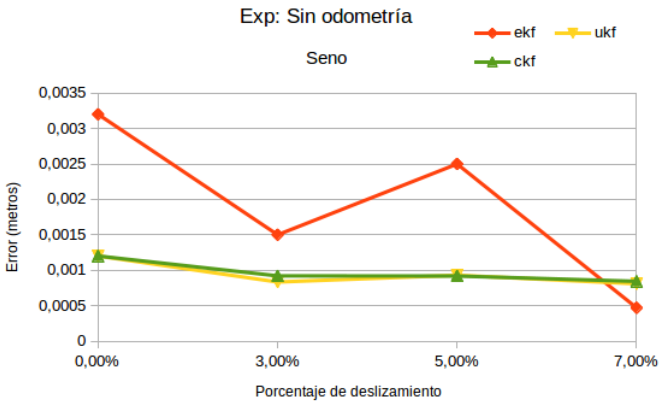

Figura 5.16. Experimento seno sin odometría

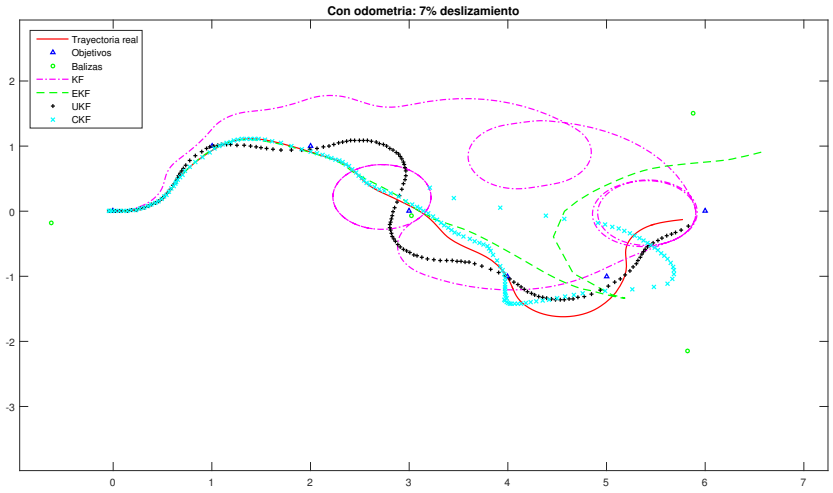

Figura 5.17. Trayectoria senoidal con odometría

zonas además de que nuestro planificador de rutas puede llegar a tomar diferentes decisiones en función de los deslizamientos, por esta razón el aspecto de la simulación es diferente y no debemos tomar las trayectorias mostradas en las figuras 5.17 y 5.18 como representación de la serie de experimentos, estas figuras son meramente orientativas y escogidas al azar dentro de la serie de pruebas realizadas.

La conclusión para esta serie de experimentos es la siguiente: **La odometría es una información que puede ayudar a localizarnos con éxito, pero con la existencia de deslizamientos puede ocurrir que esta información contribuya a la deslocalización.**

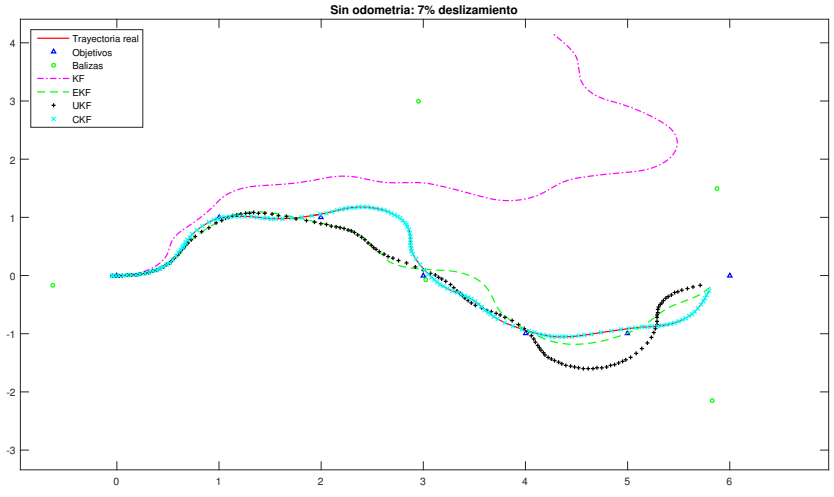

Figura 5.18. Trayectoria senoidal sin odometría

5.5. Experimento 2: Ruido en las medidas

En esta sección de experimentos veremos como afecta el ruido en los sensores a la estimación que realizan los filtros. Representaremos los resultados por medio de gráficos para una mejor comprensión y análisis de estos. Además representaremos las trayectorias seguidas por los filtros para el peor caso de estimación y usaremos dos funciones de medida distintas de las que ya hemos hablado: la función de medida de distancia (ecuación 5.2) y la de medida de ángulos (ecuación (5.2)). La sintonización de los filtros y sus parámetros es la misma que para el experimento 1. Además el deslizamiento se fijará en un 0 % para esta serie de pruebas.

5.5.1. Resultados para la trayectoria recta

En las figuras 5.19, 5.21, 5.20 y 5.22 podemos ver los resultados obtenidos.

Podemos ver claramente en la figura 5.22 que para el modelo de medida de ángulos existen muchas singularidades que hacen que la estimación sea totalmente errónea, como es el caso del EKF o el UKF en dicha figura. Además si comparamos las figuras 5.19 y 5.20 vemos que los errores son más grandes para la función de medida de ángulos que para la función de medida de distancias y algunos casos es debido a las singularidades. El CKF ha sido el filtro que menor error ha tenido en la figura 5.19 mientras que para el modelo de medida de ángulos (figura 5.20) ha sido el UKF. Por último, como era de esperar el error en la localización tiende a aumentar a medida que aumenta el

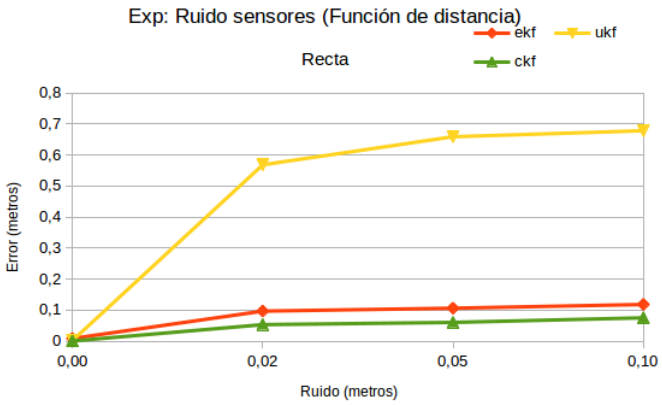

Figura 5.19. Experimento recta función de medida de distancia

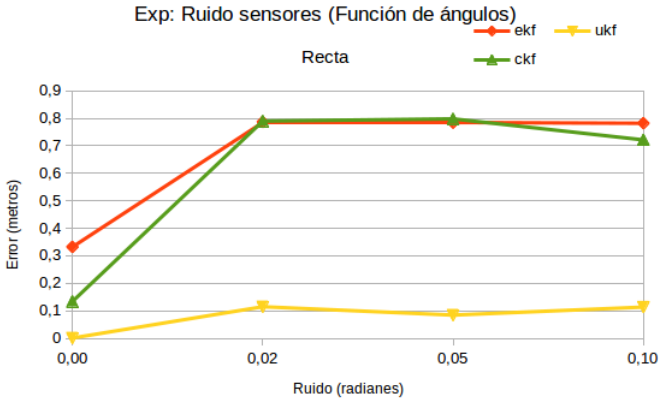

Figura 5.20. Experimento recta función de medida de ángulos

ruido presente en los sensores, lo cuál es lógico. También vemos que para el UKF es menor el error si la función de medida utilizada es la de ángulos, para el CKF y el EKF pasa totalmente lo contrario.

5.5.2. Resultados para la trayectoria cuadrada

Las figuras 5.23, 5.25, 5.24 y 5.26 muestran los resultados para esta serie de experimentos. La configuración utilizada ha sido la misma que para todas las pruebas hasta el momento.

En este caso los resultados se repiten con respecto al primer experimento de esta serie. Podemos ver que el UKF presenta errores mayores cuando usa la función de medida de distancia, además también podemos observar ciertas singularidades en la figura 5.26 provocadas por la función de medida de ángulos. En CKF ha sido el filtro que

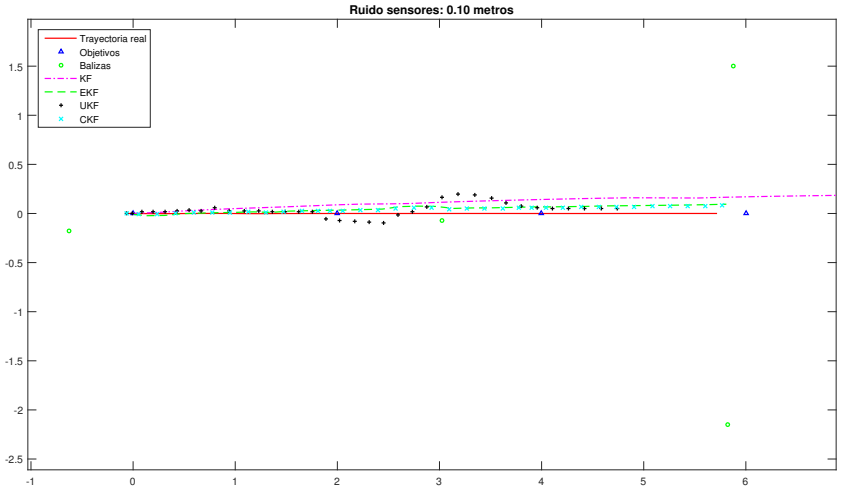

Figura 5.21. Trayectoria recta función de medida de distancia

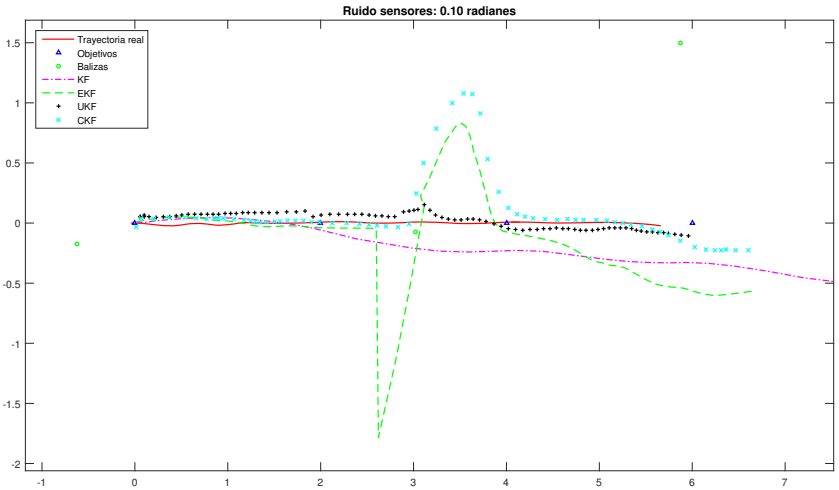

Figura 5.22. Trayectoria recta función de medida de ángulos

menor error ha presentado en general. En la figura 5.25 vemos como el error aumenta a medida que lo hace el ruido en los sensores, en cambio en la figura 5.26 vemos todo lo contrario y esto es debido a que probablemente para el primer experimento de la serie las singularidades hayan hecho aumentar el error obtenido. Aun con lo anterior para este caso la función de medida de distancias tiene un mejor comportamiento ya que los errores obtenidos de forma global son menores como vemos en la figura 5.23 en comparación con 5.24, por lo que se demuestra que la función de medida de distancias es más eficiente que la de medida de ángulos.

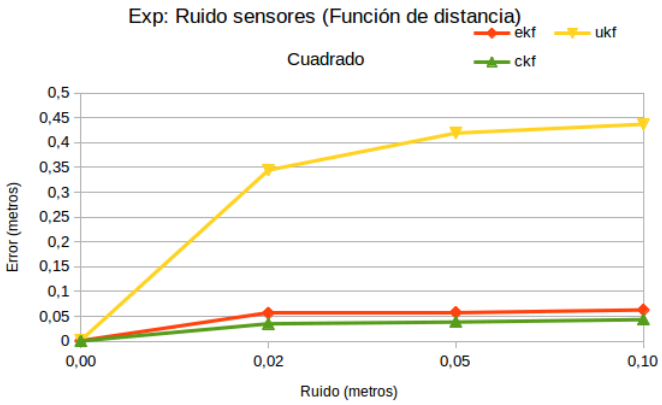

Figura 5.23. Experimento cuadrado función de medida de distancia

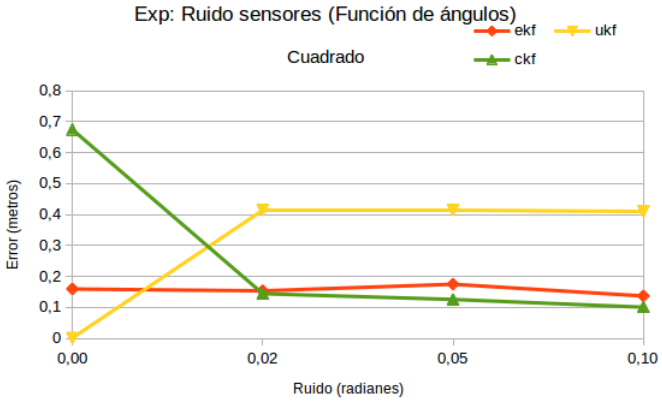

Figura 5.24. Experimento cuadrado función de medida de ángulos

5.5.3. Resultados para la trayectoria senoidal

Los resultados de esta serie de experimentos pueden verse en las figuras 5.27, 5.29, 5.28 y 5.30.

Este experimento ha sido el único en el que el error cometido por el UKF no ha sufrido variación en función del modelo de medida utilizado como podemos ver en las figuras 5.27 y 5.28. En esta prueba el filtro que ha presentado el menor error en general también ha sido el CKF. Por otro lado en este experimento vemos que los errores están muy cercanos entre sí, por lo que no hay gran diferencia entre usar un modelo de medida u otro. Sin embargo, en la figura 5.30 vemos la singularidad que se ha producido en el EKF y el CKF esta clase de efectos hacen que la estimación empeore y por lo tanto la localización sea menos acertada.

La conclusión para esta serie de experimentos es la siguiente : **El error en la esti-**

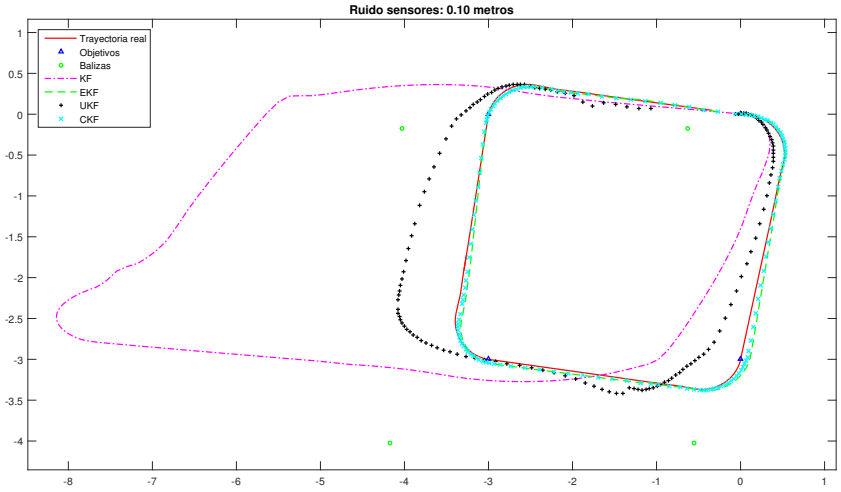

Figura 5.25. Trayectoria cuadrada función de medida de distancia

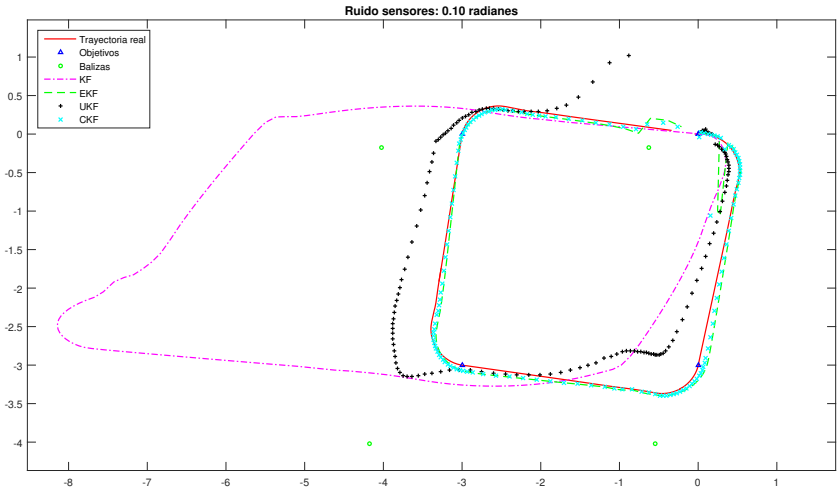

Figura 5.26. Trayectoria cuadrada función de medida de ángulos

mación aumenta a medida que aumenta el ruido presente en los sensores. El filtro que mejor se comporta ante este error es el CKF ya que por lo general es el que menor error cuadrático medio presenta. Por último, el modelo de medida de ángulos presenta mayores errores que el de distancia, además presenta singularidades que empeoran la estimación por lo que es mejor el modelo de medida de distancias.

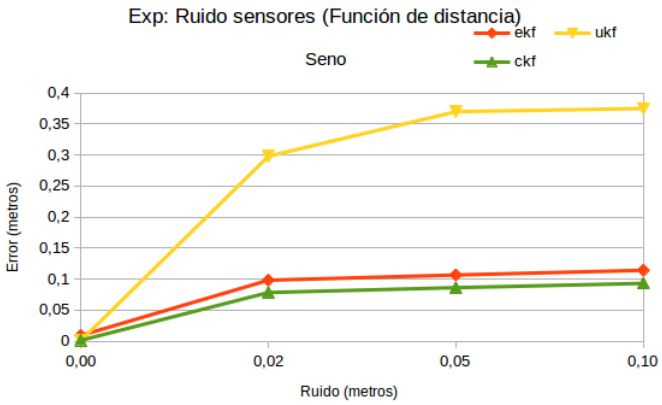

Figura 5.27. Experimento seno función de medida de distancia

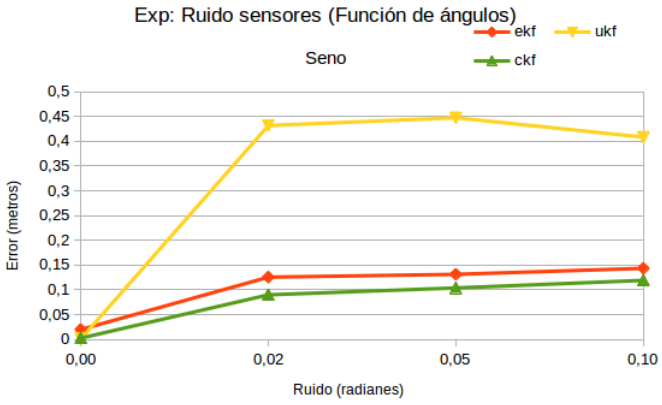

Figura 5.28. Experimento seno función de medida de ángulos

5.6. Experimentos 3: Variación del número de balizas

En esta sección experimentaremos para ver cómo afecta cómo afecta tener información más rica para el sensor de balizas. El modelo de medida que usaremos será el de distancia (ecuación 5.2) por ser el que menos error presenta. Como hasta ahora, representaremos la información obtenida por medio de gráficos que facilitarán la comprensión de los datos. Además representaremos las trayectorias seguidas por los filtros para el peor caso de estimación, que en el caso de esta serie de experimentos corresponde a la trayectoria con tres balizas disponibles.

5.6.1. Resultados para la trayectoria recta

Los resultados para estos experimentos se muestran en las figuras 5.31 y 5.32. Vemos

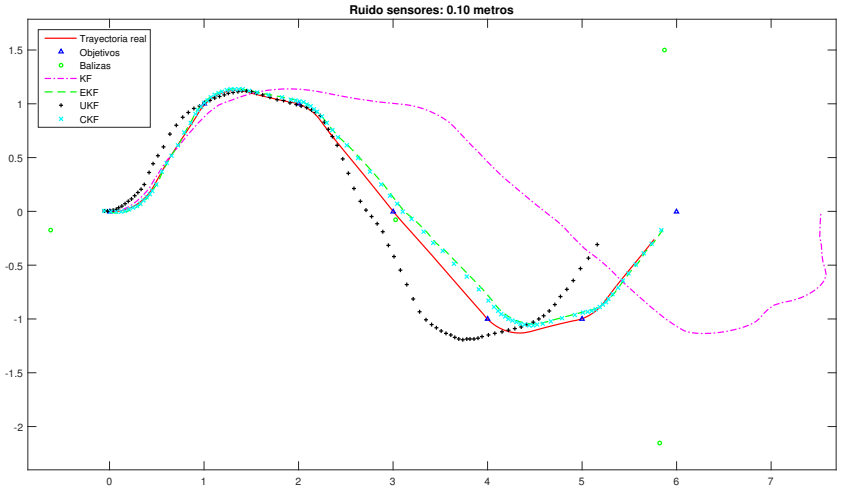

Figura 5.29. Trayectoria senoidal función de medida de distancia

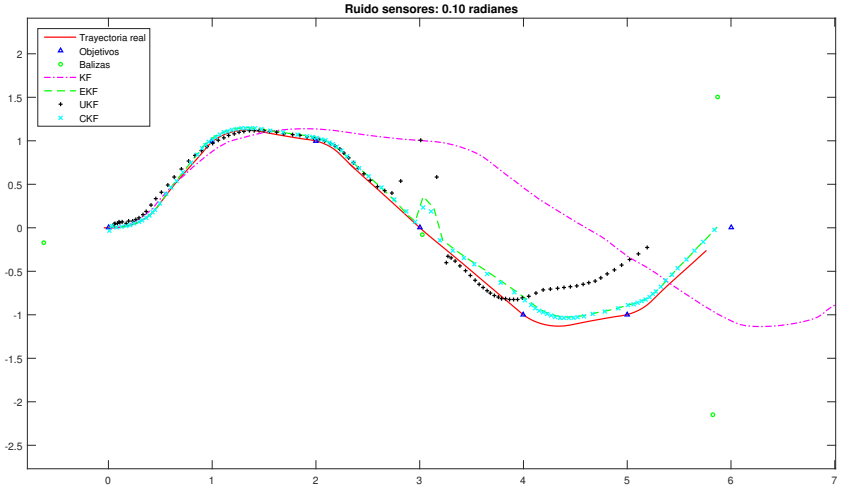

Figura 5.30. Trayectoria senoidal función de medida de ángulos

en la figura 5.31 como el error disminuye a medida que aumentamos el número de balizas en la escena, lo cual es una de las premisas del filtro de Kalman , cuanta más información mejor estimación. Además en la figura 5.32 podemos observar que cuando el EKF y CKF tienen poca información de las balizas, es decir tenemos pocas rodeando al robot, la estimación se vuelve totalmente errónea. Por esta razón en la figura 5.32 vemos como se producen unas grandes oscilaciones en la estimación. Hay que recordar que la trayectoria real mostrada en la figura difiere para cada una de las simulaciones de los filtros por lo que no hay que tomar esta como una referencia fiel de la realidad

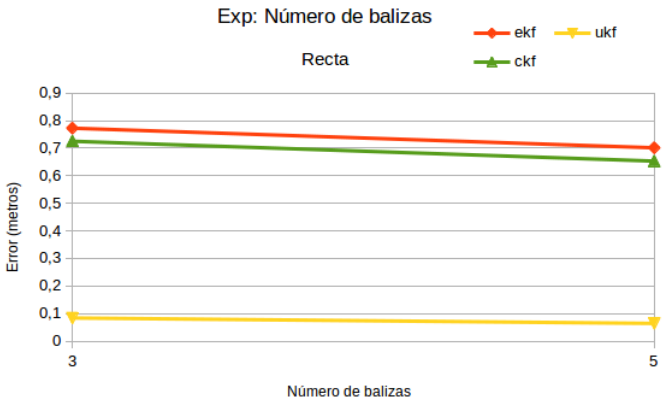

Figura 5.31. Experimento recta variando número de balizas

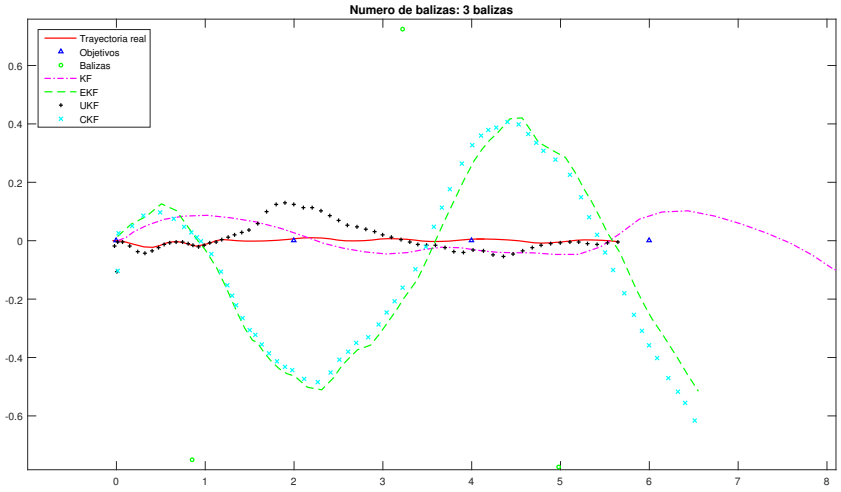

Figura 5.32. Trayectoria recta 3 balizas

sino como una representación orientativa.

5.6.2. Resultados para la trayectoria cuadrada

Los resultados para esta serie de pruebas se muestran en las figuras . Al igual que en el experimento anterior el error obtenido disminuye cuando aumentamos el número de balizas en la escena como vemos en la figura 5.33. En este caso en la figura 5.34 podemos ver que el filtro de Kalman clásico presenta mucho error aunque este detalle no es relevante porque como ya hemos dicho la sintonización de este filtro no depende del número de balizas sino que depende únicamente de la odometría. En esta figura,

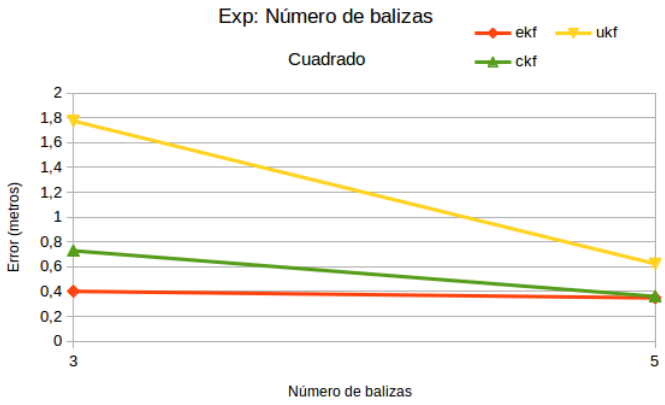

Figura 5.33. Experimento cuadrado variando número de balizas

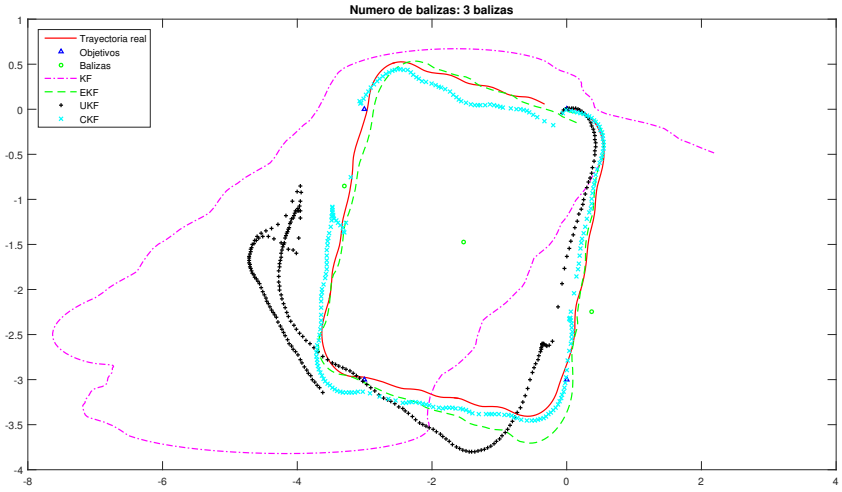

Figura 5.34. Trayectoria cuadrada 3 balizas

podemos ver como la estimación más acertada es la realizada por el CKF y EKF como también confirma la figura 5.33.

5.6.3. Resultados para la trayectoria senoidal

Los resultados de estas pruebas se muestran en las figuras 5.36 y 5.35 . Para esta serie de experimentos el error entre filtros es bastante parecido. Como vemos en la figura 5.35 el error también disminuye a medida que aumentamos la información sensorial, lo que ha sido un punto en común entre todas las pruebas realizadas. En la figura 5.36 podemos ver que las estimaciones de los filtros EKF, UKF y CKF son muy parecidas

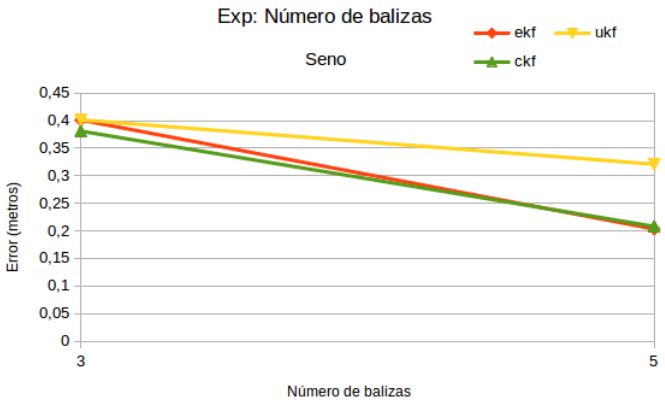

Figura 5.35. Experimento seno variando número de balizas

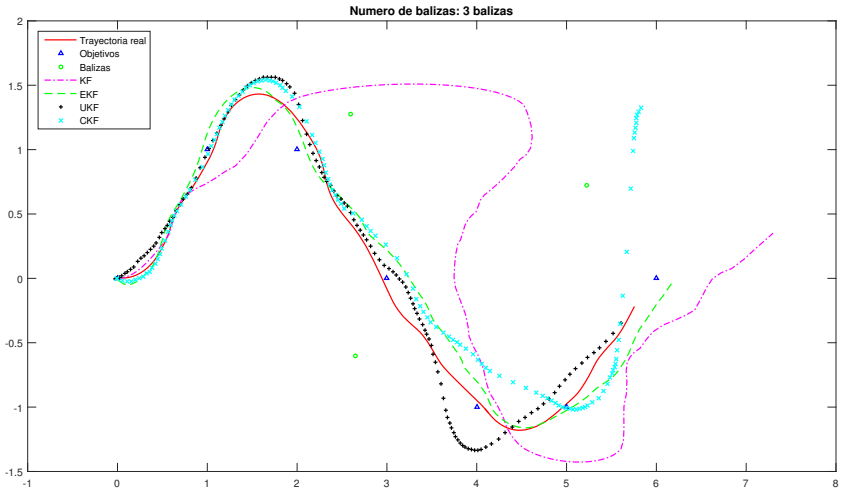

Figura 5.36. Trayectoria senoidal 3 balizas

entre sí. También podemos observar este hecho en la figura 5.35 donde vemos que los errores están próximos entre sí.

La conclusión para esta serie de experimentos es la siguiente : **El error en al estimación disminuye a medida que aumenta el número de balizas disponibles para tomar medidas. Los filtros por lo general se ven afectados en la misma medida por la falta de información sensorial. Si tuvieramos que destacar uno de ellos este sería el UKF el cual tiende a aumentar mucho su error cuando la información sensorial es pobre por otra parte el CKF es el que por lo general parece más robusto en cuanto a una disminución en la información sensorial.**

5.7. Experimento 4: Rendimiento ante una trayectoria general

En esta sección experimentaremos con la sintonización recomendada para los filtros para así ver como se comporta cada uno en unas condiciones normalizadas de funcionamiento. Usaremos el modelo de medida de distancia para este experimento ya que es el más robusto. Representaremos los resultados por medio de la tabla 5.5 para ver cual ha sido el error para cada filtro además de mostrar en las figuras con el resultado de la estimación para cada uno de los filtros utilizados. Hay que tener en cuenta que las figuras seleccionadas son las que representan el peor caso en la estimación y lo realmente relevante son las medias obtenidas para los experimentos mostradas en la tabla 5.5.

Tabla 5.5. Experimentos trayectoria general

Experimento 4						
Recorrido	Deslizamiento	Ruido sensores (m)	KF	EKF	UKF	CKF
Circuito	5,00 %	0,05	4.3	1.4	1.355	1.358

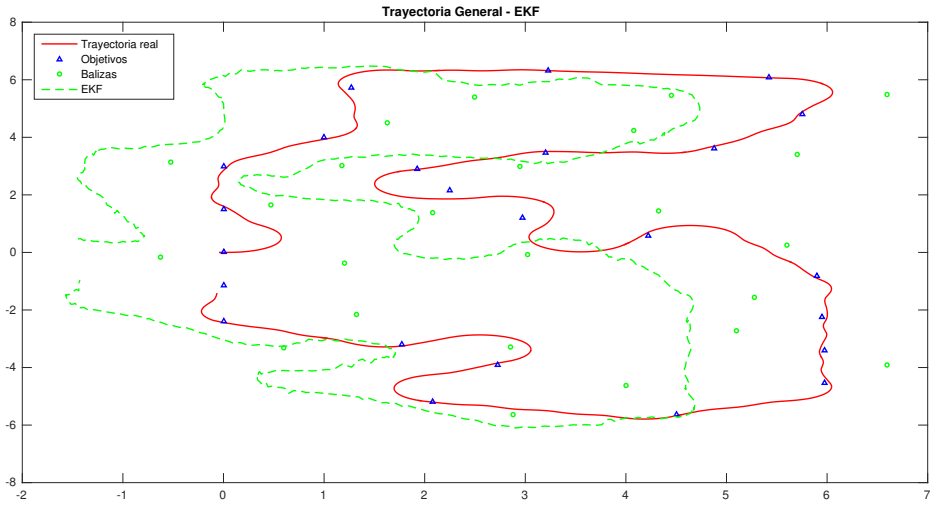

Figura 5.37. Experimento trayectoria general EKF

Podemos ver en la tabla 5.5 que el filtro que ha obtenido el menor error en su estimación ha sido el UKF seguido muy de cerca por el CKF. En las figuras 5.37, 5.38 y 5.39

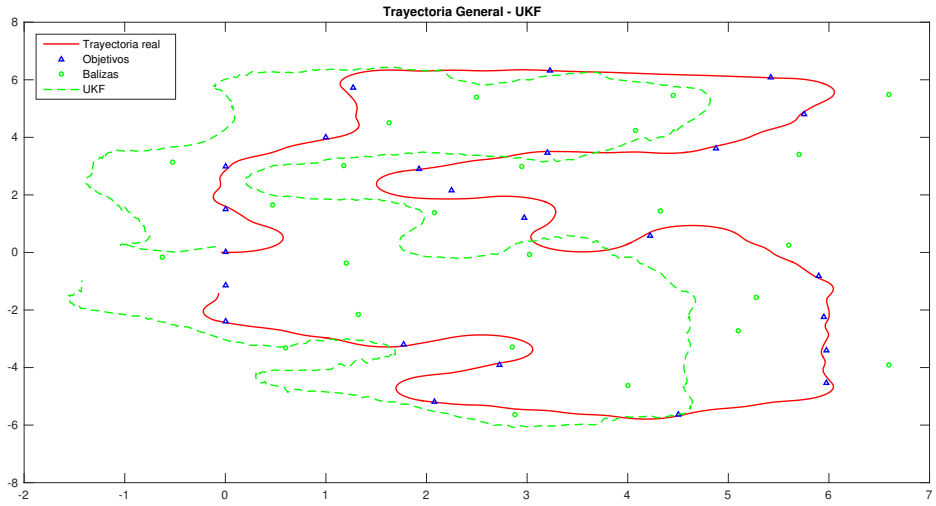

Figura 5.38. Experimento trayectoria general UKF

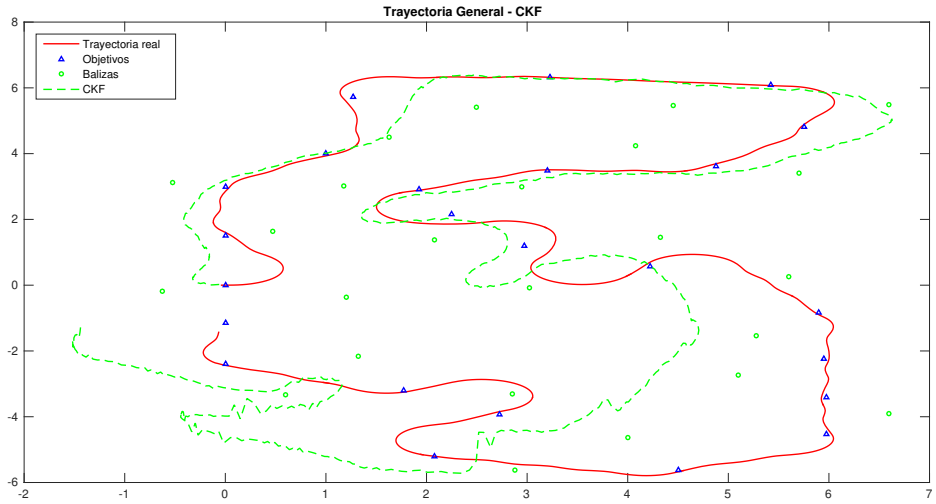

Figura 5.39. Experimento trayectoria general CKF

podemos apreciar las diferencias entre las trayectorias debido al deslizamiento existente, ya que hace que la odometría de una información que hace imprecisa la estimación. Como vemos en la comparación de estas figuras, la estimación de los filtros es muy parecida, por lo que se ven afectados en la misma media por el deslizamiento. Podemos decir es que el UKF ha sido el filtro que ha realizado la mejor estimación, aunque con muy poca diferencia sobre los demás como podemos ver en la tabla 5.5 y la figura 5.38. Por otra parte,como hemos dicho, en todos los filtros hemos implementado la

odometría por lo que quizás de no haber implementado esta información sensorial la estimación podría haber cambiado. Aunque lo anterior fuera posible, el objetivo de esta prueba es testear los filtros en una situación en la que estén configurados de manera que incluyan la fusión sensorial.

La conclusión para esta serie de experimentos es la siguiente : **El UKF ha sido el mejor adaptado a la trayectoria pero esto no quiere decir que sea el mejor filtro, como vemos los deslizamientos han generado que las trayectorias estimadas son bastante diferentes a las trayectorias reales debido a este motivo. Podemos concluir con este experimento que los deslizamientos pueden provocar muy malos resultados en la estimación y aunque el UKF muestre los mejores resultados el resto de filtros obtienen una estimación parecida. Sin embargo, el UKF parece el que mejor se adapta a una trayectoria con deslizamientos.**

5.8. Resultados generales

Una vez hemos realizado todos los experimentos, tenemos que llegar a una conclusión global sobre todas la implementaciones que hemos hecho y sobre todo debemos sacar conclusiones acerca de los resultados obtenidos para las pruebas. Para facilitar la toma de decisiones hemos decidido dar una puntuación a la posición que ha obtenido cada filtro en la realización de los experimentos. De esta manera estableceremos un ranking en el que posicionaremos cada uno de estos en orden y veremos como de cerca están unos de otros y además este nos permitirá tomar una decisión mucho más objetiva al respecto. Las puntuaciones las hemos asignado en función de la posición obtenida en la realización del experimento, es decir, si por ejemplo el EKF resulta hacer la mejor estimación en un experimento concreto se le sumarán 4 puntos, si es el segundo mejor se le sumarán 3 y así hasta llegar a 1 punto que corresponderá al filtro que quede en cuarto lugar. Podemos ver en la tabla 5.6 las puntuaciones obtenidas.

Como podemos observar el filtro que ha resultado tener el mejor desempeño general en todas la pruebas realizadas ha sido el CKF lo cual no es una sorpresa si pensamos que este filtro es una de las últimas modificaciones ideadas para el filtro de Kalman clásico y por lo tanto su funcionamiento se supone más óptimo. En segundo lugar tenemos al UKF, este ha quedado en esta posición debido muy probablemente a su poca robustez a la hora de trabajar con distintas funciones de medida y con poca información sensorial lo cual provoca que realice malas estimaciones. Por otro lado el UKF suele mostrarse muy robusto cuando trabaja con trayectorias en las que se sufren desli-

Tabla 5.6. Puntuaciones obtenidas por los filtros

Tabla de resultados																	
Experimento	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	**TOTAL**
KF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	16
EKF	2	2	2	4	2	2	3	2	3	4	3	3	2	4	3	2	43
UKF	4	3	4	2	4	4	2	4	2	2	2	2	4	2	2	4	47
CKF	3	4	3	3	3	3	4	3	4	3	4	4	3	3	4	3	54

zamientos. Al UKF lo sigue el EKF que en casi todos los experimentos se ha mantenido en la mitad de la clasificación, no es un filtro que haya destacado en nada positivamente pero tampoco lo ha hecho de manera negativa. En último lugar se encuentra el KF que como era de esperar sería el filtro que presentaría los peores resultados de los cuatro estudiados. Sin embargo, en las implementaciones que hemos realizado sobre el KF hemos visto que aunque su estimación es peor que en el caso de los otros tres filtros esta no suele cometer errores demasiado acusados cuando el deslizamiento es bajo. Por contra, como vimos en el experimento de la trayectoria completa, si el deslizamiento es grande la estimación del KF es totalmente errónea, comentiendo así un error inevitable para esta implementación del filtro.

Conclusions and future work

In this project we have developed a prototyping and simulation framework based on V-REP and MATLAB, which has demonstrated being a suitable combination of a dynamic robot simulator with a fast prototyping experimental environment. By using a toolbox with both well–known and state–of–the–art implementations of Kalman filter algorithms [8], as introduced in chapter 3, we were able to focus on designing and developing an experimental framework. Using such framework we conducted some experiments to assess the impact of different key concepts related with mobile robot localization, and the issues that classic and novel Kalman filter algorithms suffer.

Regarding the experiments, we have observed that the algorithm showing a better performance on average was the CKF, which has demonstrated being a state–of–the–art approach as shown in [26]. Furthermore, we have found that UKF is a very strong method in many situations, but with a big weakness regarding the singularities which appear with poor information from sensors. The EKF has also proven to be a filter with a decent behavior, and while not standing out over the others, its performance tend to be enough for most situations. Finally, the KF was the filter with the worst performance of the four algorithms studied, as expected. However, after the experiments conducted, we have seen that even if its estimates are worse that the ones of the other three filters, its performance is not too bad on situations with low slippery degree. On the contrary, as we saw in the experiment on a complex path, if the slippery degree is large, the estimation of the KF becomes totally wrong.

Although experiments have brought us some expected results, they have shed light on the actual version of the filter which would yield the best results, depending on the system we want to use with. We have seen that Gaussian filters and especially Kalman filters are very sensitive to modeling errors. For this reason, depending on the actual system which we are working with, we should apply a filter over others. It is clear that if our system is very simple, where an EKF would be enough, we should not waste resources trying to implement an UKF or an CKF. On the other hand, if we have a real system with high nonlinearities or a large number of states, the most suitable

implementation seems to be the CKF.

For the CKF and the UKF we have proposed in chapter 5 a modification on the odometry model. As we have seen in the experiments, the implementation of these modifications has been a success since filters have obtained lower error in the estimation.

Thus, as the final conclusions, firstly, the application of any of these state–of–the–art filters (EKF, UKF and CKF) depends on the complexity of our system, the modelization error of the disturbances, and the computational resources available. Secondly, the toolbox used in this project has shown to be a great starting point for implementing localization experiments for mobile robots using the Kalman filtering theory. Finally, the framework we have developed could be helpful for deploying simulated and real experiments to compare the algorithms described with other advanced localization algorithms, which could also be used for teaching purposes. The resulting software has been published as a public repository on GitHub [31].

Future work

It is obvious that using a simulator like V-REP eases the implementation. However, there are many possible improvements regarding the implementation of the functions, as for example the memory allocation to gain speed in executing some parts of the code. Furthermore, a faster implementation of the interaction with the simulator can be done in another programming language such as C++, Java, or Python, keeping the portability between different platforms. Another interesting point to invest additional efforts could be the development of a graphical interface in which simulation parameters can be tuned. Also, an improved representation of the results can be explored.

We could also consider the possibility of generating the position's objectives automatically, i.e. a random generator of each target. A simillar implementation for the landmark placement could be done.

From the model point of view, the API supports importing any type of model and the modification of its physical properties, e.g., the size of the wheels or the frame of the robot.

As we can see, there is much work ahead on this project, which can also deploy advanced methods like particle filters and many more algorithms for estimating the position of a mobile robot. The main idea after studying and implementing this Kalman filtering experimental framework is to allow for a platform which could be extended and become a tool to simulate different solutions to the problem of dynamic state esti-

mation.

Apéndice

A

Conceptos matemáticos para comprender el filtro de Kalman.

Para caracterizar el filtro de Kalman es necesario tener claras una serie de ideas o mejor dicho, conceptos matemáticos. Aunque el filtro tiene un desarrollo teórico-matemático bastante extenso nos quedaremos únicamente con las partes más básicas que nos ayudarán a comprender el funcionamiento es este en esencia. Para tener un trasfondo contrastado con respecto a las explicaciones teóricas, hemos usado varias referencias [32],[33],[3] y [34]. En todos ellos se introducen los conceptos matemáticos de los que hablamos.

A.1. Probabilidad de un evento.

La mayoría de nosotros tenemos alguna noción o idea de lo que significa un evento aleatorio, o la probabilidad de que un evento pueda ocurrir en un espacio experimental. Un experimento aleatorio se define como un experimento cuyo resultado es incierto, pero que se puede repetir (como podría ser el lanzamiento de un dado y que salga un número concreto). El conjunto de todos los posibles resultados de un experimento aleatorio se denomina espacio muestral, y se denomina a menudo con la siguiente notación Ω. Formalmente, el resultado para la probabilidad de un evento discreto (por ejemplo el lanzamiento de una moneda) a favor está definida de la siguiente manera (A.1) :

$$p(A) = \frac{\text{Posibles resultados favorables del evento A}}{\text{Número de todos los posibles resultados}} \tag{A.1}$$

En el caso de eventos dependientes la probabilidad de que ocurra A o B está dada por (A.2). Esta expresión también se conoce como la unión de A y B (se lee «A unión

B»). Una representación clara de lo que implica la unión puede verse en la figura A.1.

$$p(A \cup B) = p(A) + p(B) \tag{A.2}$$

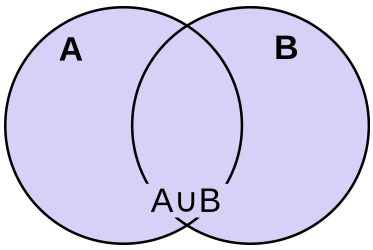

Figura A.1. Diagrama de Venn de A unión B.

Si la probabilidad de dos resultados es independiente, es decir que no afecta la una a la otra, entonces la probabilidad de que ambos ocurran está dada por (A.3). Esta operación se define como intersección de A y B (se lee «A intersección B») Un ejemplo claro de intersección es el lanzamiento de dos monedas. La probabilidad en un lanzamiento de que salga cara es de $1/2$, por lo tanto la probabilidad de que en las dos monedas salgan cara al mismo tiempo es de $1/4$ (claramente los resultados no se afectan entre sí ya que el resultado en uno de los lanzamientos no condiciona para nada al segundo de ellos)

$$p(A \cap B) = p(A) \cdot p(B) \tag{A.3}$$

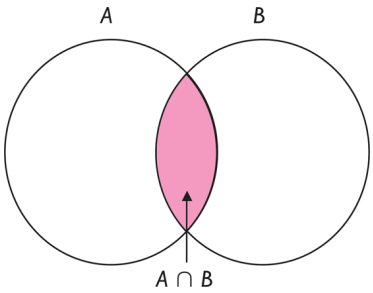

Figura A.2. Diagrama de Venn de A intersección B.

Finalmente, si A y B son dos sucesos, la probabilidad de que ocurra B dado A (es decir que A ocurre antes que B) se denota por p(B | A) y es llamada probabilidad condicional

y viene dada por (A.4).

$$p(A \mid B) = \frac{p(A \cap B)}{p(B)} \tag{A.4}$$

A.2. Variables Aleatorias.

En contraposición a los eventos discretos, en un caso el seguimiento y la captura de movimiento, estamos más interesados en la aleatoriedad asociada a suceso como pueden ser la variación de una señal eléctrica o al movimiento de una persona (son eventos que se dan en un espacio continuo de tiempo). En este caso podemos pensar en el elemento de interés como una variable aleatoria en un espacio continuo de valores como ya aclaramos anteriormente. Una variable aleatoria es básicamente una función que mapea todos los puntos en el espacio de muestra a los números reales. Por ejemplo, la variable aleatoria continua $X(t)$ que podría mapear tiempo con respecto a posición, en cualquier instante de tiempo t debería decirnos la posición esperada en dicho instante de tiempo. Las variables aleatorias pueden ser discretas, cuando se puede contar el conjunto de los resultados posibles, o continuas, cuando toma valores en una escala continua lo que significa que el número de resultados posibles es infinito.

Una definición más formal de todo lo enunciado anteriormente, es que una variable aleatoria es una función del espacio muestral Ω en el conjunto de los números reales. Las variables aleatorias se denominan generalmente X, Y, Z o con otra letra mayúscula escogida del del final del alfabeto. Por ejemplo,

$$\mathrm{X}{:}\Omega \rightarrow \mathrm{R}$$

define la variable aleatoria X como una función del espacio muestral en el conjunto de los números reales.

Las variables aleatorias se clasifican de acuerdo a su recorrido. Si el número de valores que puede tomar es finito o infinito numerable (tiene infinitos elementos que pueden ser numerados, como el conjunto de números enteros), X se denomina variable aleatoria discreta. Si el recorrido de X es en un conjunto de valores (por ejemplo, los valores de un intervalo de números reales), se denomina variable aleatoria continua.

En el caso de variables aleatorias continuas, la probabilidad de cualquier evento simple discreto A es cero, esto es, P(A)=0. Una función común que representa la probabilidad de una variable aleatoria, es definida como la función de distribución acumulativa.

Esta función (ecuación (A.5)) representa la probabilidad acumulativa de las variables aleatorias continuas X para todos (los no numerables) eventos hasta x.

$$F_x(x) = p(-\infty, x] \tag{A.5}$$

Algunas importantes características de las funciones de distribución acumulativa son:

$$F_x(x) \to 0 \text{ cuando x} \to -\infty \tag{A.6}$$

$$F_x(x) \to 1 \text{ cuando x} \to +\infty \tag{A.7}$$

$$F_x(x) \text{es una función no decreciente de x} \tag{A.8}$$

la expresión (A.5) es usada más comúnmente en su forma de su derivada y se conoce como la función de densidad de probabilidad (ecuación (A.9))

$$f_x(x) = \frac{\mathrm{d}}{\mathrm{dx}} F_x(x) \tag{A.9}$$

La función de la expresión (A.9) presenta las siguientes propiedades:

$$f_x(x) \text{es una función no negativa} \tag{A.10}$$

$$\int_{-\infty}^{+\infty} f_x(x)dx = 1 \tag{A.11}$$

Finalmente la probabilidad en la integral en cualquier intervalo [a,b] está definida como:

$$P(a < X < b) = \int_a^b f_x(x)dx = P(X < b) - P(X < a) \tag{A.12}$$

A.3. Media y varianza.

Al saber la distribución de una variable aleatoria tenemos un conocimiento total sobre ella. Sin embargo, en la práctica a menudo es imposible o innecesario conocer la distribución de probabilidad que describe un experimento aleatorio particular. En lugar de lo anterior, puede ser suficiente determinar unas pocas cantidades características, co-

mo el valor medio y una medida que describa la dispersión alrededor del valor medio (de esta forma se caracteriza en una distribución de probabilidad Normal). Muchos de nosotros estamos familiarizados con el concepto de media o promedio de un serie de números. La media para un espacio muestral N (se define como espacio muestral todos los posibles resultados de un experimento estadístico,como comentamos en la sección anterior) de una variable aleatoria X, el promedio, la media muestral o esperanza está dada por:

$$\bar{X} = \frac{X_1 + X_2 + \cdots + X_N}{N} \tag{A.13}$$

La media de una variable aleatoria X, se puede expresar como μ_X o simplemente como μ cuando sabemos a que variable aleatoria está referido. También es común referirse a la media como el valor esperado o esperanza, en cuyo caso se expresa como E(X).

Si X es una variable aleatoria discreta con distribución de probabilidad f(x) la media o valor esperado de X vendría dado de la siguiente forma:

$$\mu = E(X) = \sum_x x f(x) \tag{A.14}$$

En el caso de que X sea una variable aleatoria continua tendríamos la siguiente expresión:

$$\mu = E(X) = \int_{-\infty}^{+\infty} x f(x) dx \tag{A.15}$$

La varianza es un elemento de gran importancia que caracteriza la distribución de una variable aleatoria. Describe la amplitud del recorrido de la variable aleatoria. En otras palabras es un término que nos da información sobre la desviación o dispersión que hay con respecto a la media en las muestras usadas para calcular dicha media. Un ejemplo para explicar el concepto de varianza podría el ruido en una señal senoidal, en este caso la varianza nos daría una idea de la afectación del ruido en dicha señal. Para cualquier variable aleatoria X de media μ, la varianza σ^2 se define como:

$$\sigma^2 = var(X) = E(X - \mu)^2 \tag{A.16}$$

En el caso de que X sea una variable aleatoria discreta, entonces la varianza sería:

$$\sigma^2 = var(X) = \sum_x (x - \mu)^2 f(x) \tag{A.17}$$

Por otro lado en el caso de que X sea una variable aleatoria continua, la varianza se definiería de la siguiente manera:

$$\sigma^2 = var(X) = \int_{-\infty}^{+\infty} (x-\mu)^2 f(x) dx \tag{A.18}$$

En el caso de que X sea continua la raíz cuadrada de la varianza $\sqrt{\sigma^2} = \sigma$, se conoce como la desviación típica o estándar de X.

La varianza en una propiedad estadística muy usada en las señales aleatorias, ya que si se supiera la varianza de una señal que de otro modo fuera supuesta para ser constante alrededor de un valor, la magnitud de la varianza daría una idea de cuanto nos estamos desviando con respecto a la media. En el filtro de Kalman usaremos la varianza con el fin de saber cuanto ruido hay en nuestra medida o como de deslocalizados nos encontramos en el mapa, este asunto lo trataremos en secciones posteriores cuando hablemos de la propagación del error.

A.4. Distribución de probabilidad Normal o Gaussiana

Esta distribución de probabilidad ha tenido un uso muy extendido en el modelado de sistemas aleatorios por muchas razones. Introduciremos algunas de ellas para poder caracterizarla. Muchos procesos de la naturaleza podrían asemejarse a una distribución normal o estar muy cercana a esta, es decir que se comportan siguiendo una distribución de probabilidad que bien podría ser una normal. De hecho, bajo unas condiciones concretas, se puede demostrar que una suma de variables aleatorias con cualquier distribución tiende hacia una distribución normal o de Gauss. El teorema que define esta propiedad es conocido como el teorema del límite central, y es muy útil para ciertas aplicaciones. La distribución Normal o Gaussiana que podemos ver en la figura A.3 tiene algunas propiedades que la hacen matemáticamente muy manejable y es uno de los ejes principales sobre los que se postuló el filtro clásico de Kalman.

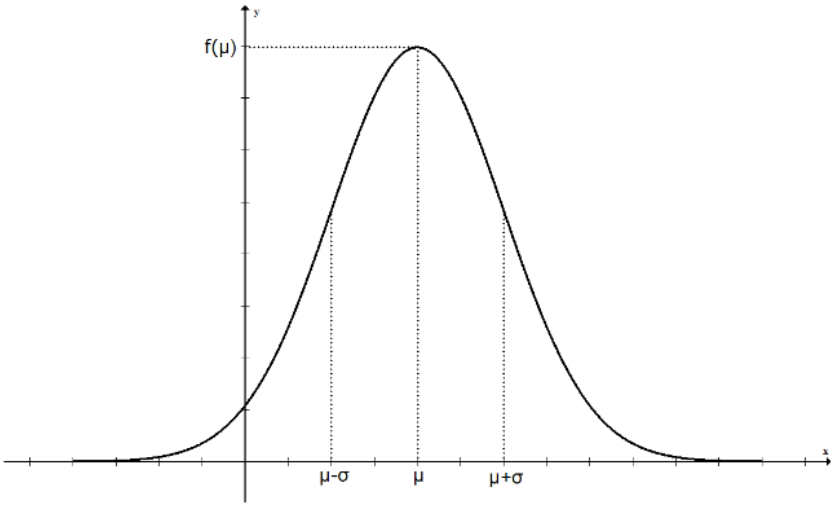

Figura A.3. Función de distribución de probabilidad Normal o Gaussiana.

La función de densidad de probabilidad de una variable aleatoria X con distribución normal de parámetros μ y σ se define matemáticamente como:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}, -\infty < x < \infty \tag{A.19}$$

La notación típica para representar los valores de densidad de una variable aleatoria X con media μ y desviación típica σ es $\mathcal{N}$(X;μ,σ) aunque en algunos textos también se usa la varianza(no implica ningún problema ya que esta se deriva de la desviación típica). Algunas propiedades de la distribución normal son las siguientes:

1. f(x) es simétrica respecto a x = μ.
2. El máximo de f(x) está en x = μ.
3. Los puntos de inflexión de f(x) están en x = μ + σ y x = μ - σ.

Como f(x) es una función de densidad cumple las siguientes propiedades:

$$f(x) \geq 0 \tag{A.20}$$

$$\int_{-\infty}^{+\infty} f(x)dx = 1 \tag{A.21}$$

$$\mu = \int_{-\infty}^{+\infty} xf(x)dx \tag{A.22}$$

$$P(a \geq X \geq b) = \int_{a}^{b} \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{A.23}$$

La propiedad de la ecuación (A.23) se da si una cantidad X está normalmente distribuida con parámetros μ y σ, en caso contrario no se cumpliría la igualdad ya que no estaríamos tratando con una distribución de probabilidad gaussiana para la cual se define la expresión.

A.5. Independencia y Probabilidad condicional

Podemos ver que en las ecuaciones (A.2) y (A.4), que la probabilidad ha sido definida para variables aleatorias continuas. Dos variables aleatorias continuas X e Y son independientes estadísticamente si la probabilidad conjunta $f_{xy}(x, y)$ es igual a la suma de sus probabilidades marginales, es decir, que se consideran independientes si que cumple lo siguiente:

$$f_{xy}(x, y) = f_x(x) f_y(y) \longrightarrow P(X \cap Y) \tag{A.24}$$

Un ejemplo algo más claro para ilustrar la independencia sería suponer que lanzamos una moneda dos veces y que dicha moneda no está trucada. Sea A el suceso de que en el primer lanzamiento salga cara y B el suceso de que en el segundo lanzamiento salga cara. Supongamos que A ocurre ¿Cambia eso la probabilidad de que B ocurra?La respuesta lógica que todos daríamos es que no y efectivamente así es. Sabemos que el resultado del lanzamiento de la primera moneda no influye en el resultado del segundo lanzamiento. En el caso de que dos eventos se condicionen entre sí en el cálculo de la probabilidad (sacar bolas de colores de una bolsa) podríamos expresarla de forma matemática. Esta idea se puede expresar matemáticamente usando el concepto de probabilidad condicional que veíamos al principio de esta sección en la ecuación (A.4) donde expresaríamos la probabilidad de que un evento ocurra cuando previamente ha ocurrido otro.

Otro concepto muy relevante a la hora de hablar de probabilidad condicional es el teorema de Bayes. Para ilustrar la motivación de esta expresión nos remitiremos a un ejemplo expuesto en [32]. Imaginemos que calculamos la probabilidad de que el resultado de un test de VIH realizado sobre un individuo escogido aleatoriamente sea positivo. Para el individuo, es mucho más importante saber si el resultado positivo del test realmente significa que está infectado por el virus. Las probabilidades estaban definidas de la siguiente manera:

A = El resultado del test es positivo.

B_1 = La persona está infectada.

B_2 = La persona no está infectada.

Estamos interesados en P(B_1| A), es decir, la probabilidad de que una persona esté infectada sabiendo que el resultado es positivo. Las probabilidades P(A| B_1) y P(A| B_2) se deducen directamente de las características del test. Ahora queremos calcular las probabilidades condicionales cuando los papeles de A y B_1 se invierten.

Antes de calcular la probabilidad en este ejemplo específico, consideraremos el caso más general. Supongamos que los conjuntos B_1,B_2,. . . ,B_n forman una partición del espacio muestral Ω, A es un suceso y las probabilidades P(A | B_i), donde i=1,2,. . . ,n son conocidas. Lo que queremos calcular son las probabilidades P(A | B_i). Para la realización del cálculo se procedería de la siguiente manera:

Utilizando la definición de probabilidades condicionales tenemos:

$$P(B_i \mid A) = \frac{P(A \cap B_i)}{P(A)} \tag{A.25}$$

Para calcular $P(A \cap B_i)$, ahora se condiciona sobre B_i :

$$P(A \cap B_i) = P(A \mid B_i)P(B_i) \tag{A.26}$$

Por otro lado para evaluar el denominador P(A) se utiliza la ley de la probabilidad total:

$$P(A) = \sum_{j=1}^{n} P(A \mid B_j)P(B_j) \tag{A.27}$$

Combinando las expresiones (A.25), (A.26) y (A.27) se llega a la siguiente expresión, conocida como fórmula de Bayes o teorema de Bayes. Formalmente, siendo B_1,B_2,. . . ,B_n una partición de Ω y A un suceso. Entonces

$$P(B_i \mid A) = \frac{P(A \mid B_i P(B_i}{\sum_{j=1}^{n} P(A \mid B_j)P(B_j)} \tag{A.28}$$

Volviendo al ejemplo, si deseamos calcular P($B_1 \mid A$), que es la probabilidad de que una persona esté infectada dado un resultado positivo en el test. Una vez el test ha dado positivo la población se divide en dos conjuntos, uno denominado B_1 en el que la persona está infectada y otro B_2 en el que la persona no está infectada. Si usamos la expresión (A.28) que es el teorema de Bayes, tenemos:

$$P(B_1 \mid A) = \frac{P(A \mid B_1 P(B_1}{P(A \mid B_1)P(B_1) + P(A \mid B_2)P(B_2)} \tag{A.29}$$

De esta manera calcularíamos la probabilidad de que una persona se encuentre in-

fectada sabiendo que el test ha resultado positivo.

Con todos estos conceptos ya tendríamos los suficientes conocimientos como para comprender la base matemática básica sobre la que se formulan los filtros Bayesianos y por supuesto los filtros de Kalman. El resto de formulaciones matemáticas relacionadas con los filtros Bayesianos las encontraremos en el capítulo 2 junto con su explicación teórica.

Apéndice B

Toolbox utilizada en Matlab.

Para la realización de este trabajo hemos utilizado una toolbox de MATLAB realizada por Jouni Hartikainen, Arno Solin y Simo Särkkä. El objetivo de esta toolbox es implementar todas las funciones vistas en el capítulo 3 de este trabajo y muchas otras. Para una mayor comprensión sobre la propia toolbox y los códigos que se presentan en algunos capítulos de este trabajo introduciremos las funciones que hemos utilizado (únicamente las básicas sobre los filtros), para ello introduciremos los segmentos de código de MATLAB con los parámetros de cada función explicados. Todas aquellas que funciones que son llamadas dentro de la implementación de los filtros no las incluiremos en este apéndice ya que haría su lectura demasiado engorrosa. Para encontrar más información acerca del resto de funciones puede consultarse [8], donde encontrarán la toolbox para su descarga además de su documentación.

B.1. Filtro clásico de Kalman

B.1.1. Ciclo de predicción

```
1 %KF_PREDICT  Perform Kalman Filter prediction step
2% Syntax:
3%   [X,P] = KF_PREDICT(X,P,A,Q,B,U)
4% In:
5%   X - Nx1 mean state estimate of previous step
6%   P - NxN state covariance of previous step
7%   A - Transition matrix of discrete model (optional, default identity)
8%   Q - Process noise of discrete model (optional, default zero)
9%   B - Input effect matrix (optional, default identity)
10%  U - Constant input (optional, default empty)
11%
12% Out:
13%   X - Predicted state mean
```

```
14%   P - Predicted state covariance
15% Description:
16%   Perform Kalman Filter prediction step. The model is
17%     x[k] = A*x[k-1] + B*u[k-1] + q,   q ~ N(0,Q).
18%   The predicted state is distributed as follows:
19%     p(x[k] | x[k-1]) = N(x[k] | A*x[k-1] + B*u[k-1], Q[k-1])
20%   The predicted mean x-[k] and covariance P-[k] are calculated
21%   with the following equations:
22%     m-[k] = A*x[k-1] + B*u[k-1]
23%     P-[k] = A*P[k-1]*A' + Q.
24%   If there is no input u present then the first equation reduces to
25%     m-[k] = A*x[k-1]
26function [x,P] = kf_predict(x,P,A,Q,B,u)
27  %Check arguments
28  %
29  if nargin < 3
30    A = [];
31  end
32  if nargin < 4
33    Q = [];
34  end
35  if nargin < 5
36    B = [];
37  end
38  if nargin < 6
39    u = [];
40  end
41  %
42  %Apply defaults
43  %
44  if isempty(A)
45    A = eye(size(x,1));
46  end
47  if isempty(Q)
48    Q = zeros(size(x,1));
49  end
50  if isempty(B) & ~isempty(u)
51    B = eye(size(x,1),size(u,1));
52  end
53  %
54  %Perform prediction
55  %
56  if isempty(u)
```

```
57      x = A * x;
58      P = A * P * A' + Q;
59    else
60      x = A * x + B * u;
61      P = A * P * A' + Q;
62    end
```

B.1.2. Ciclo de actualización

```
1 %KF_UPDATE  Kalman Filter update step
2 % Syntax:
3 %   [X,P,K,IM,IS,LH] = KF_UPDATE(X,P,Y,H,R)
4 % In:
5 %   X - Nx1 mean state estimate after prediction step
6 %   P - NxN state covariance after prediction step
7 %   Y - Dx1 measurement vector.
8 %   H - Measurement matrix.
9 %   R - Measurement noise covariance.
10 % Out:
11 %   X  - Updated state mean
12 %   P  - Updated state covariance
13 %   K  - Computed Kalman gain
14 %   IM - Mean of predictive distribution of Y
15 %   IS - Covariance or predictive mean of Y
16 %   LH - Predictive probability (likelihood) of measurement.
17 % Description:
18 %   Kalman filter measurement update step. Kalman Filter
19 %   model is
20 %     x[k] = A*x[k-1] + B*u[k-1] + q,  q ~ N(0,Q)
21 %     y[k] = H*x[k] + r,r ~ N(0,R)
22 %   Prediction step of Kalman filter computes predicted
23 %   mean m-[k] and covariance P-[k] of state:
24 %     p(x[k] | y[1:k-1]) = N(x[k] | m-[k], P-[k])
25 function [X,P,K,IM,IS,LH] = kf_update(X,P,y,H,R)
26   % Check which arguments are there
27   %
28   if nargin < 5
29     error('Too few arguments');
30   end
31   %
32   % update step
33   %
```

```
34  IM = H*X;
35  IS = (R + H*P*H');
36  K = P*H'/IS;
37  X = X + K * (y-IM);
38  P = P - K*IS*K';
39  if nargout > 5
40    LH = gauss_pdf(y,IM,IS);
41  end
```

B.2. Filtro extendido de Kalman de primer orden

B.2.1. Ciclo de predicción

```
1 %EKF_PREDICT1  1st order Extended Kalman Filter prediction step
2% Syntax:
3%   [M,P] = EKF_PREDICT1(M,P,[A,Q,a,W,param])
4% In:
5%   M - Nx1 mean state estimate of previous step
6%   P - NxN state covariance of previous step
7%   A - Derivative of a() with respect to state as
8%       matrix, inline function, function handle or
9%       name of function in form A(x,param)(optional, default eye())
10%  Q - Process noise of discrete model(optional, default zero)
11%  a - Mean prediction E[a(x[k-1],q=0)] as vector,
12%       inline function, function handle or name
13%       of function in form a(x,param)(optional, default A(x)*X)
14%  W - Derivative of a() with respect to noise q
15%       as matrix, inline function, function handle
16%       or name of function in form W(x,param)(optional, default identity)
17%  param - Parameters of a (optional, default empty)
18% Out:
19%  M - Updated state mean
20%  P - Updated state covariance
21% Description:
22%  Perform Extended Kalman Filter prediction step.
23function [M,P] = ekf_predict1(M,P,A,Q,a,W,param)
24  %Check arguments
25  %
26  if nargin < 3
27    A = [];
28  end
```

```
29 if nargin < 4
30   Q = [];
31 end
32  if nargin < 5
33   a = [];
34 end
35 if nargin < 6
36   W = [];
37 end
38 if nargin < 7
39   param = [];
40 end
41 %Apply defaults
42 if isempty(A)
43   A = eye(size(M,1));
44 end
45 if isempty(Q)
46   Q = zeros(size(M,1));
47 end
48 if isempty(W)
49   W = eye(size(M,1),size(Q,2));
50 end
51 if isnumeric(A)
52   %nop
53 elseif isstr(A) | strcmp(class(A),'function_handle')
54   A = feval(A,M,param);
55 else
56   A = A(M,param);
57 end
58 %Perform prediction
59 if isempty(a)
60   M = A*M;
61 elseif isnumeric(a)
62   M = a;
63 elseif isstr(a) | strcmp(class(a),'function_handle')
64   M = feval(a,M,param);
65 else
66   M = a(M,param);
67 end
68 if isnumeric(W)
69   %nop
70 elseif isstr(W) | strcmp(class(W),'function_handle')
71   W = feval(W,M,param);
```

```
72  else
73    W = W(M,param);
74  end
75
76  P = A * P * A' + W * Q * W';
```

B.2.2. Ciclo de actualización

```
1 %EKF_UPDATE1  1st order Extended Kalman Filter update step
2 % Syntax:
3 %   [M,P,K,MU,S,LH] = EKF_UPDATE1(M,P,Y,H,R,[h,V,param])
4 % In:
5 %   M - Nx1 mean state estimate after prediction step
6 %   P - NxN state covariance after prediction step
7 %   Y - Dx1 measurement vector.
8 %   H - Derivative of h() with respect to state as matrix,
9 %       inline function, function handle or name
10%       of function in form H(x,param)
11%   R - Measurement noise covariance.
12%   h - Mean prediction (innovation) as vector,
13%       inline function, function handle or name
14%       of function in form h(x,param).(optional, default H(x)*X)
15%   V - Derivative of h() with respect to noise as matrix,
16%       inline function, function handle or name
17%       of function in form V(x,param).(optional, default identity)
18%   param - Parameters of h (optional, default empty)
19% Out:
20%   M - Updated state mean
21%   P - Updated state covariance
22%   K - Computed Kalman gain
23%   MU - Predictive mean of Y
24%   S - Predictive covariance of Y
25%   LH - Predictive probability (likelihood) of measurement.
26% Description:
27%   Extended Kalman Filter measurement update step.
28%   EKF model is
29%     y[k] = h(x[k],r), r ~ N(0,R)
30function [M,P,K,MU,S,LH] = ekf_update1(M,P,y,H,R,h,V,param)
31  %Check which arguments are there
32  if nargin < 5
33    error('Too few arguments');
34  end
```

```
35  if nargin < 6
36    h = [];
37  end
38  if nargin < 7
39    V = [];
40  end
41  if nargin < 8
42    param = [];
43  end
44  % Apply defaults
45  %
46  if isempty(V)
47    V = eye(size(R,1));
48  end
49  % Evaluate matrices
50  %
51  if isnumeric(H)
52     % nop
53  elseif isstr(H) | strcmp(class(H),'function_handle')
54    H = feval(H,M,param);
55  else
56    H = H(M,param);
57  end
58  if isempty(h)
59    MU = H*M;
60  elseif isnumeric(h)
61    MU = h;
62  elseif isstr(h) | strcmp(class(h),'function_handle')
63    MU = feval(h,M,param);
64  else
65    MU = h(M,param);
66  end
67  if isnumeric(V)
68     % nop
69  elseif isstr(V) | strcmp(class(V),'function_handle')
70    V = feval(V,M,param);
71  else
72    V = V(M,param);
73  end
74  % update step
75  %
76  S = (V*R*V' + H*P*H');
77  K = P*H'/S;
```

```
78  M = M + K * (y-MU);
79  P = P - K*S*K';
80
81  if nargout > 5
82    LH = gauss_pdf(y,MU,S);
83  end
```

B.3. Filtro *unscented* de Kalman *NonAugmented*

B.3.1. Ciclo de predicción

```
1 %UKF_PREDICT1  Nonaugmented (Additive) UKF prediction step
2 % Syntax:
3 %   [M,P] = UKF_PREDICT1(M,P,f,Q,f_param,alpha,beta,kappa,mat)
4 % In:
5 %   M - Nx1 mean state estimate of previous step
6 %   P - NxN state covariance of previous step
7 %   f - Dynamic model function as a matrix A defining
8 %       linear function a(x) = A*x, inline function,
9 %       function handle or name of function in
10%       form a(x,param)(optional, default eye())
11%   Q - Process noise of discrete model(optional, default zero)
12%   f_param - Parameters of f (optional, default empty)
13%   alpha - Transformation parameter(optional)
14%   beta  - Transformation parameter(optional)
15%   kappa - Transformation parameter(optional)
16%   mat   - If 1 uses matrix form (optional, default 0)
17% Out:
18%   M - Updated state mean
19%   P - Updated state covariance
20% Description:
21%   Perform additive form Unscented Kalman Filter prediction step.
22%   Function a should be such that it can be given
23%   DxN matrix of N sigma Dx1 points and it returns
24%   the corresponding predictions for each sigma
25%   point.
26function [M,P,D] = ukf_predict1(M,P,f,Q,f_param,alpha,beta,kappa,mat)
27  %Check which arguments are there
28  if nargin < 2
29    error('Too few arguments');
30  end
```

```
31  if nargin < 3
32    f = [];
33  end
34  if nargin < 4
35    Q = [];
36  end
37  if nargin < 5
38    f_param = [];
39  end
40  if nargin < 6
41    alpha = [];
42  end
43  if nargin < 7
44    beta = [];
45  end
46  if nargin < 8
47    kappa = [];
48  end
49  if nargin < 9
50    mat = [];
51  end
52  % Apply defaults
53  %
54  if isempty(f)
55    f = eye(size(M,1));
56  end
57  if isempty(Q)
58    Q = zeros(size(M,1));
59  end
60  if isempty(mat)
61    mat = 0;
62  end
63  % Do transform
64  % and add process noise
65  tr_param = {alpha beta kappa mat};
66  [M,P,D] = ut_transform(M,P,f,f_param,tr_param);
67  P = P + Q;
```

B.3.2. Ciclo de actualización

```
1 %UKF_UPDATE1 - Additive form Unscented Kalman Filter update step
2 % Syntax:
```

```
3%   [M,P,K,MU,S,LH] = UKF_UPDATE1(M,P,Y,h,R,param,alpha,beta,kappa,mat)
4% In:
5%   M - Mean state estimate after prediction step
6%   P - State covariance after prediction step
7%   Y - Measurement vector.
8%   h - Measurement model function as a matrix H defining
9%       linear function h(x) = H*x, inline function,
10%      function handle or name of function in
11%      form h(x,param)
12%  R - Measurement covariance.
13%  h_param - Parameters of h (optional, default empty)
14%  alpha - Transformation parameter (optional)
15%  beta  - Transformation parameter (optional)
16%  kappa - Transformation parameter (optional)
17%  mat   - If 1 uses matrix form (optional, default 0)
18% Out:
19%  M - Updated state mean
20%  P - Updated state covariance
21%  K - Computed Kalman gain
22%  MU - Predictive mean of Y
23%  S - Predictive covariance Y
24%  LH - Predictive probability (likelihood) of measurement.
25% Description:
26%  Perform additive form Discrete Unscented Kalman Filter (UKF)
27%  measurement update step. Assumes additive measurement
28%  noise.
29%  Function h should be such that it can be given
30%  DxN matrix of N sigma Dx1 points and it returns
31%  the corresponding measurements for each sigma
32%  point. This function should also make sure that
33%  the returned sigma points are compatible such that
34%  there are no 2pi jumps in angles etc.
35% Example:
36%  h = inline('atan2(x(2,:)-s(2),x(1,:)-s(1))','x','s');
37%  [M2,P2] = ukf_update(M1,P1,Y,h,R,S);
38% See also:
39%  UKF_PREDICT1, UKF_PREDICT2, UKF_UPDATE2, UKF_PREDICT3, UKF_UPDATE3,
40%  UT_TRANSFORM, UT_WEIGHTS, UT_MWEIGHTS, UT_SIGMAS
41function [M,P,K,MU,S,LH] = ukf_update1(M,P,Y,h,R,h_param,alpha,beta,kappa,
    mat)
42  %
43  %Check that all arguments are there
44  if nargin < 5
```

```
45    error('Too few arguments');
46  end
47  if nargin < 6
48    h_param = [];
49  end
50  if nargin < 7
51    alpha = [];
52  end
53  if nargin < 8
54    beta = [];
55  end
56  if nargin < 9
57    kappa = [];
58  end
59  if nargin < 10
60    mat = [];
61  end
62  %Apply defaults
63  %
64  if isempty(mat)
65    mat = 0;
66  end
67  %Do transform and make the update
68  %
69  tr_param = {alpha beta kappa mat};
70  [MU,S,C] = ut_transform(M,P,h,h_param,tr_param);
71  S = S + R;
72  K = C / S;
73  M = M + K * (Y - MU);
74  P = P - K * S * K';
75  if nargout > 5
76    LH = gauss_pdf(Y,MU,S);
77  end
```

B.4. Filtro de Kalman de Cubatura

B.4.1. Ciclo de predicción

```
1function [M,P] = ckf_predict(M,P,f,Q,f_param)
2%CKF_PREDICT - Cubature Kalman filter prediction step
3%Syntax:
```

```
4%   [M,P] = CKF_PREDICT(M,P,[f,Q,f_param])
5% In:
6%   M - Nx1 mean state estimate of previous step
7%   P - NxN state covariance of previous step
8%   f - Dynamic model function as a matrix A defining
9%       linear function f(x) = A*x, inline function,
10%      function handle or name of function in
11%      form f(x,param)(optional, default eye())
12%  Q - Process noise of discrete model (optional, default zero)
13%  f_param - Parameters of f (optional, default empty)
14% Out:
15%  M - Updated state mean
16%  P - Updated state covariance
17% Description:
18%  Perform additive form spherical-radial cubature Kalman filter (CKF)
19%  prediction step.
20%
21%  Function f(.) should be such that it can be given a
22%  DxN matrix of N sigma Dx1 points and it returns
23%  the corresponding predictions for each sigma
24%  point.
25  %Check which arguments are there
26  if nargin < 2
27    error('Too few arguments');
28  end
29  if nargin < 3
30    f = [];
31  end
32  if nargin < 4
33    Q = [];
34  end
35  %Apply defaults
36  if isempty(f)
37    f = eye(size(M,1));
38  end
39  if isempty(Q)
40    Q = zeros(size(M,1));
41  end
42  %Do transform and add process noise
43  if nargin < 5
44    [M,P] = ckf_transform(M,P,f);
45  else
46    [M,P] = ckf_transform(M,P,f,f_param);
```

```
47  end
48  P = P + Q;
```

B.4.2. Ciclo de actualización

```
1
2function [M,P,K,MU,S,LH] = ckf_update(M,P,Y,h,R,h_param)
3%CKF_UPDATE - Cubature Kalman filter update step
4% Syntax:
5%   [M,P,K,MU,S,LH] = CKF_UPDATE(M,P,Y,h,R,param)
6% In:
7%   M - Mean state estimate after prediction step
8%   P - State covariance after prediction step
9%   Y - Measurement vaector.
10%  h - Measurement model function as a matrix H defining
11%       linear function h(x) = H*x, inline function,
12%       function handle or name of function in
13%       form h(x,param)
14%  R - Measurement covariance.
15%  h_param - Parameters of h.
16% Out:
17%  M - Updated state mean
18%  P - Updated state covariance
19%  K - Computed Kalman gain
20%  MU - Predictive mean of Y
21%  S - Predictive covariance Y
22%  LH - Predictive probability (likelihood) of measurement.
23% Description:
24%  Perform additive form spherical-radial cubature Kalman filter (CKF)
25%  measurement update step. Assumes additive measurement noise.
26%  Function h should be such that it can be given
27%  DxN matrix of N sigma Dx1 points and it returns
28%  the corresponding measurements for each sigma
29%  point. This function should also make sure that
30%  the returned sigma points are compatible such that
31%  there are no 2pi jumps in angles etc.
32% Example:
33%  h = inline('atan2(x(2,:)-s(2),x(1,:)-s(1))','x','s');
34%  [M2,P2] = ckf_update(M1,P1,Y,h,R,S);
35
36  %Check that all arguments are there
37  if nargin < 5
```

```
38    error('Too few arguments');
39  end
40  %Do transform and make the update
41  if nargin < 6
42    [MU,S,C,X] = ckf_transform(M,P,h);
43  else
44    [MU,S,C,X] = ckf_transform(M,P,h,h_param);
45  end
46  S = S + R;
47  K = C / S;
48  M = M + K * (Y - MU);
49  P = P - K * S * K';
50  if nargout > 5
51    LH = gauss_pdf(Y,MU,S);
52  end
```

Apéndice C

Funciones implementadas para la API con V-REP.

En este apéndice explicaremos las funciones que hemos implementado en MATLAB para que la API tenga un funcionamiento correcto y podamos realizar simulaciones desde ella. Introduciremos los bloques de función que hemos creado y que han sido usados en el código de MATLAB para implementar los filtros de Kalman en nuestro robot móvil. Para más información acerca de la API de V-REP puede consultarse en su web [29]

C.1. Abrir y cerrar conexión con V-REP

C.1.1. Abrir conexión

Con esta función estableceremos la conexión entre MATLAB y V-REP. En caso de que la conexión no pueda darse se nos devolvería un mensaje de error en la línea de comandos de MATLAB. Por otro lado hay que tener precaución con las violaciones de segmento generadas desde MATLAB ya que esto cerraría la conexión con V-REP y no permitiría la reconexión hasta reiniciar MATLAB.

```
function clientID = Abrir_conexion(vrep,IP)
        clientID=vrep.simxStart(IP,19999,true,true,5000,5);
        %Activamos el modo sincrono

        %Comprobamos si ha sido posible conectarnos o no con V-REP.
        if clientID == -1
            disp('Imposible conectar!');

        end
        if clientID > -1
            disp('Conexion establecida!');
```

```
12              vrep.simxSynchronous(clientID,true);
13          end
14      end
```

C.1.2. Cerrar conexión

Con esta función cerramos la conexión entre V-REP y uno de sus clientes.

```
1 function Cerrar_conexion(vrep,clientID)
2         fprintf('Conexion del cliente %d cerrada \n',clientID);
3     end
```

C.2. Estado de la simulación

C.2.1. Iniciar simulación

```
1function Iniciar_simulacion(vrep,clientID)
2         disp('-----------------------------------------------');
3         disp('Iniciamos la simulacion...');
4         vrep.simxStartSimulation(clientID,vrep.simx_opmode_oneshot);
5     end
```

C.2.2. Reanudar simulación

Función utilizada para reanudar la simulación cuando esta ha sido pausada.

```
1function Reanudar_simulacion(vrep,clientID) %Usaremos esta funcion en el
    caso de que previamente pausemos la simulacion.     vrep.
    simxStartSimulation(clientID,vrep.simx_opmode_oneshot_wait);
2     end
```

C.2.3. Parar simulación

Detener la simulación devolviendo los objetos a su posición inicial (momento previo al comienzo de la simulación).

```
1function Parar_simulacion(vrep,clientID)
2         disp('-----------------------------------------------');
```

```
3         disp('Paramos la simulacion.');          vrep.simxStopSimulation(
              clientID,vrep.simx_opmode_oneshot);
4 end
```

C.2.4. Pausar simulación

Función para pausar la simulación conservando la posición de los objetos dentro de la escena.

```
1 function [Estado]=Pausar_simulacion(vrep,clientID) %Funcion para pausar una
    simulacion. Esto significa que la simulacion mantendra todas las
    propiedades al reanudarla.       [Estado]=vrep.simxPauseSimulation(
    clientID,vrep.simx_opmode_oneshot_wait);
2  end
```

C.3. Creación de objetos,modelos y escenas

C.3.1. Crear modelo de test

Función para crear un modelo que permita testear propiedades dentro de la escena.

```
1 function [Estado, Handle] = Crear_dummy(vrep,clientID,size) %Creamos una
    esfera que nos permite hacer un test de las fisicas        vrep.
    simxCreateDummy(clientID,size,[],vrep.simx_opmode_oneshot_wait);
2  end
```

C.3.2. Cargar modelo

Con esta función cargaremos modelos que tengamos guardados en una carpeta del equipo local en la escena de V-REP. Desde esta función podemos cargar cualquier tipo de modelo simplemente indicando la ruta en la que está guardado.

```
1 function [Estado,Identificador] = Cargar_modelo(vrep,clientID,
    modelPathAndName,options)
2         disp('-------------------------------------------------');
3         disp('Vamos a agregar un modelo a nuestra escena...');
4         [Estado,Identificador] = vrep.simxLoadModel(clientID,
              modelPathAndName,options,vrep.simx_opmode_oneshot_wait);
5         %Comprobamos si el modelo ha sido cargado correctamente o hemos
6         %generado algun tipo de error.
```

```
7         if Estado ~= 32
8             disp('Modelo cargado con exito!');
9         else
10            disp('Error al intentar cargar el modelo (Func:Cargar_modelo)');
11        end
12    end
```

C.3.3. Cargar escena

Con esta función cargaremos las escenas que tengamos guardados en una carpeta del equipo local en la escena de V-REP. Desde esta función podemos cargar cualquier tipo de modelo simplemente indicando la ruta en la que está guardado.

```
1function Estado = Cargar_escena(vrep,clientID,path,opciones)
2         disp('-----------------------------------------------------');
3         disp('Vamos a cargar la escena elegida...');
4         Estado = vrep.simxLoadScene(clientID,path,opciones, vrep.
              simx_opmode_oneshot_wait);
5         %Comprobamos si la escena se ha podido cargar correctamente.
6         if Estado ~= 32
7             disp('Escena cargada con exito!');
8         else
9             disp('Error al intentar cargar la escena (Func:Cargar_escena)');
10        end
11    end
```

C.3.4. Cerrar escena

```
1function [Estado]= Cerrar_Escena(vrep,clientID)      [Estado]=vrep.
    simxCloseScene(clientID,vrep.simx_opmode_oneshot_wait);
2    end
```

C.4. Mensajes por pantalla e interfaz

C.4.1. Mensaje por terminal

. Función para enviar mensajes desde Matlab a la consola de V-Rep. Los mensajes serán enviados a la consola por defecto creada al iniciar el programa.

```
1function Mensaje_terminal(vrep,clientID,texto) %Funcion que nos permite
    mandar un mensaje via terminal.        vrep.simxAddStatusbarMessage(
    clientID,texto,vrep.simx_opmode_oneshot);
2    end
```

C.4.2. Mostrar el ping de la conexión

```
1function [Ping] = Mostrar_ping(vrep,clientID) %Funcion que nos permite
    conocer el ping de nuestra conexion con V-REP
2        Ping =  vrep.simxGetPingTime(clientID);
3    end
```

C.4.3. Abrir consola auxiliar

```
1function [Estado,Handle_consola] = Abrir_consolaAux(vrep,clientID,nombre,
    lines_max,posicion,size,color_texto,color_fondo) %Abrir una consola para
     poder mostrar informacion
2        [Estado,Handle_consola] = vrep.simxAuxiliaryConsoleOpen(clientID,
            nombre,lines_max,0,posicion,size,color_texto,color_fondo,vrep.
            simx_opmode_oneshot_wait);
3    end
```

C.4.4. Cerrar consola auxiliar

```
1function [Estado] = Cerrar_consolaAux(vrep,clientID,Handle)
2        [Estado] = vrep.simxAuxiliaryConsoleClose(clientID,Handle,vrep.
            simx_opmode_oneshot);
3    end
```

C.4.5. Mostrar cuadro de diálogo

```
1function [Estado , Handle_dialogo, Handle_ui] = Mostrar_dialogo(vrep,
    clientID,titulo,texto_principal,modo,texto_inicial)
2        [Estado, Handle_dialogo, Handle_ui] = vrep.simxDisplayDialog(
            clientID,titulo,texto_principal,modo,texto_inicial,[],[],vrep.
            simx_opmode_oneshot_wait);
3    end
```

C.4.6. Mostrar datos en el cuadro de diálogo

```
function [Estado, Numero_resultado] = Obtener_resultados_dialogo(vrep,
    clientID,Handle_dialogo)
        [Estado,Numero_resultado] = vrep.simxGetDiaglogResult(clientID,
            Handle_dialogo,vrep.simx_opmode_oneshot);
    end
```

C.4.7. Obtener diálogo de entrada al cuadro

```
function [Estado, Texto_entrada] = Obtener_dialogo_entrada(vrep,clientID,
    Handle_dialogo)

        [Estado, Texto_entrada] = vrep.simxGetDialogInput(clientID,
            Handle_dialogo,vrep.simx_opmode_oneshot_wait);
    end
```

C.4.8. Cerrar cuadro de diálogo

```
function [Estado] = Cerrar_dialogo(vrep,clientID,Handle_dialogo)

        [Estado] = vrep.simxEndDialog(clientID,Handle_dialogo,vrep.
            simx_opmode_oneshot);
    end
```

C.5. Propiedades de los objetos

C.5.1. Posición de un objeto en la escena

Esta función nos devuelve la posición del objeto indicado dentro de la escena.

```
function [Estado,Posicion] = Obtener_posicion(vrep,clientID,
    identificador_obj) %Funcion para obtener las coordenadas cartesianas de
    un objeto en la escena

        [Estado,Posicion] = vrep.simxGetObjectPosition(clientID,
            identificador_obj,-1,vrep.simx_opmode_oneshot_wait);
        Posicion = Posicion';
    end
```

C.5.2. Orientación de un objeto en la escena

```
function [Estado,Rotacion] = Obtener_rotacion(vrep,clientID,
    identificador_obj) %Funcion para obtener las rotaciones de un objeto en
    la escena.

        [Estado,Rotacion] = vrep.simxGetObjectOrientation(clientID,
            identificador_obj,-1,vrep.simx_opmode_oneshot_wait);
        Rotacion = Rotacion';
    end
```

C.5.3. Obtener el identificador de un objeto

Función utilizada para obtener el handle de un objeto en la escena para posteriormente poder modificar cualquiera de sus propiedades.

```
function [Estado,Numero] = Obtener_handle(vrep,clientID,nombre) %Funcion
    que usamos para obtener el handle de un objeto que necesitemos

        [Estado,Numero] = vrep.simxGetObjectHandle(clientID,nombre,vrep.
            simx_opmode_oneshot_wait);
    end
```

C.5.4. Definir la posición de un objeto en la escena

```
function [Estado] = Definir_posicion(vrep,clientID,Handle,posicion) %Funcion
    para definir la posicion de un objeto en la escena.

        [Estado] = vrep.simxSetObjectPosition(clientID,Handle,-1,posicion,
            vrep.simx_opmode_oneshot);
    end
```

C.5.5. Definir la orientación de un objeto en la escena

```
  function [Estado] = Definir_orientacion(vrep,clientID,Handle,
      orientaciones) %Funcion para definir la orientacion de un objeto en
      la escena.

        [Estado] = vrep.simxSetObjectOrientation(clientID,Handle,-1,
            orientaciones,vrep.simx_opmode_oneshot);
```

```
4     end
```

C.5.6. Obtener el identificador de un grupo de objetos

Función utilizada para obtener el conjunto handles de un grupo de objetos en la escena para posteriormente poder modificar cualquiera de sus propiedades.

```
1 function [Estado,Handle] = Obtener_handle_coleccion(vrep,clientID,nombre) %
    Funcion que usamos para conocer el handle de una coleccion de objetos
2
3         [Estado,Handle] = vrep.simxGetCollectionHandle(clientID,nombre,vrep.
              simx_opmode_oneshot_wait);
4     end
```

C.6. Funciones para el Robot

C.6.1. Obtener pose del robot

Función que nos devuelve la pose del robot dentro de la escena, por defecto serán las coordenadas en x e y además de la orientación.

```
1     function [Estado,Pose] = Obtener_pose(vrep,clientID,identificador_obj) %
          Esta funcion nos devuelve las posiciones X e Y ademas de la
          orientacion.
2         %Obtenemos la posicion del robot en la escena.
3         [Estado_1,pose] = vrep.simxGetObjectPosition(clientID,
              identificador_obj,-1,vrep.simx_opmode_oneshot_wait);
4         %Obtenemos la orientacion de nuestro robot.
5         [Estado_2,angulos] = vrep.simxGetObjectOrientation(clientID,
              identificador_obj,-1,vrep.simx_opmode_oneshot_wait);
6         Pose = [pose(1);pose(2);angulos(3)];
7         %Comprobamos que todo el proceso se ha dado de forma correcta, en
8         %caso contrario mandamos un mensaje de error.
9         if Estado_1==32 || Estado_2 == 32
10            Estado = 32;
11            disp('Error al calcular la pose (Func:Obtener_pose)');
12        else
13            Estado = Estado_1;
14        end
15    end
```

C.6.2. Mover Robot

Función para mover el Robot a lo largo de la escena. Pasamos como parámetros la velocidad lineal y angular que deseamos para nuestro Robot y esta función por medio de la ecuación cinemática nos devuelve las velocidades que deben aplicarse a cada una de las ruedas.

```
function [vl,vr] = Mover_robot(vrep,clientID,robot,v,w) %Funcion que nos
    permite que robot siga unos objetivos prestablecidos
        if abs(v) < 0.0001 && abs(w) < 0.0001
            vl = 0; %Detenemos la rueda izquierda
            vr = 0; %Detenemos la rueda derecha
        else
            v_hat = v ; %Aqui podemos poner un termino de ruido. Hemos
                optado por considerar ese ruido como parte de la escena (
                friccion del suelo)
            w_hat = w ; %Aqui tambien podemos poner un termino de ruido.
            vl = (v_hat - robot.Radio_Robot * w_hat) / robot.
                Radio_Rueda_Izquierda; %calculamos los comandos que
                tendriamos que mandar a cada una de las ruedas.
            vr = (v_hat + robot.Radio_Robot * w_hat) / robot.
                Radio_Rueda_Derecha;
        end
        Enviar_signal(vrep,clientID,[vr vl]); %Enviamos el string de
            movimiento al robot en V-REP.
    end
     function Parar_robot(clientID) %Funcion que hace que el robot pare sus
         motores.
        Enviar_signal(clientID,[0 0]);
     end
```

C.6.3. Definir parámetros del Robot

Esta función crea una estructura de datos llamada Robot que contendrá todos los datos relacionados con el modelo y la simulación. Entre estos datos se encuentran el radio del robot, el ruido de medida, la trayectoria seguida, la información de la odometría, etc.

```
function Parametros = Definir_robot(vrep,clientID,identificador_obj) %
    Funcion que inicializa todos los parametros de un robot insertado en la
     escena.
        %Definimos la pose del robot. Primero consultamos cual es su estado
```

```
3         %y luego la guardamos.
4         [Estado,Pose] = Obtener_pose(vrep,clientID,identificador_obj);
5         if Estado ~= 32
6             field1 = 'x';  value1 = Pose(1,1);
7             field2 = 'y';  value2 = Pose(2,1);
8             field3 = 'alfa';  value3 = Pose(3,1);
9         else
10            field1 = 'x';  value1 = -9999;
11            field2 = 'y';  value2 = -9999;
12            field3 = 'alfa';  value3 = -9999;
13            disp('Error calculando la pose del robot (Func:Definir_robot)');
14        end
15
16        %Calculamos el radio de nuestro robot ya que lo necesitamos para
17        %saber que comandos aplicar para tomar curvas.
18
19        [Estado_1,Numero_1] = Obtener_handle(vrep,clientID,'
              Pioneer_p3dx_leftWheel');
20        [Estado_2,Numero_2] = Obtener_handle(vrep,clientID,'
              Pioneer_p3dx_rightWheel');
21        if Estado_1 ~= 0 || Estado_2 ~= 0
22            Radio_robot = -9999;
23            disp('Error al obtener el handle de las ruedas del robot (Func:
                  Definir_robot)');
24        else
25            [Estado_1,Posicion_izq] = Obtener_posicion(vrep,clientID,
                  Numero_1);
26            [Estado_2,Posicion_der] = Obtener_posicion(vrep,clientID,
                  Numero_2);
27            if Estado_1 ~= 0 || Estado_2 ~= 0
28                Radio_robot = -7777;
29                disp('Error al obtener el radio de las ruedas del robot (
                      Func:Definir_robot)');
30            else
31                Radio_robot = sqrt((Posicion_izq(1,1)-Posicion_der(1,1))^2 +
                      (Posicion_izq(2,1)-Posicion_der(2,1))^2 );
32                Radio_rueda_der = Posicion_der(3);
33                Radio_rueda_izq = Posicion_izq(3);
34            end
35        end
36
37        field4 = 'Radio_Robot';  value4 = {Radio_robot};
38        field5 = 'Radio_Rueda_Derecha'; value5 = {Radio_rueda_der};
```

```
39        field6 = 'Radio_Rueda_Izquierda'; value6 = {Radio_rueda_izq};
40        field7 = 'Client_ID'; value7 = {clientID};
41        field8 = 'Handle_Robot'; value8 = {identificador_obj};
42        field9 = 'Pose_Anterior'; value9 = {[0;0;0]};
43        field10 = 'Odometria'; value10 = {[0;0;0]};
44
45        [Estado,Numero] = Obtener_handle(vrep,clientID,'
              Pioneer_p3dx_leftMotor');
46        if Estado ~= 0
47            field11 = 'Rueda_izq_handle'; value11 = {9999};
48            disp('Error al obtener el handle del motor izquierdo (Func:
                  Definir_robot)');
49        else
50            field11 = 'Rueda_izq_handle'; value11 = {Numero};
51        end
52        field12 = 'Ult_ori_izq'; value12 = {0};
53
54        [Estado,Numero] = Obtener_handle(vrep,clientID,'
              Pioneer_p3dx_rightMotor');
55        if Estado ~= 0
56            field13 = 'Rueda_der_handle'; value13 = {9999};
57            disp('Error al obtener el handle del motor derecho (Func:
                  Definir_robot)');
58        else
59            field13 = 'Rueda_der_handle'; value13 = {Numero};
60        end
61        field14 = 'Ult_ori_der'; value14 = {0};
62
63        Y = [];
64        field15 = 'Medidas'; value15 = {Y};
65        T_real = [];
66        field16 = 'Trayectoria_real'; value16 = {T_real};
67        T_estimada = [];
68        field17 = 'Trayectoria_est'; value17 = {T_estimada};
69        Odom_total = [];
70        field18 = 'Odometria_total'; value18 = {Odom_total};
71        Y_real = [];
72        field19 = 'Medidas_reales'; value19 = {Y_real};
73        %Definimos unos valores bajos para los parametros de ruido ya que
74        %si es cero el filtro de Kalman no funcionaria.
75        field20 = 'Ruido_avance'; value20 = {0.0001};
76        field21 = 'Ruido_giro'; value21 = {0.0001};
77        field22 = 'Ruido_medida'; value22 = {0.0001};
```

```
78
79          %Guardamos todos los parametros de la estrucutra de datos.
80          Parametros = struct(field1,value1,field2,value2,field3,value3,field4
                ,value4,field5,value5,field6,value6,field7,value7,field8,value8,
                field9,value9,field10,value10,field11,value11,field12,value12,
                field13,value13,field14,value14,field15,value15,field16,value1,
                field17,value17,field18,value18,field19,value19,field20,value20,
                field21,value21,field22,value22);
81
82      end
```

C.6.4. Enviar comandos de movimiento

Función utilizada para enviar datos entre Matlab y V-rep. En este caso empaquetamos los datos sobre los comandos a enviar a las ruedas, V-rep posteriormente los desempaqueta y los aplica.

```
1 function [Estado] = Enviar_signal(vrep,clientID,Valor_signal) %Funcion que
      usamos para enviar los datos de los motores a V-REP.
2
3           v = [Valor_signal 0];
4           packedData = vrep.simxPackFloats(v);
5           [Estado] = vrep.simxSetStringSignal(clientID,'PioneerDiffData',
                packedData,vrep.simx_opmode_oneshot);
6       end
```

C.6.5. Leer posición de Balizas

Función utilizada para leer la posición de las Balizas dentro de la escena.

```
1function [Pos_bal,Numero_bal] = Balizas(vrep,clientID)
2           %Lo que haremos en esta funcion sera recorrer los handle en orden
3           %para ir sacando la posicion de cada una de las balizas y
4           %recogeremos esos datos en un vector. Posteriormente es lo que
5           %usaremos como parametro de entrada al filtro de Kalman.
6           Pos_bal = [];
7           Identificador = '80cmHighPillar25cm';
8           Identificador_Obj = Identificador;
9           [Estado, Numero] = Obtener_handle(vrep,clientID,Identificador);
10          a = -1;
11
```

```
12        while Estado == 0
13            [Estado, Numero] = Obtener_handle(vrep,clientID,
                  Identificador_Obj);
14            [Estado_2, Posicion_bal] = Obtener_posicion(vrep,clientID,Numero
                  );
15            if Estado == 0 && Estado_2 == 0
16                Pos_bal = [Pos_bal Posicion_bal];
17            end
18
19            a = a + 1;
20            a = num2str(a);
21            Identificador_Obj = [Identificador a];
22            a = str2num(a);
23
24        end
25
26        Numero_bal = size(Pos_bal,2);
27
28    end
```

C.6.6. Leer posición de los objetivos

Función para leer los objetivos dentro de la escena y guardarlos en un vector.

```
1function [Pos_objetivos,Num_objetivos] = Objetivos_loc(vrep,clientID,
    pose_inicial)
2       %Lo que haremos en esta funcion sera recorrer los handle en orden
3        %para ir sacando la posicion de cada uno de los objetivos y
4        %recogeremos esos datos en un vector.
5        Pos_objetivos = [];
6        Identificador = 'Cylinder';
7        Identificador_Obj = Identificador;
8        [Estado, Numero] = Obtener_handle(vrep,clientID,Identificador);
9        a = -1;
10
11       while Estado == 0
12            [Estado, Numero] = Obtener_handle(vrep,clientID,
                  Identificador_Obj);
13            [Estado_2, Posicion_objetivo] = Obtener_posicion(vrep,clientID,
                  Numero);
14            if Estado == 0 && Estado_2 == 0
15                Pos = Posicion_objetivo ;
```

```
16                  Pos_objetivos = [Pos_objetivos Pos];
17              end
18
19              a = a + 1;
20              a = num2str(a);
21              Identificador_Obj = [Identificador a];
22              a = str2num(a);
23          end
24
25          Angulos = [];
26          angulos = [];
27          a = 2;
28          %Ahora necesitamos calcular los angulos con los que afrontamos los
29          %objetivos, esto solo se usaria si queremos realizar una
30          %interpolacion lineal. En el caso de realizar otro tipo de
31          %movimiento no necesitamos saber los angulos relativos entre cada
32          %uno de los objetivos.
33          for i=1:size(Pos_objetivos,2)
34              if i == 1
35                  angulos = atan2(Pos_objetivos(2,1)-pose_inicial(2,1),
                        Pos_objetivos(1,1)-pose_inicial(1,1));
36              else
37                  angulos = atan2(Pos_objetivos(2,a)-Pos_objetivos(2,a-1),
                        Pos_objetivos(1,a)-Pos_objetivos(1,a-1));
38                  a = a+1;
39              end
40
41              Angulos = [Angulos angulos];
42
43          end
44
45          Pos_objetivos = [Pos_objetivos(1,:);Pos_objetivos(2,:);Angulos(1,:)]
46          Num_objetivos = size(Pos_objetivos,2);
47      end
```

C.6.7. Incrementar vueltas de los encoders

Función utilizada para leer las vueltas de los encoders y guardar esa información dentro de la estructura de datos del Robot.

```
1function [inc_izq,inc_der,robot] = Inc_encoder(vrep,clientID,robot) %Funcion
      que nos calcula el incremento de los encoders al mover el robot.
```

```
2
3       [Estado, Rot_izq] = vrep.simxGetJointPosition(clientID,robot.
            Rueda_izq_handle,vrep.simx_opmode_oneshot_wait);
4       angulo = dif_angulo(Rot_izq,robot.Ult_ori_izq);
5       inc_izq = angulo * robot.Radio_Rueda_Izquierda;
6       robot.Ult_ori_izq = Rot_izq;
7
8       [Estado, Rot_der] = vrep.simxGetJointPosition(clientID,robot.
            Rueda_der_handle,vrep.simx_opmode_oneshot_wait);
9       angulo = dif_angulo(Rot_der,robot.Ult_ori_der);
10      inc_der = angulo * robot.Radio_Rueda_Derecha;
11      robot.Ult_ori_der = Rot_der;
12
13
14  end
```

C.6.8. Diferencia angular entre dos posiciones

```
1 function [angulo] = dif_angulo(a,b)
2       %Funcion para calcular la diferencia angular entre dos posiciones
3       a = atan2(sin(a),cos(a));
4       b = atan2(sin(b),cos(b));
5
6       d1 = a-b;
7       d2 = 2*pi - abs(d1);
8
9       if (d1 > 0)
10          d2 = d2*(-1);
11      end
12      if (abs(d1) < abs(d2))
13          angulo = d1;
14      else
15          angulo = d2;
16      end
17  end
```

C.6.9. Actualizar la odometría del Robot

Función que actualiza la odometría del robot sirviéndose de la información recogida en los incrementos de los encoders.

```
function [Mov_centro , robot] = Actualizar_odom(vrep,clientID,robot) %
    Funcion para calcular al odometria de nuestro robot, es decir, sacar la
    posicion en funcion de lo que nos hemos movido.
        robot.Pose_Anterior = robot.Odometria ;
        [inc_izq,inc_der,robot] = Inc_encoder(vrep,clientID,robot);
        Mov_centro = (inc_izq + inc_der)/2;
        inc_yaw = ((inc_der - inc_izq) / robot.Radio_Robot);
        a_norm = norm_angulo(inc_yaw + robot.Pose_Anterior(3,1));
        robot.Odometria(3,1) = a_norm;
        factor = 0;
        if (abs(inc_yaw) < 0.00001)
            robot.Odometria(1,1) = robot.Pose_Anterior(1,1) + Mov_centro*cos
                (robot.Pose_Anterior(3,1));
            robot.Odometria(2,1) = robot.Pose_Anterior(2,1) + Mov_centro*sin
                (robot.Pose_Anterior(3,1));
        else
            factor = Mov_centro / norm_angulo(inc_yaw);
            robot.Odometria(1,1) = robot.Pose_Anterior(1,1) + (sin(inc_yaw)*
                cos(robot.Pose_Anterior(3,1))- sin(robot.Pose_Anterior(3,1))
                *(1 -cos(inc_yaw)))*factor ;
            robot.Odometria(2,1) = robot.Pose_Anterior(2,1) + (sin(inc_yaw)*
                sin(robot.Pose_Anterior(3,1))+ cos(robot.Pose_Anterior(3,1))
                *(1 -cos(inc_yaw)))*factor ;

        end
    end
```

C.6.10. Normalizar ángulos

```
function [a_norm] = norm_angulo(angulo) %Funcion para normalizar el angulo.
        a_norm = mod(mod(angulo,2*pi) + 2*pi, 2*pi);
        if (a_norm > pi)
            a_norm = a_norm - 2*pi;
        end
    end
```

C.6.11. Modelo de odometría para vector de control u_t [1]

Modelo de odometría basado en una rotación, translación y rotación entre dos puntos [1].

```
function [ u_t ] = odometry_motion(pose_ant,pose_act)
        Rotacion_1 = atan2(pose_act(2,1) - pose_ant(2,1),pose_act(1,1) -
            pose_ant(1,1)) - pose_act(3,1);
        Traslacion = sqrt((pose_ant(1,1) - pose_act(1,1))^2 + (pose_ant(2,1)
            - pose_act(2,1))^2 );
        Rotacion_2 = pose_act(3,1) - pose_ant(3,1) - Rotacion_1;

        u_t = [Traslacion*cos(pose_ant(3,1));Traslacion*sin(pose_ant(3,1));
            Rotacion_1 + Rotacion_2];

    end
```

C.6.12. Seguimiento de objetivos

Función que realiza el seguimiento de los objetivos disponibles dentro de la escena. La trayectoria descrita por el Robot será una curva que pase por los objetivos marcados.

```
function [w,v] = Seguir_objetivos(w,v,W,V) %Funcion de seguimiento de
    objetivos, el robot tratara de describir una curva suavizada.
    %El metodo consiste en realizar una interpolacion con el suavizado
    %entre tramos de la trayectoria.
        dist = v;
        if w > W
            w = W;
        end
        if w < -W
            w = -W;
        end

        if (v > V)
            v = V;
        end

        if (v < 0)
            v = 0;
        end

        if  abs(w) > 0.05 && abs(w) < 0.15 && dist > V
            v = V/4;
        else
            if abs(w) > 0.01 && abs(w) < 0.05 && dist > V
                v = V/2;
```

```
25              else
26                  if abs(w) > 0.15
27                      v = 0;
28                  end
29              end
30          end
31      end
```

C.6.13. Tomar medidas de distancia con los sensores

Función para tomar medidas de distancia del Robot con respecto a las balizas y respetando una distancia máxima.

```
1function [y_bal,y_real,y_adap,Pos_bal_adap,Lecturas] = Tomar_medidas_dist(
    Numero_bal,Pos_bal,pose,sd_baliza,dist_max)
2       %Funcion que realiza las medidas alrededor del robot, conforme a
3       %las balizas que tiene dentro del rango circular especificado en la
4       %de distancia.
5
6       for l=1:Numero_bal
7           y = sqrt((pose(1,1)-Pos_bal(1,l))^2 + (pose(2,1)-Pos_bal(2,l))^2
                ) + sd_baliza*randn;
8           y_bal(l,1) = y;
9           if (y <= dist_max ) %Establecemos el limite de metros que el
               telemetro es capaz de medir
10              y_real(l,1) = y_bal(l,1);
11          else
12              y_real(l,1) = 0;
13          end
14      end
15
16      Lecturas = 1;
17      y_adap = [];
18      Pos_bal_adap = [];
19
20      for h=1:size(y_real,1)
21          if (y_real(h,1) ~= 0)
22              y_adap(Lecturas,1) = y_real(h,1);
23              Pos_bal_adap(:,Lecturas) = Pos_bal(:,h);
24              Lecturas = Lecturas +1;
25          end
26      end
```

```
27
28      end
```

C.6.14. Tomar medidas de ángulo con los sensores

Función para tomar medidas de ángulo del Robot con respecto a las balizas y respetando una distancia máxima.

```
1 function [y_bal,y_real,y_adap,Pos_bal_adap,Lecturas] = Tomar_medidas_angle(
    Numero_bal,Pos_bal,pose,sd_baliza,dist_max)
2       %Funcion que realiza las medidas alrededor del robot, conforme a
3       %las balizas que tiene dentro del rango circular especificado en la
4       %distancia maxima, aunque para este caso lo que medimos es el angulo
            .
5
6       for l=1:Numero_bal
7           y_dist = sqrt((pose(1,1)-Pos_bal(1,l))^2 + (pose(2,1)-Pos_bal(2,
                l))^2 );
8           y = atan2(pose(2,1)-Pos_bal(2,l),pose(1,1)-Pos_bal(1,l)) +
                sd_baliza*randn;
9           y_bal(l,1) = y;
10          if (y_dist <= dist_max ) %Establecemos el limite de metros que
                el telemetro es capaz de medir
11              y_real(l,1) = y_bal(l,1);
12          else
13              y_real(l,1) = 0;
14          end
15      end
16
17      Lecturas = 1;
18      y_adap = [];
19      Pos_bal_adap = [];
20
21      for h=1:size(y_real,1)
22          if (y_real(h,1) ~= 0)
23              y_adap(Lecturas,1) = y_real(h,1);
24              Pos_bal_adap(:,Lecturas) = Pos_bal(:,h);
25              Lecturas = Lecturas +1;
26          end
27      end
28
29  end
```

C.6.15. Establecer ruidos del Robot

Función para parametrizar los ruidos por los que se ve afectado el robot dentro de la escena.

```
function Establecer_ruido(robot,ruido_avance,ruido_giro,ruido_medida) %
    Funcion para caracterizar los ruidos de nuestro robot.
        robot.Ruido_avance = ruido_avance;
        robot.Ruido_giro = ruido_giro;
        robot.Ruido_medida = ruido_medida;
    end
```

C.6.16. Angulo relativo normalizado

```
function [w]=Angulo_relativo(pose,objetivo)
        %Diferencia entre dos orientaciones (la del robot y el punto que
        %quiere alcanzar) Finalmente usamos esta funcion para calcular como
        %debe darse la trayectoria.
        w = atan2(objetivo(2,1)-pose(2,1),objetivo(1,1)-pose(1,1))-pose(3,1)
            ;
        while w > pi
            w = w-2*pi;
        end
        while w < -pi
            w = w+2*pi;
        end
    end
```

C.6.17. Representar trayectoria al finalizar la simulación (Plot 2D)

Función para representar la trayectoria realizada por el Robot en un plano bidimensional dentro de Matlab.

```
function Representar_trayectoria(Objetivos,Balizas,Tray_real,Tray_estimada)
        for i=1:size(Objetivos,2)
            plot(Objetivos(1,i),Objetivos(2,i),'k^');
            hold on
        end
        for j=1:size(Balizas,2)
            plot(Balizas(1,j),Balizas(2,j),'m^');
            hold on
        end
```

```
10
11          plot(Tray_real(1,:),Tray_real(2,:),'b');
12          hold on
13          plot(Tray_estimada(1,:),Tray_estimada(2,:),'r');
14
15          axis square
16          title('Recorrido realizado y estimacion');
17
18      end
```

C.7. Funciones experimentales de los filtros

C.7.1. Experimentos filtro de Kalman clásico (KF)

```
1function [Robot_1,KF_RMS,objetivos,Pos_bal] = V_rep_KF(Escena,sd_baliza)
2     clf;
3     KF_RMS = [];
4     semilla = 1;
5
6          rng(semilla)
7          disp('Pruebas de comuniacion con V-REP');
8          disp('   Autor: Ivan Rodriguez Mendez');
9          disp(' ');
10         vrep=remApi('remoteApi'); %importamos el archivo de la API.
11         disp('>Funciones cargadas!');
12         disp(' ');
13
14         %Creamos varias opciones para poder elegir que escena nos gustaria
15         %cargar
16
17         %% Seleccion de escena
18
19
20
21          if Escena == 1
22              link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_01.
                    ttt';
23         end
24         if Escena == 2
25              link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_003
                    .ttt';
```

```
26          end
27           if Escena == 3
28              link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_005
                    .ttt';
29           end
30          if Escena == 4
31              link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_007
                    .ttt';
32          end
33           if Escena == 5
34              link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_01.ttt
                    ';
35           end
36           if Escena == 6
37              link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_003.
                    ttt';
38           end
39          if Escena == 7
40              link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_005.
                    ttt';
41          end
42          if Escena == 8
43              link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_007.
                    ttt';
44          end
45          if Escena == 9
46              link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_01.ttt'
                    ;
47          end
48          if Escena == 10
49              link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_003.ttt
                    ';
50          end
51          if Escena == 11
52              link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_005.ttt
                    ';
53          end
54          if Escena == 12
55              link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_007.ttt
                    ';
56          end
57          if Escena == 13
```

```
58          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_3.
                ttt';
59      end
60      if Escena == 14
61          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_5.
                ttt';
62      end
63      if Escena == 15
64          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_3.ttt'
                ;
65      end
66      if Escena == 16
67          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_5.ttt'
                ;
68      end
69      if Escena == 17
70          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_3.ttt';
71      end
72      if Escena == 18
73          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_5.ttt';
74      end
75      if Escena == 19
76          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/
                Circuito_completo.ttt';
77      end
78
79
80      %Aqui es donde empieza el codigo principal
81      disp('Nos conectaremos a la siguiente');
82      IP_UNI = '10.213.13.228'
83      IP_PERSONAL = '192.168.1.38'
84      IP_LOCAL = '127.0.0.1'
85
86      clientID = Abrir_conexion(IP_LOCAL);
87
88      %Comprobamos si estamos conectados a V-REP
89      if clientID > -1
90          conectado = 1;
91      else
92          conectado = 0;
93      end
94
95      if conectado ==  1
```

```
96          fprintf('El cliente con el que se ha establecido conexion es el
               numero %d \n',clientID);
97          disp(' ');
98          disp('Conectado al servidor remoto API y ejecutando el codigo
               principal...');
99
100         %Enviamos alguna informacion al terminal de V-rep
101         Mensaje_terminal(clientID,'Simulacion de localizacion de un
               robot diferencial');
102         %% Preparacion de la escena y definicion de parametros
103         Cargar_escena(clientID,link,1); %Cargamos la escena en la que se
               ejecutara la simulacion
104         Cargar_modelo(clientID,'/home/ivan/Dropbox/TFG/V-REP/ModeloVREP/
               pioneer_pen.ttm',1); %Cargamos el modelo del robot que
               usaremos, este es el robot real.
105
106         [Estado,Handle] = Obtener_handle(clientID,'Pioneer_p3dx');
107         Robot_1 = Definir_robot(clientID,Handle)
108
109         Cargar_modelo(clientID,'/home/ivan/Dropbox/TFG/V-REP/ModeloVREP/
               pioneer_estimado.ttm',1); %Cargamos el modelo del robot que
               usaremos, este es el estimado.
110         [Estado,Robot_2] = Obtener_handle(clientID,'Pioneer_p3dx#0');
111         Definir_posicion(clientID,Robot_2,[0 0 0.1388]); %Definimos la
               posicion inicial para el robot estimado.
112
113         disp('Calculando la posicion de las balizas...');
114         [Pos_bal,Numero_bal] = Balizas(clientID); %Calculamos con ayuda
               de la funcion la colocacion de las balizas.
115         Numero_bal %Mostramos el numero de balizas que encontramos en la
               escena
116
117         disp('posicion de las balizas calculada');
118
119
120         %% Caracterizacion del robot
121         %Generamos las velocidades lineales y angulares de nuestro robot
122         V_LINEAL = 0.5; %Velocidad lineal maxima
123         V_ANGULAR = 7.2; %Velocidad de rotacion maxima
124         Limite_trayectoria = 500 ; %Definimos el limite de la
               trayectoria, es decir la cantidad maxima de puntos que
               cogemos.
```

```
125         dist_max = 5 ; %distancia maxima que es capaz de medir el
                telemetro.
126
127         %Definimos la pose inicial que tendra nuestro robot
128         x_inicial = Robot_1.x;
129         y_inicial = Robot_1.y;
130         alfa_inicial = Robot_1.alfa;
131         pose = [x_inicial;y_inicial;alfa_inicial]; %Guardamos la pose
                inicial del robot real insertado.
132
133
134         %Definimos los parametros que nos ayudaran a calcular la
135         %trayectoria
136
137         Epsilon = 0.3; %Umbral de distancia maxima permitida (error)
138         V = V_LINEAL;
139         W = V_ANGULAR*pi/(180);
140
141         %Definimos el primer punto de la trayectoria que vamos a
142         %seguir, logicamente el primer punto es la posicion del robot.
143         trayectoria = [x_inicial;y_inicial;alfa_inicial];
144
145         %Calculamos los objetivos que tenemos que alcanzar, es decir
146         %las coordenadas que tienen para poder alcanzarlas.
147         disp('Calculando la localizacion de los objetivos...');
148         objetivos = Objetivos_loc(clientID,pose);
149         disp('posicion de los objetivos calculada');
150
151         %Definimos un indice para recorrer el vector de objetivos (para
152         %conocer cual es el objetivo que queremos buscar)
153         obj_actual = 1;
154
155
156         %Inicializamos los vectores de medida que usaremos en el
157         %recorrido para la simulacion.
158         %% Inicializacion de parametros de vectores
159         y = [];
160         y_bal = [];
161         Y = [];
162         Y_real = [];
163         x = [];
164         X = [];
165         odom = [];
```

```
166         Odom = [];
167         %% Inicializacion de parametros del filtro
168
169         %% Ruidos del robot
170
171         sd_avance = 0.05;
172         sd_giro = 0.00001;
173         Establecer_ruido(Robot_1,sd_avance,sd_giro,sd_baliza); %
                Guardamos el ruido en la estructura que creamos para el
                robot.
174         dt = 0.05; %Definimos el diferencial de tiempo para calcular las
                 matrices A y Q.
175
176         M = [0;0;0]; %Media inicial para el filtro
177         P = diag([0.2 0.2 0.2]); %Matriz de covarianzas inicial
178         R = sd_baliza^2; %Ruido R definido como parametro del filtro
179
180         qx = 0.1;
181         qy = 0.1;
182
183         A = [1 dt 0;
184              0 1  0;
185              0 0  1]; %Definimos las matrices A y Q tal y como se
                    demuestra en la teoria de la toolbox
186
187         Q = [(1/3)*(dt^3)*qx (1/2)*(dt^2)*qx 0;
188              (1/2)*(dt^2)*qx    dt*qx        0;
189              0                  0            dt*qy];
190
191         MM = []; %Inicializamos los vectores que guardaran toda la
                informacion sacada de los filtros
192         PP = [];
193         %% Simulacion
194         Iniciar_simulacion(clientID); %Iniciamos una simulacion
195
196         %Iniciamos un indice para usar dentro del bucle
197         k=1;
198         %% Seguimiento de puntos en la trayectoria y aplicacion de
                Kalman
199         for i=1:size(objetivos,2)
200             while norm(trayectoria(1:2,size(trayectoria,2))-objetivos
                    (1:2,obj_actual)) > Epsilon && size(trayectoria,2) <=
                    Limite_trayectoria
```

```
201                 %Dentro de este bucle lo que intentamos hacer es que
202                 %el robot se mueva de tal manera que se vaya
203                 %aproximando al siguiente objetivo pendiente en el
204                 %vector. Basicamente aplicamos un pequeno algoritmo en
205                 %forma de interpolacion lineal entre objetivos.
206
207                 %Lo primero que hacemos es recuperar la posicion
208                 %actual del robot y guardarla.
209                 [Estado,pose] = Obtener_pose(clientID,Robot_1.
                        Handle_Robot);
210                 Robot_1.x = pose(1,1);
211                 Robot_1.y = pose(2,1);
212                 Robot_1.alfa = pose(3,1);
213
214                 x = pose;
215                 X = [X x];
216
217
218               %% Aplicamos el filtro de Kalman clasico
219               % Durante el movimiento del robot aplicaremos el filtro
220               % de Kalman para realizar la extimacion. Como
221               % anteriormente hemos tomado la informacion de la
222               % odometria, se la introduciremos a Kalman.
223                u_t = odometry_motion(Robot_1.Pose_Anterior,pose);
224                [M,P] = kf_predict(M,P,A,Q,eye(3),u_t);
                                        %Realizamos las medidas de las
                        medidas que captamos
225                 %desde nuestro robot.
226                 % Por otra parte medimos el diferencial de rotacion que
227                 % existe en la ruedas y de esta manera calculamos la
228                 % odometria.
229                 [Mov_centro,Robot_1] = Actualizar_odom(clientID,Robot_1
                        );
230                 odom = [Robot_1.Odometria(1,1);Robot_1.Odometria(2,1);
                        Robot_1.Odometria(3,1)];
231                 Odom = [Odom odom];
232
233                 [M,P] = kf_update(M,P,odom,eye(3),R*eye(3));
234                  MM(:,k) = M;
235                  PP(:,:,k) = P; %Guardamos las variables estimadas en
                         los vectores para poder tener un historico.
236
237                 %Colocamos el robot segun lo que hemos estimado.
```

```
238                 Definir_posicion(clientID,Robot_2,[M(1,1) M(2,1)
                        0.1388]);
239                 Definir_orientacion(clientID,Robot_2,[0 0 M(3,1)]);
240                 %% Seguimiento de objetivos
241                 %Una vez aplicamos Kalman para la posicion en la que
242                 %nos encontramos aplicamos el algortimo que permite al
243                 %robot seguir los objetivos.
244
245                 Objetivo = objetivos(:,obj_actual); %Guardamos en una
                        variable el objetivo que debemos alcanzar.
246                 angulo = Angulo_relativo(pose,Objetivo); % Calculamos el
                        angulo entre la posicion actual y el objetivo que
                        pretendemos alcanzar.
247                 v = norm(pose(1:2)-objetivos(1:2,obj_actual)); %
                        Calculamos la distancia entre la posicion actual y el
                         objetivo que queremos alcanzar.
248
249                 w = angulo; %Guardamos el angulo que hemos calculado para
                         pasar a trabajar con el.
250
251                 [w,v] = Seguir_objetivos(w,v,W,V); %Aplicamos nuestro
                        algortimo para que el robot siga los objetivos
                        describiendo una trayectoria curva.
252
253                 %Movemos el robot con los parametros que hemos
254                 %calculado anteriormente para cada una de las ruedas.
255                 Mover_robot(clientID,Robot_1,v,w);
256                 %Obtenemos la pose despues de haber movido el robot
257                 [Estado,pose] = Obtener_pose(clientID,Robot_1.
                        Handle_Robot);
258                 trayectoria = [trayectoria pose]; %Guardamos la pose del
                         robot en el vector que define la trayectoria.
259
260                 %Imprimimos dos variables de interes para saber en que
261                 %paso del bucle nos encontramos y que objetivo
262                 %pretendemos alcanzar.
263                 obj_actual
264                 Obj_real = obj_actual;
265                 size(trayectoria,2)
266                 k= k +1; %Indices que usamos para Kalman
267                 vrep.simxSynchronousTrigger(clientID); %Mandamos un
                        trigger de sincronizacion.
268             end
```

```
269             obj_actual = obj_actual+1; %Cuando salimos del bucle pasamos
                    a buscar el siguiente objetivo.
270         end
271
272         %Comprobamos que durante la simulacion hemos alcanzado todos
273         %los objetivos.
274         Parar_simulacion(clientID); %Paramos la simulacion en V-REP
275
276         if (Obj_real == size(objetivos,2))
277             disp('Se han alcanzado todos los objetivos');
278         else
279             disp('No se han alcanzado todos los objetivos');
280         end
281
282         %Guardamos todos los datos, el primero de ellos el error
283         %cuadratico medio.
284         kf_rms = sqrt(mean((X(1,:)-MM(1,:)).^2 + (X(2,:) - MM(2,:)).^2))
285
286         KF_RMS = [[kf_rms;Obj_real]];
287
288         %Guardamos todas las medidas en la estructura de datos para
289         %disponer de todos ellos una vez la funcion termine su
290         %ejecucion.
291         Robot_1.Medidas = Odom;
292         Robot_1.Trayectoria_real = X;
293         Robot_1.Trayectoria_est = MM;
294         Robot_1.Odometria_total = Odom;
295         Robot_1.Medidas_reales = [];
296
297         Representar_trayectoria(objetivos,Pos_bal,X,MM);
298
299         Cerrar_Escena(clientID);
300         %Nos aseguramos de que el ultimo comando a llegado
301         %correctamente.
302         %Cerramos las conexiones con V-REP:
303         Cerrar_conexion(clientID);
304         vrep.delete(); %llamamos al destructor para dejar de usar el
                metodo que hemos definido al principio de la funcion.
305
306         disp('>Programa finalizado!');
307     else
308         disp('La conexion no se ha podido establecer, asi que el
                programa se cerrara');
```

```
309
310         end
311
312
313 end
```

C.7.2. Experimentos filtro de Kalman extendido (EKF)

```
1 function [Robot_1,EKF_RMS] = V_rep_EKF(Escena,sd_baliza,Modelo_medida)
2
3     EKF_RMS = [];
4     semilla = 1;
5
6         rng(semilla)
7         disp('Pruebas de comuniacion con V-REP');
8         disp('   Autor: Ivan Rodriguez Mendez');
9         disp(' ');
10        vrep=remApi('remoteApi'); %importamos el archivo de la API.
11        disp('>Funciones cargadas!');
12        disp(' ');
13
14        %Creamos varias opciones para poder elegir que escena nos gustaria
15        %cargar
16        %% Seleccion del modelo de medida
17        % Si el modelo de medida es igual a 1 usaremos la distancia
18        % euclidea, en el caso de ser igual a 2 usaremos la informacion
19        % del angulo.
20
21
22
23        %% Seleccion de escena
24
25
26         if Escena == 1
27            link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_01.
                  ttt';
28        end
29        if Escena == 2
30            link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_003
                  .ttt';
31        end
32         if Escena == 3
```

```
33          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_005
                .ttt';
34      end
35      if Escena == 4
36          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_007
                .ttt';
37      end
38      if Escena == 5
39          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_01.ttt
                ';
40      end
41      if Escena == 6
42          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_003.
                ttt';
43      end
44      if Escena == 7
45          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_005.
                ttt';
46      end
47      if Escena == 8
48          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_007.
                ttt';
49      end
50      if Escena == 9
51          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_01.ttt'
                ;
52      end
53      if Escena == 10
54          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_003.ttt
                ';
55      end
56      if Escena == 11
57          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_005.ttt
                ';
58      end
59      if Escena == 12
60          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_007.ttt
                ';
61      end
62      if Escena == 13
63          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_3.
                ttt';
64      end
```

```
65        if Escena == 14
66            link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_5.
                  ttt';
67        end
68        if Escena == 15
69            link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_3.ttt'
                  ;
70        end
71        if Escena == 16
72            link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_5.ttt'
                  ;
73        end
74        if Escena == 17
75            link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_3.ttt';
76        end
77        if Escena == 18
78            link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_5.ttt';
79        end
80        if Escena == 19
81            link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/
                  Circuito_completo.ttt';
82        end
83        %Aqui es donde empieza el codigo principal
84        disp('Nos conectaremos a la siguiente');
85        IP_UNI = '10.213.13.228'
86        IP_PERSONAL = '192.168.1.38'
87        IP_LOCAL = '127.0.0.1'
88
89        clientID = Abrir_conexion(IP_LOCAL);
90
91        %Comprobamos si estamos conectados a V-REP
92        if clientID > -1
93            conectado = 1;
94        else
95            conectado = 0;
96        end
97
98        if conectado ==  1
99            fprintf('El cliente con el que se ha establecido conexion es el
                  numero %d \n',clientID);
100           disp(' ');
101           disp('Conectado al servidor remoto API y ejecutando el codigo
                  principal...');
```

```
102
103
104         %Enviamos alguna informacion al terminal de V-rep:
105         Mensaje_terminal(clientID,'Simulacion de localizacion de un
                robot diferencial');
106         %% Preparacion de la escena y definicion de parametros
107         Cargar_escena(clientID,link,1); %Cargamos la escena en la que se
                ejecutara la simulacion
108         Cargar_modelo(clientID,'/home/ivan/Dropbox/TFG/V-REP/ModeloVREP/
                pioneer_pen.ttm',1); %Cargamos el modelo del robot que
                usaremos, este es el robot real.
109
110         [Estado,Handle] = Obtener_handle(clientID,'Pioneer_p3dx');
111         Robot_1 = Definir_robot(clientID,Handle)
112
113         Cargar_modelo(clientID,'/home/ivan/Dropbox/TFG/V-REP/ModeloVREP/
                pioneer_estimado.ttm',1); %Cargamos el modelo del robot que
                usaremos, este es el estimado.
114         [Estado,Robot_2] = Obtener_handle(clientID,'Pioneer_p3dx#0');
115         Definir_posicion(clientID,Robot_2,[0 0 0.1388]); %Definimos la
                posicion inicial para el robot estimado.
116
117         disp('Calculando la posicion de las balizas...');
118         [Pos_bal,Numero_bal] = Balizas(clientID); %Calculamos con ayuda
                de la funcion la colocacion de las balizas.
119         Numero_bal %Mostramos el numero de balizas que encontramos en la
                escena
120
121         disp('posicion de las balizas calculada');
122
123
124         %% Caracterizacion del robot
125         %Generamos las velocidades lineales y angulares de nuestro robot
126         V_LINEAL = 0.5; %Velocidad lineal maxima
127         V_ANGULAR = 7.2; %Velocidad de rotacion maxima
128         Limite_trayectoria = 3000 ; %Definimos el limite de la
                trayectoria, es decir la cantidad maxima de puntos que
                cogemos.
129         dist_max = 5 ; %distancia maxima que es capaz de medir el
                telemetro.
130
131         %Definimos la pose inicial que tendra nuestro robot
132         x_inicial = Robot_1.x;
```

```
133         y_inicial = Robot_1.y;
134         alfa_inicial = Robot_1.alfa;
135         pose = [x_inicial;y_inicial;alfa_inicial]; %Guardamos la pose
                inicial del robot real insertado.
136
137
138         %Definimos los parametros que nos ayudaran a calcular la
139         %trayectoria
140
141         Epsilon = 0.3; %Umbral de distancia maxima permitida (error)
142         V = V_LINEAL;
143         W = V_ANGULAR*pi/(180);
144
145         %Definimos el primer punto de la trayectoria que vamos a
146         %seguir, logicamente el primer punto es la posicion del robot.
147         trayectoria = [x_inicial;y_inicial;alfa_inicial];
148
149         %Calculamos los objetivos que tenemos que alcanzar, es decir
150         %las coordenadas que tienen para poder alcanzarlas.
151         disp('Calculando la localizacion de los objetivos...');
152         objetivos = Objetivos_loc(clientID,pose);
153         disp('posicion de los objetivos calculada');
154
155         %Definimos un indice para recorrer el vector de objetivos (para
156         %conocer cual es el objetivo que queremos buscar)
157         obj_actual = 1;
158
159
160         %Inicializamos los vectores de medida que usaremos en el
161         %recorrido para la simulacion.
162         %% Inicializacion de parametros de vectores
163         y = [];
164         y_bal = [];
165         Y = [];
166         Y_real = [];
167         x = [];
168         X = [];
169         odom = [];
170         Odom = [];
171         %% Inicializacion de parametros del filtro
172
173         %% Importamos modelos de medida
174
```

```
175         if Modelo_medida == 1
176             %Importamos los Script que nos modelan la ecuacion de medida
177             %como la distancia euclidea.
178             h_func = @bot_dist_h; %Modelo de medida
179             dh_dx_func = @bot_dist_dh; %Derivada del modelo
180         end
181         if Modelo_medida == 2
182             %Importamos los Script que nos modelan la ecuacion de medida
183             %como el angulo entre la baliza y la pose del robot.
184             h_func = @bot_h; %Modelo de medida
185             dh_dx_func = @bot_dh_dx; %Derivada del modelo
186         end
187         %% Ruidos del robot
188         sd_avance = 0.05;
189         sd_giro = 0.00001;
190         Establecer_ruido(Robot_1,sd_avance,sd_giro,sd_baliza); %
                Guardamos el ruido en la estructura que creamos para el
                robot.
191         dt = 0.05; %Definimos el diferencial de tiempo para calcular las
                matrices A y Q.
192
193         M = [0;0;0]; %Media inicial para el filtro
194         P = diag([0 0 0]); %Matriz de covarianzas inicial
195         R = sd_baliza^2; %Ruido R definido como parametro del filtro
196
197         qx = 0.1;
198         qy = 0.1;
199
200         A = [1 dt 0;
201              0 1  0;
202              0 0  1]; %Definimos las matrices A y Q tal y como se
                    demuestra en la teoria de la toolbox
203
204         Q = [(1/3)*(dt^3)*qx (1/2)*(dt^2)*qx 0;
205              (1/2)*(dt^2)*qx    dt*qx        0;
206              0                  0            dt*qy];
207
208         MM = []; %Inicializamos los vectores que guardaran toda la
                informacion sacada de los filtros
209         PP = [];
210         %% Simulacion
211         Iniciar_simulacion(clientID); %Iniciamos una simulacion
212
```

```
213             %Iniciamos un indice para usar dentro del bucle
214             k=1;
215             %% Seguimiento de puntos en la trayectoria y aplicacion de
                    Kalman
216             for i=1:size(objetivos,2)
217                 while norm(trayectoria(1:2,size(trayectoria,2))-objetivos
                        (1:2,obj_actual)) > Epsilon && size(trayectoria,2) <=
                        Limite_trayectoria
218                   %Dentro de este bucle lo que intentamos hacer es que
219                   %el robot se mueva de tal manera que se vaya
220                   %aproximando al siguiente objetivo pendiente en el
221                   %vector. Basicamente aplicamos un pequeno algoritmo en
222                   %forma de interpolacion lineal entre objetivos.
223
224                   %Lo primero que hacemos es recuperar la posicion
225                   %actual del robot y guardarla.
226                  [Estado,pose] = Obtener_pose(clientID,Robot_1.
                         Handle_Robot);
227                  Robot_1.x = pose(1,1);
228                  Robot_1.y = pose(2,1);
229                  Robot_1.alfa = pose(3,1);
230
231                  x = pose;
232                  X = [X x];
233
234                  %Por otra parte medimos el diferencial de rotacion que
235                  %existe en la ruedas y de esta manera calculamos la
236                  %odometria.
237                  [Mov_centro ,Robot_1] = Actualizar_odom(clientID,Robot_1
                         );
238                  odom = [Robot_1.Odometria(1,1);Robot_1.Odometria(2,1);
                         Robot_1.Odometria(3,1)];
239                  Odom = [Odom odom];
240
241
242                 %% Aplicamos el filtro de Kalman extendido
243                 %Durante el movimiento del robot aplicaremos el filtro
244                 %de Kalman para realizar la extimacion. Como
245                 %anteriormente hemos tomado la informacion de la
246                 %odometria, se la introduciremos a Kalman.
247                  [M,P] = ekf_predict1(M,P,A,Q,odom);
248                  %Realizamos las medidas de las medidas que captamos
249                  %desde nuestro robot.
```

```
250                 if Modelo_medida == 1
251                     [y_bal,y_real,y_adap,Pos_bal_adap,Lecturas] =
                            Tomar_medidas_dist(Numero_bal,Pos_bal,pose,
                            Robot_1.Ruido_medida,dist_max);
252                 end
253                 if Modelo_medida == 2
254                     [y_bal,y_real,y_adap,Pos_bal_adap,Lecturas] =
                            Tomar_medidas_angle(Numero_bal,Pos_bal,pose,
                            Robot_1.Ruido_medida,dist_max);
255                 end
256                 %Guardamos los datos
257                  Y = [Y y_bal];
258                  Y_real = [Y_real y_real];
259                 [M,P] = ekf_update1(M,P,y_adap,dh_dx_func,R*eye(Lecturas
                        -1),h_func,[],Pos_bal_adap);
260                  MM(:,k) = M;
261                  PP(:,:,k) = P; %Guardamos las variables estimadas en
                        los vectores para poder tener un historico.
262
263                 %Colocamos el robot segun lo que hemos estimado.
264                 Definir_posicion(clientID,Robot_2,[M(1,1) M(2,1)
                        0.1388]);
265                 Definir_orientacion(clientID,Robot_2,[0 0 M(3,1)]);
266                 %% Seguimiento de objetivos
267                 %Una vez aplicamos Kalman para la posicion en la que
268                 %nos encontramos aplicamos el algortimo que permite al
269                 %robot seguir los objetivos.
270
271                 Objetivo = objetivos(:,obj_actual); %Guardamos en una
                        variable el objetivo que debemos alcanzar.
272                 angulo = Angulo_relativo(pose,Objetivo); %Calculamos el
                        angulo entre la posicion actual y el objetivo que
                        pretendemos alcanzar.
273                 v = norm(pose(1:2)-objetivos(1:2,obj_actual)); %
                        Calculamos la distancia entre la posicion actual y el
                         objetivo que queremos alcanzar.
274
275                 w = angulo; %Guardamos el angulo que hemos calculado para
                         pasar a trabajar con el.
276
277                 [w,v] = Seguir_objetivos(w,v,W,V); %Aplicamos nuestro
                        algortimo para que el robot siga los objetivos
                        describiendo una trayectoria curva.
```

```
278
279                 %Movemos el robot con los parametros que hemos
280                 %calculado anteriormente para cada una de las ruedas.
281                 Mover_robot(clientID,Robot_1,v,w);
282                 %Obtenemos la pose despues de haber movido el robot
283                 [Estado,pose] = Obtener_pose(clientID,Robot_1.
                        Handle_Robot);
284                 trayectoria = [trayectoria pose]; %Guardamos la pose del
                        robot en el vector que define la trayectoria.
285
286                 %Imprimimos dos variables de interes para saber en que
287                 %paso del bucle nos encontramos y que objetivo
288                 %pretendemos alcanzar.
289                 obj_actual
290                 Obj_real = obj_actual;
291                 size(trayectoria,2)
292                 k= k +1; %Indices que usamos para Kalman
293                 vrep.simxSynchronousTrigger(clientID); %Mandamos un
                        trigger de sincronizacion.
294             end
295             obj_actual = obj_actual+1; %Cuando salimos del bucle pasamos
                    a buscar el siguiente objetivo.
296         end
297
298         %Comprobamos que durante la simulacion hemos alcanzado todos
299         %los objetivos.
300         Parar_simulacion(clientID); %Paramos la simulacion en V-REP
301         clf;
302         if (Obj_real == size(objetivos,2))
303             disp('Se␣han␣alcanzado␣todos␣los␣objetivos');
304         else
305             disp('No␣se␣han␣alcanzado␣todos␣los␣objetivos');
306         end
307
308         %Guardamos todos los datos, el primero de ellos el error
309         %cuadratico medio.
310         ekf_rms = sqrt(mean((X(1,:)-MM(1,:)).^2 + (X(2,:) - MM(2,:)).^2)
                )
311
312         EKF_RMS = [[ekf_rms;Obj_real]];
313
314         %Guardamos todas las medidas en la estructura de datos para
315         %disponer de todos ellos una vez la funcion termine su
```

```
316             %ejecucion.
317             Robot_1.Medidas = Y;
318             Robot_1.Trayectoria_real = X;
319             Robot_1.Trayectoria_est = MM;
320             Robot_1.Odometria_total = Odom;
321             Robot_1.Medidas_reales = Y_real;
322
323             Representar_trayectoria(objetivos,Pos_bal,X,MM);
324
325             Cerrar_Escena(clientID);
326             %Nos aseguramos de que el ultimo comando a llegado
327             %correctamente.
328             %Cerramos las conexiones con V-REP:
329             Cerrar_conexion(clientID);
330             vrep.delete(); %llamamos al destructor para dejar de usar el
                    metodo que hemos definido al principio de la funcion.
331
332             disp('>Programa finalizado!');
333         else
334             disp('La conexion no se ha podido establecer, asi que el
                    programa se cerrara');
335
336         end
337
338
339end
```

C.7.3. Experimentos filtro de Kalman *unscented* (UKF)

```
1function [Robot_1,UKF_RMS] = V_rep_UKF(Escena,sd_baliza,Modelo_medida)
2
3    UKF_RMS = [];
4    semilla = 1;
5
6         rng(semilla)
7         disp('Pruebas de comuniacion con V-REP');
8         disp('   Autor: Ivan Rodriguez Mendez');
9         disp(' ');
10        vrep=remApi('remoteApi'); %importamos el archivo de la API.
11        disp('>Funciones cargadas!');
12        disp(' ');
13
```

```
14        %Creamos varias opciones para poder elegir que escena nos gustaria
15        %cargar
16        %% Seleccion del modelo de medida
17        % Si el modelo de medida es igual a 1 usaremos la distancia
18        % euclidea, en el caso de ser igual a 2 usaremos la informacion
19        % del angulo.
20
21
22
23        %% Seleccion de escena
24
25
26
27         if Escena == 1
28            link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_01.
                  ttt';
29        end
30        if Escena == 2
31            link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_003
                  .ttt';
32        end
33         if Escena == 3
34            link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_005
                  .ttt';
35         end
36        if Escena == 4
37            link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_007
                  .ttt';
38        end
39         if Escena == 5
40            link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_01.ttt
                  ';
41         end
42         if Escena == 6
43            link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_003.
                  ttt';
44         end
45        if Escena == 7
46            link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_005.
                  ttt';
47        end
48        if Escena == 8
```

```
49          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_007.
                ttt';
50      end
51      if Escena == 9
52          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_01.ttt'
                ;
53      end
54      if Escena == 10
55          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_003.ttt
                ';
56      end
57      if Escena == 11
58          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_005.ttt
                ';
59      end
60      if Escena == 12
61          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_007.ttt
                ';
62      end
63      if Escena == 13
64          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_3.
                ttt';
65      end
66      if Escena == 14
67          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_5.
                ttt';
68      end
69      if Escena == 15
70          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_3.ttt'
                ;
71      end
72      if Escena == 16
73          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_5.ttt'
                ;
74      end
75      if Escena == 17
76          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_3.ttt';
77      end
78      if Escena == 18
79          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_5.ttt';
80      end
81      if Escena == 19
```

```
82          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/
                Circuito_completo.ttt';
83      end
84
85      %Aqui es donde empieza el codigo principal
86      disp('Nos conectaremos a la siguiente');
87      IP_UNI = '10.213.13.228'
88      IP_PERSONAL = '192.168.1.38'
89      IP_LOCAL = '127.0.0.1'
90      IP_UNI2 = '10.213.13.213'
91
92      clientID = Abrir_conexion(IP_LOCAL);
93
94      %Comprobamos si estamos conectados a V-REP
95      if clientID > -1
96          conectado = 1;
97      else
98          conectado = 0;
99      end
100
101     if conectado ==  1
102         fprintf('El cliente con el que se ha establecido conexion es el
                numero %d \n',clientID);
103         disp(' ');
104         disp('Conectado al servidor remoto API y ejecutando el codigo
                principal...');
105
106
107         %Enviamos alguna informacion al terminal de V-rep:
108         Mensaje_terminal(clientID,'Simulacion de localizacion de un
                robot diferencial');
109         %%Preparacion de la escena y definicion de parametros
110         Cargar_escena(clientID,link,1); %Cargamos la escena en la que se
                 ejecutara la simulacion
111         Cargar_modelo(clientID,'/home/ivan/Dropbox/TFG/V-REP/ModeloVREP/
                pioneer_pen.ttm',1); %Cargamos el modelo del robot que
                usaremos, este es el robot real.
112
113         [Estado,Handle] = Obtener_handle(clientID,'Pioneer_p3dx');
114         Robot_1 = Definir_robot(clientID,Handle)
115
116         Cargar_modelo(clientID,'/home/ivan/Dropbox/TFG/V-REP/ModeloVREP/
                pioneer_estimado.ttm',1); %Cargamos el modelo del robot que
```

```
        usaremos, este es el estimado.
117     [Estado,Robot_2] = Obtener_handle(clientID,'Pioneer_p3dx#0');
118     Definir_posicion(clientID,Robot_2,[0 0 0.1388]); %Definimos la
        posicion inicial para el robot estimado.
119
120     disp('Calculando la posicion de las balizas...');
121     [Pos_bal,Numero_bal] = Balizas(clientID); %Calculamos con ayuda
        de la funcion la colocacion de las balizas.
122     Numero_bal %Mostramos el numero de balizas que encontramos en la
         escena
123
124     disp('posicion de las balizas calculada');
125
126
127     %% Caracterizacion del robot
128     %Generamos las velocidades lineales y angulares de nuestro robot
129     V_LINEAL = 0.5; %Velocidad lineal maxima
130     V_ANGULAR = 7.2; %Velocidad de rotacion maxima
131     Limite_trayectoria = 500 ; %Definimos el limite de la
        trayectoria, es decir la cantidad maxima de puntos que
        cogemos.
132     dist_max = 5 ; %distancia maxima que es capaz de medir el
        telemetro.
133
134     %Definimos la pose inicial que tendra nuestro robot
135     x_inicial = Robot_1.x;
136     y_inicial = Robot_1.y;
137     alfa_inicial = Robot_1.alfa;
138     pose = [x_inicial;y_inicial;alfa_inicial]; %Guardamos la pose
        inicial del robot real insertado.
139
140
141     %Definimos los parametros que nos ayudaran a calcular la
142     %trayectoria
143
144     Epsilon = 0.3; %Umbral de distancia maxima permitida (error)
145     V = V_LINEAL;
146     W = V_ANGULAR*pi/(180);
147
148     %Definimos el primer punto de la trayectoria que vamos a
149     %seguir, logicamente el primer punto es la posicion del robot.
150     trayectoria = [x_inicial;y_inicial;alfa_inicial];
151
```

```
152         %Calculamos los objetivos que tenemos que alcanzar, es decir
153         %las coordenadas que tienen para poder alcanzarlas.
154         disp('Calculando la localizacion de los objetivos...');
155         objetivos = Objetivos_loc(clientID,pose);
156         disp('posicion de los objetivos calculada');
157
158         %Definimos un indice para recorrer el vector de objetivos (para
159         %conocer cual es el objetivo que queremos buscar)
160         obj_actual = 1;
161
162
163         %Inicializamos los vectores de medida que usaremos en el
164         %recorrido para la simulacion.
165         %% Inicializacion de parametros de vectores
166         y = [];
167         y_bal = [];
168         Y = [];
169         Y_real = [];
170         x = [];
171         X = [];
172         odom = [];
173         Odom = [];
174         %% Inicializacion de parametros del filtro
175
176         %% Importamos modelos de medida
177
178         if Modelo_medida == 1
179             %Importamos los Script que nos modelan la ecuacion de medida
180             %como la distancia euclidea.
181             h_func = @bot_dist_h; %Modelo de medida
182             dh_dx_func = @bot_dist_dh; %Derivada del modelo
183         end
184         if Modelo_medida == 2
185             %Importamos los Script que nos modelan la ecuacion de medida
186             %como el angulo entre la baliza y la pose del robot.
187             h_func = @bot_h; %Modelo de medida
188             dh_dx_func = @bot_dh_dx; %Derivada del modelo
189         end
190         %% Ruidos del robot
191
192         sd_avance = 0.05;
193         sd_giro = 0.00001;
```

```
194 Establecer_ruido(Robot_1,sd_avance,sd_giro,sd_baliza); % Guardamos el ruido en la estructura que creamos para el robot.
195 dt = 0.05; %Definimos el diferencial de tiempo para calcular las matrices A y Q.
196
197 M = [0;0;0]; %Media inicial para el filtro
198 P = diag([0.2 0.2 0.2]); %Matriz de covarianzas inicial
199 R = sd_baliza^2; %Ruido R definido como parametro del filtro
200
201 qx = 0.1;
202 qy = 0.1;
203
204 A = [1 dt 0;
205 0 1 0;
206 0 0 1]; %Definimos las matrices A y Q tal y como se demuestra en la teoria de la toolbox
207
208 Q = [(1/3)*(dt^3)*qx (1/2)*(dt^2)*qx 0;
209 (1/2)*(dt^2)*qx dt*qx 0;
210 0 0 dt*qy];
211
212 MM = []; %Inicializamos los vectores que guardaran toda la informacion sacada de los filtros
213 PP = [];
214 %% Simulacion
215 Iniciar_simulacion(clientID); %Iniciamos una simulacion
216
217 %Iniciamos un indice para usar dentro del bucle
218 k=1;
219 %% Seguimiento de puntos en la trayectoria y aplicacion de Kalman
220 for i=1:size(objetivos,2)
221 while norm(trayectoria(1:2,size(trayectoria,2))-objetivos(1:2,obj_actual)) > Epsilon && size(trayectoria,2) <= Limite_trayectoria
222 %Dentro de este bucle lo que intentamos hacer es que
223 %el robot se mueva de tal manera que se vaya
224 %aproximando al siguiente objetivo pendiente en el
225 %vector. Basicamente aplicamos un pequeno algoritmo en
226 %forma de interpolacion lineal entre objetivos.
227
228 %Lo primero que hacemos es recuperar la posicion
```

```
229                 %actual del robot y guardarla.
230                 [Estado,pose] = Obtener_pose(clientID,Robot_1.
                        Handle_Robot);
231                 Robot_1.x = pose(1,1);
232                 Robot_1.y = pose(2,1);
233                 Robot_1.alfa = pose(3,1);
234
235                 x = pose;
236                 X = [X x];
237
238                 %Por otra parte medimos el diferencial de rotacion que
239                 %existe en la ruedas y de esta manera calculamos la
240                 %odometria.
241                 [Mov_centro ,Robot_1] = Actualizar_odom(clientID,Robot_1
                        );
242                 odom = [Robot_1.Odometria(1,1);Robot_1.Odometria(2,1);
                        Robot_1.Odometria(3,1)];
243                 Odom = [Odom odom];
244
245
246                 %%Aplicamos el filtro de Kalman extendido
247                 %Durante el movimiento del robot aplicaremos el filtro
248                 %de Kalman para realizar la extimacion. Como
249                 %anteriormente hemos tomado la informacion de la
250                 %odometria, se la introduciremos a Kalman.
251
252                 [M,P] = ukf_predict1(M,P,A,Q,[],1.5,2.05,2);
253                 M = (10*odom + 90*M)/100;
254                 %Realizamos las medidas de las medidas que captamos
255                 %desde nuestro robot.
256                 if Modelo_medida == 1
257                     [y_bal,y_real,y_adap,Pos_bal_adap,Lecturas] =
                            Tomar_medidas_dist(Numero_bal,Pos_bal,pose,
                            Robot_1.Ruido_medida,dist_max);
258                 end
259                 if Modelo_medida == 2
260                     [y_bal,y_real,y_adap,Pos_bal_adap,Lecturas] =
                            Tomar_medidas_angle(Numero_bal,Pos_bal,pose,
                            Robot_1.Ruido_medida,dist_max);
261                 end
262
263                 %Guardamos los datos
264                 Y = [Y y_bal];
```

```
265                 Y_real = [Y_real y_real];
266                 [M,P] = ukf_update1(M,P,y_adap,h_func,R*eye(Lecturas-1),
                        Pos_bal_adap);
267
268                 MM(:,k) = M;
269                 PP(:,:,k) = P; %Guardamos las variables estimadas en
                        los vectores para poder tener un historico.
270
271                 %Colocamos el robot segun lo que hemos estimado.
272                 Definir_posicion(clientID,Robot_2,[M(1,1) M(2,1)
                        0.1388]);
273                 Definir_orientacion(clientID,Robot_2,[0 0 M(3,1)]);
274                 %% Seguimiento de objetivos
275                 %Una vez aplicamos Kalman para la posicion en la que
276                 %nos encontramos aplicamos el algortimo que permite al
277                 %robot seguir los objetivos.
278
279             Objetivo = objetivos(:,obj_actual); %Guardamos en una
                    variable el objetivo que debemos alcanzar.
280             angulo = Angulo_relativo(pose,Objetivo); %Calculamos el
                    angulo entre la posicion actual y el objetivo que
                    pretendemos alcanzar.
281             v = norm(pose(1:2)-objetivos(1:2,obj_actual)); %
                    Calculamos la distancia entre la posicion actual y el
                     objetivo que queremos alcanzar.
282
283             w = angulo; %Guardamos el angulo que hemos calculado para
                     pasar a trabajar con el.
284
285             [w,v] = Seguir_objetivos(w,v,W,V); %Aplicamos nuestro
                    algortimo para que el robot siga los objetivos
                    describiendo una trayectoria curva.
286
287                 %Movemos el robot con los parametros que hemos
288                 %calculado anteriormente para cada una de las ruedas.
289                 Mover_robot(clientID,Robot_1,v,w);
290                 %Obtenemos la pose despues de haber movido el robot
291                 [Estado,pose] = Obtener_pose(clientID,Robot_1.
                        Handle_Robot);
292                 trayectoria = [trayectoria pose]; %Guardamos la pose del
                         robot en el vector que define la trayectoria.
293
294                 %Imprimimos dos variables de interes para saber en que
```

```
295                 %paso del bucle nos encontramos y que objetivo
296                 %pretendemos alcanzar.
297                 obj_actual
298                 Obj_real = obj_actual;
299                 size(trayectoria,2)
300                 k= k +1; %Indices que usamos para Kalman
301                 vrep.simxSynchronousTrigger(clientID); %Mandamos un
                        trigger de sincronizacion.
302             end
303             obj_actual = obj_actual+1; %Cuando salimos del bucle pasamos
                    a buscar el siguiente objetivo.
304         end
305         clf;
306         %Comprobamos que durante la simulacion hemos alcanzado todos
307         %los objetivos.
308         Parar_simulacion(clientID); %Paramos la simulacion en V-REP
309
310         if (Obj_real == size(objetivos,2))
311             disp('Se han alcanzado todos los objetivos');
312         else
313             disp('No se han alcanzado todos los objetivos');
314         end
315
316         %Guardamos todos los datos, el primero de ellos el error
317         %cuadratico medio.
318         ukf_rms = sqrt(mean((X(1,:)-MM(1,:)).^2 + (X(2,:) - MM(2,:)).^2)
                )
319
320         UKF_RMS = [[ukf_rms;Obj_real]];
321
322         %Guardamos todas las medidas en la estructura de datos para
323         %disponer de todos ellos una vez la funcion termine su
324         %ejecucion.
325         Robot_1.Medidas = Y;
326         Robot_1.Trayectoria_real = X;
327         Robot_1.Trayectoria_est = MM;
328         Robot_1.Odometria_total = Odom;
329         Robot_1.Medidas_reales = Y_real;
330
331         Representar_trayectoria(objetivos,Pos_bal,X,MM);
332
333         Cerrar_Escena(clientID);
334         %Nos aseguramos de que el ultimo comando a llegado
```

```
335             % correctamente.
336             % Cerramos las conexiones con V-REP:
337             Cerrar_conexion(clientID);
338             vrep.delete(); % llamamos al destructor para dejar de usar el
                    metodo que hemos definido al principio de la funcion.
339
340             disp('>Programa finalizado!');
341         else
342             disp('La conexion no se ha podido establecer, asi que el
                    programa se cerrara');
343
344         end
345
346
347 end
```

C.7.4. Experimentos filtro de Kalman de Cubatura (CKF)

```
1 function [Robot_1,CKF_RMS] = V_rep_CKF(Escena,sd_baliza,Modelo_medida)
2
3     CKF_RMS = [];
4     semilla = 1;
5
6         rng(semilla)
7         disp('Pruebas de comuniacion con V-REP');
8         disp('    Autor: Ivan Rodriguez Mendez');
9         disp(' ');
10        vrep=remApi('remoteApi'); %importamos el archivo de la API.
11        disp('>Funciones cargadas!');
12        disp(' ');
13
14        %Creamos varias opciones para poder elegir que escena nos gustaria
15        %cargar
16        %% Seleccion del modelo de medida
17        % Si el modelo de medida es igual a 1 usaremos la distancia
18        % euclidea, en el caso de ser igual a 2 usaremos la informacion
19        % del angulo.
20
21
22
23        %% Seleccion de escena
24
```

```
25
26
27          if Escena == 1
28              link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_01.
                    ttt';
29          end
30          if Escena == 2
31              link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_003
                    .ttt';
32          end
33          if Escena == 3
34              link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_005
                    .ttt';
35          end
36          if Escena == 4
37              link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_007
                    .ttt';
38          end
39          if Escena == 5
40              link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_01.ttt
                    ';
41          end
42          if Escena == 6
43              link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_003.
                    ttt';
44          end
45          if Escena == 7
46              link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_005.
                    ttt';
47          end
48          if Escena == 8
49              link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_007.
                    ttt';
50          end
51          if Escena == 9
52              link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_01.ttt'
                    ;
53          end
54          if Escena == 10
55              link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_003.ttt
                    ';
56          end
57          if Escena == 11
```

```
58          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_005.ttt
                ';
59      end
60      if Escena == 12
61          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_007.ttt
                ';
62      end
63      if Escena == 13
64          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_3.
                ttt';
65      end
66      if Escena == 14
67          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Cuadrado_LF_5.
                ttt';
68      end
69      if Escena == 15
70          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_3.ttt'
                ;
71      end
72      if Escena == 16
73          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Recta_LF_5.ttt'
                ;
74      end
75      if Escena == 17
76          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_3.ttt';
77      end
78      if Escena == 18
79          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/Seno_LF_5.ttt';
80      end
81      if Escena == 19
82          link = '/home/ivan/Dropbox/TFG/V-REP/Escenas-Exp/
                Circuito_completo.ttt';
83      end
84
85      %Aqui es donde empieza el codigo principal
86      disp('Nos conectaremos a la siguiente');
87      IP_UNI = '10.213.13.228'
88      IP_PERSONAL = '192.168.1.38'
89      IP_LOCAL = '127.0.0.1'
90
91      clientID = Abrir_conexion(IP_LOCAL);
92
93      %Comprobamos si estamos conectados a V-REP
```

```
94          if clientID > -1
95              conectado = 1;
96          else
97              conectado = 0;
98          end
99
100         if conectado ==  1
101             fprintf('El cliente con el que se ha establecido conexion es el
                    numero %d \n',clientID);
102             disp(' ');
103             disp('Conectado al servidor remoto API y ejecutando el codigo
                    principal...');
104
105
106             %Enviamos alguna informacion al terminal de V-rep:
107             Mensaje_terminal(clientID,'Simulacion de localizacion de un
                    robot diferencial');
108             %% Preparacion de la escena y definicion de parametros
109             Cargar_escena(clientID,link,1); %Cargamos la escena en la que se
                     ejecutara la simulacion
110             Cargar_modelo(clientID,'/home/ivan/Dropbox/TFG/V-REP/ModeloVREP/
                    pioneer_pen.ttm',1); %Cargamos el modelo del robot que
                    usaremos, este es el robot real.
111
112             [Estado,Handle] = Obtener_handle(clientID,'Pioneer_p3dx');
113             Robot_1 = Definir_robot(clientID,Handle)
114
115             Cargar_modelo(clientID,'/home/ivan/Dropbox/TFG/V-REP/ModeloVREP/
                    pioneer_estimado.ttm',1); %Cargamos el modelo del robot que
                    usaremos, este es el estimado.
116             [Estado,Robot_2] = Obtener_handle(clientID,'Pioneer_p3dx#0');
117             Definir_posicion(clientID,Robot_2,[0 0 0.1388]); %Definimos la
                    posicion inicial para el robot estimado.
118
119             disp('Calculando la posicion de las balizas...');
120             [Pos_bal,Numero_bal] = Balizas(clientID); %Calculamos con ayuda
                    de la funcion la colocacion de las balizas.
121             Numero_bal %Mostramos el numero de balizas que encontramos en la
                     escena
122
123             disp('posicion de las balizas calculada');
124
125
```

```
126 %% Caracterizacion del robot
127 %Generamos las velocidades lineales y angulares de nuestro robot
128 V_LINEAL = 0.5; %Velocidad lineal maxima
129 V_ANGULAR = 7.2; %Velocidad de rotacion maxima
130 Limite_trayectoria = 500 ; %Definimos el limite de la
        trayectoria, es decir la cantidad maxima de puntos que
        cogemos.
131 dist_max = 5 ; %distancia maxima que es capaz de medir el
        telemetro.
132
133 %Definimos la pose inicial que tendra nuestro robot
134 x_inicial = Robot_1.x;
135 y_inicial = Robot_1.y;
136 alfa_inicial = Robot_1.alfa;
137 pose = [x_inicial;y_inicial;alfa_inicial]; %Guardamos la pose
        inicial del robot real insertado.
138
139
140 %Definimos los parametros que nos ayudaran a calcular la
141 %trayectoria
142
143 Epsilon = 0.3; %Umbral de distancia maxima permitida (error)
144 V = V_LINEAL;
145 W = V_ANGULAR*pi/(180);
146
147 %Definimos el primer punto de la trayectoria que vamos a
148 %seguir, logicamente el primer punto es la posicion del robot.
149 trayectoria = [x_inicial;y_inicial;alfa_inicial];
150
151 %Calculamos los objetivos que tenemos que alcanzar, es decir
152 %las coordenadas que tienen para poder alcanzarlas.
153 disp('Calculando la localizacion de los objetivos...');
154 objetivos = Objetivos_loc(clientID,pose);
155 disp('posicion de los objetivos calculada');
156
157 %Definimos un indice para recorrer el vector de objetivos (para
158 %conocer cual es el objetivo que queremos buscar)
159 obj_actual = 1;
160
161
162 %Inicializamos los vectores de medida que usaremos en el
163 %recorrido para la simulacion.
164 %% Inicializacion de parametros de vectores
```

```
165         y = [];
166         y_bal = [];
167         Y = [];
168         Y_real = [];
169         x = [];
170         X = [];
171         odom = [];
172         Odom = [];
173         %% Inicializacion de parametros del filtro
174
175         %% Importamos modelos de medida
176
177         if Modelo_medida == 1
178             %Importamos los Script que nos modelan la ecuacion de medida
179             %como la distancia euclidea.
180             h_func = @bot_dist_h; %Modelo de medida
181             dh_dx_func = @bot_dist_dh; %Derivada del modelo
182         end
183         if Modelo_medida == 2
184             %Importamos los Script que nos modelan la ecuacion de medida
185             %como el angulo entre la baliza y la pose del robot.
186             h_func = @bot_h; %Modelo de medida
187             dh_dx_func = @bot_dh_dx; %Derivada del modelo
188         end
189         %% Ruidos del robot
190
191         sd_avance = 0.05;
192         sd_giro = 0.00001;
193         Establecer_ruido(Robot_1,sd_avance,sd_giro,sd_baliza); %
                Guardamos el ruido en la estructura que creamos para el
                robot.
194         dt = 0.05; %Definimos el diferencial de tiempo para calcular las
                matrices A y Q.
195
196         M = [0;0;0]; %Media inicial para el filtro
197         P = diag([0.2 0.2 0.2]); %Matriz de covarianzas inicial
198         R = sd_baliza^2; %Ruido R definido como parametro del filtro
199
200         qx = 0.1;
201         qy = 0.1;
202
203         A = [1 dt 0;
204              0 1  0;
```

```
205          0 0  1]; %Definimos las matrices A y Q tal y como se
                 demuestra en la teoria de la toolbox
206
207     Q = [(1/3)*(dt^3)*qx  (1/2)*(dt^2)*qx  0;
208          (1/2)*(dt^2)*qx   dt*qx           0;
209          0                 0               dt*qy];
210
211     MM = []; %Inicializamos los vectores que guardaran toda la
             informacion sacada de los filtros
212     PP = [];
213     %% Simulacion
214     Iniciar_simulacion(clientID); %Iniciamos una simulacion
215
216     %Iniciamos un indice para usar dentro del bucle
217     k=1;
218     %% Seguimiento de puntos en la trayectoria y aplicacion de
             Kalman
219     for i=1:size(objetivos,2)
220         while norm(trayectoria(1:2,size(trayectoria,2))-objetivos
                 (1:2,obj_actual)) > Epsilon && size(trayectoria,2) <=
                 Limite_trayectoria
221             %Dentro de este bucle lo que intentamos hacer es que
222             %el robot se mueva de tal manera que se vaya
223             %aproximando al siguiente objetivo pendiente en el
224             %vector. Basicamente aplicamos un pequeno algoritmo en
225             %forma de interpolacion lineal entre objetivos.
226
227             %Lo primero que hacemos es recuperar la posicion
228             %actual del robot y guardarla.
229             [Estado,pose] = Obtener_pose(clientID,Robot_1.
                     Handle_Robot);
230             Robot_1.x = pose(1,1);
231             Robot_1.y = pose(2,1);
232             Robot_1.alfa = pose(3,1);
233
234             x = pose;
235             X = [X x];
236
237             %Por otra parte medimos el diferencial de rotacion que
238             %existe en la ruedas y de esta manera calculamos la
239             %odometria.
240             [Mov_centro ,Robot_1] = Actualizar_odom(clientID,Robot_1
                     );
```

```
241 odom = [Robot_1.Odometria(1,1);Robot_1.Odometria(2,1);
        Robot_1.Odometria(3,1)];
242 Odom = [Odom odom];
243
244
245 %% Aplicamos el filtro de Kalman extendido
246 % Durante el movimiento del robot aplicaremos el filtro
247 % de Kalman para realizar la extimacion. Como
248 % anteriormente hemos tomado la informacion de la
249 % odometria, se la introduciremos a Kalman.
250
251 [M,P] = ckf_predict(M,P,A,Q); %
        Realizamos las medidas de las medidas que captamos
252 M = (12*odom + 3*M) / 15;
253 % desde nuestro robot.
254 if Modelo_medida == 1
255     [y_bal,y_real,y_adap,Pos_bal_adap,Lecturas] =
            Tomar_medidas_dist(Numero_bal,Pos_bal,pose,
            Robot_1.Ruido_medida,dist_max);
256 end
257 if Modelo_medida == 2
258     [y_bal,y_real,y_adap,Pos_bal_adap,Lecturas] =
            Tomar_medidas_angle(Numero_bal,Pos_bal,pose,
            Robot_1.Ruido_medida,dist_max);
259 end
260 % Guardamos los datos
261 Y = [Y y_bal];
262 Y_real = [Y_real y_real];
263 [M,P] = ckf_update(M,P,y_adap,h_func,R*eye(Lecturas-1),
        Pos_bal_adap);
264
265 MM(:,k) = M;
266 PP(:,:,k) = P; % Guardamos las variables estimadas en
        los vectores para poder tener un historico.
267
268 % Colocamos el robot segun lo que hemos estimado.
269 Definir_posicion(clientID,Robot_2,[M(1,1) M(2,1)
        0.1388]);
270 Definir_orientacion(clientID,Robot_2,[0 0 M(3,1)]);
271 %% Seguimiento de objetivos
272 % Una vez aplicamos Kalman para la posicion en la que
273 % nos encontramos aplicamos el algortimo que permite al
274 % robot seguir los objetivos.
```

```
275
276             Objetivo = objetivos(:,obj_actual); %Guardamos en una
                    variable el objetivo que debemos alcanzar.
277             angulo = Angulo_relativo(pose,Objetivo); %Calculamos el
                    angulo entre la posicion actual y el objetivo que
                    pretendemos alcanzar.
278             v = norm(pose(1:2)-objetivos(1:2,obj_actual)); %
                    Calculamos la distancia entre la posicion actual y el
                     objetivo que queremos alcanzar.
279
280             w = angulo; %Guardamos el angulo que hemos calculado para
                     pasar a trabajar con el.
281
282             [w,v] = Seguir_objetivos(w,v,W,V); %Aplicamos nuestro
                    algortimo para que el robot siga los objetivos
                    describiendo una trayectoria curva.
283
284              %Movemos el robot con los parametros que hemos
285              %calculado anteriormente para cada una de las ruedas.
286              Mover_robot(clientID,Robot_1,v,w);
287              %Obtenemos la pose despues de haber movido el robot
288              [Estado,pose] = Obtener_pose(clientID,Robot_1.
                     Handle_Robot);
289              trayectoria = [trayectoria pose]; %Guardamos la pose del
                     robot en el vector que define la trayectoria.
290
291              %Imprimimos dos variables de interes para saber en que
292              %paso del bucle nos encontramos y que objetivo
293              %pretendemos alcanzar.
294              obj_actual
295              Obj_real = obj_actual;
296              size(trayectoria,2)
297              k= k +1; %Indices que usamos para Kalman
298              vrep.simxSynchronousTrigger(clientID); %Mandamos un
                     trigger de sincronizacion.
299          end
300          obj_actual = obj_actual+1; %Cuando salimos del bucle pasamos
                 a buscar el siguiente objetivo.
301      end
302      clf;
303      %Comprobamos que durante la simulacion hemos alcanzado todos
304      %los objetivos.
305      Parar_simulacion(clientID); %Paramos la simulacion en V-REP
```

```
306
307             if (Obj_real == size(objetivos,2))
308                 disp('Se han alcanzado todos los objetivos');
309             else
310                 disp('No se han alcanzado todos los objetivos');
311             end
312
313             %Guardamos todos los datos, el primero de ellos el error
314             %cuadratico medio.
315             ckf_rms = sqrt(mean((X(1,:)-MM(1,:)).^2 + (X(2,:) - MM(2,:)).^2)
                    )
316
317             CKF_RMS = [[ckf_rms;Obj_real]];
318
319             %Guardamos todas las medidas en la estructura de datos para
320             %disponer de todos ellos una vez la funcion termine su
321             %ejecucion.
322             Robot_1.Medidas = Y;
323             Robot_1.Trayectoria_real = X;
324             Robot_1.Trayectoria_est = MM;
325             Robot_1.Odometria_total = Odom;
326             Robot_1.Medidas_reales = Y_real;
327
328             Representar_trayectoria(objetivos,Pos_bal,X,MM);
329
330             Cerrar_Escena(clientID);
331             %Nos aseguramos de que el ultimo comando a llegado
332             %correctamente.
333             %Cerramos las conexiones con V-REP:
334             Cerrar_conexion(clientID);
335             vrep.delete(); %llamamos al destructor para dejar de usar el
                    metodo que hemos definido al principio de la funcion.
336
337             disp('>Programa finalizado!');
338         else
339             disp('La conexion no se ha podido establecer, asi que el
                    programa se cerrara');
340
341         end
342
343
344 end
```

Bibliografía

Las siguientes referencias bibliográficas se presentan en orden alfabético por autor. Las referencias con más de un autor aparecen ordenadas en base al primero de los mismos.

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, Mass: Mit, Oct. 2005.

[2] M. P. Leonardo, "Navegación de un robot móvil de configuración diferencial basada en fusión sensorial," Tesina del máster, Universidad politénica de Valencia, Valencia, 2011.

[3] G. Welch, "An introduction to the kalman filter," University of North Carolina at Chapel Hill, Tech. Rep., 2001.

[4] E. A. Wan and R. V. D. Merwe, "The unscented Kalman filter for nonlinear estimation," in *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, 2000, pp. 153–158.

[5] "Leyton V., Hernando - Desarrollo, implementación y prueba de un filtro de Kalman del tipo UKF para un vehículo aéreo no tripulado," 2009. [Online]. Available: http://biblioteca.versila.com/

[6] W. L. Luigi D'Alfonso, "Mobile robot localization via EKF and UKF: A comparison based on real data," *Robotics and Autonomous Systems*, vol. 74, 2015.

[7] A. Ienkaran and H. Simon, "Cubature Kalman Filters," *IEEE*, 2009. [Online]. Available: http://soma.mcmaster.ca/papers/ckf_2009.pdf

[8] "BECS / Research / Bayesian Statistical Methods / Downloads / EKF/UKF Toolbox for Matlab." [Online]. Available: http://becs.aalto.fi/en/research/bayes/ekfukf/

[9] K. Capek, *R.U.R.*, rossum's universal robots edition ed. London ; New York: Penguin Classics, Mar. 1921.

[10] I. Asimov, *Yo, Robot*. Edhasa, May 2004.

[11] "Tres leyes de la robótica," Apr. 2016, page Version ID: 90677580. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Tres_leyes_de_la_rob%C3%B3tica&oldid=90677580

[12] R. Faragher, "Understanding the basis of the kalman filter via simple and intuitive derivation," *IEEE SIGNAL PROCESSING MAGAZINE*, Septiembre 2012.

[13] Álvaro Solera Ramírez, "El filtro de kalman," Banco central de Costa Rica, Tech. Rep., Julio 2003.

[14] "Markov process | mathematics | Britannica.com." [Online]. Available: http://global.britannica.com/topic/Markov-process

[15] "Proceso de Márkov," Dec. 2014, page Version ID: 78688782. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Proceso_de_M%C3%A1rkov&oldid=78688782

[16] "Taylor series," May 2016, page Version ID: 722150864. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Taylor_series&oldid=722150864

[17] S. Julier, J. Uhlmann, and H. F. Durrant-Whyte, "A new method for the nonlinear transformation of means and covariances in filters and estimators," *IEEE Transactions on Automatic Control*, vol. 45, no. 3, pp. 477–482, Mar. 2000.

[18] S. Julier and J. K. Uhlmann, "A General Method for Approximating Nonlinear Transformations of Probability Distributions," Tech. Rep., 1996.

[19] S. J. Julier, J. K. Uhlmann, and H. F. Durrant-Whyte, "A new approach for filtering nonlinear systems," in *American Control Conference, Proceedings of the 1995*, vol. 3, June 1995, pp. 1628–1632 vol.3.

[20] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, Mar. 2004.

[21] N. Houshangi and F. Azizi, "Accurate mobile robot position determination using unscented Kalman filter," in *Canadian Conference on Electrical and Computer Engineering, 2005.*, May 2005, pp. 846–851.

[22] S.-M. Chow, E. Ferrer, and J. R. Nesselroade, "An Unscented Kalman Filter Approach to the Estimation of Nonlinear Dynamical Systems Models," *Multivariate Behavioral Research*, vol. 42, no. 2, pp. 283–321, June 2007. [Online]. Available: http://dx.doi.org/10.1080/00273170701360423

[23] S.-M. Oh and E. Johnson, "Development of UAV Navigation System Based on Unscented Kalman Filter," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*. American Institute of Aeronautics and Astronautics, 2006. [Online]. Available: http://arc.aiaa.org/doi/abs/10.2514/6.2006-6351

[24] B. Zhou, Y. Peng, and J. Han, "UKF based estimation and tracking control of non-holonomic mobile robots with slipping," in *IEEE International Conference on Robotics and Biomimetics, 2007. ROBIO 2007*, Dec. 2007, pp. 2058–2063.

[25] "Cubature-based Kalman filters for positioning (PDF) - Semantic Scholar," 2010. [Online]. Available: https://www.semanticscholar.org/paper/Cubature-based-Kalman-filters-for-positioning-Pesonen-Pich%C3%A9/72ff4a0e1a81b4ead7cf54222cddb44e9d31ede0/pdf

[26] X.-C. Zhang and C.-J. Guo, "Cubature Kalman filters: Derivation and extension," *Chinese Physics B*, vol. 22, no. 12, p. 128401, Dec. 2013. [Online]. Available: http://stacks.iop.org/1674-1056/22/i=12/a=128401?key=crossref.3e1bc517bb7beba903a784ba7c433528

[27] "Pioneer P3-DX," 2016. [Online]. Available: http://www.mobilerobots.com/ResearchRobots/PioneerP3DX.aspx

[28] "Blender," Apr. 2016, page Version ID: 90736659. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Blender&oldid=90736659

[29] "V-REP. Virtual robot experimentation platform," 2016. [Online]. Available: http://www.coppeliarobotics.com/

[30] "ROS.org | Powering the world's robots." [Online]. Available: http://www.ros.org/

[31] "Repositorio software con el framework experimental desarrollado." [Online]. Available: https://github.com/ull-isaatc/ssbayes/

[32] C. Neuhauser, *Matemáticas para ciencias*, segunda ed. Person-Prentice Hall, 2004.

[33] E. Steiner, *Matemáticas para las ciencias aplicadas*. Reverté, 2005.

[34] G. P. Obando, "Técnicas recursivas para estimación dinámica una introducción matemática al filtro de kalman," Master's thesis, Fundación Universitaria Konrad Lorenz, Junio 2005.