Trabajo Fin de Grado
Grado en Ingeniería
Electrónica, Robótica y Mecatrónica

Algoritmos de localización de robots móviles empleando filtros estadísticos

Autor: Manuel Jiménez Través

Tutor: José Ramiro Martínez de Dios

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

Sevilla, 2015

Proyecto Fin de Grado
Grando en Ingeniería Electrónica, Robótica y Mecatrónica

Algoritmos de localización de robots móviles empleando filtros estadísticos

Autor:
Manuel Jiménez Través

Tutor:
José Ramiro Martínez de Dios
Profesor titular

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2015

Trabajo Fin de Grado: Algoritmos de localización de robots móviles empleando filtros estadísticos

Autor: Manuel Jiménez Través

Tutor: José Ramiro Martínez de Dios

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2015

El Secretario del Tribunal

Resumen

El presente trabajo recoge todo el desarrollo de una serie de métodos de localización de robots móviles. A lo largo de este proceso, se ha indagado en la estructura y uso de toda una familia de filtros estadísticos relacionados con el filtro de Kalman, además del filtro de partículas. Este trabajo también sirve como introducción al software especialista en robótica, ROS. Esto permite que el trabajo sea base de multitud de futuros trabajos que requieran de algoritmos de localización o el manejo de este software, indiferentemente.

Abstract

The current proyect compiles the entire development of a series of localization methods of mobile robots. Throughout this process, it has been investigated the structure and use of a whole family of statistical filters related to the Kalman filter, besides the particle filter. This proyect, also, serves as an introduction to robotics specialized software, ROS. This allows the work to be a basis for multitude of future proyects which requires localization algorithms or the use of this software, indistinctly.

Índice

Notación

μ, Σ	Notación paramétrica, vector de estado y matriz de covarianza
ξ, Ω	Notación canónica, vector y matriz de información
x_t, u_t, z_t	Vectores de estado, control y medidas en el instante actual, t
A, B, C	Matrices con ecuaciones lineales del modelo, actuador y medidas
R, Q	Matrices que modelan el error en el modelo y en las medidas
∂y/∂x	Derivada parcial de y respecto x
H, G	Jacobianos de las matrices A y B
$p_{i,t}$, v_t, T	Coordenada de posición i en el instante t, velocidad y periodo
d_t, d_{t-1}	Distancias entre robot y nodo WSN en instante actual, t, y anterior, $t-1$
v, w_g, w_r	Incertidumbre en modelo, GPS y sensor de rango
ω, np	Peso y número de partículas
rand, randn	Funciones para generar números aleatorios flotantes o enteros.

1 INTRODUCCIÓN

Hay una fuerza motriz más poderosa que el vapor, la electricidad y la energia atómica: la voluntad.

- Albert Einstein -

1.1 Contexto

El alcance de este trabajo es el estudio y simulación de distintos métodos de localización de robots móviles, especialmente, basada en rango empleando redes de sensores inalámbricas (Wireless Sensor Network, de ahora en adelante denominada WSN).

El marco del trabajo se desarrolla dentro de los campos de la robótica ubicua, así como robótica social y de servicios. La importancia de la localización de robots viene justificada desde que es imprescindible que el robot tenga conocimiento de su posición para labores completamente autónomas como localización, navegación y guiado de robots en aplicaciones como seguimiento de altas prestaciones, búsqueda, rescate y vigilancia, domótica, entre otros.

Para acometer esta meta se usará toda una gama de filtros estadísticos que aportarán una cómoda fusión sensorial, además de un robusto seguimiento del sistema.

1.2 Fundamento teórico

La piedra angular en la que se basa la técnica de localización en rango es la triangulación. Por lo tanto, como se observa en la Figura 1-1, se requiere de un mínimo de tres nodos que provean de la distancia al robot, dado que cada nodo sólo tiene información de que el robot se encuentra a una distancia *r*, desconociendo dirección, lo que equivale a una circunferencia. Así pues, superponiendo un mínimo de tres circunferencias, se puede encontrar un punto exacto.

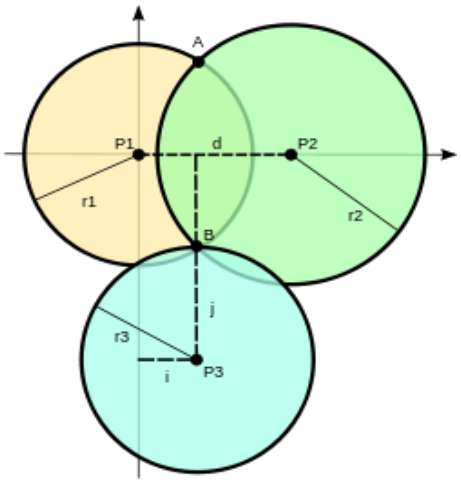


Figura 1-1. Justificación geométrica de la localización mediante triangulación

Ahora bien, esto sería una situación ideal, en la cual los sensores vienen desprovistos de incertidumbre. En el caso real, se asume que en la medida hay cierto error y se considera a la hora de calcular la posición, dado que los algoritmos, más adelante descritos, se basan en cálculos estadísticos y no absolutos. De esta forma, a medida que aumente el número de nodos que aporten información, más seguridad habrá en la posición estimada, tal y como dice el Teorema Central del Límite.

Planteando el principal objetivo, que es darle al robot conocimiento de su posición respecto unos ejes pre-escogidos, los escenarios que se podían encontrar eran dos, según la información previa del mapa: bien, previamente conocido o bosquejado, bien, desconocimiento absoluto. En este caso, sólo se tiene conocimiento de la posición de los sensores de rango.

Dentro de la ciencia del control, este trabajo se ubica en el área de los predictores, dada la naturaleza de los algoritmos empleados. Siguiendo el esquema clásico de control (Figura 1-2), dichos predictores vendrían a sustituir la función del sensor, dado que el vector de estado no se encontraría accesible y habría de ser calculado de forma indirecta.

Figura 1-2. Esquema clásico de control

El fin de esta aclaración es hacer hincapié en que el fundamento del presente trabajo es la localización, es decir, en ningún momento se recurren a acciones de control. En todo momento, el movimiento de la posición del robot está fuera del control del algoritmo.

1.3 Objetivo y procedimiento

Para demostrar la funcionalidad y comportamiento de los algoritmos, primero se han implementado en el programa de métodos numéricos, Matlab, en el cual además de usar sensores de rango, también se prueban en concretas ocasiones medidas de posición GPS, para poder ejemplificar versiones simples del algoritmo.

Posteriormente, el trabajo se trasladará a un software de realidad virtual dedicado a robótica, Gazebo, donde haciendo uso de una versión simulada de un robot existente en el laboratorio de la escuela, el Pioneer 3-AT (Figura 1-3), se implementarán los algoritmos de localización. Esto será posible mediante scripts hechos en C++ gracias a ROS (Robot Operating System), cuya jerarquía de funcionamiento será tratada en un capítulo posterior y exclusivo para ella.

En todo caso, el entorno en el cual se va a enfrentar el problema es, un escenario cualquiera, con un mínimo de tres sensores de rango estáticos, cuyas posiciones son totalmente conocidas, que se comunicarán con el robot móvil para calcular la distancia e informar de ésta, mediante técnicas ToA (Time of Arrival) o ToF (Time of Flight). Dada la naturaleza virtual de este trabajo, estos sensores vendrán simulados mediante ciertas variables de información accesibles de antemano, a las que se le incorpora ruido blanco aleatorio para emular el comportamiento real del sensor de rango.

Mediante el uso de sensores y la implementación de los distintos filtros, se realizará un estudio de la eficacia y precisión de estos en el problema de seguimiento de la posición del robot móvil.

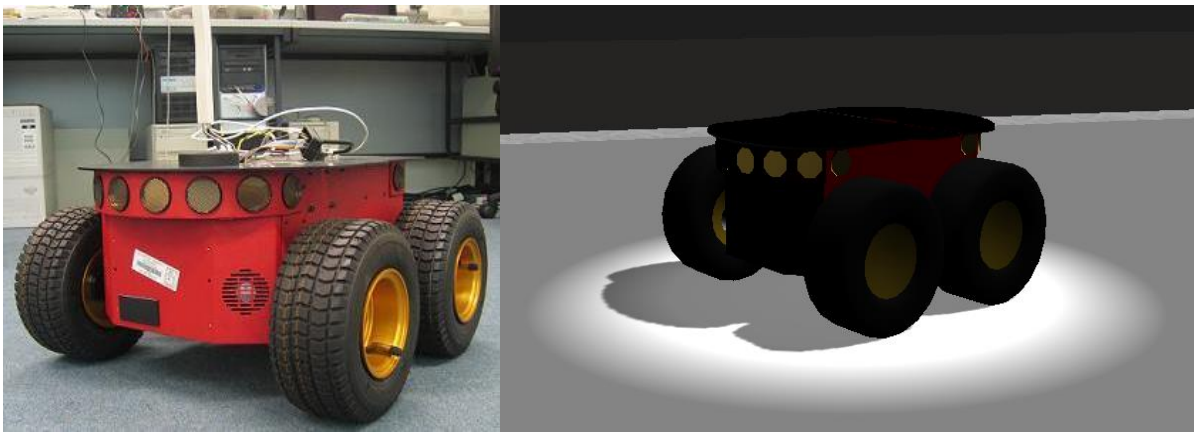


Figura 1-3. Robot móvil, Pioneer 3-AT. a) Modelo real, b) Modelo en Gazebo

1.4 Estructura del documento

El presente trabajo se desarrolla bajo un orden pensado para evolucionar de los contenidos teóricos a los prácticos. De tal forma que no se empieza a comentar el desafío a enfrentar hasta bien explicadas las bases teóricas de la solución.

- **Capítulo 1: Introducción**

Comprensión del problema, objetivos y procedimiento.

- **Capítulo 2: Filtros Estadísticos**

Sólida base teórica sobre las técnicas a implementar, explicación en detalle y personalización de cara al problema. Fundamentos de esta familia de filtros y aplicaciones.

- **Capítulo 3: Implementación Software**

Métodos y herramientas empleadas. Descripción del escenario del problema. Explicación detallada del funcionamiento de ROS.

- **Capítulo 4: Simulaciones**

Descripción de los experimentos simulados. Exposición y comprensión de los resultados obtenidos durante el proceso, analizando posibles mejoras.

- **Capítulo 5: Conclusión**

Reflexión final sobre el trabajo realizado, éxito y efectividad de la solución propuesta, futuros desarrollos y ampliaciones del proyecto.

2 FILTROS ESTADÍSTICOS

Ser consciente de la propia ignorancia es
un gran paso hacia el saber.

- Benjamin Disraeli -

2.1 Introducción

Los filtros paramétricos, también llamados filtros gaussianos, son la familia de técnicas más popular hasta la fecha. Esto es debido a que constituyen una fácil, y computacionalmente más sencilla, implementación del filtro bayesiano, comentado en el capítulo anterior. Así pues, hacen posible una estimación del vector de estado mediante la asunción de que la solución del problema es singular (unimodal), afectada con un ruido puramente gaussiano.

Esto es debido a la idea fundamental de dichas técnicas; el vector de estado viene representado por distribuciones normales multivariables, es decir, cada componente de dicho vector está caracterizado por dos parámetros, la media y la covarianza. Lo anterior se denomina parametrización de momentos dado que ambas variables representan el primer y el segundo momento, sucesivamente, de una distribución de probabilidad.

Aparte de los filtros paramétricos, se ha incluido en este capítulo el más importante de los filtros no paramétricos, viendo innecesario hacer otro capítulo completo para un único filtro. De ahí, la ambigüedad del nombre de este capítulo. Como se verá, este filtro aporta ciertas características únicas que permitirían acometer problemas de mayor complejidad.

Dicho esto, el objetivo de este capítulo es introducir y detallar tanto el Filtro de Kalman como su dual, el Filtro de Información, así como sus versiones extendidas para problemas no lineales, además del Filtro de Partículas.

2.2 Filtro de Kalman (*KF*)

Probablemente el filtro más extendido debido a su sencillez y eficacia a la hora de filtrar y estimar estados continuos, inventado por Swerling (1958) y Kalman (1960)[1].

2.2.1 Características del KF

Como se menciona en apartados anteriores, el filtro representa la creencia, esto es, el vector de estado estimado (se usarán ambas definiciones indistintamente), mediante la parametrización de momentos. En cada instante de tiempo t le corresponde al vector de estado un vector de media, μ_t, y una matriz de covarianza, Σ_t.

Es vital indicar que se deben cumplir ciertas propiedades:

1. Funciones lineales, aunque no necesariamente invariantes en el tiempo. Esto conlleva una posible variación en el tiempo, de las matrices que definen dichas fases.

2. Adición de ruido gaussiano en cada iteración. Con esto se considera la incertidumbre que hay en cada paso.

3. Además, se debe partir de una creencia inicial que sea normalmente distribuida.

[1] 3.2 The Kalman Filter > Linear Gaussian Systems. Probabilistic Robotics, S. Thrun / W. Burgard / D. Fox

Esto es necesario para conservar la forma gaussiana de la solución, ya que es una hipótesis fundamental para el correcto funcionamiento del filtro. Razón por la que se requiere que el sistema tenga solución unimodal.

2.2.2 Etapas del KF

Se explican ahora con un poco más de detalle las fases anteriormente mencionadas.

- ***Predicción:***

La primera fase que se menciona es la predicción. Mediante esta etapa el filtro consigue adelantar acontecimientos, calculando de forma anticipada el estado, x_t, partiendo del estado anterior, x_{t-1}, además de considerar las acciones de actuación, u_t, en el instante actual, posiblemente proporcionadas por un sistema de control. La indeterminación producida por la transición de estado se modela mediante la adición de ruido gaussiano ε_t, el cual tiene las mismas dimensiones que el vector de estado, media nula y covarianza denotada como R_t.

Ecuación de transición de estado, $p(x_t|x_{t-1}, u_t)$:

$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t \tag{2.1}$$

Los elementos que quedan pendientes por mencionar de la anterior expresión son matrices. A_t es una matriz cuadrada de las mismas dimensiones que el vector de estado, $n \times n$, en la cual se expresaría básicamente la evolución del estado siguiendo el modelo del sistema. A su vez B_t, recogería una evolución parecida pero en función de entradas de actuación, es decir, cómo reacciona el sistema ante entradas. Así pues, tendría un tamaño $n \times m$, donde m es la dimensión del vector de control u_t.

Más adelante se reserva un único apartado para explicar el modelo utilizado en el presente trabajo, así como otras posibles implementaciones.

- ***Actualización:***

Si en la primera fase se intenta predecir la situación del estado partiendo del conocimiento previo, aquí se busca corregir los posibles errores de forma que se obtenga una estimación lo más exacta posible del verdadero estado del sistema. Para ello, se cuenta con la ayuda de sensores que transmiten información necesaria del exterior, lo que permite contrastar la creencia predicha con lo que perciben los sensores.

Para más detalle, el filtro realiza, en función del estado que cree correcto, otra predicción con la diferencia de que, en este caso, es para calcular el vector de medidas predichas, $\bar{z}_t$. Este vector, junto al vector de medidas realmente recibidas por los sensores, permite al filtro mantener un seguimiento actualizado del estado con alta fiabilidad.

Ecuación de probabilidad de medida, $p(z_t|x_t)$:

$$z_t = C_t x_t + \delta_t \tag{2.2}$$

De forma análoga a la etapa anterior, el modelo de los sensores está representado por la matriz C_t, de tamaño $k \times n$, donde k corresponde a la dimensión del vector de medidas. Así mismo, esta fase considera cierta indeterminación en la medida representada por la adición de δ_t, idéntica a la variable asociada al ruido del apartado anterior, con la particularidad de que se denota su covarianza por Q_t.

2.2.3 Algoritmo del KF

El algoritmo del filtro de Kalman aparece descrito en la Figura 2-1. Partiendo de un estado anterior y del conocimiento de las acciones de control y medidas en el instante actual, el filtro busca calcular de forma óptima el estado más aproximado al real. Así pues, como se ha especificado anteriormente, se disciernen claramente dos fases: Predicción (líneas 2-3) y Actualización (4-6).

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

Figura 2-1. Algoritmo del filtro de Kalman

En la primera, el filtro realiza una estimación del estado, $\overline{\mu}_t$, haciendo uso del modelo del sistema por lo que la fiabilidad de esta etapa está estrechamente ligada a la bondad del modelo, lo cual se refleja en el cálculo de la covarianza predicha, $\overline{\Sigma}_t$.

De igual manera que la predicción se ve afectada por la bondad del modelo, la actualización depende de la bondad de los sensores dado que esta fase es, básicamente, la incorporación de las medidas al conocimiento que tiene el robot del entorno (lo que viene a ser el vector de estado). Sin embargo, existe una pequeña diferencia a la hora de considerar el ruido incorporado por los sensores.

En la etapa de actualización se encuentra la ganancia de Kalman, K_t (línea 4), la cual asigna la importancia, a la hora de computar el cálculo, de la información obtenida en las medidas respecto el estado predicho. Aquí es donde se tiene en cuenta la bondad de los sensores, además de la seguridad que tiene el robot del estado obtenido en la predicción (la matriz de covarianza del estado).

Entonces, el estado finalmente producido por el filtro es, en cierta manera, una media ponderada entre la predicción y la similitud entre medidas predichas y las obtenidas por los sensores.

De esta forma, si tuvieras una predicción del estado absolutamente perfecta, la predicción de las medidas (que en este algoritmo aparece en la línea 5, $C_t\overline{\mu}_t$) serían idénticas a las obtenidas por los sensores, aunque de igual manera, estos deberían estar exentos de todo ruido. Este caso converge en una total confianza por parte del algoritmo en el estado predicho, dado que el otro sumando se anula.

Esto último también sucede cuando el sistema se vuelve "ciego" al entorno, es decir, se pierde toda la información de los sensores, con lo que el segundo sumando se obvia y se asume como válida la predicción aceptando que es una estimación con mayor error, pero que permite al filtro seguir funcionando de forma robusta como predictor.

2.2.4 Ejemplo

Antes de nada, un pequeño inciso para recordar que se emplea estado y creencia indistintamente para hacer referencia al conocimiento que el robot tiene del verdadero estado, dado que éste es completamente inaccesible, sólo se puede intentar aproximar lo más fielmente posible.

La figura 2-2 muestra el comportamiento de un filtro de Kalman en un escenario unidimensional donde el robot se puede mover únicamente a la izquierda y a la derecha. El objetivo del filtro es localizar la posición del robot por lo que cuenta con sensores de localización (GPS, por ejemplo). Dado que los modelos son estadísticos, toda información se muestra en forma de función de probabilidad, concretamente una distribución normal.

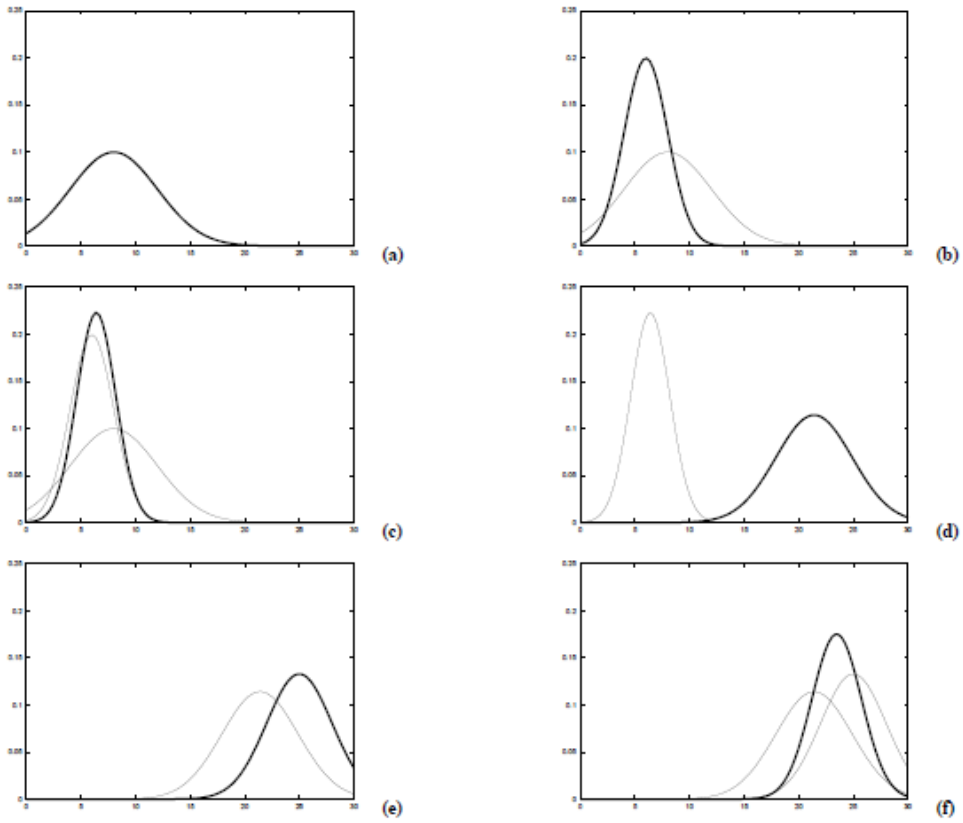


Figura 2-2. a) Creencia inicial, b) Medida, c) Estado tras incorporar la medida,
d) Estado tras movimiento a la derecha, e) Nueva medida, f) Estado final

La idea principal de esta ilustración es ver como el filtro hace seguimiento de la posición captada por la medida, de forma que, tras cada incorporación de nueva información proveniente de los sensores, la campana de Gauss se aproxima a dicha posición a la par que se vuelve más estrecha. Esta estrechez implica una disminución del error en la medida.

De forma contraria, cuando el robot se desplaza la curva de la creencia también lo hace pero ensanchándose debido a la enorme incertidumbre que implica esta acción.

En líneas generales, toda nueva información, incluso siendo redundante, contribuye a mejorar la exactitud de la creencia, mientras que cada movimiento, o cambio de situación, aporta incertidumbre.

2.3 Filtro de Información (*IF*)

2.3.1 Características del IF

Este filtro nace como la versión dual del KF, y, al igual que éste, representa la información como una gaussiana multivariable unimodal. La principal diferencia es que toma otras variables (parametrización canónica), con lo cual, el cambio varía el cómputo de la información, es decir, las ecuaciones detrás del KF. De igual manera, la parametrización canónica como la de momentos también son duales entre sí.

Esta nueva parametrización comprende de un vector de información y una matriz de información, en lugar de lo que existía antes con el KF: el vector de estado (la media o esperanza) y la matriz de covarianza. Las relaciones que guardan entre sí ambas parametrizaciones son:

$$\Omega = \Sigma^{-1} \tag{2.3}$$

$$\xi = \Sigma^{-1}\mu \tag{2.4}$$

Donde Ω es la matriz de información, también llamada de precisión, y ξ es el vector de información.

Los conceptos detrás de esta transformación implican otra forma distinta de entender la situación. Mientras que en el KF se tiene como resultado una creencia y una incertidumbre ponderable, en el IF se busca lo contrario, en lugar de incertidumbre, se cuenta con unos parámetros que indican cómo de exacta es la creencia.

De esta manera, si se parte de un caso en el cual el robot posee un absoluto desconocimiento, en el KF se tendría una matriz de covarianza infinita mientras que la matriz de información es nula y va aumentando a medida que capta información del estado. Es aquí donde se refleja la mayor virtud del IF, es más fácil computar datos prácticamente nulos que cercanos al infinito, dado que la situación de incertidumbre total es un escenario más típico que la de tener conocimiento de antemano.

2.3.2 Algoritmo del IF

Como se observa al comparar el IF (Figura 2-3) con el KF (Figura 2-1), las ecuaciones se transforman al realizar el cambio de variable, aunque en esencia son las mismas. De igual forma, comparten elementos en dichas ecuaciones tales como las matrices de modelos: $A_t,\ B_t,\ C_t,$ y de ruido: $R_t,\ Q_t$.

En este caso, también se cambian los parámetros que se reciben tanto como de entrada, como de salida, a su expresión canónica.

1: **Algorithm Information_filter**($\xi_{t-1}, \Omega_{t-1}, u_t, z_t$)**:**
2: $\bar{\Omega}_t = (A_t\ \Omega_{t-1}^{-1}\ A_t^T + R_t)^{-1}$
3: $\bar{\xi}_t = \bar{\Omega}_t(A_t\ \Omega_{t-1}^{-1}\ \xi_{t-1} + B_t\ u_t)$
4: $\Omega_t = C_t^T\ Q_t^{-1}\ C_t + \bar{\Omega}_t$
5: $\xi_t = C_t^T\ Q_t^{-1}\ z_t + \bar{\xi}_t$
6: return ξ_t, Ω_t

Figura 2-3. Algoritmo del filtro de Información

Al igual que en el KF se pueden diferenciar dos etapas en la figura anterior. En las líneas 2-3 se realiza la predicción y en 4-5 la actualización.

Dada su dualidad, lo que es computacionalmente complejo en uno es más liviano en otro y viceversa. De esta manera, mientras que el KF es aditivo en la predicción, el IF lo es en la incorporación de medidas. Esto es de gran ayuda a la hora de adquirir conocimiento del entorno sin necesidad de complejos cálculos.

Otra importante propiedad del IF es, gracias a su aditividad al incorporar medidas, la posibilidad de un cómputo descentralizado. En algunos casos, la información proviene de múltiples nodos los cuales pueden repartir carga computacional informando al nodo principal que ejecuta el algoritmo de los cambios resultantes que comprenderán únicamente de una suma o resta.

2.4 Filtro de Kalman Extendido (*EKF*)

2.4.1 Objetivo y fundamentos del EKF

Para funcionar, el filtro de Kalman simple había de cumplir una serie de propiedades fundamentales con objeto de que, a lo largo de todo el proceso, se mantuviera siempre una solución con forma de distribución normal sin ver modificada su esencia en cada iteración. La linealidad era una de esas propiedades, con el problema de que es un caso poco frecuente en entornos reales. De aquí nace la necesidad que soluciona el filtro de Kalman extendido.

Como bien indica su nombre, el EKF es una versión del KF extendida a sistemas no lineales, para lo cual hace uso del Teorema de Taylor.

Este teorema permite la descomposición de una función derivable en polinomios, de forma que estos suponen una aproximación a la función en torno cierto punto. La idea que trasciende de esto es que a pesar de tener una función no lineal, mientras sea derivable, se puede obtener una aproximación alrededor de cierto punto que sí se puede considerar lineal sin temor a grandes errores.

En este caso, el procedimiento es quedarse con los polinomios de hasta la derivada de primer orden y despreciar las de orden superior. Se asume un resultado que no es completamente exacto pero, en cambio, se posibilita una forma computacionalmente aceptable de implementar el filtro.

Entonces, en lugar de tener las ecuaciones lineales anteriores, ahora se parten de funciones no lineales tanto para la transición de estado como para las medidas. Por lo que se usa un modelo más generalista:

$$x_t = g(u_t, x_{t-1}) + \varepsilon_t \tag{2.5}$$

$$z_t = h(x_t) + \delta_t \tag{2.6}$$

Dado que son ecuaciones no lineales es imposible obtener como resultado una gaussiana, por lo que, mediante la linealización, se persigue una aproximada a ésta.

Finalmente, tras linealizar, resulta:

$$g(u_t, x_{t-1}) = g(u_t, \mu_{t-1}) + G_t \cdot (x_{t-1} - \mu_{t-1}) \tag{2.7}$$

$$h(x_t) = h(\overline{\mu}_t) + H_t \cdot (x_t - \overline{\mu}_t) \tag{2.8}$$

Donde G_t y H_t matemáticamente corresponden a la pendiente de la función si se representara la salida en función de la variable respecto la cual se deriva. Se denominan Jacobianos.

$$G_t = \frac{\partial g(u_t, x_{t-1})}{\partial x_{t-1}} \qquad H_t = \frac{\partial h(x_t)}{\partial x_t} \tag{2.9}$$

2.4.2 Algoritmo del EKF

El algoritmo de este filtro, representado en la Figura 2-4, es bastante parecido al del filtro de Kalman, el cual aparece en la Figura 2-1. Se pueden observar prácticamente las mismas etapas: predicción y actualización gracias a la información de los sensores.

1: **Algorithm Extended_Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):
2: $\bar{\mu}_t = g(u_t, \mu_{t-1})$
3: $\bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + R_t$
4: $K_t = \bar{\Sigma}_t \, H_t^T (H_t \, \bar{\Sigma}_t \, H_t^T + Q_t)^{-1}$
5: $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
6: $\Sigma_t = (I - K_t \, H_t) \, \bar{\Sigma}_t$
7: return μ_t, Σ_t

Figura 2-4. Algoritmo del filtro de Kalman extendido

Como muestra la figura 2-4, las ecuaciones que modelan tanto la transición de estado como la predicción de las medidas pasan a ser no lineales. Aparte, los Jacobianos sustituyen a las matrices lineales.

Tabla 2–1. Comparación entre KF y EKF

	Kalman filter	EKF
state prediction (Line 2)	$A_t \, \mu_{t-1} + B_t \, u_t$	$g(u_t, \mu_{t-1})$
measurement prediction (Line 5)	$C_t \, \bar{\mu}_t$	$h(\bar{\mu}_t)$

A pesar de que el EKF no está matemáticamente demostrado como óptimo, lo cual sí ocurre con el KF, está demostrado que a efectos prácticos cumple con lo requerido, además de ser simple y computacionalmente más eficiente que otros métodos. De aquí que se haya popularizado ampliamente su uso en robótica, donde pocos modelos son realmente lineales.

Su eficacia depende básicamente de dos factores: el nivel de incertidumbre y la no linealidad en torno al punto de funcionamiento.

2.5 Filtro de Información Extendido (*EIF*)

Poco se puede decir sobre el trasfondo conceptual de este filtro, dado que surge exactamente con la misma meta que el EKF, adaptar su funcionalidad a sistemas no lineales, con la diferencia de que utiliza la parametrización canónica mostrada en el IF. Por ello, se procede a ver directamente el algoritmo en sí.

2.5.1 Algoritmo del EIF

La implementación de este filtro en particular tiene un inconveniente por el cual es menos popular que el EKF. Se trata de que para realizar la predicción tanto del estado como de las medidas, aun usando diferentes parámetros, hace uso de las ecuaciones no lineales que requieren como entrada el estado del sistema. Por ello, observando la Figura 2-5, recurre a un cómputo extra (líneas 2 y 5) para recuperar el estado, lo que hace menos eficiente el algoritmo.

1: **Algorithm Extended_information_filter**$(\xi_{t-1}, \Omega_{t-1}, u_t, z_t)$**:**
2: $\mu_{t-1} = \Omega_{t-1}^{-1}\ \xi_{t-1}$
3: $\bar{\Omega}_t = (G_t\ \Omega_{t-1}^{-1}\ G_t^T + R_t)^{-1}$
4: $\bar{\xi}_t = \bar{\Omega}_t\ g(u_t, \mu_{t-1})$
5: $\bar{\mu}_t = g(u_t, \mu_{t-1})$
6: $\Omega_t = \bar{\Omega}_t + H_t^T\ Q_t^{-1}\ H_t$
7: $\xi_t = \bar{\xi}_t + H_t^T\ Q_t^{-1}\ [z_t - h(\bar{\mu}_t) - H_t\ \bar{\mu}_t]$
8: return ξ_t, Ω_t

Figura 2-5. Algoritmo del filtro de Información extendido

2.6 Filtro de Partículas (PF)

2.6.1 Filtros no paramétricos

Los filtros paramétricos representan la creencia del estado como un vector de media (la esperanza) y su covarianza, es decir, como una campana de Gauss en torno a una única solución. Esto implica asegurar que dicha forma estadística no se va a perder a lo largo de todo el proceso, de ahí la necesidad de asegurar las condiciones mencionadas en el apartado 2.2.1.

Como alternativa existen los filtros no paramétricos. Estos se dedican a representar la solución con un número finito de valores, que si pudieran tender a infinito adoptarían a la perfección la curva de probabilidad de la solución, lo cual permite resolver tanto problemas lineares como no lineares, con ruido no estrictamente gaussiano, etcétera. Esta nueva disposición toleraría sistemas multimodales, es decir, curvas de solución con medias en varias posiciones.

Los filtros más señalados dentro de esta familia son: el filtro de histogramas y el de partículas.

El primero consiste en dividir el espacio de estados en trozos (histogramas) y darle a cada uno, cierta probabilidad según más cercano a la solución esté. Mientras que el segundo dedica un montón de muestras a adoptar la forma de la curva que toma la solución en el espacio de estados.

En concreto, en este trabajo se ha estudiado e implementado el filtro de partículas.

2.6.2 Características del PF

Dado que pertenece a la familia de filtros no paramétricos, comparte la característica de ser capaz de trabajar con soluciones multimodales, además su flexibilidad a la hora de buscar soluciones permite al filtro soportar todo tipo de modelos (tanto de movimiento como de medidas), independientemente de su linealidad o ruido por el que vengan afectados.

Lo anterior es posible gracias a que coloca muchas partículas en torno a posibles soluciones, de forma que va escogiendo siempre las que más probabilidad de ser solución sean.

El gran pero que esconde es su enorme carga computacional. Como se menciona en el primer apartado, la exactitud de esta familia depende directamente de la capacidad de representar multitud de elementos que puedan adaptarse a la curva del espacio de estados. Esto implicaría que para una exactitud completamente fidedigna, es necesario un número de elementos infinito.

De igual manera, se demuestra que con un número razonable de partículas es capaz de acercarse a la solución con una exactitud razonablemente aceptable para la mayoría de escenarios. Se recurre a una solución de compromiso entre carga computacional y exactitud.

2.6.3 Algoritmo del PF

Este algoritmo funciona mediante la colocación aleatoria de un montón de partículas, cada una representando un vector de estado de igual forma que el filtro de Kalman posee su media. Así, se tendrían tantos vectores de estado como partículas, los cuales siguen un proceso afín al filtro de Kalman, con la particularidad de que a cada partícula se le añade otro parámetro: el peso.

Este peso implica la verosimilitud de la partícula respecto la realidad, es decir, a mayor peso, mayor probabilidad de que la partícula esté cerca de la solución (al menos, una de ellas).

Observando la Figura 2-6, en la línea 4, las partículas evolucionan en una fase de predicción (idéntica a la transición de estados producida en el KF, ecuación 2.1) y, en la línea 5, adquieren un peso gracias a una función en la que se involucran los sensores (mencionado como Modelo de Similitud en la Figura 2-7). Recordar que en todo sistema, las acciones, movimientos o cambios aportan incertidumbre, mientras que recolectar información del exterior hace lo contrario, el sistema recupera noción de la situación (fase de actualización), exactamente igual a lo que pasa en el filtro de Kalman.

1: **Algorithm Particle_filter**($\mathcal{X}_{t-1}, u_t, z_t$):
2: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
3: for $m = 1$ to M do
4: sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
5: $w_t^{[m]} = p(z_t \mid x_t^{[m]})$
6: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7: endfor
8: for $m = 1$ to M do
9: draw i with probability $\propto w_t^{[i]}$
10: add $x_t^{[i]}$ to $\mathcal{X}_t$
11: endfor
12: return $\mathcal{X}_t$

Figura 2-6. Pseudoalgoritmo del filtro de Partículas

Típicamente, en el proceso de asignación de pesos, las partículas obtienen un valor en una función inversamente proporcional al error entre la medida y la medida predicha (igual que en el KF, esta medida es la que teóricamente se obtendría si el vector de estado real fuera igual que el del cual se predice).

Estos valores no están normalizados, por lo que es necesario ajustar de tal manera que la suma total de los pesos sea unitaria, teniendo así, una verdadera función de distribución probabilística.

Tras estas operaciones, las cuales, es importante señalar, se han de realizar a todas y cada una de las partículas, el filtro realiza una media ponderada de las partículas y entra en un proceso de resampling (remuestreo).

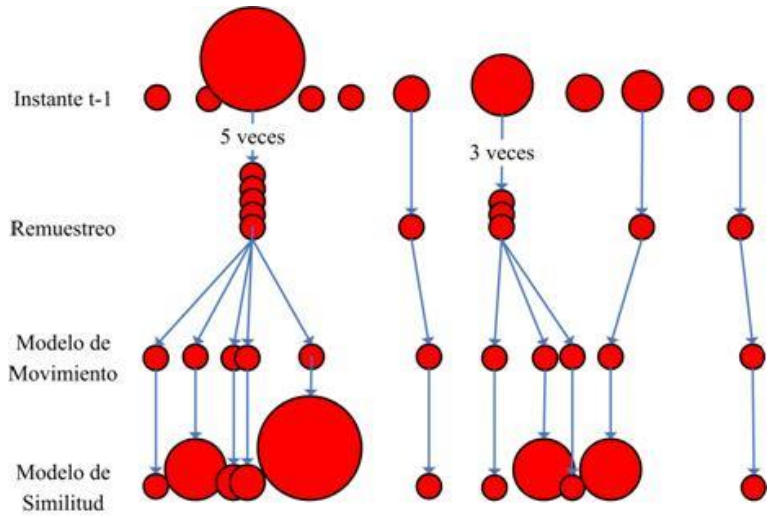


Figura 2-7. Pseudoalgoritmo del filtro de Partículas

2.6.4 Resampling

Esto significa que el filtro da prioridad a las partículas con mayor peso, llegando a limpiar las más fútiles, de forma que las reasigna cerca de las partículas pesadas, para aumentar la precisión de la solución. Este proceso se puede hacer de múltiples maneras, dando la libertad al programador de escoger el que vea más conveniente. Importante indicar que tras este proceso, se deben volver a normalizar los pesos de nuevo.

En este caso, se ha escogido implementar el método ***Resampling Wheel***[2] **(**Figura 2-8**)**:

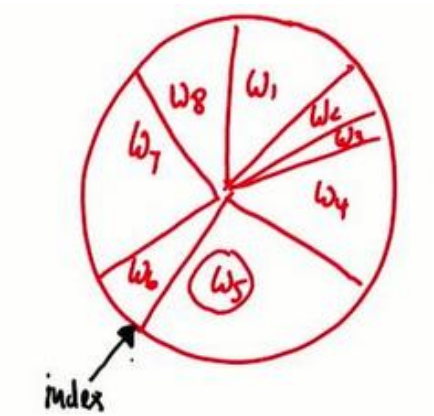


Figura 2-8. Introduciendo el algoritmo de remuestreo

Este método consiste en ordenar todas las partículas en una ruleta, una suerte de rueda de la fortuna teniendo cada partícula un área proporcional a su peso.

[2] Artificial Intelligence for Robotics. Udacity Course CS373. S. Thrun / D. Stavens / M. Sokolsky

El algoritmo a seguir en este proceso se muestra en la Figura 2-9, donde N es el número total de muestras (o partículas), recogidas en el vector P con un peso ω_i.

El índice empezaría en un número entero aleatorio dentro del conjunto y se ejecutaría el bucle N veces. Dentro del bucle una variable auxiliar (β) incrementa su tamaño, también de forma aleatoria entre nada y el doble del peso de la mayor partícula. El criterio de selección de la partícula es comprobar si el tamaño de esta variable auxiliar es mayor que el peso, lo que gráficamente indicaría (colocando β a partir del índice actual) que β ha sobrepasado la zona correspondiente a la partícula actual. Si esto ocurriera, β pierde el tramo de la partícula y el índice se coloca en la siguiente (se vuelve a iterar hasta que ocurra lo contrario para no dejar ninguna partícula sin reasignar, vigilando que el índice no salga del conjunto). El caso contrario sería que β, a pesar del incremento aleatorio, aún permaneciera dentro del área de la misma partícula, entonces esa partícula es seleccionada para sobrevivir al proceso.

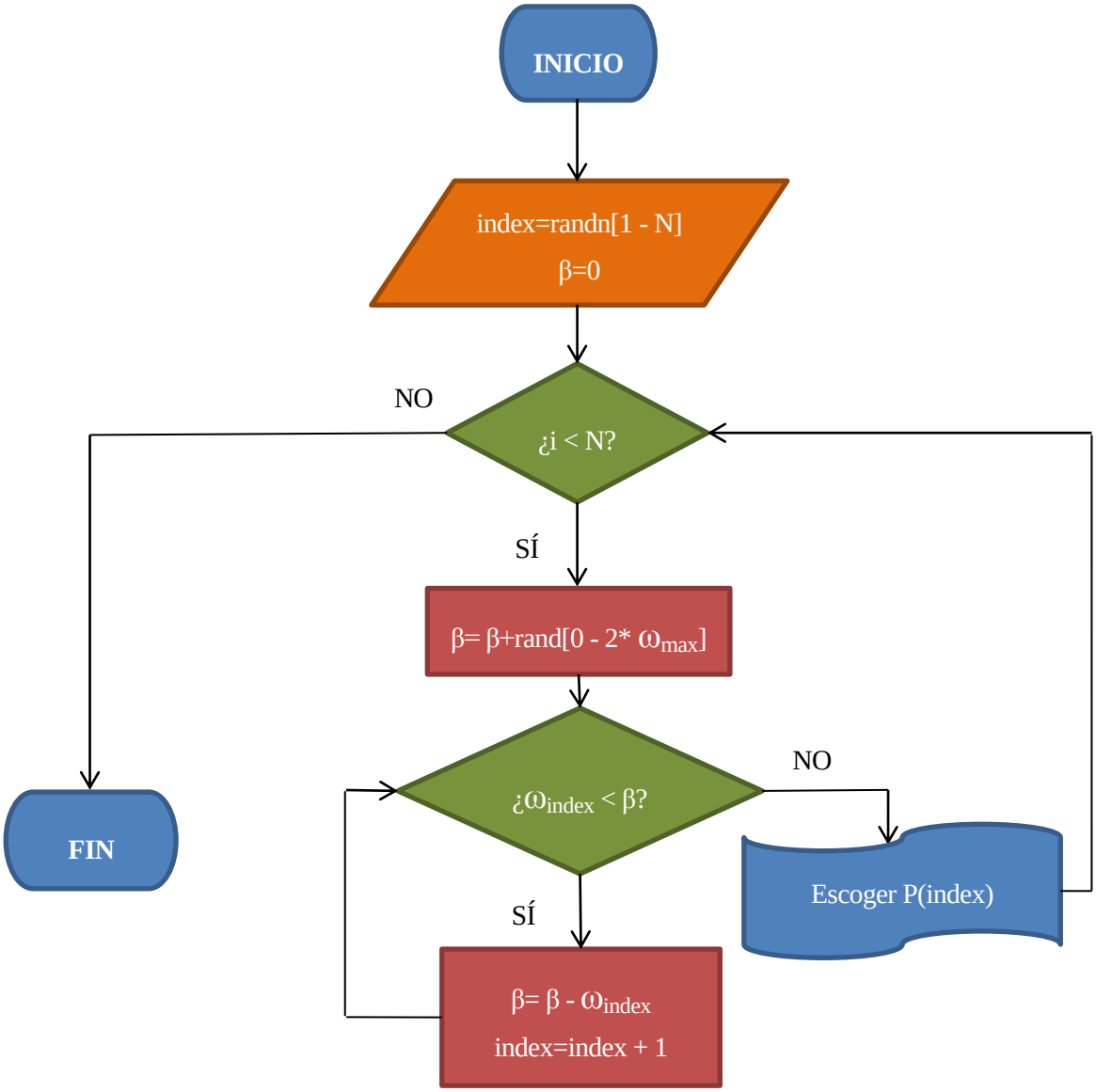


Figura 2-9. Resampling Wheel, diagrama de flujo

2.7 De la teoría a la práctica

En este apartado se va a describir con detalle los elementos genéricos que se han ido mencionando en páginas anteriores, de manera que el lector esté más familiarizado en particular con el problema a enfrentar.

Así, se encuentra que el vector de estado en cierto instante, siendo éste las variables que definen el sistema, en el problema de localización, está compuesto por las posiciones y velocidades en el plano, definidas en el instante *t*.

Este parámetro, μ, para el filtro sería la creencia, la cual viene acompañada con otro parámetro que indica cuán acertada es: la covarianza, Σ. Esta matriz tiene la característica de ser simétrica, significando los elementos fuera de la traza, cuanto afecta el error de una variable ajena a otra.

$$\mu_t = \begin{pmatrix} p_x \\ p_y \\ v_x \\ v_y \end{pmatrix} \qquad \Sigma = \begin{bmatrix} S_x & S_{xy} & S_{xvx} & S_{xvy} \\ S_{yx} & S_y & S_{yvx} & S_{yvy} \\ S_{vxx} & S_{vxy} & S_{vx} & S_{vxvy} \\ S_{vyx} & S_{vyy} & S_{vyvx} & S_{vy} \end{bmatrix} \tag{2.10}$$

A lo largo de todo el capítulo se ha estado mencionando ciertas matrices que vienen a representar tanto la evolución del sistema, como la relación entre las medidas y las coordenadas del vector de estado. Esto es, las ecuaciones de la predicción y de la actualización, respectivamente.

Pues bien, en este caso, dado que se carecen de acciones de control, como se indica en el capítulo introductorio, la matriz B de las ecuaciones de predicción es nula, lo que significa que el estado no va a ser modificado mediante ningún actuador.

Por otro lado, la matriz A recoge el modelo matemático del sistema, de tal forma que, a mejor aproximación de dicho modelo con la realidad, mayor exactitud habrá en la predicción del siguiente vector de estado, ya que recoge su cálculo a partir del estado anterior. Así pues, particularizando, en este trabajo se ha escogido el modelo de un movimiento rectilíneo uniforme (MRU), lo cual aporta errores, al no ser exactamente lo seguido por el sistema, pero tiene la ventaja de sacar a relucir la gran utilidad del filtro a la hora de autocorregirse.

$$\mathrm{A} = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.11}$$

Lo cual, teniendo en en cuenta las variables del vector de estado, viene a representar las siguientes ecuaciones:

$$\begin{aligned} x_t &= x_{t-1} + T\, x_{t-1} \\ y_t &= y_{t-1} + T\, y_{t-1} \end{aligned} \tag{2.12}$$

Las velocidades se suponen invariantes.

Este modelo trae consigo un cierto error, que aparece en el algoritmo como la matriz R, representado por un ruido blanco, *v*, que previamente debe haberse estudiado para conocerlo.

$$\mathrm{R} = \begin{bmatrix} v^2 & 0 & 0 & 0 \\ 0 & v^2 & 0 & 0 \\ 0 & 0 & (T\,v)^2 & 0 \\ 0 & 0 & 0 & (T\,v)^2 \end{bmatrix} \tag{2.13}$$

En lo que respecta a la actualización, las ecuaciones dependen del tipo de sensor que se use. Pero en cualquier caso, la matriz C sirve para hallar un vector de medidas predichas, es decir, el resultado ideal que deberían mostrar los sensores en caso de coincidir a la perfección con el vector de estado que resulta en la fase de predicción.

Por ejemplo, a lo largo del trabajo se han empleado, básicamente, sensores GPS y de rango.

En ambas fases hay que tener consideración de que las ecuaciones que las gobiernan sean lineales, lo cual gracias a un modelo tan simple, como es el MRU, no presenta dificultad. Caso distinto a la hora de implementar un sensor de rango en las ecuaciones, dado que la relación entre distancias y posición es no lineal.

Lo primero es definir el vector de medidas, *z*. En este caso para un único GPS:

$$\mathrm{z} = \begin{pmatrix} p_{x,t} \\ p_{y,t} \\ p_{x,t-1} \\ p_{y,t-1} \end{pmatrix} \tag{2.14}$$

Para el caso de GPS no hay complicación, dado que la medida es directamente la posición en el plano. En caso de incorporar más, aumentan tanto las dimensiones del vector *z* como de la matriz C.

$$\mathrm{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & -T & 0 \\ 0 & 1 & 0 & -T \end{bmatrix} \tag{2.15}$$

Donde se ha seguido la misma relación entre velocidad y posición de la que parte el MRU, ecuaciones 2.12:

$$\begin{aligned} p_t &= p_{t-1} + T\, v_t \\ p_{t-1} &= p_t - T\, v_t \end{aligned} \tag{2.16}$$

Al igual que el modelo, los sensores tienen un ruido blanco gaussiano asociado que se incorpora de forma aditiva. En este caso, la matriz Q es una matriz diagonal 4x4, siendo todos los elementos de la traza iguales a w_g^2, debido al ruido en el GPS. La única diferencia al incorporar los sensores de rango es que aumenta la matriz cuadrada de dimensión, pero con diferentes valores dado que el sensor de rango tiene un ruido diferente, w_r^2.

Es a la hora de incorporar en la matriz C las ecuaciones del sensor de rango, donde se complica el asunto. Exponiendo como ejemplo un único sensor de rango, ya que para incluir más sólo hay que ampliar el tamaño de los elementos de forma iterada.

$$z_t = \begin{pmatrix} d_t \\ d_{t-1} \end{pmatrix} \tag{2.17}$$

En este momento, el producto que resultaría en la predicción de la medida (Ecuación 2.2: $C_t x_t$, a partir de ahora $h(x_t)$ para mantener la coherencia con el apartado 2.4, ecuación 2.9) tomaría el siguiente contenido:

$$h(x_t) = \begin{bmatrix} \sqrt{(p_x - x_r)^2 + \left(p_y - y_r\right)^2} \\ \sqrt{(p_x - T\,v_x - x_r)^2 + \left(p_y - T\,v_y - y_r\right)^2} \end{bmatrix} \tag{2.18}$$

Aquí, p_r son las coordenadas del nodo sensor, recordando que son conocidas de antemano. Estas ecuaciones están estrechamente relacionadas con las anteriores, con la diferencia que se saca el valor absoluto (la distancia ortonormal) entre el nodo y el robot para poder trabajar con las medidas sacadas por en el sensor de rango.

A simple vista se observa la no linealidad de las ecuaciones. Por ello se ha de recurrir a la versión extendida de los filtros, lo que implica el uso de Jacobianos (G y H, según la fase). En este caso, sólo se necesita el de la fase de actualización debido a que el modelo sí es lineal:

$$H = \begin{bmatrix} \frac{p_x - x_r}{d_t} & \frac{p_y - y_r}{d_t} & 0 & 0 \\ \frac{p_x - Tv_x - x_r}{d_{t-1}} & \frac{p_y - Tv_y - y_r}{d_{t-1}} & \frac{-T(p_x - Tv_x - x_r)}{d_{t-1}} & \frac{-T(p_y - Tv_y - y_r)}{d_{t-1}} \end{bmatrix} \tag{2.19}$$

Dado que las ecuaciones anteriores tienen elementos de las ecuaciones originarias, se ha simplificado su notación considerando que $z_t = h(x_t)$ (Ecuaciones 2.6, 2.17, 2.18). Claro que sólo es estrictamente real en caso de que las medidas estuvieran exentas de ruido.

A la hora de computar este código, estas variables asociadas a la distancia vienen, realmente, sustituidas por el vector de medidas predichas, $\bar{z}_t$. Es decir, estas distancias son las que tendrían el sistema si fuera el vector de estado predicho, el real (por ello, lo mencionado en el párrafo anterior del caso ideal en ausencia de ruido).

2.8 Conclusiones

Para comenzar, comentando sobre la variedad de filtros paramétricos, todos basados en el filtro de Bayes, se pueden sacar las siguientes ideas principales:

- La información resultante se puede representar de dos formas: por la parametrización de momentos o la canónica. Ambas son duales y están relacionadas de forma que se puede pasar de una a otra con relativa facilidad. Las virtudes de los filtros basadas en una parametrización son desventajas en los otros, y viceversa. De esta forma, la actualización del control en el KF es fácil, mientras que en el IF lo es la incorporación de información proveniente de los sensores.

- Además, dado que son filtros bayesianos, para asegurar la pureza de la gaussiana a lo largo de todas las iteraciones, se deben de cumplir las propiedades de los sistemas gaussianos lineales, descritas en el apartado 2.2.1, además de ser de solución unimodal. Tanto la bondad del modelo como del sensor vienen afectadas por ruido de esta naturaleza.

- Ambos filtros tienen posibilidad de enfrentarse a problemas con sistemas no lineales gracias a la linealización mediante las series de Taylor. Se toma la pendiente y se evalúa en un punto de funcionamiento, esto es el Jacobiano el cual sustituirá las antiguas matrices que modelaban el sistema. Estos filtros modificados son denominadas extendidos.

Esta serie de características vienen enormemente superadas con los filtros no paramétricos. En concreto, el filtro de partículas es capaz de adaptarse a todo tipo de situación debido a las numerosas muestras capaces de tomar la forma de cualquier tipo de curva de probabilidad, sin necesidad de ser una simple gaussiana unimodal. Para este filtro, es innecesario el uso de Jacobianos, dado que soporta no linealidades. Por el contrario, como se menciona en su respectivo capítulo, requieren de una carga computacional, en ocasiones, inviable. Lo cual implica la reducción de este número de muestras, sacrificando en su lugar, exactitud.

Destacar que es interesante la flexibilidad a la hora de programar este filtro tanto en la elección del criterio de ponderación de los pesos como en el proceso de remuestreo, lo que confiere multitud de oportunidades de cara a su implementación.

3 IMPLEMENTACIÓN SOFTWARE

Si la depuración es el proceso de eliminar errores, entonces
la programación debe ser el proceso de introducirlos.

- Edsger W. Dijkstra -

3.1 Introducción

El propósito de este capítulo es recoger toda la preparación inicial en torno a la ejecución del presente trabajo, para que, de esta forma, el lector tome conciencia tanto del procedimiento realizado como del escenario en el que se produce.

Como se menciona en el capítulo inicial, el trabajo ha sido desarrollado en dos partes bien diferenciables en función de las herramientas utilizadas (Figura 3-1). Mientras que en la primera se hace uso de un software de uso extendido, Matlab, del cual no es necesario argumentar mucho, la segunda se basa en software especializado en el mundo de la robótica, por lo cual, ha de explicarse de forma explícita el funcionamiento intrínseco a ROS y, en menor medida, Gazebo.

Figura 3-1. Software empleado: a) Matlab, b) ROS, c) Gazebo

3.2 Descripción del problema, Matlab

Adelantando un poco la estructura del siguiente capítulo, la sucesión de simulaciones que se llevan a cabo vienen a mostrar la funcionalidad de todos los algoritmos mostrados previamente.

De esta manera, se encuentra que para mostrar la versión simple del filtro de Kalman, las ecuaciones han de ser obligatoriamente lineales, por lo que en las versiones más sencillas de las pruebas, los únicos sensores disponibles vendrán dados por medidas estilo GPS dada la naturaleza de sus ecuaciones. Todo esto está explicado con más detalle en el apartado 2.7, *De la teoría a la práctica*.

Una vez involucrados en los filtros más completos, ya se dispondrá de un escenario con un mínimo de 3 nodos WSN, y por mostrar la potencial funcionalidad del código, se complementará con GPS, demostrando de esta forma la gran utilidad que presentan estos filtros con el fin de fusión sensorial.

Todos los filtros programados comparten cierta estructura común, la cual está trazada en el diagrama de flujo de la Figura 3-2. Repitiendo que el objeto de este trabajo es exclusivamente la localización, es decir, el cálculo de la posición de forma pasiva sin influir en el robot, es comprensible que siga una estructura tan simple como es un bucle constante que ejecuta el filtro hasta que se dé por finalizado.

Una vez dentro del algoritmo, la fase de actualización es algo más sofisticada, teniendo el programa capacidad de discriminar ciertos sensores en caso de que haya habido algún problema en ellos o su medida llegue falseada.

También es posible mejorar la fase anterior, mediante cambios del modelo según la situación, como, por ejemplo, incorporar un modelo estático si se perciben velocidades casi nulas.

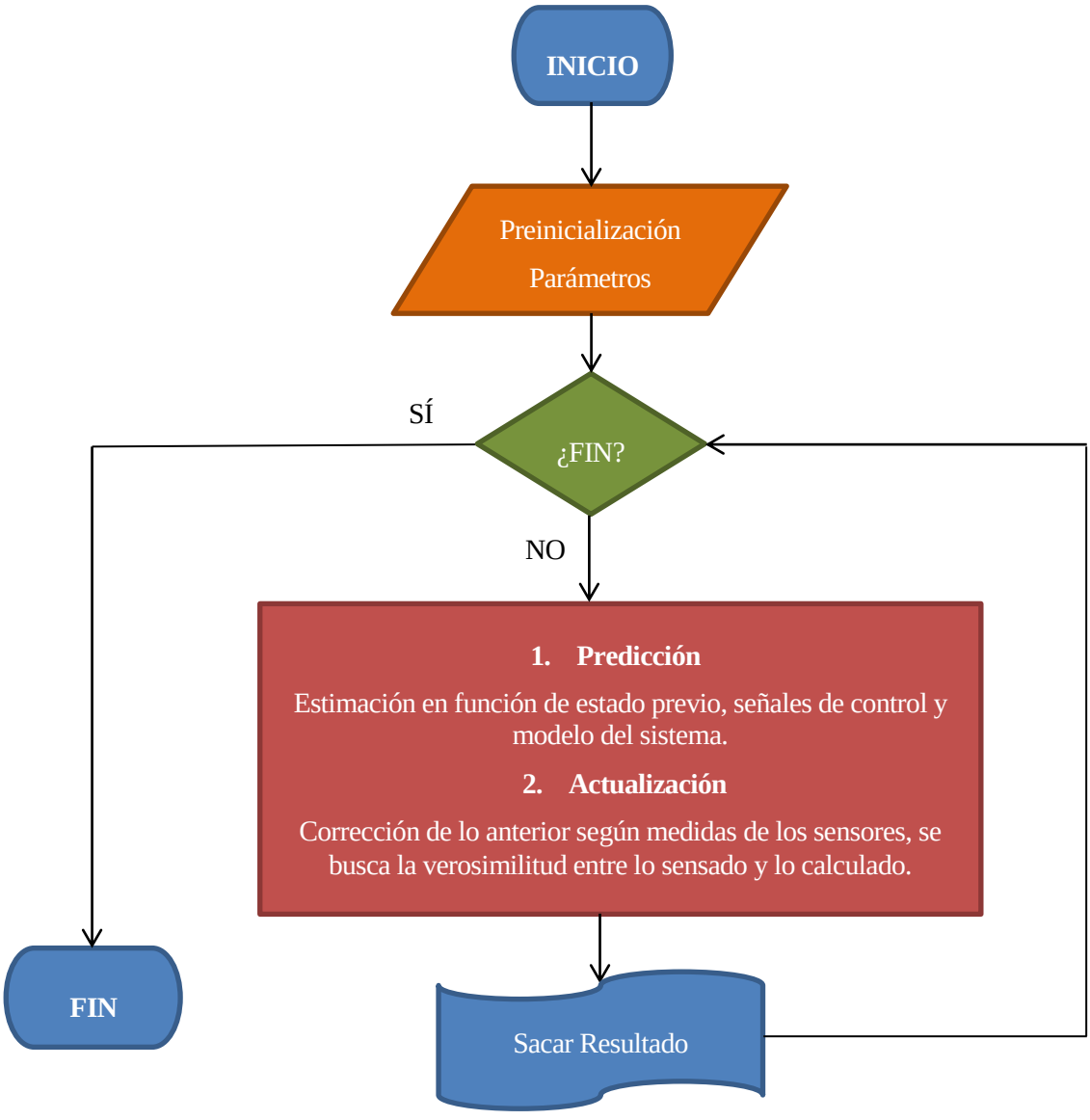


Figura 3-2. Diagrama de flujo de la programación de filtro genérico

3.2.1 Particularidades en el PF

Dado que es un filtro de una naturaleza distinta, el filtro de partículas tiene ciertos detalles a la hora de ser implementado de los que difiere con el resto.

La mayor y principal diferencia es que, en lugar de media y covarianza (filtros paramétricos), este filtro puede representar tantas medias distintas como muestras tenga, cada una con un valor llamado peso que equivaldría a su covarianza (puesto que nos indica lo acertada o no que está la partícula). Este hecho requiere que dentro del bucle que se compruebe la finalización del programa, exista otro, para que dentro de cada iteración, se realice una ejecución del algoritmo partícula a partícula (o muestra a muestra).

Mientras que la fase de predicción es idéntica (aunque sin el uso de Jacobianos, puesto que sólo desea recoger la predicción de la media, lo que incluía las ecuaciones lineares o no, tal cual), la fase de actualización sí cambia. Esto es lo que se llama en la Figura 2-7, Modelo de Similitud, proceso donde el peso de las partículas es modificado pero no, su vector de estado.

En este trabajo, el criterio para este proceso ha sido calcular el vector de medidas predichas (esto implica, un vector por muestra), de forma que cuanto menor sea la suma de errores absolutos entre este vector y las medidas recibidas por los sensores, mayor peso tendrá la muestra. Este vector tendrá una dimensión igual, como mínimo, al número de sensores de rango (denotado *nsr* en la ecuación 3.1). En caso de querer tener en cuenta *n*-medidas anteriores, su dimensión se multiplicaría *n* veces.

$$w_i = 100 \cdot \left(\sum_{k=1}^{nsr} \left| \bar{z}_{i,k} - z_k \right| \right)^{-1} \tag{3.1}$$

Lo obtenido anteriormente, ha de normalizarse (ecuación 3.2, donde *np* es la cantidad total de partículas) con objeto de que esta sucesión de partículas adopte verdaderamente la forma de una función de distribución estadística.

$$\widehat{w}_i = w_i \cdot \left(\sum_{i=1}^{np} w_i \right)^{-1} \tag{3.2}$$

En el caso, de que se deseara incorporar otro tipo de sensor, por ejemplificar, GPS, lo único que habría que hacer es extender la dimensión de ambos vectores de medidas, manteniendo la coherencia en la forma de calcular la medida predicha con la medida real.

Otra sección de importancia en el código es la implementación del proceso de resampling. Éste es vital, para poder descartar partículas totalmente erróneas, sustituyéndolas por otras más cercanas a las partículas pesadas. Como se menciona en el apartado 2.6.4, y se describe gráficamente en la Figura 2-9, el algoritmo implementado para esta función ha sido el algoritmo *Resampling Wheel*. Además de implementar dicho diagrama de flujo, se incorpora un simple método para poderse modificar la frecuencia de ejecución del remuestreo, logrando relajar el esfuerzo computacional.

3.3 Presentando ROS

Antes de mostrar cómo se implementan los filtros en este entorno, es debido hacer una correcta introducción sobre dicho software.

ROS se presenta como un sistema operativo enfocado a la robótica, de forma que surge en un intento de universalizar todo el trabajo que hay detrás de un robot como pueden ser la incorporación, uso y control de actuadores; recepción, tratamiento y respuesta a señales externas provenientes de los sensores; esquematización de los recursos necesarios para funciones tales como la navegación, detección de obstáculos, creación de mapas, comunicación tanto externa como interna, etcétera.

Así pues, ROS es considerado un middleware que facilita enormemente la labor tras cualquier aplicación robótica, gracias a su conjunto de librerías y herramientas prediseñadas tanto por el staff como por su amplia comunidad, cosa común en los mejores softwares libres.

3.3.1 Estructura de ROS

Es interesante entender el funcionamiento interno de ROS para comprender mejor las posibilidades que ofrece. Por ello, se va a hacer cierto énfasis en los elementos que lo componen (mostrados en la Figura 3-3).

- **Nodos:**

La unidad básica computacional de este software es el nodo, en él se produce toda la computación además de encargarse de gestionar la información entrante y saliente. Cada nodo tiene asociado un script (escrito en Python o C++) que le dicta su comportamiento y su configuración.

Siempre ha de haber un nodo maestro que se encarga de llevar un registro de los nodos dados de alta y de baja, lo que evita problemas del estilo establecer comunicación con un nodo inexistente.

- **Tópicos:**

Así pues, hay comunicación inter-nodal gracias a otro elemento de la red: los tópicos. Estos harían la función de canal entre nodos, a través del cual viajan los mensajes. Para conectar dos nodos es necesario preconfigurarlos, de tal forma que hay que indicar a qué tópico va a ser conectado y qué tipo de mensajes va a recibir o enviar.

En función de si un nodo publica o recibe mensajes de un tópico, se considera *publisher* o *subscriber*, respectivamente (publicador o suscriptor).

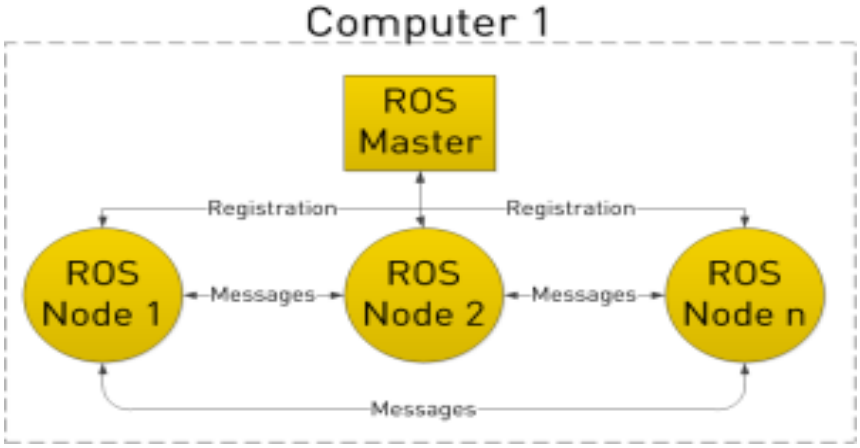


Figura 3-3. Estructura básica en ROS

- **Mensajes:**

Para poder enviar un mensaje es imprescindible definir cuál tipo es. Por dar un ejemplo, si es un mensaje con datos tipo entero, flotantes, vector, etcétera. Una de las virtudes de ROS es que tiene de serie un montón de tipos de antemano creados y adaptados a las necesidades del usuario, además de tener categorías especiales de mensajes para poder ser agrupados y localizados con facilidad según el tipo de información que le viene asociada como las medidas de odometría, por poner un caso.

El usuario también es capaz de crear y/o personalizar sus propios mensajes para cubrir sus necesidades con relativa simpleza.

- **Servicios:**

Aparte de los tópicos, hay otra forma de comunicación implementada en ROS, los servicios. Estos tienen la característica de tener dos fases: una llamada y una respuesta. Es decir, tras invocar al servicio, se recibe una respuesta. Es una forma más limpia de pedirle información a un nodo, aunque para ello, en la naturaleza (su script) del nodo debe aparecer el servicio y que tipo de información devuelve como respuesta.

3.4 Entorno gráfico

Como se ha mencionado, en la segunda parte el trabajo se derrollará en un entorno visual simulado por Gazebo. Este software es una herramienta visual para representar escenarios y robots, que permite una fácil conexión con ROS.

- **Robot:**

En este caso, se ha escogido el robot terrestre Pioneer 3-AT, el cual tiene una topología skid-steering, esto es, como la diferencial (en la cual el robot sólo controla las velocidades de dos ruedas, una en cada lado, teniendo que desplazarse mediante su uso combinado) pero con mayor número de ruedas. Sigue teniendo las mismas variables de control que el robot diferencial.

El uso de este robot es posible gracias a un paquete proporcionado por la comunidad de ROS, una demo de la navegación con este modelo, en la que viene todo preparado para manejar y leer la información más importante relacionada con el Pioneer.

El desplazamiento del robot está accesible desde un panel de control que incorpora la demo (GUI) o bien, con un nodo extra incorporado, que hará las funciones de leer desde teclado para mandar señales de acción al robot.

- **Escenario:**

El lugar donde se desplaza el Pioneer es un escenario también creado de antemano, muy tradicional en el uso de Gazebo: Willow´s Garage (el garaje de Willow). Consiste en una gran casa, con muchas habitaciones y recovecos que suponen todo un reto para algoritmos de navegación autónoma y mapeado, como se puede observar en la Figura 3-4.

- **WSN:**

La red de nodos no tiene representación gráfica. Su presencia se limita al funcionamiento interno en el código, donde su posición es arbitrariamente elegida, siempre cuidando de no dejar al robot fuera del rango de alcance.

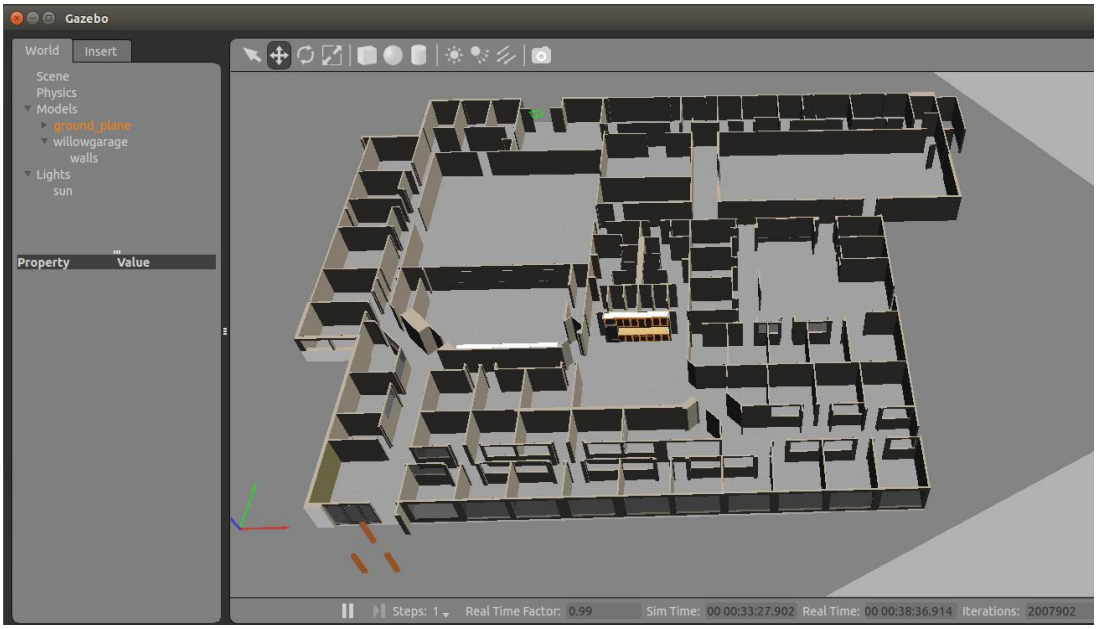


Figura 3-4. Entorno de Gazebo, Willow´s Garage

3.5 Implementación virtual con ROS/Gazebo

Gazebo se encarga del apartado externo (representación gráfica, física del entorno, obstáculos…), lo que si está bien configurado, sería el medio de percibir información del entorno y mandarla a través de nodos dedicados a esto. De igual manera que hay nodos dedicados a reflejar las órdenes de actuación en los motores, por ejemplificar, y hacer que el robot se mueva visualmente.

Detrás de todo, está la red de ROS, en la cual los nodos están continuamente trabajando para recibir y transmitir la información necesaria para que todo funcione como es debido, debiendo haber cierta coherencia entre lo que se representa por Gazebo y los datos internos que manejan los nodos.

Para hacer más clara la comprensión de la Figura 3-5, se explicará el procedimiento llevado a cabo en cuatro bloques:

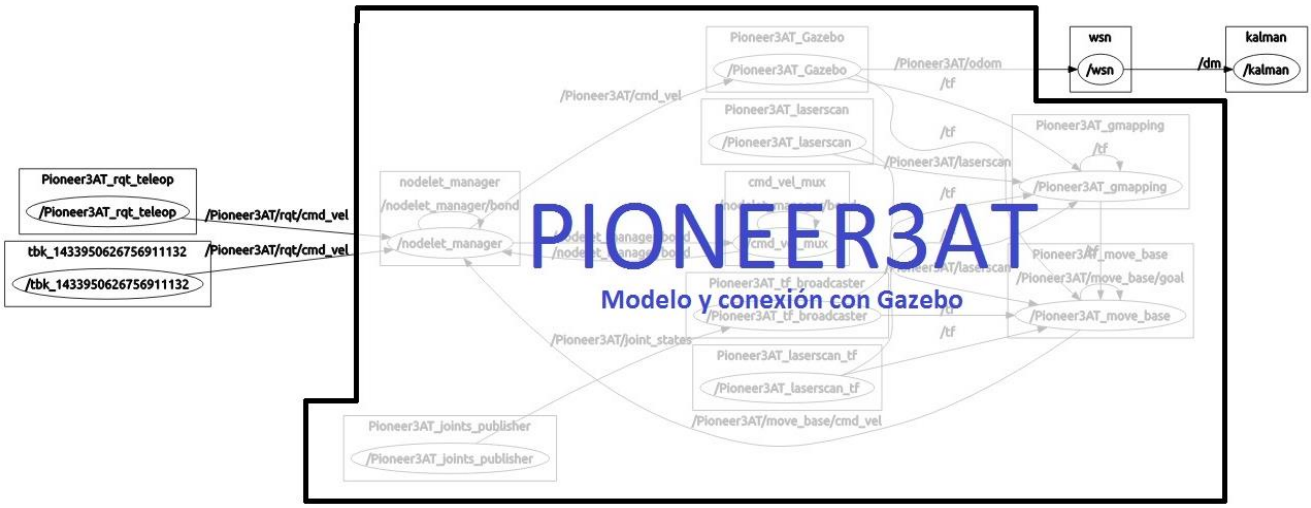


Figura 3-5. Esquema de interconexión entre nodos y tópicos

- **Kalman/EIF/PF:**

Este es el principal nodo de procesamiento de los algoritmos desarrollados a lo largo de este trabajo, habiendo sido más correcta otra denominación más genérica como filtro o predictor.

El nodo está suscrito al tópico /*dm*, que recibe las medidas de distancia de los sensores de rango, y publica en /*media* el vector de estado tras ejecutar el algoritmo.

Su funcionamiento es ése, básicamente, configura todos los parámetros necesarios en su inicialización y luego queda atrapado en un bucle, leyendo en cada iteración si hay nuevas medidas y ejecutando ambas fases del filtro en todo momento.

Para recibir mensajes los nodos disponen de una función *callback* a la que llama si ha llegado un nuevo mensaje en los puntos de comprobación (checkpoints: en ros, funciones *spin* o *spinOnce*).

Detallando un poco más el tipo de datos sobre el que se trabaja, el nodo recibe un vector de medidas (distancias) con formato de mensajes *std_msgs::Float64MultiArray*. De igual manera, a la salida publica otro mensaje del mismo tipo, pero con el vector de estado calculado (llamado media).

Computacionalmente es el nodo más complejo de los creados por el autor, debido al empleo de matrices en los cálculos, siendo necesario hasta realizar inversiones de éstas. Por ello, se requiere el uso de una librería de matrices externa a ROS: <*Eigen*>.

- **WSN:**

Debido al carácter simulado del trabajo, se ha decidido por simplicidad reunir todos los nodos WSN en un solo programa (nodo de ROS).

Éste se encuentra conectado entre el bloque del Pioneer y el filtro: publica en el tópico, anteriormente mencionado, /*dm*, y está suscrito a cierta información que proporciona el bloque del robot: /*Pioneer3AT*/*Odom*.

Este último tópico es interesante para el nodo porque le permite leer la posición absoluta en la que se encuentra el objetivo. Conocer esto es necesario, únicamente, porque cuando se habla de trabajo simulado, ha de simularse hasta las medidas sensadas. Para ello, el nodo WSN artificialmente crea las distancias, mediante un sencillo cómputo matemático, dadas por conocidas tanto la posición absoluta del robot como la de los sensores. Cosa que, efectivamente, es cierta, dado que la posición de los sensores viene predefinida por el usuario y la del robot se obtiene gracias al tópico /*Odom* del bloque de éste. A esta medida artificial se le suma una componente aleatoria para considerar el error blanco sufrido por un sensor real (ecuación 3.1).

$$d = \sqrt{(p_x - x_r)^2 + \left(p_y - y_r\right)^2} + w_r \tag{3.3}$$

Vuelta al tipo de mensajes tratado, la salida de este nodo es, obligatoriamente, del mismo formato de la entrada del anterior nodo, puesto que están comunicados. La lectura de la posición del robot sí es de un tipo diferente: /*nav_msgs::Odometry*, que permite ver tanto la posición como la velocidad, además de las matrices de covarianza que informan del error en las medidas. Es una de esas estructuras modificadas para facilitar ciertas tareas, y engloba a su vez muchos tipos más. Por ejemplo, puede verse que contiene este tipo si se observa la Figura 3-6.

nav_msgs/Odometry.msg

```
Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

Figura 3-6. Mensaje *Odometry* en ROS del paquete *nav_msgs*

Si se sigue indagando en la estructura interna, se ve que para llegar a la posición absoluta del robot en coordenadas cartesianas hay que pasar por las estructuras del paquete *geometry_msgs*: *PoseWithCovariance* > *Pose* > *Point*, donde, finalmente, se ubican *x* e *y*.

- **Bloque Pioneer 3-AT:**

Todo este conjunto forma parte de una demo externa al trabajo, motivo por el cual se ha decidido darle menos importancia a su contenido interno, más allá del uso con Gazebo, lo imprescindible para hacer que el robot se mueva y que transmita su posición para el nodo previo.

De todas formas, explicando por encima, la mayoría de los bloques de la Figura 3-7 están ahí para la correcta comunicación entre ROS y Gazebo, de forma que cumplen diversas funciones como pasar datos de posición al nodo, transformar entre los diferentes marcos de referencia del modelo, recibir las medidas del sensor láser, mandar órdenes de movimiento a la representación en Gazebo…

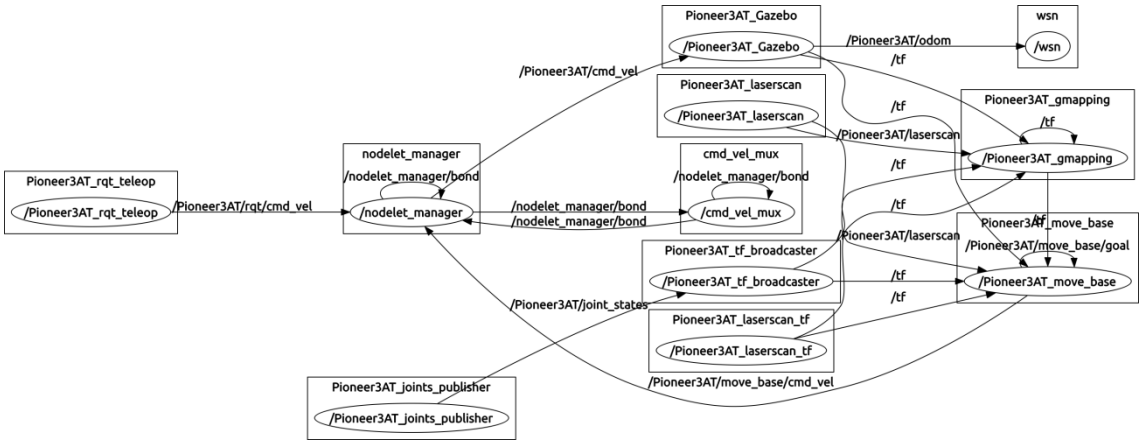


Figura 3-7. Bloque Pioneer, diagrama con herramienta rqt

Lo que más concierne de este modelo al presente trabajo son dos tópicos concretos:

- /*Pioneer3AT*/*odom*: Del cual ya se ha hablado bastante anteriormente.

- /*Pioneer3AT*/*rqt*/*cmd_vel*: Este tópico es por donde el modelo recibe órdenes de movimiento con mensajes tipo *geometry_msgs::Twist*.

Para enviar estas órdenes de movimiento, con la demo viene implementado una especie de simple panel de control (Figura 3-8) realizado con la herramienta *rqt* que permite al usuario interactuar en cualquier momento. Aparte de este método, se ha implementado un nodo de teleoperación mediante teclado que escribe en el mismo tópico y el mismo tipo de mensaje mencionados.

- **Teleoperación por teclado:**

Con objeto de enviarle al modelo de robot las órdenes de movimiento, las cuales son, principalmente, dos: girar y moverse hacia delante o hacia atrás; se implementa un nodo cuya principal función es leer valores del teclado y en función de dicha señal enviar un mensaje con un tipo de movimiento u otro.

Este nodo publica en /*Pioneer3AT*/*rqt*/*cmd_vel* los mensajes mencionados antes, tipo *Twist*. Además en su funcionalidad se contempla la posibilidad de convocar varios puntos desde donde teleoperar debido a que contempla la creación de hilos.

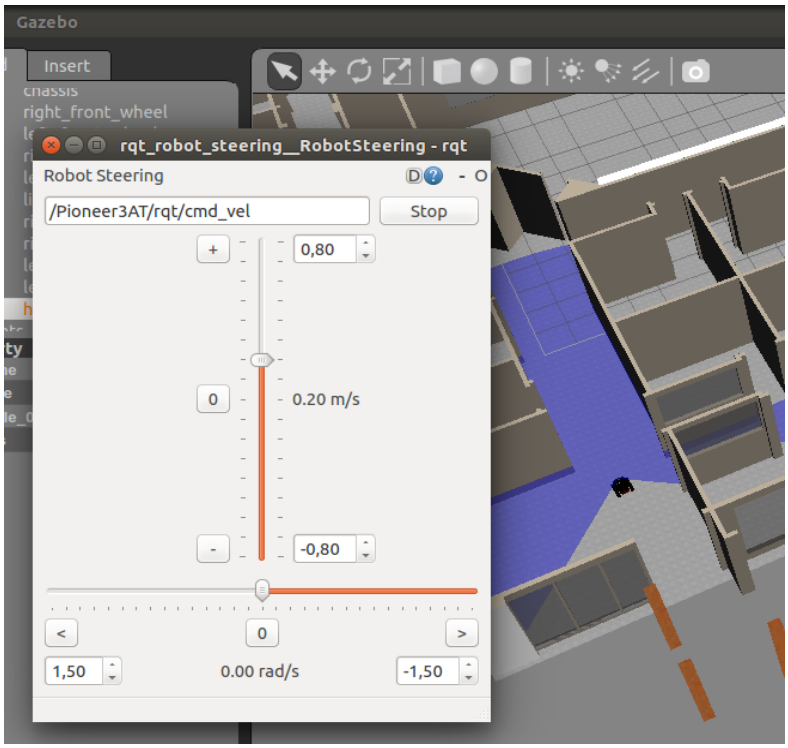


Figura 3-8. Panel de comandos, hecho de serie con herramienta rqt

- **KF/IF y GPS:**

Los filtros anteriormente implementados son las versiones extendidas del KF y el IF dado que las únicas medidas contempladas hasta el momento eran provenientes del sensor de rango. Para poder mostrar en funcionamiento sus versiones simples, se implementa un par de nodos más con estas versiones, además de otro nodo semejante al de WSN, pero para medidas tipo GPS.

3.6 Conclusiones

Concluyendo este capítulo, queda resaltar el potencial que tiene ROS como herramienta.

Se ha mostrado la importancia y el funcionamiento que poseen sus principales elementos: los nodos son los principales puntos de computación donde, para la comunicación, se envían y reciben mensajes a través de tópicos, que tienen la función de ser el canal que los conecta.

Cabe destacar la amplia gama de formatos de mensaje existente, lo cual facilita todo tipo de tarea que aparece presente en la puesta en marcha de cada robot típico. Por el contrario, se ha de depender de una librería externa para el trabajo con matrices, aunque sin mucha dificultad añadida.

Interesa comentar también la sencillez a la hora de mejorar el algoritmo del filtro, una vez preparado el esqueleto de forma funcional, pudiendo colocarse un sinnúmero de sensores mientras la capacidad computacional lo permita. Aparte de esto, el código es susceptible de disminuir errores en gran medida con un correcto estudio del modelo a usar, además de considerar cambios en este según la situación en la que se pueda encontrar el robot (en movimiento, parado, girando, etc).

4 SIMULACIONES

La verdad es lo que es, y sigue siendo verdad
aunque se piense al revés.

- Antonio Machado -

4.1 Introducción

Aquí se recogen los resultados de todas las simulaciones, frutos de la ejecución de los distintos algoritmos, tanto en Matlab como la plataforma ROS/Gazebo.

En Matlab, por norma general, se especifica una trayectoria antes de comenzar, además en todo caso se definen de antemano el número y posiciones de la red de nodos.

En cambio, en la simulación con ROS, el desplazamiento del robot se hace en directo por el usuario, bien con la herramienta de serie proporcionada, bien con el nodo de teleoperación por teclado.

El objetivo fundamental de este capítulo es demostrar virtualmente la funcionalidad y robustez que poseen tanto el filtro de Kalman como sus demás variantes, dejando claro así que son el mejor recurso a la hora de implementar aplicaciones de seguimiento.

4.2 Estructura de las pruebas

Es de utilidad dar, en primer lugar, una visión global del orden que se llevará a cabo.

Se empezará por exponer algunas simulaciones del KF en Matlab, dada la facilidad que ofrece esta plataforma para hacer variaciones en las pruebas y visualizar cómodamente los resultados.

Estas primeras simulaciones emplean únicamente el uso de GPS, tras esto se procede a la incorporación de los sensores de rango, lo que implica el uso del EKF. Tras lo cual se intercalará alguna muestra de ROS, antes de llegar a los filtros de Información, dejando para el final el filtro de Partículas.

4.2.1 Subíndice

- **Filtro de Kalman**
- **Filtro de Información**
- **Filtro de Kalman Extendido**
- **Filtro de Información Extendido**
- **Filtro de Partículas**

4.3 Parámetros comunes a la hora de simular

Tanto en una parte del trabajo como en la otra, hay ciertos parámetros comunes que son necesarios conocer para reproducir las pruebas bajo las mismas circunstancias.

- ***Media/Covarianza:*** Valores iniciales que tendrán los parámetros del filtro.

$$\mu_t = \begin{pmatrix} p_x \\ p_y \\ v_x \\ v_y \end{pmatrix} = \begin{pmatrix} 10 \\ 10 \\ 0 \\ 0 \end{pmatrix} \qquad \Sigma = \begin{bmatrix} 100 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 \\ 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 20 \end{bmatrix} \tag{4.1}$$

En la trayectoria cuadrada, los valores iniciales de la posición serán $(5,5)$ mientras que, en ROS, la media inicial es nula, pese a que el robot aparece, por defecto, en la posición $(-9, -22)$.
Para el filtro de Información, se inicializa la matriz de información en lugar de la de covarianza mientras que se mantiene el uso de la misma media de la siguiente forma:

$$\Omega = \Sigma^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot 10^{-4} \qquad \xi = \Omega\, \mu \tag{4.2}$$

- ***Periodo:*** Cada cuánto se actualizaría el algoritmo. Interviene en todas las matrices del algoritmo, excepto en la asociada al ruido del sensor.

$$T = 0{,}1\ s \tag{4.3}$$

- ***Desviación típica:*** Tanto para el modelo de predicción como para las medidas tomadas por los sensores. Error blanco por el que son afectados. Tal como están en la tabla 4.1, de arriba abajo: Error del modelo, en las medidas GPS y en las medidas de rango.

Tabla 4–1. Ruido blanco gaussiano

$\boldsymbol{v}$	$0.1\ m$
$\boldsymbol{w_g}$	$3\ m$
$\boldsymbol{w_r}$	$0.5\ m$

- ***Localización de WSN:*** Otro dato que es fundamental saber es la posición cartesiana de los nodos, puesto que es información que se necesita para calcular el Jacobiano. En general, se usarán 3 nodos, indicándose lo contrario, en el apartado correspondiente de ser necesario.

Tabla 4–2. Nodos WSN

Posición / Nodo	x (m)	y (m)
$\mathbf{N_1}$	0	0
$\mathbf{N_2}$	0	20
$\mathbf{N_3}$	23	5

- ***Alcance:*** Los sensores de rango además, vienen influenciados por otro parámetro que implica la máxima distancia a partir de la cual dejan de emitir información correctamente.

$$L = 30\ m \tag{4.4}$$

4.4 Simulaciones con el Filtro de Kalman

Volviendo a capítulos anteriores, ha de recordarse que en su versión más simple, el filtro de Kalman sólo trabaja con ecuaciones lineares (solución unimodal). Es por esto, que en este capítulo únicamente se contempla el uso de señal GPS.

4.4.1 KF 1: Zigzag, Un GPS

4.4.1.1 GPS buena calidad

En la Figura 4-1 se observa el resultado de la primera solución. En este caso, como se indica al comienzo de este apartado, se emplea como único sensor el GPS, puesto que su implementación sólo requiere ecuaciones lineales.

La primera trayectoria ha sido una línea recta, con dos quiebros a mitad del camino, para comprobar la eficacia del filtro en su seguimiento. Se observa que este seguimiento no es instantáneo porque la estimación del filtro se basa en dos elementos: la predicción del estado y la corrección mediante medidas. Si no se tuviera en cuenta la señal de GPS, el filtro seguiría en línea recta sin considerar ningún cambio, porque su modelo evoluciona linealmente, razón por la que le es tan difícil seguir ese cambio de pendiente.

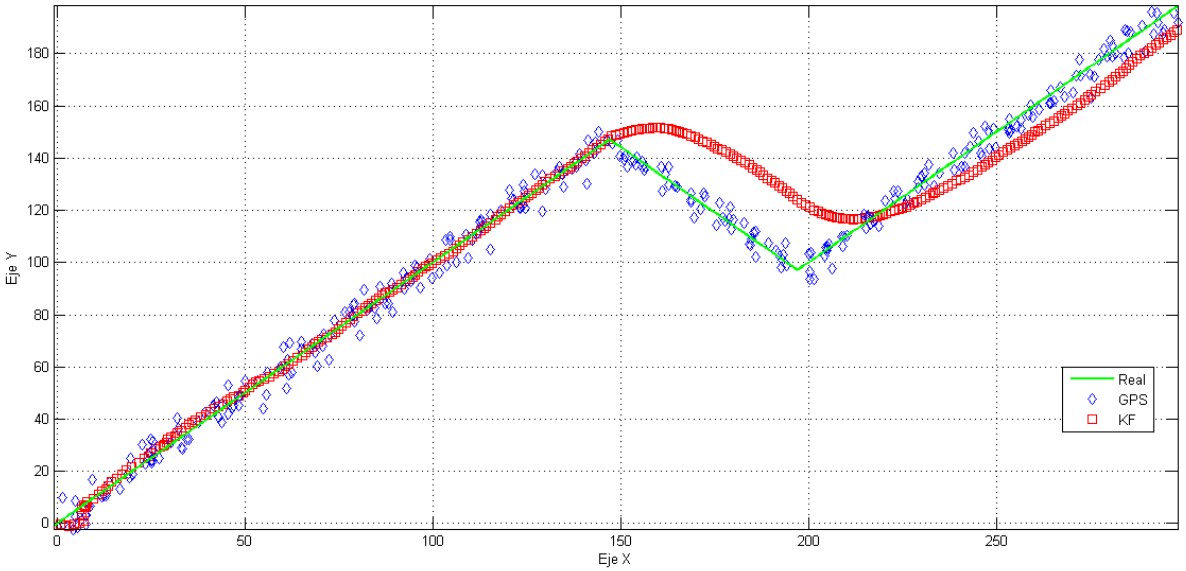


Figura 4-1. Simulación trayectoria lineal con zigzag a mitad de camino

4.4.1.2 GPS mala calidad

En la segunda simulación (Figura 4-2), se empleará un GPS mucho menos fiable, incrementando 5 veces su desviación típica.

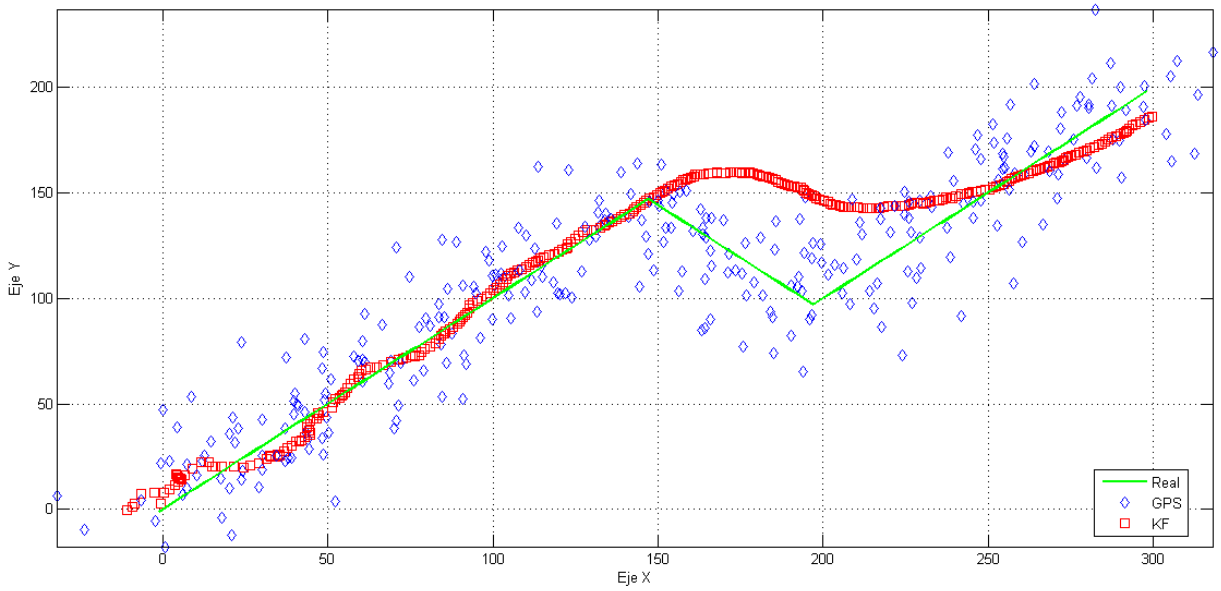


Figura 4-2. Misma simulación con $w_g = 15\ m$

Ahora, se procede a comparar el valor que toma el determinante de la covarianza en la Figura 4-3, que puede servir como indicación de la incertidumbre total que padece el sistema.

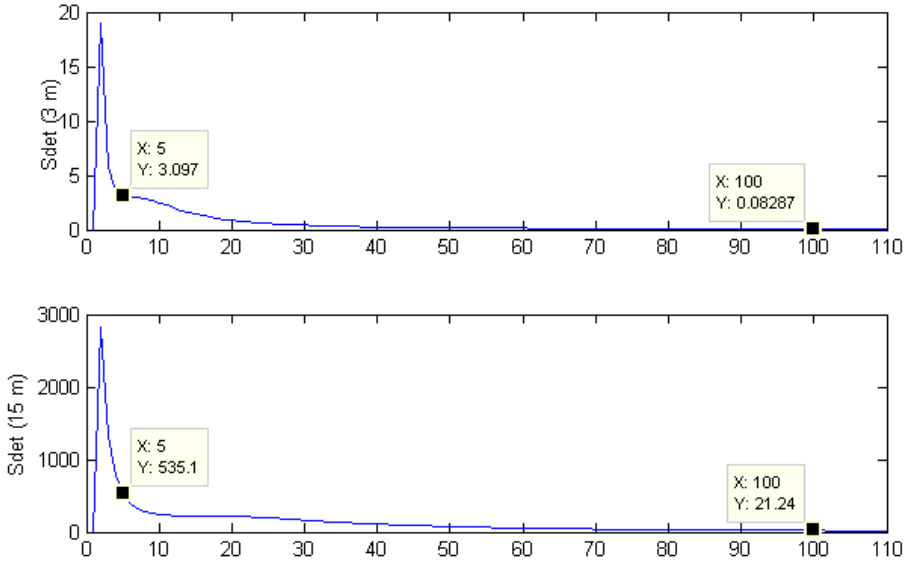


Figura 4-3. Medida de la incertidumbre durante las pruebas anteriores

Hay una gran diferencia entre el error percibido cuando se usa un GPS de 3 m respecto a uno de 15, el error se multiplica unas 100-150 veces. Este gran incremento se debe a que el error se manifiesta de forma cuadrática.

4.4.2 KF 2: Zigzag, Dos GPS

4.4.2.1 GPS buena calidad

Repetición de la simulación anterior con la incorporación de otro GPS adicional, visible en la Figura 4-4.

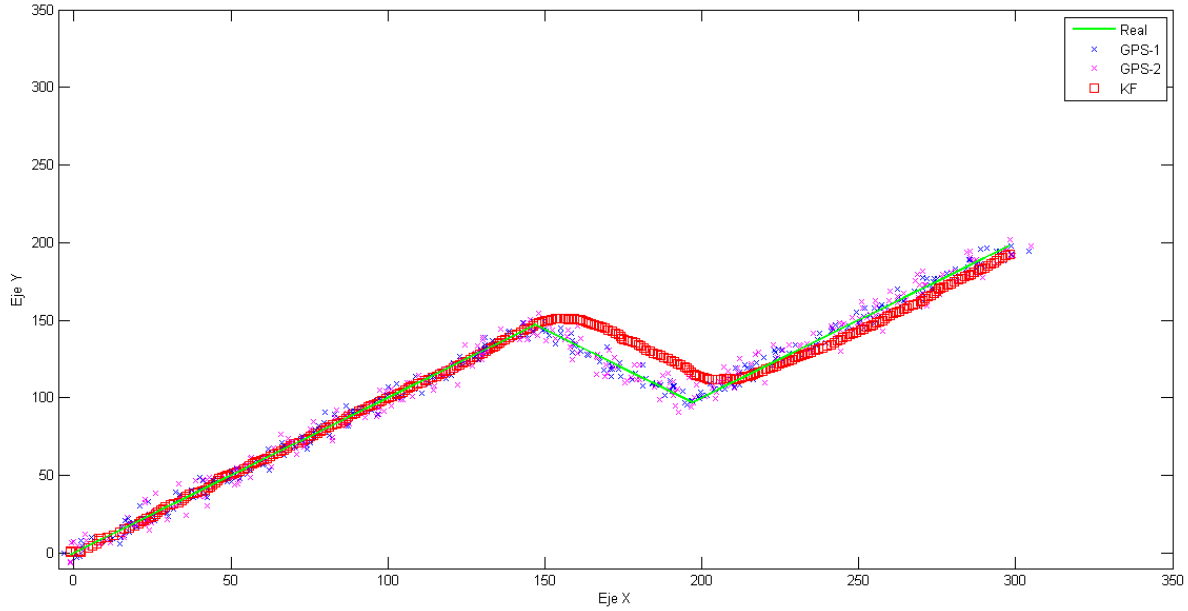


Figura 4-4. Uso de dos GPS, misma dispersión, $w_g = 3\ m$

Gracias al aumento de la información recibida por el filtro, se logra un seguimiento más eficaz, como se observa en la cercanía de la estimación a la trayectoria real en el momento del giro, que está más cerca que en cualquier prueba anterior, logrando seguir con mayor certeza el tramo final por completo. Además, mirando con detalle las Figuras 4-5 y 4-3, el error ha quedado mermado a la cuarta parte tras el nuevo GPS (de 20 a 5, aproximadamente).

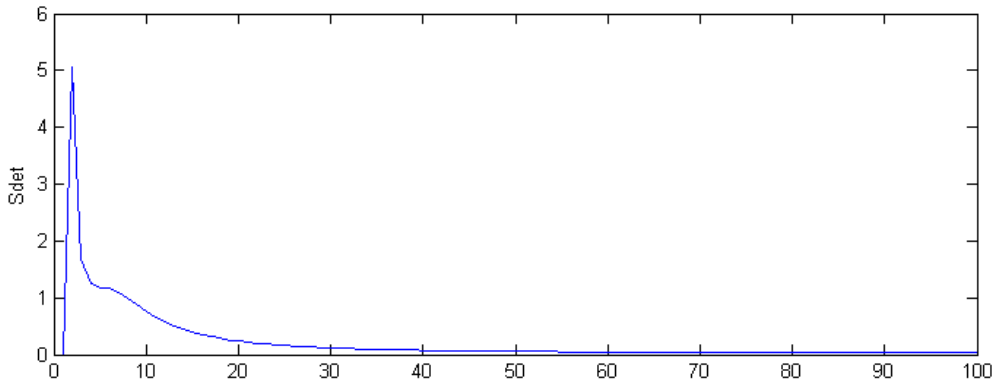


Figura 4-5. Incertidumbre en la primera prueba con dos GPS

Se vuelve a realizar la prueba sustituyendo ambos GPS por un modelo más mediocre, y, tras esto, otro combinándolos.

4.4.2.2 GPS mala calidad

En este caso, se estudia la opción con ambos GPS de mala calidad en la Figura 4-6.

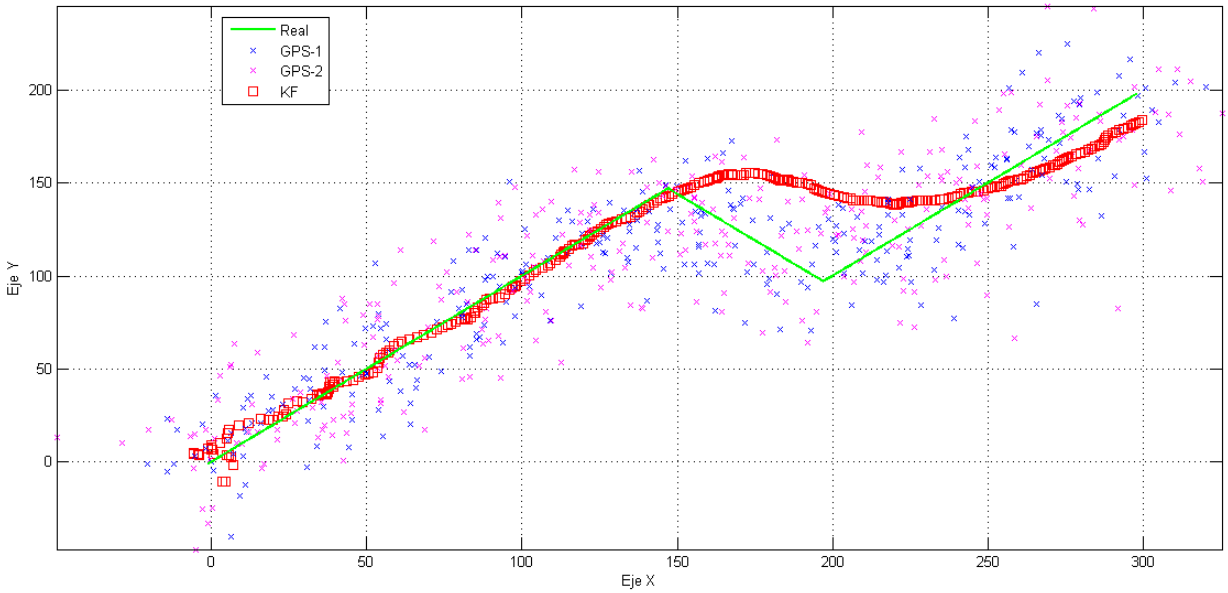


Figura 4-6. Uso de dos GPS, misma dispersión, $w_g = 15\ m$

Como era de esperar, la calidad del seguimiento ha empeorado considerablemente. El uso de 2 GPS no contrarresta lo bastante los estragos causados por la fuerte desviación de estos. Aun así, volviendo a comparar la Figura 4-3 con la Figura 4-7, se observa una considerable mejora respecto la actuación con un único GPS.

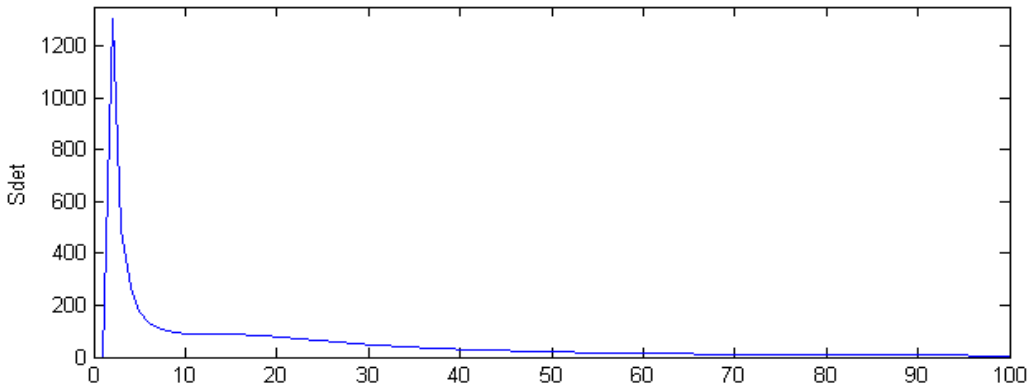


Figura 4-7. Error de dos GPS con gran desviación

4.4.2.3 GPS buena/mala calidad

En la Figura 4-8 se muestra un caso promedio en la calidad.

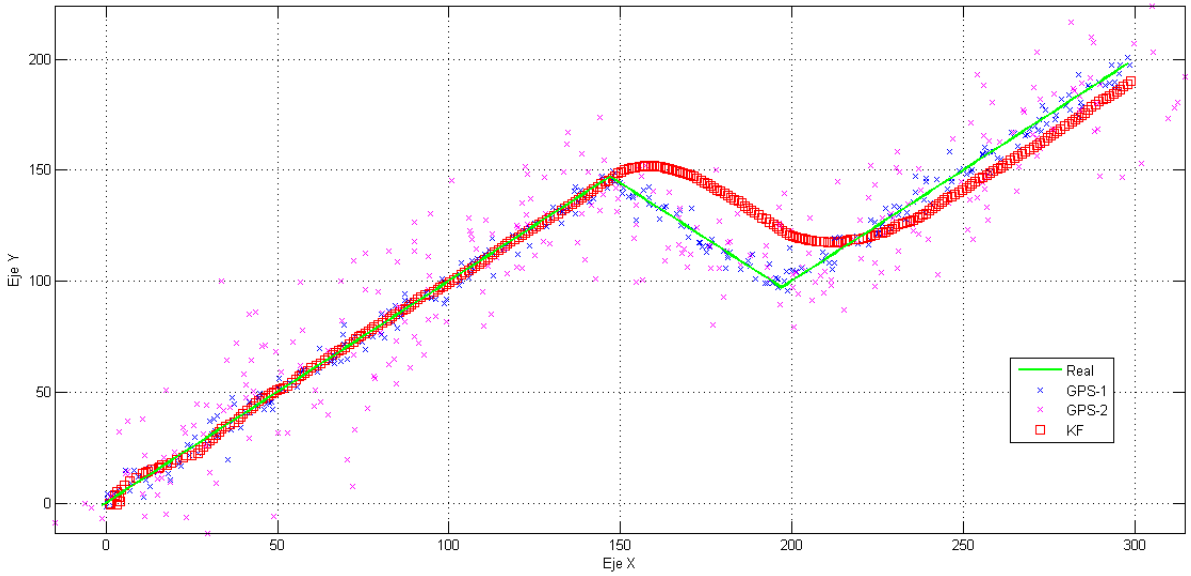


Figura 4-8. Uso de 2 GPS, dispersiones: $w_{g1} = 3\ m, w_{g2} = 15\ m$

Es interesante recordar un poco el funcionamiento del filtro. Por una parte, predice mediante el uso de un modelo cinemático, por otra, se auto-corrige mediante la incorporación de medidas. Lo que ocurre es, que a la hora de incorporar todo lo anterior, se basa en sus modelos de error para estimar con qué seguridad puede confiar en lo que recibe. De esta manera, teniendo un GPS bueno y uno malo, siendo el filtro consciente de esta situación, cuando actualiza su estado, lo hace basándose principalmente en el cálculo del GPS de calidad. Esta comprobación de errores en el momento de computar es lo que imposibilita el instantáneo seguimiento del giro brusco, porque para el filtro, el modelo sigue teniendo una alta validez dado que su covarianza no ha empeorado.

Se puede corroborar esto anterior escrito, con la comparación de las Figuras 4-2 y 4-9. No sólo, se ha mantenido la calidad de la primera simulación, sino que ha logrado ser mejorado mínimamente (al menos el pico de incertidumbre inicial), por el hecho de tener otro GPS, pese su peor calidad.

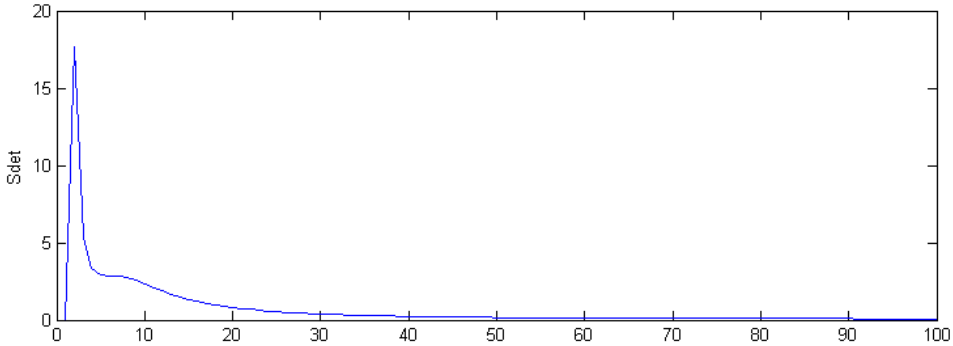


Figura 4-9. Error usando dos GPS de distinta calidad

4.4.3 KF 3: Zigzag, Dos GPS, Modelo menos fiable

Esta última reflexión lleva a comprobar el funcionamiento del filtro bajo un modelo menos fiable. Se procede a aumentar el error del modelo a $v = 0{,}5\ m$.

4.4.3.1 GPS buena/mala calidad

Los resultados son visibles en la Figura 4-10.

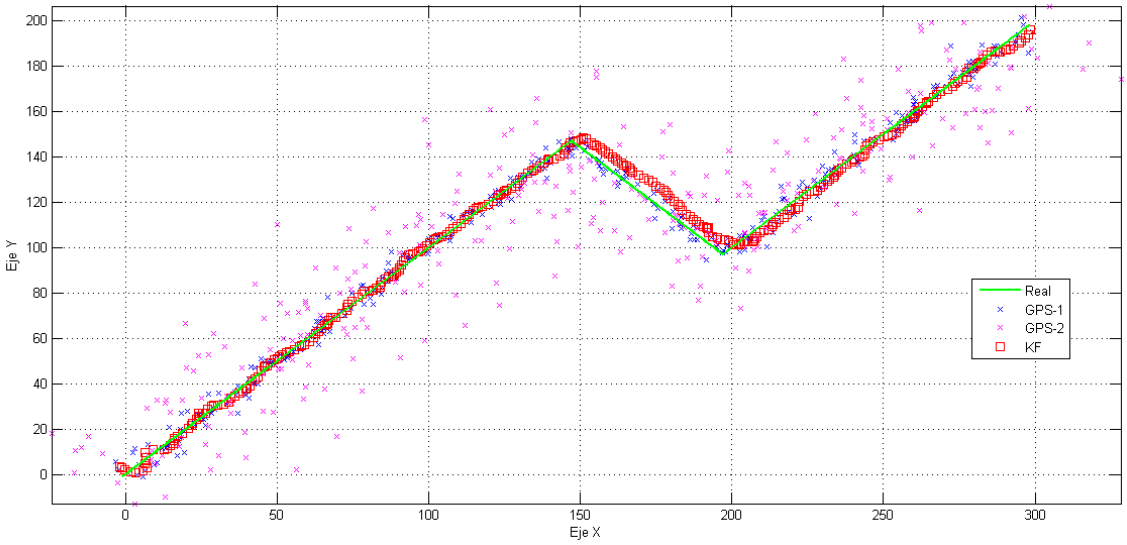


Figura 4-10. Dos GPS con $w_g = 3\ m$, error del modelo: $v = 0{,}5\ m$

Efectivamente, al indicarle al filtro que el modelo seguido no era tan fiable como en un principio, deja de hacer los cálculos con tanta "fe" en él. Esto permite darle más importancia a la información externa, por lo que logra seguir cambios bruscos, no contemplados por el modelo, con pasmosa rapidez respecto el caso anterior.

4.4.3.2 GPS mala calidad

Uso de ambos GPS con calidad mediocre. La cara oscura, de tener más confianza en los sensores que en el modelo, es que el sistema se queda a merced de la fiabilidad de estos. Por lo que, como muestra la Figura 4-11, el vector de estado sufre una tasa de cambios esporádica mayor, cuanto más ruido interfiera con los sensores.

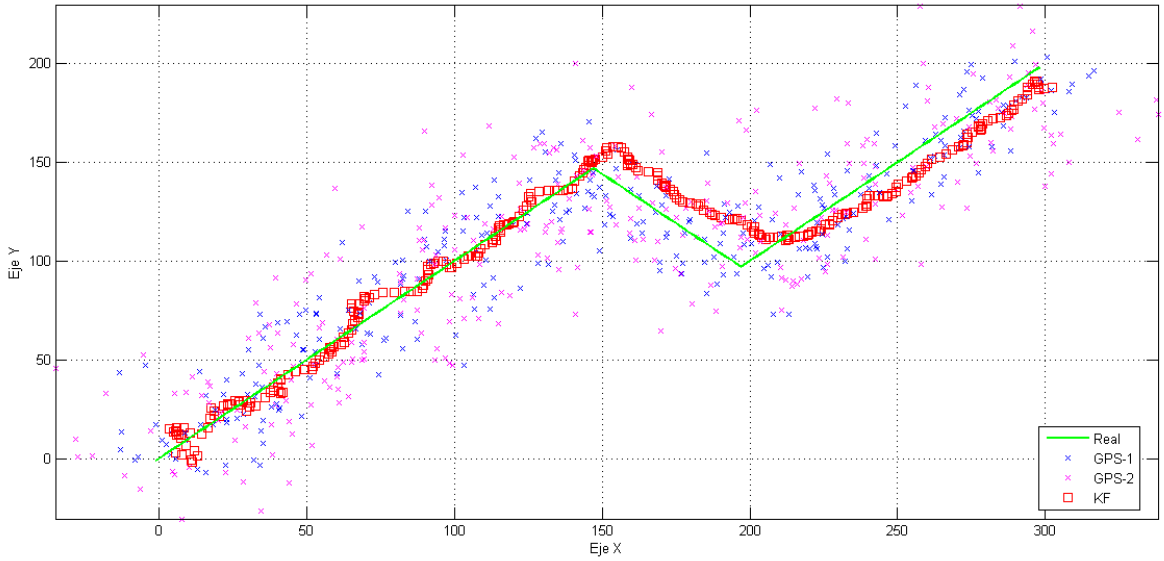


Figura 4-11. Dos GPS con $w_g = 15\ m$, error del modelo: $v = 0{,}5\ m$

Además, el filtro ya no realiza tan buen seguimiento en el giro como antes, dado que al perder calidad tanto el modelo como los GPS, no termina de dar prioridad a un modelo sobre otro.

Una posible solución a esto sería llegar a una solución de compromiso entre la calidad de ambos parámetros. También cabría la posibilidad de un cambio del modelo "en línea", es decir, un cambio dinámico de la matriz que lo representa, o lo que sería aún más fácil, de su error, en momentos de cambio brusco. De esta forma se tendría un parámetro de control con el que ajustar la precisión del filtro.

4.4.3.3 GPS media/mala calidad

Con esta solución de compromiso se llega a una calidad aceptable de forma general. El seguimiento mostrado en la Figura 4-12 reacciona aceptablemente rápido y evita un recorrido errático.

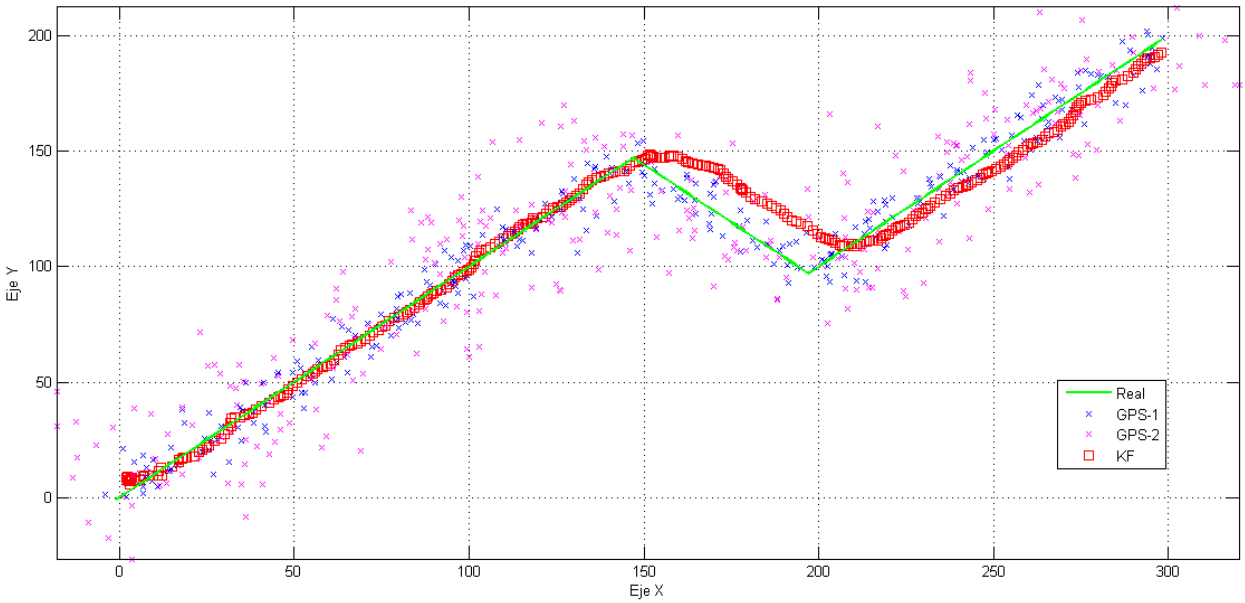


Figura 4-12. Dos GPS con $w_{g1} = 6\ m$ y $w_{g2} = 15\ m$, modelo con: $v = 0{,}3\ m$

4.4.4 KF 4: Cuadrado, Un GPS

4.4.4.1 GPS buena calidad

En una primera ocasión, como se muestra en la Figura 4-13, se realiza con un modelo de bajo error, como se establece en el primer apartado del presente capítulo.

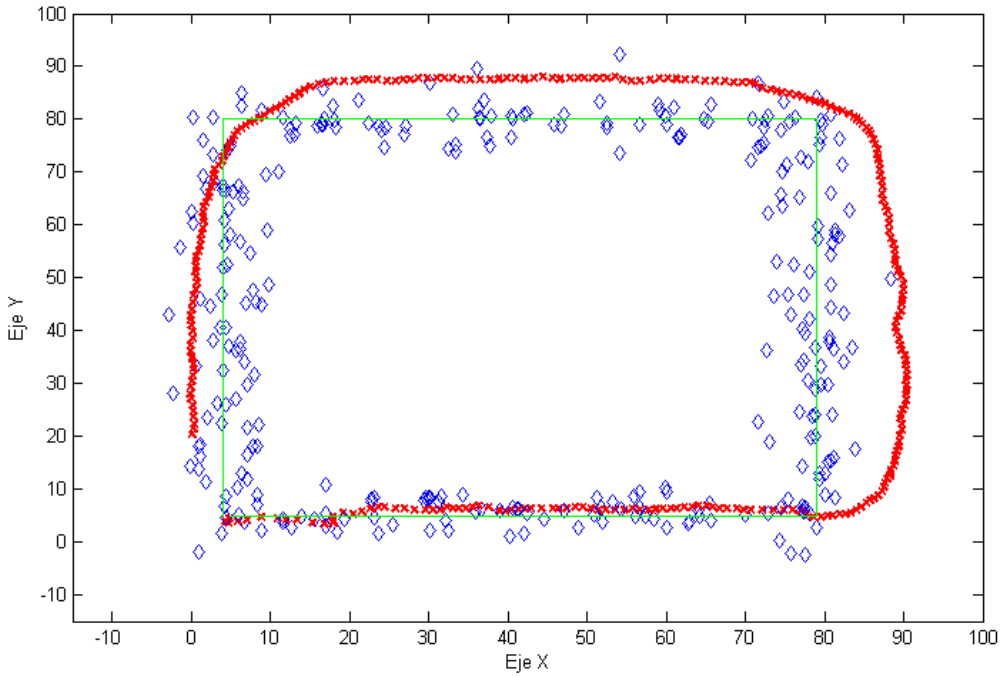


Figura 4-13. GPS con $w_g = 3\ m$, error del modelo: $v = 0{,}1\ m$

En este caso, debido a los giros bruscos, el filtro sigue bastante mal la trayectoria. Como se observa en la Figura 4-14, a partir de la primera esquina, el filtro empieza a retrasarse considerablemente. La Figura 4-14 muestra el error absoluto entre la posición estimada y la real durante la simulación actual.

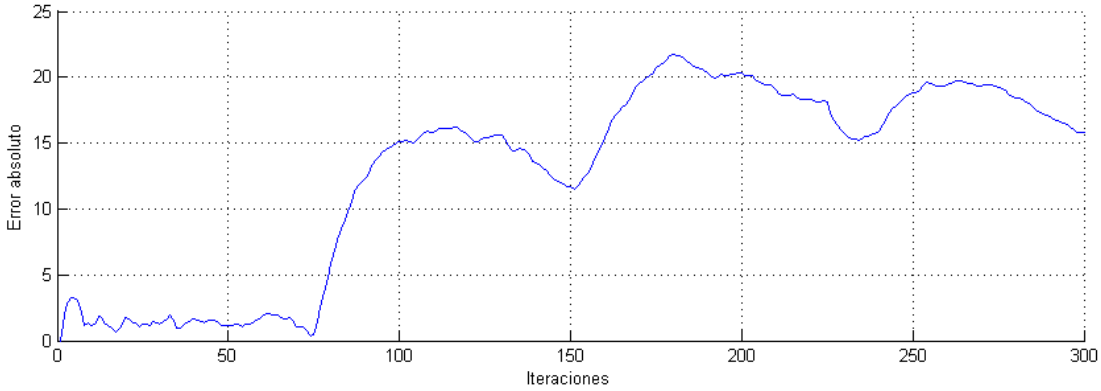


Figura 4-14. Error en posición, trayectoria cuadrada, primera prueba

4.4.4.2 GPS buena calidad, modelo menos confiable

En la segunda simulación se busca reducir el error mostrado en la Figura 4-14. Para ello, se ha reducido la confianza en el modelo, quedando el filtro más susceptible ante la llegada de información externa. Se reduce drásticamente el error, pero a cambio de una posición más inestable como muestra la Figura 4-15.

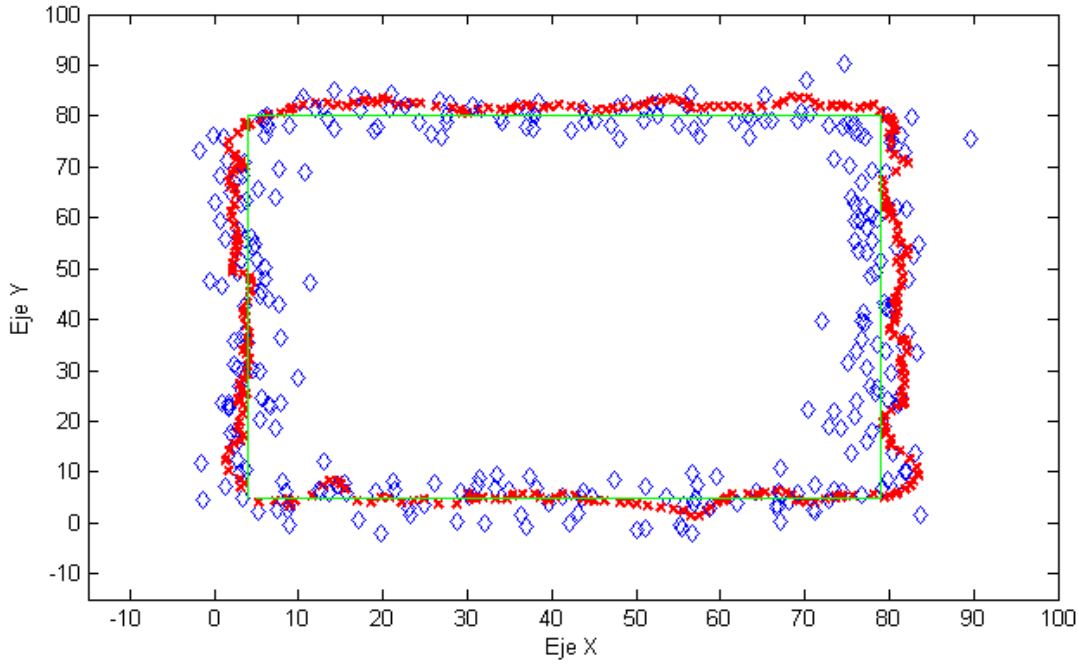


Figura 4-15. GPS con $w_g = 3\ m$, error del modelo: $v = 0{,}5\ m$

El error disminuye drásticamente al reducir la influencia del MRU como se ve la Figura 4-16.

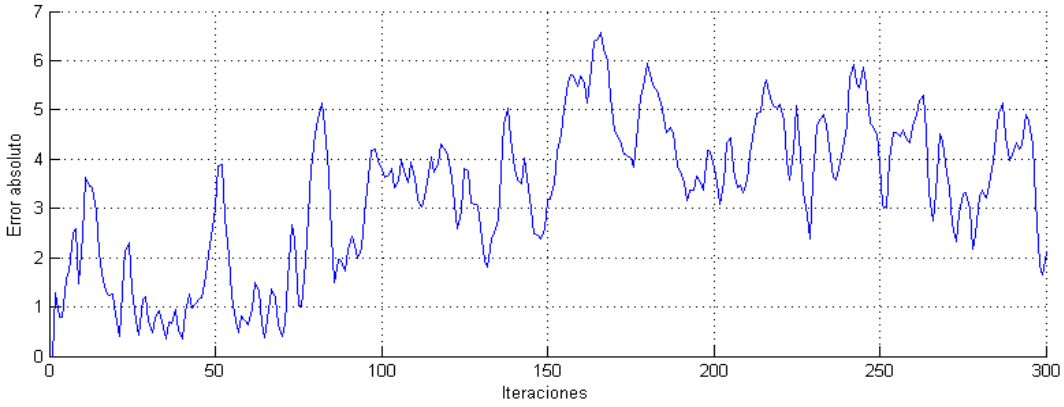


Figura 4-16. Error en posición, trayectoria cuadrada, segunda prueba

4.4.5 KF 4: Túnel, Modelo estático, Un GPS

En este apartado se implementarán distintas trayectorias con la característica de que en tramos concretos se perderá la comunicación con los sensores. Con esto, se pretende resaltar la robustez del filtro de Kalman ante este tipo de eventos, todo gracias a la etapa de predicción. Además, también se mostrará algún ejemplo de mejora en el algoritmo con la implementación de otro modelo cinemático: el modelo estático.

$$A_{est} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.5}$$

Este modelo implica que en la fase de predicción, el robot permanece en la misma posición. El seguimiento dependería por completo de la actualización mediante los sensores.

4.4.5.1 Línea, GPS buena calidad

Como se ha dicho al comienzo de este apartado y se demuestra en la Figura 4-17, el filtro continúa dando información del vector de estado gracias a la predicción. Esto es de gran interés para aplicaciones de control de extremo cuidado, donde es vital tener siempre un mínimo de seguimiento del estado, prefiriéndose, incluso, este seguimiento erróneo a no tener información ninguna.

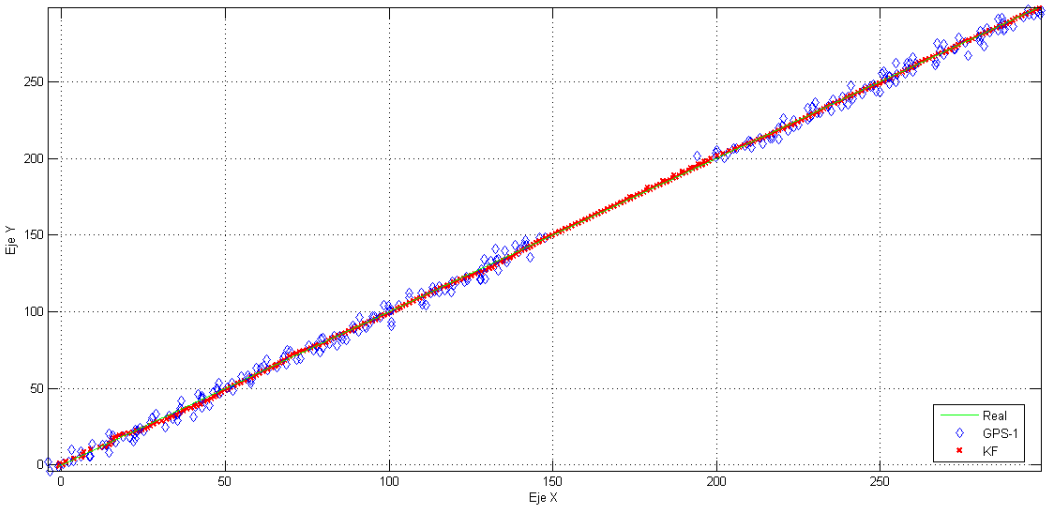


Figura 4-17. Un GPS con $w_g = 3\ m$, pérdida de señal entre la iteración 150 y la 200

Igualmente el filtro mediante su matriz de covarianza es capaz de indicar la fiabilidad del seguimiento, puesto que al ser consciente de que depende únicamente del modelo, su error se va incrementando ante la falta de corrección externa. Esto se observa en la Figura 4-18.

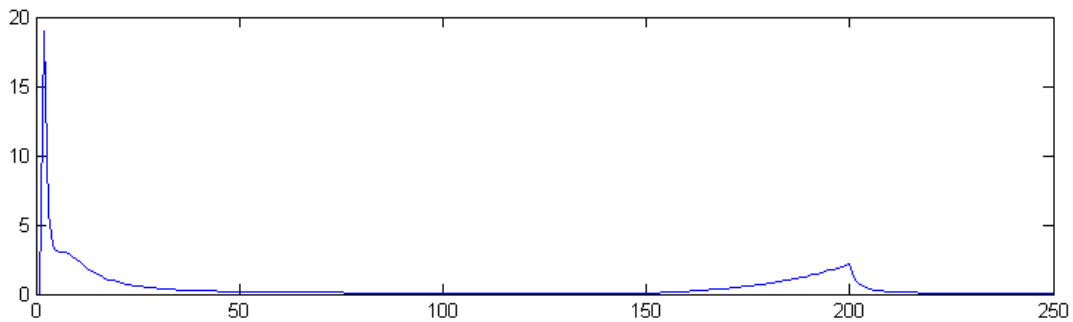


Figura 4-18. Error acumulado al perder la información externa

4.4.5.2 Lineal, GPS buena calidad

Es normal que el filtro parezca cumplir muy bien su función ante un escenario tan simple. En la trayectoria de la Figura 4-19 se pone más a prueba su robustez incorporando un cambio de pendiente.

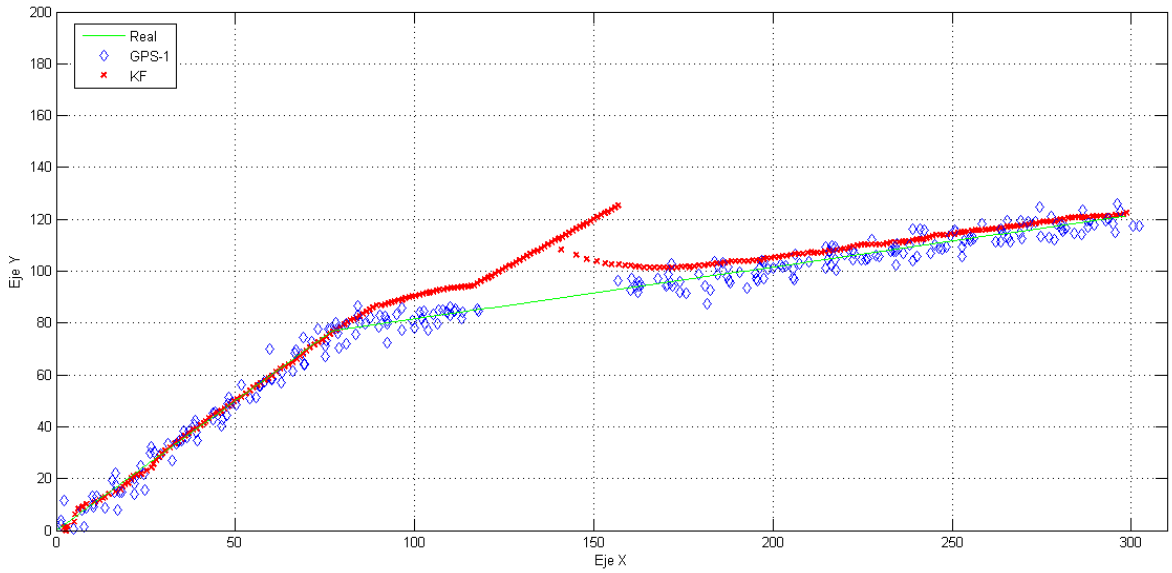


Figura 4-19. Lineal con cambio, GPS con $w_g = 3\ m$, pérdida de señal entre la iteración 120 y la 160

En esta ocasión, en el momento en el que deja de recibir señal de los sensores, el filtro deja de corregirse retomando el camino que le dictaba su modelo. Dado que sigue el modelo del MRU, esto indica que las velocidades de la media no terminan de modificarse adecuadamente para predecir bien el movimiento.

4.4.5.3 Líneal, GPS buena calidad

En las próximas simulaciones, se sustituirá las ecuaciones del modelo por su versión estática, reflexionando en qué casos puede realmente ser útil. Para una mejor visualización de cómo afecta el empleo de este modelo, se realizarán conjuntamente dos pruebas sobre la misma trayectoria como se ve en la Figura 4-20.

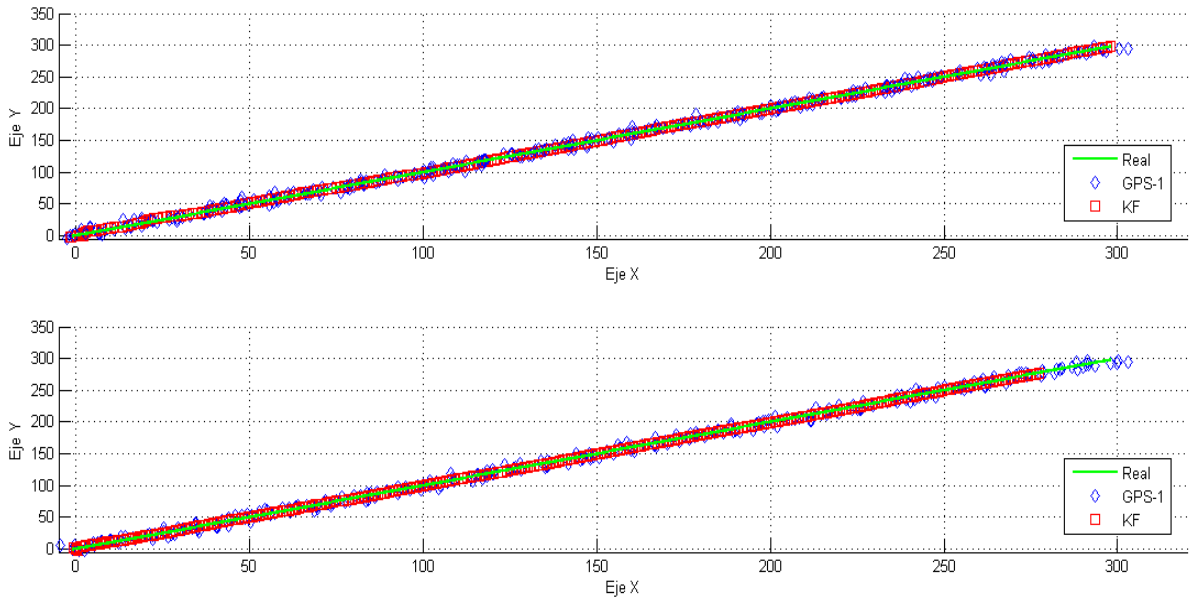


Figura 4-20. Líneal, GPS con $w_g = 3\ m$, a) Modelo rectilíneo, b) Modelo estático

Se observa que al finalizar el tramo, el filtro no llega a completarlo. Esto se debe a que, prácticamente durante todo el camino, se ha ido retrasando a causa del nuevo modelo implementado (Figura 4-21).

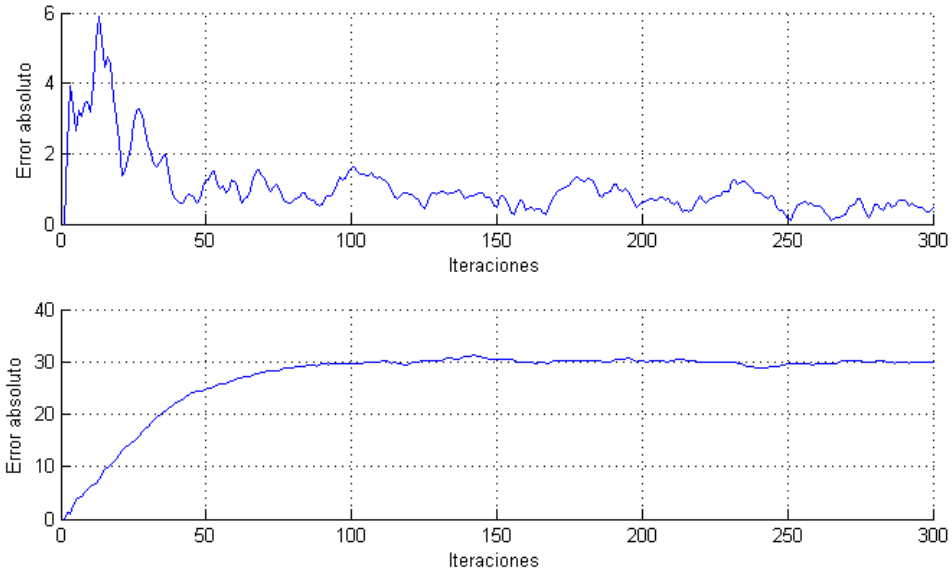


Figura 4-21. Lineal, Error en posición, a) Modelo rectilíneo, b) Modelo estático

4.4.5.4 Líneal con túnel, GPS buena calidad

Ahora se realiza la mismo simulación que la anterior mostrada, pero incluyendo un corte en el suministro de información externa entre las iteraciones 120 y 160 (Figura 4-22).

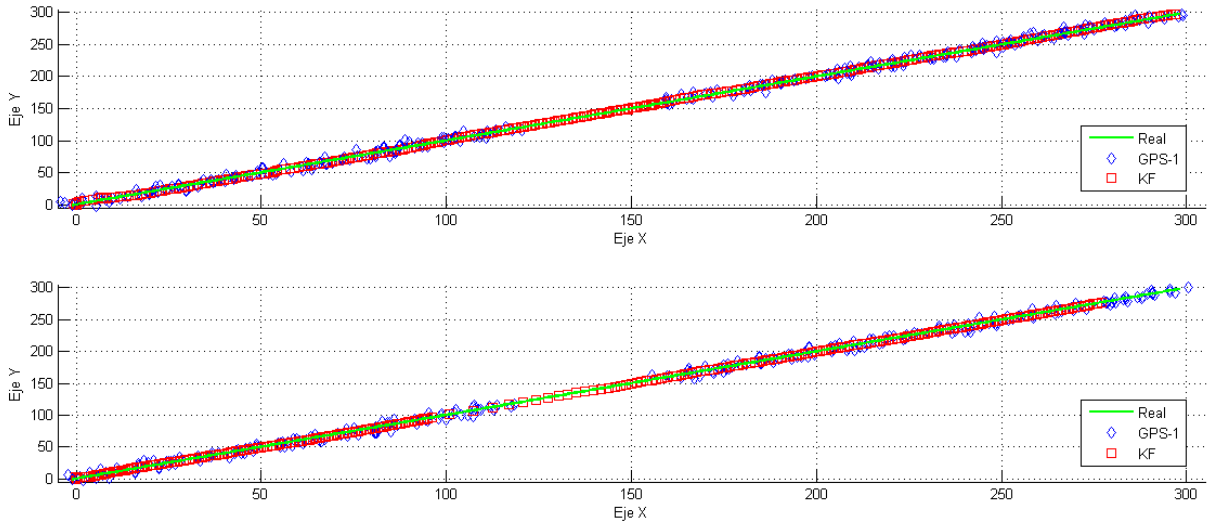


Figura 4-22. Túnel 1, GPS con $w_g = 3\ m$ error del modelo: $v = 0{,}1\ m$,
a) Modelo rectilíneo, b) Modelo estático

Es un poco difícil de apreciar, pero al momento de perderse contacto con el GPS, el filtro con el modelo estático permanece quieto, mientras que el otro prosigue gracias a su actualización. Al momento de volver a recibir medidas el estático intenta apresurarse pero acaba siendo insuficiente, dejando el tramo incompleto, al igual que sucede en el ensayo anterior.

4.4.6 KF 5: Semáforo, Dos GPS

Ahora se ponen a prueba ambos modelos ante una trayectoria con parada final como se ve en la Figura 4-23.

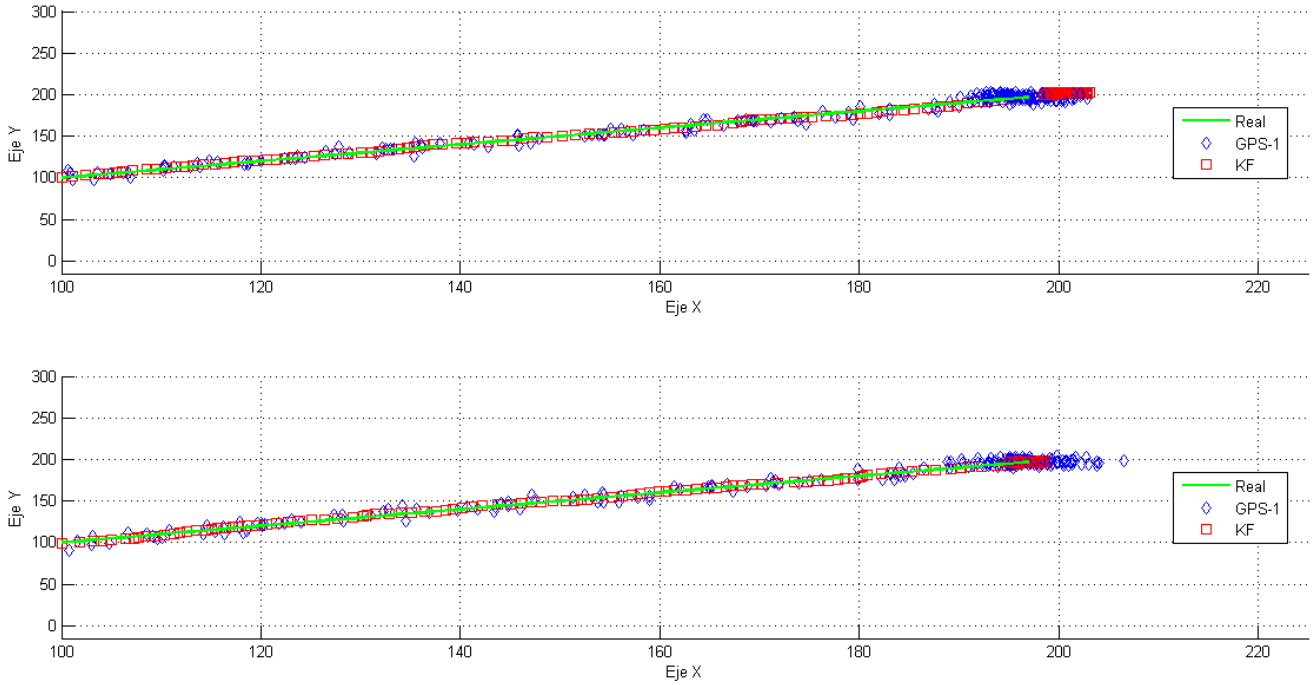


Figura 4-23. Semáforo, GPS con $w_g = 3\ m$, error del modelo: $v = 0{,}3\ m$,
a) Modelo rectilíneo, b) Modelo estático

En el recorrido actual se simula una parada a mitad del camino, permanente, en este caso. La Figura 4-24 refleja los beneficios de emplear el modelo estático, reduciéndose el error considerablemente en el momento en el que el robot permanece quieto, como era de esperar.

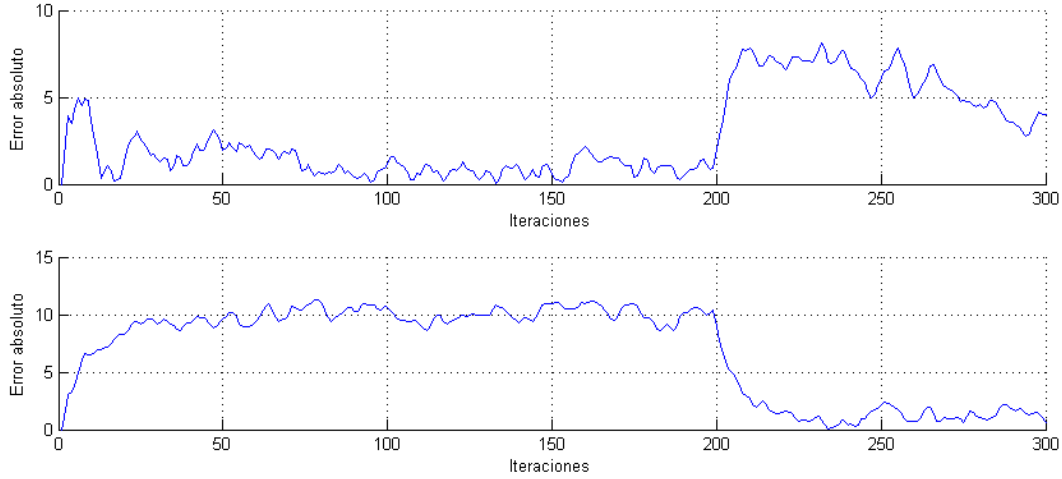


Figura 4-24. Semáforo, Error en posición, a) Modelo rectilíneo, b) Modelo estático

Un método muy conveniente de emplear el modelo estático es haciendo una sustitución sobre la marcha, en el momento que el robot detecte que su velocidad se está mermando o que la posición medida está por detrás de la que marca su predicción.

4.4.7 KF 6: Senoide, Un GPS

Otro ejemplo de una trayectoria más compleja mostrada en la Figura 4-25. En esta simulación se elige un modelo con mayor incertidumbre para facilitar un seguimiento con un cambio de dirección más continuo.

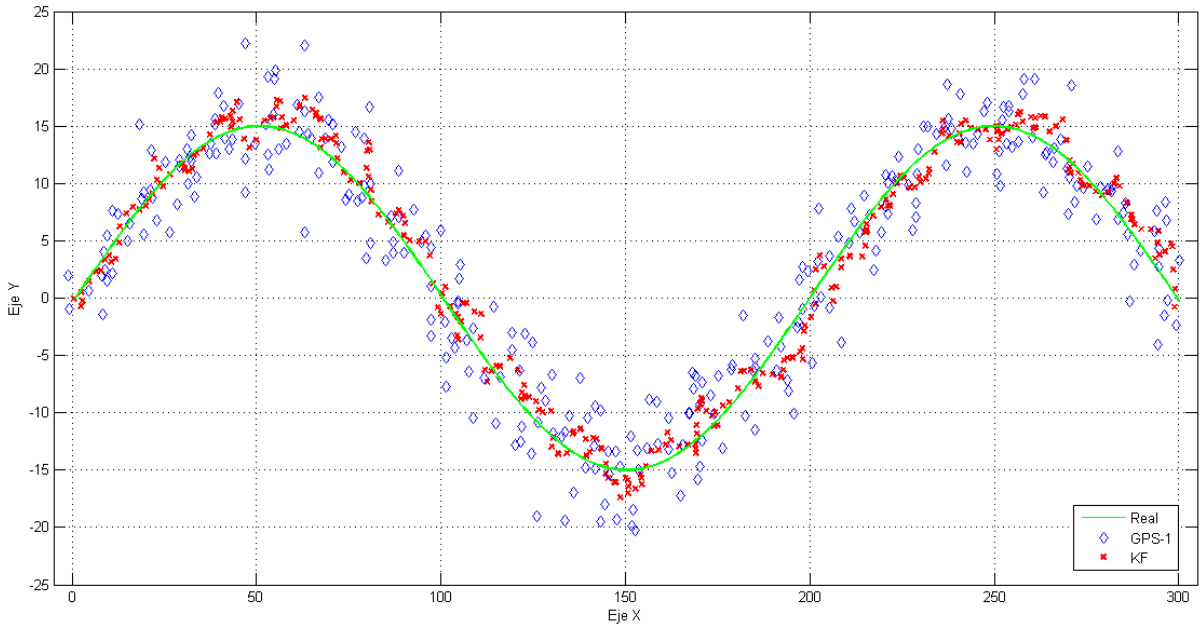


Figura 4-25. Senoide, GPS con $w_g = 3\ m$, error del modelo: $v = 1\ m$,
a) Modelo rectilíneo, b) Modelo estático

4.4.8 KF 7: ROS

Para todas las simulaciones en ROS, como se indica en el apartado 4.3, la media tiene un valor inicial nulo en todos sus componentes y la matriz de covarianza se inicializa como una matriz identidad multiplicada por el escalar 10^3. La incertidumbre del modelo se mantiene en $v = 0{,}1\ m$.

En las versiones simples de los filtros, se emplea un GPS con $w_g = 3\ m$.

En la Figura 4-26 se puede observar la simulación realizada, la cual comprende del arranque del robot mientras el filtro sigue de cerca su posición. En cierto punto se deja de recibir información del GPS (ver Figura 4-27) y el filtro depende en exclusiva de la fase de actualización con resultados bastante aceptables hasta que el robot se detiene. Dado que el filtro no es consciente de la situación externa por la falta de medidas continúa avanzando. Una vez recuperada la señal del GPS, el filtro se recupera con presteza.

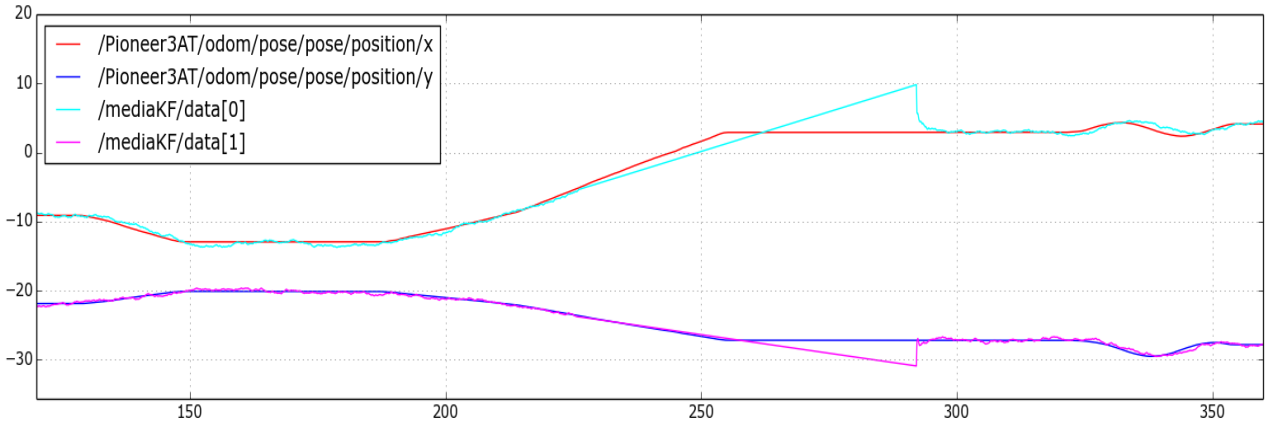


Figura 4-26. ROS, Seguimiento de la trayectoria con KF

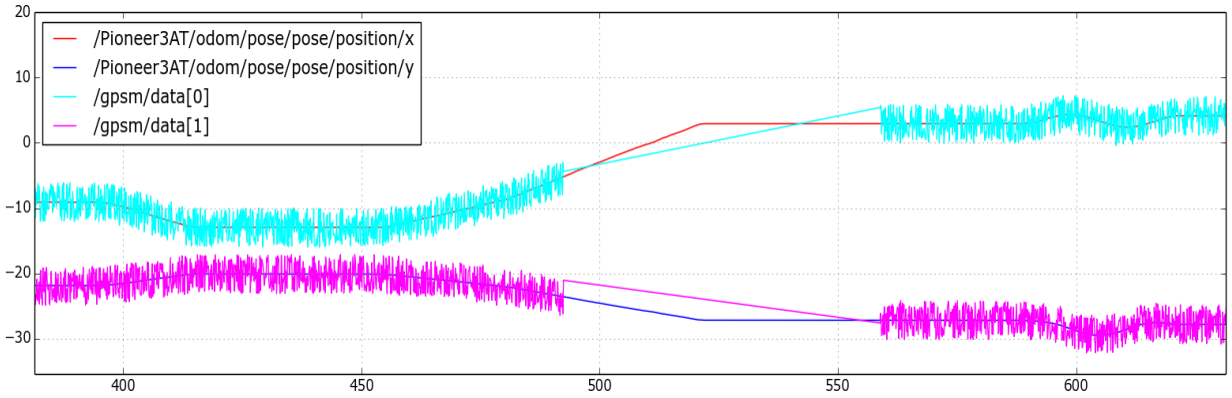


Figura 4-27. ROS, Medidas obtenidas por el GPS

Es interesante señalar que en ciertos momentos el filtro se separa un poco de la posición (ver tramo final de la Figura 4-26, líneas superiores). Esto ocurre por el continuo giro al que está siendo sometido, en ese momento, el robot. Cabe recordar los errores que tenía el modelo rectilíneo ante los cambios de dirección, aunque en este caso son mínimos puesto que el robot tiene un movimiento no holónomo y no puede cambiar muy bruscamente de dirección.

4.5 Simulaciones con el Filtro de Información

Tras la amplia gama de ensayos realizados con el filtro de Kalman, ahora es momento de ver el comportamiento de su dual: el filtro de Información. En esta apartado, se realizarán pruebas concretas del filtro anterior, poniendo mayor énfasis en compararlos que en los resultados en sí mismos. Esto es debido a que, bien programado, los resultados que arroja este filtro son idénticos, habiendo incluso que recuperar la parametrización anterior (el vector de estado) para representar resultados interpretables.

4.5.1 IF 1: Zigzag, Un GPS

4.5.1.1 GPS buena calidad

Volviendo a la primera simulación de este capítulo, es de interés repetirla con este nuevo algoritmo a la vez que se comparan resultados con el filtro de Kalman. Los resultados se reflejan en las Figuras 4-28 y 4-29.

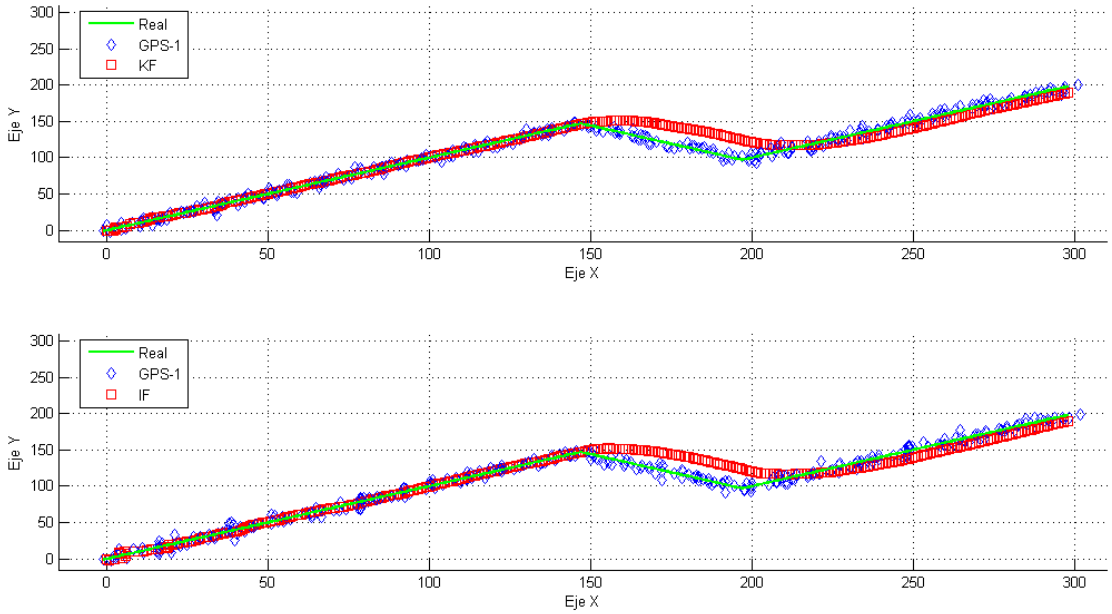


Figura 4-28. Zigzag, GPS con $w_g = 3\ m$, error del modelo: $v = 0{,}1\ m$,
a) Filtro de Kalman, b) Filtro de Información

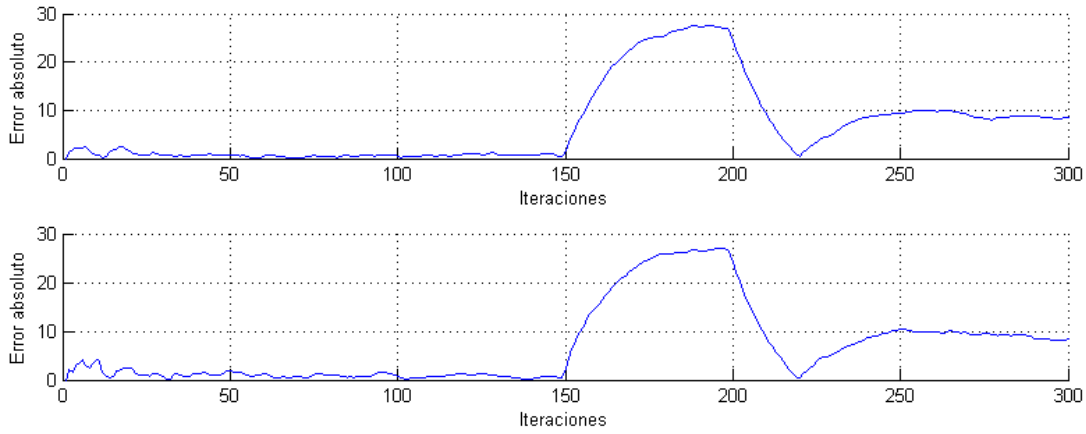


Figura 4-29. Zigzag, Error en posición, a) Modelo rectilíneo, b) Modelo estático

Efectivamente, se observa un comportamiento prácticamente idéntico y con errores prácticamente idénticos. Las nimias diferencias que se pueden encontrar son debidas a que, a pesar de que comparten trayectoria, las medidas de GPS son tomadas en algoritmos diferentes por lo que no comparten la misma aleatoriedad en la medida.

4.5.1.2 GPS buena calidad, modelo poco confiable

Anteriormente, al incrementar la incertidumbre en el modelo, el seguimiento ante giros imprevistos mejoraba notablemente, cosa que también ocurre en este filtro (Figura 4-30) disminuyendo el error en unos 20 *m*, como se observa al comparar las Figurar 4-29 y 4-31.

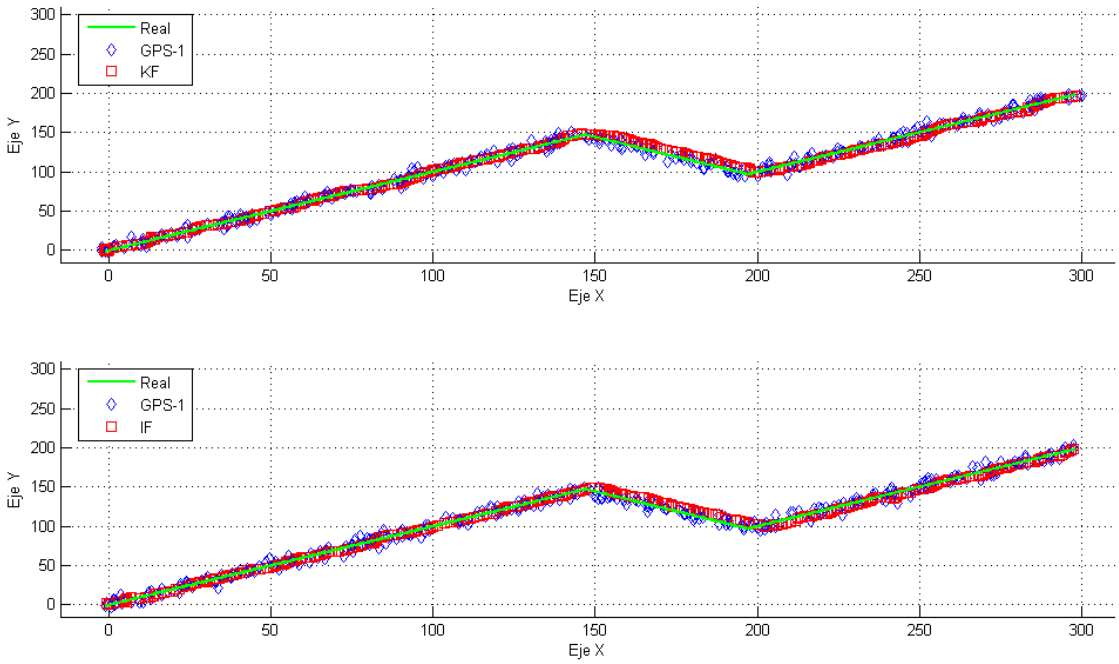


Figura 4-30. Zigzag, GPS con $w_g = 3\ m$, error del modelo: $v = 0{,}5\ m$,
a) Filtro de Kalman, b) Filtro de Información

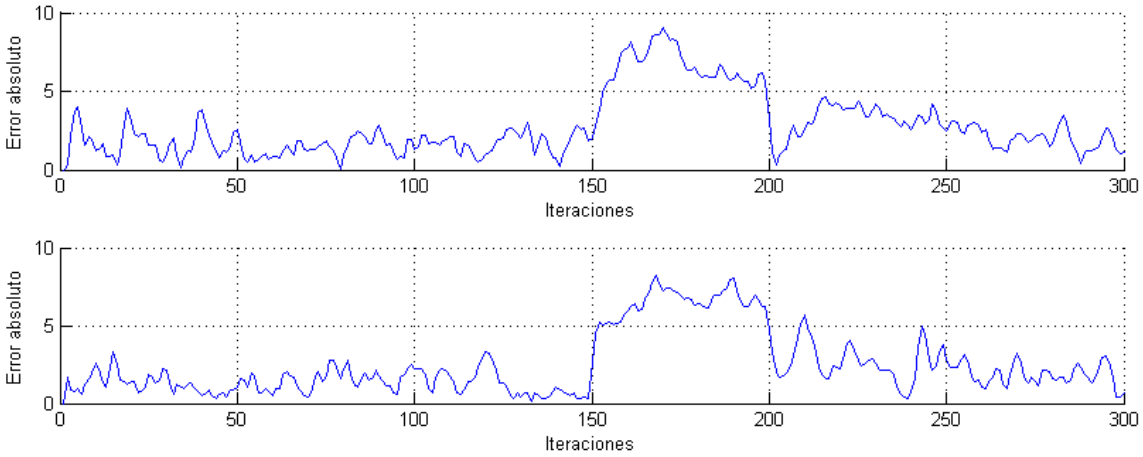


Figura 4-31. Zigzag, Error con modelo menos confiable,
a) Filtro de Kalman, b) Filtro de Información

4.5.2 IF 2: Cuadrado, Un GPS

4.5.2.1 GPS buena calidad, modelo poco confiable

Las Figuras 4-32 y 4-33 muestran los resultados de la re-simulación del apartado 4.4.4.2, con resultados semejantes entre un filtro y otro.

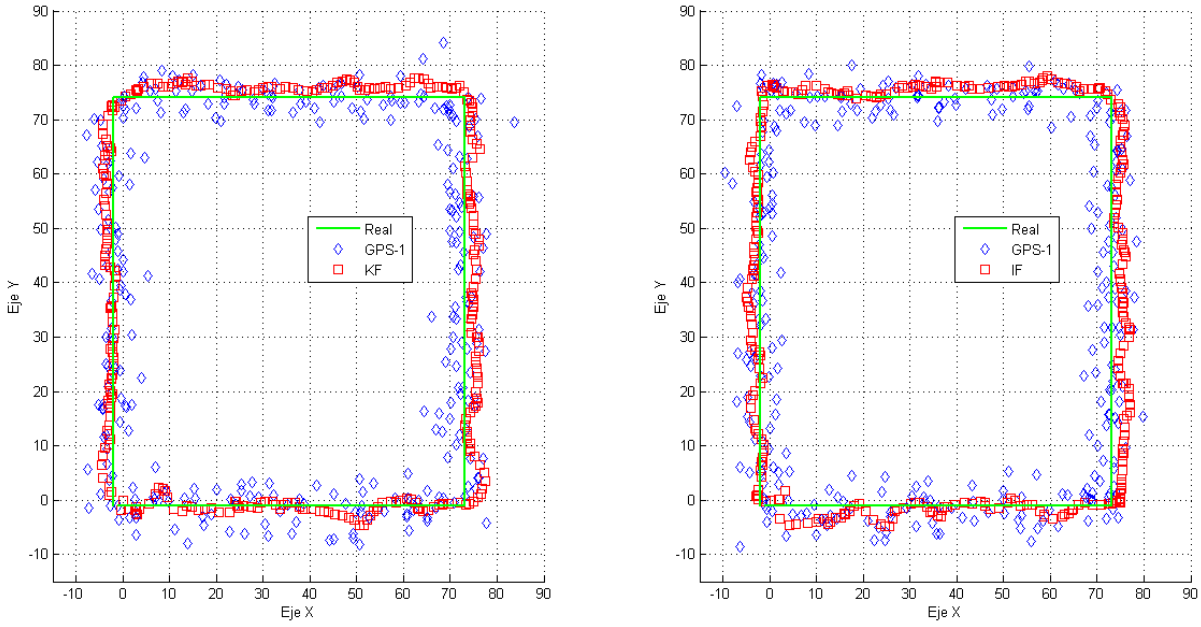


Figura 4-32. Cuadrado, GPS con $w_g = 3\ m$, error del modelo: $v = 0{,}5\ m$,
a) Filtro de Kalman, b) Filtro de Información

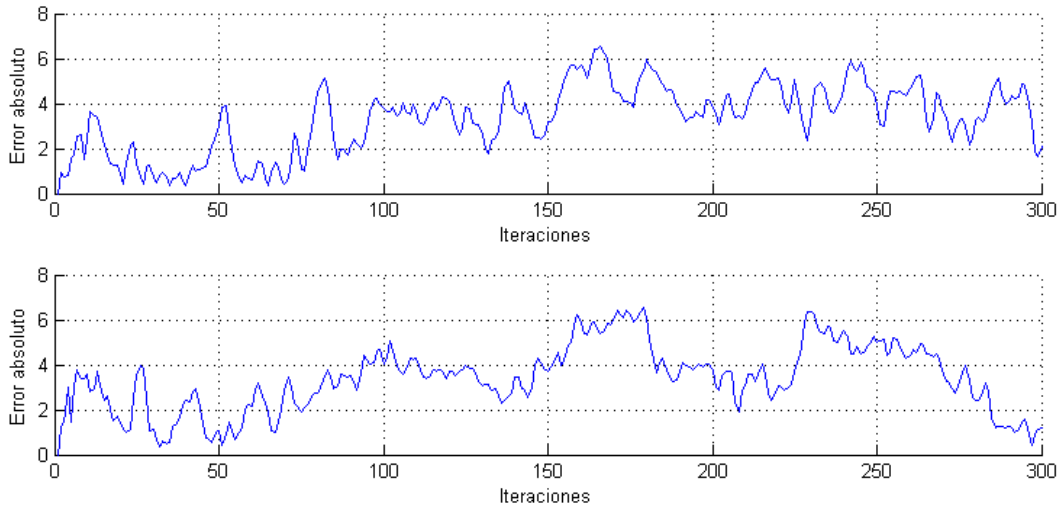


Figura 4-33. Cuadrado, Error con modelo poco confiable,
a) Filtro de Kalman, b) Filtro de Información

.

4.5.2.2 Túnel, GPS buena calidad, modelo poco confiable

Igual que la prueba anterior, ambos filtros tienen el mismo comportamiento ante el mismo suceso como muestran las Figuras -34 y 4-35.

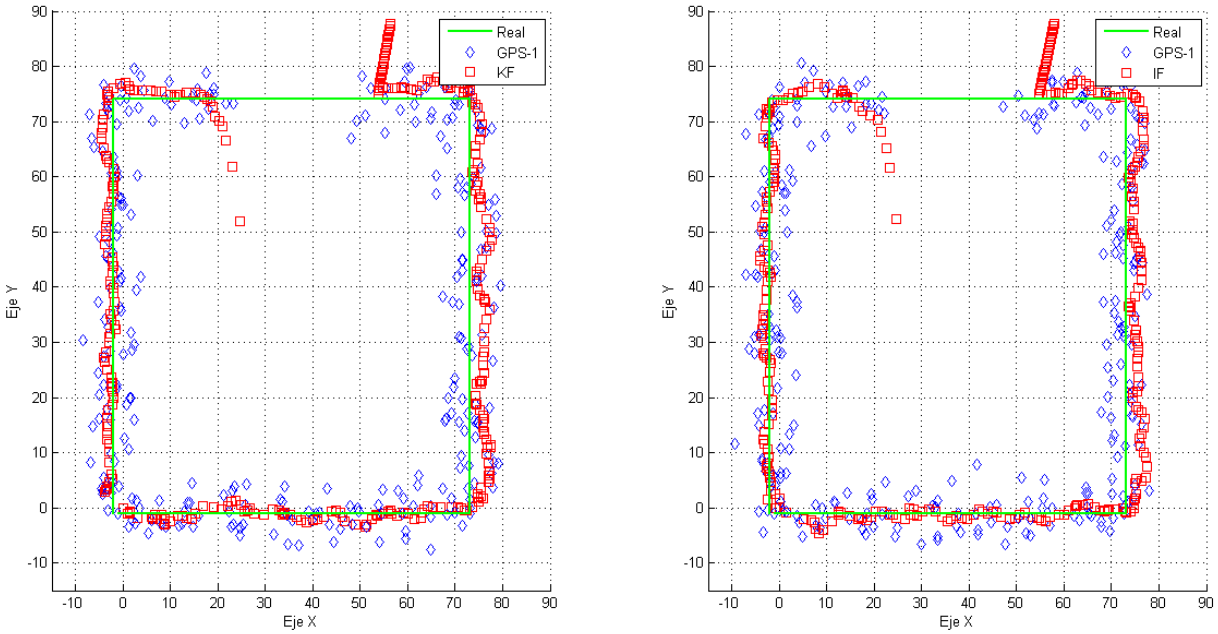


Figura 4-34. Cuadrado 2, GPS con $w_g = 3\ m$, error del modelo: $v = 0{,}5\ m$,
a) Filtro de Kalman, b) Filtro de Información

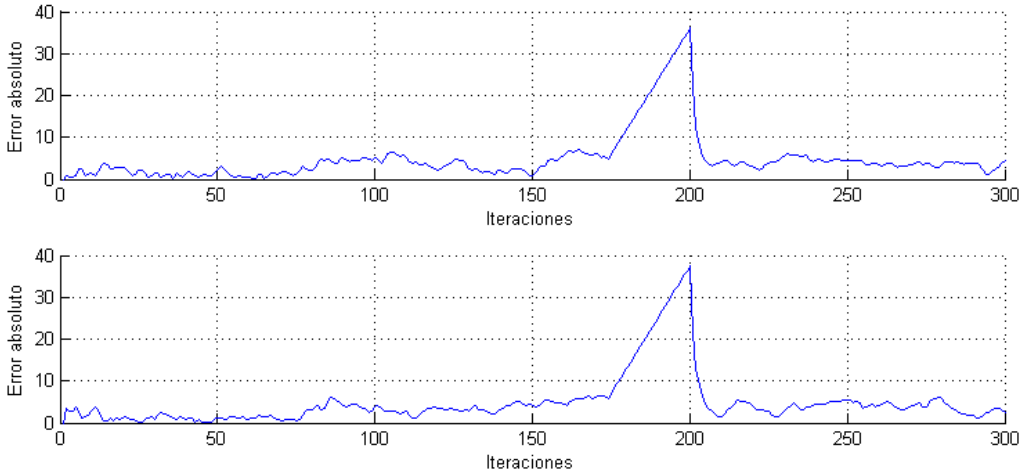


Figura 4-35. Cuadrado 2, Error con modelo poco confiable,
a) Filtro de Kalman, b) Filtro de Información

Con estos ensayos queda lo bastante claro la exactitud de resultados que muestran ambos filtros. Por lo que se dan por concluidas las simulaciones con las versiones no extendidas de los filtros en el entorno de Matlab.

4.5.3 IF 3: ROS

La configuración de este ensayo es idéntica a la del apartado 4.4.8, con la diferencia del algoritmo interno y del detalle sobre la inicialización de la matriz de información en lugar de la de covarianza (mencionado en el apartado 4.3).

De las Figuras 4-36 y 4-37, se desprende un comportamiento similar al que se produce en el apartado anteriormente mencionado: el filtro sigue correctamente la posición menos cuando se pierde contacto con la información externa y deja de ejecutarse la fase de actualización.

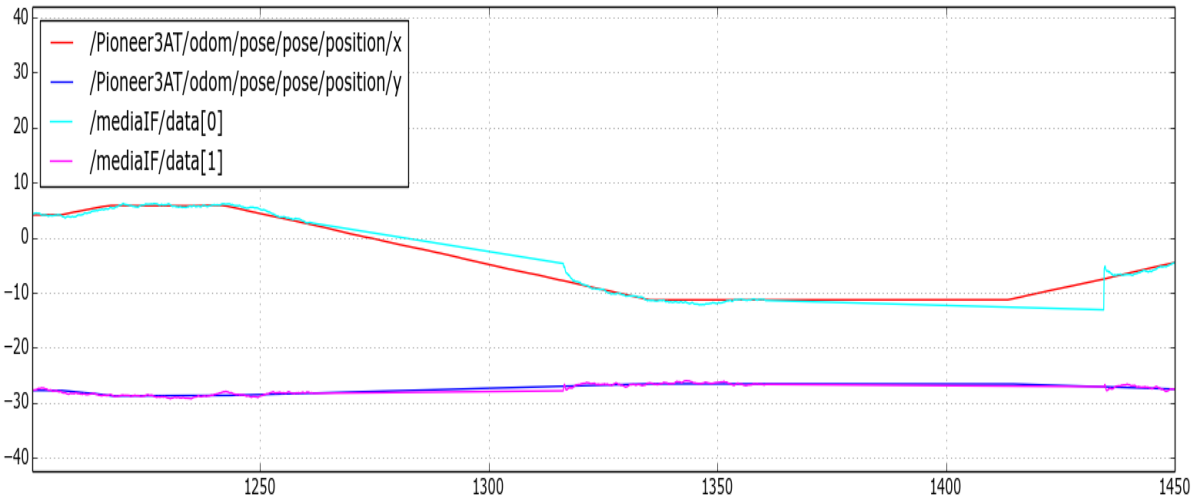


Figura 4-36. ROS, Seguimiento de la trayectoria con IF

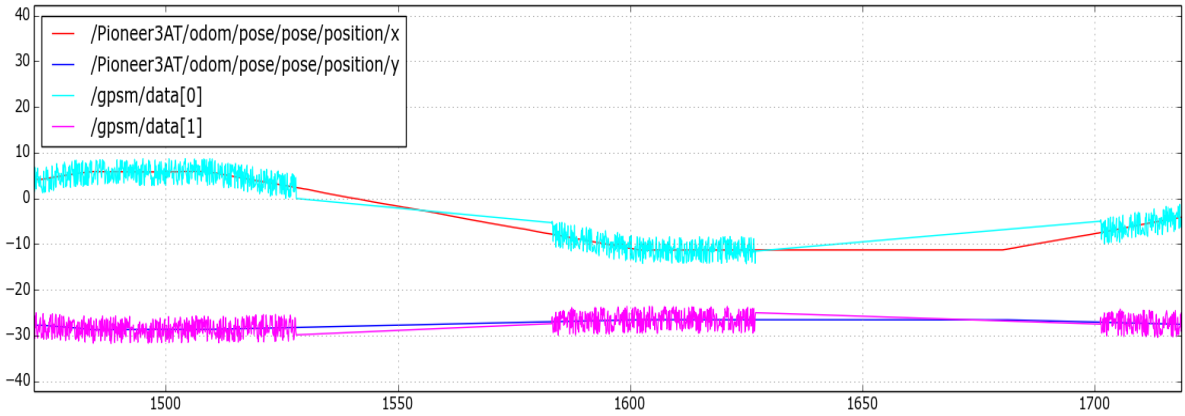


Figura 4-37. ROS, Señal GPS con dos tramos sin cobertura

Como añadido, se ha realizado con este filtro un ensayo sustituyendo el modelo rectilíneo por el estático. Durante esta prueba, el robot, estando en movimiento, pierde la comunicación con el GPS, momento en el cual la respuesta del filtro es permanecer donde estaba hasta recuperar la información externa. Esto se puede ver en la Figura 4-38, donde, al cortarse el GPS (Figura 4-39), las trayectorias del vector media se vuelven totalmente horizontales.

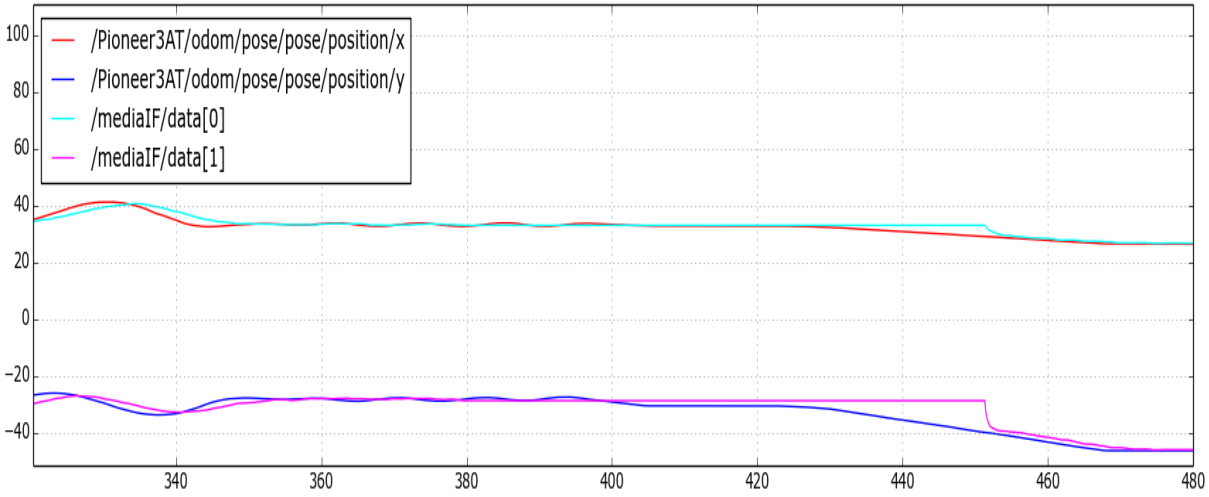


Figura 4-38. ROS, Seguimiento de la trayectoria con IF usando modelo estático

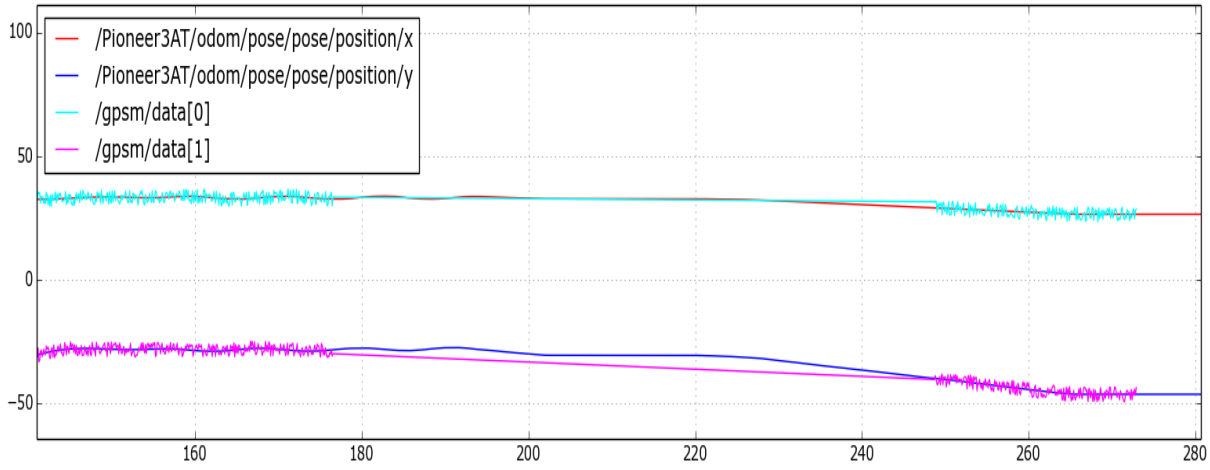


Figura 4-39. ROS, Señal GPS para prueba con modelo estático

4.6 Simulaciones con el Filtro de Kalman Extendido

A lo largo de este apartado se muestra el comportamiento de la versión extendida del filtro de Kalman, capaz de lidiar con mayor número de posibles escenarios gracias a la linealización de las ecuaciones que los envuelven.

Este filtro no será objeto de tantos ensayos como el KF debido a que comparten la misma naturaleza y comportamiento. De esta forma, un correcto EKF responderá de forma similar ante similares variaciones de parámetros.

4.6.1 EKF 1: Zigzag

Se procede a simular la primera trayectoria utilizada en el trabajo. Se realizan pruebas con sensores de diferente desviación (Figura 4-42). Comparando las Figuras 4-40 y 4-41, se aprecia una mejor actuación del filtro respecto su versión simple. Esto es debido a la baja dispersión que poseen los sensores de rango. Mirando la Figura 4-41 puede parecer que este método es algo más sensible a la calidad de los sensores, principalmente, porque necesita de varios sensores simultáneamente para localizarse, pero es importante considerar la poca longitud del trayecto, lo que favorece la apariencia de mayor error.

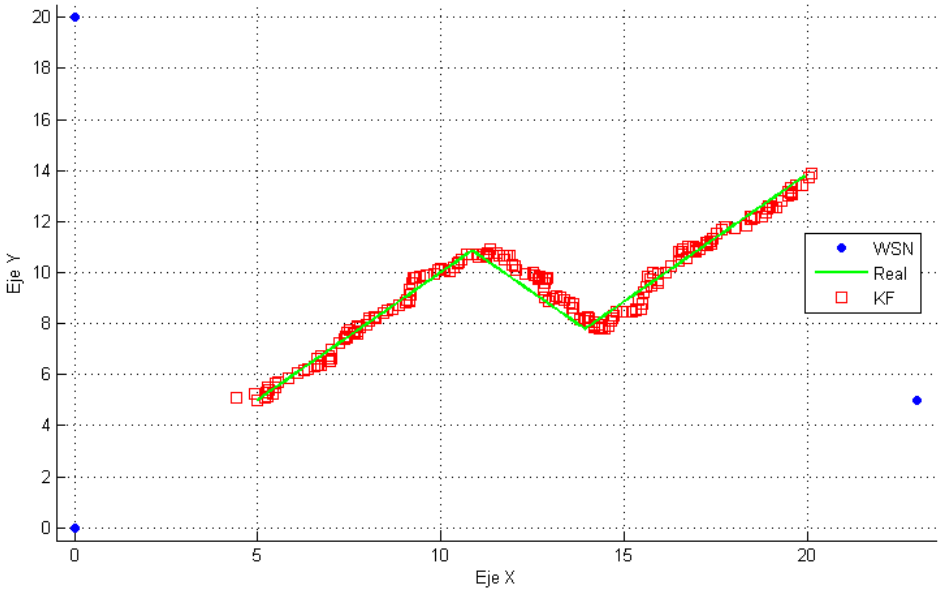


Figura 4-40. Zigzag, sensores con desviación $w_r = 0{,}5\ m$, modelo $v = 0{,}1\ m$

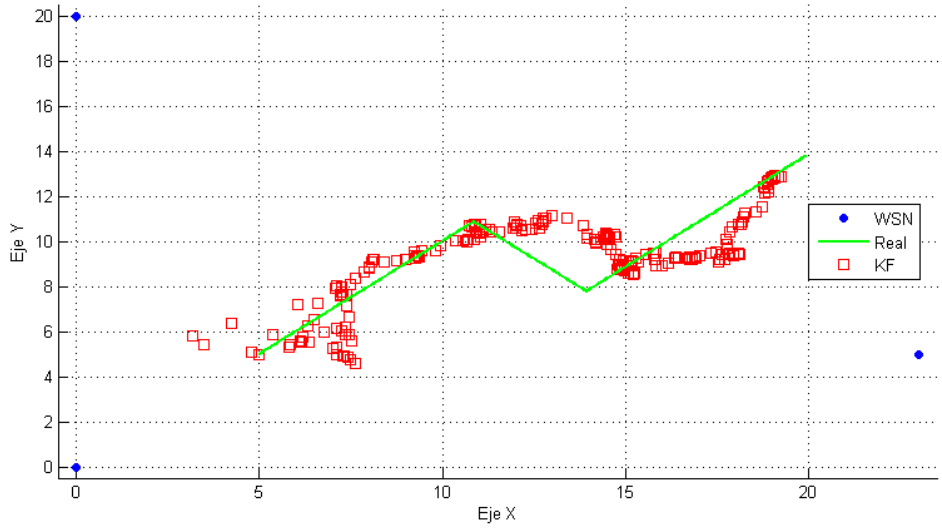


Figura 4-41. Zigzag, sensores con desviación $w_r = 2{,}5\ m$, modelo $v = 0{,}1\ m$

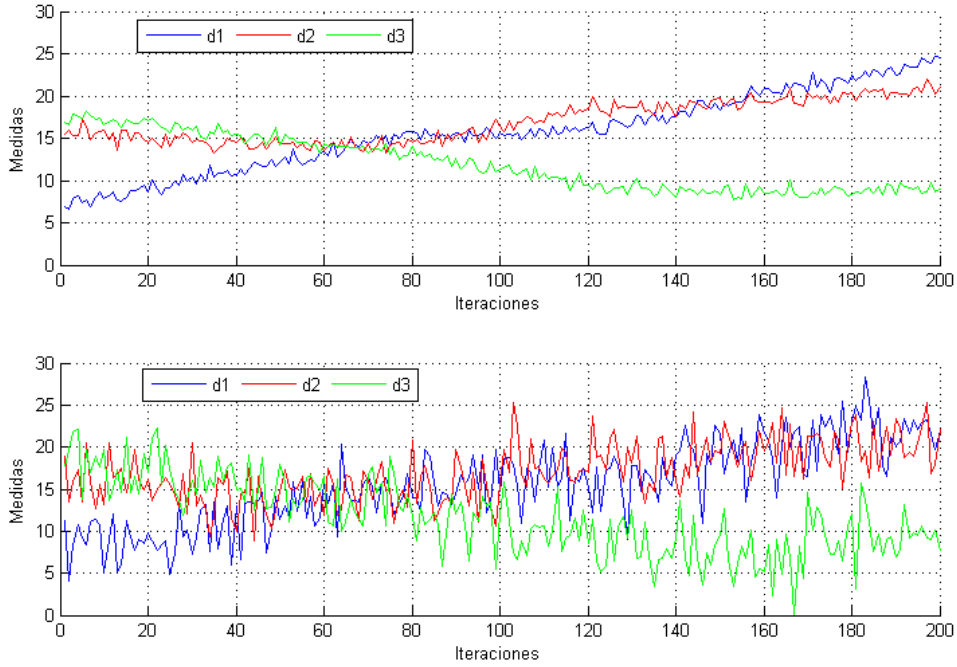


Figura 4-42. Zigzag, distancias medidas con sensores de: a) $w_r = 0{,}5\ m$, b) $w_r = 2{,}5\ m$

La Figura 4-43 muestra de forma objetiva que, a pesar de empeorar la calidad de los sensores, el error, realmente, se incrementa en poco más de un metro. Es importante señalar que la trayectoria no debe coincidir en ningún momento con los nodos puesto que dentro del Jacobiano se alcanzarían valores infinitos.

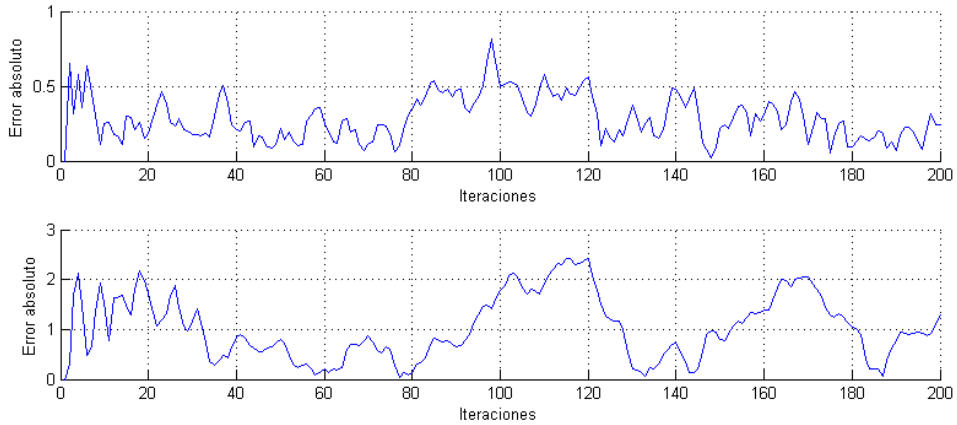


Figura 4-43. Zigzag, error en posición: a) $w_r = 0{,}5\ m$, b) $w_r = 2{,}5\ m$

4.6.2 EKF 2: Cuadrado

En esta ocasión se recorre la trayectoria cuadrada como muestra la Figura 4-44. Esta simulación logra un menor error (Figura 4-45) debido al empleo de medidas de los sensores de rango (Figura 4-46), las cuales tienen bastante menos desviación en comparación con las del GPS.

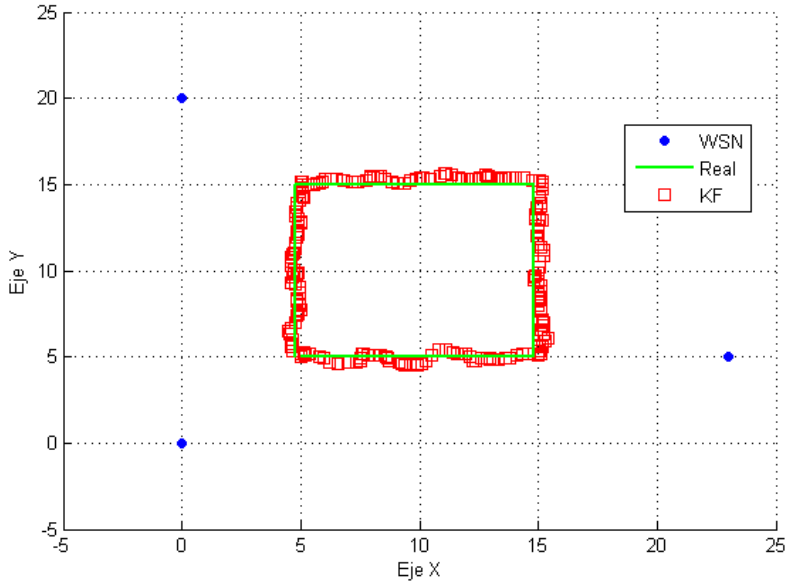


Figura 4-44. Cuadrado, sensores con desviación $w_r = 0{,}5\ m$, modelo $v = 0{,}1\ m$

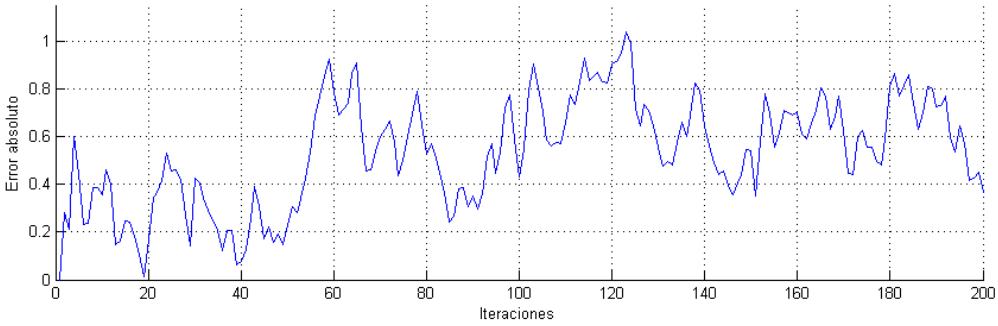
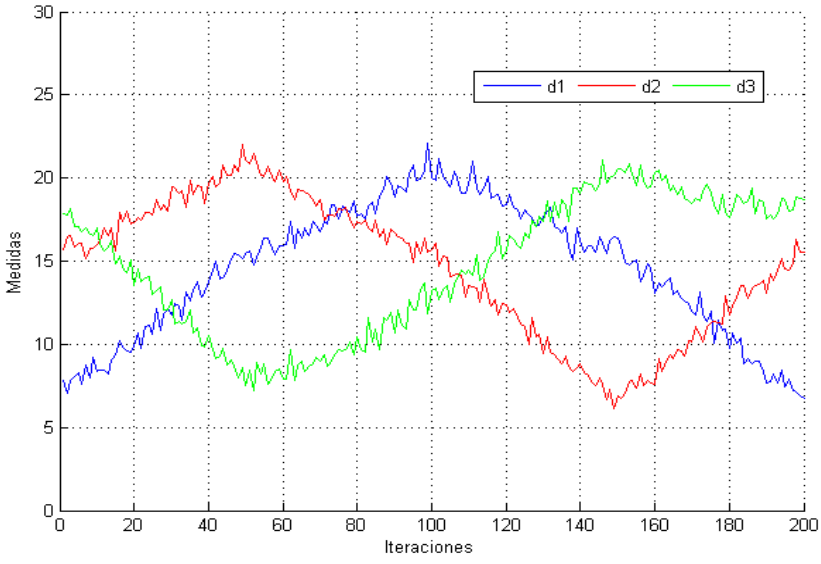


Figura 4-45. Cuadrado, error en posición



/

Figura 4-46. Cuadrado, distancias medidas

4.6.3 EKF 3: Cuadrado bajo túnel

En esta simulación se hará una prueba de lo que ocurre al perder uno o varios de los nodos de forma forzada (Figura 4-48).

Curiosamente, en la Figura 4-47 puede observarse que el algoritmo se puede sostener con sólo dos nodos, cuando la teoría de la localización por triangulación, indicaría que entrarían en conflicto dos soluciones distintas. Esto es gracias a la estimación con la que parte el filtro, que le permite decantarse por la solución más acorde a ésta. Con un único nodo y la media estimada le resulta imposible hacerlo correctamente.

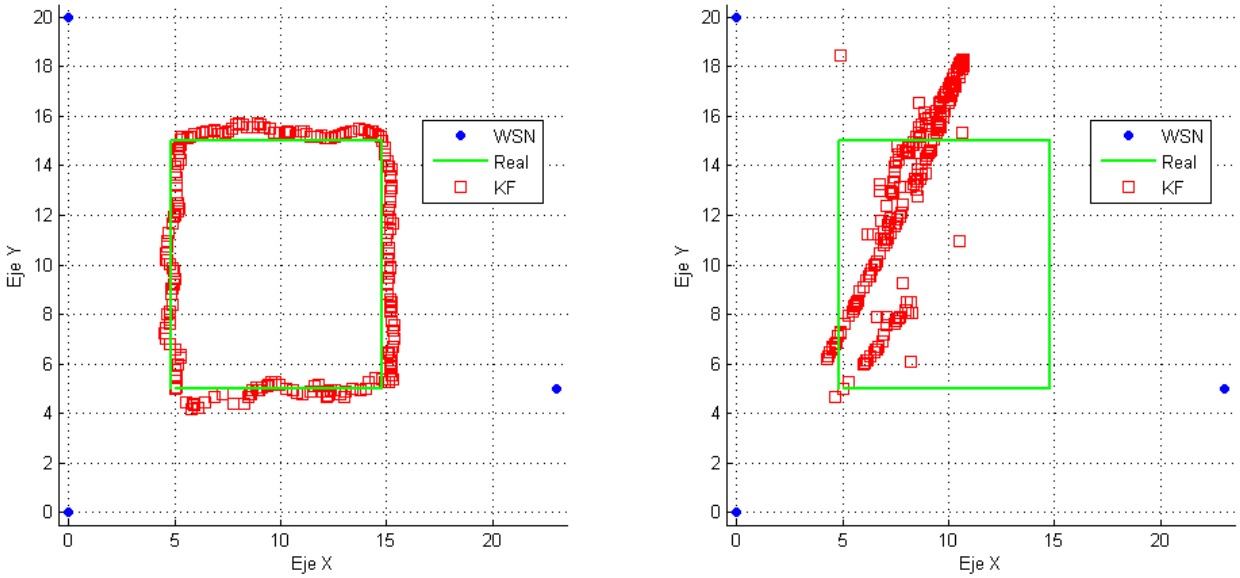


Figura 4-47. Cuadrado, trayectoria con: a) Un nodo perdido, b) Dos nodos perdidos

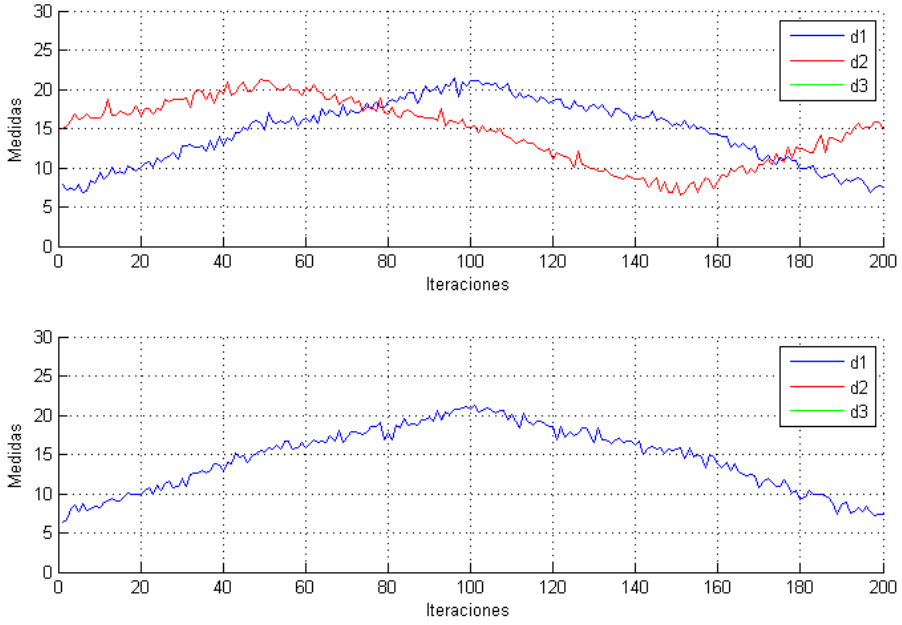


Figura 4-48. Cuadrado, medidas con: a) Un nodo perdido, b) Dos nodos perdidos

Es interesante comparar la gráfica a) de la Figura 4-49 con la Figura 4-45, observando que el error apenas aumenta al perderse el primer nodo. Exponiéndolo algebraicamente, se encuentra un problema que requiere tres restricciones para su determinación, siendo válidas para ser dichas restricciones tanto las distancias a los nodos como el vector estimado previamente.

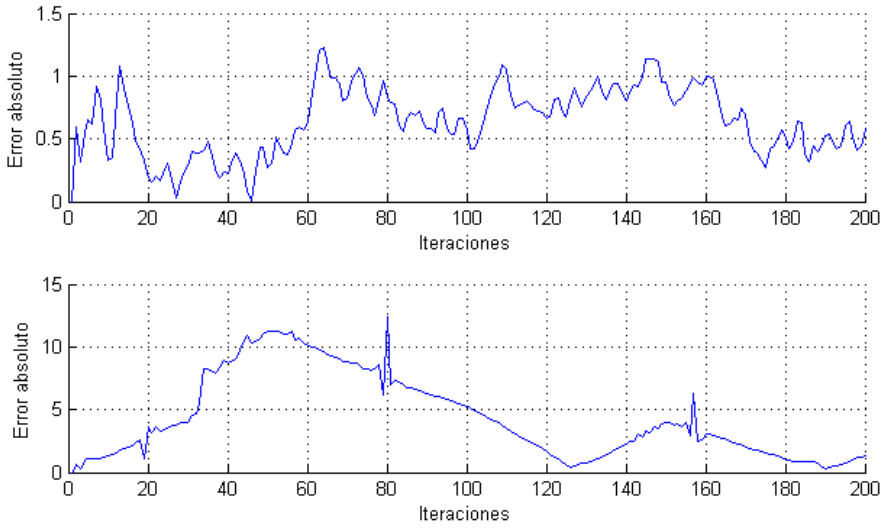


Figura 4-49. Cuadrado, errores con: a) Un nodo perdido, b) Dos nodos perdidos

4.6.4 EKF 4: Sobrepasando los límites

En esta ocasión, se analizará que ocurre cuando la trayectoria del robot queda fuera del alcance de algún nodo. Partiendo de los resultados de la simulación anterior, el filtro debe ser capaz de mantener el seguimiento sin problema, aunque con mayor error (Figura 4-52), mientras se pierda únicamente la señal de uno de los nodos, lo cual se verifica en las Figuras 4-50 y 4-51.

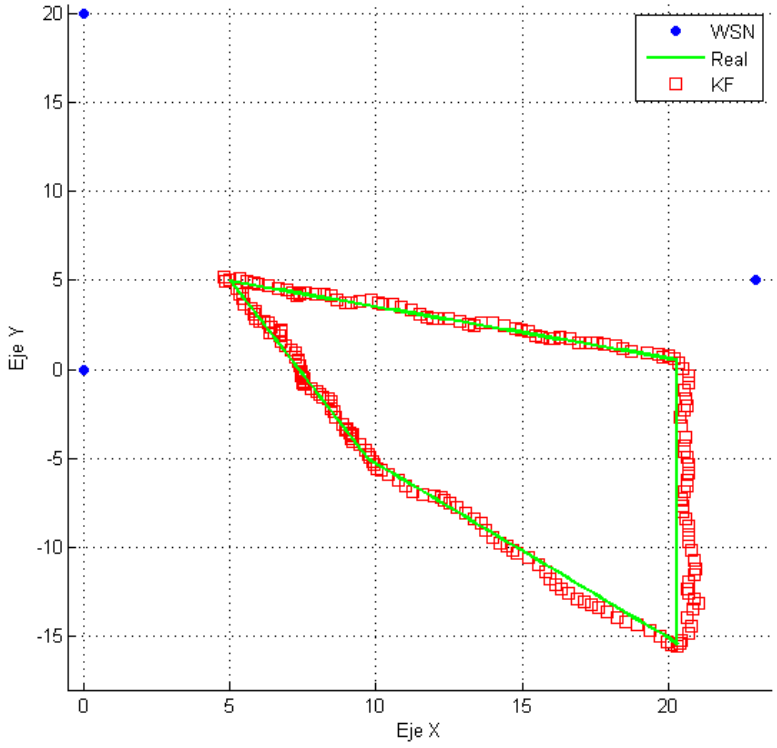


Figura 4-50. Cuadrilátero, sensores $w_r = 0{,}5\ m$, modelo $v = 0{,}1\ m$

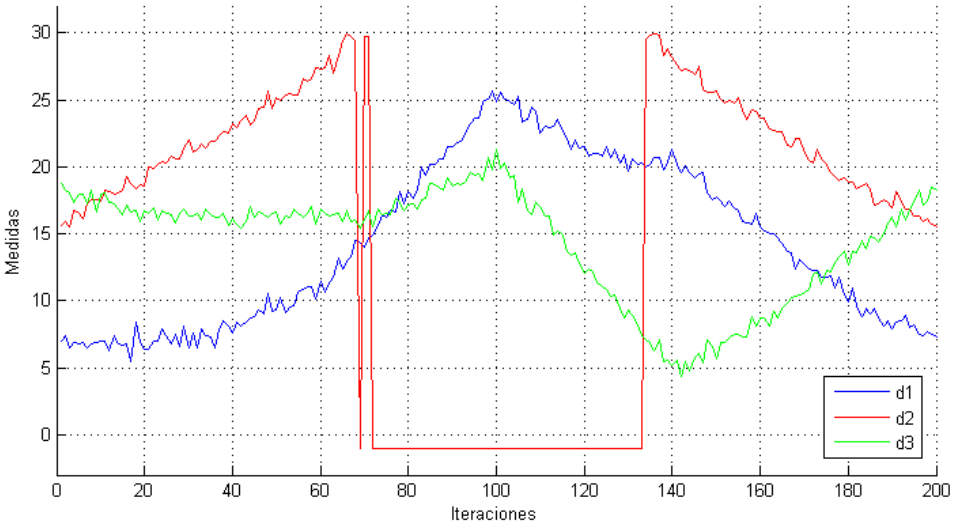


Figura 4-51. Cuadrilátero, distancias medidas

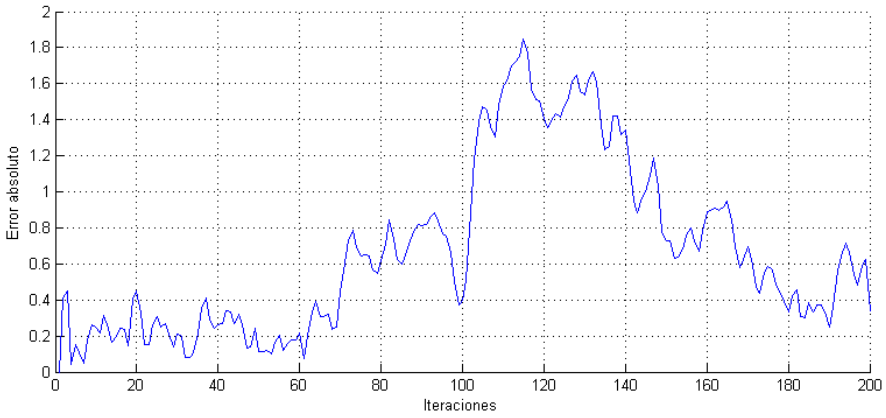


Figura 4-52. Cuadrilátero, error en posición

4.6.5 EKF 5: ROS

Antes de proceder, es necesario indicar una serie de cambios en la configuración inicial en los filtros extendidos respecto a los nodos WSN. Lo demás sigue lo indicado en el apartado 4.4.8.

Se mantiene el número de nodos pero con las siguientes posiciones:

Tabla 4–3. Nodos WSN (ROS)

Posición / Nodo	$x\ (m)$	$y\ (m)$
$\mathbf{N_1}$	0	0
$\mathbf{N_2}$	0	20
$\mathbf{N_3}$	-23	-5

El tercer nodo sufre una traslación simétrica respecto al origen de coordenadas debido a que la posición del robot, en los ejes predefinidos, es $(-9, -22)$. También con motivo de facilitar que el robot se encuentre dentro del rango de estos tres nodos, se amplía su distancia máxima a $L = 50\ m$.

El seguimiento en la Figura 4-53 puede parecer muy ruidoso, pero esto es debido a la escala de tiempo en la que se observa. Para corroborarlo, basta con observar el error en la Figura 4-54. En la Figura 4-55 se muestran las medidas a lo largo del tiempo, siendo cortadas en una gran parte del recorrido.

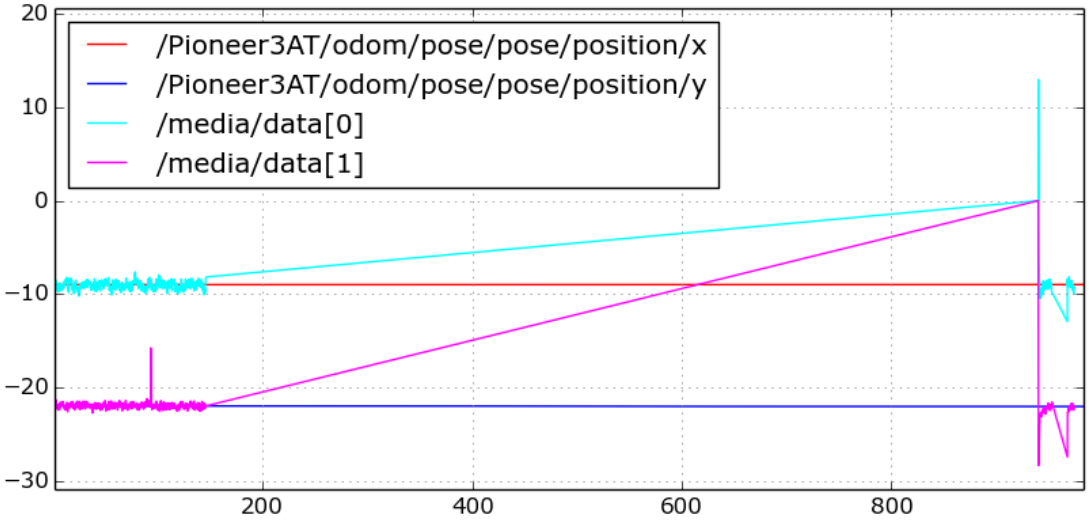


Figura 4-53. ROS, Seguimiento con EKF

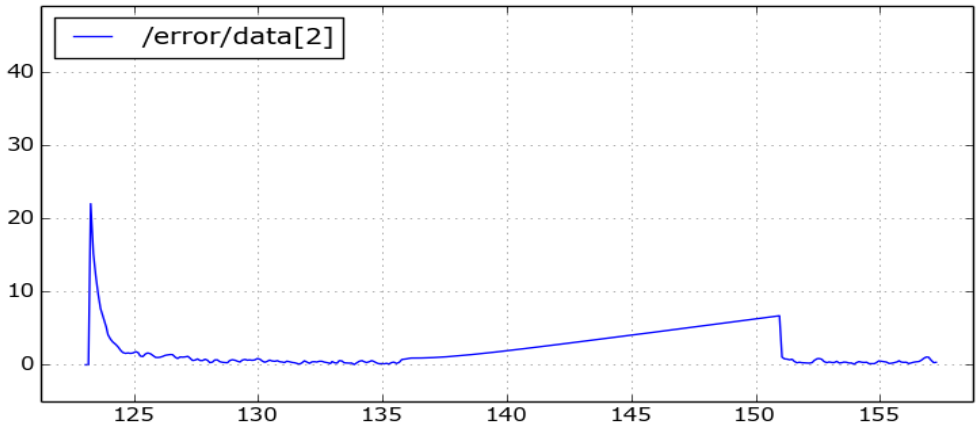


Figura 4-54. ROS, Error en posición con EKF

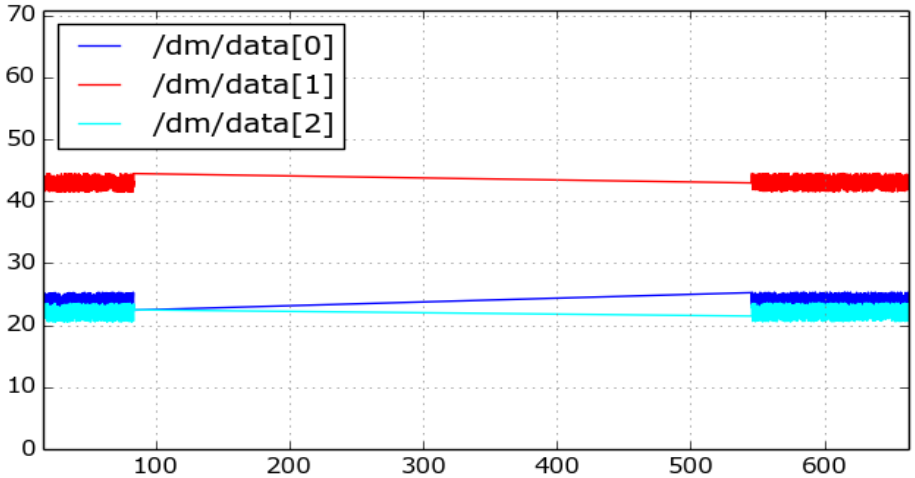


Figura 4-55. ROS, Medidas de rango con tramo sin cobertura

Analizando el error de este ensayo, el primer pico que se ve corresponde a la diferencia entre la posición inicial del robot y la parametrizada en el filtro. Posteriormente, aumenta el error siguiendo una forma de rampa de pendiente baja durante todo el tiempo que se pierde la información de los sensores. Para terminar recuperando un correcto seguimiento tras volver a recibir datos.

En una segunda simulación, se probará que acontece en caso de sobrepasar los límites del sensor. Como refleja la Figura 4-56, una vez el robot ronda el límite del sensor se produce cierta intermitencia en la información debida al ruido, tras lo cual, al avanzar un poco más, el sensor queda totalmente incapaz de enviar información. Además, anteriormente se sabotea, intencionadamente, la información global para mostrar, con un poco más de detalle, el funcionamiento del EKF.

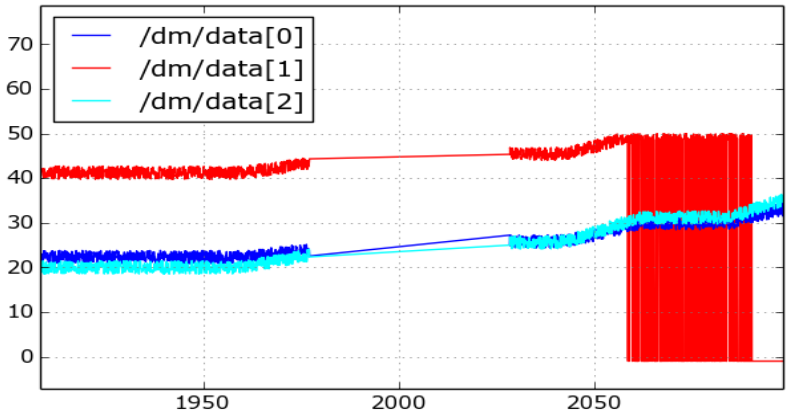


Figura 4-56. ROS, Distancias recibidas

Como es de esperar, la versión extendida del filtro también posee un funcionamiento independiente de las medidas basado en la actualización del modelo, manteniendo la trayectoria seguida en el momento de perder contacto con los nodos. El problema viene cuando se cae uno de los tres únicos nodos de los que recoge información, siendo especialmente caótica la etapa en la que se recibe información intermitente del nodo sobrepasado hasta que termina de perderse. Todo esto viene reflejado en las Figuras 4-57 y 4-58.

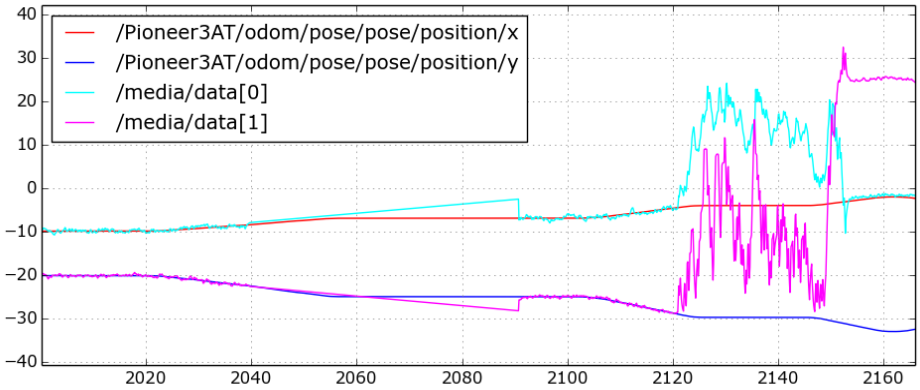


Figura 4-57. ROS, Seguimiento con EKF 2

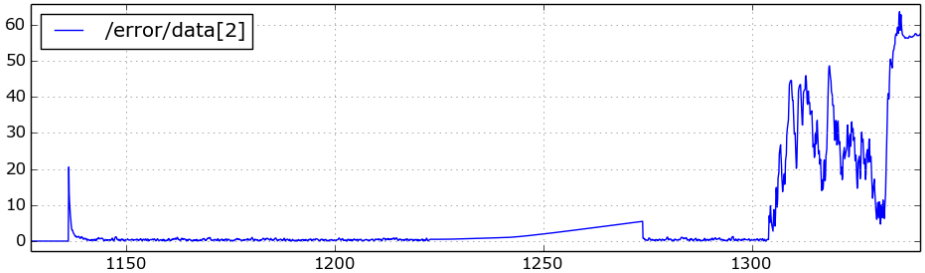


Figura 4-58. ROS, Error en posición con EKF 2

4.7 Simulaciones con el Filtro de Información Extendido

De igual manera que sucede con la relación entre el KF y el IF, este filtro arroja idénticos resultados a la versión extendida del KF, puesto que las ecuaciones que siguen son iguales. De forma que, correctamente implementado el algoritmo, no muestra diferencia alguna. La fundamental diferencia es a la hora de computar la información, cosa que en las simulaciones mostradas no se pude resaltar.

Así pues, tras comparar alguna trayectoria genérica, se reserva para el EIF alguna simulación especial como probar el modelo estático.

4.7.1 EIF 1: Cuadrilátero, EIF vs EKF

Se toma, como trayectoria a comparar, la del último ensayo (Figura 4-59) por considerarse la más interesante además de que pone a prueba la robustez del filtro al salirse del alcance de uno de los nodos.

Observando la Figura 4-60 junto con la Figura 4-61, se ve que ambos filtros actúan con una precisión semejante teniendo el máximo de error al perder uno de los nodos.

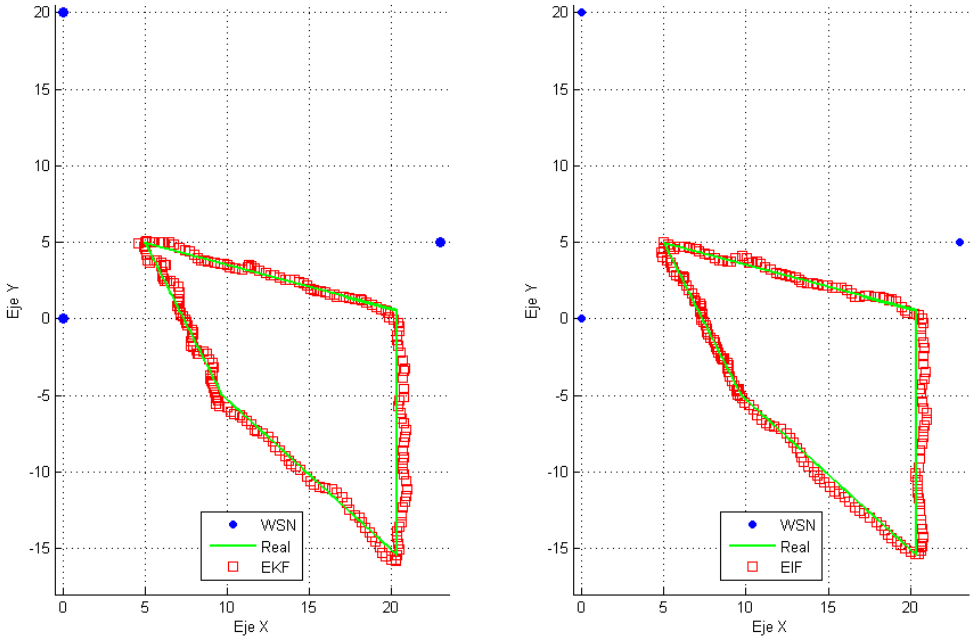


Figura 4-59. Cuadrilátero, trayectoria con: a) EKF, b) EIF

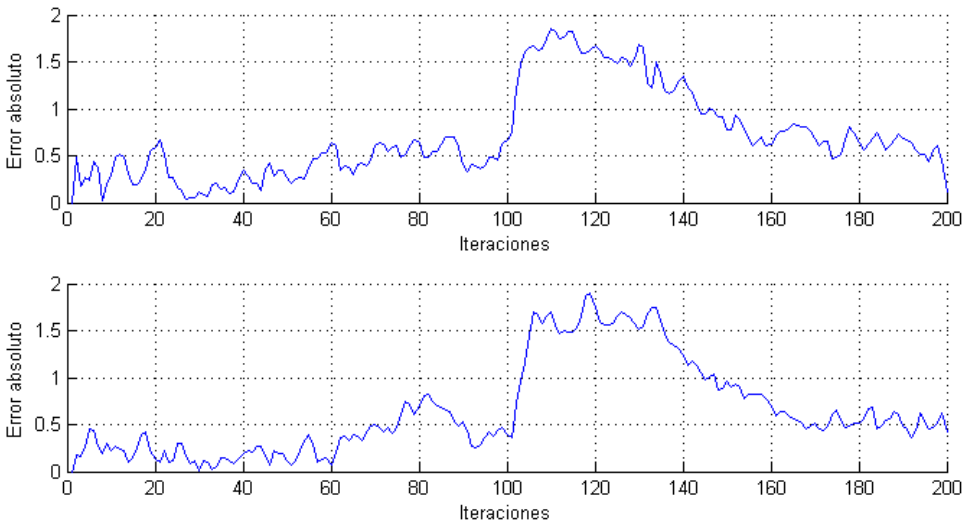


Figura 4-60. Cuadrilátero, error en posición: a) EKF, b) EIF

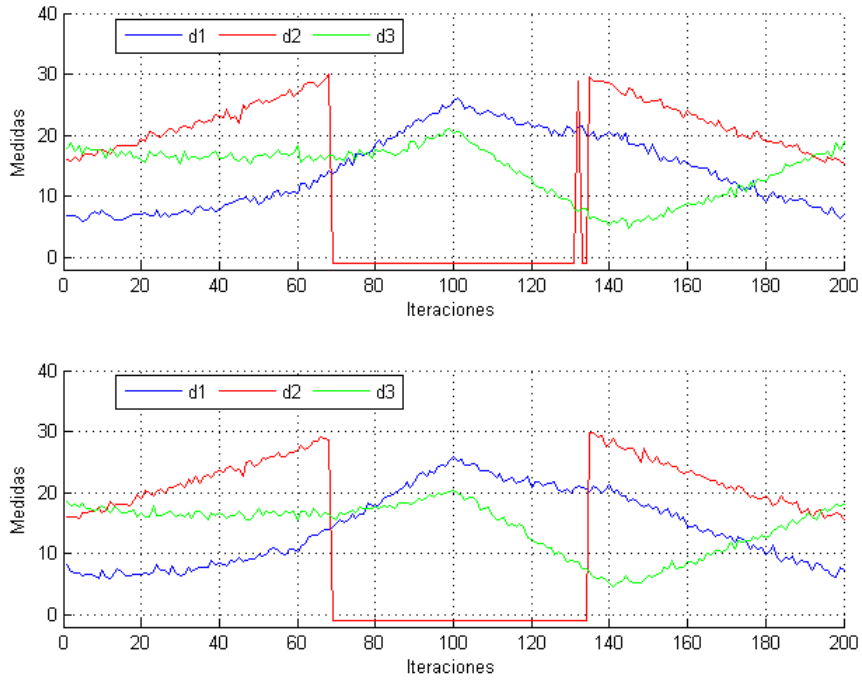


Figura 4-61. Cuadrilátero, medidas: a) EKF, b) EIF

4.7.2 EIF 2: Modelo estático

Para esta prueba se utilizará una combinación de la trayectoria del zigzag y el semáforo, quedándose el robot detenido un breve lapso de tiempo antes de realizar el primer quiebro.

Es fácil predecir los acontecimientos: por norma general, el filtro con el modelo estático realiza mejor la parada, además del giro puesto que partir del reposo produce menos conflicto que si está todo el rato intentando moverse adelante. Por otra parte, el filtro estático va lastrando el movimiento, lo que termina produciendo un retraso que se observa al final de la trayectoria en la segunda gráfica en la Figura 4-62. Todo esto, además viene respaldado por los errores en la Figura 4-63.

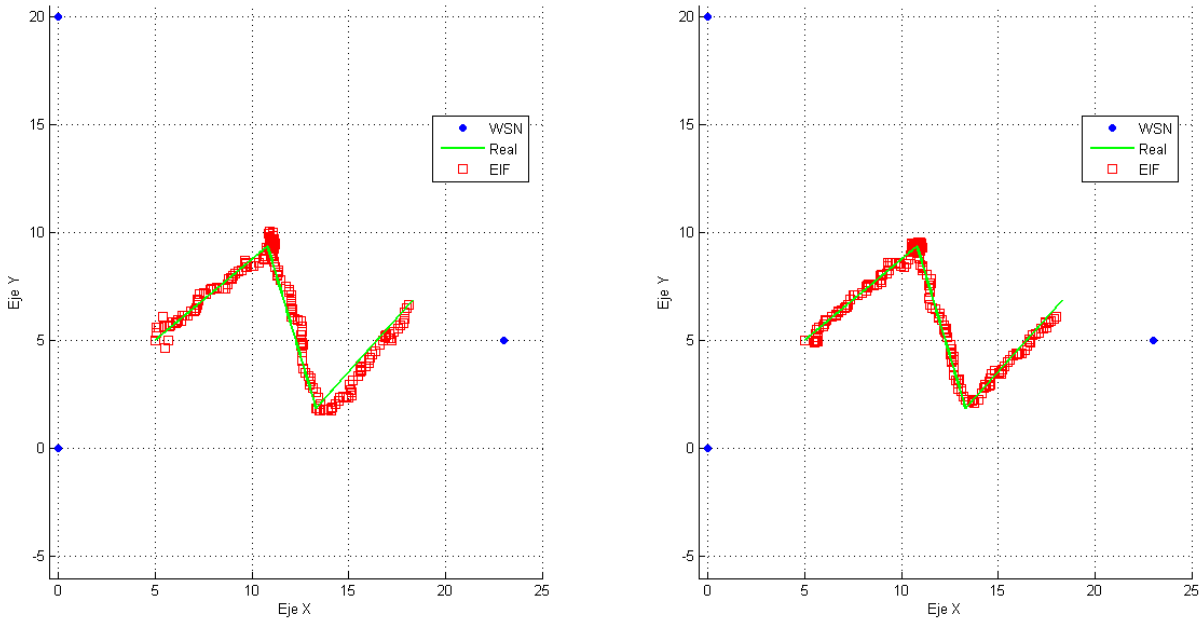


Figura 4-62. Zigzag con parada en primer giro: a) Modelo rectilíneo, b) Modelo estático

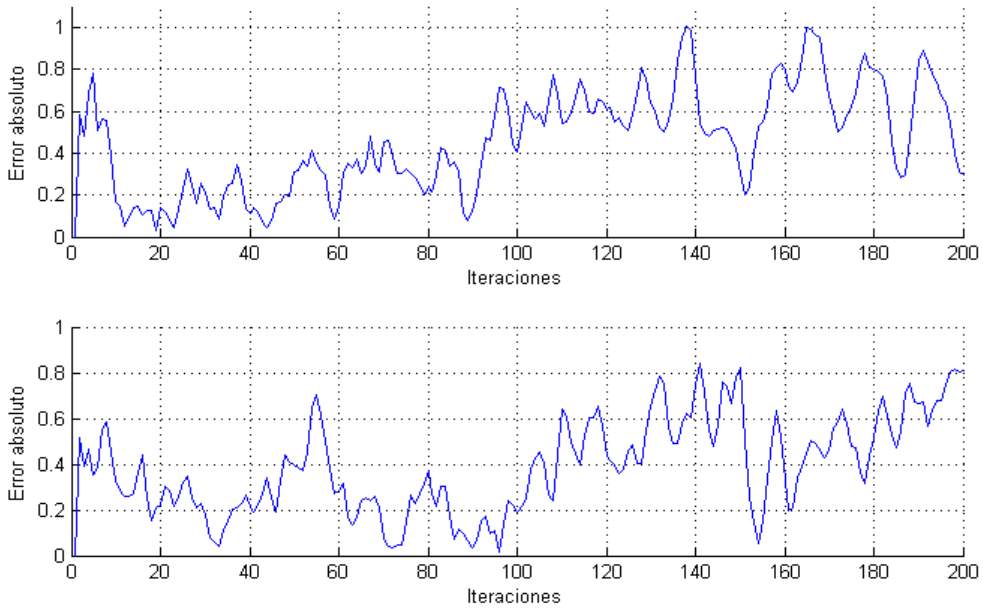


Figura 4-63. Zigzag con parada, errores: a) Modelo rectilíneo, b) Modelo estático

4.7.3 EIF 3: Fusión sensorial

Otra simulación especial que se ha reservado para completar las pruebas con este filtro, estando también implementada en el EKF, es la fusión sensorial. Esto es, la incorporación simultánea de información proveniente de sensores de rango y GPS gracias a una versión más completa del Jacobiano que considera su inclusión. Además, el código está preparado para adaptarse en circunstancias donde pierda información de todos o alguno de sus sensores.

Para este ensayo se escoge la trayectoria cuadrada, durante la cual el robot perderá a mitad de camino la señal GPS y, tras recuperarla, la de dos nodos. Para una mejor visualización de los hechos se puede recurrir a la Figura 4-64, donde, aparte de las medidas de rango, se ha añadido una representación del GPS de forma binaria, para indicar cuando está activo y cuando no.

Observando la Figura 4-65 junto a la Figura 4-66, se puede comprobar que el error, a lo largo de esta prueba, es máximo al perder dos nodos, momento en el que más confianza se deposita en el GPS siendo éste el sensor con mayor incertidumbre.

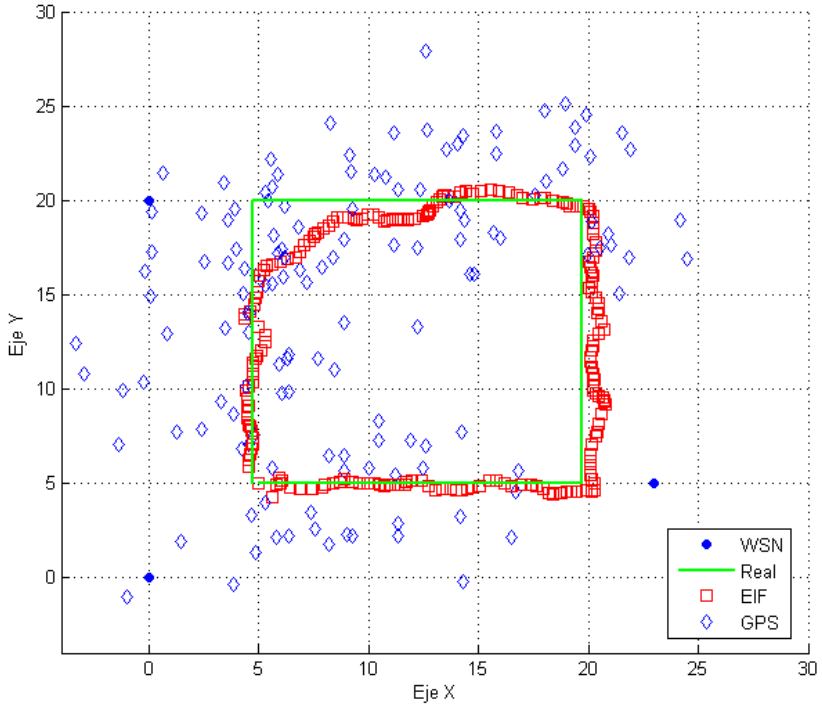


Figura 4-64. Cuadrado, sensores $w_r = 0{,}5\ m$, $w_g = 3\ m$, modelo $v = 0{,}1\ m$

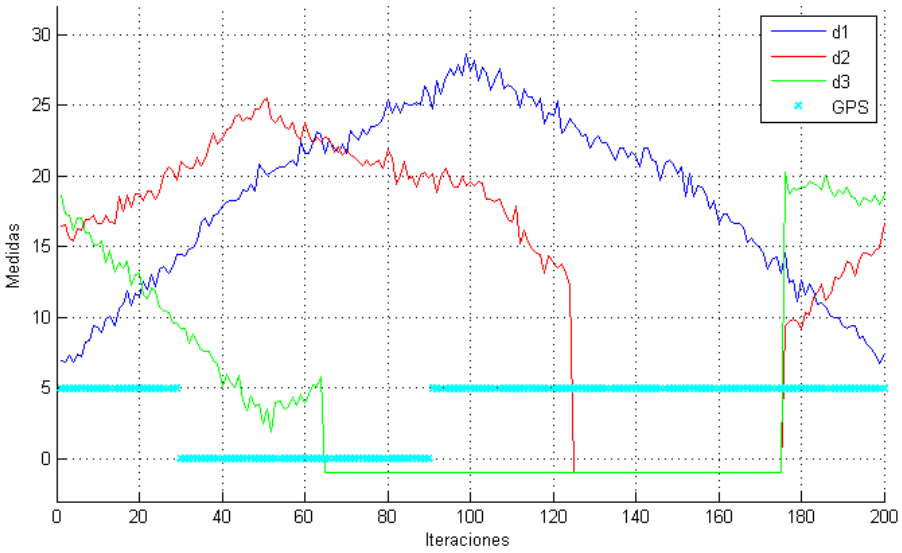


Figura 4-65. Cuadrado, sensores y rangos de funcionamiento

Figura 4-66. Cuadrado, error en prueba de fusión sensorial

4.7.4 EIF 4: ROS

La configuración en este caso es igual que la del EKF en ROS, con las particularidades mencionadas para el IF. Se realiza una demostración básica del comportamiento del filtro. Para ello, el robot es alejado del alcance de uno de los nodos, momento en que deja de incorporar medidas por su implementación. Una vez vuelve, recupera con normalidad el seguimiento hasta que se produce una desconexión completa de la información externa y recupera el modo de predicción, como se comprueba en la Figura 4-67. La disponibilidad de las medidas y el error obtenido en el recorrido se pueden observar en las Figuras 4-68 y 4-69, respectivamente.

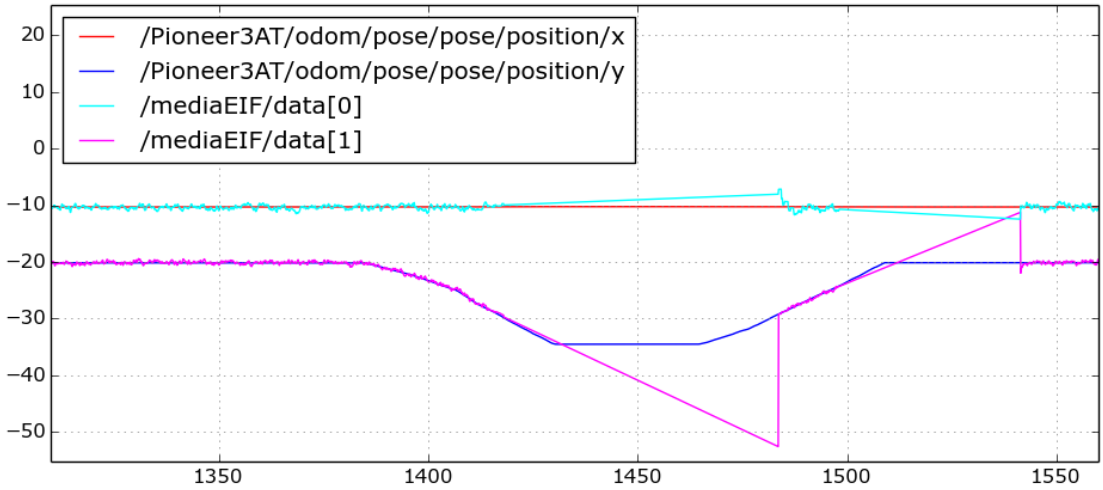


Figura 4-67. ROS, Seguimiento con EIF

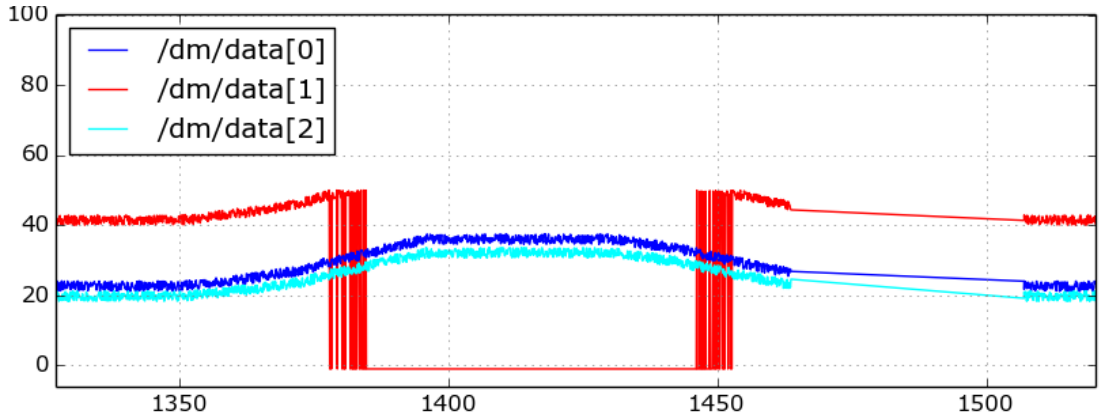


Figura 4-68. ROS, Medidas recibidas por el EIF

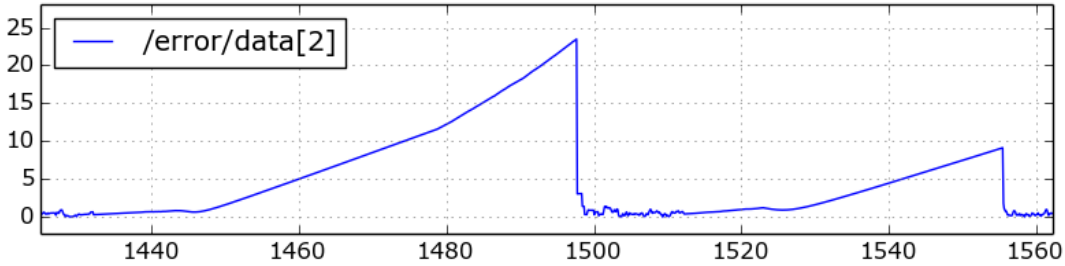


Figura 4-69. ROS, Error en posición con EIF

4.8 Simulaciones con el Filtro de Partículas

La configuración inicial de este filtro es idéntica al del EKF, con la diferencia, dado lo singular de este filtro, de que, en lugar de una única media (vector de estado), se inicializan tantas medias como partículas compongan el filtro.

Para este caso, se emplean un total de 500 partículas con una posición aleatoria entre $[-20\ ,40]$ para cada eje, además a las velocidades se les asigna valores entre $[-0.5\ ,0.5]$. Un ejemplo de esta disposición se muestra en la Figura 4-70. Como se indica en el párrafo anterior, la posición de los nodos es la misma que en los demás filtros.

En un comienzo todas las partículas poseen el mismo peso, a espera de pasar por las etapas de predicción y verosimilitud respecto a las medidas reales.

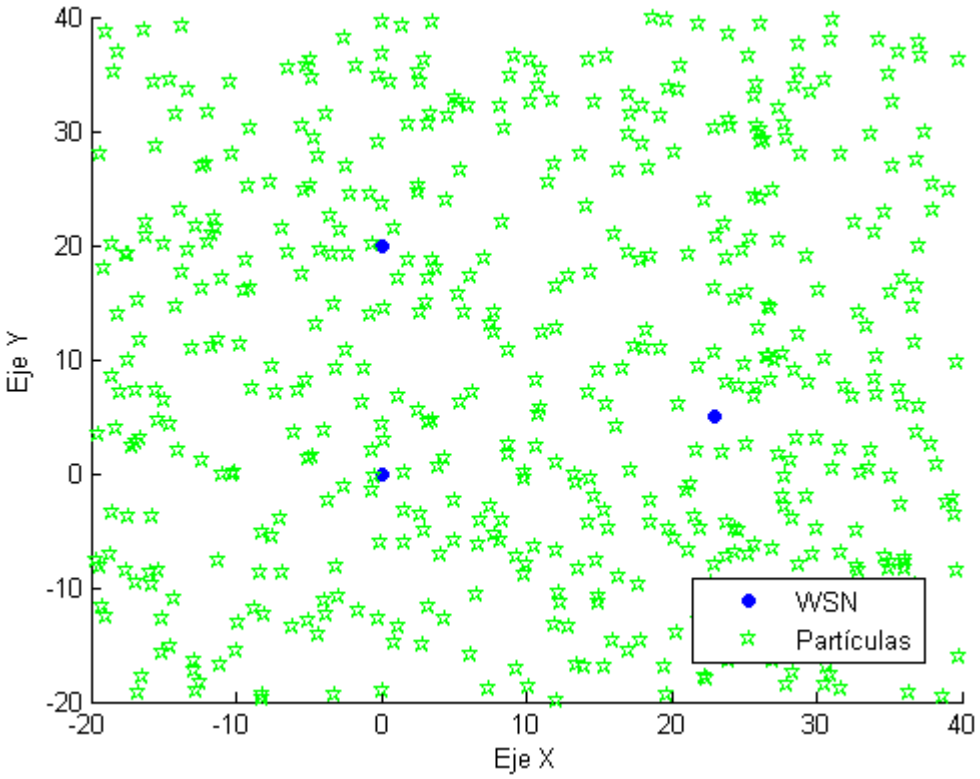


Figura 4-70. Pre-inicialización de las partículas

A medida que se va ejecutando el proceso de resampling, la posición de las partículas cambia por completo para acercarse siempre a las partículas con mayor peso. Es interesante indicar que el programa da cabida a elegir cierta frecuencia en la que se produce este muestreo, siendo innecesaria una ejecución constante tras cada ciclo. Este proceso, en las simulaciones se realiza en cada ciclo, a no ser que se indique lo contrario.

El valor de los pesos, en tanto por mil, durante la ejecución del primer ciclo se muestra en la Figura 4-71. Concretamente, se representa la situación en la inicialización, tras la fase de actualización y tras la fase de remuestreo. En un inicio, a falta de información, todas las partículas poseen la misma probabilidad (peso) de ser la correcta, tras lo cual se corrobora la verosimilitud de las partículas con el estado real mediante los sensores, dejando algunas por encima de la media. Una vez realizado el resampling, las partículas más pesadas tienen más probabilidad de perdurar e incluso de multiplicarse, explicando así que algunas compartan peso en la tercera gráfica de la Figura 4-71.

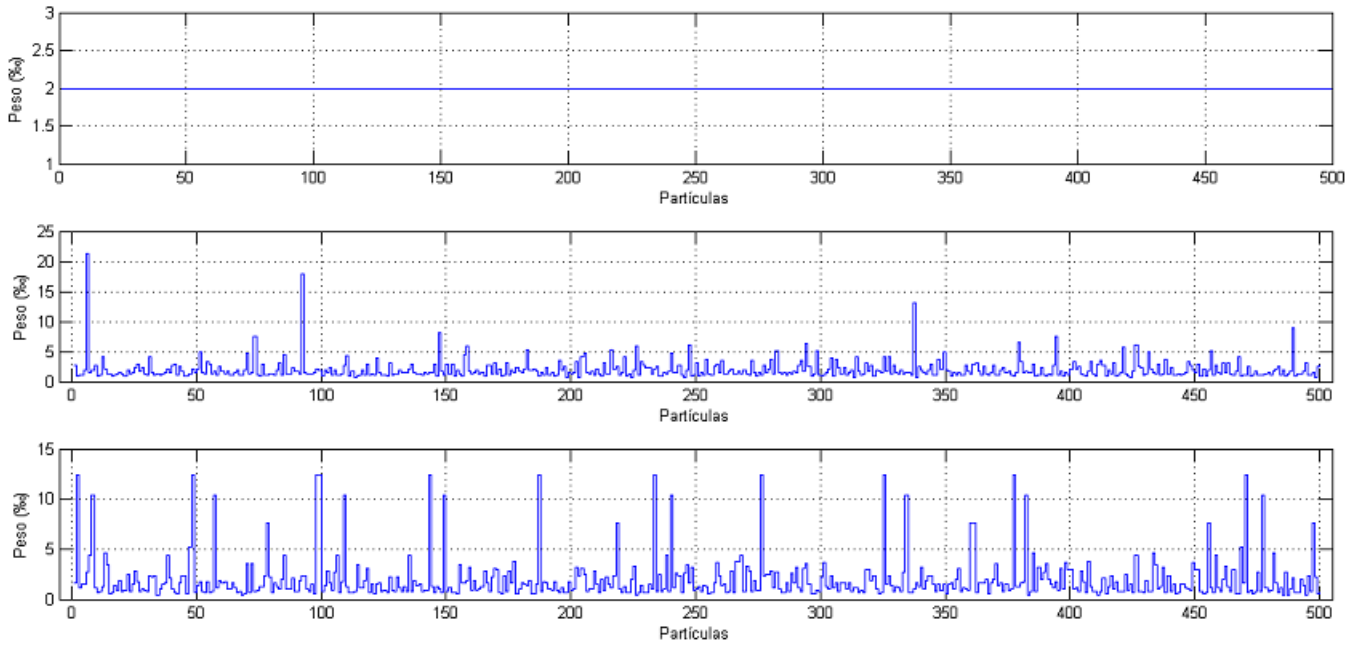


Figura 4-71. Pesos: a) Inicialización, b) Actualización, c) Resampling

A modo general, para comprobar la eficacia de las diferentes pruebas, se toma medida del error absoluto entre la media de las partículas y la trayectoria proporcionada. Además, para una referencia analítica se realiza la integral de este error a lo largo de todas las iteraciones (*ni*) de la forma:

$$e_T = \sum_{k=1}^{ni} e_k \cdot T \qquad (4.6)$$

4.8.1 PF 1: Selectividad en la media

Tras cada ciclo, el algoritmo muestra como resultado la media ponderada de las partículas según su peso, siendo ésta el vector de estado estimado. El objetivo de este ensayo es mostrar cómo mejorar la exactitud del filtro mediante la selectividad en esta media, es decir, excluir de ella a partículas menores de cierto peso proporcional a la de mayor peso.

Interesa decir que el criterio basado en el error, indicado anteriormente (ecuación 4.9), en estas simulaciones va a ser delimitado, de forma que sólo se contará el error a partir de la duodécima iteración. El objetivo de esta acción es intentar, al menos en la medida de lo posible, excluir de los resultados el error producido en la inicialización de las partículas, puesto que es totalmente aleatorio y no depende del criterio puesto a prueba en el subapartado.

4.8.1.1 Mínimo peso: 0%

En este caso, se escogen todas las partículas existentes sin aplicar ningún criterio. Se proporciona como patrón a seguir la trayectoria cuadrada (Figura 4-72), teniendo el modelo y los sensores una desviación de $v = 0{,}1\ m$ y $w_r = 0{,}5\ m$, respectivamente. La Figura 4-73 muestra el error a lo largo del recorrido.

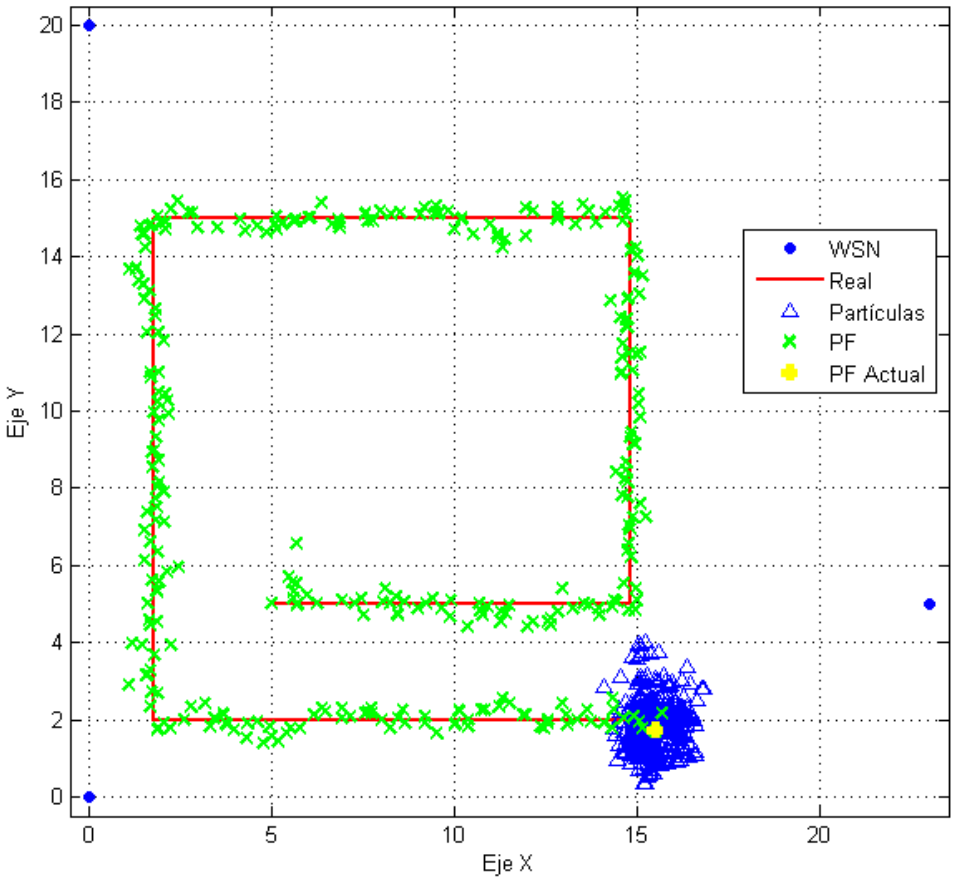


Figura 4-72. PF-1, Trayectoria, Selectividad desde el 0%

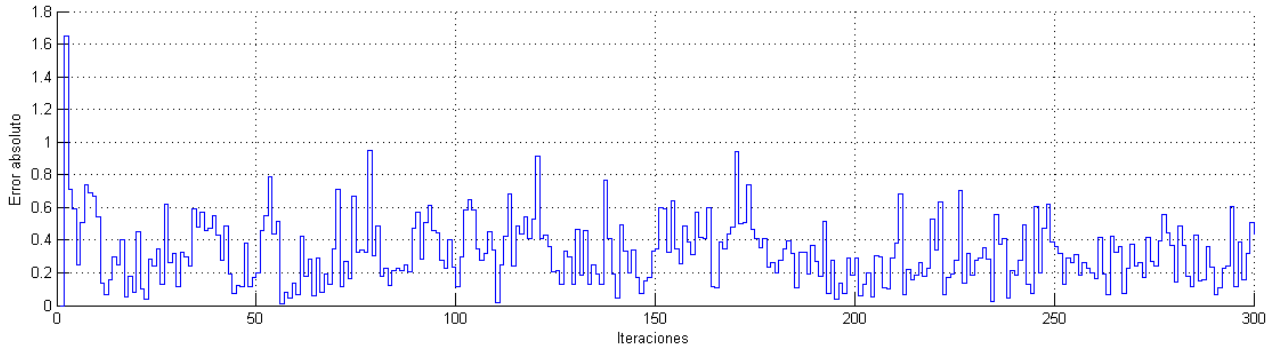


Figura 4-73. PF-1, Error, Selectividad desde el 0%

4.8.1.2 Mínimo peso: 80%

En este caso, al reducir la cantidad de partículas a considerar, el seguimiento pierde precisión como muestra la Figura 4-74, lo cual es corroborado al comparar la magnitud del error de la Figura 4-75 con la Figura 4-73.

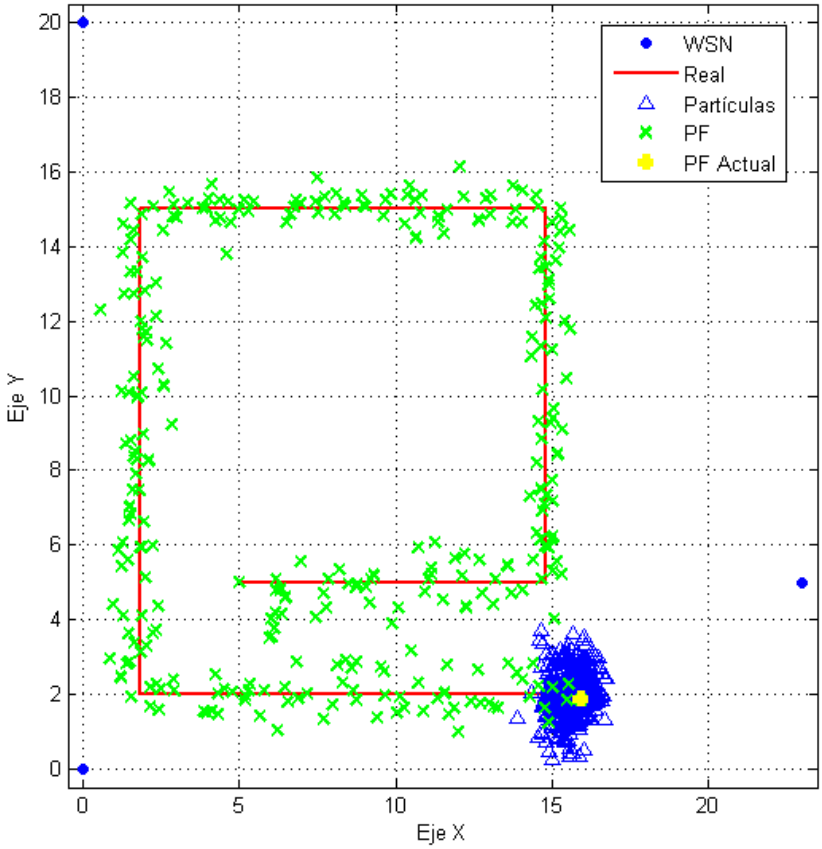


Figura 4-74. PF-1, Trayectoria, Selectividad desde el 80%

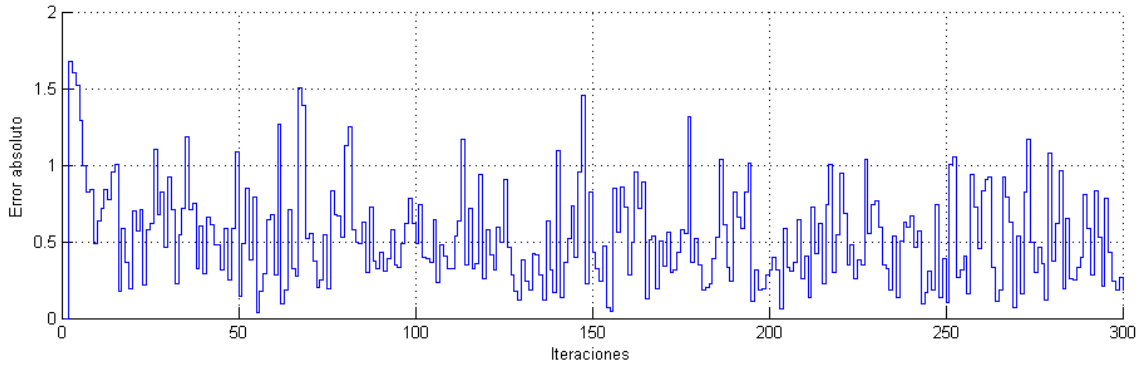


Figura 4-75. PF-1, Error, Selectividad desde el 80%

4.8.1.3 Mínimo peso: 60%

Las Figuras 4-76 y 4-77 muestran, respectivamente, el resultado de la simulación y el error de posición a lo largo de ésta.

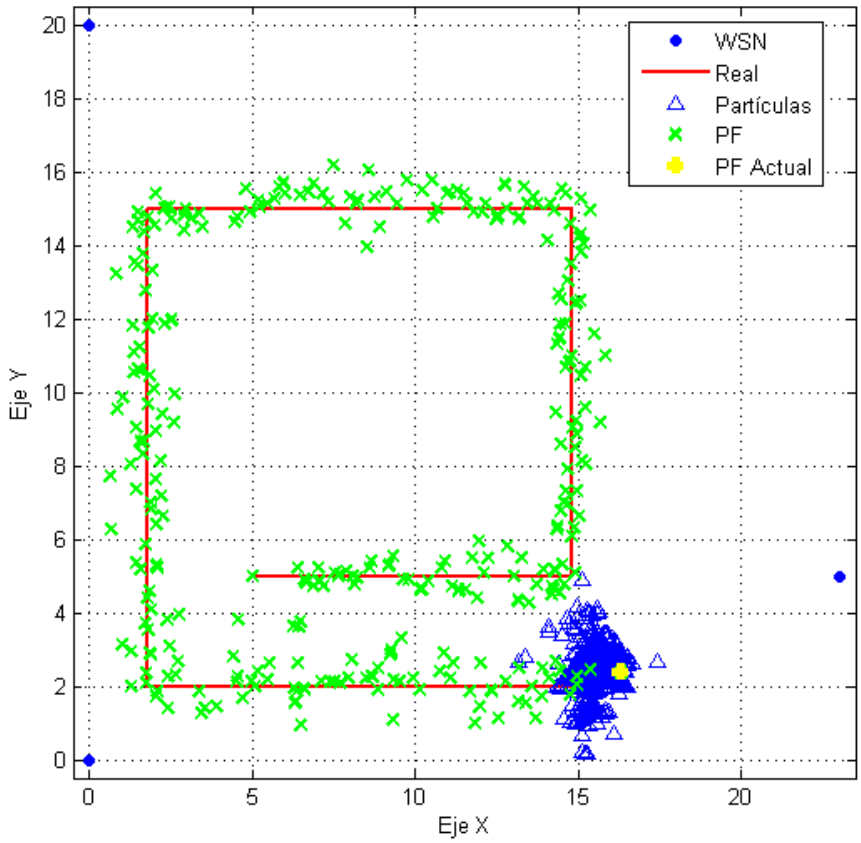


Figura 4-76. PF-1, Trayectoria, Selectividad desde el 60%

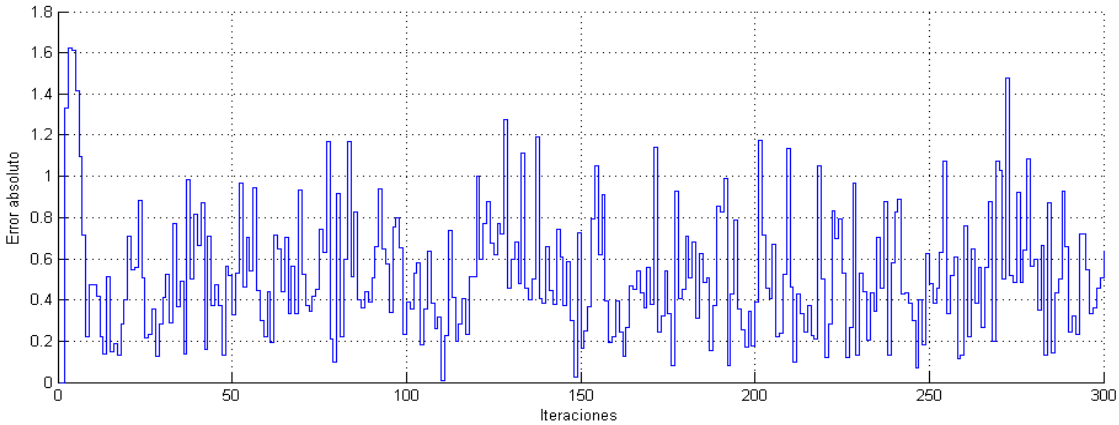


Figura 4-77. PF-1, Selectividad desde el 60%

4.8.1.4 Mínimo peso: 40%

Las Figuras 4-78 y 4-79 muestran, respectivamente, el resultado de la simulación y el error de posición a lo largo de ésta.

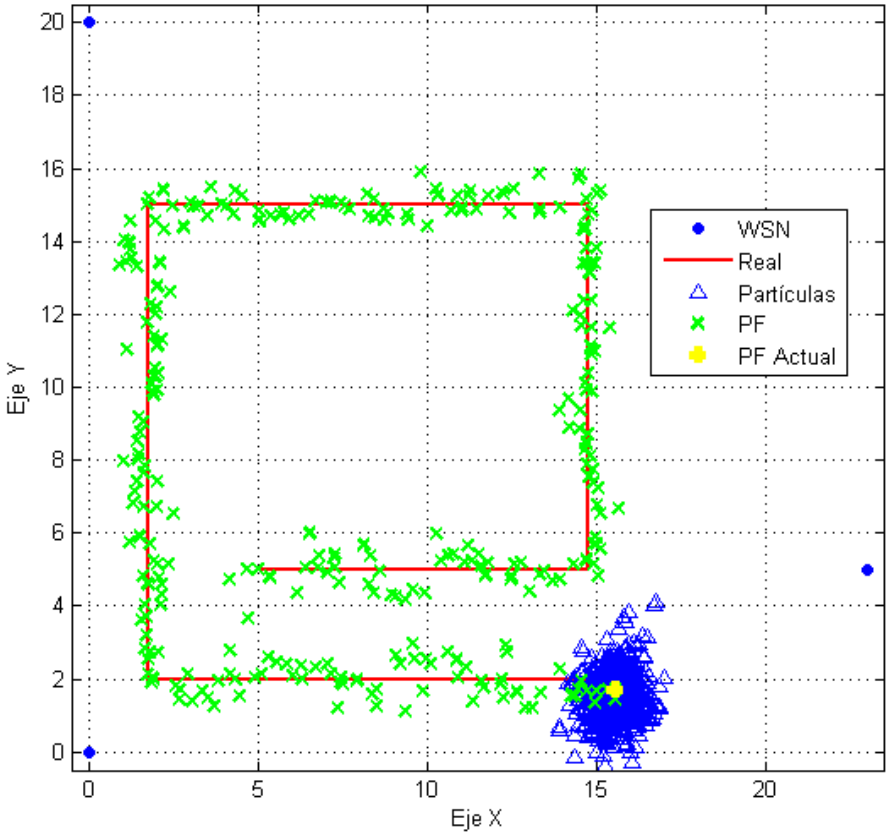


Figura 4-78. PF-1, Trayectoria, Selectividad desde el 40%

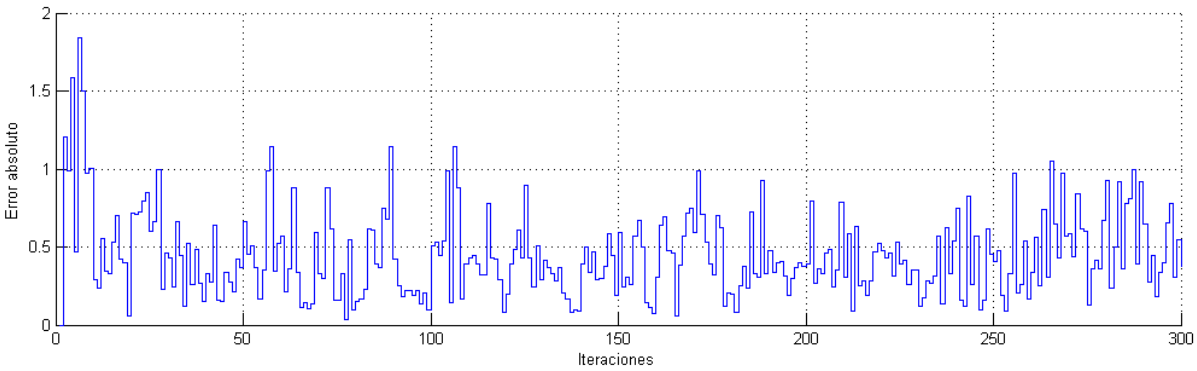


Figura 4-79. PF-1, Error, Selectividad desde el 40%

4.8.1.5 Mínimo peso: 20%

Las Figuras 4-80 y 4-81 muestran, respectivamente, el resultado de la simulación y el error de posición a lo largo de ésta. Cabe destacar que el error aumenta a medida que no se consideran la totalidad de las partículas, aunque, a partir de cierto umbral, no de forma desmedida.

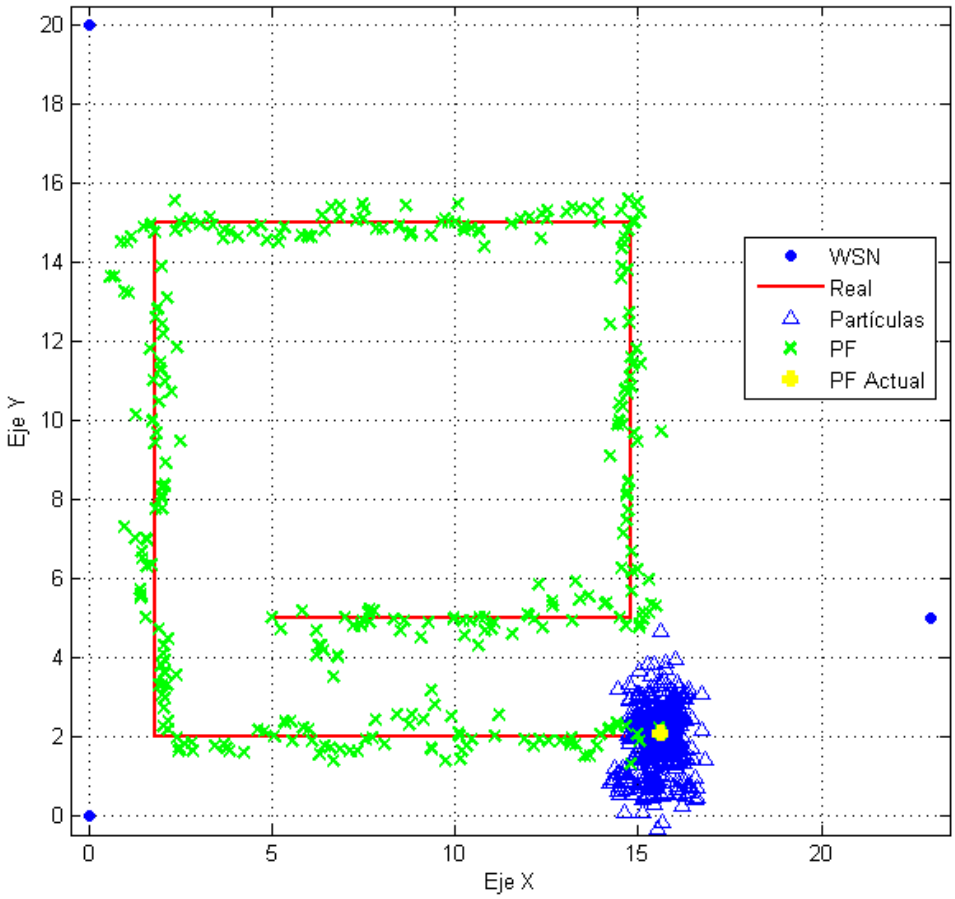


Figura 4-80. PF-1, Trayectoria, Selectividad desde el 20%

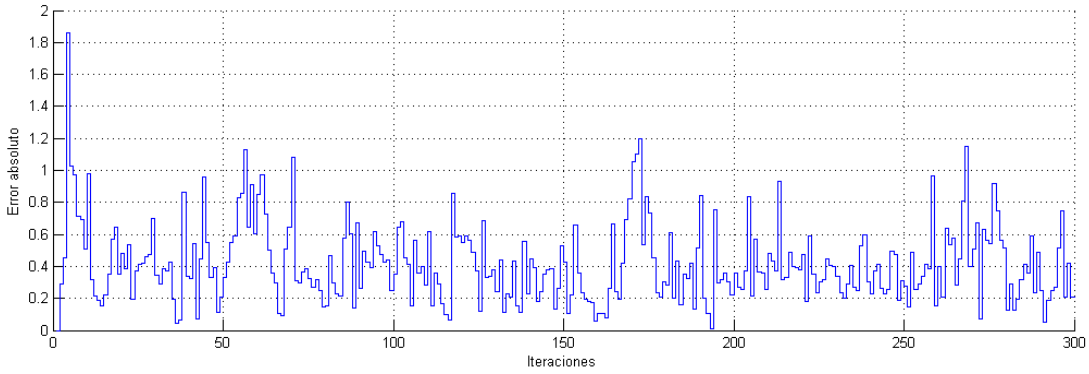


Figura 4-81. PF-1, Error, Selectividad desde el 20%

4.8.1.6 Comparativa

De la Tabla 4-4 se puede concluir que a mayor número de partículas se considere, menor error habrá en la media, contrastando de esta manera el Teorema Central del Límite. Sobre esta tabla hay que indicar que los errores pueden variar en un par de metros de una simulación a otra, dada la naturaleza aleatoria del remuestreo.

Tabla 4–4. Errores en PF-1

Selectividad	Error (m)
0%	9.5814
20%	11.6860
40%	12.3647
60%	14.6028
80%	14.7602

4.8.2 PF 2: Número de partículas

En esta prueba se realizan simulaciones con diferente cantidad de partículas, observando cómo es afectado el funcionamiento del filtro. Siguiendo las conclusiones sacadas en el ensayo anterior, a mayor número de partículas, mejor funcionamiento.

4.8.2.1 Nº Partículas: 50

La Figura 4-82 muestra la inicialización de las partículas.

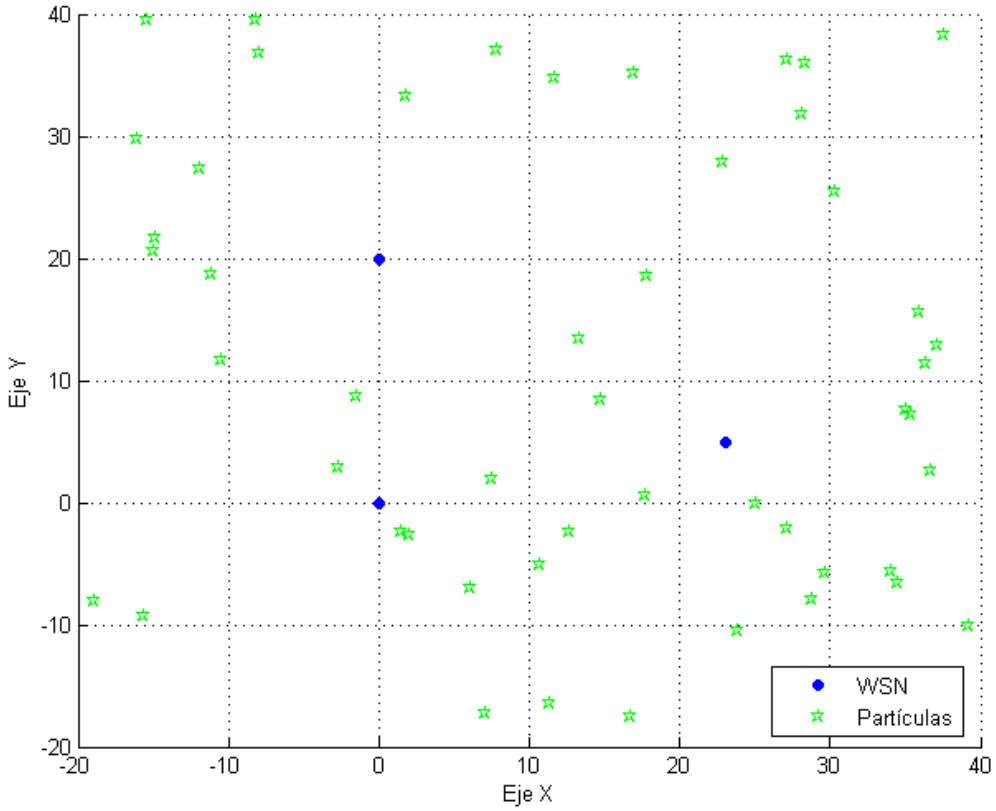


Figura 4-82. PF-2, Inicio, Uso de 50 partículas

En la Figura 4-83 puede comprobarse que la falta de suficientes muestras implica mayor dificultad de seguimiento a lo largo del recorrido y, sobre todo, en la inicialización como muestra la Figura 4-84, pues la escasez de partículas implica más posibilidades de no estar cerca de la posición del robot, cosa necesaria para que en el resampling las partículas se aproximen rápido.

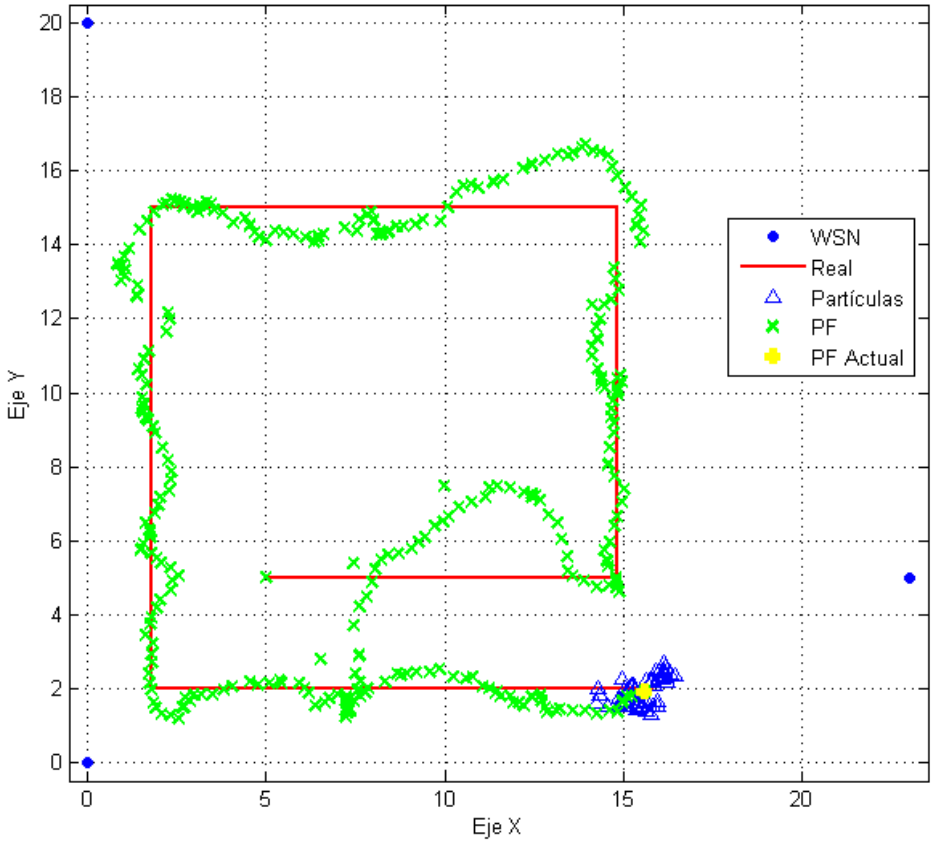


Figura 4-83. PF-2, Trayectoria, Uso de 50 partículas

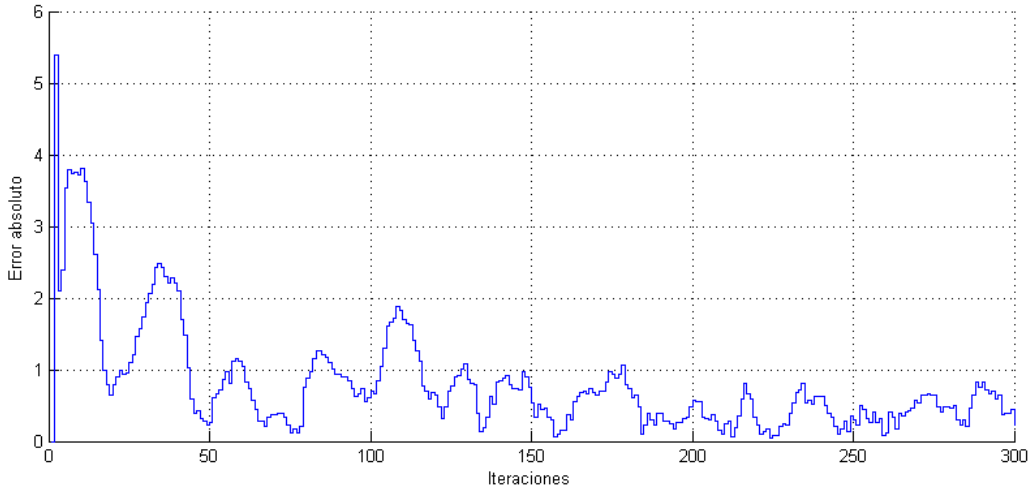


Figura 4-84. PF-2, Error, Uso de 50 partículas

4.8.2.2 Nº Partículas: 100

Inicialización aleatoria y resultado de la simulación en las Figuras 4-85 y 4-86, respectivamente.

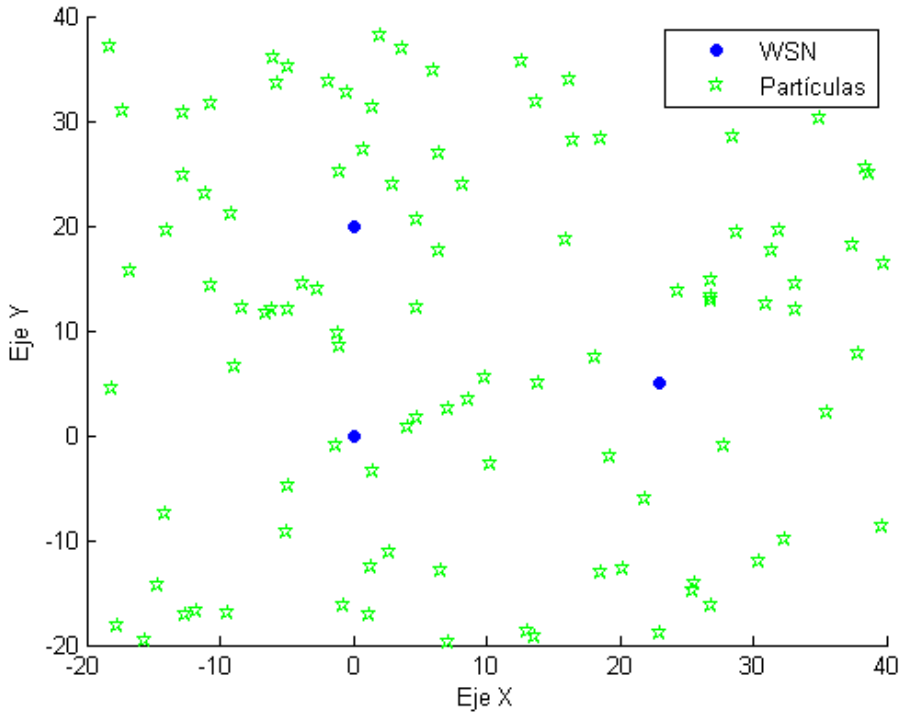


Figura 4-85. PF-2, Inicio, Uso de 100 partículas

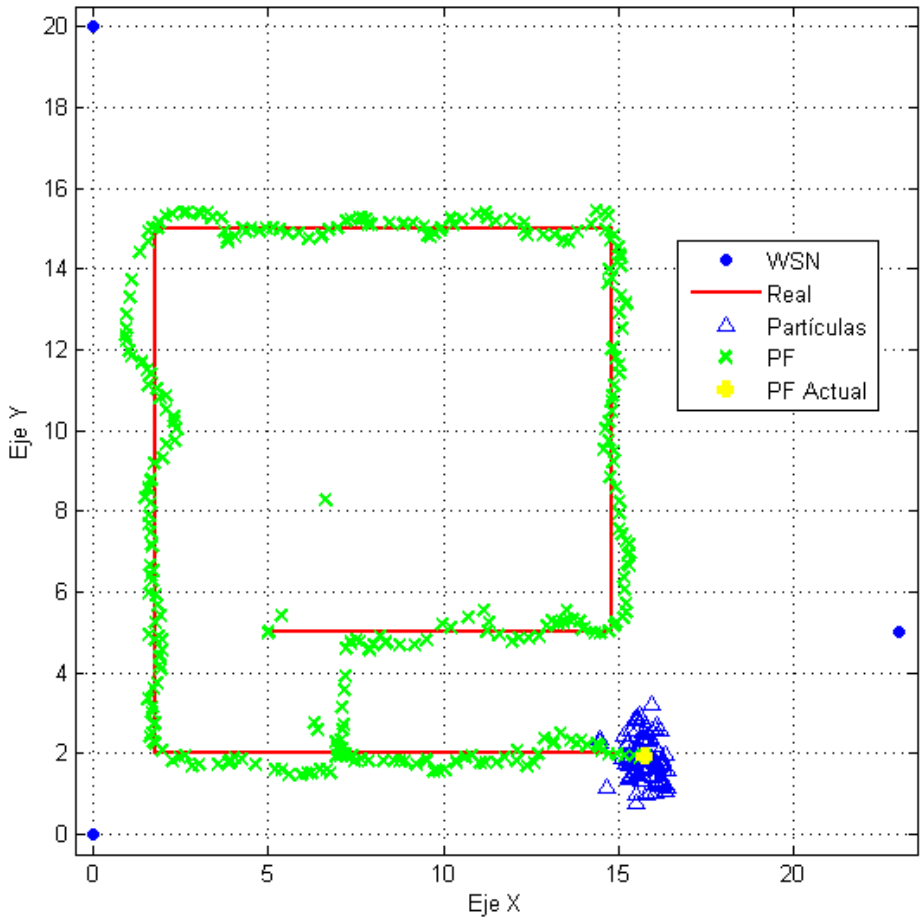


Figura 4-86. PF-2, Trayectoria, Uso de 100 partículas

Como se observa en la Figura 4-87, aumentando el número de partículas mejora la precisión del filtro respecto la anterior simulación.

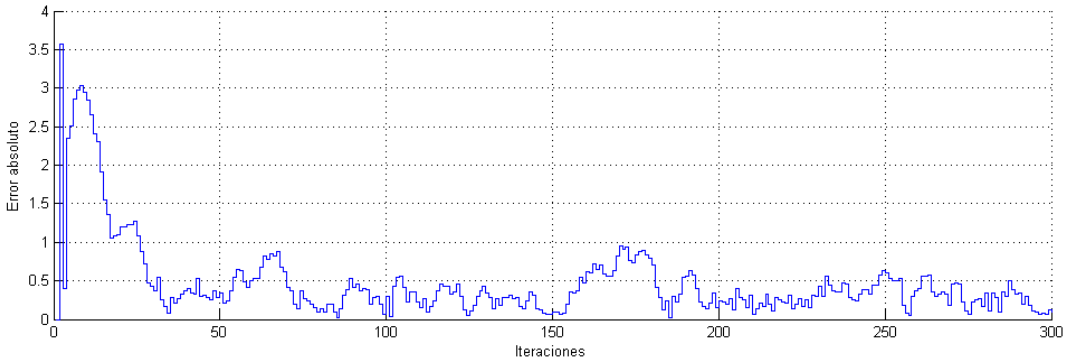


Figura 4-87. PF-2, Error, Uso de 100 partículas

4.8.2.3 Nº Partículas: 200

Inicialización aleatoria y resultado de la simulación en las Figuras 4-88 y 4-89, respectivamente, mientras que la Figura 4-90 muestra el error a lo largo de ésta.

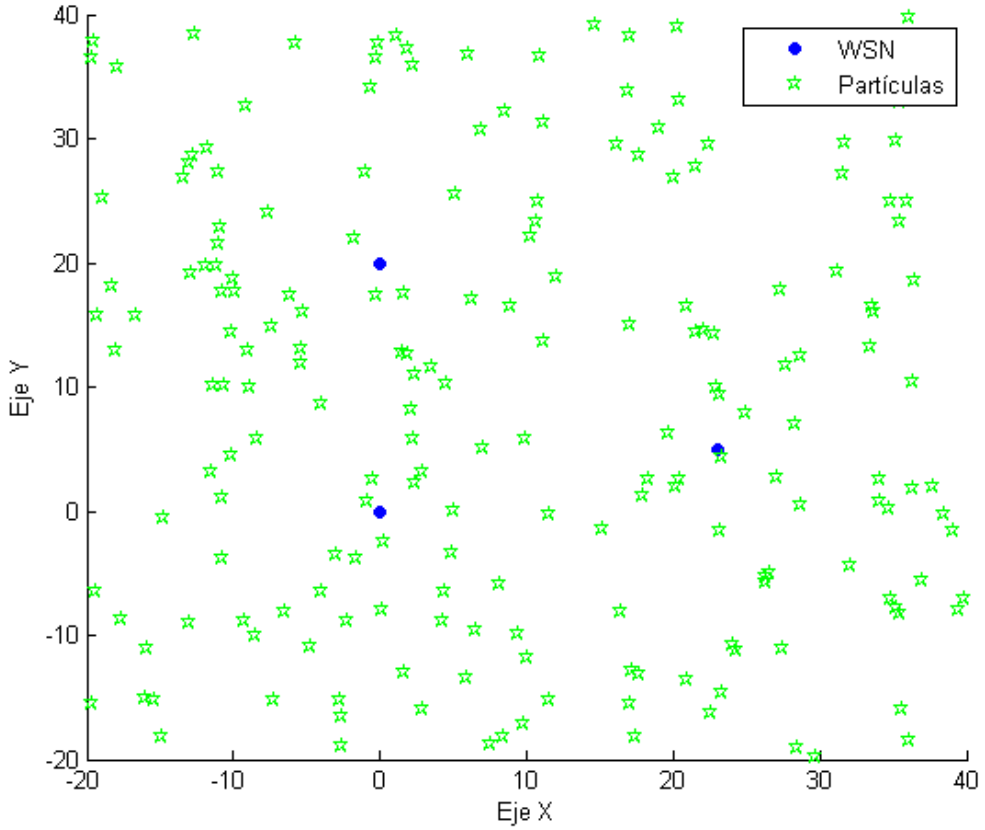


Figura 4-88. PF-2, Inicio, Uso de 200 partículas

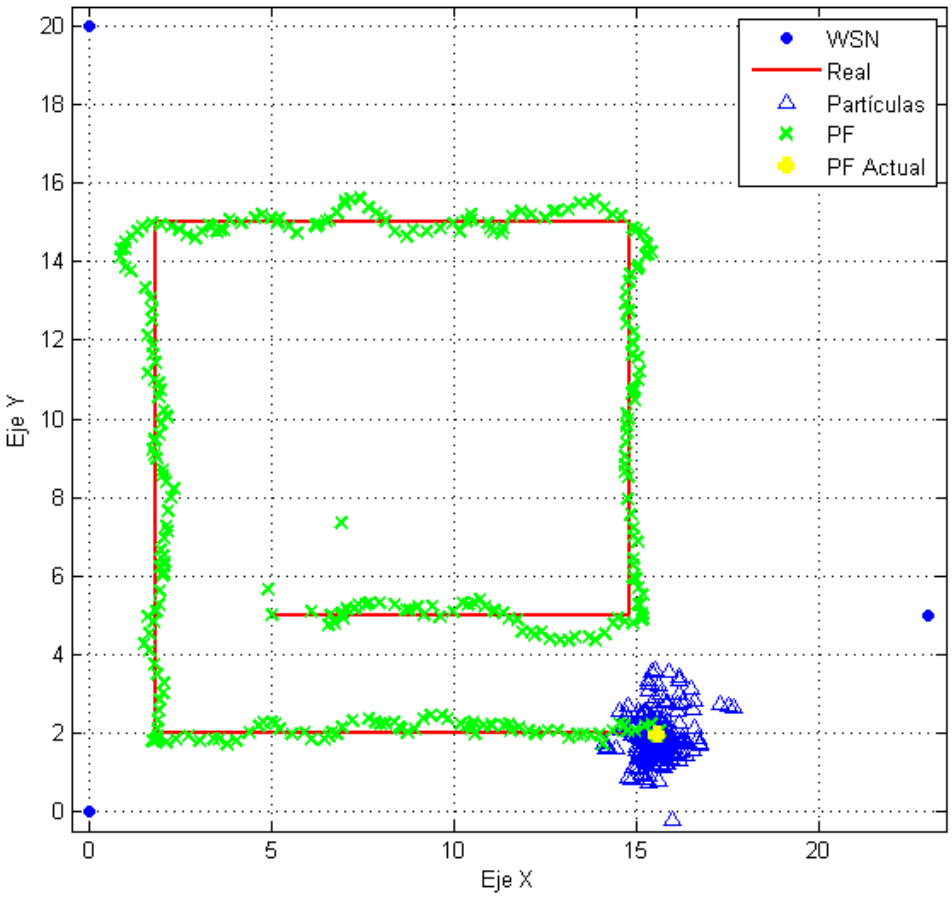


Figura 4-89. PF-2, Trayectoria, Uso de 200 partículas

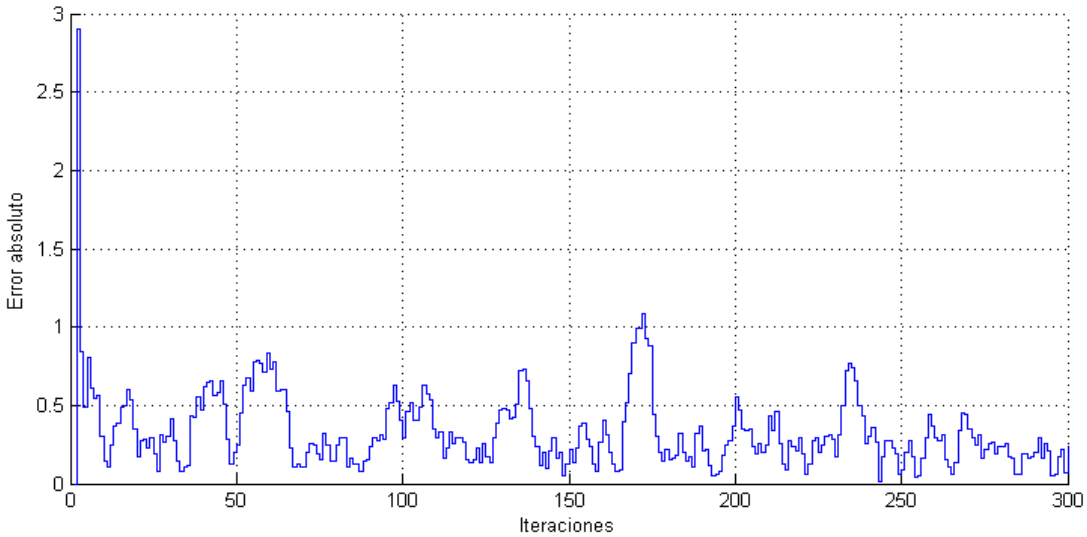


Figura 4-90. PF-2, Error, Uso de 200 partículas

4.8.2.4 Nº Partículas: 500

Inicialización aleatoria y resultado de la simulación en las Figuras 4-91 y 4-92, respectivamente, mientras que la Figura 4-93 muestra el error a lo largo de ésta.

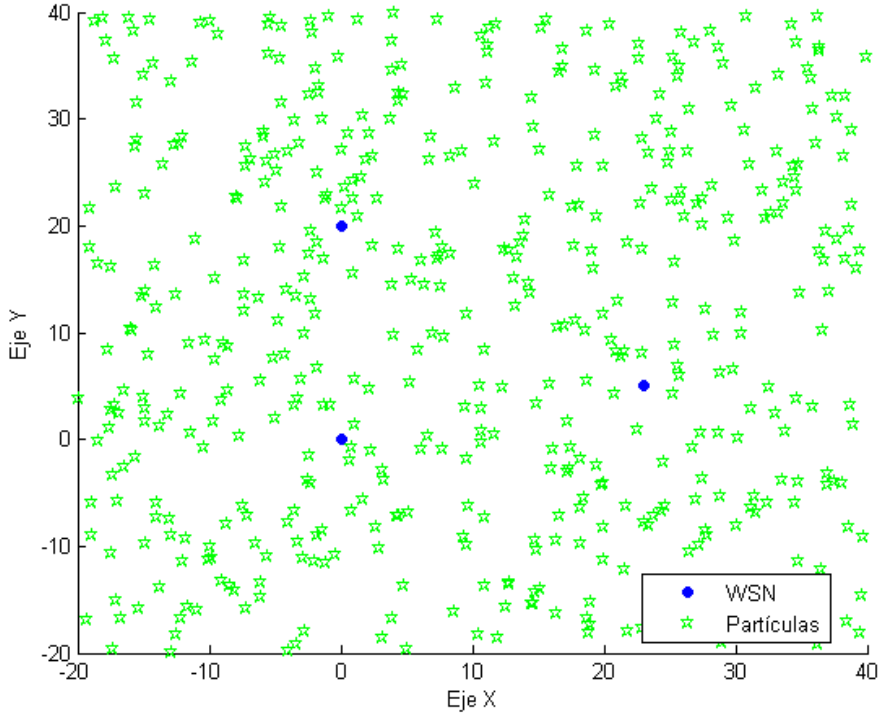


Figura 4-91. PF-2, Inicio, Uso de 500 partículas

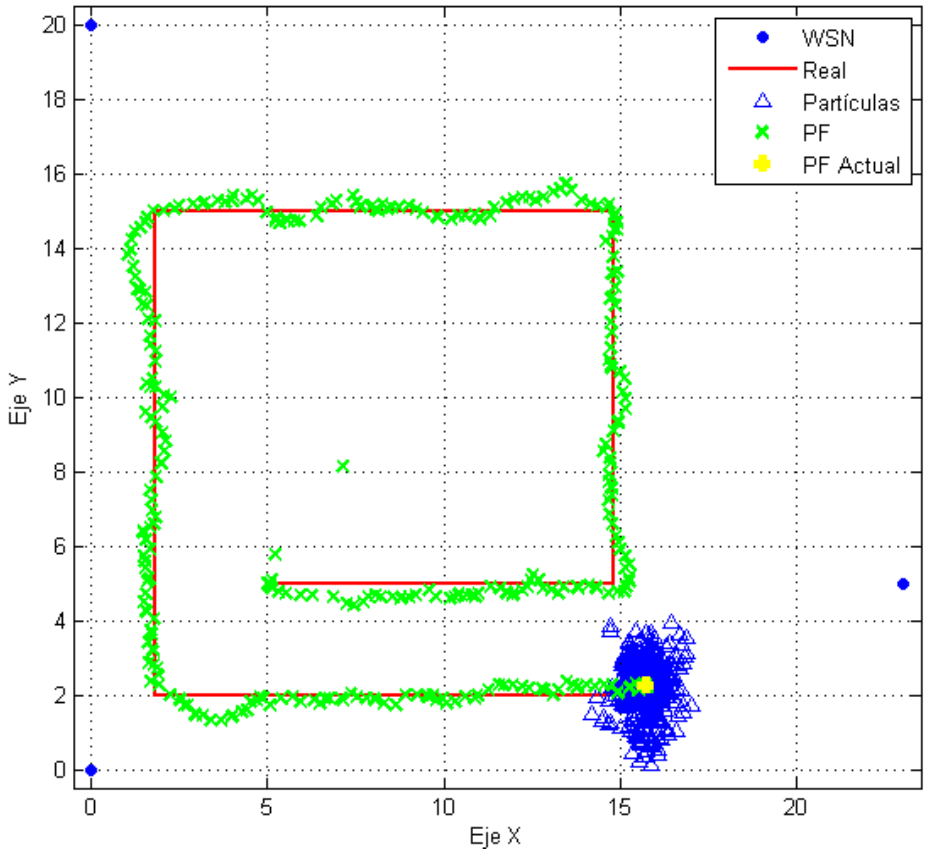


Figura 4-92. PF-2, Trayectoria, Uso de 500 partículas

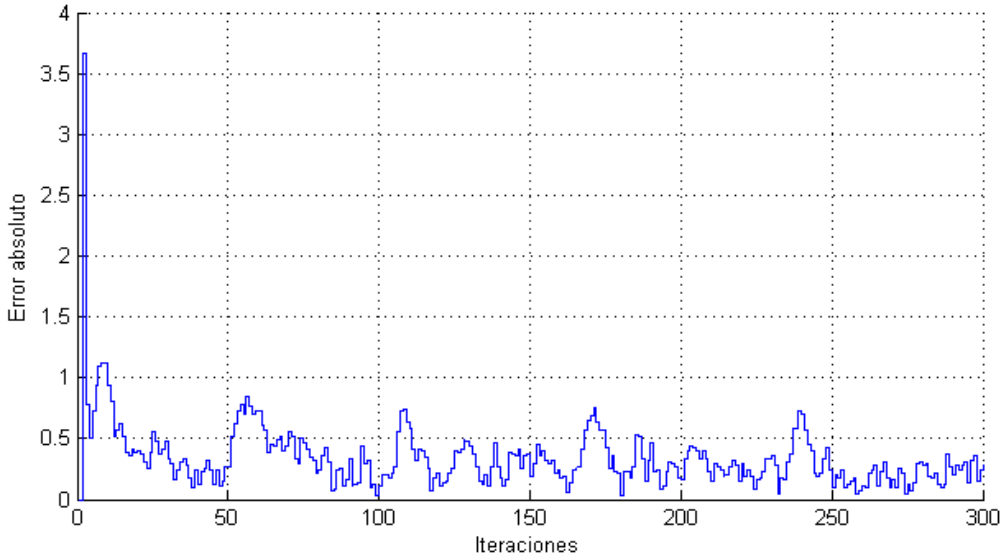


Figura 4-93. PF-2, Error, Uso de 500 partículas

4.8.2.5 Nº Partículas: 1000

Inicialización aleatoria y resultado de la simulación en las Figuras 4-94 y 4-95, respectivamente, mientras que la Figura 4-96 muestra el error a lo largo de ésta.

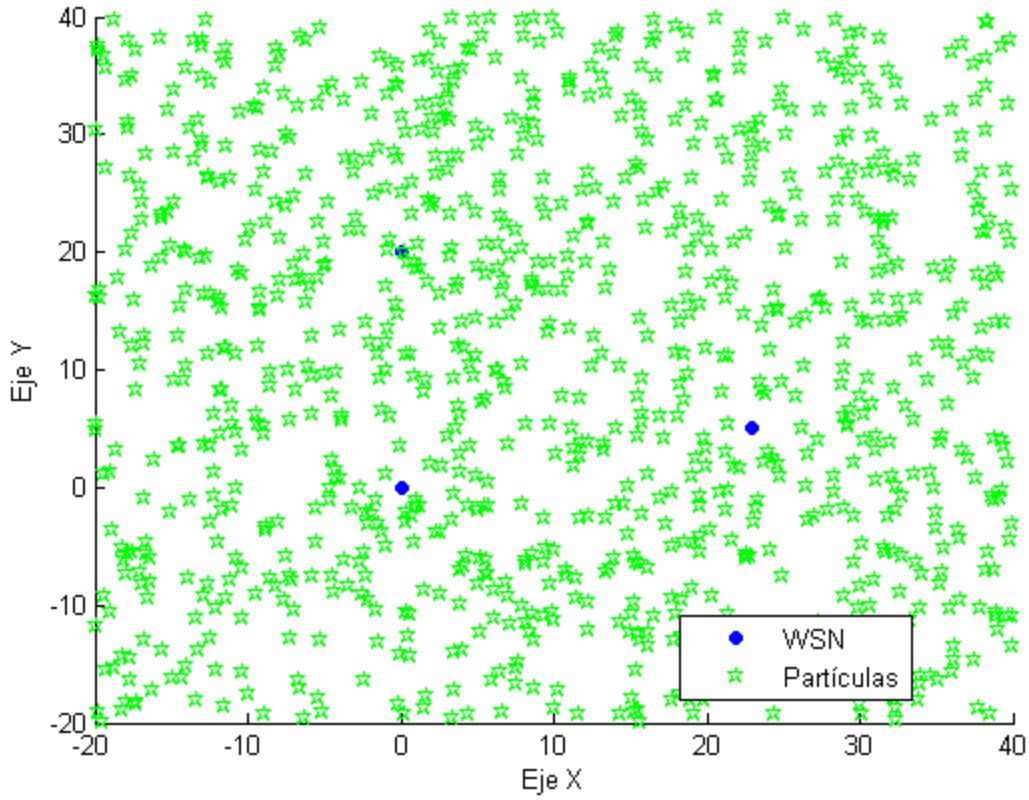


Figura 4-94. PF-2, Inicio, Uso de 500 partículas

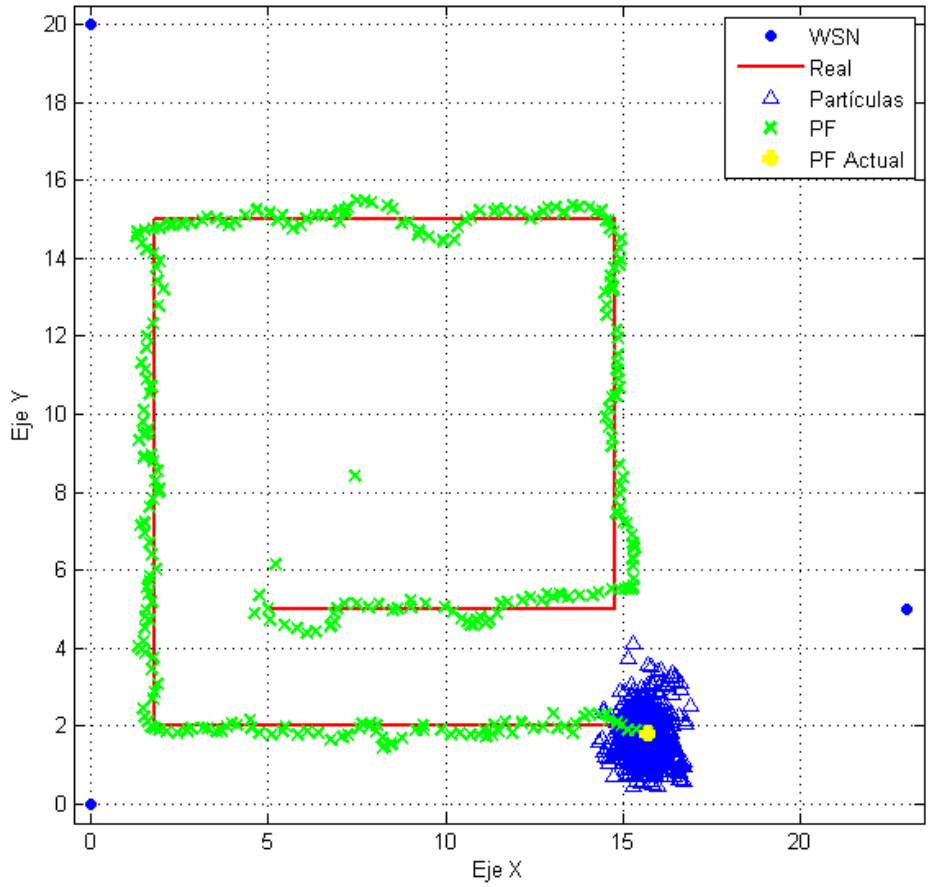


Figura 4-95. PF-2, Trayectoria, Uso de 500 partículas

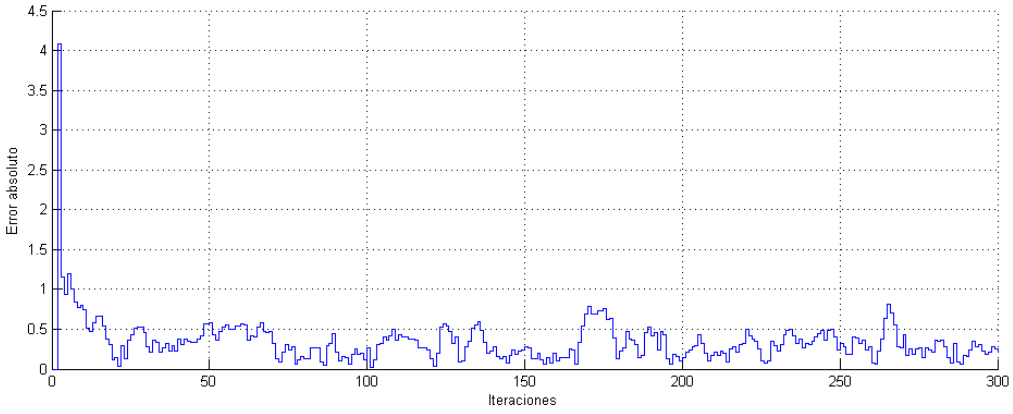


Figura 4-96. PF-2, Error, Uso de 500 partículas

4.8.2.6 Comparativa

Se comprueba que efectivamente mejora, aunque a partir de cierto punto, el error difícilmente disminuye. Esto se debe a que siempre hay una componente errática debida, bien al resampling, bien a las medidas con error aleatorio.

Tabla 4–5. Errores en PF-2

Partículas	**Error (*m*)**
50	20.4091
100	11.5453
200	9.1244
500	8.9395
1000	9.0589

4.8.3 PF 3: Frecuencia de resampling

En esta ocasión se procede a experimentar probando diferentes valores de la variable que controla cada cuánto se ejecuta este proceso. El remuestreo en el filtro de partículas viene a ser el equivalente a la fase de actualización del KF por lo que es vital, aunque se reduce su uso para relajar la carga computacional.

4.8.3.1 Cada ciclo

Las Figuras 4-97 y 4-98, respectivamente, muestran el resultado y el error de la simulación.

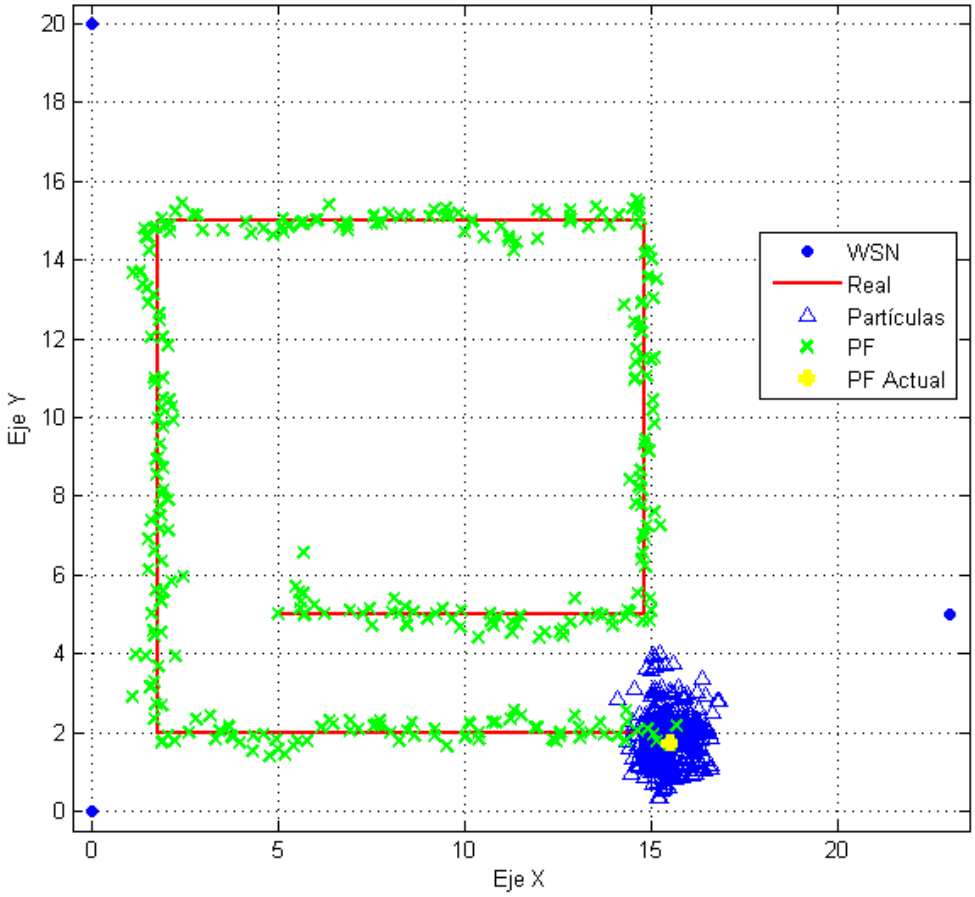


Figura 4-97. PF-3, Trayectoria, Resampling cada ciclo

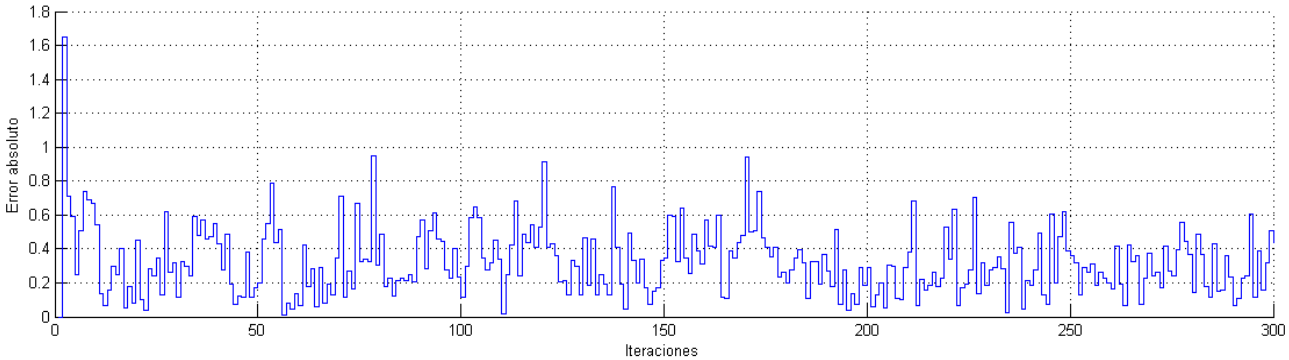


Figura 4-98. PF-3, Error, Resampling cada ciclo

4.8.3.2 Cada dos ciclos

Las Figuras 4-99 y 4-100, respectivamente, muestran el resultado y el error de la simulación.

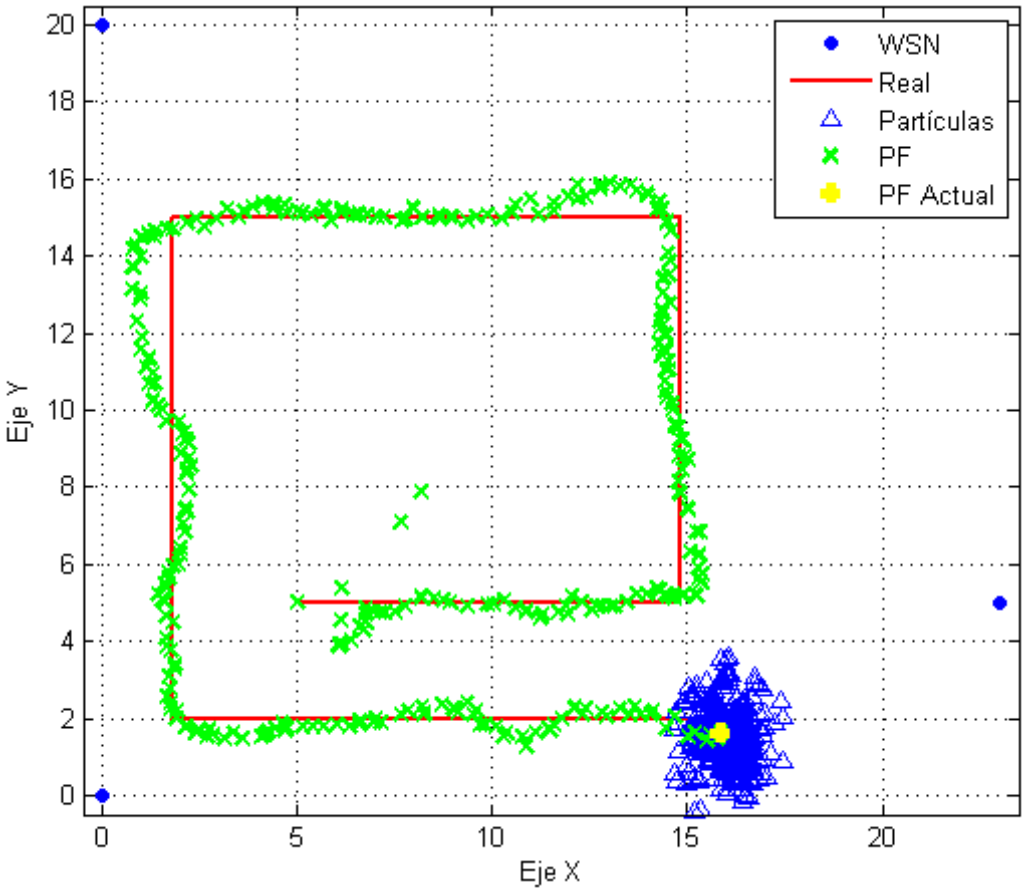


Figura 4-99. PF-3, Trayectoria, Resampling cada dos ciclos

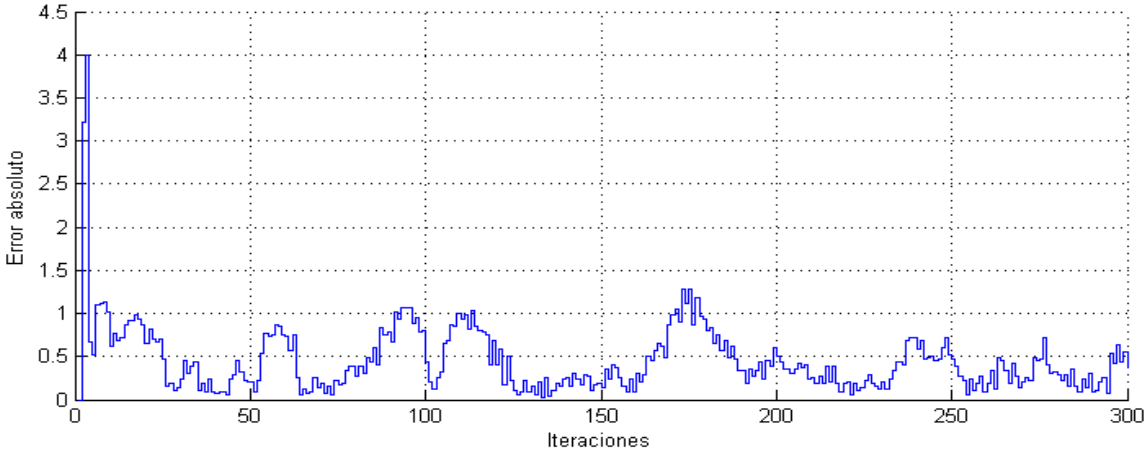


Figura 4-100. PF-3, Error, Resampling cada dos ciclos

4.8.3.3 Cada tres ciclos

Las Figuras 4-101 y 4-102, respectivamente, muestran el resultado y el error de la simulación.

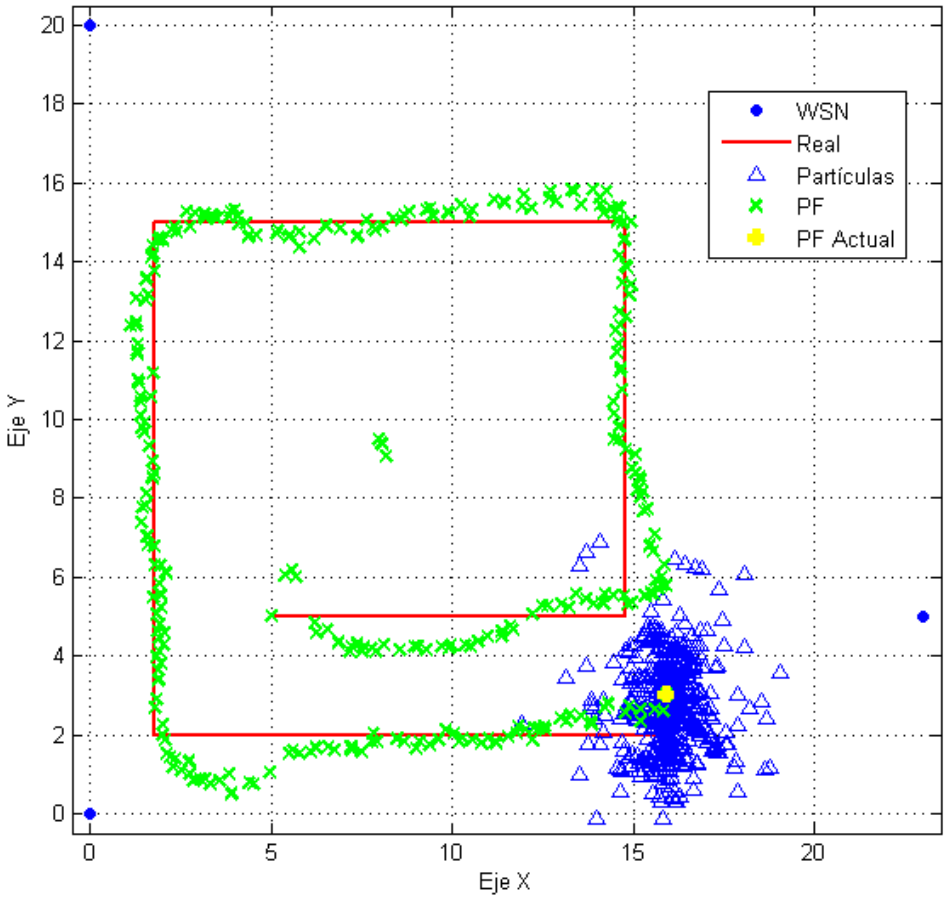


Figura 4-101. PF-3, Trayectoria, Resampling cada tres ciclos

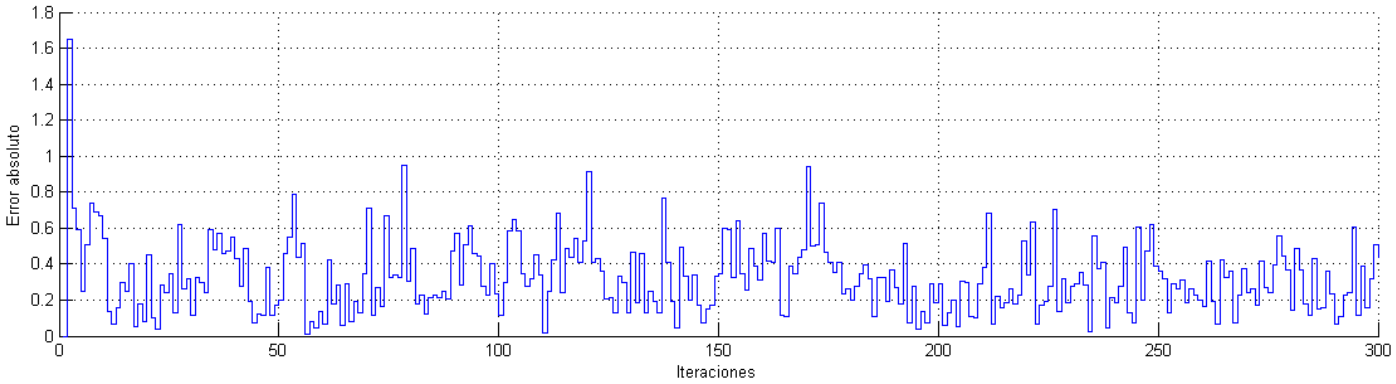


Figura 4-102. PF-3, Error, Resampling cada tres ciclos

4.8.3.4 Cada cinco ciclos

Se puede comprobar en la Figura 4-103 que permitir a las partículas confiar en el modelo implementado, con una menor intervención de la información externa, resulta en una disminución de la precisión como muestra la Figura 4-104.

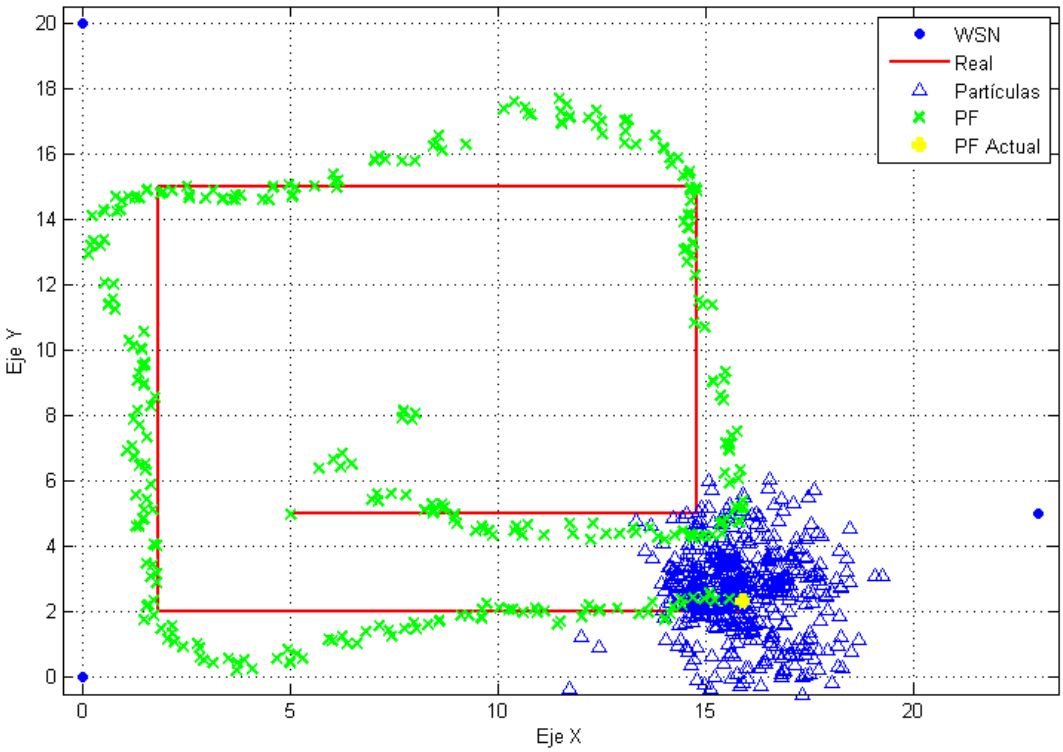


Figura 4-103. PF-3, Trayectoria, Resampling cada cinco ciclos

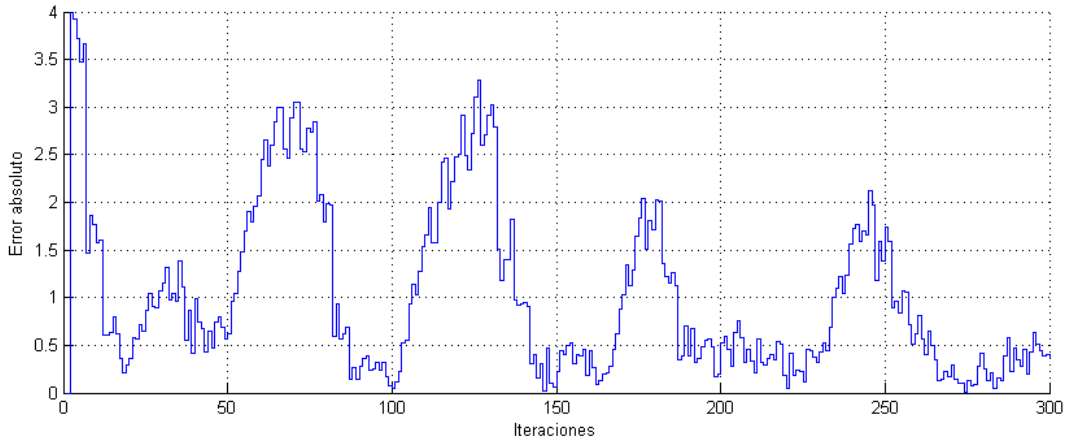


Figura 4-104. PF-3, Error, Resampling cada cinco ciclos

4.8.3.5 Comparativa

Queda probado que disminuir la frecuencia de ejecución del proceso de reduce la exactitud del seguimiento, dado que las muestras cambiarían exclusivamente de la fase de predicción sin ningún tipo de corrección.

Tabla 4–6. Errores en PF-3

T ejecución	Error (m)
1	9.5814
2	12.1695
3	15.4532
5	27.9674

4.8.4 PF 4: ROS

Al igual que en apartados similares, la configuración en ROS sigue el mismo esquema. Lo único interesante de indicar es que usa 500 partículas, realiza en cada ciclo resampling y no excluye ninguna muestra en el resampling. Al igual que en ocasiones similares, y como muestran las Figuras 4-105 y 4-106, se atraviesa una fase de funcionamiento normal, pérdida de comunicación y otra fuera del rango del sensor.

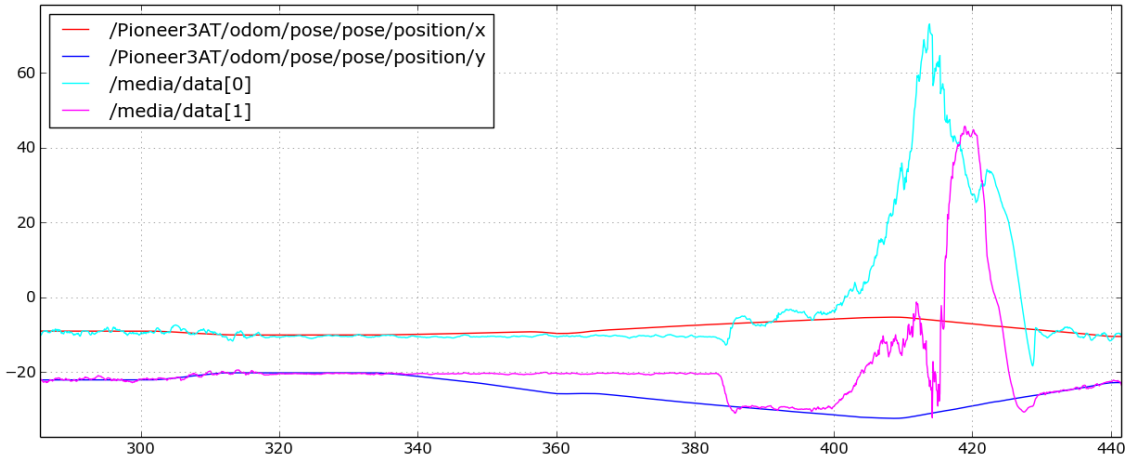


Figura 4-105. ROS, Seguimiento con PF

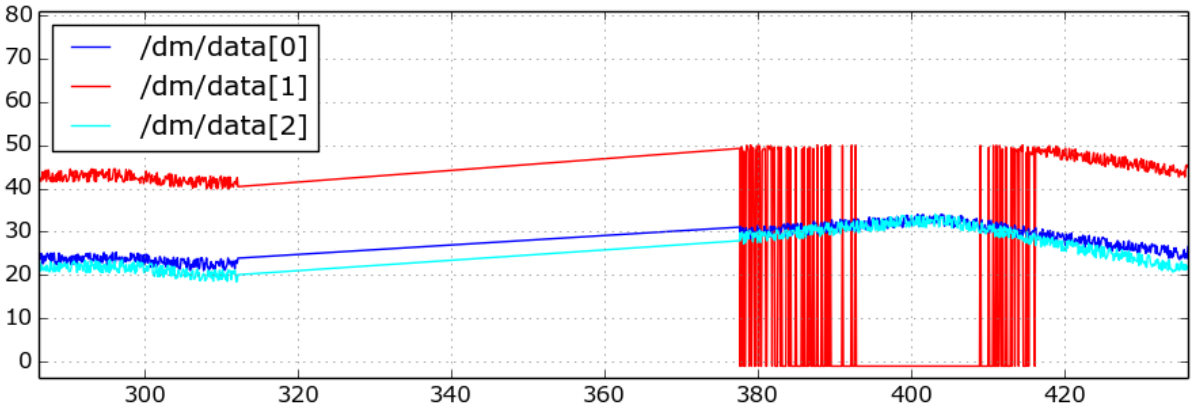


Figura 4-106. ROS, Medidas recibidas por el PF

En la Figura 4-107 viene el error a lo largo de la simulación entre la media y la posición real, el cual aparece ampliado en la Figura 4-108 durante el correcto funcionamiento del filtro. Se observa que los errores en este tramo son aproximadamente iguales a los obtenidos al implementar el filtro en Matlab, ligeramente mayores.

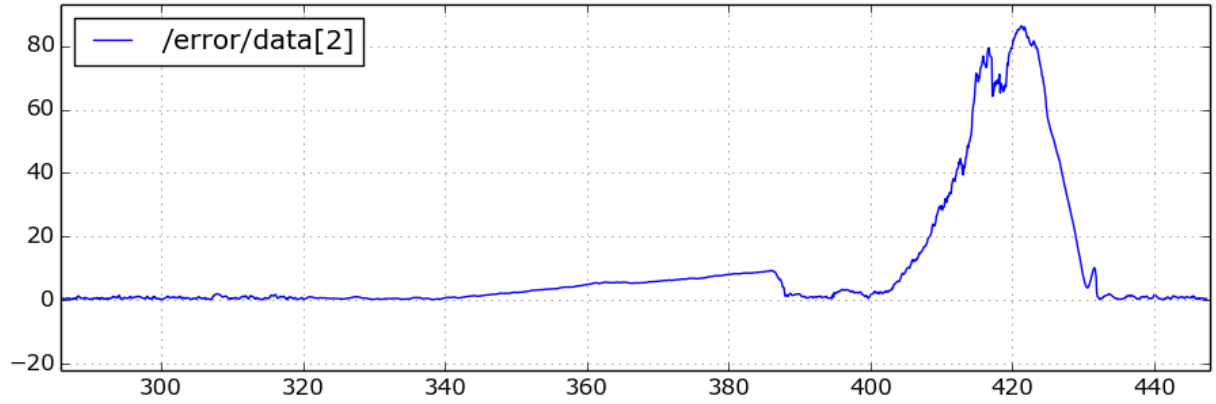


Figura 4-107. ROS, Error en posición con PF

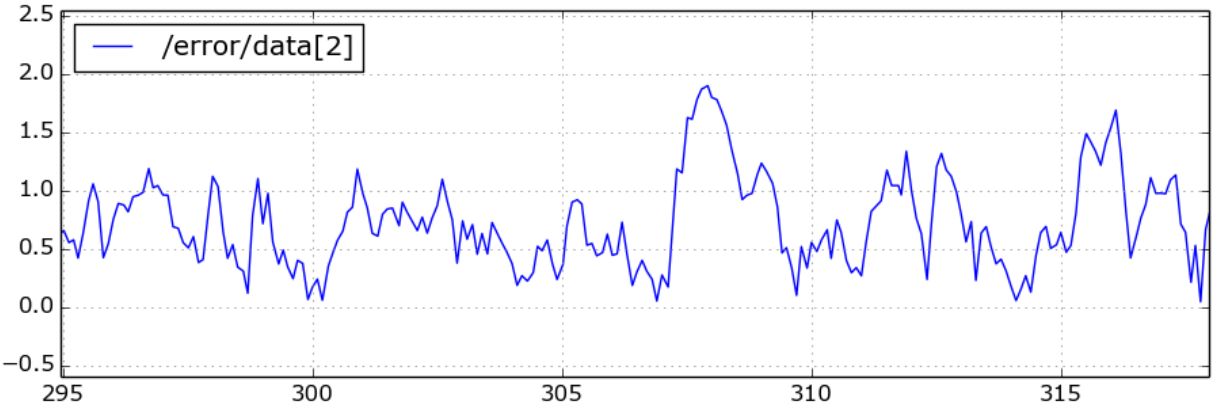


Figura 4-108. ROS, Error en posición ampliado

4.9 Comparaciones

En este apartado se recopilarán juntas las principales simulaciones, anteriormente realizadas, con objeto de destacar las similitudes y diferencias entre todos los filtros. La excepción viene dada de los filtros de Información puesto que sus correspondientes apartados se destinan a demostrar la inequívoca semejanza entre estos y sus duales.

4.9.1 C-1: KF/IF vs EKF/EIF

En esta primera comparación, se contrastan los resultados entre el KF y su versión extendida.

Para poder hacer una comparación más justa, se aumentará el ruido de los sensores de rango a $w_r = 1\,m$ mientras que la desviación del GPS seguirá con $w_g = 3\,m$. El modelo se mantendrá con una incertidumbre igual a $v = 0{,}1\,m$.

La trayectoria comenzará en las coordenadas $(-5, 0)$, misma posición en la que se inicializará la media.

4.9.1.1 KF con GPS bueno

En la Figura 4-109 se muestra el resultado de la simulación anteriormente descrita. En este caso, que el GPS posea una fiabilidad menor que los sensores de rango resulta en un peor seguimiento, como prueba la Figura 4-110.

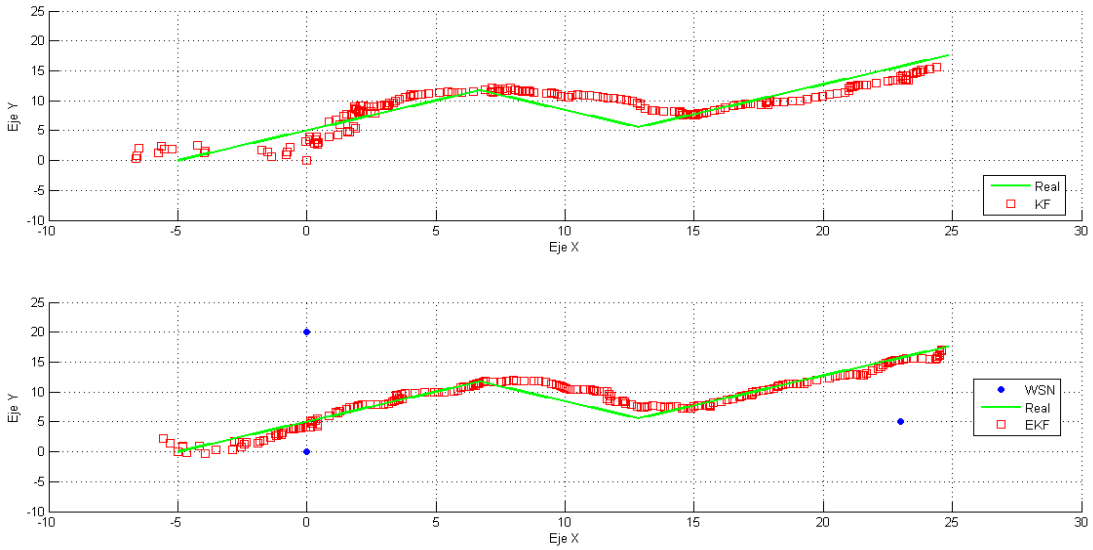


Figura 4-109. Zigzag, trayectoria con: a) KF, b) EKF

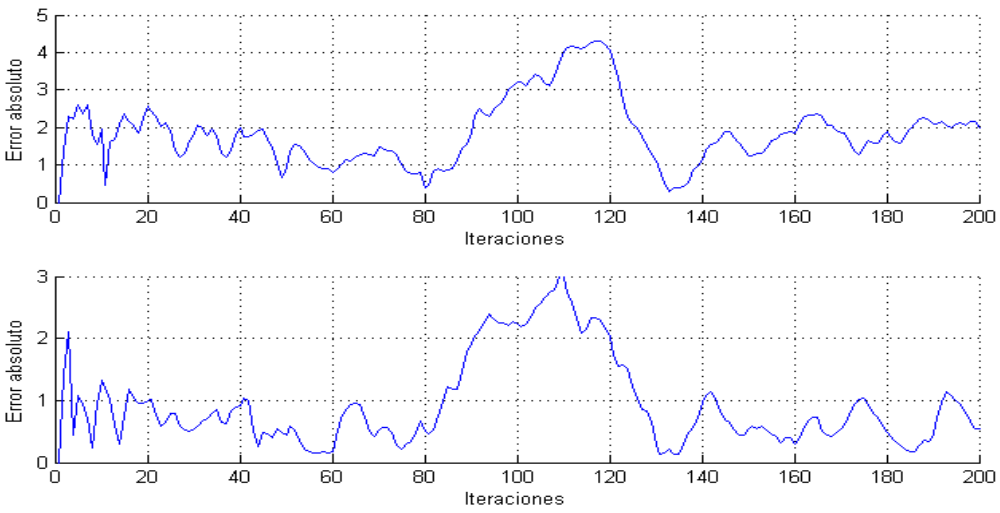


Figura 4-110. Zigzag, error con: a) KF, b) EKF

4.9.1.2 KF con GPS muy bueno

Ahora se procede a igualar la desviación del GPS con la de los sensores de rango, para intentar compensar un poco más la situación, dado que los sensores de rango son más fiables y numerosos.

LaFigura 4-111 muestra el resultado de la simulación. La actuación del KF ha mejorado al poder confiar más en las medidas del GPS frente al modelo rectilíneo, siendo incluso menores que el EKF en ocasiones, como muestra la Figura 4-112.

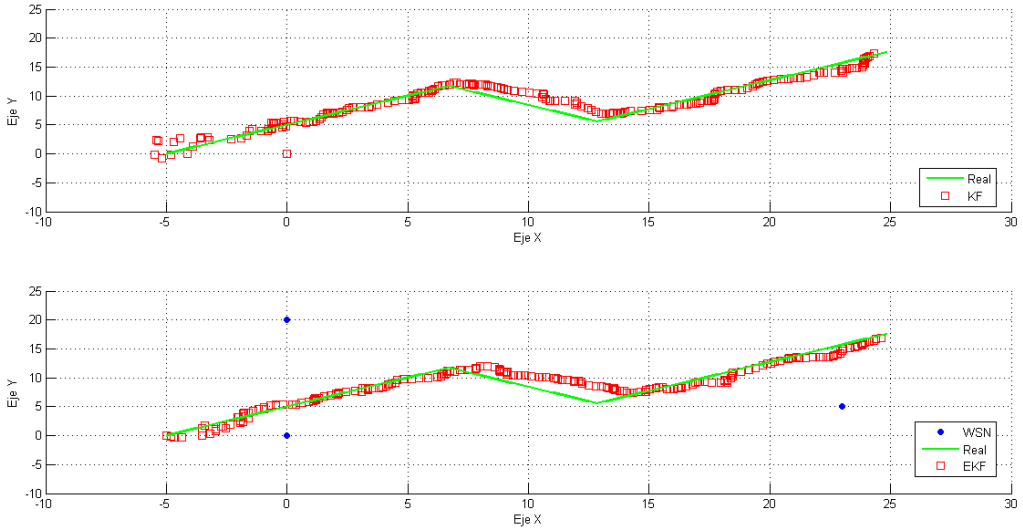


Figura 4-111. Zigzag, trayectoria con: a) KF, b) EKF

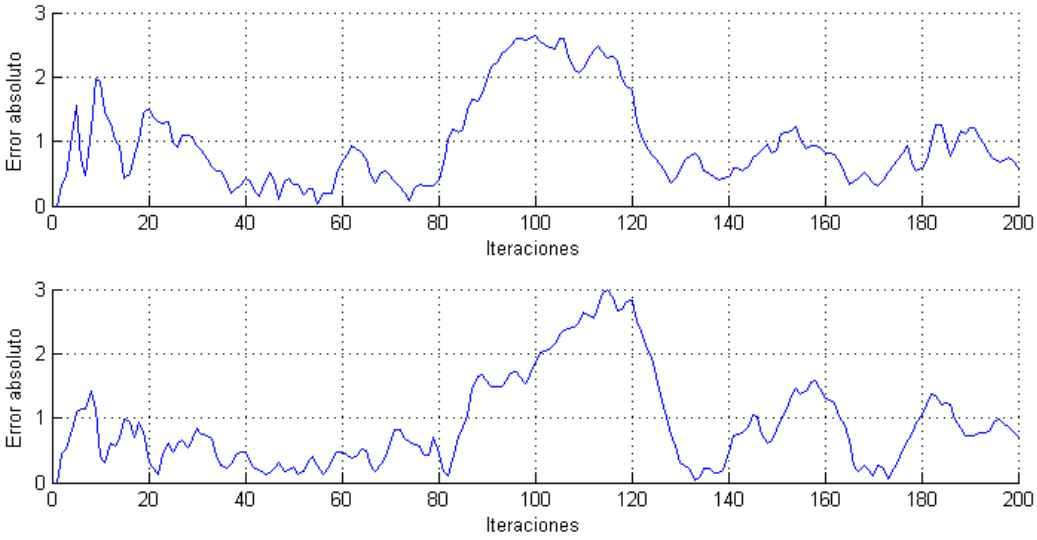


Figura 4-112. Zigzag, error con: a) KF, b) EKF

4.9.2 C-2: EKF/EIF vs PF

4.9.2.1 WSN de buena calidad

Dado que ahora ambos algoritmos usan exclusivamente medidas de rango, sí se puede comparar sin necesidad de hacer ajustes en la desviación de los sensores recuperando $w_r = 0{,}5\ m$.

La Figura 4-113 muestra el resultado a lo largo de la trayectoria de ambos filtros. El error a lo largo de ésta se expone en la Figura 4-114 con la particularidad de que se muestra a partir de la iteración 4, con el fin de facilitar la comparación dado que el PF en la inicialización tiene errores ampliamente mayores como refleja la Figura 4-115.

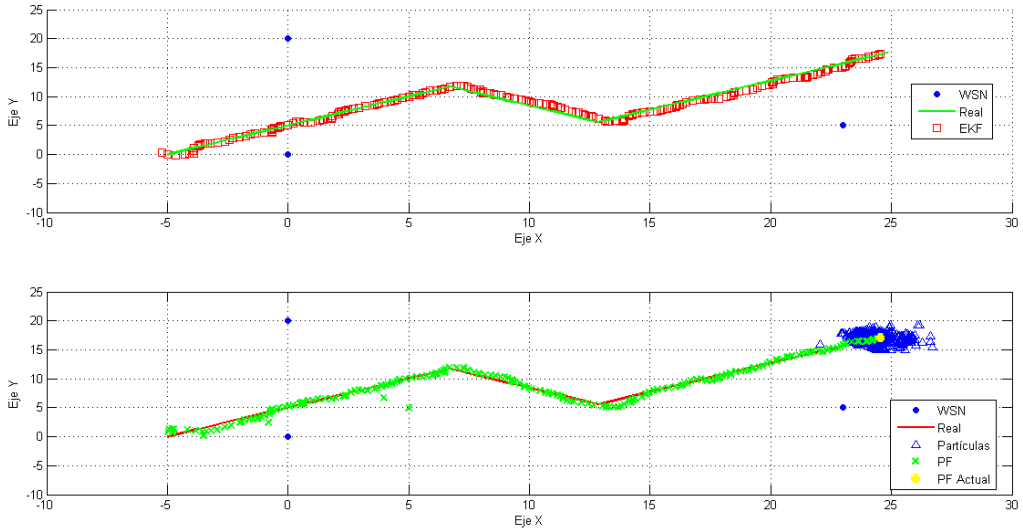


Figura 4-113. Zigzag, trayectoria con: a) EKF, b) PF

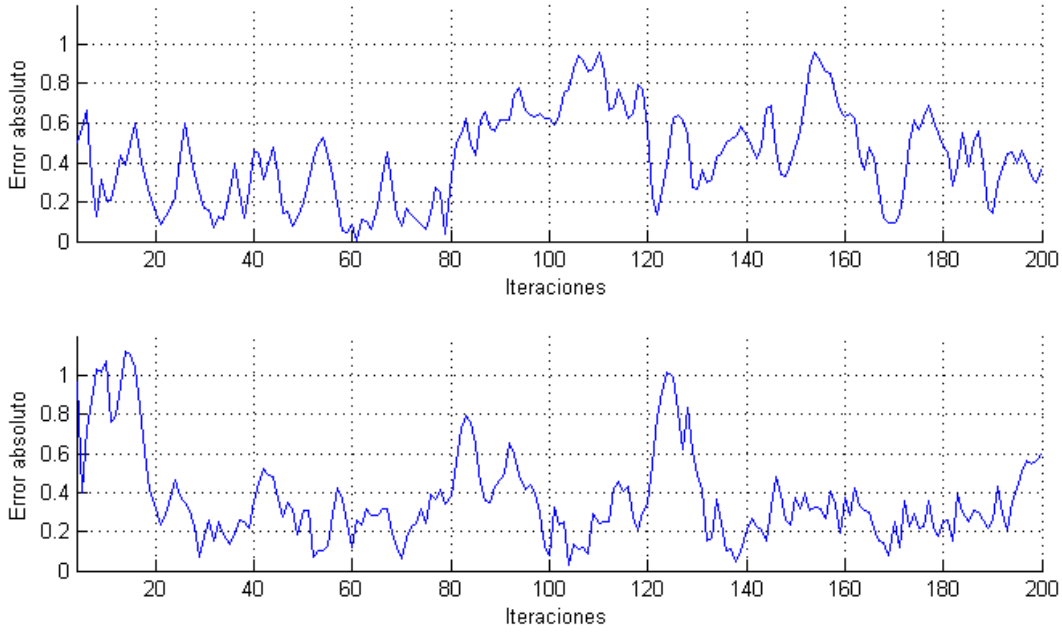


Figura 4-114. Zigzag, error con: a) EKF, b) PF

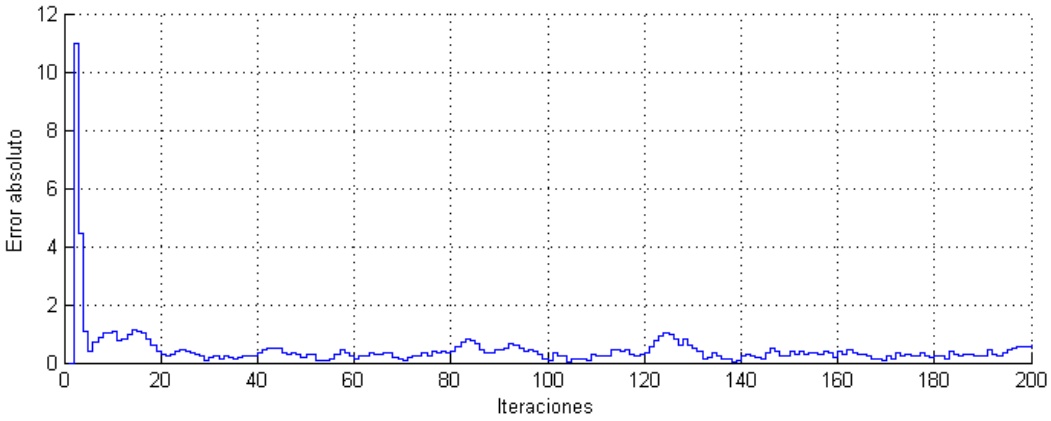


Figura 4-115. Error global en PF

4.9.2.2 WSN de peor calidad

Ahora para poder juzgar mejor entre la calidad de ambos filtros, se procede a experimentar con unos sensores de menor calidad, $w_r = 1{,}5\ m$.

Los resultados y el error se muestran en las Figuras 4-116 y 4-117. Al igual que antes se esconde el error inicial del PF para poder tener mayor precisión visual a la hora de comparar.

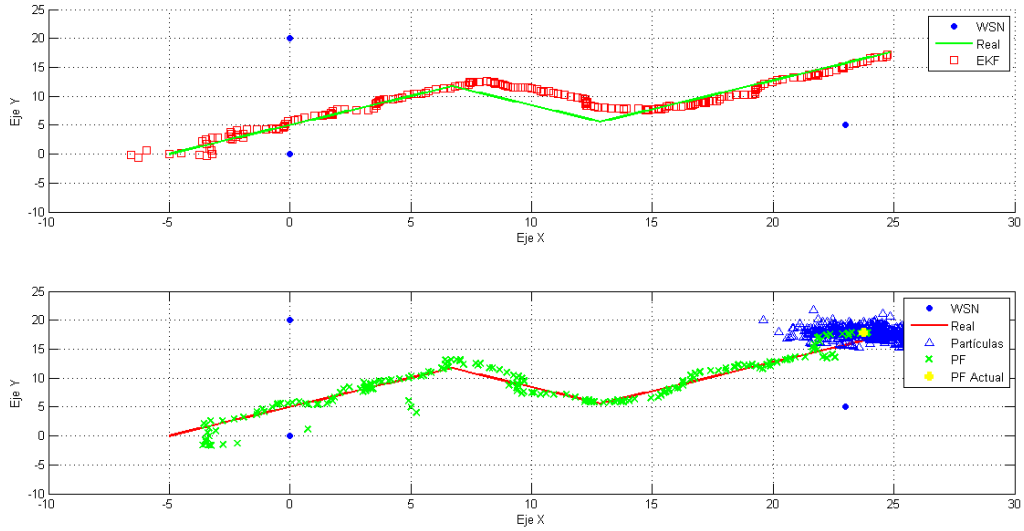


Figura 4-116. Zigzag, trayectoria con: a) EKF, b) PF

En este caso, es más fácil observar que, al reducir la fiabilidad en los sensores, ambos filtros pierden precisión pero el EKF es perturbado mucho más por el modelo a la hora de hacer cambios de dirección lo que se muestra a mitad de la Figura 4-117. Por otra parte, el FP sufre de errores aún mayores al comienzo, pues es más difícil situarse en la trayectoria con precisión.

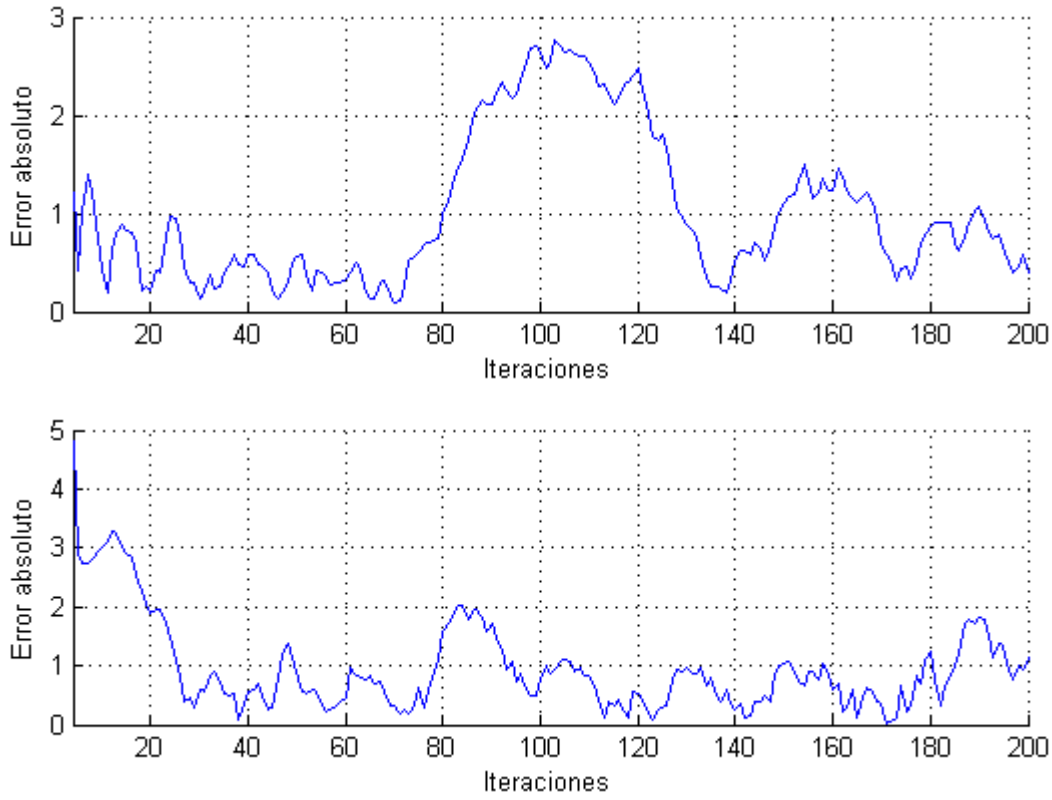


Figura 4-117. Zigzag, error con: a) EKF, b) PF

4.10 Conclusiones

Respecto a la funcionalidad del software empleado, reafirmar que ROS es de gran utilidad a la hora de manejar los numerosos elementos de un robot pero en el campo de representar resultados se ve ampliamente superado por Matlab.

Vuelta a los filtros, se ha demostrado con claras evidencias la exactitud de resultados entre el KF y el IF, así como en el caso de las versiones extendidas. Además, se ha comprobado que el EKF tiene un comportamiento similar al KF, alcanzando, el primero, menor error en los experimentos por la baja desviación de los sensores de rango, principalmente. Como ventaja, decir que el EKF puede combinar tanto el GPS como la red WSN, entre otros, cosa de la que es incapaz el KF.

Tras comentar los filtros paramétricos, el filtro de partículas demuestra ser tan capaz como el EKF con la desventaja de la dureza computacional que supone implementarlo. Hay que tomar una solución de compromiso entre la carga computacional y la eficacia del filtro (nº de partículas). Comparando errores, el PF puede ser tan preciso como el EKF si se ejecuta con un mínimo de prestaciones (suficientes muestras, constante remuestreo), teniendo el mayor pico de error en la inicialización puesto que, a diferencia del EKF, parte de muestras aleatorias y no de una media pre-escogida. También hay que reconocer que el PF sigue un movimiento algo más errático, pues, tras el remuestreo, las partículas son regeneradas con cierta aleatoriedad, lo que se refleja en la media. Este filtro presenta grandes capacidades de mejora mediante pequeñas consideraciones dentro del algoritmo.

5 CONCLUSIÓN

Nunca confíes en un ordenador que no puedas lanzar por una ventana.

- Steve Wozniak -

A lo largo de todo este trabajo, se ha revisado con detenimiento el estado del arte de los filtros paramétricos, además de uno de las más importantes de la familia de los no paramétricos. Por lo comprobado, son de una utilidad excepcional, siempre y cuando, estén apoyados por un modelo y sensores de una calidad, mínimamente, aceptable.

Además con las simulaciones (tanto en Matlab como en ROS) se ha permitido una profunda demostración de las capacidades de estos filtros. Poniendo especial énfasis en ROS y su tremendo potencial como herramienta en el mundo de la robótica.

Se dejan como futuras mejoras, realizar diferentes versiones del código, de forma que aumente la exactitud del filtro mediante:

- Implementación de diferentes modelos para adecuar la predicción al movimiento del robot, además de incorporar control clásico mediante variables de actuación como velocidad angular de las ruedas, ángulo de dirección… Considerar obstáculos a la hora de proporcionar la información para simular sensores más realistas.

- Uso de mayor diversidad de sensores enfocados a la localización, tales como unidades IMU (acelerómetro, giróscopo, también, en algunos casos, magnetómetro), encoders (odometría), uso de balizas (láser o cámara). Esto permitiría complementar al filtro con técnicas de Dead-Reckoning o incrementar su precisión en los landmarks (balizas) gracias a la corrección del error en posición, de gran utilidad para momentos en los que al sistema no le llegue información externa o llegue con muy mala calidad.

Figura 5-1. KF combinando diferentes técnicas y sensores

- Configuración básica del algoritmo ajustable durante su ejecución. El algoritmo podría elegir, en todo momento, que modelo usar (de los mencionados en el primer punto). Con esto, podría intercambiarlos según identificara que se mueve en línea recta, girando, marcha atrás, etcétera.

- Muy relacionado con lo anterior, otra mejora sería ofrecer la posibilidad de cambiar los parámetros de ruido iniciales permitiría al robot darle más importancia al modelo o a los sensores de igual manera que antes, analizando el momento adecuado. De esta manera, se conseguirían más variables de control sin necesidad de incrementar el número de actuadores, permitiendo al algoritmo ajustarse en momentos de mala cobertura o de movimientos no contemplados en el modelo.

- Actualización dinámica de sensores, permitiendo en cualquier momento agregar, por ejemplo, un nuevo nodo al conjunto de WSN, simplemente especificando sus coordenadas en el plano; o dar de baja algún nodo estropeado o con muy mala cobertura.

- El punto anterior también plantea la interesante idea de algún tipo de comunicación internodal con la que los propios nodos sean capaces de evaluar su estado entre ellos, incluso pudiendo entrar en un modo de desconexión para ahorrar energía cuando el robot esté fuera de su rango, siendo alertado por los otros nodos en el momento que éste vuelva a estar dentro.

- Relacionado con los dos primeros puntos, una funcionalidad a añadir podría ser la ampliación a aplicaciones 3D, por ejemplo, empleo de robots aéreos (UAV, R-PAS). Habiendo que modificar el vector de estado e incorporando nuevos sensores para hacerlo posible.

Concluyendo, únicamente resta señalar las posibles aplicaciones, también mencionadas en el capítulo introductorio, que tendría este trabajo.

Para poder desenvolverse en el entorno, el robot requiere de cierta localización puesto que dentro de sus tareas puede requerirse ir a un cierto punto a recoger un objeto, realizar una comprobación del terreno, buscar algún elemento, etcétera. Es imprescindible para el desplazamiento a cualquier punto de forma certera, el saber de dónde se parte y dónde se encuentra en cada momento, además de saber de alguna forma adonde se quiere llegar.

Esto implica que la localización es vital para cualquier tipo de función autónoma o, incluso, teleoperada, dado que no siempre se puede tener acceso visible al entorno donde se despliega el robot. Algunos ejemplos de estas funciones son: localización, navegación y guiado de robots en aplicaciones como seguimiento de altas prestaciones, búsqueda, rescate y vigilancia, domótica, entre otros. Sobre todo, es interesante el empleo de la localización basada en rango en lugares donde la señal GPS es completamente inaccesible.

También sirve como base a técnicas de SLAM (Simultaneous Localization and Mapping), las cuales comparten como objetivos las mismas aplicaciones mencionadas en el párrafo anterior, siendo incluso, aún más importantes en su desarrollo, dado que el presente trabajo solo considera la localización y no la construcción de mapas. En el caso de usar sensores del tipo range-only, se habla de técnicas RO-SLAM.

GLOSARIO

EIF: Extended Information Filter 11

EKF: Extended Kalman Filter 9

Gazebo: Software gráfico 24

GUI: Graphic User Interface 23

IF: Information Filter 8

KF: Kalman Filter 4

MRU: Movimiento rectilíneo uniforme 15

PF: Particles Filter 11

Pioneer 3-AT: Robot móvil 23

ROS:Robot Operating System 22

WSN: Wireless Sensor Network (sensores de rango) 1

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

BIBLIOGRAFÍA

[1] S. Thrun, W. Burgard, D. Fox, *Probabilistic Robotics,* 2005.

[2] S. Thrun, Udacity Course CS373: *Artificial Intelligence for Robotics* [en línea] .

[3] *ROS Tutorials* [en línea], *Wiki* [en línea], *nav_msgs* [en línea].

[4] Cyrill Stachniss, *Extemded Information Filter* [en línea].

[4] A. Martinez, E. Fernández, *Learning ROS for Robotics Programming*, 2013.

[5] G. G. Rigatos, *Extended Information Filtering and nonlinear control for cooperating robot harvesters* [en línea] .

[5] Clearpath Robotics, *ROS Intro*, 2014 [en línea] .

[5] J. M. Buades Rubio, UIB, *Filtro de Partículas*, 2011 [en línea] .

[3] *Eigen Library* [en línea].

[3] Gazebo [en línea].