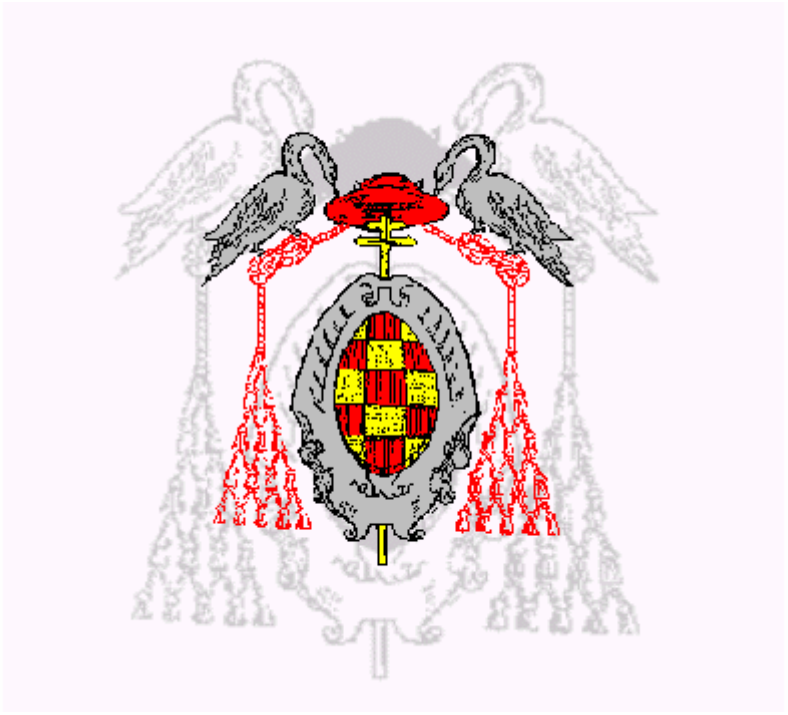
UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

INGENIERÍA TÉCNICA INDUSTRIAL
ESPECIALIDAD EN ELECTRÓNICA INDUSTRIAL

Trabajo Fin de Carrera



Comparativa teórica y empírica de métodos de estimación de la posición de múltiples objetos.

María Cabello Aguilar

Octubre de 2007

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

INGENIERÍA TÉCNICA INDUSTRIAL
ESPECIALIDAD EN ELECTRÓNICA INDUSTRIAL

Trabajo Fin de Carrera

Comparativa teórica y empírica de métodos de estimación de la posición de múltiples objetos.

Autor: María Cabello Aguilar
Tutora: Marta Marrón Romera

TRIBUNAL:

Presidente: D. Javier Gamo Aranda

Vocal 1º: D. Miguel Ángel Ruiz Arroyo

Vocal 2º: D. Marta Marrón Romera

CALIFICACIÓN.....

FECHA.....

INDICE

I. RESUMEN

Este proyecto está basado en el estudio de algoritmos probabilísticos de estimación de la posición de múltiples objetos. El objetivo es comprobar la funcionalidad y eficiencia de los mismos en función de una serie de variables cualitativas y ajustar estos algoritmos para conseguir los mejores resultados, todo esto mediante la realización de un conjunto de pruebas.

Las propuestas que se han analizado son un filtro de Kalman con un proceso de asociación probabilística (KFPDA), un filtro de Partículas con un proceso de clasificación (XPFCP) y un filtro de Partículas con un proceso de asociación probabilística (SJPDAF).

Palabras clave: algoritmos probabilísticos, métodos de estimación, visión, robótica, seguimiento de objetos.

CAPÍTULO 1. INTRODUCCIÓN

1.1. Objetivos perseguidos por el trabajo

En este proyecto se pretende comparar el funcionamiento de varios algoritmos de estimación y seguimiento de la posición de múltiples objetos mediante la realización de un conjunto de pruebas sobre tres propuestas diferentes: un filtro de Kalman con un proceso de asociación probabilística (KFPDA), un filtro de partículas con un proceso de clasificación determinístico (XPFCP) y un filtro de partículas con un proceso de asociación probabilística (SJPDAF).

El objetivo es comprobar la funcionalidad (sobre todo en medidas de eficiencia, exactitud y robustez) de estos algoritmos de estimación en función del análisis de ciertas variables cualitativas de los mismos. Para ello se realizan una serie de pruebas con el fin de ajustar las variables y parámetros y así obtener unos resultados lo más óptimos posibles.

Los algoritmos están desarrollados en Matlab ([Garcia01]) y las pruebas y presentación de resultados se harán también en este entorno.

Una vez realizado el ajuste definitivo de los estimadores se realizarán también un conjunto de pruebas en tiempo real para comprobar el funcionamiento de los algoritmos, estudio que no forma parte del presente proyecto.

Este trabajo está involucrado en la Tesis Doctoral titulada “Seguimiento de múltiples obstáculos en entornos interiores muy poblados basado en la combinación de métodos probabilísticos y determinísticos”, desarrollada por Marta Marrón Romera, actualmente en fase de desarrollo.

1.2. Pruebas utilizadas para las comparativas

Para el análisis y la posterior comparación del funcionamiento de los algoritmos se dispone de una serie de vídeos y sus respectivos ficheros de datos.

Cada vídeo consiste en una secuencia de imágenes donde aparecen varios objetos, tanto estáticos como dinámicos. En cada iteración hay una serie de puntos correspondientes a medidas de la posición de los objetos que van variando a lo largo del vídeo, aparecen nuevas y desaparecen otras en función de la aparición, desaparición u oclusión de los objetos en la escena. Los vídeos empleados incluyen todo tipo de situaciones con el fin de comprobar la funcionalidad del algoritmo a estudiar en los casos más desfavorables, como son las oclusiones, el cambio del número de objetos en la escena, la proximidad entre algunos de estos objetos, etc.

Los datos empleados en los algoritmos de seguimiento proceden de un sistema de estéreo-visión formado por dos cámaras. Para obtener las coordenadas de los puntos en 3D a partir de las imágenes en 2D se emplea un proceso de visión ([Marrón05]). Los datos procedentes de las cámaras han sido procesados y se encuentran en un fichero binario, donde se organizan de la siguiente forma para cada una de las iteraciones o frames:

- número de puntos en la iteración actual, *m*.
- coordenada *X* de lateralidad, *x*.
- coordenada *Z* de profundidad, *z*.
- coordenada *Y* de altura, *y*.
- …

Tantas coordenadas *X, Z, Y* como puntos se indiquen para cada frame mediante *m*.

En la Figura 1.1 se observa el sistema de coordenadas utilizado y los puntos de un objeto dinámico, puntos pertenecientes a los contornos de los objetos. El origen del sistema de coordenadas es el centro óptico de la cámara izquierda, tal y como ha sido el sistema de obtención de los datos de posición 3D.

Cada objeto es modelado por un cilindro cuyo centro proyectado en el plano *XZ* es conocido con el nombre de centroide, y su altura y radio tienen un valor prefijado. En la Figura 1.1 se puede ver este cilindro.

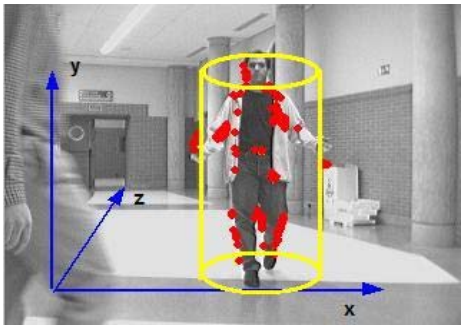


Figura 1.1. Sistema de coordenadas y puntos de borde

Se analiza el comportamiento de cada uno de los algoritmos a lo largo de estos vídeos, iteración a iteración. Cada algoritmo dispone de una serie de parámetros propios que se desean ajustar con el fin de obtener los mejores resultados y comprobar en qué medida afectan al funcionamiento de los mismos. Para cada prueba, en la que se desea ajustar un determinado parámetro, se dispone de una hoja de cálculo donde se anota, paso a paso, de qué objetos se tienen medidas y los diferentes errores que se encuentran.

Una vez finalizada la prueba se estudia la casuística de errores, el porcentaje de iteraciones erróneas que proporciona cada uno de ellos, etc. Así se intenta mejorar los resultados obtenidos, disminuyendo el número de errores en la medida de lo posible.

Este procedimiento se lleva a cabo para poder ajustar cada parámetro de cada uno de los algoritmos estudiados y sacar conclusiones sobre la funcionalidad de cada algoritmo y la influencia y efectos de de cada parámetro sobre ésta. Una vez que se ajustan la totalidad de los parámetros de todos los algoritmos, se hace una comparativa final que dé información sobre qué método es el más eficaz, el más robusto, etc.

Para hacer este estudio se realizan tres tipos de prueba:

- *Prueba corta*: También llamada prueba compleja. Fragmento del vídeo 10, desde la iteración 160 a la 260. Aparecen un gran número de objetos en la escena. El video comienza con tres elementos en la imagen que se mantienen durante el 25% de esta prueba; en el resto del video aparecen cuatro objetos. Se producen algunas oclusiones, es decir, durante el transcurso de esta prueba los objetos están en movimiento y como consecuencia de este movimiento obstruyen a otros. Se pueden clasificar las oclusiones como parciales o totales. En el total de iteraciones del video se producen un 15% de éstas con una oclusión total y un 50% de iteraciones con oclusión parcial.
- *Prueba larga*: También llamada prueba simple. Fragmento del vídeo 10, desde la iteración 750 a la 950. Esta prueba se utiliza para analizar la sensibilidad del algoritmo a los distintos parámetros y para evaluar si el ruido es filtrado correctamente. Se analiza la robustez de los estimadores con respecto a la aparición de medidas erróneas. En el 40% del vídeo aparece una sola persona como elemento dinámico y la papelera como objeto estático; en un 25% se encuentran tres objetos dinámicos; y en el resto del vídeo aparecen cuatro objetos dinámicos y el elemento estático. En el transcurso del video se detectan medidas erróneas (ruidos) en un 23% de las iteraciones de las que está compuesta esta prueba.
- *Prueba completa*: Vídeo 10 completo, con un total de 1098 iteraciones. Esta prueba se lleva a cabo después de analizar todos los parámetros del correspondiente algoritmo.

Para realizar estas pruebas se analizan los distintos errores que se encuentran a lo largo del vídeo, como se ha comentado anteriormente. A continuación se listan los más frecuentes:

- “no generado”: Error que se produce cuando no existe un filtro/clase validado encargado de llevar a cabo el seguimiento de un objeto del que se tienen medidas. Este fallo se produce principalmente cuando aparece un nuevo objeto en escena. En la asociación de partículas del PF este error implica que no existen partículas asociadas a un objeto del que se han extraído medidas.

- “unión de dos”: Error que se produce cuando existen dos objetos próximos, ambos con medidas y tienen un solo filtro/clase asignado para estimar la posición de los dos conjuntamente.
- “duplicado”: Error que se produce en el momento en que se validan dos filtros/clases de un único objeto.
- “desplazado”: Aparece el filtro/clase de un objeto desplazado respecto a las medidas extraídas de éste.

Es necesario decidir en muchas ocasiones cuál de las propuestas estudiadas es la que mejores resultados ofrece, en función de la cantidad y tipos de errores encontrados. Siempre se considera como error más importante el “no generado” ya que el objeto no es detectado. Por lo tanto es preferible que aparezcan, por ejemplo, errores de “duplicado” en los que el objeto es detectado y su clase o filtro validado.

1.3. Organización de la memoria

Esta memoria está formada por cinco capítulos:

- **Capítulo 1. Introducción**: Capítulo actual en el que se introduce brevemente el trabajo realizado.
- **Capítulo 2. Conocimiento de los algoritmos**: En este capítulo se analizan, uno a uno, los tres métodos estudiados a lo largo del proyecto. Se detalla el funcionamiento y las etapas de cada uno de ellos de forma precisa y completa, para un correcto entendimiento de éstos y de las pruebas realizadas a continuación.
- **Capítulo 3. Ajuste de los parámetros**: Este capítulo incluye el ajuste de los parámetros de los tres algoritmos, uno a uno. Se comentan aspectos importantes de las pruebas realizadas y se finaliza con las conclusiones acerca del ajuste de cada algoritmo y del efecto que tiene cada uno de los parámetros en el funcionamiento del mismo.
- **Capítulo 4. Comparativa de los algoritmos**: Se comparan los algoritmos en cuanto a errores cometidos por cada uno de ellos y se comentan las conclusiones de cada estudio. Por último se muestran otro tipo de comparativas en relación a otros aspectos, como son el tiempo de ejecución de cada algoritmo, el filtrado de ruido, etc.
- **Capítulo 5. Conclusiones y trabajos futuros**: En este capítulo final se exponen las conclusiones de todo el trabajo realizado y posibles trabajos futuros en este campo.

CAPÍTULO 2. CONOCIMIENTO DE LOS ALGORITMOS

La estimación de la posición es necesaria para conocer el entorno del robot y su localización concreta por él mismo, principalmente en tareas de navegación autónoma ([Marrón05]).

Teniendo en cuenta las restricciones impuestas por las medidas, el proceso de estimación debe tener en cuenta el ruido asociado a ellas y al modelo de movimiento de los objetos del entorno del que proceden dichas medidas. Los algoritmos probabilísticos, como el filtro de Partículas o el filtro de Kalman lo tienen en cuenta.

El objetivo de estos algoritmos estimadores es calcular la probabilidad "a posteriori" ($p(\vec{x}_t|\vec{y}_{1:t})$) del vector de estado $\vec{x}_t$ teniendo en cuenta el conjunto de vectores de salida (medidas) hasta el instante actual t, $\vec{y}_{1:t}$. El vector de estado informa sobre la posición del objeto a seguir y el cálculo de esta probabilidad se realiza a través de dos pasos recursivos de predicción-corrección, en los que al menos algunas de las variables involucradas son estocásticas.

Cuando el número de objetos a seguir es variable el proceso de estimación se complica, y es entonces necesario un proceso de asociación como la clasificación o la Asociación Probabilística de Datos (PDA). Estos métodos de asociación son indispensables para que el estimador trabaje en todo momento con la entrada correcta.

En cualquier caso, es necesario un proceso de asociación de datos y otro de estimación. Además resulta indispensable introducir un proceso de creación y eliminación de objetos ya que ninguno de los algoritmos de asociación básicos conocidos empleados en estas tareas lo incluye.

Por lo tanto en una tarea de seguimiento de múltiples objetos se hacen necesarios los siguientes procesos:

- *Proceso de estimación*: Necesario para conocer el entorno de los objetos y la posición de los mismos.

- *Proceso de asociación*: Necesario para asociar las medidas a los objetos correspondientes en la escena.

- *Proceso de creación/eliminación de objetos*: Indispensable para trabajar en entornos con un número variable de objetos.

Los algoritmos que se han analizado y que se explican a continuación son los siguientes:

- *KFPDA*: Algoritmo basado en un filtro de Kalman como estimador y un proceso de Asociación Probabilística de Datos.
- *XPFCP*: Algoritmo basado en un filtro de Partículas como estimador y un clasificador como proceso de asociación.
- *SJPDAF*: Algoritmo basado en un filtro de Partículas como estimador y un proceso de Asociación Probabilística Conjunta de Datos.

2.1. Algoritmo “KFPDA”

Este algoritmo está basado en un filtro de Kalman como estimador y un proceso de “Asociación Probabilística de Datos” (PDA, Probabilistic Data Association).

En nuestro caso se crea un KF y un PDA para cada uno de los objetos a seguir. En otros estudios se genera un único PDA para la totalidad de los objetos, en cuyo caso el algoritmo recibe el nombre de “Filtro de Asociación Probabilística de Datos” (PDAF, Probabilistic Data Association Filter). Este método ofrece la mejor solución para aquella situación en la que sólo hay un objeto a seguir y múltiples medidas o ninguna medida relacionada con el objeto ([Rasmussen01]), cosa que no ocurre en los vídeos que se utilizan en este proyecto. Sin embargo, cuando existen múltiples objetos en la escena, el PDAF por sí solo difícilmente es capaz de llevar a cabo la asociación de forma correcta.

Por lo tanto nuestra propuesta se basa en crear tantos KFs y PDAs como objetos con medidas existan en la escena en cada momento.

2.1.1. El estimador “Filtro de Kalman”

El filtro de Kalman (KF) es un algoritmo de procesamiento de datos recursivo ([Maybeck79]), que se ha revelado como una eficiente herramienta para estimar el estado de un proceso. Desde su comienzo, el KF ha sido aplicado en el área de navegación autónoma o asistida entre otras muchas.

El KF proporciona una implementación óptima del filtro Bayesiano minimizando la covarianza del error de estimación. Para el posible uso de este estimador es necesario que se cumplan las siguientes restricciones:

- *El sistema ha de ser lineal*. En caso de que esto no se cumpla existe una extensión conocida como "Filtro de Kalman Extendido" (EKF, Extended Kalman Filter) para resolver el problema de estimación.
- *El ruido de las medidas y del modelo* utilizado debe estar caracterizado por "Funciones de Densidad de Probabilidad" (PDFs) normales y parametrizado por las siguientes condiciones: media nula, independencia (correlación cruzada nula de los ruidos) y, en el caso de ser vectoriales, matrices de covarianza diagonales (vectores de ruido "blancos").

Este filtro lleva a cabo una estimación del vector de estado del sistema $\vec{x}_t$, compara la salida real con la obtenida con el modelo de estimación y mejora esta estimación a medida que se va ejecutando el algoritmo de forma recursiva, adaptando la constante de estimación (también llamada de Kalman).

El modelo utilizado debe venir definido por la siguiente expresión:

$$\begin{array}{c} \vec{x}_{t+1|t} = A \cdot \vec{x}_t + B \cdot \vec{u}_t + \vec{q}_t \rightarrow \vec{x}_{t+1|t} = f\left(\vec{x}_t, \vec{u}_t, \vec{q}_t\right) \\ \vec{y}_t = H \cdot \vec{x}_t + \vec{r}_t \rightarrow \vec{y}_t = h\left(\vec{x}_t, \vec{r}_t\right) \end{array} , \qquad (2.1)$$

donde $\vec{x}_t$ corresponde al vector de estado, $\vec{y}_t$ es el vector de salida, $\vec{u}_t$ corresponde al vector de entrada y $\vec{q}_t$ y $\vec{r}_t$ indican el ruido del modelo y de las medidas respectivamente. Las ecuaciones de la derecha corresponden al comportamiento general del sistema, mientras que las de la izquierda se usan cuando el modelo es lineal, restricción presente en el filtro de Kalman ([Marrón07]).

En nuestra aplicación el vector de estado coincide con el vector de salida, por lo tanto la matriz H es la matriz identidad y el modelo se puede reescribir como:

$$\vec{y}_t = \vec{x}_t + \vec{r}_t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_t \\ z_t \end{bmatrix} + \vec{r}_t \qquad (2.2)$$

En la ecuación (2.1) el vector de entrada se obtiene de $\vec{y}_t$ y $\vec{x}_{t|t-1}$.

$$\vec{u}_t = \frac{\vec{y}_t - \vec{x}_{t|t-1}}{T_s} \qquad (2.3)$$

Si se aplica el modelo dinámico en (2.1) se tiene:

$$\vec{x}_{t+1|t} = \vec{x}_t + B \cdot \vec{u}_t = \begin{bmatrix} x_{t+1|t} \\ z_{t+1|t} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_t \\ z_t \end{bmatrix} + \begin{bmatrix} T_s & 0 \\ 0 & T_s \end{bmatrix} \cdot \begin{bmatrix} vx_t \\ vz_t \end{bmatrix} + \vec{q}_t \qquad (2.4)$$

En (2.4), $\begin{bmatrix} X & Z \end{bmatrix}'$ son las coordenadas Cartesianas de la proyección del centroide del objeto en el plano.

El funcionamiento del filtro de Kalman consiste en dos etapas que se llevan a cabo para cada uno de los filtros creados. Estas etapas son predicción y corrección:

1. ***Predicción*** (desarrollada en el instante *t*): Se realiza una predicción del vector de estado $\vec{x}_{t+1|t}$ y de la matriz de covarianza del error de estimación $P_{t+1|t}$.

2. ***Corrección*** (desarrollada en *t+1*): A partir del nuevo conjunto de medidas $\vec{y}_{t+1}$, de la ecuación de salida del modelo y de $P_{t+1|t}$ se calcula la ganancia de Kalman K_{t+1} que permite actualizar la estimación del vector de estado $\vec{x}_{t+1|t+1}$ y la covarianza del error de estimación para ese instante $P_{t+1|t+1}$.

En la Figura 2.1 se muestra el funcionamiento del filtro de Kalman, donde se pueden ver las dos fases del estimador.

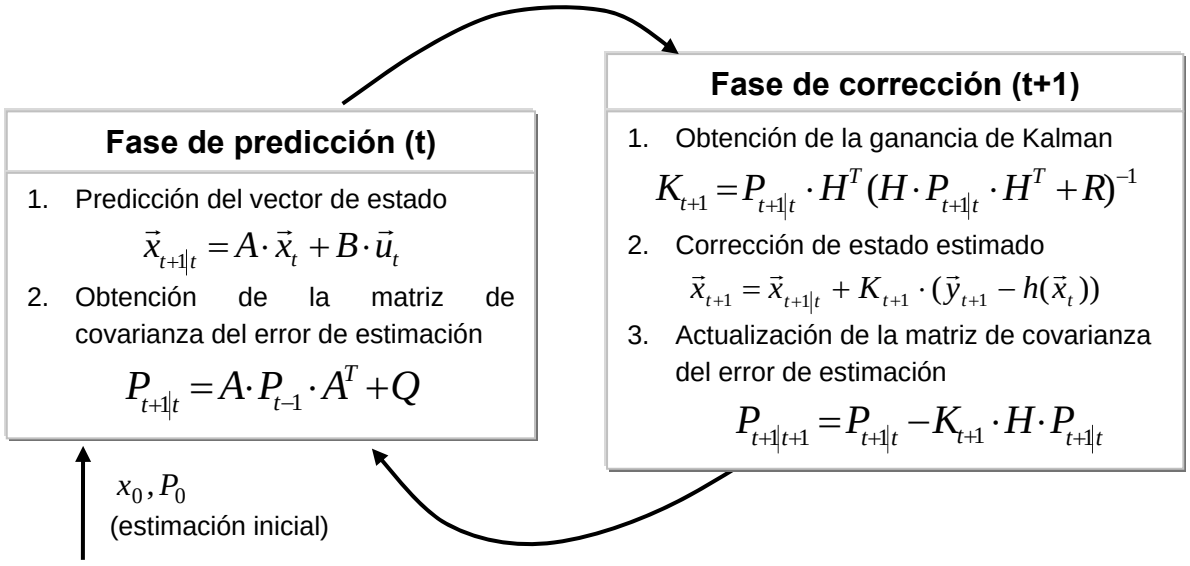


Figura 2.1. Descripción de la funcionalidad del KF.

Cuando en una escena se encuentran varios objetos, la dinámica de cada uno es distinta. Por esto, se debe emplear un KF para cada uno de estos objetos. Otra opción sería que el tamaño del vector de estado cambiara dinámicamente para incluir o eliminar las variables de estado de los objetos que aparecen o desaparecen, pero esto supondría un alto coste computacional y una difícil aplicación en tiempo real al ser el tiempo de ejecución variable.

Como consecuencia de esto, la mejor opción es utilizar tantos filtros de Kalman como objetos se encuentren en la escena de estudio ([Welch01], [MacCormick99]).

En cualquier caso, el proceso de asociación es indispensable para poder llevar a cabo correctamente el paso de corrección.

2.1.2. El algoritmo de "Asociación Probabilística de Datos"

El proceso de Asociación Probabilística de Datos se encarga de calcular la probabilidad de que una medida pertenezca a un objeto determinado. Por lo tanto es necesario crear uno por cada objeto. De esta forma se tienen $m \times k$ hipótesis de asociación, siendo k el número de filtros generados por el algoritmo y m la cantidad de medidas.

Este método no es considerado como óptimo ya que sólo tiene en cuenta las últimas medidas que han aparecido $\vec{y}_t$ en vez del set total de medidas $\vec{y}_{1:t}$ ([Karlsson02]).

Con el fin de minimizar el tiempo de ejecución se introduce un proceso de *gating* mediante el cual sólo las medidas que se encuentre dentro de un límite (*distM*) alrededor del centroide de un determinado objeto serán asociadas al mismo. A las medidas asociadas a un objeto se les denomina mj. Este método es usado recurrentemente en el área de navegación local.

La probabilidad $\beta_{i,j}$ de que cada una medida i pertenezca a un objeto j puede ser calculada de diversas maneras ([Marrón07]), pero en este caso se opta por utilizar la distancia euclídea de la medida al vector de salida predicho del objeto j ($d_{i,j}$) de la siguiente forma:

$$\beta_{i,j} = e^{-\frac{1}{2}\cdot\frac{\min(d_{i,j}^2)}{distM^2}} / i = 1:m, j = 1:k, d = \left|\vec{y}_i - \vec{y}_{t|t-1}\right|, \tag{2.5}$$

donde $distM = \sqrt{2}\cdot\sigma$ representa el efecto del *gating* en el cálculo de la probabilidad; m es el número de medidas extraídas del entorno en cada iteración y k es el número de objetos seguidos.

Si la distancia $d_{i,j}$ de la medida i al objeto más cercano j es mayor que el límite, esta medida será definida como un nuevo objeto y se creará un nuevo filtro de Kalman para estimar su posición recursivamente. De esta forma, gracias al proceso del *gating*, el algoritmo se va adaptando por sí mismo para poder predecir la posición de un número variable de objetos.

Para realizar la asociación, el PDA introduce el parámetro conocido como innovación combinada en el funcionamiento básico del KF mostrado en la Figura 2.1. Para definir este elemento es necesario en primer lugar definir la innovación individual o estándar como:

$$\vec{r}_{t+1} = \vec{y}_{t+1} - h(\vec{x}_t) \tag{2.6}$$

es decir, es la diferencia entre la medida que se obtien en el instante actual *t* y el estado estimado en el instante anterior *t-1* ([Broddfelt05]). Este elemento aparece a la hora de actualizar y corregir el estado estimado en la Figura 2.1.

La innovación combinada o residuo de cada uno de los objetos que es introducida por el PDA en nuestro KF se define como la suma de las innovaciones individuales de las mj medidas del instante actual:

$$residuo_j = \Sigma_{i=1}^{mj}\beta_{i,j}\cdot\vec{r}_j, \tag{2.7}$$

donde cada β_{ij} es la probabilidad de que la medida i sea asociada al objeto j.

En nuestra aplicación, cuando se aplica el concepto de residuo, la ecuación para corregir el estado en el paso de corrección del KF cambia (ver Figura 2.1) al sustituir la innovación individual ($r_t = \vec{y}_{t+1} - h(\vec{x}_t)$) por la combinada:

$$r_{j,t} = \sum_{i=1}^{mj} \beta_{i,j} \cdot \left(\vec{y}_t^{(i)} - h(\vec{x}_{t|t-1}^{(j)}) \right) \tag{2.8}$$

La innovación combinada se calcula para las mj medidas asociadas al objeto j en cada iteración.

2.1.3. KFPDA utilizado

La estructura del KFPDA basado en un KF como estimador y un PDA como proceso de asociación se muestra en la Figura 2.2:

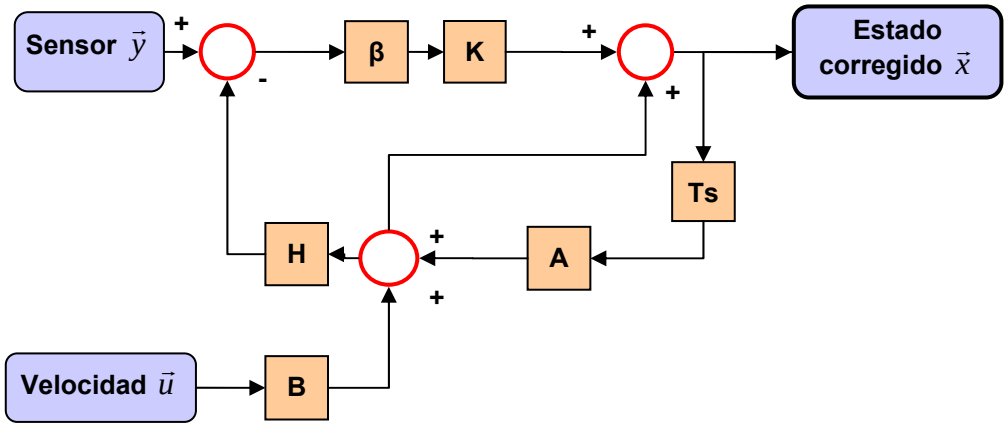


Figura 2.2. Diagrama de bloques del funcionamiento del algoritmo diseñado, incluyendo el PDA en el bucle del KF.

Una vez visto este diagrama general de funcionamiento, se procede a detallar la implementación software del algoritmo utilizado para la realización de las pruebas.

En primer lugar, se dispone de una variable llamada *iterKF*, correspondiente al número de la iteración actual, que es utilizada para poder reiniciar todo el proceso en el momento apropiado y también para inicializar el algoritmo al ejecutarlo. El proceso de reinicialización depende del número de filtros existentes k (independientemente del proceso de validación correspondiente que será comentado más adelante) y del número de medidas en la iteración actual m.

En la Figura 2.3 se muestra un flujograma donde se observan las llamadas a las distintas funciones.

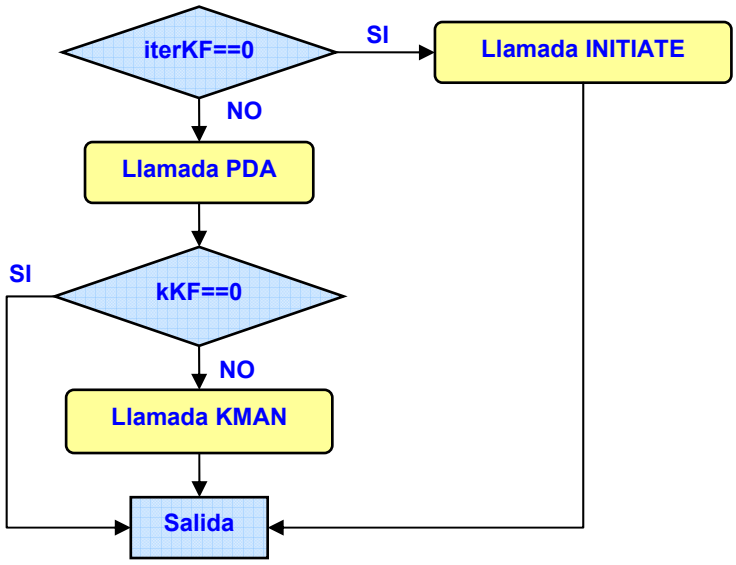


Figura 2.3. Flujograma del comportamiento general del KFPDA.

Como se puede ver en la Figura 2.3, cada vez que *iterkF* tenga asignado el valor 0, comienza de nuevo el algoritmo y se realiza la llamada a la función *Initiate,* donde se reinician todas las variables correspondientes.

Para almacenar todas los filtros se dispone de un array de estructuras llamado *Filtro*. Cada estructura corresponde a un filtro y posee los siguientes campos:

- *I*: Identificador del objeto usado para estimar su posición.
- *mj*: Número de medidas asociadas al filtro
- *M*: Matriz de valores de todas las medidas asociadas.
- *C*: Coordenadas del centroide.
- *Beta*: Array de probabilidades, donde cada una de ellas representa la probabilidad de que una de las medidas asociadas corresponda a dicho filtro. Por lo tanto las medidas y la correspondiente probabilidad de cada una deben tener el mismo orden.
- *resid*: Innovación combinada o residuo que será empleado en las ecuaciones del filtro de Kalman.
- *xp*: Valor del estado predicho del centroide.
- *xc*: Valor del estado corregido del centroide.
- *P_predicted*: Predicción de la matriz de covarianza del error de estimación.
- *P_corrected*: Corrección de la matriz de covarianza del error de estimación.
- *K*: Ganancia de Kalman.

- *U*: Velocidad del objeto.
- *candidate*: Variable que da información sobre la validación del filtro que será explicada a continuación.
- *validated*: Variable que está a "*true*" si el filtro está validado, o a "*false*" si aún no lo está.

En la Figura 2.4 se muestra el funcionamiento general de la función *PDA* (ver Figura 2.3) en la que se lleva a cabo la asociación de los datos a los distintos filtros creados, o la creación de un nuevo filtro.

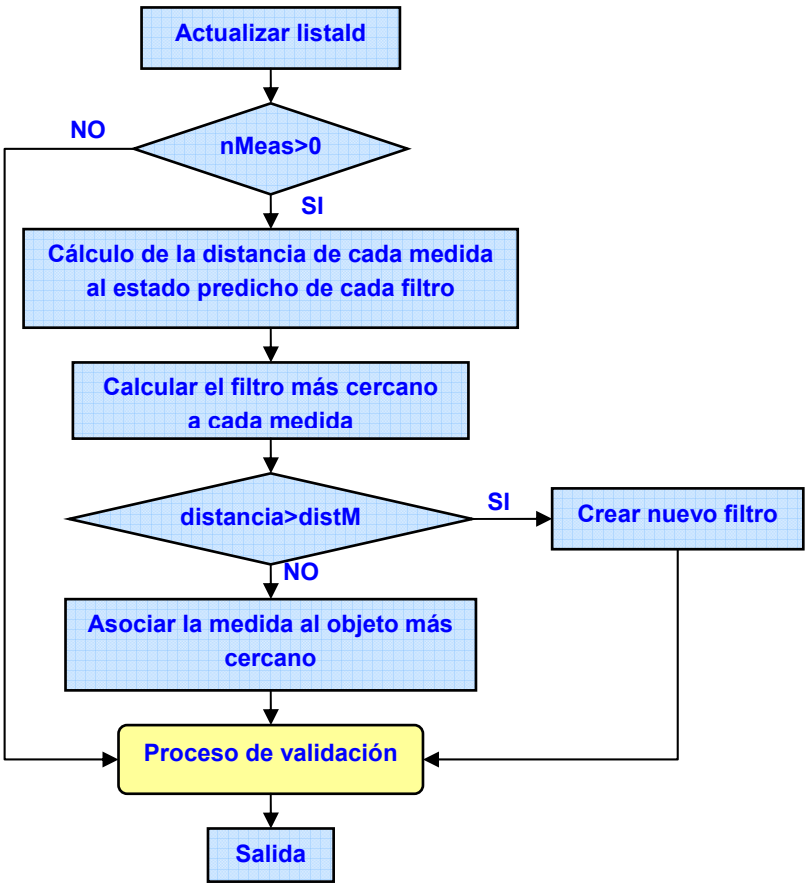


Figura 2.4. Diagrama funcional de la función PDA.

Además, se cuenta con una lista de identificadores *listaId* que permite, junto con el identificador asignado a cada filtro en el campo correspondiente (*Filtro.I*), referenciar en todo momento a cada uno de los filtros. Esta lista permite contrastar la lista actual (en el instante *t*) de filtros con la correspondiente al instante anterior (en el instante *t-1*).

Se observa en el diagrama anterior que existe un proceso de validación incluido en la función *PDA* detallado en la Figura 2.5:

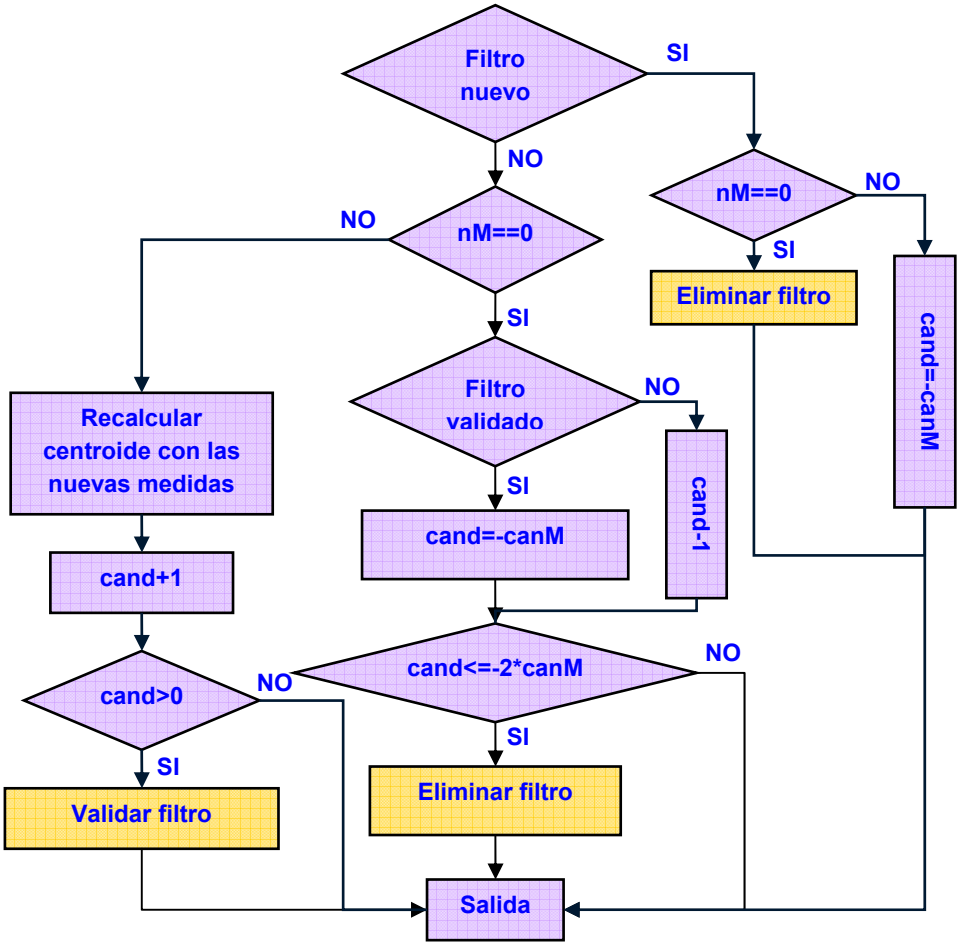


Figura 2.5. Diagrama de funcionamiento del proceso de validación.

Este proceso valida los filtros respondiendo a un sencillo criterio:

- -Cuando se crea un filtro, a su variable *candidate* (*cand* en el gráfico) se le asigna el valor '*–canM*' (se modifica este valor posteriormente para estudiar la sensibilidad del algoritmo frente a este parámetro).
- Si en una iteración no tiene ninguna medida asociada se le resta '1' a este valor, y si por el contrario sí las tiene, se le suma €'1'.
- De esta forma, cuando *candidate* llegue a '0' el filtro es validado (*validate=1*), y cuando llegue a '*-2*canM*' se invalida eliminándolo.

Por último sólo queda detallar la funcionalidad de la función *KMAN* (ver Figura 2.3), en la que se llevan a cabo todos los pasos del filtro de Kalman. En la Figura 2.6 se muestra su funcionamiento.

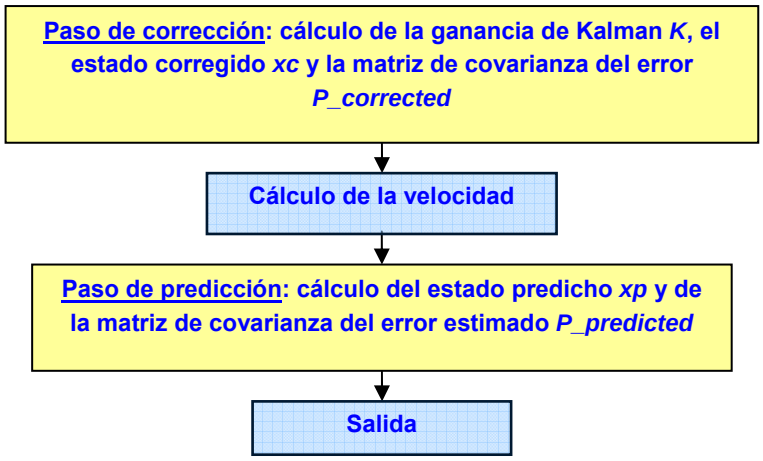


Figura 2.6. Flujograma con los pasos de la función KMAN.

En primer lugar se realiza la corrección del estado actual t para posteriormente predecir el estado en el instante siguiente $t+1$.

Los parámetros que van a ser estudiados para así comprobar su sensibilidad en el algoritmo e intentar optimizarlo son los siguientes:

- *distMKF*: límite del *gating* en el proceso de asociación.
- *canM*: parámetro empleado en el proceso de creación y eliminación de filtros.
- *dtR*: desviación típica del ruido de las medidas.
- *dtQ*: desviación típica del ruido del modelo.
- P_o : Covarianza del error de estimación inicial.

2.2. Algoritmo “XPFCP”

El algoritmo “Filtro de Partículas Extendido con proceso de Clasificación” (XPFCP, Extended Particle Filter with a Clustering Process) está compuesto por un filtro de Partículas extendido que permite realizar la estimación del vector de estado $\vec{x}_t$, y un proceso de clasificación que hace las veces de proceso de asociación y aumenta la robustez del filtro de partículas comentado ([Marrón05]).

2.2.1. El estimador “Filtro de Partículas Extendido”

Los filtros de Partículas (PF) son algoritmos de estimación que al contrario que los KFs permiten modelar el comportamiento del sistema de interés mediante funciones de probabilidad multimodales, por lo que son especialmente adecuados en aplicaciones de estimación para sistemas

cuyos modelos son no lineales y tienen asociados ruidos que pueden ser no Gaussianos. El filtro de Partículas es un método de Monte Carlo recursivo.

La idea principal de un PF es representar y actualizar la función de probabilidad *a posteriori* $p(\vec{x}_t|\vec{y}_{1:t})$ mediante un conjunto de muestras aleatorias $S_{t-1}=\{\vec{x}_{t-1}^{(i)},\tilde{w}_{t-1}^{(i)}\}_{i=1}^{n}$ que tienen pesos asociados y calcular el estado estimado $\vec{x}_{t+1|t}$ a través de esas muestras y esos pesos ([Almeida05]).

Una descripción detallada de la base matemática del PF se puede encontrar en [Smith05] y [Gordon93].

En el modelo dinámico del filtro de Partículas, el vector de estado incluye la posición y la velocidad en coordenadas Cartesianas relativas a las del robot. Dicho modelo se muestra a continuación:

$$\vec{x}_{t+1|t}=A\cdot\vec{x}_t+\vec{q}_t=\begin{bmatrix}1&0&0&T_s&0\\0&1&0&0&T_s\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\end{bmatrix}\cdot\begin{bmatrix}x_t\\y_t\\z_t\\vx_t\\vz_t\end{bmatrix}+\vec{q}_t \tag{2.9}$$

$$\vec{y}_t=H\cdot\vec{x}_t+\vec{r}_t=\begin{bmatrix}x_t\\y_t\\z_t\end{bmatrix}=\begin{bmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\end{bmatrix}\cdot\begin{bmatrix}x_t\\y_t\\z_t\\vx_t\\vz_t\end{bmatrix}+\vec{r}\,, \tag{2.10}$$

donde $\vec{x}_t$ es el vector de estado y $\vec{y}_t$ el vector de medidas.

El bucle principal del PF estándar, llamado SIR (Sequential Important Resampling) ([MacCormick99], [Gordon93]) comienza en el instante t con un set $S_{t-1}=\{\vec{x}_{t-1}^{(i)},\tilde{w}_{t-1}^{(i)}\}_{i=1}^{n}$ de n muestras o partículas aleatorias, representando la probabilidad del vector de estado $p(\vec{x}_{t-1}|\vec{y}_{1:t-1})$, estimado en el instante previo $t-1$. El resto del proceso se desarrolla a lo largo de tres pasos:

1. ***Predicción***: Las partículas se propagan a través del modelo de estado del sistema bajo estudio para obtener un nuevo set $S_{t|t-1}=\{\vec{x}_{t|t-1}^{(i)},\tilde{w}_{t-1}^{(i)}\}_{i=1}^{n}$ que representa la distribución *a priori* del vector de estado en el instante t, $p(\vec{x}_t|\vec{y}_{1:t-1})$.

2. ***Corrección***: El peso de cada partícula $\vec{w}_t = \left\{w_t^{(i)}\right\}_{i=1}^{n} \equiv w(\vec{x}_{0:t})$ se obtiene comparando el vector de salida $\vec{y}_t$ y el valor predicho basado en la estimación $h(\vec{x}_{t|t-1})$, lo que implica que se obtiene directamente de la función de verosimilitud $p(\vec{y}_t|\vec{x}_t)$ del modelo elegido:

$$w(\vec{x}_{0:t}) = w(\vec{x}_{0:t-1}) \cdot \frac{p(\vec{y}_t \mid \vec{x}_t) \cdot p(\vec{x}_t \mid \vec{x}_{t-1})}{q(\vec{x}_t|\vec{x}_{0:t-1}, \vec{y}_{1:t})} \xrightarrow[q(\vec{x}_t|\vec{x}_{0:t-1},\vec{y}_{1:t}) \propto p(\vec{x}_t|\vec{x}_{t-1})]{} w(\vec{x}_{0:t}) = w(\vec{x}_{0:t-1}) \cdot p(\vec{y}_t \mid \vec{x}_t) \tag{2.11}$$

3. ***Selección***: Usando el vector de pesos $\vec{w}_t = \left\{w_t^{(i)}\right\}_{i=1}^{n}$ calculado en el paso anterior y aplicando el método de muestreo elegido, un nuevo set $S_t = \left\{\vec{x}_t^{(i)}, \tilde{w}_t^{(i)}\right\}_{i=1}^{n}$ es obtenido con las partículas más probables, que representarán la nueva creencia $p(\vec{x}_t|\vec{y}_{1:t})$ o probabilidad *a posteriori* del vector de estado. Este conjunto de muestras es usado como entrada para la iteración del algoritmo en el instante $t+1$.

En la aplicación, el filtro de Partículas estándar se puede usar para realizar la estimación de un único objeto definido a través de su modelo de movimiento, pero no para el caso en que aparezcan objetos ya que en este caso no se asociará ninguna partícula al nuevo objeto a no ser que su vector de estado fuese similar al del ya estimado.

Para adaptar este método a un número variable de objetos se introducen algunas modificaciones. La versión utilizada en este proyecto está detallada en [Marrón07], y presenta las siguientes innovaciones:

- *Un nuevo paso de reinicialización*: En este paso previo a la predicción, n_m de las n partículas totales son sustituidas por medidas procedentes del vector de observación $\vec{y}_{t-1}$. Es la única forma de obtener una creencia que represente a múltiples hipótesis sin que sean rechazadas en el paso de selección.
- *En el paso de corrección*: Por una parte, sólo $n - n_m$ muestras del set de partículas son extraídas en este paso, y por otra el cálculo de los pesos de las partículas cambia al emplear para su cómputo el proceso de clasificación del que se hablará posteriormente.

2.2.2. El proceso clasificador

Como se ha comentado previamente, el clasificador aumenta la robustez del proceso de estimación probabilístico y le permite adaptarse al seguimiento de un número variable de objetos.

El algoritmo de clasificación organiza el set de medidas en grupos o clases que representan a todos los objetos que se encuentran en una escena ([Marrón07]).

Se ha usado como algoritmo de clasificación en el XPFCP una versión adaptada del K-Medias para un número variable de objetos. Este método de clasificación es recursivo y *booleano* y se caracteriza principalmente por los siguientes aspectos ([Cerro07]):

- Los elementos pertenecientes a cada grupo deben ser similares entre sí, y diferentes a los de otros grupos, en relación a la característica que se emplee como variable de clasificación.
- Los datos clasificados han de pertenecer a un grupo y sólo a uno.
- No puede quedar ninguna clase vacía.

El algoritmo K-Medias consta de las siguientes etapas ([Alsabti98], [Kanungo02]):

1. Se eligen k datos del set Y para utilizarlos como valor inicial de los centroides de las clases a crear.
2. Se calcula la distancia de cada dato a todos los centroides y se asocia el dato a la clase más próxima.
3. Realizada la asignación de todo el set de datos Y, se recalcula el valor del centroide de cada clase, como la media geométrica del conjunto de datos asignados a la clase en el espacio de definición de la distancia Euclídea.
4. Se analiza si ha habido algún cambio en la última iteración en el valor del centroide de cada clase o, lo que es equivalente, en los miembros asignados a cada clase. Si es así, el algoritmo se repite desde el paso 2.
5. En caso contrario, o si el número de veces que se ha ejecutado el proceso excede a un limite predefinido, el algoritmo finaliza, antes de lo cual se eliminan las clases que han quedado sin miembros asociados.

Uno de los principales inconvenientes de este segmentador es que los ruidos reciben el mismo tratamiento que el resto de datos, y no se pueden distinguir ni asociarse a algún grupo especial ([LeSaux02]).

Para dar solución al problema de las medidas de ruido se incluye un proceso de validación que aprovecha la aparición esporádica de dichos ruidos para conseguir filtrarlos.

Aparte del proceso de validación comentado, se incluye al K-Medias básico un proceso de estimación inicial de los centroides de las clases con el fin de mejorar la convergencia y el tiempo de ejecución del segmentador.

Con todo esto se obtiene el algoritmo K-Medias utilizado en este trabajo, cuyo flujograma de funcionamiento se muestra en la Figura 2.9 .

2.2.3. XPFCP utilizado

Al igual que en el KFPDA, en este algoritmo se dispone de una variable llamada *iterXPF*, como se ve en la Figura 2.7, que permite reinicializar el algoritmo en el momento apropiado y también arrancar el proceso al ejecutarlo por primera vez. El proceso se reinicia cuando no hay ninguna clase, momento en el que se para la ejecución, y no vuelve a comenzar (*iterXPF*=0) hasta que no se disponga de medidas nuevas.

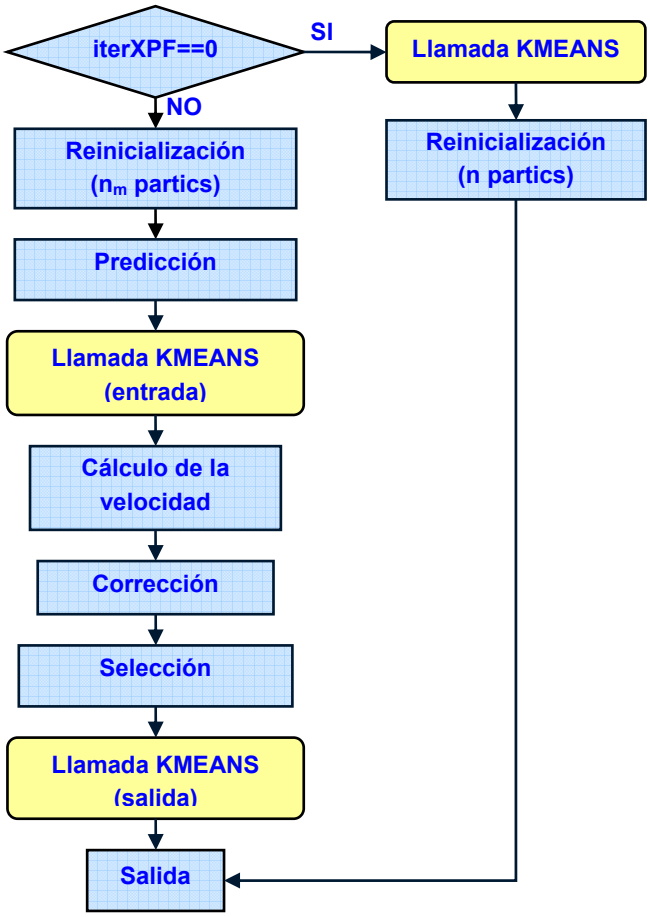


Figura 2.7. Diagrama de funcionamiento general del XPFCP.

Se observan dos diferentes pasos de reinicialización. Cuando *iterXPF* es 0, es decir, o bien se ha arrancado por primera vez el algoritmo o bien se ha reiniciado por falta de clases, hay que distribuir las *n* partículas totales entre las clases que ha devuelto el K-Medias. Por el contrario, si es cualquier otra iteración, sólo n_m partículas de las *n* totales serán sustituidas por medidas procedentes del vector de observación $\vec{y}_{t-1}$.

Como se puede ver en la Figura 2.7, después de la etapa de selección, aparece una llamada al proceso de clasificación. Esto se debe a que se realizan dos tipos de procesos de clasificación a los que se llama *clasificación de entrada* y *clasificación de salida*.

En el proceso de *clasificación de entrada*, en primer lugar se completa el set de partículas con n_m medidas nuevas, en función de la asociación en el instante anterior. Posteriormente se hace la llamada a la función *K-Medias*, que organiza la predicción de las partículas en clases o grupos.

En la *clasificación de salida*, se llama de nuevo a la función *K-Medias* de modo que las partículas que se obtienen tras el paso de selección sean agrupadas en clases o grupos. De esta forma se puede conocer cuántos objetos existen en la escena.

Como consecuencia de estas dos etapas diferenciadas, existen dos estructuras llamadas *ClusterIn* y *ClusterOut*, cada una de ellas con los siguientes campos:

- *I*: Identificador del objeto usado para estimar su posición.
- *mj*: Número de medidas asociadas a la clase.
- *M*: Matriz de valores de todas las medidas asociadas.
- *C*: Coordenadas y velocidad del centroide.
- *P*: Probabilidad de la clase que también sirve como parámetro para la validación.
- *candidate*: Variable que da información sobre la validación del filtro que será explicada a continuación.
- *validated*: Variable que está a "*true*" si la clase está validada, o a "*false*" si aún no lo está.

En la Figura 2.8 se ve de forma más detallada el lazo principal del algoritmo *XPFCP* con detalles sobre los distintos pasos. Se muestra únicamente la clasificación de las partículas en la entrada:

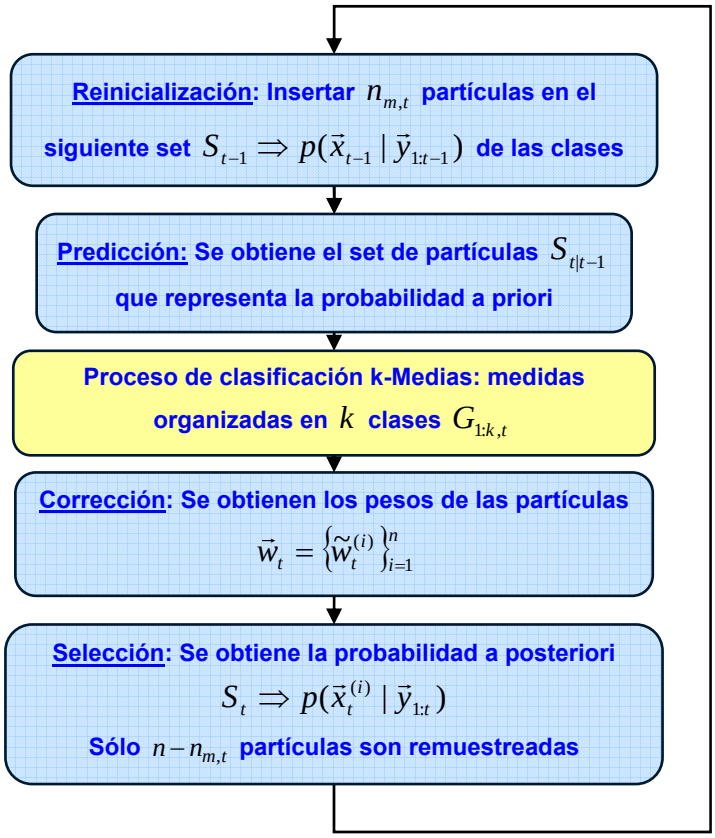


Figura 2.8. Flujograma detallado del lazo principal del XPFCP.

Por último en la Figura 2.9 se muestra un diagrama funcional del algoritmo K-Medias.

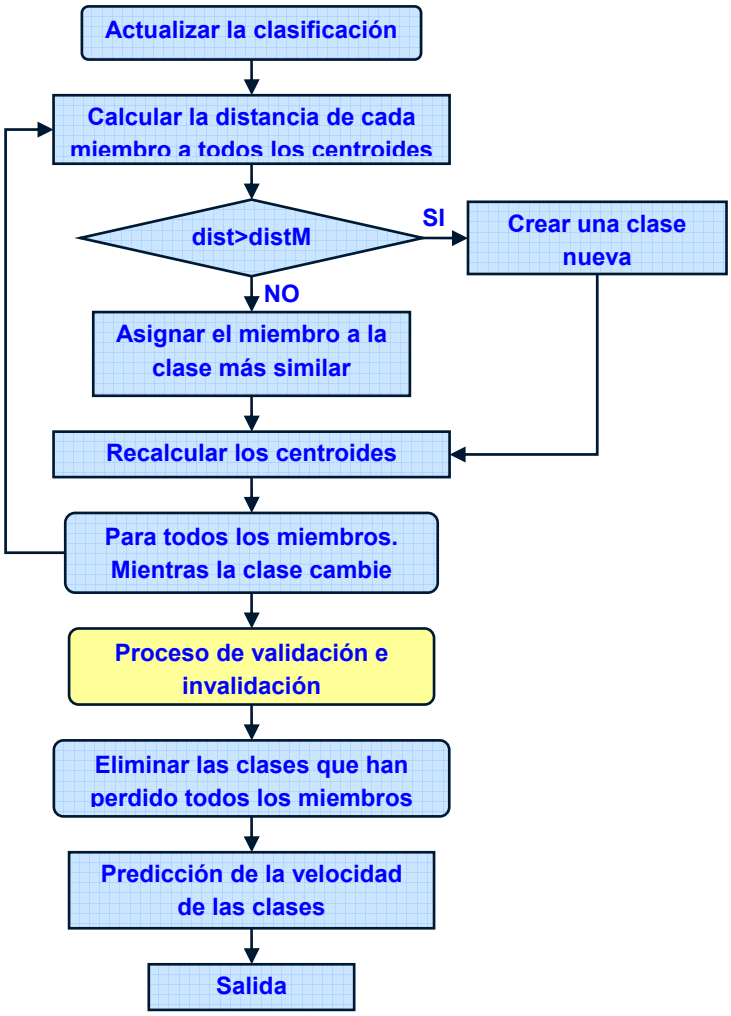


Figura 2.9. Diagrama funcional del algoritmo de clasificación K-Medias.

En la Figura 2.10 y la Figura 2.11 se muestran los procesos de validación e invalidación.

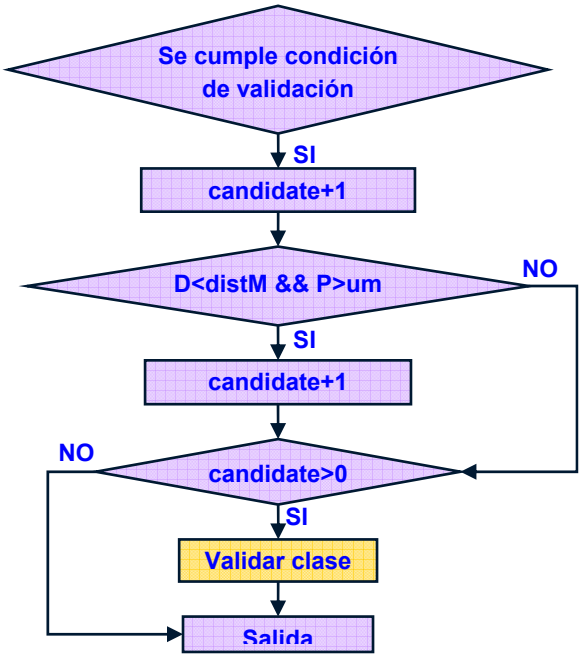


Figura 2.10. Diagrama de funcionamiento del proceso de validación de las clases.

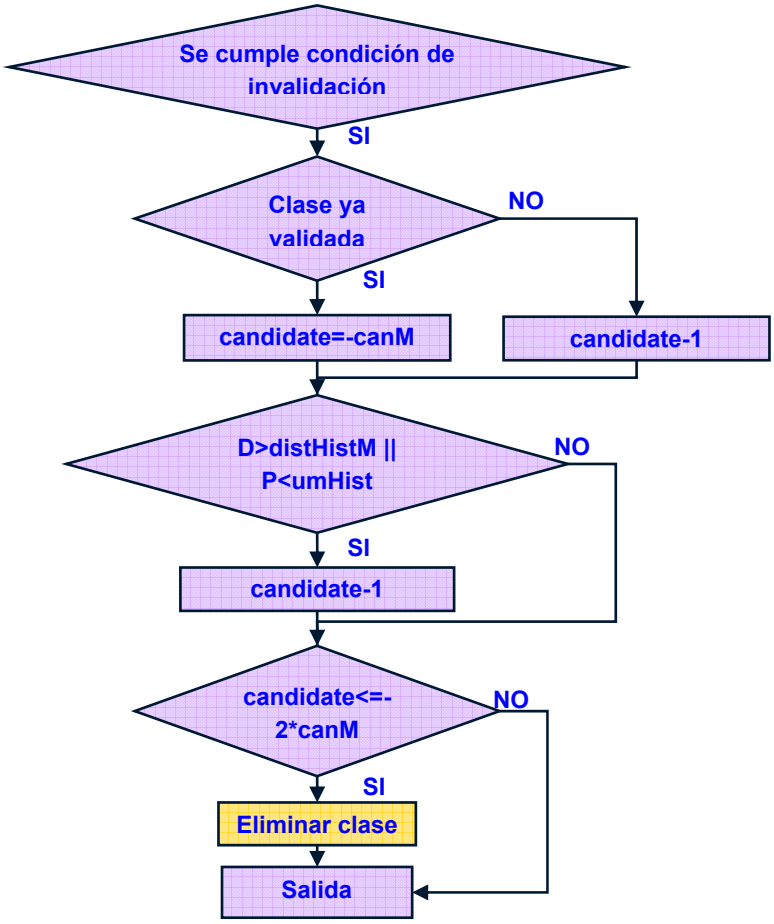


Figura 2.11. Diagrama de funcionamiento del proceso de invalidación de las clases.

El proceso de validación asociado al algoritmo de clasificación es semejante al del KFPDA, por lo que también hay que declarar la variable *canM* de la que depende dicho proceso. Evidentemente existe un *canM* para la función de clasificación de entrada y otro para la de salida.

El proceso de validación de las clases depende de dos características de las mismas:

1. Su proximidad en el espacio de clasificación a la clase más semejante de las obtenidas en el instante anterior $t-1$. La semejanza entre cada clase y cada una de las generadas en el instante anterior se refiere a la distancia euclídea entre el centroide de la clase y la predicción del de la clase.
2. Su verosimilitud $\beta_{j,t}$, $\beta_{j,1:t}$. Su función es valorar la densidad de datos asociados a cada clase. Para suavizar el efecto de la variable $\beta_{j,t}$ en el proceso de validación, se usa un factor de olvido *factOlv* normalizado para ponderar su valor en función de la evolución de la clase a lo largo del tiempo y obtener $\beta_{j,1:t}$ de la siguiente forma:

$$\beta_{j,1:t} = factOlv \cdot \beta_{j,t} + (1 - factOlv) \cdot \beta_{j,1:t-1} \,/\, j = 1 : k_t \tag{2.12}$$

El algoritmo de validación resultante mostrado en la Figura 2.10 es, por tanto, más complejo que el mostrado correspondiente al KFPDA (Figura 2.5).

En el XPFCP existen una serie de parámetros a ajustar para intentar mejorar los resultados obtenidos y analizar la influencia de cada uno de ellos en el comportamiento del algoritmo en cuestión.

Los parámetros a optimizar son los siguientes:

- *n*: Número de partículas totales.
- n_m: Cantidad de partículas que se introducen en el paso de reinicialización del filtro de Partículas.
- *cX*: Desviación típica del ruido del modelo.
- *cY*: Desviación típica del ruido de las medidas.

Además de éstos, existen otros parámetros correspondientes al K-Medias que ya han sido ajustados en un trabajo anterior ([Cerro07]), y son los siguientes:

- *hist*: Este parámetro influye en la validación de clases en función de la distancia que se desplazan los centroides.
- *factOlv*: La influencia de este parámetro sobre el clasificador radica en el cálculo de la probabilidad.
- *um*: Utilizado en la validación por probabilidad.
- *KLIKM*: Indica un punto a partir del cual se fija el valor de “um” en el proceso de validación.

2.3. Algoritmo "SJPDAF"

El "Filtro Probabilístico de Asociación de Datos Conjunta Muestreado" (SJPDAF, Sample-based of Joint Probabilistic Data Association Filter) ([Schulz03]) combina la ventaja de los estimadores con la eficiencia del enfoque para seguir múltiples objetos que proporciona el JPDA.

El método "Probabilístico de Asociación de Datos Conjunta (JPDA, Joint Probabilistic Data Association) es un método muy utilizado en el seguimiento de múltiples objetos en movimiento ([Karlsson02], [Pao94]).

Se define el JPDA como una extensión del PDA para múltiples objetos, donde se emplean tantos PDAs como objetos estén implicados en el proceso de asociación. Se incluye la probabilidad conjunta de existencia de las distintas hipótesis de asociación. Es decir, en lugar de $mj \times k$ valores de probabilidad se tienen $m \times k$, siendo mj el número de medidas asociadas a una clase, *k* el número de clases y *m* el número total de medidas en un instante determinado. Esta probabilidad se calcula según la siguiente ecuación:

$$\beta_{i,j} = \frac{1}{cY \cdot \sqrt{2 \cdot \pi}} \cdot e^{-0.5\frac{dist^2}{cY^2}}, \qquad (2.13)$$

donde *cY* es la desviación típica del ruido de las medidas y *dist* es la distancia de la medida i al centroide de la clase j.

Al igual que en caso del PDA es necesario conocer *a priori* el número de objetos *k* por lo que se emplea un proceso de *gating* que permite añadir nuevos objetos.

El tiempo de ejecución del JPDA es el mayor problema del algoritmo; aunque supone una carga computacional alta se usa en muchas aplicaciones ya que ofrece alta fiabilidad en sus resultados.

En los últimos años se han empleado algoritmos basados en el uso del JPDA junto con un PF ([Schulz01]) que recibe por ello el nombre SJPDAF.

El origen del SJPDAF se debe a que el JPDAF tiene problemas de divergencia en aplicaciones de seguimiento de objetos con un modelo de movimiento muy diferente al Gaussiano. En estos casos el uso de un filtro de Kalman Extendido (EKF) no asegura la convergencia del estimador. El PF está especialmente indicado en estos casos ya que es capaz de modelar cualquier tipo de comportamiento.

2.3.1. Explicación de las distintas versiones

En este trabajo se han desarrollado dos variantes del algoritmo SJPDAF, que se explican a continuación:

- Uno de ellos corresponde al SJPDAF propiamente dicho. Como estimador se cuenta con un PF y como proceso de asociación el JPDA, en el que todas las posibilidades de asociación son tenidas en cuenta a la hora del cálculo de la probabilidad de cada partícula. El peso de

las partículas w_i es, por lo tanto, calculado en función de todas las medidas existentes como se muestra a continuación:

$$w_i = \sum_{j=1}^{m} \beta_{j,i} \,, \tag{2.14}$$

donde m el número de medidas y n la cantidad de partículas.

- La otra versión desarrollada consiste en un PF junto con el JPDA, y además un *Nearest Neighbour* (NN). El peso de cada partícula w_i es calculado únicamente en función de la medida más cercana. Por tanto diremos que es un SJPDAF+NN.

$$w_i = \beta_{\max,i} \,, \tag{2.15}$$

donde *max* corresponde a la medida más próxima a la partícula i.

Al igual que en el XPFCP existe un proceso de clasificación de las partículas a la salida para conocer el número de objetos en todo momento, siendo también en este caso el clasificador escogido el K-Medias. Por otra parte en el XPFCP se tiene como organizador del set de partículas inicial al K-Medias mientras que en el SJPDAF se emplea un JPDA.

2.3.2. SJPDAFs utilizados

A continuación se muestra un diagrama funcional básico del comportamiento del SJPDAF. Este flujograma es prácticamente igual que el correspondiente al XPFCP, salvo que el proceso de asociación de entrada no es el K-Medias, sino el JPDA descrito en párrafos anteriores y cuya funcionalidad se muestra en la Figura 2.13 y la Figura 2.14:

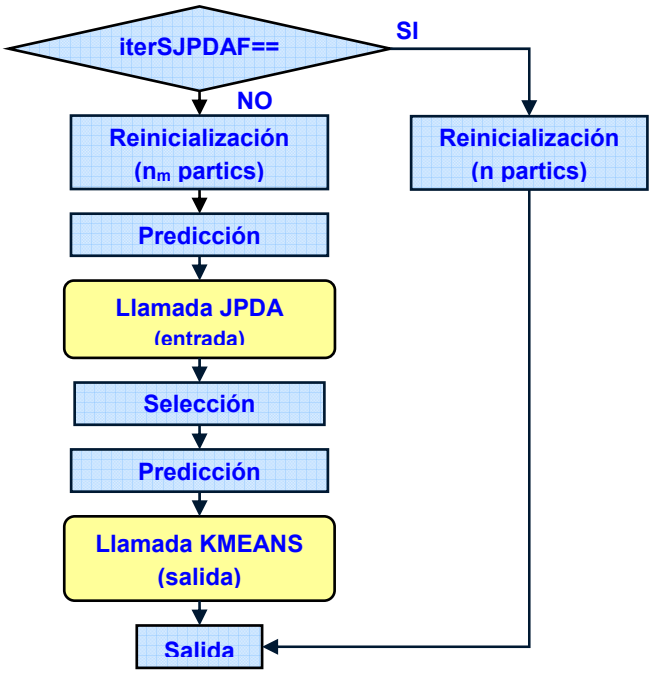


Figura 2.12. Flujograma del comportamiento del SJPDAF y del SJPDAF+NN.

Como se puede comprobar comparando la Figura 2.7 y la Figura 2.12, ambas son iguales salvo en el proceso de asociación de entrada, como se ha dicho previamente. Se observa que la etapa de corrección del XPF no aparece en el flujograma. Esto es debido a que para llevar a cabo dicha etapa son necesarios los pesos de las partículas, y al ser parte del proceso del SJPDA este cálculo se realiza dentro de la función de asociación. Esto en el XPCP no es así ya que los pesos son calculados en función de la distancia entre las partículas y los centroides de las clases.

El resto de procesos, como son el K-Medias o la validación e invalidación de clases no se detallan al ser exactamente igual a los correspondientes al XPFCP.

Se muestra a continuación, en la Figura 2.13 y Figura 2.14, el comportamiento del JPDA para el SJPDAF y SJPDAF+NN:

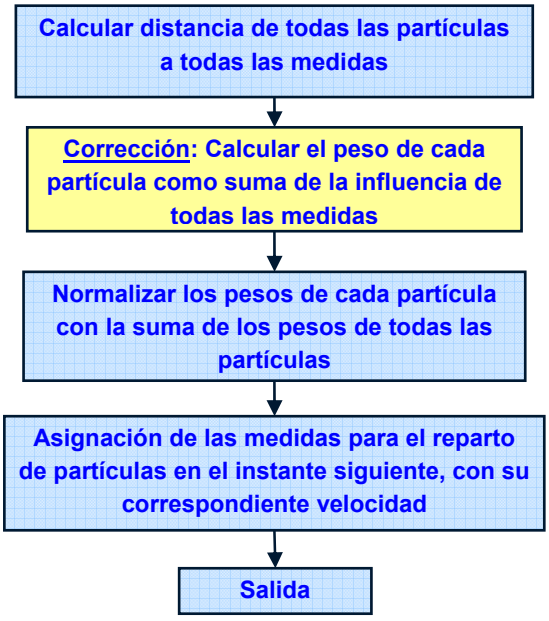


Figura 2.13. Diagrama funcional del algoritmo JPDA de asociación del SJPDAF.

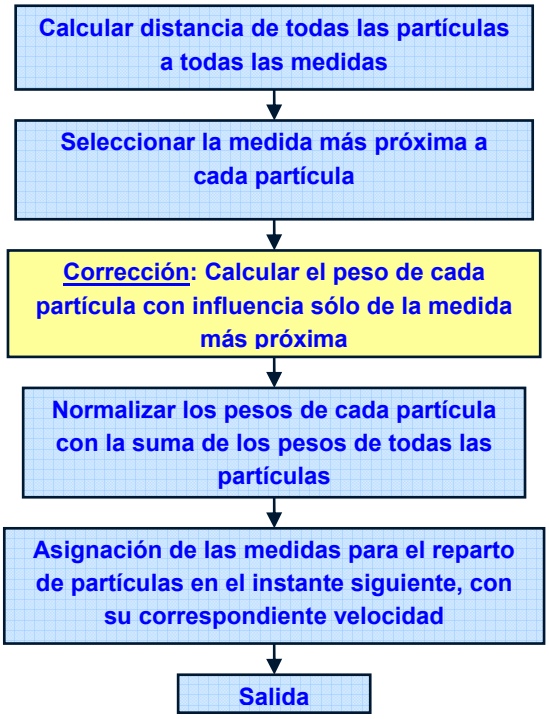


Figura 2.14. Diagrama funcional del algoritmo de asociación JPDA para el SJPDAF+NN.

Los parámetros que van a ser analizados posteriormente con el fin de comprobar la influencia en dichos algoritmos son los siguientes:

- *canM*: Parámetro empleado en el proceso de creación y eliminación de clases.
- *cY*: Desviación típica del ruido de las medidas.
- *cX*: Desviación típica del ruido del modelo.
- *n*: Número total de partículas.
- n_m: Número de partículas introducidas en el paso de reinicialización del filtro de Partículas.

CAPÍTULO 3. AJUSTE DE LOS PARÁMETROS

En este capítulo se detallan, una a una, cada una de las pruebas realizadas con el fin de ajustar los parámetros de los algoritmos descritos en el Capítulo 2. De este modo se intenta conseguir la mínima tasa de error posible en los experimentos detallados en el Capítulo 1 y analizar la sensibilidad de los algoritmos frente a cada uno de sus parámetros.

Para llevar a cabo la prueba de un determinado parámetro, se le asigna a éste un margen de valores tan amplio como sea necesario. De esta forma se comprueba de forma clara el valor que da lugar a un mejor comportamiento del algoritmo y las posibles divergencias que se generan en el seguimiento al asignarle valores excesivamente bajos o altos. Una vez ajustado cada parámetro, su valor será fijo para el resto de las pruebas.

El orden de las pruebas es tal que los primeros parámetros a ajustar son los de mayor influencia en el algoritmo.

3.1. Ajuste de los parámetros correspondientes a “KFPDA”

3.1.1. Pruebas a realizar específicas

Las pruebas a realizar para el ajuste de los parámetros del algoritmo KFPDA enumeradas en el capítulo anterior, son las siguientes y se hacen en el siguiente orden:

1. *distMKF:* Límite de *gating* en el proceso de asociación PDA por lo que se usa en el cálculo de la probabilidad de que una medida sea asociada a un determinado objeto (ver

ecuación (2.5) del capítulo 2) y establece qué medidas son asignadas a cada uno de ellos y en qué momento hay que crear uno nuevo.

2. *canM*: Parámetro que se emplea únicamente en el proceso de validación e invalidación de los filtros. El valor de *canM* indica el número de iteraciones necesarias para validar o invalidar un objeto.
3. *dtR*: Desviación típica del ruido de las medidas que se emplea en el cálculo la ganancia de Kalman (ver Figura 2.1).
4. *dtQ*: Desviación típica del ruido del modelo que se utiliza en el cálculo de la matriz de covarianza del error de estimación (*P_predicted*).
5. P_o: Matriz de covarianza del error de estimación inicial. Este parámetro influye en el cálculo de la matriz de covarianza del error de estimación corregida (*P_corrected*) al crearse un nuevo filtro.

Para comenzar las pruebas es necesario asignar a los parámetros unos valores iniciales que se muestran en la Figura 3.1:

distMKF=700	***canM=4***	***dtR=*** $\sqrt{0.1}$	***dtQ=1***	$P_o = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

Figura 3.1. Valores iniciales de los parámetros del KFPDA.

3.1.2. Comentarios respecto a las pruebas realizadas

1ª Prueba: distMKF

Como se ha comentado anteriormente, este parámetro es de gran importancia pues define el proceso de *gating* explicado. Dependiendo de su valor, una medida será asociada o no a un determinado objeto. Por otro lado también influye en cálculo de la probabilidad de que dicha medida pertenezca a ese objeto.

Para su correcto ajuste se ha dado al parámetro unos valores comprendidos entre 400 milímetros (mm.) y 1000 mm. con incrementos de 100 en 100. Es decir, con *distMKF*=1000 se asociarían a un objeto todas las medidas que se encuentren a su alrededor en un radio de 1 metro. Por esto no se prueba con menores o mayores valores ya que no tiene sentido al tratarse la mayor parte de los objetos de personas.

Como se explicó en el Capítulo 1 las medidas de los objetos corresponden al contorno de los mismos. Teniendo en cuenta esto y que se ha modelado a cada objeto como un cilindro es lógico pensar que con valores pequeños la asociación no será la esperada ya que sólo se asociarán las medidas que estén muy próximas. Esto tendrá como consecuencia que se creen gran cantidad de filtros distintos en situaciones donde sólo se debe crear uno.

En la Figura 3.2 y la Figura 3.3 se muestra el porcentaje de errores de los dos experimentos en función de los distintos valores de *distMKF*:

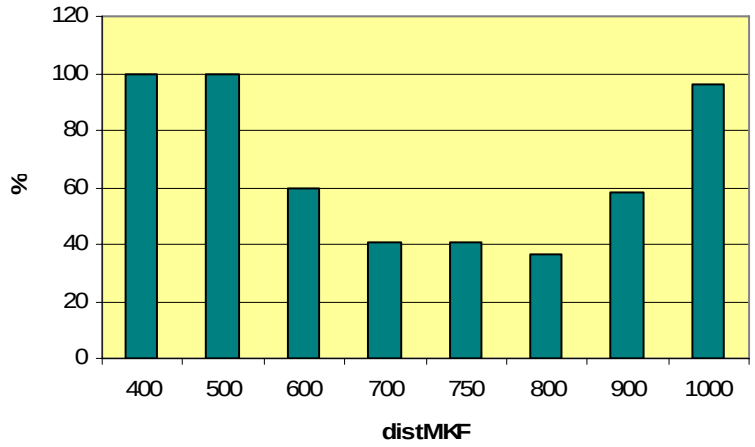


Figura 3.2. Evolución de la tasa de error en la prueba corta en función de distMKF.

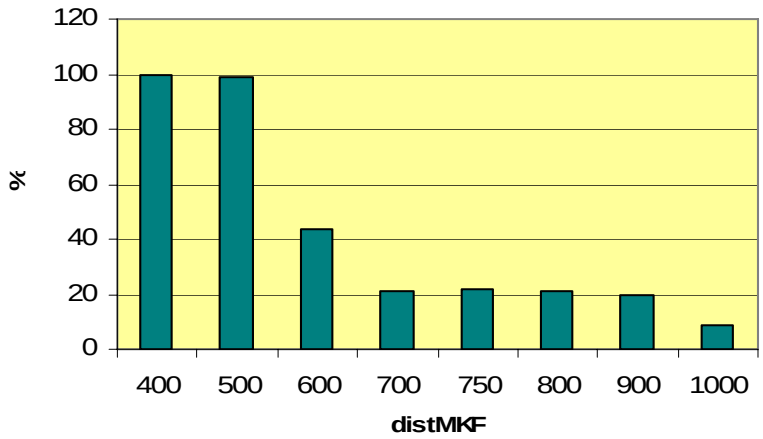


Figura 3.3. Evolución de la tasa de error en la prueba larga en función de distMKF.

Con valores de *distMKF* entre 600 y 800 mm. el número de errores disminuye notablemente, como a la mitad. La mayor parte de los errores obtenidos son de tipo “duplicado”.

Si se aumenta *distMKF* a 900 mm. y 1000 mm. ocurren cosas distintas en los dos experimentos analizados como se puede comprobar en la Figura 3.2 y en la Figura 3.3. En la prueba corta el número total de errores aumenta considerablemente debido a la aparición de numerosos fallos “unión de dos”, mientras que en la prueba larga la tasa de error se mantiene prácticamente constante. Este resultado es lógico ya que la prueba corta es la más compleja como se ha dicho en otras ocasiones, mientras que en la prueba larga hay menos situaciones adversas. Si se analizan los dos experimentos en conjunto, se observa que con estos dos valores de *distMKF* el número de errores ha aumentado respecto al obtenido con valores inferiores.

A la vista del análisis expuesto se fija el valor de *distMKF* a 800 mm. al obtener con él la menor tasa de error, un 26.16%.

En la Figura 3.4 se comprueba cómo cambian y mejoran los resultados al ir aumentando el valor de *distMKF* de 400 mm. a 800 mm.:

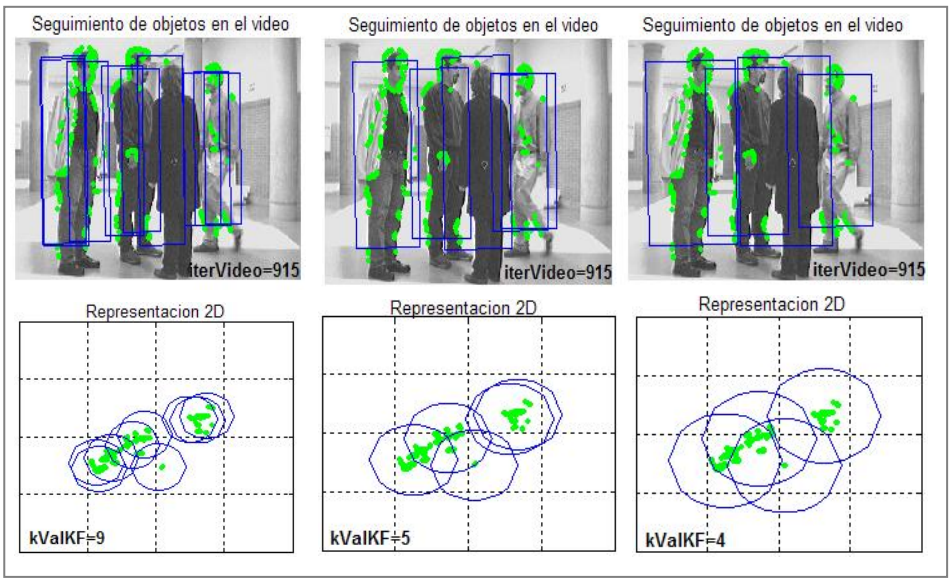


Figura 3.4. Resultados de seguimiento obtenidos con distMKF igual a 400 (izquierda), 600 (centro) y 800 (derecha).

En la Figura 3.4 se ve que a medida que se aumenta el valor de *distMKF*, el número de filtros validados disminuye considerablemente, hasta que con un valor de *distMKF* igual a 800 mm. se obtienen los resultados más correctos.

2ª Prueba: canM

Como se ha comentado con anterioridad, *canM* es un parámetro que influye únicamente en los procesos de validación e invalidación de filtros.

Para realizar el estudio completo se ha asignado un margen de valores de 0 a 5, ambos inclusive, a este parámetro. Dar algún valor mayor no tiene sentido ya que el número de errores crece de forma considerable a partir de *canM* =5.

Si *canM*=0 cualquier objeto al que se asocien medidas será validado en una sola iteración. Del mismo modo, en el instante en que no se le asocien medidas, éste será eliminado.

Con valores pequeños de *canM* (0 y 1) se producen muchos errores “unión de dos” (13% y 11% respectivamente). Cuando a un objeto no se le asocian medidas, el filtro que realiza su seguimiento es eliminado inmediatamente o en la siguiente iteración. En el momento en que reaparecen sus medidas, éstas suelen ser asociadas al objeto vecino debido a que hay varias personas muy próximas en los vídeos. A medida que se va aumentando el valor de *canM*, este error se reduce considerablemente.

En la Figura 3.5 se observa con más detalle la evolución en el porcentaje de iteraciones con error debido a los fallos más significativos en el experimento corto:

	canM=0	***canM=1***	***canM=2***	***canM=3***	***canM=4***	***canM=5***
Unión de dos	*19.8*	*11.9*	*8.9*	*8.9*	*8.9*	*8.9*
Duplicados	*0*	*25.8*	*7.9*	*13.9*	*9.9*	*9.9*
No generados	*0*	*4*	*9.9*	*8.9*	*13.9*	*16.9*
Otros	*12.2*	*9.3*	*10.3*	*5.3*	*4.3*	*8.3*
Total	***32***	***51***	***32***	***37***	***37***	***44***

Figura 3.5. Tabla con un resumen del porcentaje de errores obtenidos en el ajuste del parámetro canM en la prueba corta.

Los valores muy grandes de *canM* dan lugar a "desplazamientos", principalmente en objetos móviles. Hay objetos que aparecen en la escena muy próximos entre sí y al ser la validación más lenta, sus medidas son asociadas a algún objeto de su alrededor. De esta forma, en el momento en que se cree el filtro de este nuevo objeto estará desplazado al asociarse parte de sus medidas a otro. En cambio, asignando a *canM* valores 2 y 3 esta tasa de error es menor.

Otro de los errores destacados es el "duplicado" de objetos. La cantidad de estos errores es máxima para *canM*=1.

Por otra parte, a medida que se incrementa el valor de *canM* la cantidad de errores "no generado" aumenta de forma importante, ya que este valor es el que permite que se produzca la validación e invalidación del filtro en una iteración o en otra.

Un aspecto muy importante a estudiar es en qué medida afecta este parámetro a la sensibilidad del algoritmo respecto a los ruidos.

Al realizar este análisis se obtiene las siguientes conclusiones:

- Si *canM* tiene un valor pequeño, se considera el ruido de la columna como un objeto; no se filtra y se valida. La parte positiva es, que de la misma manera, se procederá a su eliminación con prontitud en el momento de no asociársele medidas. De esta forma, en un 20% de la prueba larga aparecen ruidos sin filtrar con valores de *canM* 0, 1 y 2.
- Si *canM* tiene valores mayores de 2 se produce el efecto contrario en el filtrado de los ruidos, algo positivo. Pero por el contrario no hay ventajas en cuanto al proceso de eliminación. Como se ha comentado antes, aparecen constantemente medidas de la columna, por lo que es difícil que el filtro sea eliminado. Como consecuencia de esto, el ruido de la columna aparece sin filtrar en un 30% de la prueba larga para *canM* 3, 4 y 5.

A partir de todas estas apreciaciones se concluye que el valor óptimo de *canM* es en este caso 2. La tasa de error obtenida con este valor es de un 18%. Por lo tanto se escoge éste como el valor empleado de *canM* para continuar con el resto de las pruebas.

En la Figura 3.6 se muestra una iteración extraída del vídeo en las que *canM* se fija a 0 y a 2. Se observa que en el primer caso aparece sin filtrar el ruido de la puerta del fondo mientras que en el segundo no. El ruido de la columna al tener una frecuencia mayor aparece sin filtrar en ambos casos. Por último, el filtro de la papelera sólo aparece validado con *canM*=2, aún no teniendo medidas.

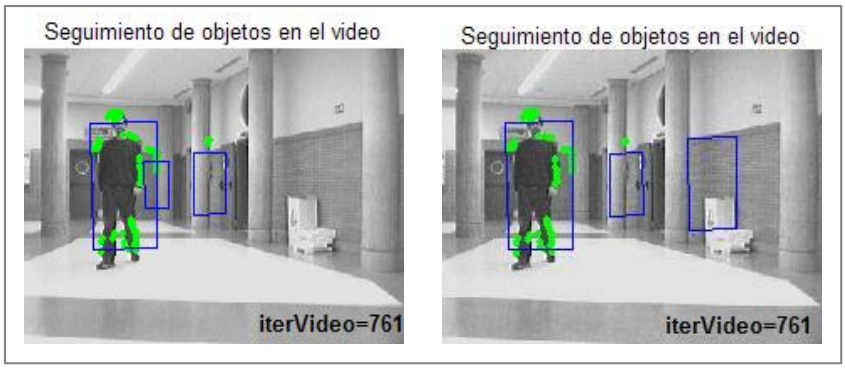


Figura 3.6. Imágenes en las que se puede observar los diferentes filtros validados con canM= 0 (izquierda) y canM=2 (derecha) en el experimento de ruido.

3ª Prueba: dtR

Este parámetro corresponde a la desviación típica del ruido de las medidas, y por tanto la matriz *R* es la matriz de covarianza del ruido de las medidas. La matriz *R* tiene un papel importante en el filtro de Kalman pues se usa en el cálculo de la ganancia de Kalman a través de la siguiente expresión:

$$K_{t+1} = P_{t+1|t} \cdot H^T (H \cdot P_{t+1|t} \cdot H^T + R)^{-1}, \text{ donde } R = \begin{pmatrix} dtR_x^2 & 0 \\ 0 & dtR_z^2 \end{pmatrix}, \tag{3.1}$$

en el modelo elegido que se puede ver en la ecuación (2.1) del Capítulo 2.

- Si los sensores con los que se trabaja son fiables y precisos, se da un valor pequeño a *R*. Si *dtR* tiene valores pequeños la ganancia de Kalman aumenta. Como consecuencia de esto, con menores valores de *dtR*, el filtro responde más rápidamente, se estabiliza antes.
- En caso contrario, si se considera que las medidas no van a ser muy precisas, se le asigna valores mayores. Si *dtR* tiene valores grandes la ganancia de Kalman disminuye y el filtro tarda más en llegar a ser estable.

Se van a realizar tres estudios distintos para ajustar el valor de esta matriz:

1. Calcular en cada iteración la varianza de cada uno de los filtros existentes, en *x* y en *z*, de la siguiente forma: $\text{var} = \frac{1}{mj} \cdot \sum (M - C)^2$.
2. Calcular la varianza en cada iteración en *x* y en *z*, y dividiéndolo entre $distMKF^2$ (se emplea el valor ajustado en el estudio anterior, es decir, 800 mm.) al comprobar que en el primero de los casos el filtro diverge y hacerse necesario una normalización de estos

valores de varianza. Se divide la varianza entre $distMKF^2$ al ser esta distancia la empleada para modelar los objetos mediante un cilindro. El cálculo queda de la siguiente manera: $\text{var} = \frac{1}{distMKF^2} \cdot \frac{1}{n_m} \cdot \sum (M - C)^2$.

3. Dar valores fijos a la varianza a lo largo de todo el vídeo, asignando los mismos valores a varianzas en *x* y en *z* con el fin de comprobar la semejanza, si es que la existiera, entre estos valores y los teóricos obtenidos el anterior estudio.

En el la Figura 3.7 se puede ver la tasa de error para cada caso de varianza y se observa con claridad cuáles son los casos más favorables:

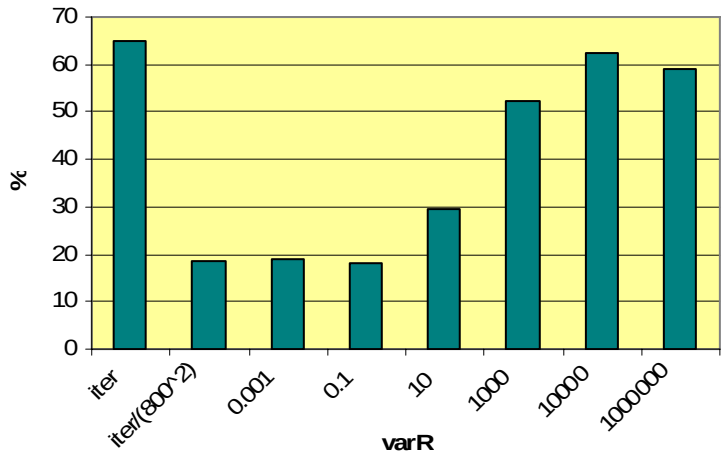


Figura 3.7. Gráfico con la evolución en el porcentaje de error para las distintas opciones de varianza del ruido de las medidas.

Al completar el primero de los estudios (caso 1) se hace una media de los valores de la varianza de todos los objetos en la coordenada *x* y en la *z* para obtener así un valor sea orientativo. Los valores obtenidos son 20480 mm^2 en la coordenada *x* y 8320 mm^2 en la coordenada *z*. Es lógico que la varianza en *z* de las medidas sea menor que en la coordenada *x* ya que, como se ha comentado en otras ocasiones, las medidas corresponden al contorno de los objetos y su proyección en el plano *XZ* toma la forma aproximada de una elipse.

Se producen muchos errores de "desplazado". Esto es debido a que en muchas ocasiones habiendo dos o más personas próximas, cada una con su correspondiente filtro, alguna de ellas cambia su dinámica, por lo que la predicción no se corresponde con la medida. Como consecuencia de esto la persona pasa a tener dos filtros asociados, produciéndose un duplicado y la pérdida del estimador por parte de la otra; siendo ésta la causa también de la gran cantidad de errores "duplicado". Estos objetos duplicados suponen el 60% de la totalidad de fallos.

Todos estos errores son debidos a que *dtR* tiene unos valores muy elevados, el ruido es demasiado grande en comparación con el real y el filtro diverge. Por esto surge la idea de minimizar la varianza del ruido de las medidas dividiendo estos valores por $distMKF^2$.

En el segundo estudio (opción 2), donde $dtR_{x'}^2 = \frac{dtR_x^2}{800^2}$ y $dtR_{z'}^2 = \frac{dtR_z^2}{800^2}$, se obtiene una gran mejoría con respecto al caso anterior, pasando el porcentaje total de error del 65% al 19.20%.

Para el tercer estudio (caso 3), se ha asignado a la varianza del ruido de las medidas $\left(dtR^2\right)$ valores de 0.001, 0.1, 10, 1000, 10000 y 1000000, todos ellos en milímetros cuadrados (mm^2). Con estos valores se obtiene unas tasas de error de 18.80%, 18.20%, 29.50%, 52.31%, 62.60% y 58.94% respectivamente.

El comportamiento del algoritmo con los valores 0.001 y 0.1 es muy semejante. La única gran diferencia es que con dtR^2=0.001 aparecen más errores "desplazado, mientras que con dtR^2=0.1 los fallos son de "duplicado".

Finalmente se elige el valor de 0.001 para continuar con las pruebas puesto que el error de "desplazado" es menos relevante que el de "duplicado".

Es necesario explicar que con la opción de calcular la varianza por cada iteración y dividirla por $distMKF^2$ se obtienen también buenos resultados y prácticamente la misma cantidad de errores que con dtR^2=0.001 pero, al igual que en el caso anterior, hay más objetos duplicados. Los resultados son semejantes porque, como se puede ver en la Figura 3.8, ambos valores no difieren demasiado.

Una vez hecho este ajuste se compara la opción 2 y la opción 3 para comprobar que, en efecto, los valores obtenidos en ambos casos no son muy diferentes. El resultado de esta comparación se muestra en la Figura 3.8. Se puede observar el valor medio calculado de la varianza en *x* y en *z*, y por otra parte el valor final que se ha asignado a este parámetro.

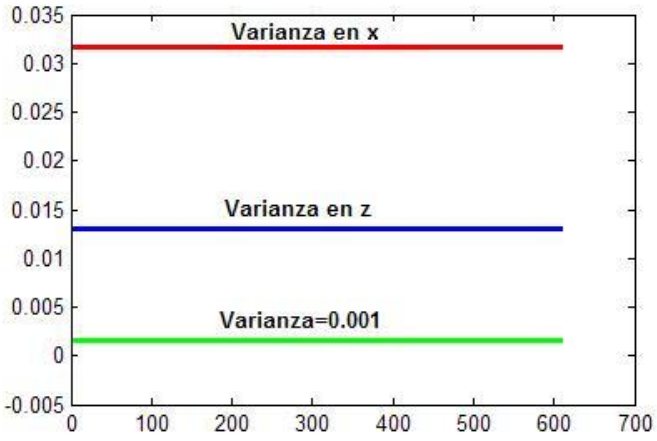


Figura 3.8. Gráfico con los valores medios de la varianza calculados en x y z, y su comparación con el valor asignado tras el ajuste.

Se ve que la varianza en *x* es mayor que la calculada en *z*, es decir, los datos están más dispersos en la coordenada *x* como se ha anunciado con anterioridad.

4ª Prueba: dtQ

La magnitud de la matriz *Q* indica la incertidumbre del modelo de estimación usado. Como se observa en la Figura 2.1, si aumenta el valor de la covarianza del ruido del sistema *Q*, donde $Q = \begin{pmatrix} dtQ_x^2 & 0 \\ 0 & dtQ_z^2 \end{pmatrix}$, aumenta también la covarianza del error de predicción, es decir, aumenta la incertidumbre del valor estimado. Esto significa que hay que hacer este valor grande para obtener buenos resultados si se tiene un modelo que no es conocido.

Para ajustar el valor de esta matriz empíricamente se ha asignado a las componentes en *x* y en *z* valores iguales, y estos han sido: 0.01, 1, 10 y 100, 1000 y 1000000 mm.

A continuación, en la Figura 3.9 se muestran los datos más importantes de esta prueba, donde se puede ver la poca variación en la tasa de error entre unos casos y otros. Esta tabla corresponde al estudio realizado en la prueba corta del vídeo 10.

	varQ=0.01	***varQ=1***	***varQ=10***	***varQ=100***	***varQ=1000***	***varQ=1000000***
Unión de dos	*8.9*	*8.9*	*8.9*	*8.9*	*8.9*	*8.9*
Duplicados	*5.9*	*0*	*0*	*0*	*0*	*0*
No generados	*5.9*	*5.9*	*9.9*	*5.9*	*5.9*	*6.9*
Desplazamientos	*0*	*10.9*	*7.9*	*7.9*	*9.9*	*12.9*
Otros	*12.3*	*3.3*	*2.3*	*7.3*	*7.3*	*6.3*
Total	***33***	***29***	***29***	***30***	***32***	***35***

Figura 3.9. Tabla resumen de las tasas de error en función de la varianza del ruido del modelo.

Con dtQ^2=0.01 se obtiene una tasa de error de 18.54%. Los errores más significativos son "duplicado", "no generado" y "unión de dos". Los objetos duplicados aparecen sólo en la prueba corta, proporcionando una tasa de error del 6%. El error de "no generado" es más frecuente encontrarlo en esta prueba, dando lugar a un 6% y 7.50% de iteraciones con fallos en ambas pruebas del vídeo a analizar. Por último se obtiene un porcentaje de error de 9% y 3.50% debido a errores de "unión de dos" en las dos pruebas del vídeo 10.

Con los valores de 1 y 10 se obtiene el mismo número de errores, siendo la tasa de fallo de 17.21%. Con dtQ^2=1 aparecen más desplazamientos.

Por último, si se asigna a dtQ^2 un valor de 100, 1000 ó 1000000, se comprueba que el número de desplazamientos va en aumento con respecto a los casos anteriores. El porcentaje de error es de 17.54%, 18.21% y 19.20% respectivamente, siendo los nuevos errores de "desplazado".

Finalmente se elige dtQ^2=10 para seguir adelante con el último ajuste. La elección de este valor se debe al hecho de que con él se producen menos desplazamientos, aunque el número total de errores sea el mismo que con dtQ^2=1. Si se mira la Figura 3.9 parece más lógico escoger dtQ^2=1 al ser menor la cantidad de errores "no generado", pero el total de errores de este tipo es menor en el caso de dtQ^2=10.

5ª Prueba: Ajuste Po

Se procede a ajustar el valor inicial la matriz de covarianza del error de predicción P_{0t}, donde $P_{0t} = \begin{pmatrix} P_0 & 0 \\ 0 & P_0 \end{pmatrix}$ que refleja la incertidumbre del valor de la estimación inicial. Este valor no debe ser excesivamente pequeño ya que tardaría demasiado tiempo en converger el filtro, pero si es demasiado grande puede hacer que nunca converja.

En la Figura 3.10 se muestra un diagrama de barras con la evolución del porcentaje de errores al variar este parámetro, donde se observa que prácticamente ocurre lo mismo con todos los valores probados:

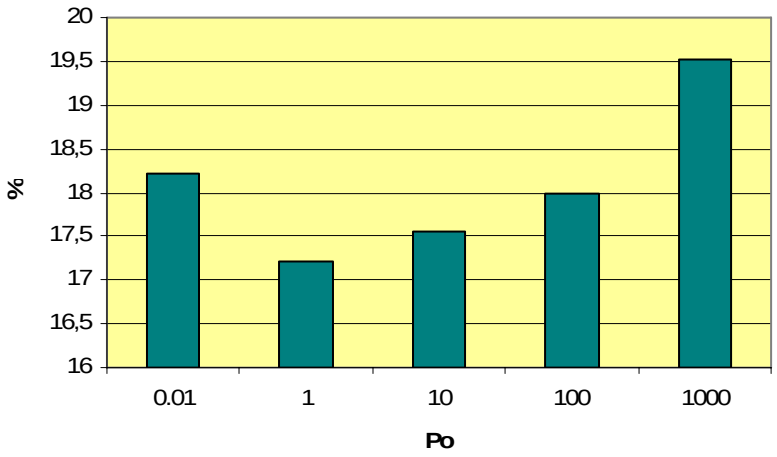


Figura 3.10. Tasas de error en función de Po.

Para ajustar el valor de esta matriz se han asignado a P_o los siguientes valores: 0.01, 1, 10, 100 y 1000.

El número de errores obtenidos es prácticamente el mismo en todos los casos, variando la tasa de error entre 17.22% y 19.53%. Aparte de los errores propios de la validación, aparecen algunos de "desplazado".

Como valor óptimo se escoge P_o=1, con un porcentaje de fallo del 17.22%, el mínimo que se ha hallado.

3.1.3. Conclusiones acerca del ajuste de "KFPDA"

Una vez realizados el ajuste de todos los parámetros del algoritmo KFPDA se llegan a las siguientes conclusiones:

- El valor del parámetro *distMKF* es fundamental en este filtro, y concretamente en el proceso de validación. El algoritmo es muy sensible al cambio del valor de esta distancia. Con valores demasiado pequeños los errores son muy frecuentes, prácticamente uno por

iteración, al llevarse a cabo la asociación de forma incorrecta. A partir de 600 mm. la cantidad de errores disminuye de forma drástica, y a medida que se aumenta el valor siguen disminuyendo de forma gradual. Con distancias mayores que 900 mm. se producen de nuevo multitud de fallos en la asociación. El valor escogido es de 800 mm., al ser con el que mejores resultados se obtienen y ser la tasa de error de un 26%.

- El parámetro *canM* es quizá el que mayor importancia tenga en este algoritmo al ser el responsable de que un filtro quede validado o invalidado en un momento o en otro. Se ha comprobado al realizar las pruebas que la cantidad de errores es menor para valores intermedios y mayor para valores muy bajos (0 ó 1) o altos (5). Por otra parte este parámetro es fundamental en la validación errónea de ruidos; si el valor de *canM* es muy pequeño los ruidos existentes se validan con facilidad, y si es demasiado grande no se invalidan con prontitud y aparecen en demasiadas iteraciones. Finalmente el valor escogido es *canM* =2, con el que se obtiene una tasa de error del 18.20%.

- La modificación de la varianza del ruido de las medidas (dtR^2) produce grandes variaciones en los resultados obtenidos con el KFPDA. Si se calcula la varianza en cada iteración de cada uno de los objetos el filtro diverge al ser este valor demasiado grande en relación al ruido real de las medidas. Si la varianza calculada es dividida por $distMKF^2$ se obtienen buenos resultados. Por último se observa que a medida que se aumenta el valor del parámetro el número de errores crece gradualmente porque, como se ha comentado anteriormente, el ruido de las medidas es demasiado grande en relación al real. El valor final que se ha escogido es dtR^2=0.001, semejante a la media de la varianza calculada en cada iteración. La elección de este valor no mejora la tasa de error del algoritmo, pero sí hace que los errores hallados sean menos importantes.

- La modificación de la varianza del ruido del modelo (dtQ^2) no produce prácticamente ninguna variación en los resultados obtenidos hasta el momento de su ajuste. El valor final se ha establecido como dtQ^2=10, con el que la tasa de error del KFPDA baja ligeramente hasta alcanzar el 17.20%.

- La variación de valor inicial de la matriz de covarianza del error de estimación P_o no produce grandes mejoras en el funcionamiento conseguido hasta el momento. Este parámetro mantiene su valor tras el ajuste, es decir, P_o=1, al ser el que mejor resultados nos reporta. La tasa de error final del KFPDA se mantiene en el 17.20%.

Por lo tanto el **porcentaje de fallo** tras todos los ajustes es de un **17.20%.**

Los valores finales de los parámetros del KFPDA son los mostrados en la Figura 3.11:

distMKF=800	***canM=2***	***dtR=*** $\sqrt{0.001}$	***dtQ=10***	$P_o = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

Figura 3.11. Valores finales de cada parámetro después de los ajustes correspondientes.

En la Figura 3.12 se ve de forma gráfica cómo ha ido mejorando la tasa de error tras cada uno de los ajustes:

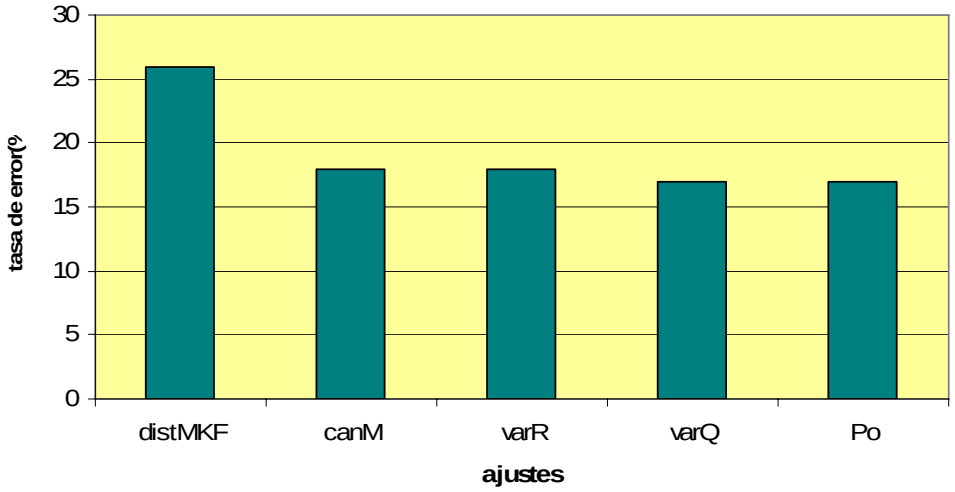


Figura 3.12. Gráfico con la tasa de error en % después del ajuste de cada uno de los parámetros.

3.2. Ajuste de los parámetros correspondientes a "XPFCP"

3.2.1. Pruebas a realizar específicas

Los parámetros a ajustar son los que ya se han explicado en el Capítulo 2, pero sólo los detallados a continuación ya que el resto, correspondientes al proceso de clasificación, han sido ya ajustados en un trabajo anterior ([Cerro07]). Las pruebas se van a realizar en el siguiente orden:

1. *n*: Este parámetro representa el número total de partículas que se emplean a lo largo de todo el proceso de estimación generado por el PF. Por lo tanto su modificación tiene efecto en la totalidad de las fases del algoritmo.
2. n_m: En el paso de reinicialización del PF se inyectan n_m partículas nuevas. De esta forma es más probable que a un objeto nuevo se le asocien partículas. Este parámetro influye en el proceso de selección y de reinicialización.
3. cX: Desviación típica del modelo Gaussiano de ruido asociado al vector de estado. Este parámetro está implicado en el cálculo del ruido que se sumará a la predicción del estado de cada partícula.
4. cY: Desviación típica del ruido de las medidas. En el cálculo de los pesos de cada una de las partículas interviene este parámetro, por lo tanto será fundamental para que las partículas sobrevivan al paso de selección.

Al igual que en el KFPDA se realizan la prueba corta y larga del vídeo 10 para ajustar los parámetros del XPFCP.

El estimador de la posición en este algoritmo es un filtro de Partículas y por lo tanto en cada iteración, aparte de las medidas y las partículas, se visualizan las clases generadas a la salida. Por una parte se analizan los errores en el proceso de asociación de las partículas y por otra los errores en la clasificación en clases de las partículas a la salida. Se tienen entonces dos estudios diferentes que se comentan en cada uno de los ajustes.

El método empleado para realizar el análisis es el mismo que el usado en el análisis del KFPDA. Se parte de unos valores iniciales para cada parámetro que se modifican a medida que se estudia la sensibilidad del algoritmo frente a cada uno de ellos. Se ajusta cada parámetro de forma independiente, dejando fijos el resto y buscando el valor con el que se obtiene un funcionamiento óptimo del estimador.

Los valores iniciales de cada parámetro se muestran en la Figura 3.13:

n=250	***n_m=50***	***cX=70***	***cY=70***

Figura 3.13. Valores iniciales de los parámetros del XPFCP.

3.2.2. Comentarios respecto a las pruebas realizadas

1ª Prueba: n

Como se ha comentado anteriormente, este parámetro indica el número total de partículas que se emplean en el filtro de Partículas. Para comprobar cómo influye el número de partículas en los resultados obtenidos del algoritmo se ha asignado a *n* valores comprendidos entre 100 y 1000, con intervalos de 100.

En el proceso de asociación de partículas, la mayor parte de los errores son “no generado”, que en este proceso implica que un objeto del que se ha sensado alguna medida no tiene asociada ninguna partícula. Este error se mantiene prácticamente constante en el estudio, es decir, pese a aumentar la cantidad de partículas totales, el número de errores sigue siendo el mismo. La tasa de error es de aproximadamente un 40%.

Prácticamente todos los errores encontrados en el proceso de clasificación de partículas son del tipo “no generado”, es decir, a la salida del clasificador de partículas no aparece una clase validada. En la prueba corta la cantidad de errores no sufre muchas fluctuaciones al ser una prueba más sencilla, y la tasa de fallo total se mantiene alrededor del 90%. Como se puede ver en la Figura 3.15 en la prueba larga ocurre algo distinto al ser ésta más compleja y darse situaciones más adversas.

En la Figura 3.14 y la Figura 3.15 se muestra la evolución del porcentaje de errores de los dos experimentos comentados, tanto a la salida del PF como a la salida del clasificador de partículas:

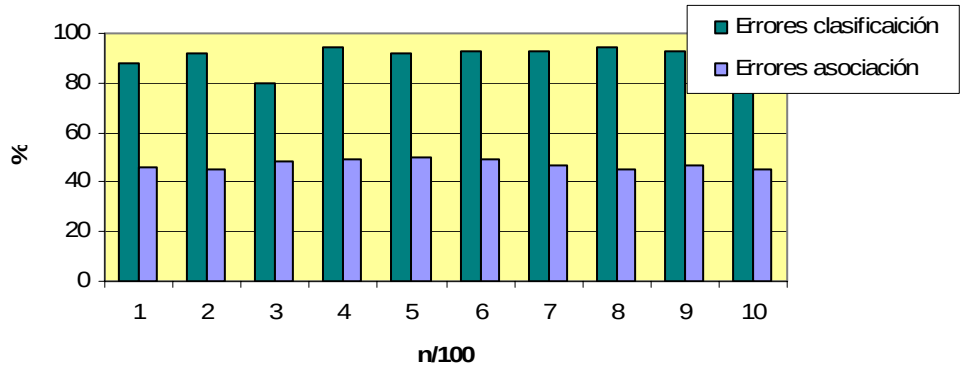


Figura 3.14. Evolución de porcentajes de error en la prueba corta en función del número de partículas.

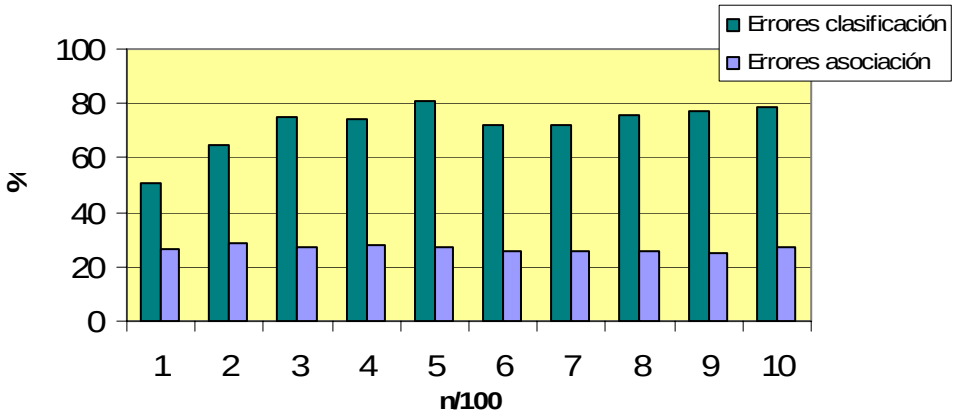


Figura 3.15. Evolución de los porcentajes de error en la prueba larga en función del número de partículas.

El resto de errores generados por el algoritmo de clasificación de partículas, además de ser mínimos en cuanto a su número, no son de gran importancia; es decir, son fallos del tipo "desplazado".

Se ha comprobado cómo crece el tiempo de ejecución del XPFCP al aumentar el número de partículas. A medida que aumenta el número total de partículas el tiempo de ejecución crece como se puede observa en la Figura 3.16:

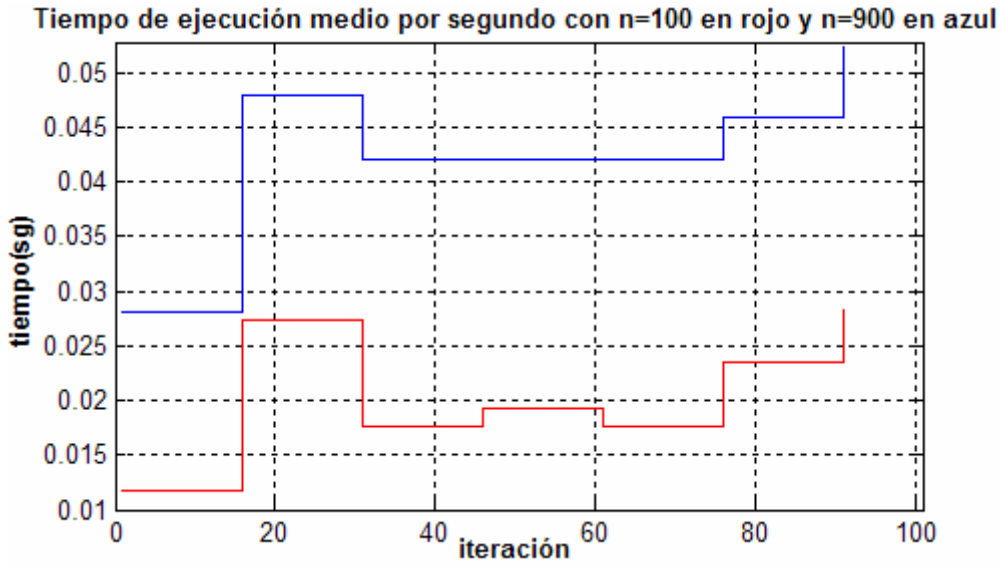


Figura 3.16. Tiempo medio de ejecución de XPFCP en la prueba corta en función del número de partículas.

Para continuar con el resto de las pruebas se escoge un número de partículas de 100, ya que con este valor se obtiene la menor cantidad de errores posibles. La tasa de error con n= 100 es del 33% y del 63% en la asociación de partículas y en la clasificación de las mismas a la salida respectivamente.

2ª Prueba: n_m

El parámetro n_m indica el número de partículas que se insertan en el paso de reinicialización del PF, tal y como se explica en el Capítulo 2.

Con el fin de ajustar n_m, y una vez fijado el valor de n a 100 partículas, se modifica su valor desde el 10% hasta el 90% de éste; es decir, de 10 a 90 partículas.

En la Figura 3.17 y la Figura 3.18 se observa la evolución en la tasa de error tanto en el proceso de asociación como en el de clasificación de partículas:

	n_m=10	n_m=30	n_m=50	n_m=70	n_m=90
No generados	44.70	35.10	31.79	32.78	45.70
Otros	2.32	1.32	1.32	2.32	2.64
Total	**47.02**	**36.42**	**33.11**	**35.10**	**48.34**

Figura 3.17. Tabla con un resumen de las tasas de error en el proceso de asociación de partículas.

La cantidad de fallos en la asociación de partículas disminuye según aumenta n_m y hasta que alcanza este parámetro un valor del 70% de n. Con n_m=80 vuelven a aumentar los errores, y con n_m=90% de n se obtiene el menor número de errores. La cantidad de fallos del proceso de clasificación a la salida disminuye a medida que aumenta el valor de n_m.

	n_m=10	n_m=30	n_m=50	n_m=70	n_m=90
Unión de dos	*15.20*	*16.88*	*23.51*	*22.51*	*20.53*
Desplazados	*0*	*1.00*	*3.00*	*3.32*	*4.30*
No generados	*43.70*	*45.70*	*32.45*	*26.50*	*27.50*
Otros	*24.21*	*4.64*	*4.28*	*5.95*	*12.57*
Total	***83.11***	***68.22***	***63.24***	***58.28***	***64.90***

Figura 3.18. Tabla con un resumen de las tasas de error en el proceso de clasificación de partículas a la salida.

Como primera observación se destaca que a medida que aumenta el valor de n_m la validación e invalidación de clases de partículas se realiza más correctamente, es lógico ya que los nuevos objetos tienen mayor probabilidad de que les sean asignadas partículas.

Con valores de n_m del 80% y del 90% de *n* aparecen multitud de errores "desplazado" de clases. Si se analizan estos casos en los vídeos se comprueba que en el momento en que aparece un nuevo objeto, al ser tan elevado el valor de n_m, las clases anteriormente validadas apenas mantienen partículas pues se eliminan en el proceso de selección. Esto provoca que se produzcan los desplazamientos, al existir pocas partículas que den información de objetos en iteraciones anteriores del PF.

El resto de los errores son "no generado" o "unión de dos" al aparecer objetos próximos entre sí. Este tipo de errores supone prácticamente la totalidad del número de fallos.

Analizando las tablas donde se dispone de toda la información detallada, se elige como valor óptimo n_m=70% de *n*, es decir, teniendo en cuenta que en la prueba anterior se ha asignado a *n* un valor de 100, n_m=70 partículas. La tasa de error es de un 35% y de un 61% en el proceso de asociación y clasificación a la salida respectivamente. Con un valor de n_m=80 se obtiene prácticamente el mismo número de fallos, pero como se ha dicho anteriormente, aparecen algunas clases desplazadas.

3ª Prueba: cX

Como se ha explicado con anterioridad, *cX* es la desviación típica del modelo Gaussiano del ruido asociado al vector de estado. Por tanto si aumenta el valor de *cX* aumenta el ruido introducido en el modelo, y el valor de las partículas predicho $S_{t|t-1}$ difiere en mayor medida del corregido en el instante anterior $S_{t-1|t-1}$.

Para las pruebas de ajuste de este parámetro se le asigna unos valores de 1, 10, 100 y 1000 mm. con el fin de cubrir un rango de valores amplio y comprobar el funcionamiento del algoritmo en todos los casos. Además, se dispone del estudio realizado con *cX*=70, ya que las pruebas anteriores se han realizado con dicho valor. La prueba con *cX*=1000 se ha sustituido por *cX*=700 debido a que aparece un número excesivo de errores. Además, una vez realizadas todas estas pruebas, se ha incluido el estudio para *cX*=400 con el fin de encontrar datos concluyentes para escoger un valor como óptimo.

En la Figura 3.19 se muestra una tabla con las tasas de error en el proceso de clasificación para distintos valores de *cX* en el experimento realizado:

	cX=1	***cX=10***	***cX=100***	***cX=400***	***cX=700***
Unión de dos	*21.20*	*25.16*	*22.51*	*11.60*	*8.28*
Desplazados	*5.30*	*4.63*	*4.00*	*6.00*	*7.28*
No generados	*42.38*	*27.81*	*24.17*	*27.50*	*24.50*
Otros	*4.32*	*15.91*	*4.32*	*5.00*	*9.94*
Total	***73.20***	***73.51***	***55.00***	***50.00***	***50.00***

Figura 3.19. Porcentajes de errores en función del parámetro cX.

En el proceso de asociación de partículas la tasa de error se mantiene constante para valores intermedios de cX; 10, 70, 100 y 400 mm., siendo de un 35%. Para los valores extremos, *cX*=1 y *cX*=700 la tasa de error es de un 42%. El error más significativo es, como en casos anteriores, el "no generado", suponiendo éste más de un 80% del total de los errores.

En la clasificación de partículas a la salida el número de errores disminuye a medida que aumenta cX, pasando la tasa de fallo de un 73.20% con *cX*=1 a un 50% con *cX*=700, como se ve en la Figura 3.19. Los fallos del tipo "no generado" suponen, en la mayoría de los casos, prácticamente el 50% del total de los errores.

Finalmente, y tras analizar todos los errores, se elige *cX*=400 como valor óptimo ya que con él se obtiene una menor tasa de error.

4ª Prueba: cY

La desviación típica del ruido de las medidas (*cY*) interviene en el cálculo de los pesos que influye en la selección de partículas, como se ve en la siguiente fórmula:

$$w = \frac{1}{cY \cdot \sqrt{2 \cdot \pi}} \cdot e^{-0.5 \frac{dist^2}{cY^2}}, \tag{3.2}$$

donde *dist* es la distancia entre las partículas y los centroides de las clases existentes.

Para ajustar *cY* se le han asignado los siguientes valores: 1, 10, 70, 100, 150 y 200 mm.

Los errores en el proceso de asociación de partículas, en su totalidad fallos "no generado" disminuyen a medida que se aumenta el valor de *cY*, como es lógico ya que al aumentar *cY* es más probable que un objeto tenga asociadas partículas al aumentar el peso de las mismas. La tasa de error pasa de un 96% con *cY*=1 a un 18.20% con *cY*=200 como se observa en la Figura 3.20.

En la Figura 3.20 se muestra la evolución en los porcentajes de fallos en este experimento:

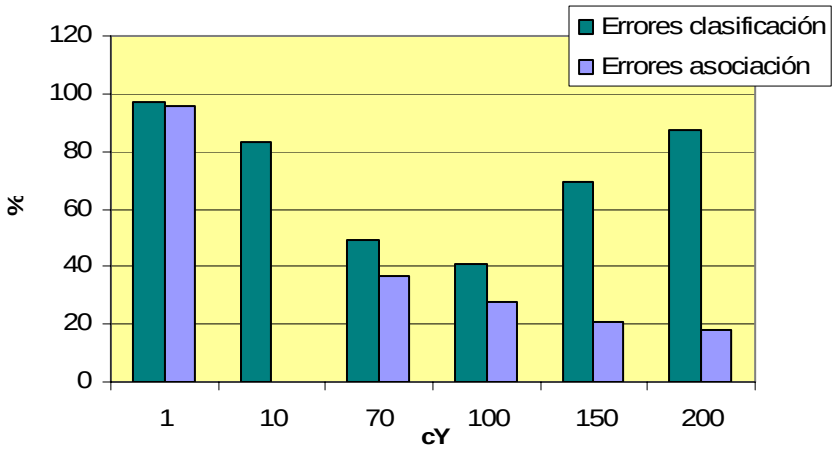


Figura 3.20. Tasa de error de los procesos de asociación y clasificación a la salida de las partículas en función de cY.

El número de errores del proceso de clasificación es máximo para *cY*=1 con una tasa de error del 97%, disminuye a medida que aumenta el valor hasta encontrar el mínimo en *cY*=100 y un porcentaje de fallo del 41%. Si se sigue aumentando *cY* la cantidad de errores vuelve a aumentar considerablemente.

Con *cY*=1 y *cY*=10, casi todos los errores que se encuentran son "no generado". Esto es debido a que con dichos valores prácticamente en ninguna iteración aparecen partículas asociadas a los objetos a seguir, de modo que es muy difícil que sean validados. Concretamente, sólo el 1% y el 3% de las iteraciones están exentas de error para *cY*=1 y *cY*=10. Con valores mayores de *cY* este error disminuye considerablemente.

Si se le asignan a *cY* valores de 150 y 200 aparecen multitud de fallos de "duplicado", siendo una fuente de error importante, con una tasa de fallo del 32% y del 47.68% respectivamente. Esto se debe a que con estos valores casi todas las partículas sobreviven al paso de selección, y por tanto se encuentran menos concentradas alrededor del centroide.

Se elige como valor óptimo *cY*=100, después de analizar exhaustivamente el número y tipo de errores en todos los casos.

3.2.3. Conclusiones acerca del ajuste de "XPFCP"

Una vez finalizados los ajustes se llega a las siguientes conclusiones sobre los distintos parámetros:

- El número total de partículas *n* influye principalmente en el tiempo de ejecución. El algoritmo tarda más en ejecutarse a medida que se incrementa el valor de *n*, al existir cálculos en los que intervienen todas las partículas. Ni el proceso de asociación de partículas ni la clasificación sufren mejorías relevantes ni variaciones importantes. Esto es debido a que, pese a haberse aumentado *n*, n_m se mantiene constante y la relación entre *n* y

n_m no es la correcta. Lo más adecuado es mantener la relación entre ambos parámetros y si uno de ellos se aumenta, el otro ha de hacerlo en la misma medida; medida que se tomará en las siguientes pruebas. El valor final es n=100, algo razonable teniendo en cuenta que n_m=70.

- El número de partículas que se introducen en el paso de reinicialización del PF, n_m, acelera el proceso de validación de clases al ser más probable que se le asocien partículas a los objetos nuevos. El porcentaje de errores "no generado" en el proceso de clasificación pasa de un 88% con n_m=10 a un 53.50% con n_m=90, al disminuir del mismo modo los fallos en la asociación de partículas. El valor final elegido es n_m=70 partículas, es decir, el 70% del número total n.
- El XPFCP es sensible a las modificaciones del parámetro *cX*. La cantidad de errores "no generado" en el proceso de clasificación disminuye a medida que se aumenta este valor hasta llegar a *cY*=100. Si se sigue aumentando la cantidad de fallos vuelve a aumentar. Finalmente se escoge *cX*=400 como mejor opción.
- Por último, se ajusta *cY*. Al estar involucrado este parámetro directamente en la asignación de partículas distintas en cada iteración son lógicos los resultados obtenidos. Los errores en la asociación de partículas disminuyen a medida que se aumenta este parámetro, y por tanto los fallos "no generado" de la clasificación también. Por otra parte, si se le asigna a *cY* valores demasiado elevados aparecen duplicados al esparcirse las partículas debido a que muchas de ellas sobreviven al paso de selección. A este parámetro se asigna después de las pruebas un valor de 100.

Por tanto, los parámetros ya ajustados tienen los valores finales mostrados en la Figura 3.21:

n=100	***n_m=70***	***cX=400***	***cY=100***

Figura 3.21. Tabla con los valores de cada uno de los parámetros del XPFCP que se han ajustado.

En Figura 3.22 se han representado la tasa de error obtenida después de cada uno de los ajustes:

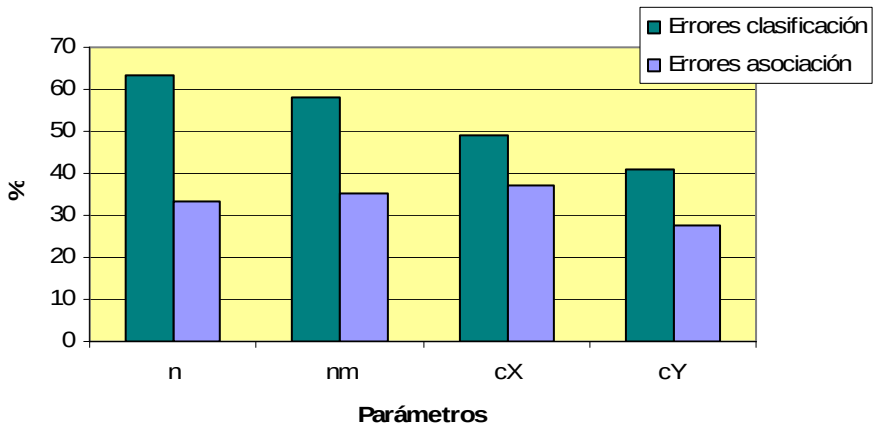


Figura 3.22. Tasa de error obtenida después del ajuste de cada parámetro.

Como conclusión comentar que la **tasa de error** obtenida en el proceso de **asociación** de partículas es de un **27.48%**, mientras que en la clasificación es de un 41%.

3.3. Ajuste de los parámetros correspondientes a "SJPDAF"

3.3.1. Pruebas a realizar específicas

Una vez definidas las dos versiones del algoritmo SJPDAF, se buscan los posibles parámetros que se pueden modificar para mejorar los resultados obtenidos con dicho estimadores.

Esos parámetros son los siguientes y serán analizados en el siguiente orden:

1. *canM*
2. *cY*
3. *cX*
4. *n*
5. n_m

El orden del estudio es diferente respecto a las pruebas del XPFCP con el fin de intentar seguir un orden más lógico en el ajuste y evitar así definir un parámetro con un valor que no sea el que nos proporcione los mejores resultados. Este orden se ha establecido tras el análisis de los resultados obtenidos con el XPFCP.

Al igual que en XPFCP se analizan los errores por una parte del proceso de asociación de partículas a la entrada del algoritmo, y por otra parte de la clasificación de las partículas a la salida.

En el análisis de los SJPDAFs sólo se emplea la prueba larga del vídeo 10, al ser ésta la más compleja y la que contiene situaciones más adversas.

En la Figura 3.23 se muestran los valores de cada parámetro antes de los ajustes. Estos valores son comunes inicialmente para ambas versiones:

cY=10	***cX=100***	***n=600***	***n_m=200***

Figura 3.23. Tabla con los parámetros de las dos versiones del SJPDAF sin ajustar.

3.3.2. Ajuste de parámetros comunes

Primeramente, y previo a ningún otro análisis, se fija un valor de *canM* común para ambas versiones. Posteriormente se analizan el resto de parámetros para los dos casos individualmente.

1ª Prueba: canM

Este parámetro influye, al igual que en el KFPDA y en el XPFCP, en la validación e invalidación de las clases.

Para ajustar este parámetro se lleva a cabo la prueba larga con valores de *canM*=1 y *canM*=2. Se comprueba que con *canM*= 2 el comportamiento es ligeramente mejor pese a aparecer a un elevado número de errores tanto en el proceso de asociación, con una tasa de fallo del 36.81%; como en la clasificación de las partículas, donde el porcentaje de error es de un 78.60%.

El número de errores, todos del tipo “no generado”, es excesivamente alto. Por esto se intenta encontrar la causa del problema.

En primer lugar se revisan los pesos de las partículas del objeto de la papelera para comprobar que no son demasiado bajos como para ser eliminados en el proceso de selección. Se comprueba que los pesos son adecuados.

El problema se encuentra en el proceso de validación ya que nunca se cumple la condición que da lugar a que una clase sea validada. En la condición para validar/invalidar intervienen los parámetros *um*, *umHist* y parámetros correspondientes a distancias como se vio en el Capítulo 2. Se comprueba que la papelera no se valida debido a que su probabilidad *P* nunca alcanza el valor *um* que es el límite que debe superar para poder ser validada, mientras que *umHist* es el límite por debajo del cual se invalidan las clases.

El valor inicial de *um* es 0.6. Se hacen pruebas consecutivas con valores de 0.4, 0.3, 0.2, 0.1 y finalmente con 0, observando en las imágenes si se valida o no la clase correspondiente al objeto de la papelera. Finalmente se asigna a *um* el valor 0 y que la validación no dependa de él para el correcto funcionamiento del algoritmo SJPDAF.

Con esta modificación se obtiene una tasa de error menor con *canM*= 2, siendo de un 28.85% en la asociación de partículas y un 32.83% en la clasificación a la salida.

En la Figura 3.24 y la Figura 3.25 se observa la diferencia en el proceso de validación en función del parámetro *um*. En la Figura 3.24 la clase de la papelera no se valida a lo largo de las 6 iteraciones pese a tener partículas con un peso apropiado, mientras que en la Figura 3.25 la clase de la papelera es validada en la tercera iteración.

Figura 3.24. Secuencia de imágenes de un objeto estático con el parámetro um=0.6.

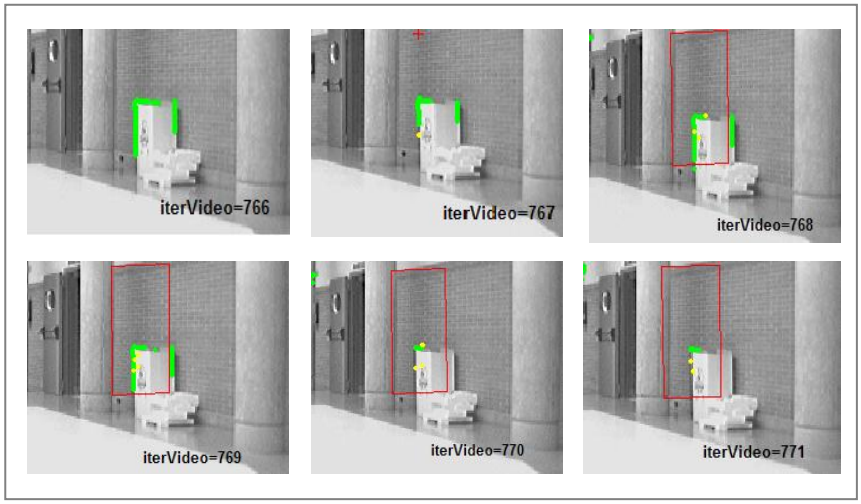


Figura 3.25. Secuencia de imágenes de un objeto estático con el parámetro um=0.

3.3.3. Ajuste de parámetros de "SJPDAF"

3.3.3.1. Comentarios sobre las pruebas realizadas

1ª Prueba: cY

Este parámetro, correspondiente a la desviación típica del ruido de las medidas, influye directamente en el cálculo del peso de cada una de las partículas como puede verse en la ecuación (3.3). Si *cY* tiene un valor pequeño, las probabilidades o "betas" serán bajas y sus pesos no serán lo suficientemente elevados como para sobrevivir en el proceso de selección. Habrá pocas partículas

distintas y muchas repeticiones de ellas, es decir, en la imagen se ven aparentemente pocas partículas.

$$w = \frac{1}{cY \cdot \sqrt{2 \cdot \pi}} \cdot e^{-0.5 \frac{dist^2}{cY^2}} \tag{3.3}$$

Esto se puede comprobar en la Figura 3.26. Se representan en amarillo las partículas para *cY*= 1 y a *cY*=50. En el primer caso no existen prácticamente miembros distintos, mientras que en el segundo caso sí los hay:

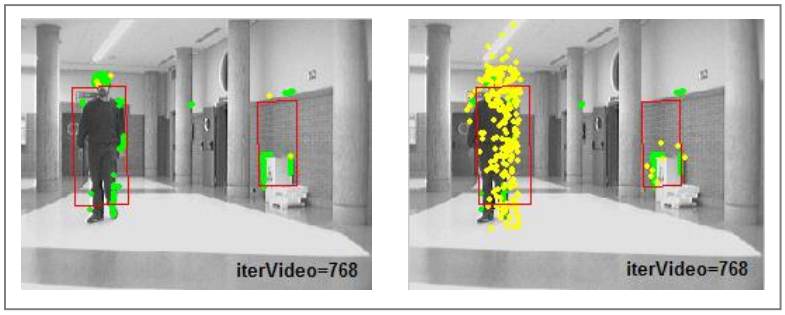


Figura 3.26. Imágenes que muestran la diferente disposición de partículas con cY=1 (izquierda) y cY=50 (derecha).

Para el estudio de la sensibilidad del algoritmo a este parámetro se le asignan los siguientes valores: 1, 10, 50, 70 y 100 mm.

En la Figura 3.27 se muestra una tabla con la tasa de error en el proceso de clasificación en función del parámetro *cY*.

	cY=1	***cY=10***	***cY=50***	***cY=70***	***cY=100***
Unión de dos	*4.5*	*7.5*	*4.5*	*5.0*	*0*
Desplazados	*0*	*7.5*	*4.0*	*5.0*	*0*
No generados	*57.7*	*24.8*	*26.3*	*17.4*	*22.9*
Otros	*11.93*	*33.83*	*0.52*	*21.35*	*26.35*
Total	***74.13***	***73.63***	***35.32***	***48.75***	***49.25***

Figura 3.27. Tasas de error debidas a los distintos fallos en función de cY.

En el proceso de asociación de partículas del PF, los fallos del tipo “no generado” disminuyen de forma gradual a medida que se asigna un mayor valor a *cY*. Con *cY*=1 el 73% de las iteraciones tienen un error de este tipo, mientras que con *cY*=100 sólo en un 32% de ellas no hay partículas de algún objeto.

Se observa en la Figura 3.28 una gráfica en la que se recogen los errores tanto de la asociación de las partículas como de la clasificación y los ruidos:

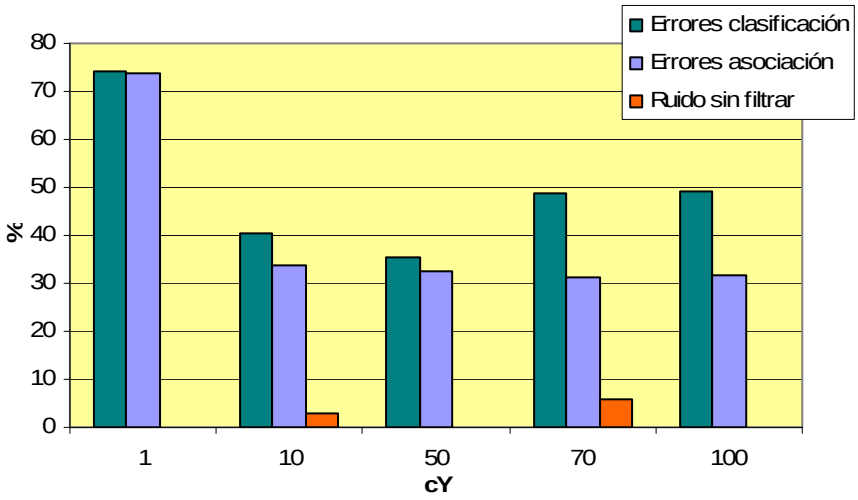


Figura 3.28. Gráfico con el porcentaje de error en función de cY.

Los errores en el proceso de clasificación son prácticamente todos "no generado". Este fallo aparece menos a medida que aumenta *cY*, de forma que la tasa de error debido a este fallo pasa de un 55.22% con *cY*=1 a un 23.38% con *cY*=100.

Con los valores más altos de *cY*, 70 y 100, se obtiene un 25% y 11% respectivamente de iteraciones con fallos de "duplicado".

El ruido es filtrado en todos los casos correctamente.

En este ajuste se escoge *cY*=50 como mejor opción debido a que con este valor se halla el menor número de errores tanto en la asociación como en la clasificación de partículas. Además la cantidad de errores "no generado" en la clasificación es mínimo, y tal y como se ha mencionado en anteriores ocasiones, este fallo es el más importante.

2ª Prueba: cX

Este parámetro indica la desviación típica del ruido del modelo Gaussiano utilizado. Si el ruido es pequeño la predicción del estado de cada partícula para el instante siguiente $S_{t|t-1}$ será más semejante a la corrección en el instante anterior $S_{t-1|t-1}$, ya que este ruido se suma directamente a cada uno de los parámetros de la ecuación de estado, mientras que si el ruido del modelo aumenta estos dos valores difieren más.

Se puede prever entonces que cuanto mayor sea el valor de este parámetro más dispersas se encuentran las partículas.

Se muestra en la Figura 3.29 la disposición de las partículas en función del valor de *cX*:

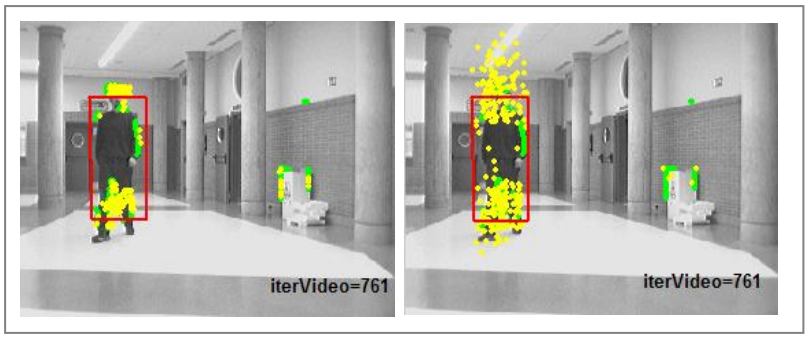


Figura 3.29. Disposición de partículas con cX=1 (izquierda) y cX=100 (derecha).

Se observa como con *cX*=1 las partículas están más concentradas, y con *cX*=100 mm. las partículas se esparcen más.

El resumen de la evolución de la tasa de fallo en cada caso se puede ver en la Figura 3.30:

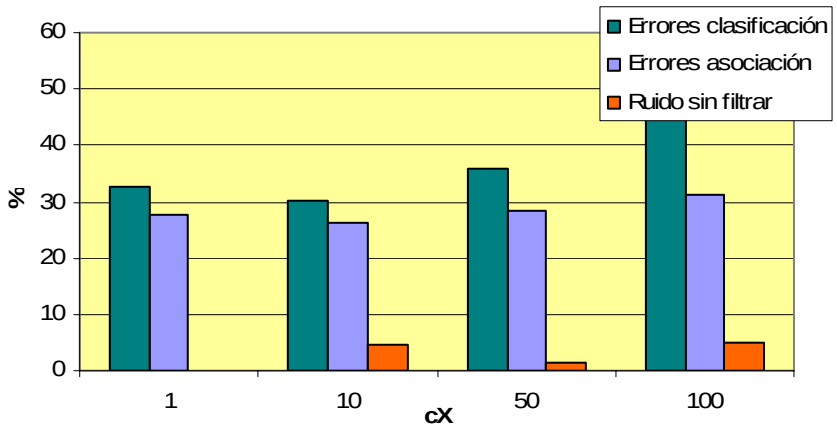


Figura 3.30. Diagrama de barras con el porcentaje de errores en función de los distintos valores de cX.

Para obtener el valor óptimo en el ajuste se realizan las pruebas con los siguientes valores de *cX*: 1, 10, 50 y 100 mm.

En el proceso de asociación de partículas a la entrada se obtiene la mayor tasa de error con *cX*=100, siendo ésta de un 31.34%. Por otra parte la menor tasa de fallo es de un 26% y se obtiene con *cX*=10. Como se comprueba la cantidad de fallos no aumenta ni disminuye considerablemente al variar *cX*.

La cantidad de fallos en el proceso de clasificación aumenta a medida que se aumenta el valor de *cX* como se puede ver en la Figura 3.30.

Sólo para *cX*=100 aparecen fallos de “duplicado”, dando lugar a un 18% de iteraciones con error.

Únicamente se encuentran ruidos sin filtrar para los mayores valores de *cX*: 10, 50 y 100 mm.; pero como se ve en la Figura 3.30 los porcentajes son insignificantes.

Tras este análisis se escoge *cX*=10 ya que su tasa de error es la menor obtenida y poseer menos errores "no generado" tanto en el proceso de asociación como en el de clasificación de partículas a la salida.

3ª Prueba: n

Este parámetro indica el número total de partículas empleadas en el PF.

A este parámetro se le dan valores alrededor de un rango razonable, puesto que no es apropiado asignar un número de partículas muy pequeño (aparecerían muchos errores tanto en la asociación de partículas como en la clasificación a la salida, como se ha comprobado en el XPFCP), o muy grande (el tiempo de ejecución sería muy alto). El rango de valores elegido para el número de partículas totales es entre *n*=400 y *n*=700. Para el correcto ajuste de *n*, se mantiene n_m igual a la tercera parte del valor total de partículas *n*, es decir, se adapta el valor de n_m a *n*. De esta forma la proporción entre el número total de partículas y la cantidad de ellas que se insertar en el paso de reinicialización se mantiene constante.

En la Figura 3.31 se muestra la tasa de error obtenida con cada uno de los valores de *n*:

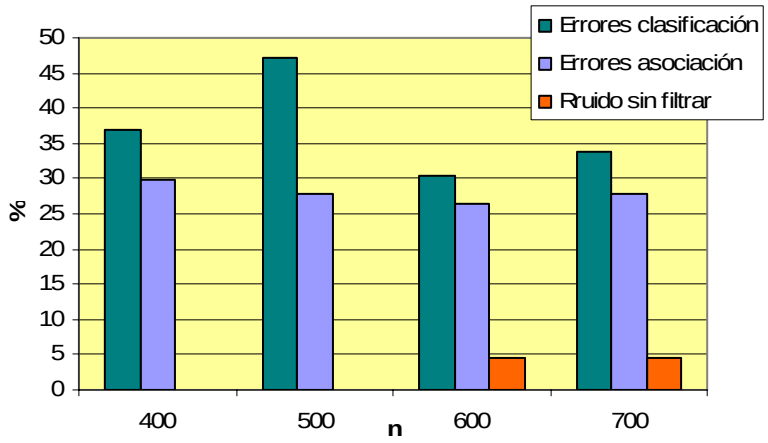


Figura 3.31. Gráfico que muestra la tasa de error en función de n.

En el proceso de asociación de partículas la tasa de error se mantiene prácticamente constante, variando de un 26.37% con *n*=600 a un 29.85% con *n*=400.

En la clasificación a la salida de las partículas se obtiene una tasa de fallo máxima para *n*=500 (47.26%) y mínima para *n*=600 (30.34%).

El porcentaje de fallos "no generado" en el proceso de clasificación se mantiene en torno a un 30% para todos los valores de *n*.

Por último los errores de "duplicado" sólo aparecen para *n*=500, siendo la tasa de error correspondiente de un 15%. Con el resto de los valores no aparece este tipo de fallo, y es por esto

que para *n*=500 la cantidad de errores es mucho mayor que en el resto de los casos como se observa en la Figura 3.31.

El incorrecto filtrado de ruidos sigue dando una tasa de error pequeña como se vde en la Figura 3.31.

Finalmente se elige *n*=600 ya que con este valor, tanto los errores de asociación como de clasificación de partículas son mínimos. Además los fallos "no generado" en ambos procesos son mínimos.

4ª Prueba: n_m

Como se ha explicado en el ajuste del XPFCP, el parámetro n_m indica el número de partículas nuevas que se inyectan en el paso de reinicialización con el fin de que el sistema evolucione y sea más probable que a un objeto nuevo se le asocien partículas, y por tanto, clases.

Se analizan valores de n_m comprendidos entre una quinta parte del total de las partículas *n* y un medio de éstas.

En la Figura 3.32 se observan las tasas de error para los distintos valores de n_m:

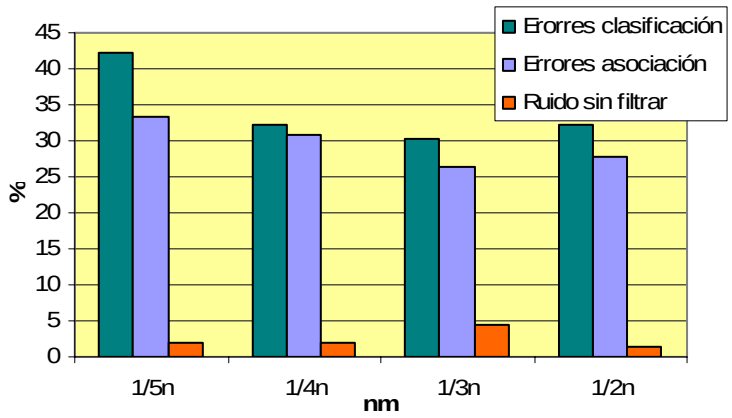


Figura 3.32.Tasas de error en función de n_m.

La tasa de error en la asociación de partículas disminuye a medida que aumenta n_m alcanzando el mínimo de un 26% para n_m igual a la tercera parte de *n*, y posteriormente sufriendo un ligero incremento con n_m un medio de *n*.

En el proceso de clasificación a la salida la evolución en el porcentaje de fallos es la misma que en la asociación de partículas, siendo el máximo de un 41.30% con n_m igual a un quinto de *n* y el mínimo de un 26.37% siendo n_m la tercera parte del número total de partículas.

El número de errores "no generado" en la clasificación es máximo para n_m un quinto de *n*, con un 34% de tasa de fallo. En los restantes tres casos se mantiene alrededor del 29%, siendo mínimo para n_m=200 partículas (la tercera parte de *n*) con un 28%.

El filtrado del ruido no sufre grandes cambios respecto a los anteriores ajustes del SJPDAF como se puede comprobar en la Figura 3.32.

De la misma forma que en los ajustes anteriores, se selecciona el valor del parámetro que da lugar al menor número de errores de todo tipo. En este caso es nm=200 partículas, es decir, la tercera parte del total, ya que posee menor número de errores de falta de clases y partículas y menor cantidad de desplazamiento en el proceso de asociación.

3.3.3.2. Conclusiones acerca del ajuste de "SJPDAF"

Tras finalizar la totalidad de los ajustes se obtienen las siguientes conclusiones:

- El SJPDAF es muy sensible al ajuste de *cY*. Si se le asigna un valor demasiado pequeño los errores "no generado" son elevadísimos ya que los pesos no son adecuados, pero si se le asigna un valor demasiado elevado el número de fallos del tipo "duplicado" asciende considerablemente al sobrevivir muchas partículas al paso de selección. El valor escogido es *cY*=50 con el que se obtiene una tasa de error de 32.34% y 35.32% para el proceso de asociación y clasificación de las partículas.
- *cX* influye en los resultados que se obtienen con el SJPDAF. Al terminar el ajuste correcto se ha conseguido reducir la tasa de error del proceso de asociación a un 26.38% y a un 30.35% la de la clasificación. De este parámetro depende la dispersión de las partículas, por lo tanto es necesario asignarle un valor adecuado que haga que las mismas se encuentren repartidas uniformemente alrededor de los objetos; de lo contrario, si se expanden demasiado, se producen fallos de "duplicado" al aparecer miembros alejados. El valor final escogido es *cX*=10.
- El número total de partículas *n* es un parámetro fundamental como ya se ha comentado. El estudio se ha hecho asignándole valores dentro de un rango razonable entre 400 y 700 partículas, y no se aprecian variaciones significativas en los errores entre un caso u otro. Tras el ajuste no se ha encontrado ninguna opción mejor que mantener *n* con el valor inicial, es decir, 600.
- Con el último parámetro n_m sucede lo mismo que con el número de partículas. Es un parámetro fundamental en el comportamiento del algoritmo, pero en el rango estudiado no se observan variaciones en cuanto a la cantidad de errores. Tras concluir la prueba el valor escogido es el que se ha usado en el estudio desde el principio, es decir, n_m igual a la tercera parte de *n*.

En la Figura 3.33 se observa la variación en el número de errores tras finalizar los ajustes de cada uno de los parámetros.

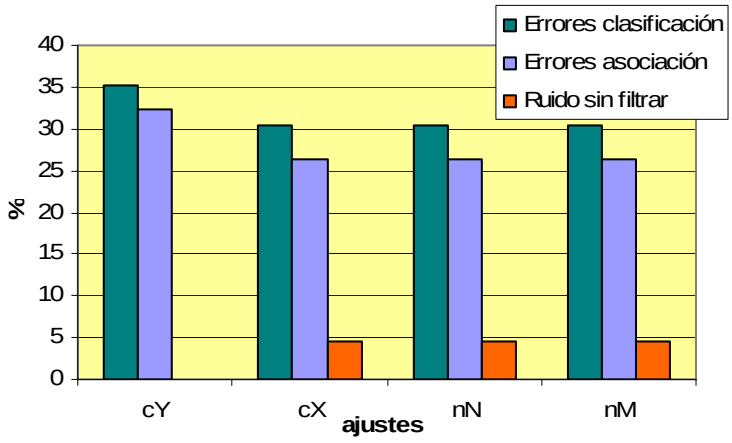


Figura 3.33. Gráfico con la variación de la tasa de error tras cada uno de los ajustes.

Después de finalizar todos los ajustes, los parámetros tienen los valores mostrados en la Figura 3.34:

canM=2	***cY=50***	***cX=10***	***nN=600***	***nM=200***

Figura 3.34. Tabla con los valores finales de los parámetros ajustados del SJPDAF.

Por lo tanto, la **tasa de error** al término de los ajustes es de un **26.38%** y un 30.35% para los procesos de **asociación de partículas** y clasificación a la salida respectivamente.

3.3.4. Ajuste de parámetros de "SJPDAF+NN"

3.3.4.1. Comentarios respecto a las pruebas realizadas

Las pruebas a realizar para optimizar el funcionamiento de este algoritmo son las mismas que en el caso anterior:

1. *cY*
2. *cX*
3. *n*
4. n_m

1ª Prueba: cY

A la desviación típica correspondiente al ruido de las medidas se le asignan los mismos valores que en el estudio del SJPDAF: 1, 10, 50, 70 y 100 mm.

En la Figura 3.35 se muestra una tabla con la evolución de los errores más significativos:

	cY=1	***cY=10***	***cY=50***	***cY=70***	***cY=100***
Duplicados y Triplicados	*0*	*0*	*11.4*	*29.3*	*33.8*
No generados	*56.8*	*26.8*	*14.5*	*9.5*	*9.0*
Otros	*20.81*	*15.0*	*9.42*	*14.93*	*11.92*
Total	***77.61***	***41.79***	***35.32***	***53.73***	***54.72***

Figura 3.35. Tabla resumen con las tasas de error de los fallos más importantes en función de cY.

Los errores en la asociación de las partículas disminuyen a medida que aumenta el valor asignado a *cY* al ser los pesos más altos. La tasa de error pasa del 73.13% con *cY*=1 al 14.43% con *cY*=100.

La tasa de fallo en la clasificación a la salida de las partículas es máxima para *cY*=1 y mínima para *cY*=50 como se ve en la Figura 3.35.

Los errores "no generado" en la clasificación disminuyen al aumentar *cY* al estar más repartidas las partículas.

A partir de *cY*=50 aparecen errores de "duplicado" de clases, al sobrevivir muchas partículas tras el proceso de selección.

En la Figura 3.36 se puede ver cómo aparecen los duplicados al aumentar el valor de *cY* de 10 a 100 mm:

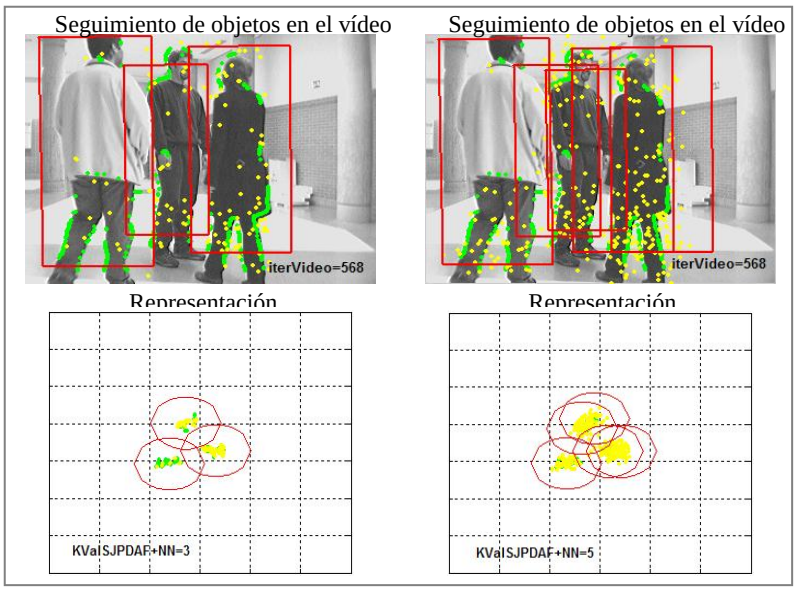


Figura 3.36. Resultado del seguimiento con cY=10 (izquierda) y cY=100 (derecha).

Otro aspecto a comentar es el filtrado del ruido. El porcentaje de iteraciones en las que se encuentra ruido sin filtrar es de 0%, 1.50%, 8.95%, 17% y 9.45% para valores de *cY* de 1, 10,50, 70 y 100 mm.

Se elige *cY*=100 como valor óptimo, con un mínimo de errores "no generado". Pese a obtener un muchos fallos en la clasificación, la mayor parte de ellos son del tipo "duplicado". Además, el número de errores en el proceso de asociación es el mínimo obtenido.

2ª Prueba: cX

Para el correcto ajuste y estudio de la desviación típica del ruido del modelo se le han asignado los siguientes valores: 1, 10 y 100 mm.

En la Figura 3.37 se observa la tasa de error para cada valor de *cX*:

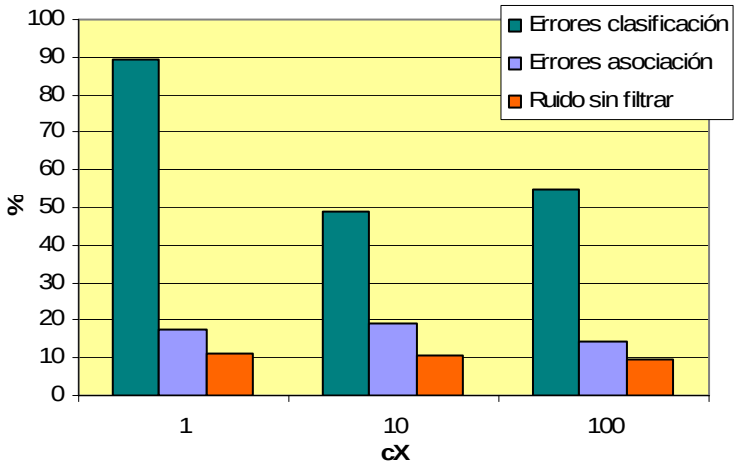


Figura 3.37. Diagrama de barras con los errores en función de cX.

En el proceso de asociación de las partículas a los diferentes objetos la tasa de error no se ve muy influenciada por las variaciones de *cX*, manteniéndose alrededor de un 16%.

Los fallos de la clasificación de las partículas son máximos para *cX*=1 y mínimo para *cX*=10 como se ve en la Figura 3.37.

Se encuentra muchos fallos de "duplicado" de clases. Con *cX*=1 las clases duplicadas y triplicadas suponen un 70.60% de las iteraciones con error, porcentaje que disminuye a medida que aumenta *cX* como se observa en la Figura 3.38.

Por otra parte el número de errores "no generado" de la clasificación a la salida es máximo para *cX*=10 y mínimo para *cX*=1.

En cuanto al ruido el porcentaje de iteraciones con ruidos sin filtrar decrece a medida que aumenta *cX* como se ve en la Figura 3.37.

En la Figura 3.38 se muestra la variación en la cantidad de clases validas para *cX*=1 y *cX*=100:

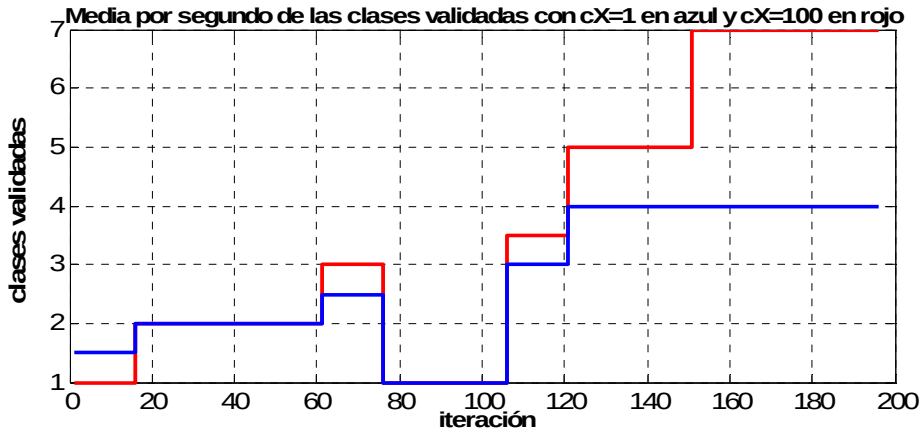


Figura 3.38. Cantidad de clases validadas para cX igual a 1 y a 100 mm.

Por todo esto se escoge como mejor opción *cX*=100 con una tasa de error del 14.43% en el proceso de asociación y de un 54.70% en el proceso de clasificación.

3ª Prueba: n

Al igual que en el ajuste del SJPDAF, al número total de partículas se le asignan los siguientes valores: 400, 500, 600 y 700.

En la Figura 3.39 se muestran los porcentajes de error de en función del número de partículas:

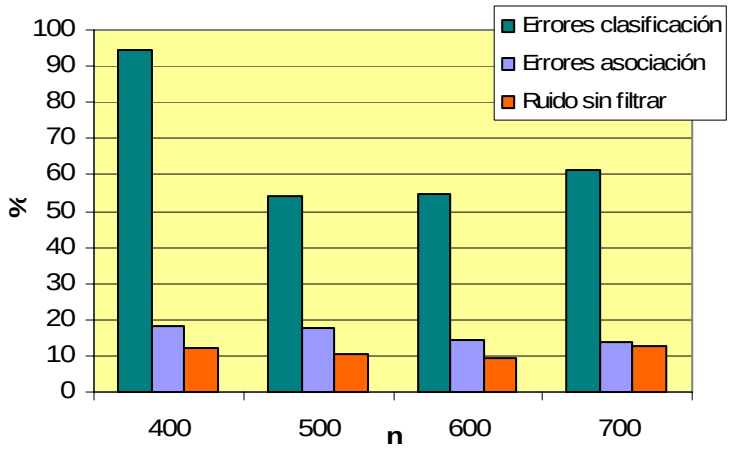


Figura 3.39. Gráfico con el porcentaje de fallos en función del número de partículas.

La tasa de error en el proceso de asociación disminuye ligeramente a medida que aumenta *n*, pasando de un 18.40% con *n*=400 a 14% con *n*=700. Algo lógico ya que al existir más partículas la probabilidad de que éstas sean asociadas a los objetos es mayor.

En el proceso de clasificación se obtienen muchos errores como se ve en la Figura 3.39, siendo prácticamente todos fallos de “no generado” y “duplicado” de clases.

La cantidad de "duplicados" es máxima para *n*=400, ya que sólo este error genera una tasa de fallo de un 36.32%. En el resto de casos este error es mucho menor.

En cuanto a los errores "no generado" aumentan a medida que el número de partículas es mayor pasando de suponer un 4% de fallos con *n*=400 al 11% con *n*=700.

El número de iteraciones en las que se encuentran ruidos sin filtrar no varía al modificar *n*. Es decir, este parámetro no influye de forma importante en el filtrado de los posibles ruidos.

Con todas estas consideraciones, se escoge el valor de 600 como óptimo ya que los "duplicados" no alcanzan un valor excesivo y los errores "no generado" en el proceso de asociación y de clasificación no son máximos.

4ª Prueba: n_m

Al igual que en ajuste del SJPDAF, se varía el valor de n_m de un quinto de *n* a un medio de *n*.

En la Figura 3.40 se muestra el diagrama de barras con la evolución de la tasa de error en función de n_m:

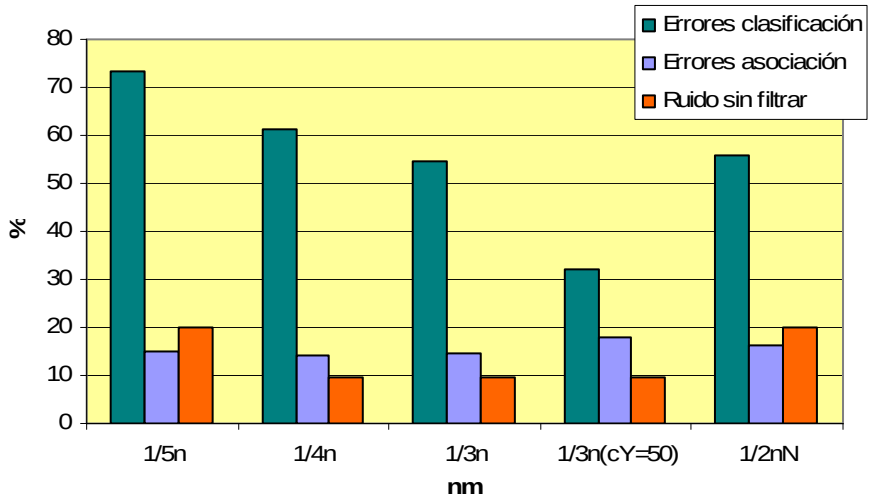


Figura 3.40. Evolución de la tasa de error en función de n_m.

Una vez realizado el ajuste de n_m se observa que, pese a analizar un amplio rango del mismo, la tasa de error es muy alta como se ve en la Figura 3.40.

Se comprueba que existe un altísimo número de duplicados y triplicados (alrededor de un 40% para todos los valores de n_m), error que generalmente es arrastrado hasta que desaparece la clase con dicho fallo. Se opta por modificar, ya con todos los demás parámetros ajustados, alguno de ellos para comprobar si mejora algo o no.

Tras varias pruebas se modifica el parámetro *cY* y se le asigna el valor 50. El número de duplicados desciende de un 37% a un 15%. Sin embargo, el resto de errores tanto en el proceso de asociación como en la clasificación se mantiene prácticamente constante.

La opción que se ha dejado como definitiva ya había sido probada, pero en la anterior ocasión no dio los resultados obtenido en esta última prueba.

Esto tiene una explicación: los algoritmos que se están probando se basan en métodos probabilísticos y aleatorios, no son repetibles. Por ejemplo la disposición de las partículas se debe en medida a un método aleatorio y por tanto, cada vez que se ejecuta quedan distribuidas de forma distinta. Esto puede llevar a que aparezcan distintos errores ya sean duplicados, falta de clases, etc.

Se observa en la Figura 3.40 que con n_m la tercera parte de n y cY=50 se obtienen los mejores resultados, siendo la tasa de fallo del proceso de asociación de un 18% y de la clasificación a la salida de un 32%.

En este caso los errores de "duplicado" de clases suponen un 11.44% de tasa de fallos, y los errores "no generado" un 12%.

En cuanto al ruido se observa que en todos los casos analizados el número de iteraciones con algún tipo de ruido sin filtrar es semejante, por lo que este aspecto no aporta nada a la hora de decidir el mejor valor para el parámetro.

Como conclusión, se escoge n_m igual a la tercera parte de n y además se modifica cY, asignándole un nuevo valor de 50 mm.

3.3.4.2. Conclusiones acerca del ajuste de "SJPDAF+NN"

Una vez finalizados los ajustes de todos los parámetros y tras el análisis de los resultados obtenidos se llega a las siguientes conclusiones:

- La desviación típica del ruido de las medidas cY es, al igual que en el SJPDAF, uno de los parámetros fundamentales al ser responsable de la disposición de las partículas. Al variar el valor de cY los resultados obtenidos con el SJPDAF+NN cambian en gran medida. En una primera prueba se escoge cY=100 como valor óptimo, pero tras terminar todos los ajustes, y como se ha explicado ya anteriormente, se modifica su valor a 50 mm.
- El parámetro cX que indica el ruido del modelo es también de gran importancia en el SJPDAF+NN. El incorrecto filtrado de ruidos disminuye a medida que aumenta el valor de cX. El valor final elegido es cX=100.
- El número total de partículas n es fundamental en el SJPDAF+NN. Los resultados cambian dependiendo del valor de este parámetro. El número de fallos de "duplicado" de clases disminuye a medida que aumenta n, y el número de errores "no generado" en la clasificación de partículas aumenta. Finalmente se escoge n=600, valor que tenía inicialmente.
- n_m influye en los resultados pero no en la medida que los anteriores parámetros. La mayor parte de los errores se mantienen prácticamente constantes al modificar su valor. Únicamente el "duplicado" desciende al aumentar n_m.

En la Figura 3.41 se muestra la variación en la cantidad de errores después de cada una de las pruebas. Se puede observar que tras el primero de los ajustes no se ha conseguido mejoría en la cantidad de fallos obtenida modificando ninguno de los parámetros. Únicamente en la última prueba donde se ha modificado *cY* de nuevo y han disminuido los "duplicados" se han mejorado los resultados:

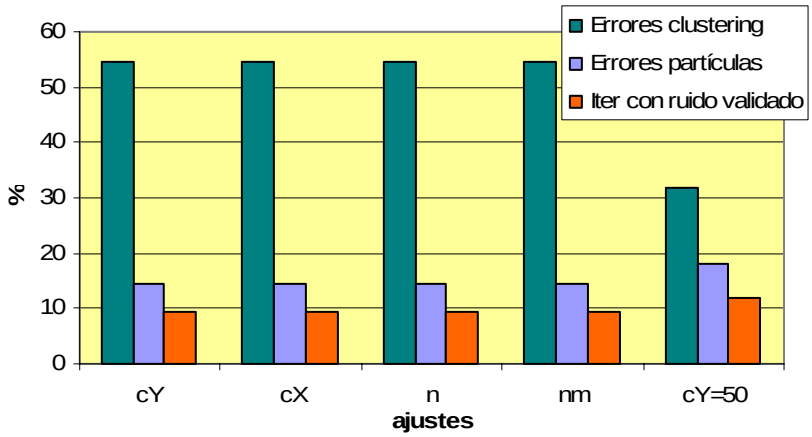


Figura 3.41. Diagrama de barras con la tasa de error tras los ajustes correspondientes.

En la Figura 3.42 se muestran los valores finales de los parámetros ajustados:

canM=2	***cY=50***	***cX=100***	***nN=600***	***nM=200***

Figura 3.42. Tabla con los valores ajustados de cada parámetro.

Por lo tanto, tras finalizar todos los ajustes del SJPDAF+NN, la **tasa de error** obtenida en el proceso de **asociación de partículas** es de un **18%** y la correspondiente a la clasificación a la salida alcanza un 32%.

CAPÍTULO 4. COMPARATIVA DE LOS ALGORITMOS

4.1. Pruebas realizadas

Una vez que ajustados los cuatro algoritmos: KFPDA, XPFCP, SJPDAF y SJPDAF+NN, se realiza una comparativa empírica entre ellos para así conocer cuál es el que mejor resultados ofrece con los parámetros que se les han establecido.

Para llevar a cabo esta comparativa se prueba cada uno de los algoritmos a lo largo del vídeo 10 completo, anotando todos los errores encontrados. De esta forma se obtendrán resultados más concluyentes al usar un experimento más largo y completo.

En primer lugar se compara la cantidad y tipos de errores que se obtienen con cada uno de los algoritmos. Para ello se comparan los estimadores de dos en dos, el KFPDA con el XPFCP y el SJPDAF con el SJPDAF+NN. Una vez conocido cuál es el que mejor resultados ofrece de cada pareja, se comparan los más fiables para concluir cuál es el que arroja mejores resultados de todos.

Además se realiza una valoración del tiempo de ejecución de cada algoritmo observando el tiempo que tarda cada uno de ellos y encontrando una explicación para los datos encontrados.

4.2. Resultados de la comparativa KFPDA + XPFCP

Para realizar esta comparativa se emplean los parámetros ajustados del KFPDA (Figura 4.1) y del XPFCP (Figura 4.2):

distMKF=800	***canM=2***	***dtR=*** $\sqrt{0.001}$	***dtQ=10***	$P_o = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

Figura 4.1. Tabla con los parámetros ajustados del KFPDA.

nN=100	***nM=70***	***cX=400***	***cY=100***

Figura 4.2. Tabla con los parámetros ajustados del XPFCP.

Además de estos parámetros, es necesario tener ajustados los correspondientes al proceso de clasificación implícito en el XPFCP, cuyo ajuste se realizó en [Cerro07] y que se observan en la Figura 4.3:

distMXPF=700	***canMIn=0***	***canMOut=2***	***factOlv=0.8***	***KLIKM=10***	***hist=0.5***	***um=0.6***

Figura 4.3. Tabla con parámetros de clasificación ajustados en trabajos previos del XPFCP.

Una vez especificado el valor de todos los parámetros se procede a hacer la comparativa entre ambos algoritmos.

En primer lugar se observa en la Figura 4.4 en qué medida ha disminuido la cantidad de errores en el KFPDA y XPFCP al ajustar los parámetros. El primero de ellos es el que más ha mejorado en cuanto a resultados obtenidos, mientras que en el XPFCP los fallos han disminuido ligeramente. Esto nos da una idea de la sensibilidad de cada uno de ellos frente a los correspondientes parámetros.

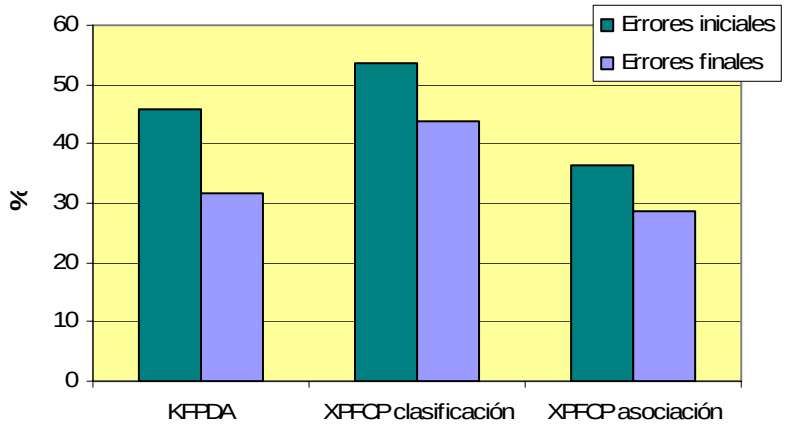


Figura 4.4. Diagrama de barras con la tasa de error antes y después del ajuste de los parámetros del KFPDA y el XPFCP.

La tasa de error que se obtiene con el algoritmo KFPDA después de ajustar todos sus parámetros es de un 31.78%, frente al 46% obtenido inicialmente. Es decir, prácticamente la tercera parte de los errores iniciales desaparecen.

En el caso del XPFCP, primeramente se obtuvo una tasa de fallo del 37.58% en el proceso de asociación y del 55.71% en el proceso de clasificación. Tras ajustar los parámetros, la tasa de error ha bajado a un 28.50% y a un 43.71% en ambos procesos, es decir, una cuarta parte y una quinta parte respectivamente.

Por lo tanto, y como se ha comentado anteriormente, esto implica que el KFPDA es más sensible a la variación de los parámetros que el XPFCP.

En la Figura 4.5 se observan los distintos tipos de error de cada algoritmo tras el ajuste de sus parámetros:

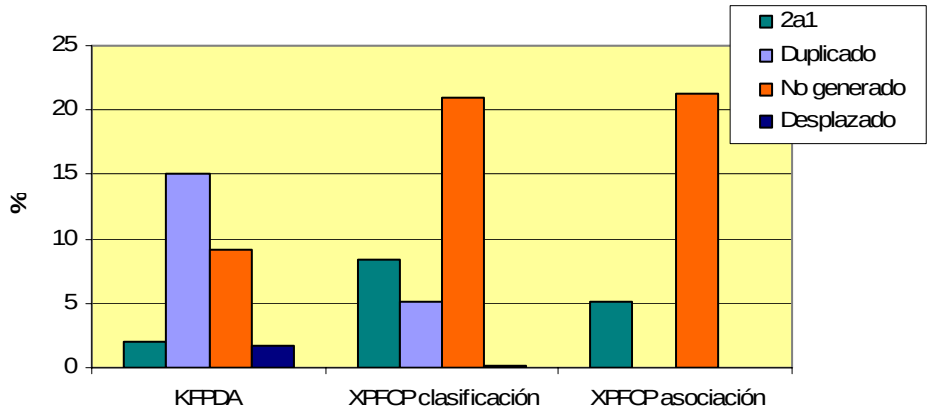


Figura 4.5. Porcentaje de los distintos tipos de error en el KFPDA y el XPFCP con sus parámetros ajustados.

Una vez conocida la mejoría conseguida en cada uno de los algoritmos, se pasa a detallar algunas diferencias importantes entre ellos.

Cada uno de los estimadores tiene un problema que supone la fuente principal de error:

- Con el **KFPDA** se encuentran una gran cantidad de fallos de "duplicado", apareciendo éstos en 164 iteraciones. Sólo este error supone el 47% de la tasa de fallo total que es un 31.78% como se ha comentado con anterioridad. Es decir, casi la mitad del número total de errores corresponde a filtros que han sido duplicados.
- En el **XPFCP** el error fundamental es la falta de partículas en objetos con medidas, dando lugar sólo este fallo al 75% del total de errores en el proceso de asociación. La ausencia de partículas tiene como consecuencia la falta de clases. Estos fallos en el proceso de clasificación aparecen en 236 ocasiones distintas, dando lugar a un 48% de la tasa total de error que es de un 43.71%.

De lo anteriormente dicho se deduce que la cantidad de clases validadas en el XPFCP debe ser menor de media al número de filtros que son validados en cada iteración por el KFPDA, al ser precisamente los errores principales la falta y el duplicado de clases/filtros respectivamente.

Esto se puede observar en la Figura 4.6 donde en prácticamente todas las iteraciones la cantidad de filtros del KFPDA supera a la de clases del XPFCP:

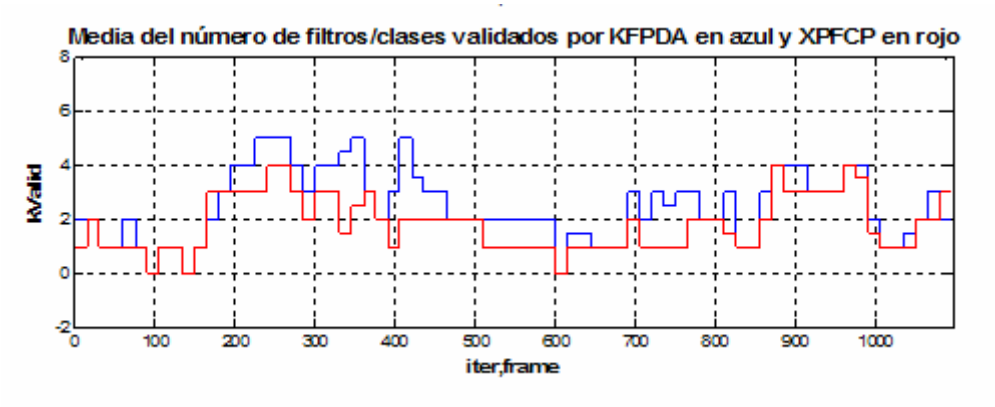


Figura 4.6. Filtros/clases validados por el algoritmo KFPDA y XPFCP respectivamente.

Como consecuencia del elevado número de filtros duplicados en el KFPDA, su tiempo de ejecución es, lógicamente, mayor que el del XPFCP, cuestión que se analiza en detalle en el apartado 4.5.

Teniendo en cuenta que el error "no generado" es considerado como más importante que el "duplicado", y debido también a que la tasa de error del KFPDA no alcanza el 32% frente al 43.71% que se obtiene con el XPFCP en la clasificación y el 30% de error en la asociación, se escoge el KFPDA como algoritmo que ofrece mejores resultados de los dos comparados. En el apartado 4.4 se hace una comparación de este último con el que arroje mejores resultados del par comparado en el siguiente apartado.

4.3. Resultados de la comparativa SJPDAF + SJPDAF+NN

Para realizar esta comparativa se prueban los algoritmos SJPDAF y SJPDAF+NN con todos sus parámetros ajustados, tal y como se muestra en el apartado anterior.

Los valores asignados a los parámetros del SJPDAF y del SJPDAF+NN se observan en la Figura 4.7 y la Figura 4.8 respectivamente:

canM=2	***cY=50***	***cX=10***	***nN=600***	***nM=200***

Figura 4.7. Tabla con los parámetros ajustados del SJPDAF.

canM=2	***cY=50***	***cX=100***	***nN=600***	***nM=200***

Figura 4.8. Tabla con los parámetros ajustados del SJPDAF+NN.

En la Figura 4.9 se observa cómo han variado la cantidad de errores del SJPDAF y SJPDAF+NN antes y después de ajustar sus parámetros (en la prueba larga).

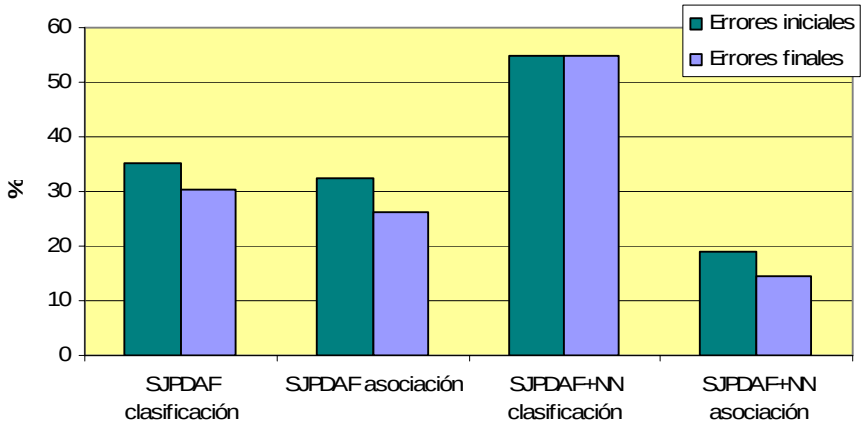


Figura 4.9 Porcentajes de errores antes y tras el ajuste de los parámetros del SJPDAF y el SJPDAF+NN.

La tasa de error del SJPDAF en la prueba larga ha variado de 32.33% a 26.37% en la asociación de partículas y de un 35.32% a un 30.34% en la clasificación a la salida. La sensibilidad de este algoritmo es por tanto baja frente a la modificación de los parámetros correspondientes.

El SJPDAF+NN poseía una tasa de error de un 18.90% y un 54.72% en los procesos de asociación y clasificación. Tras ajustar sus parámetros estos porcentajes se han quedado en un 14.42% y 54.72%. Es decir, prácticamente no existe sensibilidad alguna de este estimador frente a la alteración de las variables cualitativas.

Se puede afirmar entonces que tanto el SJPDAF como el SJPDAF+NN muestran poca sensibilidad frente a la variación del valor de los parámetros.

En la Figura 4.10 se observan los distintos tipos de error en el proceso de clasificación de ambos estimadores una vez que todos los parámetros están ajustados:

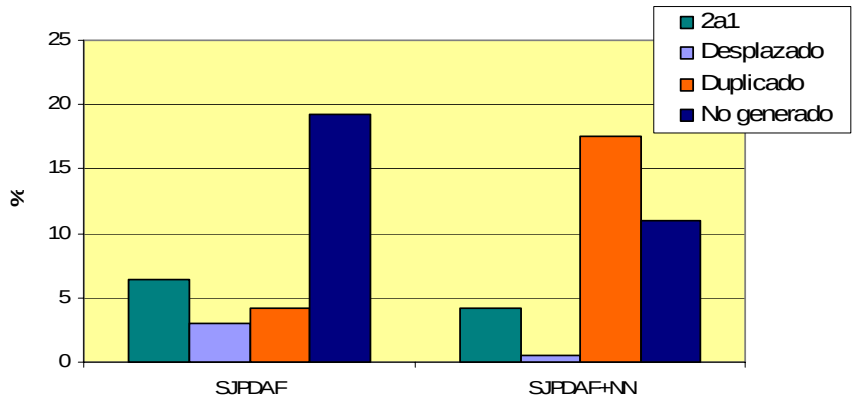


Figura 4.10. Porcentaje de errores del SJPDAF y SJPDAF+NN después del ajuste de los parámetros.

Se comentan ciertos aspectos de ambos algoritmos que se han observado al realizar esta prueba:

- **SJPDAF**: Los errores "no generado" en la asociación de partículas suponen un 24.3% de errores. Por otra parte los errores más importantes en el proceso de clasificación son los "no generado" y "duplicado".
- **SJPDAF+NN**: Anteriormente ya se explicó que el PF es un método no repetibles y aleatorio, es decir, si se vuelve a probar el algoritmo ante el mismo set de entrada los resultados no tienen por qué ser los mismos. Esto se ha comprobado al llevar a cabo esta última prueba ya que la proporción de errores ha aumentado considerablemente y algunos tipos de errores disminuyen o aumentan su aparición.

  La tasa de error por ausencia de partículas ("no generado") en la asociación se mantiene alrededor de un 18%.

  El hecho de que aparezcan clases duplicadas o triplicadas supone un 21.10%.Los errores de "no generado" en la clasificación dan lugar al 13.8% de tasa de error.

En la Figura 4.11 se puede apreciar cómo la cantidad de clases validadas en el SJPDAF+NN es mucho mayor que la generada por el SJPDAF debido a los errores de duplicación:

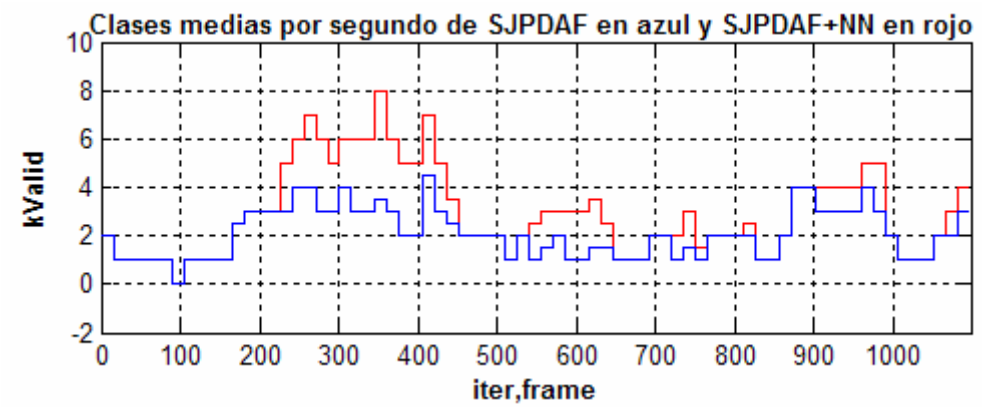


Figura 4.11. Cantidad media de clases de SJPDAF y SJPDA+NN.

Se muestra en la Figura 4.12 la cantidad de errores totales (en la totalidad del vídeo 10) de ambos algoritmos, tanto en el proceso de asociación como en la clasificación.

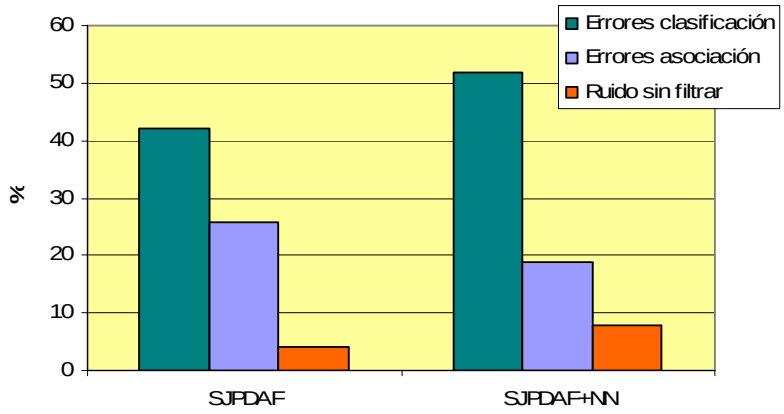


Figura 4.12. Diagrama de barras con la tasa de error tanto en el SJPDAF como en SJPDAF+NN.

Por último se muestra en la Figura 4.13 una tabla con los porcentajes de fallo debido a los diferentes errores encontrados en los distintos algoritmos:

	SJPDAF	***SJPDAF+NN***
No generados	*23.30*	*13.80*
Duplicados	*10.29*	*21.10*
Partículas	***24.30***	***18***

Figura 4.13. Tabla con los porcentajes de los errores más importantes en SJPDAF y SJPDAF+NN.

La tasa de error en la clasificación del SJPDAF ya con todos sus parámetros ajustados es de un 41.89%, mientras que la del SJPDAF+NN asciende a un 51.82%. Por otra parte el porcentaje de fallo en la asociación de las partículas es de un 25.77% y 18.94% para el SJPDAF y SJPDAF+NN respectivamente.

A la vista de todas estas apreciaciones se selecciona el SJPDAF puro como el mejor estimador de los dos analizados. A primera vista el SJPDAF+NN parece más eficiente ya que los errores “no generado” son mucho menores. El problema es que la aparición de clases duplicadas, triplicadas e incluso cuadriplicadas es demasiado frecuente y el tiempo de ejecución se hace muy elevado como se verá más adelante.

La cantidad de errores generados por el SJPDAF en el proceso de clasificación es mucho menor, además se filtran mejor los ruidos. La desventaja de elegirlo es que los fallos en la asignación de partículas son más elevados que en el SJPDAF+NN.

4.4 Resultados de la comparativa KFPDA + SJPDAF

De las dos comparativas realizadas con anterioridad se han escogido el KFPDA y el SJPDAF como los dos algoritmos que mejores resultados ofrecen. Se hace ahora una valoración para decidir cuál de estos dos resulta más eficiente.

A continuación, en la Figura 4.14 se muestran los porcentajes debidos a los distintos fallos en el proceso clasificador encontrados con el KFPDA y el SJPDAF:

	KFPDA	***SJPDAF***
No generados	*13.03*	*23.30*
Duplicados	*18.98*	*10.29*

Figura 4.14. Porcentajes de error en los algoritmos KFPDA y SJPDAF.

En primer lugar se observa en la Figura 4.14 que el KFPDA genera menos fallos "no generado" ya que únicamente en un 13% de las iteraciones aparecen objetos con medidas que no se encuentran validados frente al 23% del SJPDAF. En cambio se generan menos errores de duplicación de clases en este último que en el KFPDA, concretamente un 8% menos.

En la Figura 4.15 se percibe con más facilidad la diferencia en la cantidad de errores generados por ambos estimadores en la clasificación de las partículas a la salida:

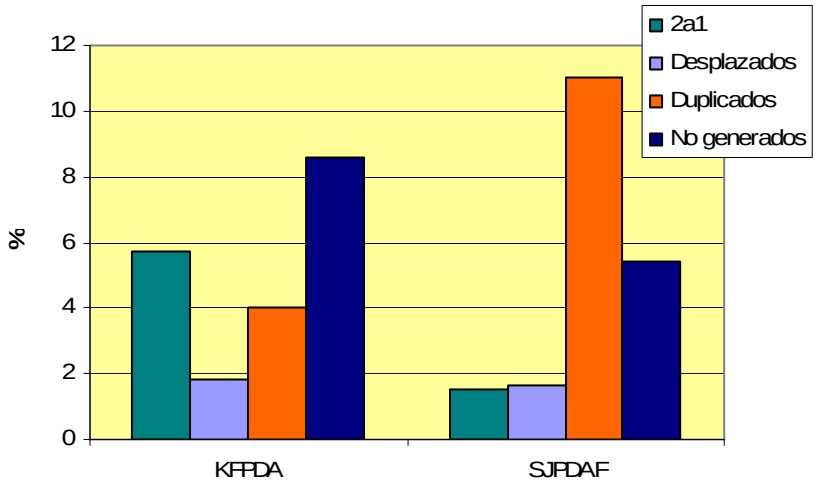


Figura 4.15. Cantidad de distintos errores en KFPDA y SJPDAF.

La diferencia en la cantidad de filtros/clases validados debe ser, por tanto, importante. Esto se puede comprobar en la Figura 4.16 que se muestra a continuación, donde se observa que en la mayoría de las iteraciones el KFPDA genera más filtros que el SJPDAF:

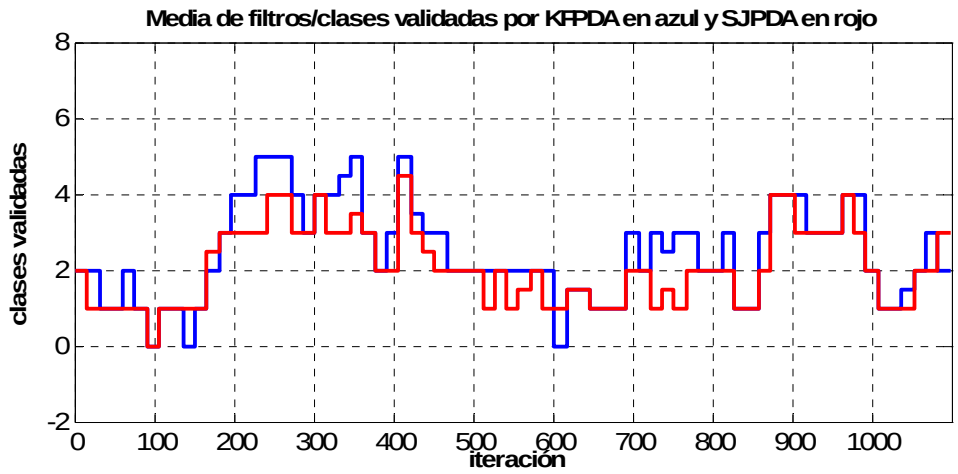


Figura 4.16. Media por segundo de clases validas por los algoritmos KFPDA y SJPDAF.

Por último se muestra en la Figura 4.17 una tabla con la tasa de fallos totales que generan ambos algoritmos:

	KFPDA	***SJPDAF***
Errores	*31.70*	*41.89*

Figura 4.17. Tabla con el porcentaje de errores de KFPDA y XPFCP.

Teniendo en cuenta lo anteriormente comentado se escoge el KFPDA como mejor de los dos algoritmos por los motivos que se exponen a continuación:

- Genera menor número de fallos en la clasificación.
- Genera menor número de errores “no generado” tanto en la asociación como en el proceso clasificador, error que es considerado como el más importante.
- La tasa de error del KFPDA es inferior al 40% mientras que la de la clasificación del SJPDAF es de un 41.89%.

4.5. Comparativa de los tiempos de ejecución

El tiempo de ejecución de cada uno de los algoritmos analizados es muy diferente. En algunos de ellos hay implícitas operaciones que ralentizan el proceso.

Para analizar el tiempo de ejecución de cada estimador se usa el mismo procedimiento que en la anterior comparativa, es decir, compararlos de dos en dos para concluir en el que menor tiempo de ejecución emplee.

a) Tiempo de ejecución del KFPDA y el XPFCP

En la Figura 4.18 se muestran los tiempos de ejecución empleados por los algoritmos KFPDA y XPFCP en el experimento del vídeo 10 completo:

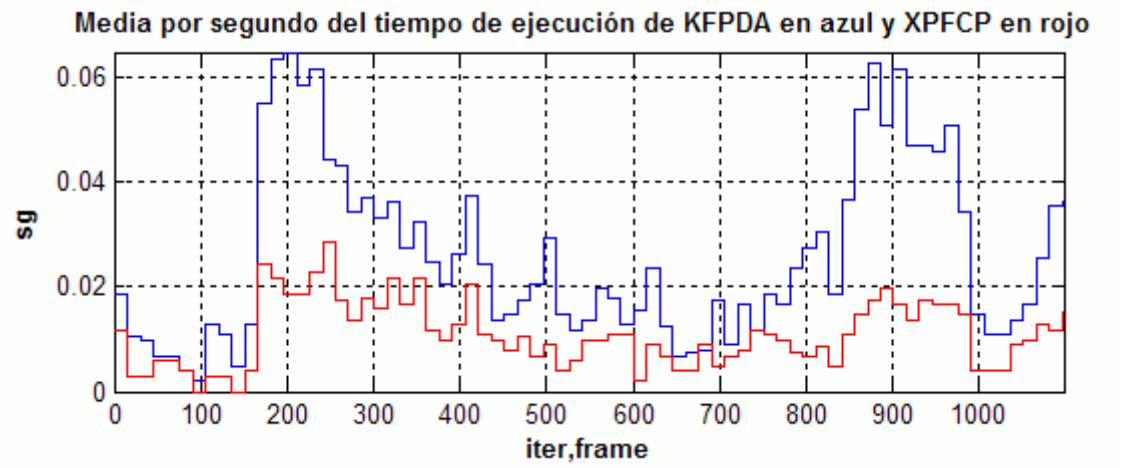


Figura 4.18. Gráfico de los tiempos de ejecución de los algoritmos KFPDA y XPFCP.

Se puede ver que el tiempo medio de ejecución del KFPDA es mucho más elevado que el del XPFCP. Esto es debido a que en el algoritmo del KFPDA existen muchos cálculos matriciales (sobre todo los correspondientes a las fases del filtro de Kalman) y principalmente al proceso de asociación PDA en el que se realizan cálculos de distancias entre todas las medidas y centroides.

b) Tiempo de ejecución del SJPDAF y SJPDAF+NN.

El tiempo de ejecución de estos estimadores es muy diferente al del KFPDA y al del XPFCP. Aparte de los cálculos correspondientes al PDA que ralentizan mucho el proceso, en estos dos algoritmos existen otros en los que intervienen todas las partículas y todas las medidas. Concretamente se lleva a cabo el cómputo de la distancia de cada partícula a cada medida, algo que ralentiza la ejecución, y posteriormente se calcula la probabilidad con cada partícula.

La diferencia entre el tiempo de ejecución de los dos estimadores en el experimento del vídeo 10 completo es mínima ya que ambos emplean la misma cantidad de partículas al tener el parámetro *n* fijado al mismo valor.

Lo anteriormente comentado se puede comprobar en la Figura 4.19:

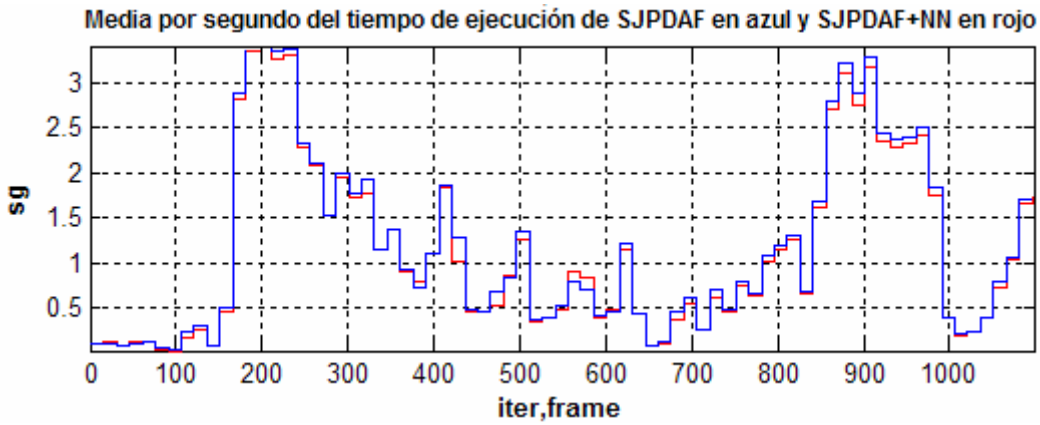


Figura 4.19. Tiempo de ejecución de los algoritmos SJPDAF y SJPDAF+NN.

Finalmente se compara el tiempo de ejecución del XPFCP y del SJPDAF en la Figura 4.20, observándose la gran diferencia que existe entre ambos. Aproximadamente el tiempo de ejecución del SJPDAF es 60 veces el del XPFCP.

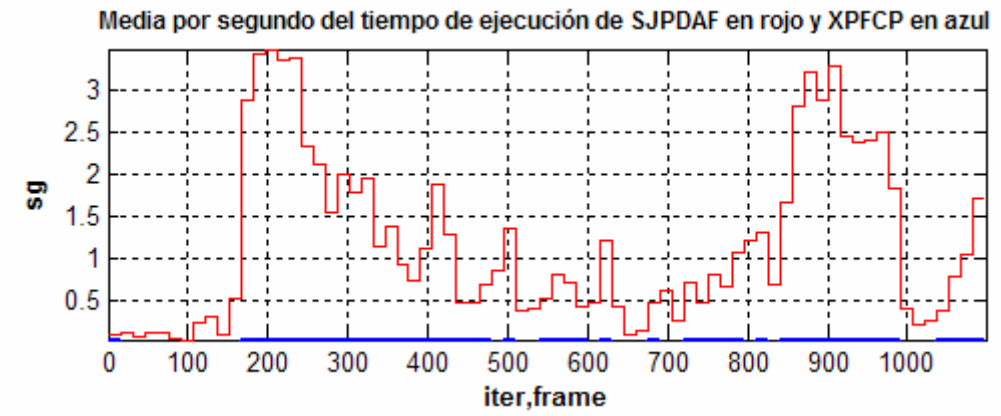


Figura 4.20. Tiempo de ejecución de los algoritmos XPFCP y SJPDAF.

Al ser el tiempo de ejecución del SJPDAF tan elevado frente al del XPFCP, éste prácticamente no se puede observar en el gráfico.

El algoritmo que emplea menos tiempo medio en cada iteración de los cuatro analizados es, por tanto, el XPFCP.

4.6. Comparativa en el filtrado del ruido

En muchas ocasiones aparecen medidas que no han sido extraídas de ningún objeto sino que pueden ser considerados ruidos espúreos en el set de medidas. Hay que filtrarlas y no considerarlas en el proceso de seguimiento.

En la Figura 4.21 se ven algunos ejemplos de ruidos que aparecen en los vídeos utilizados: el cambio de color de la pared, la columna del fondo a la derecha, la puerta situada al fondo del pasillo o el cambio de color del suelo.

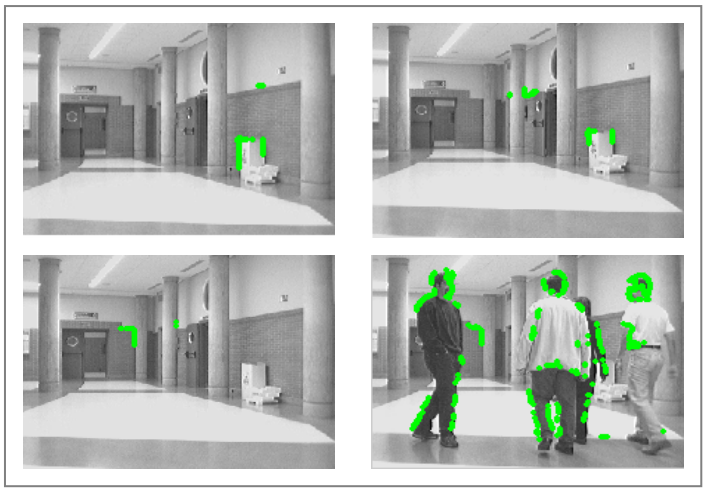


Figura 4.21.Medidas extraídas de objetos considerados como ruidos.

Lo ideal sería que estas medidas nunca dieran lugar a un filtro o clase validado pero esto no ocurre siempre ya que la validación depende de la verosimilitud, de la probabilidad de esas medidas, de si están próximas o no a la posición predicha de algún objeto, etc.

A la hora de analizar los errores de los algoritmos se han anotado también las iteraciones en las que se encuentra algún tipo de ruido sin filtrar. De esta forma se ha podido utilizar esta característica para escoger el mejor de los ajustes.

La validación de ruidos es distinta en los diferentes algoritmos como se va a ver a continuación.

En la Figura 4.22 se puede ver gráficamente el porcentaje de iteraciones con ruidos validados en los diferentes algoritmos:

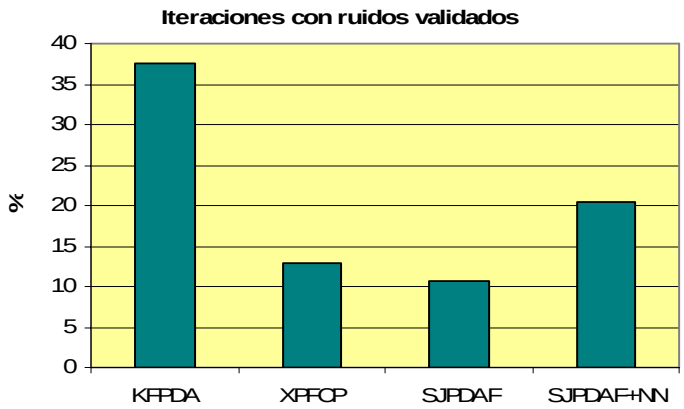


Figura 4.22. Porcentaje de iteraciones en las que el ruido no es filtrado.

El **KFPDA** no filtra los ruidos en un 37.50% de las ocasiones en las que aparecen estas medidas. Esto es debido a que en dicho algoritmo la validación de filtros depende principalmente del parámetro *canM* y de si en la iteración anterior existía dicho filtro o no. Por lo tanto cuando aparece ruido cerca de la columna por ejemplo, si aparece en *canM* iteraciones consecutivas el objeto probablemente será validado, teniendo en cuenta que también depende de la distancia entre dichas medidas y la posición predicha del objeto correspondiente.

Por su parte con el estimador **XPFCP** únicamente genera clases validadas asociadas a ruidos en un 13% de las iteraciones con algún tipo de ruido, resultado mucho mejor que el arrojado por el KFPDA.

Para que una clase sea validada o eliminada intervienen parámetros como el número de medidas, la probabilidad, distancias, etc. Al depender de más factores este proceso es más selectivo con los ruidos existentes.

El **SJPDAF** tiene como algoritmo de asociación de entrada el PDA y a la salida el K-Medias. Por lo tanto el proceso de validación a la salida es el mismo que en el XPFCP.

Con este estimador se han encontrado en el 10.73% de las iteraciones con ruido clases validadas correspondientes a dichos ruidos, siendo éste el mejor de los casos.

Por último el SJPDAF+NN, al igual que el SJPDAF y el XPFCP posee como proceso de asociación a la salida el k-Medias.

En este caso en el 20.52% de las ocasiones el ruido no se ha filtrado.

El algoritmo que realiza un mejor tratamiento con los ruidos es el SJPDAF al filtrarlos en más ocasiones. Por el contrario el KFPDA, al ser su proceso de validación poco restrictivo, es el que en mayor número de ocasiones valida los filtros de dichos ruidos.

4.7. Factores de calidad

4.7.1. Análisis de la matriz de covarianza del error de estimación en el filtro de Kalman

Con el fin de comprobar si el filtro de Kalman converge rápida o lentamente se realiza el siguiente experimento:

Se escoge un fragmento del vídeo 10, 100 iteraciones a partir de la 685, en el que sólo aparece un objeto dinámico. Se anota, iteración a iteración, los valores de la matriz de covarianza del error de estimación **corregida** (*P_corrected*) de este elemento. Esta matriz es empleada como factor de calidad para indicarnos si el KF implementado es correcto o no.

Este filtro tiene como valor inicial $P_x, P_z = 0.99900099900085 \cdot 10^{-3}$, donde P_x y P_z son los valores de la diagonal de la matriz *P*, correspondientes al valor en la coordenada *x* y en la *z*.

Este valor disminuye hasta que en la cuarta iteración del estudio se estabiliza con un valor de $P_x, P_z = 0.9990001999456 \cdot 10^{-3}$ que se mantiene a lo largo de las 100 iteraciones analizadas.

En primer lugar se comprueba que el filtro responde con rapidez, se estabiliza con prontitud. En segundo lugar se aprecia que el valor de la matriz de covarianza del error de predicción disminuye a medida que se estabiliza el filtro ya que cada vez es más estable y la estimación es más correcta.

4.7.2. Número eficaz de partículas en el XPFCP, SJPDAF y SJPDAF+NN

El número eficaz de partículas (*neff*) corresponde al número de partículas que deben ser empleadas y se define de la siguiente manera ([MacCormick99]):

$$neff = \frac{1}{\sum_{i=1}^{n} w_i^2} \tag{4.1}$$

donde *n* es el número total de partículas y *w* el peso de cada una de ellas.

Si todas tuvieran el mismo peso *neff=n*, pero si por el contrario todas las partículas menos una tuvieran peso 0 *neff=1*. En cualquier otro caso *neff* se encontraría entre estos dos valores.

El parámetro *neff* nos da, por tanto, información de las partículas que van a sobrevivir tras le paso de reinicialización del PF. Es decir, cuanto mayor sea el número eficaz de partículas mejor, significando esto que los pesos de todas ellas son semejantes.

Un número más o menos aceptable sería que el número eficaz fuese la mitad del número total de partículas, $neff = 1/2 \cdot n$.

Para hacer el estudio del número eficaz de partículas se ha calculado *neff* en cada iteración de la prueba larga del vídeo 10, al ser ésta más completa.

En la Figura 4.23, Figura 4.24 y Figura 4.25 se muestra el número eficaz de partículas en el XPFCP, el SJPDAF y el SJPDAF+NN:

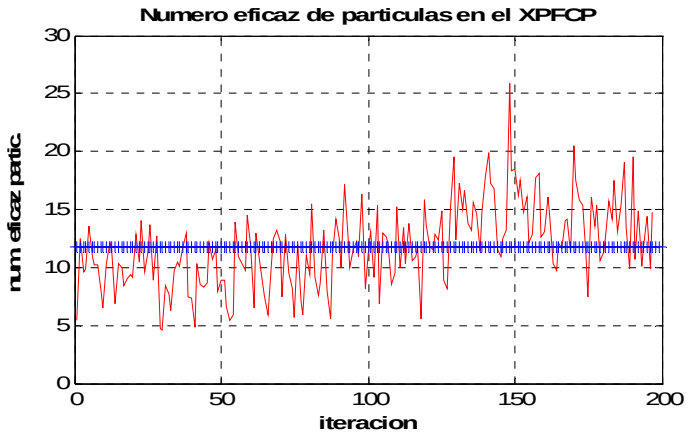


Figura 4.23. Número eficaz de partículas en cada iteración (rojo) y la media (azul) del XPFCP.

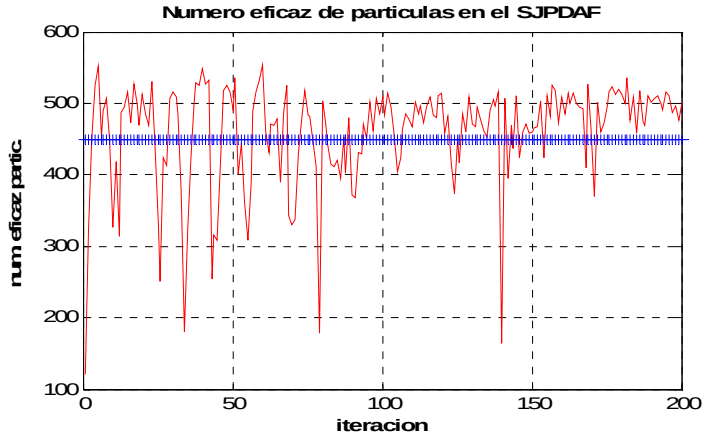


Figura 4.24. Número eficaz de partículas en cada iteración (rojo) y la media (azul) del SJPDAF.

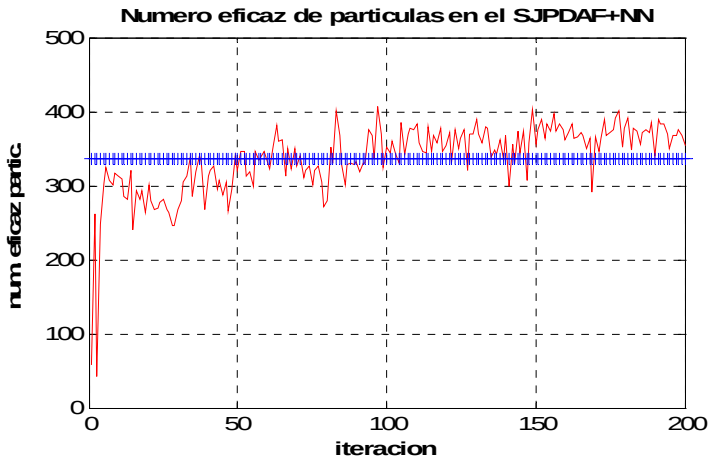


Figura 4.25. Número eficaz de partículas en cada iteración (rojo) y la media (azul) del SJPDAF+NN.

- En el XPFCP el número total de partículas (*n*) es 100. Se comprueba que el número eficaz es aproximadamente una cuarta parte de *n*, valor demasiado pequeño que informa de que el algoritmo empleado no es muy eficiente.
- El número total de partículas empleado en el SJPDAF es de 600 y el número eficaz está en el rango de las 400 y 500 partículas, es decir, mayor que $n/2$. Esto implica que las estimaciones del algoritmo son precisas.
- El SJPDAF+NN utiliza 600 partículas también. El número eficaz de miembros supera la mitad del número total, por lo tanto es un valor aceptable, aunque algo peor que el obtenido con el SJPDAF.

Es decir, el algoritmo que ofrece más certeza y fiabilidad es el SJPDAF al ser el valor de *neff* muy alto. Se puede decir, pues, que es un estimador de calidad.

4.7.3. Trayectorias de los objetos con cada uno de los algoritmos

Como última comparativa se ha representado gráficamente la trayectoria de un objeto hallada con los distintos algoritmos, con el fin de ver la semejanza entre ellas.

Algo realmente interesante sería poder comprobar el error que comete cada estimador a lo largo de toda la trayectoria, pero esto no es posible al no tener información de la posición real en cada momento.

Para realizar este estudio se han empleado los ficheros *.mat* obtenidos en los ajustes previos, en los que se dispone de los listados de identificadores, posiciones, etc. Concretamente se han usado los correspondientes a la prueba larga del vídeo 10.

Se traza la trayectoria de un objeto dinámico que permanece a lo largo de toda la prueba obteniendo el resultado que se muestra en la Figura 4.26:

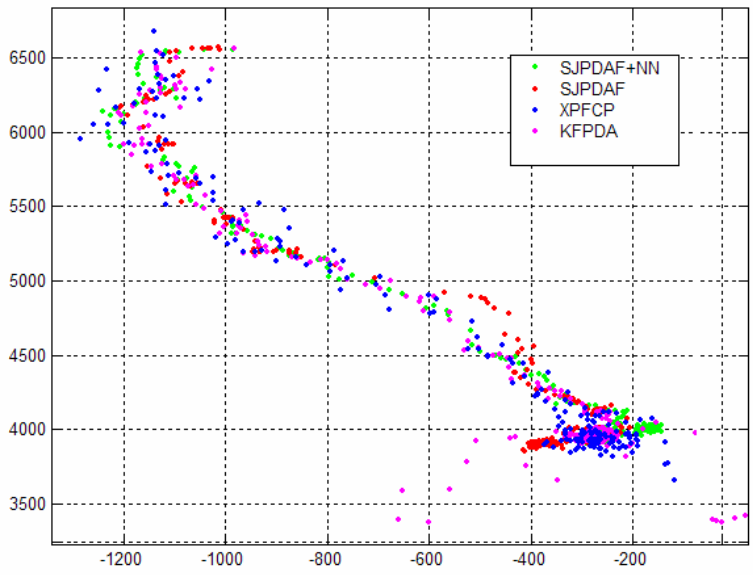


Figura 4.26. Trayectoria de un objeto representada para los distintos algoritmos en la prueba larga.

Las posiciones marcadas por cada algoritmo en cada instante de tiempo son aproximadamente semejantes a lo largo de todo el movimiento. Únicamente al principio de la trayectoria son muy distintas unas de otras.

Además se comprueba que los puntos de la posición obtenidos con el XPFCP son los que están más alejados del resto (la posición del objeto se desvía constantemente), algo de esperar tras ver en el anterior estudio del número eficaz de partículas que el XPFCP no es demasiado eficiente.

Por otra parte, se observa una zona de la trayectoria en la que el SJPDAF establece posiciones alejadas del resto, desviándose los puntos de la posición del objeto temporalmente.

Los algoritmos que trazan caminos más semejantes y continuos son el SJPDAF+NN y el KFPDA como se puede comprobar.

CAPÍTULO 5. CONCLUSIONES Y TRABAJOS FUTUROS

5.1. Conclusiones

La finalidad de este proyecto ha sido comprobar la eficiencia de los algoritmos en función de los distintos parámetros. Por otra parte se han ajustado dichos parámetros para obtener los mejores resultados en la tasa de error de estos estimadores.

En primer lugar es necesario aclarar que la mayor o menor influencia de un determinado parámetro en un algoritmo depende, principalmente del parámetro en sí; pero además el orden en el que realizan las pruebas es también fundamental.

Una vez terminado el estudio se obtienen las siguientes conclusiones de cada uno de los estimadores:

- ***KFPDA***: Este algoritmo es muy sensible a la variación del límite del *gating* (*distMKF*), el parámetro fundamental en la creación y eliminación de filtros (*canM*) y la varianza del ruido de las medidas (dtR^2). Por otra parte la modificación de la varianza del ruido del modelo (dtQ^2) y de la matriz de covarianza del error de estimación inicial (P_o) no supone grandes cambios en el comportamiento del estimador. El ruido es filtrado en muy pocas ocasiones.
- ***XPFCP***: Este algoritmo no presenta demasiados cambios al modificar el número total de partículas (*n)* debido a que no se ha variado el valor del parámetro que indica el número de partículas que se introducen en el paso de reinicialización (n_m). Respecto al resto de parámetros sí se muestra muy sensible ya que variaciones en ellos tienen como

consecuencia modificaciones en el comportamiento del algoritmo. Estos parámetros son n_m y las desviaciones típicas del ruido tanto del modelo como de las medidas (*cX*, *cY*). El ruido se filtra en muchas ocasiones.

- ***SJPDAF:*** Este estimador no es muy sensible a la variación de *n* y de n_m ya que a diferencia del caso anterior, la cantidad de partículas que son introducidas en el paso de reinicialización se aumenta a medida que aumenta *n*. Las desviaciones típicas de los ruidos del modelo y de las medidas sí influyen, y en gran medida, en los resultados obtenidos con este algoritmo. El filtrado del ruido es el mejor de los cuatro casos analizados.
- ***SJPDAF+NN:*** En el funcionamiento de este algoritmo todos los parámetros analizados influyen: *cX*, *cY* y *n* principalmente. Por otra parte n_m produce menos variaciones en los resultados obtenidos. Se obtienen unos resultados intermedios en cuanto al filtrado de los ruidos.

Por otro lado, el tiempo de ejecución es mínimo con el XPFCP y máximo con los SJPDAFs. En una aplicación en tiempo real el tiempo de ejecución es un factor fundamental a tener en cuenta, ya que si es alto es poco probable que su funcionamiento sea el correcto. Por tanto sería difícil emplear los SJPDAFs en aplicaciones de este tipo, siendo más recomendable usar el XPFCP o el KFPDA.

5.2. Trabajos futuros

a) Aplicaciones en la industria de los algoritmos estudiados

El objetivo del desarrollo, estudio e implementación de estos algoritmos es poder aplicarlos en un espacio inteligente con el fin de realizar el seguimiento de personas, robots u otros elementos en entornos cerrados.

Se pueden emplear estos estimadores en aplicaciones de seguridad, control de acceso, control del robot, aprendizaje de rutinas de los humanos, etc. También en un hospital para el control de enfermos, en un centro de día para personas con discapacidad, de la tercera edad.

Además se puede utilizar el seguimiento para que un robot sea capaz de modificar trayectorias y evitar obstáculos.

b) Mejora en el filtrado del ruido del KFPDA

Una posible mejora en el filtrado del ruido consiste en que en el momento de crearse un objeto nuevo en el PDA, si a éste se le asocian un número muy bajo de medidas, no entre en el proceso de validación, sino que se espere a la siguiente iteración para comprobar si se le asocian más o ninguna medida. De este modo, y ya que normalmente aparecen pocas medidas extraídas de objetos considerados como ruido, se conseguiría el mejor filtrado de los mismos. La desventaja es que en ocasiones, un objeto nuevo que ha de considerarse aparece con pocas medidas inicialmente, por lo que podrían surgir más errores del tipo “no generado”. Habría que

realizar un estudio para comprobar si este cambio implica mejoría, o por el contrario, se aumenta el número de errores.

Un proceso similar a éste se encuentra en el proceso de clasificación K-Medias, donde la probabilidad de la clase *P* se calcula en función del número de partículas asociadas al objeto. Esta probabilidad se usa posteriormente como condición de validación e invalidación de clases, por lo que es poco probable que la clase de un objeto considerado como ruido sea validada.

c) Mejora en el ajuste de parámetros de los algoritmos

Existen algunos parámetros que, una vez que se tiene una visión global de su influencia en los distintos estimadores, podrían mejorarse.

En primer lugar todos los algoritmos analizados presentan mucha sensibilidad al parámetro *distM*, parámetro fundamental a la hora de realizar una asociación correctamente. En segundo lugar sería óptimo calcular la varianza de cada objeto en cada iteración. De esta forma se podría modificar *distM* de cada uno de los objetos en función de la varianza y así se iría adaptando el algoritmo a los distintos objetos. La desventaja es que el tiempo de ejecución aumentaría.

En el XPFCP una mejoría evidente es aumentar el número de partículas empleadas en el proceso de reinicialización n_m a medida que se aumente el número total de partículas *n*. Esto no se hizo en un primer instante para comprobar cómo responde un algoritmo de este tipo al disponer de pocas partículas y ajustar el resto de sus parámetros.

c) Mejora en el PDA y el K-Medias mediante el uso de la distancia de Mahalanobis

Como se ha explicado en anteriores ocasiones, las medidas que se obtienen del sistema de visión corresponden al contorno de los objetos. Por tanto estas medidas están más dispersas a lo largo del eje *x* que a lo largo del eje *z*, como se puede ver en la Figura 5.1.

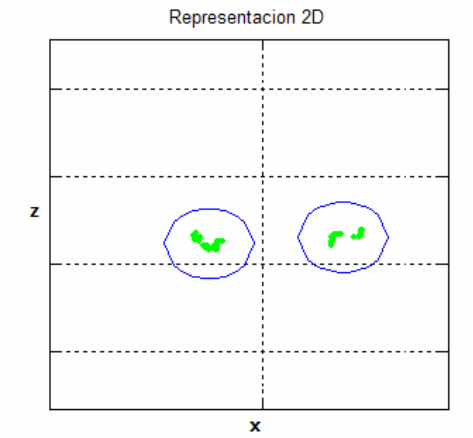


Figura 5.1. Representación en el plano XZ de las medidas de dos objetos.

Para realizar la asociación en el PDA y en el K-medias se emplea la distancia euclídea y se modela a los objetos como cilindros, que en el plano *XZ* se representan como círculos.

A la vista de la dispersión de las medidas se puede usar, en un trabajo futuro, la distancia de Mahalanobis para la asociación, introducida por P. C. Mahalanobis en 1936. Esta distancia es una extensión de la distancia euclídea que surge por la necesidad de determinar la similitud entre dos variables aleatorias multidimensionales.

La diferencia de la distancia de Mahalanobis y la euclídea es que tiene en cuenta la correlación entre datos y se calcula como se muestra a continuación:

$$d_{Mahalabobis|i,j} = \sqrt{\left(\vec{y}_i - \vec{g}_j\right)' \Omega_{i,j}^{-1} \left(\vec{y}_i - \vec{g}_j\right)} \tag{5.1}$$

$$d_{Mahalabobis|i,j} = \sqrt{\sum_{\lambda=\lambda_1}^{\lambda_T} \frac{\left(a_{\lambda,i} - b_{\lambda,j}\right)}{\sigma_\lambda^2}} \tag{5.2}$$

En (5.1) se computa la distancia entre la variable $\vec{y}_i$ y $\vec{g}_j$, y $\Omega_{i,j}$ es su matriz de covarianza. En la ecuación (5.2) se muestra la simplificación de la anterior para el caso en que las variables implicadas estén incorreladas, situación en la que la matriz de covarianza es diagonal, donde λ indica la cantidad de dimensiones del espacio y *a* y *b* son las coordenadas (coordenada *x* y coordenada *z*) de las variables $\vec{y}_i$ y $\vec{g}_j$ respectivamente.

Por lo tanto, en nuestro modelo tendríamos una varianza del modelo del vector de estado distinta para la coordenada *x* (σ_x^2) y para la *z* (σ_z^2), al igual que pasaría con la varianza del ruido asociado a las medidas.

La distancia de Mahalanobis del punto i al punto j aplicado a nuestro estudio en el caso en que las medidas en *x* y en *z* estuvieran incorreladas se calcula de la siguiente manera:

$$d_{Mahalanobis|i,j} = \sqrt{\left(\frac{x_i - x_j}{\sigma_x}\right)^2 + \left(\frac{z_i - z_j}{\sigma_z}\right)^2} \tag{5.3}$$

De esta forma los objetos quedarían modelados por elipses y se mejoraría el proceso de asociación de datos.

d) Utilización de sensores de identidad para identificar a los objetos correctamente

Mediante el uso de una red de sensores colocados en un entorno la localización de personas se puede realizar de forma eficiente.

Existen aplicaciones como por ejemplo el control de personas de la tercera edad en centros especializados que requieren una exhaustiva identificación de los objetos ([Wilson05]). Si se tienen correctamente identificadas a las personas se pueden ver qué habitaciones del centro están ocupadas, qué personas hay en su interior, comprobar si se están moviendo o no, ver a dónde se dirigen en caso de que se desplacen, etc.

Para conseguir esta correcta localización ([Schulz03]) y evitar confusiones en el seguimiento de objetos se emplean infrarrojos, láser, cámaras, GPS, micrófonos, etc.

De esta forma, si además de los estimadores empleados incluimos sensores en los objetos a seguir se tiene en todo momento información de su posición. Así en un cruce por ejemplo, no existe confusión sobre qué objeto es cada uno y se puede asegurar su correcto seguimiento.

III. MANUAL DE USUARIO

En esta sección se explica con detalle cómo ejecutar e interpretar los algoritmos de estimación de la posición que han sido desarrollados.

Para la ejecución de los estimadores en los videos se necesita ajustar previamente los datos intrínsecos y extrínsecos del sistema de estéreo-visión.

El formato que se da al video es 320 x 240. Se configura un nivel de gris en cada cámara de tal forma que se tiene un dato en binario (entre 0 y 255) con el nivel de gris de cada píxel. Para cargar las imágenes del video se utiliza la función *fread.*

Para ejecutar los distintos algoritmos se tienen los siguientes ficheros:

- *KFPDA_Total.m:* Este fichero se utiliza para ejecutar el algoritmo KFPDA únicamente.
- *XPFCP_Total.m:* Este fichero se emplea para ejecutar el algoritmo XPFCP.
- *FiltroTotal.m:* Cuando se llama a este fichero se ejecutan paralelamente los algoritmos KFPDA y XPFCP. De este modo se comprueba visualmente las diferencias en los resultados que ofrecen ambos.
- *SJPDAF_Total.m*: Al ejecutar este archivo comienza la ejecución del algoritmo SJPDAF.
- *SJPDAF_NN_Total*: Este fichero se utiliza para ejecutar el SJPDAF+NN.

Como se puede ver no se ha desarrollado un fichero para que el SJPDAF y el SJPDAF+NN sean ejecutados al mismo tiempo, y esto es debido fundamentalmente a la siguiente razón: si se ejecutaran a la vez los dos algoritmos el tiempo de ejecución sería demasiado elevado ya que como se vio en el Capítulo 4 ambos algoritmos emplean un tiempo muy alto.

Todos estos archivos reciben los mismos parámetros de entrada y devuelven los mismos archivos que se detallan a continuación:

- PARÁMETROS DE ENTRADA:

  - *file*: Número del archivo del vídeo que se quiere abrir. Únicamente es necesario introducir dicho número, no el nombre completo. Si se quiere ejecutar el vídeo 10 se debe emplear el archivo *data010.dat* que contiene los datos de las medidas y el archivo *left010.str* que contiene la información del vídeo; y para hacerlo debemos insertar como parámetro '010'.

  - *videoOn*: Parámetro utilizado para que la secuencia del vídeo sea mostrada o no junto con las medidas, las partículas en caso de que las haya y el resultado de la estimación y la asociación. Si se quiere visualizar el vídeo hay que pasar como parámetro un '1', de lo contrario un '0'. El hecho de que se muestre el vídeo en cada iteración ralentiza la ejecución en gran medida.

  - *plotOn*: Este parámetro da la posibilidad de obtener una gran cantidad de gráficas relacionadas con el funcionamiento del algoritmo. Para activarlo se pasa un '1' y de lo contrario un '0'.

  - *saveOn*: Si se marca este parámetro con un '1' se procederá a guardar en un archivo *.avi* el vídeo con la proyección 2D y el vídeo con las imágenes en 3D; y también un archivo *.mat* con gran cantidad de información guardada de distintas variables.

- INFORMACIÓN GENERADA

  - Archivo *.mat* que contiene información de variables en cada una de las iteraciones, y que son de interés para la aplicación. Se guarda el número de clases de cada iteración, el tiempo de ejecución así como datos de cada uno de las clases generadas: centroide, predicción del estado, medidas asociadas, candidato, validado, etc.

  - Dos archivos *.avi*, uno con la proyección en 2D de las medidas, las partículas y las clases; y otro con la secuencia en 3D.

  - Dos archivos *.fig* que pertenecen a dos gráficos, uno de ellos correspondiente al tiempo de ejecución empleado y otro del número de clases creadas por cada iteración. Cuando se termina la ejecución del algoritmo se crean dichas figuras y se tiene la posibilidad de guardarlas o modificarlas para emplearlas en el posterior estudio.

  - En todos los estimadores excepto en el KFPDA se generan si se desea multitud de archivos *.fig* sobre distintos aspectos de los mismos como son densidades de probabilidad, predicción de los estados, velocidades, etc.

Para ejecutar los distintos algoritmos son necesarias una serie de funciones que se detallan a continuación:

- KFPDA_Total

- *KFPDA_Total.m*: Función global desde la que se declaran las variables, se realiza la llamada al KFPDA, se guarda la información en cada iteración, se reinicia el filtro, se plotea la información, etc.

- *kfpda.m*: En ella se realiza la llamada a la función *initiate.m, pdaf.m y kman.m.*

- *constants.m*: Función en la que se declaran todas las constantes necesarias en la totalidad del algoritmo, como son los parámetros del filtro de Kalman, parámetros de las imágenes, de cámara, de rotación, etc.

- *initiate.m*: Función de reinicialización que se ejecuta en la primera iteración o cada vez que se reinicia el filtro.

- *pdaf.m*: Función que contiene el método probabilístico de asociación de datos.

- *kman.m*: Función con los cálculos propios del filtro de Kalman.

- *circle.m*: Dibuja círculos que son usados en la proyección en 2D para representar los filtros validados.

- XPFCP_Total

- *XPFCP_Total.m*: Función global desde la que se declaran las variables, se realiza la llamada a *XPFCPSinBuff.m*, se guarda la información en cada iteración, se reinicia el filtro, se plotea la información, etc.

- *XPFCPSinBuff*: Desde esta función se llama a la función de clasificación, se distribuyen las medidas y partículas, se llama a las funciones necesarias para llevar a cabo las distintas fases del XPFCP, se realizan algunos gráficos, etc.

- *clusterKMeans3SinBuff.m*: Contiene el proceso completo de asociación, en este caso el empleado es el K-Medias aunque también se cuenta con la función perteneciente al método Substract.

- *MIffun*: Función del modelo del proceso en las que se suma el ruido correspondiente al mismo.

- *MIhfun*: Función del modelo de las medidas en la que se suma el ruido correspondiente al mismo.

- *MIresidualR.m*: Función necesaria en el paso de selección del XPFCP. El método de selección empleado es el residual.

- *randsample.m*: Función que se emplea para seleccionar muestras aleatorias de las medidas sin reemplazamiento.

- *calculaDist.m*: Función empleada por el K-Medias para calcular la distancia de los datos a los centroides.

- *circle.m*: Dibuja círculos que son usados en la proyección 2D para representar las clases validados.

-**SJPDAF y SJPDAF+NN**:

- *SJPDAF_Total.m/SJPDAF_NN_Total.m*: Función global desde la que se declaran las variables, se realiza la llamada a SJPDAF/SJPDAF_NN, se guarda la información en cada iteración, se reinicia el proceso, se plotea la información, etc.

- *SJPDAF.m/SJPDAF_NN.m*: Desde esta función se llama a la función de clasificación y de asociación, se distribuyen las medidas y partículas, se llama a las funciones necesarias para llevar a cabo las distintas fases del proceso, se realizan algunos gráficos, etc.

- *PDA2.m/PDA3.m*: Funciones en las que se realiza la asociación probabilística de datos en cada algoritmo.

El resto de funciones de estos estimadores están ya explicadas en los algoritmos anteriores, por lo que simplemente son enumeradas a continuación:

- *clusterKMeans3SinBuff.m*

- *initiate.m*

- *MIffun.m*

- *MIhfun.m*

- *MIresidualR.m*

- *randsample.m*

- *calculaDis.mt*

- *circle.m*

- *constants.m*

Todas estas funciones se muestran en el capítulo de Planos. Las funciones *SJPDAF_Total.m* y *SJPDAF_NN_Total.m* son prácticamente iguales en todo; la única diferencia es que en la primera de ellas se llama a *SJPDAF.m* y en la otra a *SJPDAF_NN.m* Por lo tanto se mostrará únicamente una de ellas.

Además de estas funciones se ha empleado el fichero *ploteaTray.m* con el fin de representar gráficamente las trayectorias de los objetos mediante los distintos algoritmos, para lo cual debe estar cargado el fichero *.mat* del experimento del que queramos obtener las trayectorias. Esta función se expone en el capítulo de Planos.

Debido a que se han realizado muchas pruebas sobre los distintos parámetros a ajustar y los diferentes algoritmos se ha establecido una nomenclatura específica a la hora de nombrar a los archivos:

- En primer lugar se especifica qué video o qué fragmento de él se ha empleado para la realización de la prueba:
  - Vídeo 41: se inicia con ‘041’.
  - Vídeo 10 completo: se inicia con ‘010’.
  - Primer fragmento del vídeo 10, prueba corta: se inicia con ‘010a’.
  - Segundo fragmento del vídeo, prueba larga: se inicia con ‘010b’.
- En segundo lugar se nombra con una abreviatura de la prueba realizada.
- Por último se escribe el valor que se ha asignado al parámetro a ajustar en la prueba.

Por ejemplo si se ha finalizado el ajuste del parámetro *canM* con un valor de 2 en la prueba corta se obtienen los siguientes ficheros:

- “2Dproyection_010a_canm2”

- “video_010a_canm2”

- “result_010a_canm2”

- “Figura3_010a_canm2”

- “Figura4_010a_canm2”

De esta forma se consigue que en todo momento se tengan localizados los archivos y se sepa sin ninguna dificultad a qué prueba concreta pertenecen.

IV. PLIEGO DE CONDICIONES

Los aspectos técnicos de los equipos utilizados y los programas requeridos para la realización del proyecto se detallan a continuación.

1. Equipos físicos

- Ordenador portátil Packard Bell

Microprocesador	Intel - Centrino DUO
Velocidad de reloj	1.83 GHz
Memoria RAM	1 Gbyte
Disco duro	80 GBytes
Alimentación	GP280AFH 9.6V 2800mA/h

- Impresora HP Laserjet 2100

Modo de impresión	Láser
Velocidad de páginas	16ppm
Buffer de entrada	4Mbytes
Comunicaciones	Serie y paralelo
Resolución	600ppp

- Impresora HP Laserjet 4500 C

Modo de impresión	Láser
Velocidad de páginas	8ppm
Buffer de entrada	4Mytes
Comunicaciones	Serie y paralelo
Resolución	600ppp (color)

2. Equipos lógicos

- Sistema operativo Windows XP.
- MATLAB (versión 6.5).
- Procesador de textos (Microsoft Word).
- Hoja de cálculo (Microsoft Excel).

V. PLANOS

A continuación se muestran todas las funciones utilizadas por los distintos algoritmos.

Varios de estos estimadores requieren las mismas funciones, aunque éstas sólo aparecen en una ocasión.

"KFPDA_Total.m"

```
function []=KFPDA_Total(file,videoOn,plotOn,saveOn)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TRACKING OF MULTIPLE OBJECTS WITH KALMAN FILTER AND PDA PROCESS ASSOCIATION
% Author  : Maria Cabello Aguilar
% Date    : March 2007
% Sintaxis: []=KFPDA_Total(file,videoOn,plotOn,saveOn);
% Inputs  : - file: The path of the data file to segment
%           - plotOn: If plotting execution info is wanted to visualize set to '1'
%           - videoOn: If plotting in the image is wanted set to '1'
%           - saveOn: If wanted to store the results in a matlab file set to '1'
% Outputs :  All info generated is stored in a file named "result.mat"
% CAREFUL: WE ARE ALWAYS WORKIKG ONLY WITH 'X' AND 'Z'. 'Y' IS ONLY USED TO PLOT IN THE
%          IMAGE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

warning off MATLAB:colon:operandsNotRealScalar;
warning off MATLAB:divideByZero;

% ==========================================================================================
% INITIALIZING SYSTEM PARAMETERS
% ==========================================================================================

% Initializing System Params. - SistemaKF: Stucture with system information
%       * y: Array of measurements [x,z,y]
```

```
%       * Params: Params of the system
%           - nX: Number of states of the estimated obstacle [x,z]
%           - nY: Number of elements of the observation vector [x,z,y]
%           - dtQ: Standard deviation (NO Variance) of the Gaussian process noise
%           - dtR: Standard deviation (NO Variance) of the Gaussian measurement noise
SistemaKF.Params=[2 3 sqrt(10) sqrt(0.001)];
distMKF=800;

% ================================================================================
% VIDEO CONSTANTS
% ================================================================================

%Call function to define constants
constants(videoOn);

global FILE_SIZE FXL FYL U0L V0L TRAS_Y ROTATION_SCC;

% ================================================================================
% VARIABLES
% ================================================================================

% Define global variables
global INTINF T;
T=1/15;
INTINF=32000000;

% Define others variables
iterKF=0;
ksKF=[0 0];
kKFTotal=[];
xTotalKF=[];
xKF=[];
tTotal=[];
tTotalPlot=[];
tMedia=[];
kValid=[];
kMedia=[];

% ================================================================================
% INITIALIZING PARAMETERS-STRUCT
% ================================================================================

% Initializing Parameters Array - Filtro: Filter's parameters
%       - iterKF: the state variable of the filter state machine
%       - nN: Number total of particles (not to use in KFPDA)
%        - nM: Number of samples inserted directly from observation in the pdf reinit %
   %             (not to use in KFPDA)
%       - nB: (>=.nP) Number of measures can be in the buffer (not to use in KFPDA)
%       - rS: possible choices: residual (1),systematic sampling (2) and multinomial (3)
   %             NORMAL 1 (not to use in KFPDA)
%       - distMKF: Different uses

iterKF=0;
ParamsKF=[iterKF 0 0 0 0 distMKF^2];

% ================================================================================
% INITIALIZING CLASES-STRUCT
% ================================================================================

% Initializing Clases struct
%       * out: Params of the measurements clustering
%           - nXY: Lenght of the data vector to associate, can be nX or nY, or a
%             mixture
```

```
%           - iterM: Not used
%           - canM: Number of iterKF that filter is invalid/valid before valid/invalid
%           - hist: Not used
%           - factOlv: Not used
%           - kLikM: Not used
%           - distMKF: Value to plot filters, validate/invalidate...
%           - um: Not used
%           - distRM: Not used
%           - umOutliers: Not used
%       * type: Not used

ClasesKF.in=[];
ClasesKF.out=[SistemaKF.Params(2) 0 2 0 0 0 distMKF^2 0 0 0];
ClasesKF.type=[0 0];

% Filtro: K-1 array of structures of info about the objects tracked by the KFPDA
%       * U: 1-nX array with the velocities values updated in coordinates x and z
%       * I: number to identify a filter and to do association over time
%       * xc: 1-nX array with the correcrted state variables (the position in xz plane)
%       * xp: 1-nX array with the predicted state variables (the position in xz plane)
%       * P_corrected: covariance error matrix. Corrected value
%       * K: K matrix of the Kalman Filter
%       * Beta: number in %1 presenting the measures association process probability
%       * resid: combined innovation
%       * candidate: number presenting if the filter is a candidate
%       * validated: to '1' if the filter is validated, to '0' otherwise
%       * M: nM-1 meas associated to this filter
%       * nM: number informing about the number of meas associated to this filter

Filtro=[];
UKFOut=[]; IKFOut=[]; BetaKFOut=[]; canKFOut=[]; valKFOut=[];
nmKFOut=[]; PCorrectedOut=[]; residuoKFOut=[];
kKFOut=0;


% ================================================================================
% TAKING INFORMATION FROM FILES
% ================================================================================

% Data open
str1=['..\data' file '.dat'];
pfdata=fopen(str1,'rb');
nMeas=fread(pfdata,1,'int32');

% Video Open
if videoOn
    str=['..\left' file '.str'];;
    pfVideo=fopen(str,'rb');
    I=fread(pfVideo,FILE_SIZE);
    square=zeros(3,6);
    square(2,:)=TRAS_Y-[200 200 1700 1700 200 1500];
end
iterVideo=1;
iter=1;

% ================================================================================
% WHILE THERE ARE DATAS
% ================================================================================

% FlagVideo is used to remove the first iters
flagVideo=1;
while isempty(nMeas)==0
```

```
%=========================================================================================
% TO REMOVE ANY FRAMES AT THE BEGINNING
%=========================================================================================

if iterVideo<=750 & flagVideo
    y=fread(pfdata,[3 nMeas],'float32');
    iterVideo=iterVideo+1;
    iter=iter+1;
    nMeas=fread(pfdata,1,'int32');
    if videoOn
        I=fread(pfVideo,FILE_SIZE);
    end
    continue
elseif iterVideo>750 & flagVideo
    flagVideo=0;
    iter=1;
end

y=fread(pfdata,[3 nMeas],'float32');
SistemaKF.y=y';

%============================================================
% VISUALIZATION OF DATAS AT XZ PLANE AND THE VIDEO
%============================================================

% Plotting in XZ projection and 2D
figure(1); clf;
plot(y(1,:),y(2,:),'g.');
axis([-5000 5000 0 18000]);
xlabel('x in mm'); ylabel('z in mm');
title('Representacion 2D');
grid; hold on;

% Plotting image in the film
if videoOn

    % Show the film
    figure(2); clf;
    imshow(uint8((I')));
    title('Seguimiento de objetos en el video');
    hold on;

    % Show the measurements coordinates
    y3=TRAS_Y-y(3,:);
    y(3,:)=y(2,:);
    y(2,:)=y3;

    % Transform the measures
    SCCmatr=ROTATION_SCC*y;
    U=round(FXL*(SCCmatr(1,:)./SCCmatr(3,:)) + U0L);
    V=round(FYL*(SCCmatr(2,:)./SCCmatr(3,:)) + V0L);

    % Plot the measures
    plot(U,V,'g.');
end

%============================================================
%CALL KFPDA
%============================================================

if kKFOut==0
    iterKF=-INTINF;
end
```

```
if iterKF==-INTINF & nMeas
    iterKF=0;
end
ParamsKF(1)=iterKF;
tic
if iterKF~=-INTINF
    [xKF,Filtro,ksKF]=kfpda(SistemaKF,ParamsKF,ClasesKF,xKF,Filtro,ksKF);
end
kKFOut=ksKF(1);
tTotal=[tTotal;toc];

%=============================================================
% STORING RESULTS
%=============================================================

kKFTotal=[kKFTotal;ksKF];
if isempty(xKF)==0
    xTotalKF=[xTotalKF;xKF];
end

%=============================================================
% PLOTTINGS
%=============================================================

for i=1:kKFOut
    % Save the filter in independent arrays
    IKFOut=[IKFOut;Filtro(i).I];
    UKFOut=[UKFOut;Filtro(i).U];
    BetaKFOut=[BetaKFOut;Filtro(i).Beta];
    canKFOut=[canKFOut;Filtro(i).candidate];
    valKFOut=[valKFOut;Filtro(i).validated];
    nmKFOut=[nmKFOut;Filtro(i).nM];
    residuoKFOut=[residuoKFOut;Filtro(i).resid];
    PCorrectedOut=[PCorrectedOut; Filtro(i).P_corrected];

    % Starting to paint in this frame
    if videoOn
        square(1,1:5)=[Filtro(i).xc(1,1)-distMKF/2 Filtro(i).xc(1,1)+distMKF/2 ...
                (Filtro(i).xc(1,1)+distMKF/2) Filtro(i).xc(1,1)-distMKF/2 ...
                Filtro(i).xc(1,1)-distMKF/2];
        square(3,1:5)=Filtro(i).xc(1,2)*ones(1,5);
        square(:,6)=[Filtro(i).xc(1,1);TRAS_Y-1500;Filtro(i).xc(1,2)];
        SCCmatr=ROTATION_SCC*square;
        U=round(FXL*(SCCmatr(1,:)./SCCmatr(3,:))+U0L);
        V=round(FYL*(SCCmatr(2,:)./SCCmatr(3,:))+V0L);
    end

    % Painting the circles
    figure(1);
    if Filtro(i).validated
        circle(Filtro(i).xc(1,1),Filtro(i).xc(1,2),distMKF,20,'b');
        if videoOn
            figure(2);
            plot(U(1:5),V(1:5),'b');
        end
    else
        plot(Filtro(i).xc(1,1),Filtro(i).xc(1,2),'b+');
        if videoOn
            figure(2);
            plot(U(6),V(6),'b+');
        end
    end
end
```

```
    %============================================================
    % WRITTING INFORMATION IN THE IMAGE
    %============================================================

    str=sprintf('kKF=%d kValKF=%d',ksKF(1),ksKF(2));
    str2=sprintf('tExecKF=%d' ,round(tTotal(end,1)*1000));
    str3=sprintf('iter=%d',iterKF);
    str4=sprintf('iterVideo=%d',iterVideo);
    figure(1);
    text(-4700,500,str);
    text(-4700,17000,str2);
    text(2000,17000,str3);
    text(2000,16000,str4);

    if videoOn
        figure(2);
        text(20,230,str);
        text(20,10,str2);
        text(230,10,str3);
        text(230,20,str4);
    end

    if saveOn
        figure(1);
        proy(iter)=getframe;
        if videoOn
            figure(2);
            video(iter)=getframe;
        end
    end

    %Update iter
    if (iterKF==0 & kKFOut) | iterKF>0
        iterKF=iterKF+1;
    end
    iterVideo=iterVideo+1;
    iter=iter+1;

   pause(0.01);
   nMeas=fread(pfdata,1,'int32');
   if videoOn
       I=fread(pfVideo,FILE_SIZE);
   end
end

% ================================================================================================
% PLOTTING EXECUTION TIMES
% ================================================================================================

kValid=[kKFTotal(:,2)'];
tTotalPlot=tTotal';

% Calculating median
aux=size(kValid,2);
for i=1:(1/T):aux
    j=i+(1/T);
    if j>aux
        j=aux;
    end
    tMedia=[tMedia mean(tTotalPlot(1,i:j),2)];
    kMedia=[kMedia mean(kValid(1,i:j),2)];
end
```

```
% Plotting results and kKF
figure(3);

subplot(2,1,1);
plot(1:aux,kKFTotal(:,2)','b+');
grid; axis([0 aux -2 max(max(kValid))+2]);
title('Number of valid KFPDA filters in blue')
xlabel('iter,frame'); ylabel('kValid');
subplot(2,1,2);
stairs(1:(1/T):aux,kMedia(1,:),'b');
grid; axis([0 aux -2 max(max(kValid))+2]);
title('Median in a second of the Number of valid KFPDA filters in blue')
xlabel('iter,frame'); ylabel('kValid');
hold off;

% Plotting time
figure(4);

subplot(2,1,1);
plot(1:aux,tTotalPlot(1,:),'b');
grid; axis([0 aux min(min(tTotalPlot)) max(max(tTotalPlot))]);
title('Execution time of each-loop of the KFPDA algorithm in blue')
xlabel('iter,frame'); ylabel('sec');

subplot(2,1,2);
stairs(1:(1/T):aux,tMedia(1,:),'b');
grid;
axis([0 aux min(min(tMedia)) max(max(tMedia))]);
hold on;
title('Mean in a second of each-loop Execution time of the KFPDA algorithm in blue')
xlabel('iter,frame'); ylabel('sec');
hold off;

fclose(pfdata);
if videoOn
    fclose(pfVideo);
end

% Saving videos
if saveOn
    movie2avi(proy,'2Dproyection');
    if videoOn
        movie2avi(video,'video');
    end
end

% ======================================================================================
% ENDING THE KFPDA EXECUTION
% ======================================================================================

% Saving results
clear kKFOut;
save result *Total *Out;
```

“Constants.m”

```
function []=constants(videoOn)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% DEFINING CONSTANTS: KALMAN MATRIXES, COORDINATE TRANFORMATION
% PARAMETERS, IMAGE PARAMETERS, AXIS COORDINATES
% Sintaxis: []=constants(videoOn);
% Inputs  : - videoOn: If plotting in the image is wanted set to 1 one
% Outputs :
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Kalman parameters
global A B H;
A=eye(2);
B=eye(2);
H=eye(2);

%Image parameters
global WIDTH HIGHT FILE_SIZE;
if videoOn
    WIDTH=320;
    HIGHT=240;
    FILE_SIZE=[WIDTH HIGHT];

    %Inrinsic parameters, left camera
    global FXL FYL U0L V0L;
    FXL=430.79014;
    FYL=431.72027;
    U0L=151.26555;
    V0L=117.03242;

    %Camera rotation angles
    alpha=0.94972*3.14159/180;
    beta=0.019508;
    phi=-0.014053;

    %Translation of Y-axis
    global TRAS_Y;
    TRAS_Y=970;

    %Rotation matrix
    global ROTATION_SCC;
    ROTATION_SCC=...
            [cos(beta)*cos(phi) -cos(beta)*sin(phi) sin(beta);...
            sin(alpha)*sin(beta)*cos(phi)+cos(alpha)*sin(phi)...
            sin(alpha)*sin(beta)*sin(phi)+cos(alpha)*cos(phi)...
            -sin(alpha)*cos(beta);...
            -cos(alpha)*sin(beta)*cos(phi)+sin(alpha)*sin(phi)...
            cos(alpha)*sin(beta)*sin(phi)+sin(alpha)*cos(phi)...
            cos(alpha)*cos(beta)];
end

%Axis coordinates
global X_MIN X_MAX Z_MIN Z_MAX Y_MIN Y_MAX;
X_MIN=-8000;
X_MAX=8000;
Z_MIN=500;
Z_MAX=20500;
Y_MIN=100;
Y_MAX=2100;
```

“KFPDA.m”

```
function [xKF,Filtro,ksKF]=kfpda(SistemaKF,ParamsKF,ClasesKF,xKF,Filtro,ksKF);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FUNCTION WHICH CALL KALMAN AND PDA FUNCTIONS
% Sintaxis: [xKF,Filtro,ksKF]=kfpda(SistemaKF,ParamsKF,ClasesKF,xKF,Filtro,ksKF);
% Inputs  : - SistemaKF: structure with system information
%           - ParamsKF: filter's parameters
%           - ClasesKF: more filter's parameters
%           - xKF: array with de corrected states
%           - Filtro: array with all the created filters
%           - ksKF: array with the quantity of filters [total, validated]
% Outputs : - xKF: array with de corrected states
%           - Filtro: array with all the created filters
%           - ksKF: array with the quantity of filters [total, validated]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global FILE_SIZE;

data=SistemaKF.y(:,1:2);
iter=ParamsKF(1);

if iter==0
    [ksKF,xKF,Filtro]=initiate(SistemaKF,ksKF,xKF,ParamsKF,ClasesKF);
else

    % ==========================================
    % PDAF FUNCTION
    % ==========================================

    [ksKF,Filtro]=pda(Filtro,data,ksKF,ClasesKF);

    % ==========================================
    % KALMAN FUNCTION
    % ==========================================

    if ksKF(1)~=0
        [Filtro,xKF]=kman(Filtro,ksKF,SistemaKF,xKF);

    % If the scene is empty then clear and next iter find objects in scene
    else
        Filtro=[];
    end
end
```

"Inititate.m"

```
function [ksKF,xKF,Filtro]=initiate(SistemaKF,ksKF,xKF,ParamsKF,ClasesKF)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INITIATE TRACKS AND PREDICT STATE
% Sintaxis: initiate(SistemaKF,ksKF,xKF,ParamsKF,ClasesKF)
% Inputs  : - SistemaKF: structure with system information
%           - ksKF: array with the quantity of filters [total, validated]
%           - xKF: array with de corrected states
%           - ParamsKF: filter's paramemeters
%           - ClasesKF: more parameters of the filter
% Outputs : - ksKF: array with the quantity of filters [total, validated]
%           - xKF: array with de corrected states
%           - Filtro: array with all the created filters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
global A B Q INTINF;
ksKF(1)=ksKF(1)+1;
max=30;
data=SistemaKF.y(:,1:2);
distM=ParamsKF(6);
canM=ClasesKF.out(3);
listaId=zeros(1,max);

%=========================================================================
% INITIALIZING LIST OF FILTERS IDENTIFIERS
% =========================================================================

%Save the measure's position as object center
Filtro(1).C=[data(1,1) 1500 data(1,2)];
Filtro(1).U=[0 0];
Filtro(1).I=1;
Filtro(1).nM=0;
Filtro(1).M=data(1,:);
Filtro(1).candidate=-INTINF;
Filtro(1).validated=[];
Filtro(1).resid=[0 0];
Filtro(1).Beta=1;
listaId(1)=1;

for m=2:ksKF(1)
    Filtro(m).M=[];
    Filtro(m).nM=0;
end

%=========================================================================
%   ASSIGNMENT
%=========================================================================

for m=[1:size(data,1)]
   %Calculate the distance from each measure to each object
   for i=1:ksKF(1)
       distance(:,i)=sum((data-repmat(Filtro(i).C(1,1:2:3),size(data,1),1)).^2,2);
   end

   %Find the shortest distance
   minimum=min(distance(m,:));
   %Find the column that contains the shortest distance
   column=find(distance(m,:)==minimum);

   % ============================================
   % ADD MEASURE TO EXISTING OBJECT
   % ============================================

   %If distance is close to an existing object then associate
   if distance(m,column)<distM
       Filtro(column).M=[Filtro(column).M; data(m,:)];
       Filtro(column).C(1,1:2:3)=mean(Filtro(column).M,1);
       Filtro(column).nM=Filtro(column).nM+1;
       Filtro(column).Beta=[Filtro(column).Beta; exp(-0.5*(distance(m,column))/distM)];

   % ============================================
   % DEFINE MEASURE AS A NEW OBJECT
   % ============================================

    else
        ksKF(1)=ksKF(1)+1;
        %Saves the measure's position as object center
        Filtro(ksKF(1)).candidate=-INTINF;
```

```
        Filtro(ksKF(1)).C=[data(m,1) 1500 data(m,2)];
        Filtro(ksKF(1)).M=data(m,:);
        Filtro(ksKF(1)).nM=0;
        Filtro(ksKF(1)).Beta=1;
        Filtro(ksKF(1)).resid=[0 0];
        j=1;
        while (listaId(j)~=0)
            j=j+1;
        end
        for i=1:max
            k=0;
            for q=1:j-1
                if listaId(q)~=i
                    k=k+1;
                end
            end
            if k==(j-1)
                Filtro(ksKF(1)).I=i;
                listaId(j)=i;
                break;
            end
        end
    end
end

% ================================================
% INITIALIZATION
% ================================================

for m=1:ksKF(1)
    Filtro(m).validated=false;
    Filtro(m).xp=Filtro(m).C(1,1:2:3);
    Filtro(m).xc=Filtro(m).C(1,1:2:3);
    Filtro(m).P_predicted=1*eye(2);
    Filtro(m).P_corrected=1*eye(2);
    Filtro(m).U=zeros(1,2);
    xKF=[xKF;Filtro(m).xc];
end

changedKF=[];
changedKFT=[];

for m=1:ksKF(1)
    if Filtro(m).candidate==-INTINF
        if Filtro(m).nM==0
            changedKF=[changedKF Filtro(m).I];
            listaId(Filtro(m).I)=0;
        else
            if canM==0
                Filtro(m).validated=true;
                Filtro(m).candidate=1;
            else
                Filtro(m).candidate=-canM;
            end

        end
    else
        if Filtro(m).nM==0
            Filtro(m).Beta=0;
            Filtro(m).resid=[0 0];
            if Filtro(m).candidate>0
                Filtro(m).candidate=-canM;
            else
```

```
                Filtro(m).candidate=Filtro(m).candidate-1;
            end
            if Filtro(m).candidate<=-2*canM
                changedKF=[changedKF Filtro(m).I];
                listaId(Filtro(m).I)=0;
                if Filtro(m).validated
                    Filtro(m).validated=false;
                end
            end
        else
            Filtro(m).C(1,1:2:3)=mean(Filtro(m).M,1);
            Filtro(m).candidate=Filtro(m).candidate+1;
            if Filtro(m).candidate>=0
                Filtro(m).candidate=1;
                Filtro(m).validated=true;
            end

        end
    end

end
ksKF(2)=0;
for m=1:ksKF(1)
    if Filtro(m).nM
        Filtro(m).Beta=Filtro(m).Beta/(sum(Filtro(m).Beta(:,1)));
    end
    if Filtro(m).validated
        ksKF(2)=ksKF(2)+1;
    end
end
for i=1:length(changedKF)
    for m=1:ksKF(1)
        if Filtro(m).I==changedKF(i)
            changedKFT=[changedKFT m];
        end
    end
end
Filtro(changedKFT)=[];
ksKF(1)=ksKF(1)-length(changedKF);
```

“PDA.m”

```
function [ksKF,Filtro]=pda(Filtro,data,ksKF,ClasesKF);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PDA ASSOCIATION PROCESS
% Sintaxis: [ksKF,Filtro]=pdaf(Filtro,data,ksKF,ClasesKF)
% Inputs  : - Filtro: array with all the created filters
%           - data: input data array. X and Z coordinates
%           - ksKF: array with the quantity of filters [total, validated]
%           - ClasesKF: more filter's parameters
% Outputs : - ksKF: array with the quantity of filters [total, validated]
%           - Filtro: array with all the created filters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global INTINF

distM=ClasesKF.out(7);
canM=ClasesKF.out(3);
listaId=zeros(1,30);
```

```
for m=1:ksKF(1)
    %Create identifiers list
    listaId(Filtro(m).I)=Filtro(m).I;
    Filtro(m).nM=0;
    Filtro(m).M=[];
    Filtro(m).Beta=[];
    Filtro(m).resid=[0 0];
    new(m)=false;
end

%==========================================================================
%   CALCULATION OF DISTANCE
%==========================================================================

if isempty(data)==0
    for m=1:ksKF(1)
        distance(:,m)=sum((data-repmat(Filtro(m).xp(1,:),size(data,1),1)).^2,2);
    end

    %======================================================================
    %   ASSIGNMENT
    %======================================================================

    for m=[1:size(data,1)]

        %Finds the shortest distance
        minimum=min(distance(m,:));
        %Finds the column that contains the shortest distance
        column=find(distance(m,:)==minimum);

        % ================================================
        % ADD MEASURE TO EXISTING OBJECT
        % ================================================

        %If distance is close to an existing object then associate
        if distance(m,column)<distM
            Filtro(column).M=[Filtro(column).M; data(m,:)];
            Filtro(column).nM=Filtro(column).nM+1;
            Filtro(column).Beta=…
                …[Filtro(column).Beta;exp(0.5*(distance(m,column))/distM)];
            %If objects was created in this iteration calc new distance
            if new(column)==true
                new(column)=false;
                distance(:,column)=…
                      …sum((data-repmat(Filtro(column).xp(1,:),size(data,1),1)).^2,2);
            end

        %================================================
        % DEFINE MEASURE AS A NEW OBJECT
        %================================================

        else
            ksKF(1)=ksKF(1)+1;
            new(ksKF(1))=true;
            Filtro(ksKF(1)).candidate=-INTINF;
            Filtro(ksKF(1)).validated=false;
            Filtro(ksKF(1)).C=[data(m,1) 1500 data(m,2)];
            Filtro(ksKF(1)).xp=data(m,:);
            Filtro(ksKF(1)).xc=data(m,:);
            Filtro(ksKF(1)).P_predicted=1*eye(2);
            Filtro(ksKF(1)).nM=0;
            Filtro(ksKF(1)).M=[];
```

```
            Filtro(ksKF(1)).M=data(m,:);
            Filtro(ksKF(1)).Beta=1;
            Filtro(ksKF(1)).resid=[0 0];
            distance(:,ksKF(1))=sum((data-repmat(data(m,:),size(data,1),1)).^2,2);
            j=1;
            while (listaId(j)~=0)
                j=j+1;
            end
            for i=1:length(listaId);
                k=0;
                for q=1:j-1
                    if listaId(q)~=i
                        k=k+1;
                    end
                end

                if k==(j-1)
                    Filtro(ksKF(1)).I=i;
                    listaId(j)=i;
                    break;
                end
            end

        end
    end

end
changedKF=[];
changedKFT=[];

%================================================
% VALIDATION PROCESS
%================================================

for m=1:ksKF(1)
    if Filtro(m).candidate==-INTINF
        if Filtro(m).nM==0
            changedKF=[changedKF Filtro(m).I];
            listaId(Filtro(m).I)=0;
        else
            if canM==0
                Filtro(m).validated=true;
                Filtro(m).candidate=1;
            else
                Filtro(m).candidate=-canM;
            end

        end
    else
        if Filtro(m).nM==0
            Filtro(m).Beta=0;
            Filtro(m).resid=[0 0];
            if Filtro(m).candidate>0
                Filtro(m).candidate=-canM;
            else
                Filtro(m).candidate=Filtro(m).candidate-1;
            end
            if Filtro(m).candidate<=-2*canM
                changedKF=[changedKF Filtro(m).I];
                listaId(Filtro(m).I)=0;
                if Filtro(m).validated
                    Filtro(m).validated=false;
                end
```

```
            end
        else
            Filtro(m).C(1,1:2:3)=mean(Filtro(m).M,1);
            Filtro(m).candidate=Filtro(m).candidate+1;
            if Filtro(m).candidate>=0
                Filtro(m).candidate=1;
                Filtro(m).validated=true;
            end

        end
    end

end
ksKF(2)=0;
for m=1:ksKF(1)
    %Probabilities normalitation
    if Filtro(m).nM
        Filtro(m).Beta=Filtro(m).Beta/(sum(Filtro(m).Beta(:,1)));
        %Calculate the innovation
        for j=1:size(Filtro(m).M,1)
            residuo(j,1)=Filtro(m).Beta(j,1)*(Filtro(m).M(j,1)-(Filtro(m).xp(1,1)));
            residuo(j,2)=Filtro(m).Beta(j,1)*(Filtro(m).M(j,2)-(Filtro(m).xp(1,2)));
        end
        Filtro(m).resid(1,1)=sum(residuo(:,1));
        Filtro(m).resid(1,2)=sum(residuo(:,2));
    end
    if Filtro(m).validated
        ksKF(2)=ksKF(2)+1;
    end
end

%===============================================
% REMOVE INVALIDATED FILTERS
%===============================================

for i=1:length(changedKF)
    for m=1:ksKF(1)
        if Filtro(m).I==changedKF(i)
            changedKFT=[changedKFT m];
        end
    end
end
Filtro(changedKFT)=[];
ksKF(1)=ksKF(1)-length(changedKF);
```

“KMAN.m”

```
function [Filtro,xKF]=kman(Filtro,ksKF,SistemaKF,xKF);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% KALMAN FILTERING
% Sintaxis: [Filtro,xKF]=kman(Filtro,ksKF,SistemaKF,xKF);
% Inputs  : - Filtro: array with all the created filters
%           - ksKF: array with the quantity of filters [total, validated]
%           - SistemaKF: structure with system information
%           - xKF: array with the corrected states
% Outputs : - Filtro: array with all the created filters
%           - xKF: array with de corrected states
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
global A B H T;

%===========================================================================
%   CORRECTION STEP
%===========================================================================
xc_old=[];
for m=1:ksKF(1)
    xc_old(m,:)=Filtro(m).xc;
end

varR=SistemaKF.Params(4)^2;
varQ=SistemaKF.Params(3)^2;
R=varR*eye(2);
Q=varQ*eye(2);

for m=1:ksKF(1)
    Filtro(m).K=Filtro(m).P_predicted*H'/(H*(Filtro(m).P_predicted)*H'+R);
    Filtro(m).xc=Filtro(m).xp+(Filtro(m).K*(Filtro(m).resid)')';
    Filtro(m).P_corrected=(eye(2)-Filtro(m).K*H)*Filtro(m).P_predicted;
    xKF=[xKF;Filtro(m).xc];
end

%===========================================================================
%   UPDATE VELOCITIES
%===========================================================================
for m=1:ksKF(1)
    Filtro(m).U=[0 0];
end

for m=1:ksKF(1)
    Filtro(m).U=((Filtro(m).xc-xc_old(m,:))/T);
end

%===========================================================================
%   PREDICTION STEP
%===========================================================================

for m=1:ksKF(1)
    clear Filtro(m).P_predicted;
end

for m=1:ksKF(1)
    V=Filtro(m).U;
    if Filtro(m).I~=0
        Filtro(m).xp=[A*[Filtro(m).xc]'+(B*V'*T)]';
        Filtro(m).P_predicted=A*Filtro(m).P_corrected*A'+Q;
    end
end
```

"XPFCP_Total.m"

```
function []=XPFCP_Total(file,videoOn,plotOn,saveOn)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TRACKING OF MULTIPLE OBJECTS
% Author  : Maria Cabello Aguilar
% Date    : April 2007
% Tracking multiple objects in a scene using XPFCP filters.
% Sintaxis: XPFCP_Total(file,videoOn,plotOn,saveOn);
```

```
% Inputs  : - file: The path of the data file to segment
%           - plotOn: If plotting execution info is wanted to visualize set to '1'
%           - videoOn: If plotting in the image is wanted set to '1'
%           - saveOn: If wanted to store the results in a matlab file set to '1'
% Outputs :  All info generated is stored in a file named "resutlt.mat"
% CAREFUL: WE ARE ALWAYS WORKING ONLY WITH X AND Z. Y IS ONLY USED TO PLOT IN THE IMAGE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

warning off MATLAB:colon:operandsNotRealScalar;
warning off MATLAB:divideByZero;

% ================================================
% INMITIALIZING PARAMETERS STRUCT
% ================================================

% Initializing Sistema Params. - Sistema: Stucture with system information
%       * y: Array of measurements [x,z,y]
%       * Params: Parameters of the system
%           - nX: Number of states of the estimated obstacle [x,z,y,vz,vx]
%           - nY: Number of elements of the observation vector [x,z,y]
%           - cX: Standard deviation (NO Variance) of the Gaussian process noise
%           - cY: Standard deviation (NO Variance) of the Gaussian measurement noise
distMXPF=700;
SistemaXPF.Params=[5 3 400 100];

% ================================================
% VIDEO CONSTANTS
% ================================================

%CALL FUNCTION TO DEFINE CONSTANTS
constants(videoOn,SistemaKF);
global FILE_SIZE FXL FYL U0L V0L TRAS_Y ROTATION_SCC;

% ================================================
% VARIABLES
% ================================================

%Define global variables
global INTINF;
T=1/15;
INTINF=32000000;

%Define others variables
iterXPF=0;
kXPFIn=0;
kXPFOut=0;
ksXPF=[0 0 0 0];
kXPFTotal=[];
xTotalXPF=[];
xXPF=[];
nParticTotal=[];
nPartic=0;
tTotal=[];
tTotalPlot=[];
tMedia=[];
kValid=[];
kMedia=[];

% Initializing Params Array -
%       - iterXPF: the state variable of the filter state machine
%       - nN: Number total of particles
%       - nM: Number of samples inserted directly from observation in the pdf reinit
%       - nB: (>=.nP) Number of meas can be in the buffer
```

```
%       - rS: Possible choices: residual (1), systematic sampling (2),and multinomial
%             (3)
%       - distM: Different uses

iterXPF=0;
ParamsXPF=[iterXPF 100 70 200 1 distMXPF^2];

% Initializing Clases Array.- Clases: Structure with cluster information
%       * in: Params of the measurements clustering
%           - nXY: Lenght of the data vector to cluster, can be nX or nY, or a mixture
%           - iterM: Number maximum of iterations at the clustering algorithm
%           - canM: Number of iterXPF that cluster is invalid/valid before valid/invalid
%           - hist: Histheresys to valid/invalid due to cluster lik | dist to prediction
%                 centroide
%           - factOlv: Forgeting factor in the cluster likelihood
%           - kLikM: Normalizing factor to calculate cluster likelihood according to the
%                  number clusters
%           - distM: Different uses. Fixed here to 8*Sistema.cY; (10*) THINK AND SEE
%           - um: Threshold to validate the cluster according to its likelihood
%           - distRM: Not used
%           - umOutliers: Not used
%       * out: Idem than parameters 'in' but for the particles clustering
%       * type: For the input and output [typeIn,typeOut]. choices: kMeans (1),
%              Subtractive (2)

ClasesXPF.type=[1 1];
ClasesXPF.in=[SistemaXPF.Params(2) 100 2 0.5 0.8 10 distMXPF^2 0.6 (1.5*distMXPF)^2…
             …0.1];
if  ClasesXPF.type(2)==1
    ClasesXPF.out=[SistemaXPF.Params(1) 100 1 0.5 0.8 10 distMXPF^2 0.6…
                 …(1.5*distMXPF)^2 0.1];
else
    ClasesXPF.out=[SistemaXPF.Params(1) 100 2 0.5 0.8 10 550^2 0.6 (1.25*550)^2 0.1];
end

% cluster: K-1 array of structures of info about the classes
%       * I: Number to identify a cluster and to do clustering association over time
%       * C: 1-nY/1-nX array with the state variables (the position in xz plane) of the
%            centroid
%       * P: number in %1 presenting the cluster probability
%       * candidate: Number presenting if the cluster is a candidate
%       * validated: To '1' if the cluster is validated, to '0' otherwise
%       * M: nM-1 cluster M
%       * nM: Number informing about the number cluster M

clusterIn=[];
clusterOut=[];
IXPFOut=[]; CXPFOut=[]; PXPFOut=[]; canXPFOut=[]; valXPFOut=[]; nmXPFOut=[];

% ========================================================================
% TAKING INFORMATION FROM FILES
% ========================================================================

% Data Open
str1=['..\data' file '.dat'];
pfdata=fopen(str1,'rb');
nMeas=fread(pfdata,1,'int32');

% Video Open
if videoOn
    str=['..\left' file '.str'];;
    pfVideo=fopen(str,'rb');
    I=fread(pfVideo,FILE_SIZE);
```

```
    square=zeros(3,6);
    square(2,:)=TRAS_Y-[200 200 1700 1700 200 1500];
end
iterVideo=1;
iter=1;

% ========================================================================
% WHILE THERE ARE DATAS
% ========================================================================

% FlagVideo is used to remove the first iters
flagVideo=1;
while isempty(nMeas)==0

    % ====================================================================
    % TO REMOVE ANY FRAMES AT THE BEGINNING
    % ====================================================================

    if iterVideo<=750& flagVideo
        y=fread(pfdata,[3 nMeas],'float32');
        iterVideo=iterVideo+1;
        iter=iter+1;
        nMeas=fread(pfdata,1,'int32');
        if videoOn
            I=fread(pfVideo,FILE_SIZE);
        end
        continue
    elseif iterVideo>750 & flagVideo
        flagVideo=0;
        iter=1;
    end

    y=fread(pfdata,[3 nMeas],'float32');
    SistemaXPF.y=y';

    % ====================================================================
    % VISUALIZATION OF DATAS AT XZ PLANE AND THE VIDEO
    % ====================================================================

    % Plotting in XZ projection and 2D
    figure(1); clf;
    plot(y(1,:),y(2,:),'g.');
    axis([-5000 5000 0 18000]);
    xlabel('x in mm'); ylabel('z in mm');
    title('Representacion 2D');
    grid; hold on;

    % Plotting image in the Film
    if videoOn

        % Show the film
        figure(2); clf;
        imshow(uint8((I')));
        title('Tracking objects 3D');
        hold on;

        % Show the measurements coordinates
        y3=TRAS_Y-y(3,:);
        y(3,:)=y(2,:);
        y(2,:)=y3;

        % Transform the measures
        SCCmatr=ROTATION_SCC*y;
```

```
        U=round(FXL*(SCCmatr(1,:)./SCCmatr(3,:)) + U0L);
        V=round(FYL*(SCCmatr(2,:)./SCCmatr(3,:)) + V0L);

        % Plotting the measures
        plot(U,V,'g.');
    end

    % ========================================================================
    % CALL XPFCP
    % ========================================================================

    if kXPFIn==0 & kXPFOut==0
        iterXPF=-INTINF;
    end
    if iterXPF==-INTINF & nMeas
        iterXPF=0;
    end
    ParamsXPF(1)=iterXPF;
    tic
    if iterXPF~=-INTINF
        [xXPF,clusterIn,clusterOut,ksXPF]=...

   XPFCPsinBuff(SistemaXPF,ParamsXPF,ClasesXPF,xXPF,clusterIn,clusterOut,ksXPF,T,plotOn)
   ;
    end
    kXPFIn=ksXPF(1);
    kXPFOut=ksXPF(3);
    nPartic=size(xXPF,1);
    tTotal=[tTotal;toc];

    % ========================================================================
    % STORING RESULTS
    % ========================================================================


    kXPFTotal=[kXPFTotal;ksXPF];
    nParticTotal(end+1)=nPartic;
    if isempty(xXPF)==0
        xTotalXPF=[xTotalXPF;xXPF];
    end

    %=========================================================================
    % PLOTTINGS
    %=========================================================================

    % Plotting in XZ plane
    if nPartic
        figure(1);
        plot(xXPF(:,1),xXPF(:,2),'y.');

        % Plotting image in the film
        if videoOn

            % Show the particles coordinates
            xPlot=xXPF(:,1:3)';
            x3=TRAS_Y-xPlot(3,:);
            xPlot(3,:)=xPlot(2,:);
            xPlot(2,:)=x3;

            % Transform the particles
            SCCmatr=ROTATION_SCC*xPlot;
            U=round(FXL*(SCCmatr(1,:)./SCCmatr(3,:))+U0L);
            V=round(FYL*(SCCmatr(2,:)./SCCmatr(3,:))+V0L);
```

```
        % Plot the particles
        figure(2);
        plot(U,V,'y.');
    end
end

%=========================================================================
% PLOTTING CLUSTERS-OUT
%=========================================================================
for i=1:kXPFOut
    IXPFOut=[IXPFOut;clusterOut(i).I];
    CXPFOut=[CXPFOut;clusterOut(i).C];
    PXPFOut=[PXPFOut;clusterOut(i).P];
    canXPFOut=[canXPFOut;clusterOut(i).candidate];
    valXPFOut=[valXPFOut;clusterOut(i).validated];
    nmXPFOut=[nmXPFOut;clusterOut(i).nM];

    if videoOn
        square(1,1:5)=[clusterOut(i).C(1,1)-distMXPF/2 clusterOut(i).C(1,1)+…
                    …distMXPF/2 (clusterOut(i).C(1,1)+distMXPF/2)…
                    …clusterOut(i).C(1,1)-distMXPF/2…
                    …clusterOut(i).C(1,1)-distMXPF/2];
        square(3,1:5)=clusterOut(i).C(1,2)*ones(1,5);
        square(:,6)=[clusterOut(i).C(1,1);TRAS_Y-1500;clusterOut(i).C(1,2)];
        SCCmatr=ROTATION_SCC*square;
        U=round(FXL*(SCCmatr(1,:)./SCCmatr(3,:))+U0L);
        V=round(FYL*(SCCmatr(2,:)./SCCmatr(3,:))+V0L);
    end

    % Paintng the cirlces
    figure(1);
    if clusterOut(i).validated
        circle(clusterOut(i).C(1,1),clusterOut(i).C(1,2),distMXPF,20,'r');
        if videoOn
            figure(2);
            plot(U(1:5),V(1:5),'r');
        end
    else
        plot(clusterOut(i).C(1,1),clusterOut(i).C(1,2),'r+');
        if videoOn
            figure(2);
            plot(U(6),V(6),'r+');
        end
    end
end

%=========================================================================
% WRITTING INFORMATION IN THE IMAGE
%=========================================================================

str1=sprintf('KXPF=%d kValXPF=%d',ksXPF(3),ksXPF(4));
str2=sprintf('tExecXPF=%d' ,round(tTotal(end,1)*1000));
str3=sprintf('iter=%d',iter);
str4=sprintf('iterVideo=%d',iterVideo);

figure(1);
text(700,500,str1);
text(-4700,17000,str2);
text(2000,17000,str3);
text(2000,16000,str4);

if videoOn
```

```
        figure(2);
        text(200,230,str1)
        text(20,10,str2);
        text(230,10,str3);
        text(230,20,str4);
    end

    if saveOn
        figure(1);
        proy(iter)=getframe;
        if videoOn
            figure(2);
            video(iter)=getframe;
        end
    end

    %Update iterVideo
    if (iterXPF==0 & kXPFIn) | iterXPF>0
        iterXPF=iterXPF+1;
    end

    iterVideo=iterVideo+1;
    iter=iter+1;

   pause(0.01);
   nMeas=fread(pfdata,1,'int32');
   if videoOn
       I=fread(pfVideo,FILE_SIZE);
   end
end

% ========================================================================
% PLOTTING EXECUTION TIMES
% ========================================================================

kValid=[kXPFTotal(:,4)'];
tTotalPlot=tTotal';

% Calculating median
aux=size(kValid,2);
for i=1:(1/T):aux
    j=i+(1/T);
    if j>aux
        j=aux;
    end
    tMedia=[tMedia mean(tTotalPlot(:,i:j),2)];
    kMedia=[kMedia median(kValid(:,i:j),2)];
end

% To plot results and compare k
figure(3);

subplot(2,1,1);
plot(1:aux,kXPFTotal(:,4)','ro');
grid; axis([0 aux -2 max(max(kValid))+2]);
title('Number of XPFCP clusters in red')
xlabel('iter,frame'); ylabel('kValid');
subplot(2,1,2);
grid; axis([0 aux -2 max(max(kValid))+2]);
hold on;
stairs(1:(1/T):aux,kMedia(1,:),'r');
title('Median in a second of the Number of clusters in red')
xlabel('iter,frame'); ylabel('kValid');
```

```
hold off;

% Time plot
figure(4);

subplot(2,1,1);
plot(1:aux,tTotalPlot(1,:),'r');
grid; axis([0 aux min(min(tTotalPlot)) max(max(tTotalPlot))]);
title('Execution time of each-loop of XPFCP algorithm in red')
xlabel('iter,frame'); ylabel('sec');

subplot(2,1,2);
grid;
axis([0 aux min(min(tMedia)) max(max(tMedia))]);
hold on;
stairs(1:(1/T):aux,tMedia(1,:),'r');
title('Mean in a second of each-loop Execution time of the XPFCP algorithm in red')
xlabel('iter,frame'); ylabel('sec');
hold off;

fclose(pfdata);
if videoOn
    fclose(pfVideo);
end

% Saving videos
if saveOn
    movie2avi(proy,'2Dproyection');
    if videoOn
        movie2avi(video,'video');
    end
end

% ========================================================================
% ENDING THE XPFCP EXECUTION
% ========================================================================

%Saving results
clear clusterOut;
nParticTotal=nParticTotal';
save result *Total *Out;
```

"XPFCPSinBuff.m"

```
function [xPartic,clusterIn,clusterOut,ks]=...
        XPFCPsinBuff(Sistema,ParamsXPF,Clases,xPartic,clusterIn,clusterOut,ks,t,plotOn);

% XPFCP execution
% Author  : Marta Marron Romera
% Date    : 12-11-06 / 12-12-06
% Sintaxis:[xPartic,clusterIn,clusterOut,ks]=XPFCPsinBuff…
%          (Sistema,ParamsXPF,Clases,xPartic,clusterIn,clusterOut,ks,t,plotOn);
% Inputs  :
%   - Sistema: Stucture with system information
%       * y: Array of measurements [x,z,y]
%       * Params: Params of the system
%           - nX: Number of states of the estimated obstacle [x,z,y,vz,vx]
%           - nY: Number of elements of the observation vector [x,z,y]
%           - cX: Standard deviation (NO Variance) of the Gaussian process
```

```
%           - cY: Standard deviation (NO Variance) of the Gaussian measurement
%   - ParamsXPF: Params of the XPFCP
%       - iter: the state variable of the XPF state machine
%       - nN: Number total of particles
%       - nM: Number of samples inserted directly from observation in the pdf reinit
%       - nB: (>=.nP) Number of meas can be in the buffer. Not used
%       - rS: The possible choices are: systematic (2), residual (1),
%             and multinomial (3)
%       - distM: Different uses
%   - Clases: Structure with clustering information
%       * in: Params of the measurements clustering
%           - nXY: Lenght of the data vector to cluster, can be nX or nY, or a mixture
%           - iterM: Number maximum of iterations at the clustering algorithm
%           - canM: Number of frames that cluster is invalid/valid before valida/invalid
%           - hist: Histheresys to valid/invalid due to cluster lik | dist
%                   to pred centroide
%           - factOlv: Forgeting factor in the cluster likelihood
%           - kLikM: Normalizing factor to calc cluster lik according
%                    to the number clusters
%           - distM: Different uses
%           - um: Threshold to validate the cluster according to its likelihood
%           - distRM: Not used
%           - umOutliers: Not used
%       * out: Idem than parameters 'IN' but for the particles clustering
%       * type: For the input and output [typeIn,typeOut]. Possible:
%               kMeans (1), Subtractive (2)
%   - xPartic: nN-nX array with the particles value at the end of the XPF
%   - clusterIn: K-1 array of structures of info about the classes
%       * I: Number to identify a cluster and to do clustering association over time
%       * C: 1-nY/1-nX array with the state variables (the position in xz plane)
%            of the centroid
%       * P: Number in %1 presenting the cluster probability
%       * candidate: Number presenting if the cluster is a candidate
%       * validated: To '1' if the cluster is validated, to '0' otherwise
%       * M: nM-1 cluster M
%       * nM: Number informing about the number cluster M
%   - clusterOut: K-1 array of structures of info about the output classes
%   - ks: Array of info about the number of classes [k,kValid,kXPFOut,kValidOut]
%   - t: Time increment from last execution time. To implement the state vector equation
%   - plotOn: If plotting XPF execution info (pdf,weights,ETC) is wanted to '1'
% Outputs :
%   - xParticInit: nN-nX array with the particles value after the re-init step
%   - xParticPred: nN-nX array with the particles after the pred/corr step,
%                  before the resampling
%   - wParticPred: nN-1 array with the particles' weight after the
%                  pred/corr step, before resampling
%   - xPartic: nN-nX array with the particles value at the end of the XPF
%   - clusterIn: K-1 array of structures of info about the classes
%   - clusterOut: K-1 array of structures of info about the output classes
%   - ks: array of info about the number of classes [k,kValid,kXPFOut,kValidOut]
% K number of clusters, and nY is the almost-half number of states
% (2=>X,Z of 5=>X,Z,Y,VX,VZ)
% CAREFUL: WE ARE ALWAYS WORKING ONLY WITH X AND Z. Y IS ONLY USED TO PLOT IN THE IMAGE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ============================================================
% DEFINE STATIC/PERMANENT VARIABLESD
% ============================================================

persistent cenAntIn;
persistent cenAntOut;
persistent bufferIn;
persistent validIn;
```

```
% ========================================================
% DETERMINE INITIAL UNCERTAINTY OF EVERY OBJECT POSITION
% ========================================================

y=Sistema.y;

if ParamsXPF(1)==0

    % ========================================================
    % CALLING THE CLUSTERING PROCESS
    % ========================================================

    cenAntIn=[]; cenAntOut=[]; bufferIn=[]; validIn=[];

    % Calling the cluster algorithm as initialization
    if Clases.type(1)==1

   [clusterIn,cenAntIn,kXPFIn,validIn]=clusterKMeans3SinBuff(y,clusterIn,cenAntIn,...
                                      Clases.in,t);
    else
        [xzDensityIn,clusterIn,cenAntIn,kXPFIn,validIn]=...
            clusterSubtract3SinBuff(y,clusterIn,cenAntIn,Clases.in,t);
    end
    kValidIn=length(validIn);
    ks(1:2)=[kXPFIn kValidIn];

    % ========================================================
    % GENERATING THE FIRST PDF
    % ========================================================

    % Generating the first pdf. probar si uso k o kValid
    nParticRest=ParamsXPF(2);
    nParticAux=fix(nParticRest/kValidIn);
    nPartic=0;
    for i=1:kValidIn
        nParticAsig=nParticRest-(kValidIn-i)*nParticAux;
        nMeasCluster=clusterIn(validIn(i)).nM;
        if nMeasCluster<=nParticAsig
            xPartic(nPartic+1:nPartic+nMeasCluster,[1:3])=clusterIn(validIn(i)).M;

        else
            xPartic(nPartic+1:nPartic+nMeasCluster,[1:3])=...
                clusterIn(validIn(i)).M(randsample(nMeasCluster,nParticAsig),:);
            nMeasCluster=nParticAsig;
        end

        xPartic(nPartic+1:nPartic+nMeasCluster,[4:5])=...
            repmat(clusterIn(validIn(i)).C(4:5),nMeasCluster,1);
        nParticRest=nParticRest-nMeasCluster;
        nPartic=nPartic+nMeasCluster;
    end
    xParticInit=[];
    xParticPred=[];
    wParticPred=[];

% ============================================================
% PF LOOP
% ============================================================


else
```

```
% ========================================================
% REINITIALIZATION STEP
% ========================================================

kXPFIn=ks(1);
kValidIn=ks(2);
kXPFOut=ks(3);
nPartic=size(xPartic,1);
xParticInit=xPartic;

% Distributing the measurements at t-1 within the nM particles
nParticRest=ParamsXPF(3);
nParticAux=fix(nParticRest/kValidIn);
for i=1:kValidIn
    nParticAsig=nParticRest-(kValidIn-i)*nParticAux;
    nMeasCluster=clusterIn(validIn(i)).nM;
    if nMeasCluster<=nParticAsig
        xParticInit(nPartic+1:nPartic+nMeasCluster,[1:3])=clusterIn(validIn(i)).M;
    else
        xParticInit(nPartic+1:nPartic+nParticAsig,[1:3])=...
                  clusterIn(validIn(i)).M(randsample(nMeasCluster,nParticAsig),:);
        nMeasCluster=nParticAsig;
    end

    % Suppose that I want to use the unsused M/k M of other clusters
    xParticInit(nPartic+1:nPartic+nMeasCluster,[4:5])=...
                      repmat(clusterIn(validIn(i)).C(4:5),nMeasCluster,1);
    nParticRest=nParticRest-nMeasCluster;
    nPartic=nPartic+nMeasCluster;
end

% ================================================================
% PREDICTION STEP: We use the transition prior as proposal
% Actualization of the particles
% ================================================================

% Here important to insert noise in order to increase particles dispersion
processNoise=Sistema.Params(3)*randn(nPartic,Sistema.Params(1));
xParticPred=MIffun(xParticInit,processNoise,t);

% ================================================================
% CENTROIDS AND BUFFER MEMBERS PREDICTION
% ================================================================

% Estimating the centroids for this time step
 for i=1:kXPFIn
    cenAntIn(clusterIn(i).I,:)=clusterIn(i).C;
    clusterIn(i).C=MIffun(clusterIn(i).C,zeros(1,5),t);
 end

% ================================================================
% CALLING THE CLUSTERING PROCESS
% ================================================================

% Calling the cluster algorithm for measurements at t. Ver lo comentado
y=Sistema.y;
if Clases.type(1)==1
    [clusterIn,cenAntIn,kXPFIn,validIn]=...
        clusterKMeans3SinBuff(y,clusterIn,cenAntIn,Clases.in,t);
else
    [xzDensityIn,clusterIn,cenAntIn,kXPFIn,validIn]=...
        clusterSubtract3SinBuff(y,clusterIn,cenAntIn,Clases.in,t);
end
```

```
kValidIn=length(validIn);
ks=[kXPFIn kValidIn];

% ================================================================
% OBTAINING THE 'U' ARRAY FOR CENTROIDS:
% ================================================================

for i=1:kXPFIn
    clusterIn(i).C(4:5)=(clusterIn(i).C(1:2)-cenAntIn(clusterIn(i).I,1:2))/t;
end

% ================================================================
% EVALUATE IMPORTANCE WEIGHTS
% ================================================================

if kXPFIn
    yPred=MIhfun(xParticPred,zeros(nPartic,Sistema.Params(2)));
end

j=1;
D=[];
wParticPred=[];
if nPartic
    for i=1:kXPFIn
        D(:,j)=sum((yPred(:,1:2)-repmat(clusterIn(i).C(1:2),nPartic,1)).^2,2);
        j=j+1;
    end
end
[auxRuido,foo]=min(D,[],2);

if kXPFIn
    auxRuido=-0.5*auxRuido/(Sistema.Params(4)*Sistema.Params(4));
    wParticPred=exp(auxRuido)/(Sistema.Params(4)*sqrt(2*pi));
end

% Normalise the weights
auxRuido=sum(wParticPred);
if auxRuido
    wParticPred=wParticPred./sum(wParticPred);
end

% ================================================================
% PLOTTING
% ================================================================

if plotOn

    figure(20); clf;
    subplot(411);
    if nPartic
        [hx,vx]=hist(xParticPred(:,1),50);
        bar(vx,hx); axis([-4000 4000 0 nPartic/5]);
    end
    ylabel('xPred'); title('xPred histogram. all vbles in mm and over nPartic/5');
    subplot(412);
    if nPartic
        [hz,vz]=hist(xParticPred(:,2),50);
        bar(vz,hz); axis([0 16000 0 nPartic/5]);
    end
    ylabel('zPred');
    subplot(413);
    if nPartic
        [hdx,vdx]=hist(xParticPred(:,4),50);
```

```
            bar(vdx,hdx);
            axis([min(xParticPred(:,4)) max(xParticPred(:,4))+1 0 nPartic/5]);
        end
        ylabel('dxPred');
        subplot(414);
        if nPartic
            [hdz,vdz]=hist(xParticPred(:,5),50);
            bar(vdz,hdz);
            axis([min(xParticPred(:,5)) max(xParticPred(:,5))+1 0 nPartic/5]);
        end
        ylabel('dzPred');

        % weigths
        figure(21); clf;
        stem(wParticPred);
        axis([0 nPartic+1 0 0.01]);
        title('lik histogam before resampling ');
        ylabel('weigths over 0.01');

        % Also a nice representation xPred with their weigth
        figure(22); clf;
        if isempty(wParticPred)==0 & isempty(xParticPred)==0
            plot3(xParticPred(:,1),xParticPred(:,2),wParticPred,'r.');
        end
        axis([-4000 4000 0 16000 0 0.01]); grid;
        title('xPred pdf with their weigth before resampling');
        xlabel('x mm'); ylabel('z mm'); zlabel('weigh over 0.01');

    end

    % ================================================================
    % SELECTION STEP
    % ================================================================

    % See if it is needed to reinitialize the algorithm
    idxOut=[];
    if auxRuido

        % Residual resampling
        if ParamsXPF(5)==1
            idxOut=MIresidualR(1:nPartic,wParticPred,ParamsXPF(2)-ParamsXPF(3));

            % Systematic resampling
        elseif ParamsXPF(5)==2
            idxOut=MIsystematicR(1:nPartic,wParticPred,ParamsXPF(2)-ParamsXPF(3));

            % Multinomial resampling
        else
            idxOut=MImultinomialR(1:nPartic,wParticPred,ParamsXPF(2)-ParamsXPF(3));
        end
    end

    % Keep particles with resampled indices
    xPartic=xParticPred(idxOut,:);

    % ================================================================================
    % PLOTTING
    % ================================================================================

    if plotOn

        figure(24); clf;
        subplot(411);
```

```
        if nPartic
            hist(xPartic(:,1),50); axis([-4000 4000 0 nPartic/5]);
        end
        ylabel('xPartic');
        title('xPartic histogram. all vbles in mm and over nPartic/5');
        subplot(412);
        if nPartic
            hist(xPartic(:,2),50); axis([0 16000 0 nPartic/5]);
        end
        ylabel('zPartic');
        subplot(413);
        if nPartic & isempty(xPartic)==0
            hist(xPartic(:,4),50);
           axis([min(xPartic(:,4)) max(xPartic(:,4))+1 0 nPartic/5]);
        end
        ylabel('dxPartic');
        subplot(414);
        if nPartic & isempty(xPartic)==0
            hist(xPartic(:,5),50);
            axis([min(xPartic(:,5)) max(xPartic(:,5))+1 0 nPartic/5]);
        end
        ylabel('dzPartic');

        % Babies
        idxIn=[];
        if isempty(idxOut)==0
            for i=1:size(xParticPred,1)
                foo=find(idxOut==i);
                idxIn(i)=length(foo);
            end
        end
        figure(25); clf;
        stem(idxIn);
        axis([0 size(xParticPred,1)+1 0 (nPartic+1)/100]);
        title('lik histogram after resampling');
        ylabel('babies over nPartic/100');

        % Also a nice representation xPred with their weigth
        figure(26); clf;
        if isempty(idxIn)==0 & isempty(xParticPred)==0
            plot3(xParticPred(:,1),xParticPred(:,2),idxIn,'r.');
            axis([-4000 4000 0 16000 0 nPartic/100]); grid;
        end
        title('x with their babies after resampling');
        xlabel('x mm'); ylabel('z mm'); zlabel('babies over nPartic/100');
    end

    % ================================================================
    % CLUSTER-OUT CENTROIDS PREDICTION
    % ================================================================

    % Estimating the centroids for this time step
    for i=1:kXPFOut
        cenAntOut(clusterOut(i).I,:)=clusterOut(i).C;
        clusterOut(i).C=MIffun(clusterOut(i).C,zeros(1,Sistema.Params(1)),t);
    end

    % ================================================================================
    % OBTAINING THE OUTPUTS
    % ================================================================================

    % Calling the clustering algorithm
    if Clases.type(2)==1
```

```
        [clusterOut,cenAntOut,kXPFOut,validOut]=...
            clusterKMeans3SinBuff(xPartic,clusterOut,cenAntOut,Clases.out,t);
    else
        [xzDensityOut,clusterOut,cenAntOut,kXPFOut,validOut]=...
            clusterSubtract3SinBuff(xPartic,clusterOut,cenAntOut,Clases.out,t);
    end
    ks(3:4)=[kXPFOut length(validOut)];

    % ========================================================================
    % OBTAINING THE 'U' ARRAY FOR CLUSTER-OUT CENTROIDS
    % ========================================================================

    % Estimating the rest of x-vector for validated and candidate clusters
    for i=1:kXPFOut
        clusterOut(i).C(4:5)=(clusterOut(i).C(1:2)-cenAntOut(clusterOut(i).I,1:2))/2;
    end
end
```

"calculaDist.m"

```
function [C,D,I,k,ident,cl] = calculaDist(datos,distM,changed,C,D,I,k,ident,cl)
% Calculate point to cluster centroid distances. Deleting clusters with near centroids
% Author  : Marta Marron Romera
% Date    : 12-06-06 / 12-12-06
% Sintaxis: [C,D,I,k,ident,cl]=calculaDist(datos,distM,changed,C,D,I,k,ident,cl)
% Inputs  :   - datos: nDatos-nX/nY datos matrix. Rows of datos correspond to points,
   columns to variables
%            - distM: Maximal distance to consider another cl
%                - changed: list of order identifiers that have changed its cluster
   correspondence
%            - C: k-nX/nY list of centroids
%            - D: 1-k distances between centroids
%            - I: 1-k list of identifier
%            - k: escalar number of clusters
%            - ident: 1-? list of free identifiers
%            - cl: K-1 array of Structures with the previous clustering task
%                   * I: number to identify a cluster and to do clustering
%                        association over time
%                   * C: 1-nX/nY array with the state variables of
%                        the centroid of the cl
%                   * P: number in %1 presenting the cluster probability
%                   * candidate: number presenting if the cluster is a candidate
%                   * validated: to 1 if the cluster is validated, to 0 otherwise
%                   * M: nM-nX/nY cluster M
%                   * nM: number informing about the number cluster M
% Outputs  : - C: k-nX/nY list of centroids
%            - D: 1-k distances between centroids
%            - I: 1-k list of identifier
%            - k: escalar number of clusters
%            - ident: 1-? list of free identifiers
%            - cl: k-1 array of Structures generated
% K is the number of clusters, and P is the half of the number of states (2:X,Z)
% CAREFUL: WE ARE ALWAYS WORKING ONLY WITH X AND Z (this is the reason for P to be 2)
%          Y IS ONLY USED TO PLOT IN THE IMAGE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ====================================
% INIT
```

```
% ====================================

nDatos=size(datos,1);
INTINF=32000000;

% ====================================
% calculating distancies to existing clusters
% ====================================

if nDatos
    for i=changed
        D(:,i)=sum((datos(:,1:2)-repmat(C(i,1:2),nDatos,1)).^2,2);
    end
end

for i=1:nDatos

    % The distance is bigger than distM
    if isempty(find(D(i,:)<distM))
        k=k+1;
        cl(k).C=zeros(1,5);
        C=[C;datos(i,:)];

        % D and I are also augmented
        D=[D sum((datos(:,1:2)-repmat(C(end,1:2),nDatos,1)).^2,2)];
        if isempty(ident)
            foo=max(I)+1;
        else
            foo=ident(1);
            ident(1)=[];
        end
        I=[I foo];
        cl(k).I=foo;
        cl(k).P=0;
        cl(k).candidate=-INTINF;
        cl(k).validated=false;
    end
end
```

"ClusterKMeans3SinBuff.m"

```
function [cl,cenAnt,k,valid] = clusterKMeans3SinBuff(datos,cl,cenAnt,params,t)
% Clustering with KMEANS SIN BUFFER
% Author  : Marta Marron Romera
% Date    : 12-06-06 / 12-12-06
% Sintaxis: [cl,cenAnt,k,valid]=clusterKMeans3SinBuff(datos,cl,cenAnt,params,t);
% Inputs  :
%   - datos: nDatos-nY/nX datos matrix. Rows of datos correspond to points,
%            columns to variables
%   - cl: K-1 array of Structures with the previous clustering task
%       * I: number to identify a cluster and to do clustering association over time
%       * C: 1-nY/1-nX array with the state variables (the position in xz plane)
%            of the centroid
%       * P: number in %1 presenting the cluster probability
%       * candidate: number presenting if the cluster is a candidate
%       * validated: to 1 if the cluster is validated, to 0 otherwise
%       * M: nM-nX/nY cluster M
%       * nM: number informing about the number cluster M
%   - cenAnt: K-nX/nY array with the centroids of previous cluster.
%             To erase when cl dissapears
```

```
%   - params: Array with clustering parameters (in the order indicated)
%       - nXY: Lenght of the data vector to cluster, can be nX or nY, or a mixeture
%       - iterM: Num max of iterations at the clustering algorithm
%       - canM: Numberof iter. that cluster is invalid/valid before valida/invalid
%       - hist: histheresys to valid/invalid due to cluster lik | dist to pred centroide
%       - factOlv: forgeting factor in the cluster likelihood
%       - kLikM: normalizing factor to calc cluster lik according to the number clusters
%       - distM: Different uses
%       - um: threshold to validate the cluster according to its likelihood
%       - distRM: Not used
%       - umOutliers: (subtract) pdf threshold to create new cluster
%   - t: to ponderate speed (if used) when obtaining distances
% Outputs :
%   - cl: K-1 array of Structures generated
%   - cenAnt: K-nX/nY array with the centroids of the previous cluster arrangement
%   - k: escalar number of clusters
%   - valid: Array of cluster indexes (not identifiers) that are validated.
% K is the number of clusters
% CAREFUL: WE ARE ALWAYS WORKING ONLY WITH X AND Z, Y IS ONLY USED TO PLOT IN THE IMAGE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ====================================
% Defining variables
% ====================================

% Defining variables to validate/invalidate
nDatos=size(datos,1);
iter=0;
valid=[];
I=[];
C=[];
INTINF=32000000;

% Params
nXY=params(1);
iterM=params(2);
canM=params(3);
hist=params(4);
factOlv=params(5);
kLikM=params(6);
distM=params(7);
distHistM=(1+hist)^2*distM;
um=params(8);

% ====================================
% Initialization.
% ====================================

% Obtaining variables. To begin only one cl
if isempty(cl) & nDatos
    k=1;
    cl(1).C=datos(1,:);
    if nXY==3
        cl(1).C(4:5)=[0 0];
    end
    cl(1).I=1;
    cl(1).candidate=-INTINF;
    cl(1).validated=false;
    cl(1).P=0;
else
    k=length(cl);
end
```

```
% Obtaining info from the cluster structure
for i=1:k
    I(i)=cl(i).I;
    if nXY==5
        C(i,:)=cl(i).C;
    else
        C(i,:)=cl(i).C(1:3);
    end
end

% Creating list of free identifiers
ident=1:max(I);
ident(I)=[];

% clusters that have changed their contents. everything is newly assigned
changed=1:k;
D=[];
nidx=[];

% ====================================
% cluster loop
% ====================================

% Obtaining info from the cl structure
while true

    % Compute the distance from every point to each cl centroid
    [C,D,I,k,ident,cl]=calculaDist(datos,distM,changed,C,D,I,k,ident,cl);

    % Determine closest cl for each point and reassign points to clusters
    pidx=nidx;
    [foo,nidx]=min(D,[],2);

    % Every point moved, every cl will need an update
    if iter==0
        moved=1:nDatos;
        changed=1:k;

    % Determine which points moved
    else
        moved=find(nidx~=pidx);

        % If none moved the algorithm has to finish
        if length(moved)==0
            break;
        end

        % Find clusters that gained or lost M
        changed=unique([nidx(moved); pidx(moved)])';
    end

    % Calculate the new cl centroids for the next step
    for i=1:length(changed)
        M=find(nidx==changed(i));
        nm(changed(i))=length(M);
        if nm(changed(i))>0
            C(changed(i),:)=sum(datos(M,:),1)/nm(changed(i));
        end
    end

    iter=iter+1;
    if (iter>iterM)
        disp('Excesivo numero de iteraciones');
```

```
        break;
    end
end

% ====================================
% HYSTERESIS
% ====================================

if k<kLikM
%     um=(kLikM-k+1)*um/kLikM;
    um=um/k;
else
    um=um/kLikM;
end
umHist=um*(1-hist);

% ====================================
% Validating and invalidating clusters
% ====================================

changed=[];
for i=1:k
    pidx=find(nidx==i);
    cl(i).nM=length(pidx);

    % Inserted by Marta to have the M
    cl(i).M=datos(pidx,:);
    D=sum((C(i,1:2)-cl(i).C(1:2)).^2,2);
    if nXY==5
        cl(i).C=C(i,:);
    else
        cl(i).C(1:3)=C(i,:);
    end

    % if new cluster. Here cl.C should be 0,0 and D should be very big?
    if cl(i).candidate==-INTINF

        % This has been changed by Marta. NECESSARY
        if nm(i)==0
            changed=[changed i];
        else
            cenAnt(cl(i).I,:)=cl(i).C;
            cl(i).P=cl(i).nM/nDatos;
            if canM==0
                cl(i).candidate=1;
                cl(i).validated=true;
            else
                cl(i).candidate=-canM;
            end
        end

    % With clusters that already existed. if cl(i).candidate~=-INTINF,
    else

        % Probability, ojo a vacios
        if nm(i)==0
            cl(i).P=0;
        else
            cl(i).P=factOlv*(cl(i).nM/nDatos)+(1-factOlv)*cl(i).P;
        end

        % INVALIDATING. (cl(i).P<umHist && D>distM)
        if nm(i)==0 || cl(i).P<umHist || (D>distHistM && cl(i).P<um)
```

```
            if cl(i).candidate>0
                cl(i).candidate=-canM;
            else
                cl(i).candidate=cl(i).candidate-1;
            end
            if D>distHistM && cl(i).P<umHist
                cl(i).candidate=cl(i).candidate-1;
            end

            % ERASING PART I
            if cl(i).candidate<=-2*canM
                changed=[changed i];
            end

        % VALIDATING
        elseif  cl(i).candidate<0  &&  ((cl(i).P>um  &&  D<distHistM)  ||  (D<distM  &&
   cl(i).P>umHist))
            cl(i).candidate=cl(i).candidate+1;
            if D<distM && cl(i).P>um
                cl(i).candidate=cl(i).candidate+1;
            end
            if cl(i).candidate>=0
                cl(i).candidate=1;
                cl(i).validated=true;
            end
        end
    end
end

% ====================================
% Definitly erasing
% ====================================

for i=changed
    cenAnt(cl(i).I,:)=zeros(1,size(cenAnt,2));
end
cl(changed)=[];
k=k-length(changed);

% ====================================
% Rellenando array de validos
% ====================================

for i=1:k
    if cl(i).validated==true
        valid=[valid i];
    end
end
```

“SJPDAF_Total.m”/ “SJPDAF_NN_Total.m”

```
function []=SJPDAF_Total(file,videoOn,plotOn,saveOn)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TRACKING OF MULTIPLE OBJECTS WITH A SJPDAF ALGORITHM
% Author  : Maria Cabello Aguilar
% Date    : June 2007
% Sintaxis: SJPDAF_Total(file,videoOn,plotOn,saveOn)
% Inputs  : - file: The path of the datas file to segment
%           - plotOn: If plotting execution info (pdf,weights,etc) is wanted to '1'
%           - videoOn: If plotting in the image is wanted set to '1'
```

```
%           - saveOn: If wanted to store the results in a video set to '1'
% Outputs :  All info generated is stored in a file named resutlt.mat
% CAREFUL: WE ARE ALWAYS WORING ONLY WITH X AND Z. Y IS ONLY USED TO PLOT IN THE IMAGE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

warning off MATLAB:colon:operandsNotRealScalar;
warning off MATLAB:divideByZero;

% =========================================================================
% INITIALIZING SISTEMA-STRUCT
% =========================================================================

% Initializing Sistema Parameters. - Sistema: Stucture with system information
%       * y: Array of measurements [x,z,y]
%       * Params: Parameters of the system
%           - nX: Number of states of the estimated obstacle [x,z,y,vz,vx]
%           - nY: Number of elements of the observation vector [x,z,y]
%           - cX: Standard deviation (NO Variance) of the Gaussian process noise
%           - cY: Standard deviation (NO Variance) of the Gaussian measurement noise
SistemaXPF.Params=[5 3 10 50];
distMXPF=700;
iterXPF=0;


% =========================================================================
% INITIALIZING PARAM-STRUCT
% =========================================================================

% Initializing Params Array
%       - iterXPF: the state variable of the filter state machine
%       - nN: Number total of particles
%       - nM: Number of samples inserted directly from observation in the pdf reinit
%       - nB: (>=.nP) Number of meas can be in the buffer.Not used.
%             Not erased for coherency
%       - rS: possible choices: systematic sampling (2),residual (1),and multinomial (3)
%       - distM: Different uses

ParamsXPF=[iterXPF 600 200 200 1 distMXPF^2];

IXPFOut=[]; CXPFOut=[]; BetaXPFOut=[]; nmXPFOut=[]; canXPFOut=[]; valXPFOut=[];
FiltroOut=[];

% =========================================================================
% INITIALIZING CLASES-STRUCT
% =========================================================================

% Initializing Clases struct.Structure with cluster inform
%       * in: Params of the measurements clustering
%           - nXY: Lenght of the data vector to cluster, can be nX or nY, or a mixture
%           - iterM: Number maximum of iterations at the clustering algorithm
%           - canM: Number of iter that cluster/filter is invalid/valid
%                   before valida/invalid
%           - hist: Histheresys to valid/invalid due to cluster lik | dist
%                   to predicted centroide
%           - factOlv: Forgeting factor in the cluster likelihood (only XPFCP)
%           - kLikM: Normalizing factor to calc cluster lik according
%                    to the number clusters (only XPFCP)
%           - distM: Different uses
%           - um: Threshold to validate the cluster according to its likelihood
%           - distRM: (subtract) Enable area for cluster intersection.
%           - umOutliers: (subtract) Pdf threshold to create new cluster
%       * out: IDEM THAN PARAMS but for the particles clustering
%       * type: For the input and output [typeIn,typeOut].
```

```
%              Choices: kMeans (1), subtract (2)(only XPFCP)

ClasesXPF.type=[1 1];
ClasesXPF.in=[SistemaXPF.Params(2) 100 2 0.5 0.8 10 distMXPF^2 0.6...
      (1.5*distMXPF)^2 0.1];
if  ClasesXPF.type(2)==1
    ClasesXPF.out=[SistemaXPF.Params(1) 100 2 0.5 0.8 10 distMXPF^2...
            0 (1.5*distMXPF)^2 0.1];
else
    ClasesXPF.out=[SistemaXPF.Params(1) 100 2 0.5 0.8 10 550^2 0.6 (1.25*550)^2 0.1];
end


% ========================================================================
% VIDEO CONSTANTS
% ========================================================================

%Call function to define constants
constants(videoOn);

global FILE_SIZE FXL FYL U0L V0L TRAS_Y ROTATION_SCC;

% ========================================================================
% VARIABLES
% ========================================================================

% Define global variables
global INTINF T;
T=1/15;
INTINF=32000000;

% Define others variables
kXPFOut=0;
ksXPF=[0 0 0 0];
kXPFTotal=[];
xTotalXPF=[];
xXPF=[];
nParticTotal=[];
nPartic=0;
tTotal=[];
tTotalPlot=[];
tMedia=[];
kValid=[];
kMedia=[];

% ========================================================================
% TAKING INFORMATION FROM FILES
% ========================================================================

% Data Open
str1=['..\data' file '.dat'];
pfdata=fopen(str1,'rb');
nMeas=fread(pfdata,1,'int32');

% Video Open
if videoOn
    str=['..\left' file '.str'];;
    pfVideo=fopen(str,'rb');
    I=fread(pfVideo,FILE_SIZE);
    square=zeros(3,6);
    square(2,:)=TRAS_Y-[200 200 1700 1700 200 1500];
end
iterVideo=1;
```

```
% ========================================================================
% WHILE THERE ARE DATAS
% ========================================================================

flagVideo=1;
while isempty(nMeas)==0

    % ====================================================================
    % TO REMOVE SOME FRAMES AT THE BEGINNING
    % ====================================================================

    if iterVideo<=750 & flagVideo
        y=fread(pfdata,[3 nMeas],'float32');
        iterVideo=iterVideo+1;
        nMeas=fread(pfdata,1,'int32');
        if videoOn
            I=fread(pfVideo,FILE_SIZE);
        end
        continue
    elseif iterVideo>750 & flagVideo
        flagVideo=0;
        iterVideo=1;
    end

    y=fread(pfdata,[3 nMeas],'float32');
    SistemaXPF.y=y';

    % ====================================================================
    % MEASURES VISUALITATION AT VIDEO AND 2D
    % ====================================================================

    % Plotting at 2D
    figure(1); clf;
    plot(y(1,:),y(2,:),'g.');
    axis([-5000 5000 0 18000]);
    xlabel('x in mm'); ylabel('z in mm');
    title('Representation 3D points obtained with matching epipolar');
    grid; hold on;

    % Plotting image in the film
    if videoOn

        % Show the film
        figure(2); clf;
        imshow(uint8((I')));
        title('Tracking objects 3D');
        hold on;

        % Show the measures coordinates
        y3=TRAS_Y-y(3,:);
        y(3,:)=y(2,:);
        y(2,:)=y3;

        % Transform the measures
        SCCmatr=ROTATION_SCC*y;
        U=round(FXL*(SCCmatr(1,:)./SCCmatr(3,:)) + U0L);
        V=round(FYL*(SCCmatr(2,:)./SCCmatr(3,:)) + V0L);

        % Plot the measures
        plot(U,V,'g.');
    end
```

```
% ========================================================================
% CALLING SJPDAF
% ========================================================================

if nPartic==0
    iterXPF=-INTINF;
end
if iterXPF==-INTINF & nMeas
    iterXPF=0;
end
ParamsXPF(1)=iterXPF;
tic
if iterXPF~=-INTINF

[xXPF,ksXPF,FiltroOut]=SJPDAF(SistemaXPF,ClasesXPF,ParamsXPF,xXPF,FiltroOut,ksXPF,plo
tOn);
end
kXPFOut=ksXPF(3);
nPartic=size(xXPF,1);
tTotal=[tTotal;toc];

% ========================================================================
% STORING RESULTS
% ========================================================================

kXPFTotal=[kXPFTotal;ksXPF];
nParticTotal(end+1)=nPartic;
if isempty(xXPF)==0
    xTotalXPF=[xTotalXPF;xXPF];
end

% ========================================================================
% PLOTTINGS
% ========================================================================

% Plotting 2D projection
if nPartic
    figure(1);
    plot(xXPF(:,1),xXPF(:,2),'y.');

    % Plotting image in the film
    if videoOn

        % Show the particles coordinates
        xPlot=xXPF(:,1:3)';
        x3=TRAS_Y-xPlot(3,:);
        xPlot(3,:)=xPlot(2,:);
        xPlot(2,:)=x3;

        % Transfor the particles
        SCCmatr=ROTATION_SCC*xPlot;
        U=round(FXL*(SCCmatr(1,:)./SCCmatr(3,:))+U0L);
        V=round(FYL*(SCCmatr(2,:)./SCCmatr(3,:))+V0L);

        % Plot the particles
        figure(2);
        plot(U,V,'y.');
    end
end
% ========================================================================
% PLOTTINGS. FILTRO-out
% ========================================================================
```

```
for i=1:kXPFOut
    % Save the clusters in independent arrays
    IXPFOut=[IXPFOut;FiltroOut(i).I];
    CXPFOut=[CXPFOut;FiltroOut(i).C];
    BetaXPFOut=[BetaXPFOut;FiltroOut(i).P];
    canXPFOut=[canXPFOut;FiltroOut(i).candidate];
    valXPFOut=[valXPFOut;FiltroOut(i).validated];
    nmXPFOut=[nmXPFOut;FiltroOut(i).nM];

    % Starting to paint in this frame
    if videoOn
        square(1,1:5)=[FiltroOut(i).C(1,1)-distMXPF/2 FiltroOut(i).C(1,1)+...
                  distMXPF/2  (FiltroOut(i).C(1,1)+distMXPF/2)...
                  FiltroOut(i).C(1,1)-distMXPF/2 FiltroOut(i).C(1,1)-distMXPF/2];
        square(3,1:5)=FiltroOut(i).C(1,2)*ones(1,5);
        square(:,6)=[FiltroOut(i).C(1,1);TRAS_Y-1500;FiltroOut(i).C(1,2)];
        SCCmatr=ROTATION_SCC*square;
        U=round(FXL*(SCCmatr(1,:)./SCCmatr(3,:))+U0L);
        V=round(FYL*(SCCmatr(2,:)./SCCmatr(3,:))+V0L);
    end

    % Painting the circles
    figure(1);
    if FiltroOut(i).validated
        circle(FiltroOut(i).C(1,1),FiltroOut(i).C(1,2),distMXPF,20,'r');
        if videoOn
            figure(2);
            plot(U(1:5),V(1:5),'r');
        end
    else
        plot(FiltroOut(i).C(1,1),FiltroOut(i).C(1,2),'r+');
        if videoOn
            figure(2);
            plot(U(6),V(6),'r+');
        end
    end
end

    %UPDATE iterXPF
if (iterXPF==0 & nMeas) | iterXPF>0
    iterXPF=iterXPF+1;
end


% =========================================================================
% WRITTING IMAGES INFORMATION
% =========================================================================

str1=sprintf('kSJPDAF=%d',ksXPF(4));
str2=sprintf('tExecSJPDAF=%d' ,round(tTotal(end,1)*1000));
str3=sprintf('iter=%d',iterXPF);
str4=sprintf('iterVideo=%d',iterVideo);

figure(1);

text(2000,500,str1);
text(-4700,17000,str2);
text(2000,17000,str3);
text(2000,16000,str4);

if videoOn
    figure(2);
```

```
        text(230,230,str1);
        text(20,10,str2);
        text(230,10,str3);
        text(230,20,str4);
    end

    if saveOn
        figure(1);
        proy(iterVideo)=getframe;
        if videoOn
            figure(2);
            video(iterVideo)=getframe;
        end
    end

    if iterVideo==10
        stop=1;
    end

   iterVideo=iterVideo+1;
   pause(0.01);
   nMeas=fread(pfdata,1,'int32');
   if videoOn
       I=fread(pfVideo,FILE_SIZE);
   end
end

% ========================================================================
% PLOTTING EXECUTION TIME
% ========================================================================

kValid=[kXPFTotal(:,4)'];
tTotalPlot=tTotal';

% Calculating Median
aux=size(kValid,2);
for i=1:(1/T):aux
    j=i+(1/T);
    if j>aux
        j=aux;
    end
    tMedia=[tMedia mean(tTotalPlot(:,i:j),2)];
    kMedia=[kMedia median(kValid(:,i:j),2)];
end

% To plot results and compare k
figure(3);

subplot(2,1,1);
plot(1:aux,kXPFTotal(:,4)','ro');
grid; axis([0 aux -2 max(max(kValid))+2]);
title('Number of SJPDAF clusters in red')
xlabel('iter,frame'); ylabel('kValid');
subplot(2,1,2);
grid; axis([0 aux -2 max(max(kValid))+2]);
hold on;
stairs(1:(1/T):aux,kMedia(1,:),'r');
title('Median in a second of the Number of clusters in red')
xlabel('iter,frame'); ylabel('kValid');
hold off;

% Time plot
figure(4);
```

```
subplot(2,1,1);
plot(1:aux,tTotalPlot(1,:),'r');
grid; axis([0 aux min(min(tTotalPlot)) max(max(tTotalPlot))]);
title('Execution time of each-loop of SJPDAF algorithm in red')
xlabel('iter,frame'); ylabel('sec');

subplot(2,1,2);
grid;
axis([0 aux min(min(tMedia)) max(max(tMedia))]);
hold on;
stairs(1:(1/T):aux,tMedia(1,:),'r');
title('Mean in a second of each-loop Execution time of the SJPDAF algorithm in red')
xlabel('iter,frame'); ylabel('sec');
hold off;

fclose(pfdata);
if videoOn
    fclose(pfVideo);
end

% Saving videos
if saveOn
    movie2avi(proy,'2Dproyection');
    if videoOn
        movie2avi(video,'video');
    end
end

% ========================================================================
% SJPDAF EXECUTION ENDING
% ========================================================================

clear FiltroOut;
clear kXPFOut;

% Saving results
nParticTotal=nParticTotal';
save result *Total *Out;
```

"SJPDAF.m"

```
function
   [xPartic,ks,FiltroOut]=SJPDAF(Sistema,Clases,ParamsXPF,xPartic,FiltroOut,ks,plotOn);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author  : Maria Cabello Aguilar
% Date    : June 2007
% Sintaxis: SJPDAF(Sistema,Clases,ParamsXPF,xPartic,FiltroOut,ks,plotOn);
% Inputs  : - Sistema: Stucture with system information
%           - Clases: Structure with clustering information
%           - ParamsXPF: Parameters of the filter
%           - xPartic: nN-nX array with the particles value at the end of the XPF
%           - FiltroOut: K-1 array of structures of info about the classes
%           - ks: array of info about the number of classes
%           - plotOn: If plotting execution info (pdf,weights,etc) is wanted to '1'
% Outputs : - xPartic: nN-nX array with the particles value at the end of the XPF
%           - ks: array of info about the number of classes
%           - FiltroOut: K-1 array of structures of info about the classes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% ========================================================
% DEFINE STATIC/PERMANENT VARIABLES
% ========================================================

global T;
persistent yInitAug;
persistent cenAntOut;

y=Sistema.y;
ny=size(y,1);
iter=ParamsXPF(1);
nN=ParamsXPF(2);
nM=ParamsXPF(3);
nX=Sistema.Params(1);
nY=Sistema.Params(2);
cX=Sistema.Params(3);
cY=Sistema.Params(4);
rS=ParamsXPF(5);
t=T;

if iter==0

    cenAntOut=[];
    % ========================================================
    % GENERATING THE FIRST PDF
    % ========================================================

    % Generating the first pdf
    if ny<=nN
        xPartic=y;
        nPartic=ny;
    else
        xPartic=y(randsample(ny,nN),:);
        nPartic=nN;
    end
    xPartic(:,[4:5])=zeros(nPartic,2);

    % Init
    xParticInit=[];
    xParticPred=[];
    wParticPred=[];
    yInitAug=y;
    yInitAug(:,4:5)=zeros(ny,2);

% ========================================================
% PF LOOP
% ========================================================

else

    % ========================================================
    % REINITIALIZATION STEP
    % ========================================================

    kXPFOut=ks(3);
    nPartic=size(xPartic,1);
    xParticInit=xPartic;
    nyInitAug=size(yInitAug,1);

    % Distributing the measurements at t-1 within the nM particles... nM
    if nyInitAug<=nM
        xParticInit(nPartic+1:nPartic+nyInitAug,:)=yInitAug;
        nPartic=nPartic+nyInitAug;
```

```
else
    xParticInit(nPartic+1:nPartic+nM,:)=yInitAug(randsample(nyInitAug,nM),:);
    nPartic=nPartic+nM;
end

% ========================================================
% PREDICTION STEP: We use the transition prior as proposal
% Actualization of the particles.
% ========================================================

% Here important to insert noise in order to increase particles dispersion
processNoise=cX*randn(nPartic,nX);
xParticPred=MIffun(xParticInit,processNoise);

% Obtaining the 'u' variable for the sampling.
yPred=MIhfun(xParticPred,zeros(nPartic,nY));
yInit=MIhfun(xParticInit,zeros(nPartic,nY));

% ========================================================
% EVALUATE IMPORTANCE WEIGHTS:
% ========================================================

% Calling the PDA algorithm for measurements at t
[yInitAug,wParticPred]=PDA3(y,yInit,yPred,cY);

% ================================================================
% PLOTTING
% ================================================================

if plotOn
    figure(20); clf;
    subplot(411);
    if nPartic
        [hx,vx]=hist(xParticPred(:,1),50);
        bar(vx,hx); axis([-4000 4000 0 nPartic/5]);
    end
    ylabel('xPred'); title('xPred histogram. all vbles in mm and over nPartic/5');
    subplot(412);
    if nPartic
        [hz,vz]=hist(xParticPred(:,2),50);
        bar(vz,hz); axis([0 16000 0 nPartic/5]);
    end
    ylabel('zPred');
    subplot(413);
    if nPartic
        [hdx,vdx]=hist(xParticPred(:,4),50);
        bar(vdx,hdx);
        axis([min(xParticPred(:,4)) max(xParticPred(:,4))+1 0 nPartic/5]);
    end
    ylabel('dxPred');
    subplot(414);
    if nPartic
        [hdz,vdz]=hist(xParticPred(:,5),50);
        bar(vdz,hdz);
        axis([min(xParticPred(:,5)) max(xParticPred(:,5))+1 0 nPartic/5]);
    end
    ylabel('dzPred');

    % weigths
    figure(21); clf;
    stem(wParticPred);
    axis([0 nPartic+1 0 0.01]);
    title('lik histogam before resampling ');
```

```
        ylabel('weigths over 0.01');

        % Also a nice representation xPred with their weigth
        figure(22); clf;
        if isempty(wParticPred)==0 & isempty(xParticPred)==0
            plot3(xParticPred(:,1),xParticPred(:,2),wParticPred,'r.');
        end
        axis([-4000 4000 0 16000 0 0.01]); grid;
        title('xPred pdf with their weigth before resampling');
        xlabel('x mm'); ylabel('z mm'); zlabel('weigh over 0.01');

    end

    % ================================================================
    % SELECTION STEP:
    % ================================================================

    % See if it is needed to reinitialize the algorithm
    idxOut=[];
    if sum(wParticPred)

        % Residual resampling
        if rS==1
            idxOut=MIresidualR(1:nPartic,wParticPred,nN-nM);

            % Systematic resampling
        elseif rS==2
            idxOut=MIsystematicR(1:nPartic,wParticPred,nN-nM);

            % Multinomial resampling
        else
            idxOut=MImultinomialR(1:nPartic,wParticPred,nN-nM);
        end
    end

    % Keep particles with resampled indices
    xPartic=xParticPred(idxOut,:);

    % ================================================================================
    % PLOTTING
    % ================================================================================

    if plotOn

        figure(24); clf;
        subplot(411);
        if nPartic
            hist(xPartic(:,1),50); axis([-4000 4000 0 nPartic/5]);
        end
        ylabel('xPartic');
        title('xPartic histogram. all vbles in mm and over nPartic/5');
        subplot(412);
        if nPartic
            hist(xPartic(:,2),50); axis([0 16000 0 nPartic/5]);
        end
        ylabel('zPartic');
        subplot(413);
        if nPartic & isempty(xPartic)==0
            hist(xPartic(:,4),50); axis([-2000 2000 0 nPartic/5]);
%             axis([min(xPartic(:,4)) max(xPartic(:,4))+1 0 nPartic/5]);
        end
        ylabel('dxPartic');
        subplot(414);
```

```
        if nPartic & isempty(xPartic)==0
            hist(xPartic(:,5),50); axis([-2000 2000 0 nPartic/5]);
%             axis([min(xPartic(:,5)) max(xPartic(:,5))+1 0 nPartic/5]);
        end
        ylabel('dzPartic');

        % Babies
        idxIn=[];
        if isempty(idxOut)==0
            for i=1:size(xParticPred,1)
                foo=find(idxOut==i);
                idxIn(i)=length(foo);
            end
        end
        figure(25); clf;
        stem(idxIn);
        axis([0 size(xParticPred,1)+1 0 (nPartic+1)/100]);
        title('lik histogram after resampling');
        ylabel('babies over nPartic/100');

        % Also a nice representation xPred with their weigth
        figure(26); clf;
        if isempty(idxIn)==0 & isempty(xParticPred)==0
            plot3(xParticPred(:,1),xParticPred(:,2),idxIn,'r.');
            axis([-4000 4000 0 16000 0 nPartic/100]); grid;
        end
        title('x with their babies after resampling');
        xlabel('x mm'); ylabel('z mm'); zlabel('babies over nPartic/100');
    end
    % ================================================================
    % CLUSTER-OUT CENTROIDS PREDICTION
    % ================================================================

    % Estimating the centroids for this time step. No meto ruido, hay que mirar si
    quiero meterlo
    for i=1:kXPFOut
        cenAntOut(FiltroOut(i).I,:)=FiltroOut(i).C;
        FiltroOut(i).C=MIffun(FiltroOut(i).C,zeros(1,Sistema.Params(1)));
    end

    % ================================================================================
    % OBTAINING THE OUTPUTS
    % ================================================================================

    % Calling the clustering algorithm
    if Clases.type(2)==1
        [FiltroOut,cenAntOut,kXPFOut,validOut]=...
            clusterKMeans3SinBuff(xPartic,FiltroOut,cenAntOut,Clases.out,t);
    else
        [xzDensityOut,FiltroOut,cenAntOut,kXPFOut,validOut]=...
            clusterSubtract3SinBuff(xPartic,FiltroOut,cenAntOut,Clases.out,t);
    end
    ks(3:4)=[kXPFOut length(validOut)];

    % ================================================================
    % OBTAINING THE 'U' ARRAY FOR CLUSTER-OUT CENTROIDS
    % ================================================================

    % Estimating the rest of x-vector for validated and candidate clusters... AQUI NO!!!
    for i=1:kXPFOut
        FiltroOut(i).C(4:5)=(FiltroOut(i).C(1:2)-cenAntOut(FiltroOut(i).I,1:2))/2;
    end
```

```
end
```

“PDA3.m”

```
function [yInitAug,wPred]=PDA3(y,yInit,yPred,cY)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PDA.M - ASSOCIATION ALGORITHM
% Author    : Maria Cabello Aguilar
% Date      : June 2007
% Sintaxis  : PDA3(y,yInit,yPred,cY)
% Input     : - y: Containes the input coordinates of the measures
%             - yInit: Particles value in last iter
%             - yPred: Predicted state of each particle
%             - cY: Standard deviation (NO Variance) of the Gaussian measurement noise
% Output    : - yInitAug: Particles value in this iter
%             - wPred: nN-1 array with the particles' weigh after the pred/corr step
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global INTINF T;

nPartic=size(yPred,1);
ny=size(y,1);

%==========================================================================
%   CALCULATION OF DISTANCE
%==========================================================================

D=[];
for i=1:nPartic
    D(:,i)=sum((y(:,1:2)-repmat(yPred(i,1:2),ny,1)).^2,2);
end

%Finds the shortest distance
[auxRuido,idxMeas]=min(D,[],1);

%======================================================================
%   BETAS
%======================================================================

% Calculating Gauss
if nPartic
    auxRuidoT=((-0.5)*D)./(cY*cY);
    wPredT=exp(auxRuidoT)./(cY*sqrt(2*pi));
end

% Normalise the weights
for i=1:nPartic
    auxRuidoSum(:,i)=sum(wPredT(:,i));
end
auxSumBetas=sum(auxRuidoSum);
if auxSumBetas
    for i=1:nPartic
        wPred(:,i)=auxRuidoSum(:,i)/auxSumBetas;
    end
else
    wPred=zeros(1,nPartic);
end
wPred=wPred';
%======================================================================
%   ASSIGNMENT
```

```
%=========================================================================

% Inserting meas that are false alarms
idxFalse=[1:ny];
idxFalse(idxMeas)=[];
nyFalse=length(idxFalse);

if ny
    yInitAug(1:nyFalse,:)=y(idxFalse,:);
    yInitAug(1:nyFalse,4:5)=zeros(nyFalse,2);
%     Inserting meas that are related with a particle
    yInitAug(nyFalse+1:nyFalse+nPartic,1:3)=y(idxMeas,:);
    yInitAug(nyFalse+1:nyFalse+nPartic,4:5)=(y(idxMeas,1:2)-yInit(:,1:2))/T;
else
    yInitAug=[];
end
```

"SJPDAF_NN.m"

```
function [xPartic,ks,FiltroOut]=...
   SJPDAF_NN(Sistema,Clases,ParamsXPF,xPartic,FiltroOut,ks,plotOn);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author  : Maria Cabello Aguilar
% Date    : June 2007
% Sintaxis: SJPDAF_NN(Sistema,Clases,ParamsXPF,xPartic,FiltroOut,ks,plotOn);
% Inputs  : - Sistema: Stucture with system information
%           - Clases: Structure with clustering information
%           - ParamsXPF: Params of the Filter
%           - xPartic: nN-nX array with the particles value at the end of the XPF
%           - FiltroOut: K-1 array of structures of info about the classes
%           - ks: array of info about the number of classes
%           - plotOn: If plotting execution info (pdf,weights,etc) is wanted to '1'
% Outputs : - xPartic: nN-nX array with the particles value at the end of the XPF
%           - ks: array of info about the number of classes
%           - FiltroOut: K-1 array of structures of info about the classes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ========================================================
% DEFINE STATIC/PERMANENT VARIABLES
% ========================================================

global T;
persistent yInitAug;
persistent cenAntOut;

y=Sistema.y;
ny=size(y,1);
iter=ParamsXPF(1);
nN=ParamsXPF(2);
nM=ParamsXPF(3);
nX=Sistema.Params(1);
nY=Sistema.Params(2);
cX=Sistema.Params(3);
cY=Sistema.Params(4);
rS=ParamsXPF(5);
t=T;

if iter==0

    cenAntOut=[];
```

```
    % ========================================================
    % GENERATING THE FIRST PDF
    % ========================================================

    % Generating the first pdf
    if ny<=nN
        xPartic=y;
        nPartic=ny;
    else
        xPartic=y(randsample(ny,nN),:);
        nPartic=nN;
    end
    xPartic(:,[4:5])=zeros(nPartic,2);

    % Init
    xParticInit=[];
    xParticPred=[];
    wParticPred=[];
    yInitAug=y;
    yInitAug(:,4:5)=zeros(ny,2);

% ========================================================
% PF LOOP
% ========================================================

else

    % ========================================================
    % REINITIALIZATION STEP
    % ========================================================

    kXPFOut=ks(3);
    nPartic=size(xPartic,1);
    xParticInit=xPartic;
    nyInitAug=size(yInitAug,1);

    % Distributing the measurements at t-1 within the nM particles... nM
    if nyInitAug<=nM
        xParticInit(nPartic+1:nPartic+nyInitAug,:)=yInitAug;
        nPartic=nPartic+nyInitAug;
    else
        xParticInit(nPartic+1:nPartic+nM,:)=yInitAug(randsample(nyInitAug,nM),:);
        nPartic=nPartic+nM;
    end

    % ========================================================
    % PREDICTION STEP: We use the transition prior as proposal
    % Actualization of the particles.
    % ========================================================

    % Here important to insert noise in order to increase particles dispersion!!!
    processNoise=cX*randn(nPartic,nX);
    xParticPred=MIffun(xParticInit,processNoise);

    yPred=MIhfun(xParticPred,zeros(nPartic,nY));
    yInit=MIhfun(xParticInit,zeros(nPartic,nY));

    % ========================================================
    % EVALUATE IMPORTANCE WEIGHTS:
    % ========================================================

    % Calling the PDA algorithm for measurements at t
    [yInitAug,wParticPred]=PDA2(y,yInit,yPred,cY);
```

```
% ================================================================
% PLOTTING:
% ================================================================

if plotOn
    figure(20); clf;
    subplot(411);
    if nPartic
        [hx,vx]=hist(xParticPred(:,1),50);
        bar(vx,hx); axis([-4000 4000 0 nPartic/5]);
    end
    ylabel('xPred'); title('xPred histogram. all vbles in mm and over nPartic/5');
    subplot(412);
    if nPartic
        [hz,vz]=hist(xParticPred(:,2),50);
        bar(vz,hz); axis([0 16000 0 nPartic/5]);
    end
    ylabel('zPred');
    subplot(413);
    if nPartic
        [hdx,vdx]=hist(xParticPred(:,4),50);
        bar(vdx,hdx);    axis([min(xParticPred(:,4))    max(xParticPred(:,4))+1    0
nPartic/5]);
    end
    ylabel('dxPred');
    subplot(414);
    if nPartic
        [hdz,vdz]=hist(xParticPred(:,5),50);
        bar(vdz,hdz);    axis([min(xParticPred(:,5))    max(xParticPred(:,5))+1    0
nPartic/5]);
    end
    ylabel('dzPred');

    % weigths
    figure(21); clf;
    stem(wParticPred);
    axis([0 nPartic+1 0 0.01]);
    title('lik histogam before resampling ');
    ylabel('weigths over 0.01');

    % Also a nice representation xPred with their weigth
    figure(22); clf;
    if isempty(wParticPred)==0 & isempty(xParticPred)==0
        plot3(xParticPred(:,1),xParticPred(:,2),wParticPred,'r.');
    end
    axis([-4000 4000 0 16000 0 0.01]); grid;
    title('xPred pdf with their weigth before resampling');
    xlabel('x mm'); ylabel('z mm'); zlabel('weigh over 0.01');
end

papel=[];
for i=1:nPartic
    if xParticPred(i,1)>1000
        papel=[papel i];
    end
end

% ================================================================
% SELECTION STEP:
% ================================================================

% See if it is needed to reinitialize the algorithm
```

```
    idxOut=[];
    if sum(wParticPred)

        % Residual resampling
        if rS==1
            idxOut=MIresidualR(1:nPartic,wParticPred,nN-nM);

            % Systematic resampling
        elseif rS==2
            idxOut=MIsystematicR(1:nPartic,wParticPred,nN-nM);

            % Multinomial resampling
        else
            idxOut=MImultinomialR(1:nPartic,wParticPred,nN-nM);
        end
    end

    % Keep particles with resampled indices
    xPartic=xParticPred(idxOut,:);

    % ========================================================================
    % PLOTTING
    % ========================================================================

    if plotOn

        % OJO, EL HIST SE HACE EN 50 PARA EQUIPARAR CON PLOT ANTERIOR
        figure(24); clf;
        subplot(411);
        if nPartic
            hist(xPartic(:,1),50); axis([-4000 4000 0 nPartic/5]);
        end
        ylabel('xPartic');  title('xPartic  histogram.  all  vbles  in  mm  and  over
    nPartic/5');
        subplot(412);
        if nPartic
            hist(xPartic(:,2),50); axis([0 16000 0 nPartic/5]);
        end
        ylabel('zPartic');
        subplot(413);
        if nPartic & isempty(xPartic)==0
            hist(xPartic(:,4),50);axis([-2000 2000 0 nPartic/5]);
        end
        ylabel('dxPartic');
        subplot(414);
        if nPartic & isempty(xPartic)==0
            hist(xPartic(:,5),50);axis([-2000 2000 0 nPartic/5]);
        end
        ylabel('dzPartic');

        % Babies
        idxIn=[];
        if isempty(idxOut)==0
            for i=1:size(xParticPred,1)
                foo=find(idxOut==i);
                idxIn(i)=length(foo);
            end
        end
        figure(25); clf;
        stem(idxIn);
        axis([0 size(xParticPred,1)+1 0 (nPartic+1)/100]);
        title('lik histogram after resampling');
        ylabel('babies over nPartic/100');
```

```
        % Also a nice representation xPred with their weigth
        figure(26); clf;
        if isempty(idxIn)==0 & isempty(xParticPred)==0
            plot3(xParticPred(:,1),xParticPred(:,2),idxIn,'r.');
            axis([-4000 4000 0 16000 0 nPartic/100]); grid;
        end
        title('x with their babies after resampling');
        xlabel('x mm'); ylabel('z mm'); zlabel('babies over nPartic/100');
    end

    % ===============================================================
    % CLUSTER-OUT CENTROIDS PREDICTION
    % ===============================================================

    % Estimating the centroids for this time step
    for i=1:kXPFOut
        cenAntOut(FiltroOut(i).I,:)=FiltroOut(i).C;
        FiltroOut(i).C=MIffun(FiltroOut(i).C,zeros(1,Sistema.Params(1)));
    end

    % ========================================================================
    % OBTAINING THE OUTPUTS
    % ========================================================================

    % Calling the clustering algorithm
    if Clases.type(2)==1
        [FiltroOut,cenAntOut,kXPFOut,validOut]=...
            clusterKMeans3SinBuff(xPartic,FiltroOut,cenAntOut,Clases.out,t);
    else
        [xzDensityOut,FiltroOut,cenAntOut,kXPFOut,validOut]=...
            clusterSubtract3SinBuff(xPartic,FiltroOut,cenAntOut,Clases.out,t);
    end
    ks(3:4)=[kXPFOut length(validOut)];

    % ===============================================================
    % OBTAINING THE 'U' ARRAY FOR CLUSTER-OUT CENTROIDS
    % ===============================================================

    for i=1:kXPFOut
        FiltroOut(i).C(4:5)=(FiltroOut(i).C(1:2)-cenAntOut(FiltroOut(i).I,1:2))/2;
    end
end
```

"PDA2.m"

```
function [yInitAug,wPred]=PDA2(y,yInit,yPred,cY)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PDA.M - ASSOCIATION ALGORITHM
% Author    : Maria Cabello Aguilar
% Date      : June 2007
% Sintaxis  : PDA2(y,yInit,yPred,cY)
% Input     : - y: Containes the input coordinates of the measures
%             - yInit: Particles value in last iter
%             - yPred: Predicted state of each particle
%             - cY: Standard deviation (NO Variance) of the Gaussian measurement noise
% Output    : - yInitAug: Particles value in this iter
%             - wPred: nN-1 array with the particles' weight after the pred/corr step
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
global INTINF T;

nPartic=size(yPred,1);
ny=size(y,1);

%=========================================================================
%   CALCULATION OF DISTANCE
%=========================================================================

D=[];
for i=1:nPartic
    D(:,i)=sum((y(:,1:2)-repmat(yPred(i,1:2),ny,1)).^2,2);
end

%finds the shortest distance
[auxRuido,idxMeas]=min(D,[],1);

%=========================================================================
%   BETAS
%=========================================================================

% Calculando Gauss
if nPartic
    auxRuido=-0.5*auxRuido/(cY*cY);
    wPred=exp(auxRuido)/(cY*sqrt(2*pi));
end

% Normalise the weights
auxRuido=sum(wPred);
if auxRuido
    wPred=wPred'./auxRuido;
end

%=========================================================================
%   ASSIGNMENT
%=========================================================================

% Inserting measures that are false alarms
idxFalse=[1:ny];
idxFalse(idxMeas)=[];
nyFalse=length(idxFalse);

if ny
    yInitAug(1:nyFalse,:)=y(idxFalse,:);
    yInitAug(1:nyFalse,4:5)=zeros(nyFalse,2);
    % Inserting meas that are related with a particle
    yInitAug(nyFalse+1:nyFalse+nPartic,1:3)=y(idxMeas,:);
    yInitAug(nyFalse+1:nyFalse+nPartic,4:5)=(y(idxMeas,1:2)-yInit(:,1:2))/T;
else
    yInitAug=[];
end
```

"MIffun.m"

```
function xPred = MIffun(x,O,t)
% PURPOSE : Process model function
% AUTHORS : Marta Marron
% DATE    : 27 September 2003 / 12-12-06
% SINTAXIS: y=MIffun(x,O,t)
% INPUTS  : - x: The state vector at t N-nX
```

```
%           - O: The process noise vector at t N-nX
%           - t: Time
% OUTPUTS : - xPred: The state vector at t+1 N-nX
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

xPred=[];
A=[1 0 0 t 0;0 1 0 0 t;0 0 1 0 0;0 0 0 1 0;0 0 0 0 1];
if isempty(x)==0
    xPred=A*x';
    xPred=xPred'+O;
end
```

"MIhfun.m"

```
function y = MIhfun(x,R)
% PURPOSE : Measurement model function
% AUTHORS : Marta Marron
% DATE    : 27 September 2003 / 12-12-06
% SINTAXIS: y=MIhfun(x,R)
% INPUTS  : - x: The state vector at t N-nX
%           - R: The measurement noise vector at t N-nY
% OUTPUTS : - y: The measurement vector at t N-nY

y=[];
C=[1 0 0 0 0;0 1 0 0 0;0 0 1 0 0];
% C = [1 0 0 0; 0 0 1 0];
if isempty(x)==0
    y=C*x';
    y=y'+R;
end
```

"MiresidualR.m"

```
function idxOut = MIresidualR(idxIn,wn,nOut)
% PURPOSE : Performs the resampling stage of the SIR in order(number of samples) steps.
%           It uses Liu's residual resampling algorithm and Niclas' magic line
% AUTHORS : Arnaud Doucet and Nando de Freitas - Thanks for the acknowledgement
%             Marta Marron. Innovation to obtain different number of selected than the
   input size
% DATE    : 08-09-98 / 13-10-03 / 12-12-06
% SINTAXIS: idxOut=MIresidualR(idxIn,wn,nOut);
% INPUTS  : - idxIn = Input particle indices 1-nIn
%           - wn = Normalised importance ratios nIn-1
%           - nOut = number of output indexes
% OUTPUTS : - idxOut = Resampled indices 1-nOut
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ====================================
% N = Number of particles
% ====================================

N=size(wn,1);

% ====================================
% RESIDUAL RESAMPLING
% ====================================

% first integer part. Before it was wnRes=N.*wn';
```

```
nBabies=zeros(1,N);
wnRes=nOut*wn';

% residual number of particles to sample
nBabies=fix(wnRes);
nRes=nOut-sum(nBabies);
if nRes~=0 & isempty(wnRes)==0
    wnRes=(wnRes-nBabies)/nRes;
    distCum=cumsum(wnRes);

    % ====================================
    % generate nRes ordered random variables uniformly distributed in [0,1]
    % ====================================

    u=fliplr(cumprod(rand(1,nRes).^(1./(nRes:-1:1))));
    j=1;
    for i=1:nRes
        while u(i)>distCum(j)
            j=j+1;
        end
        nBabies(j)=nBabies(j)+1;
    end
end

% ====================================
% COPY RESAMPLED TRAJECTORIES
% ====================================

index=1;
idxOut=[];
for i=1:N
    if nBabies(i)>0
        for j=index:index+nBabies(i)-1
            idxOut(j)=idxIn(i);
        end
    end
    index=index+nBabies(i);
end
```

"Circle.m"

```
function [xcircle,ycircle] = circle(xc,yc,radius,npoints,color)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% To draw a circle in a cartesian 2D plane
% Author  : Marta Marron Romera
% Date    : 15-06-06
% Sintaxis: [xcircle,ycircle]=circle(xc,yc,radius,npoints,color);
% Inputs  : - xc,yc: cartesian coordinates of the circle's center
%           - radius: the radius of the circle
%           - npoints: the number of points of the circle
% Outputs : - xcircle,ycircle: 1-npoints arrays with the circle points
% Notes   : Probably there is something similar in Matlab but I do not find it
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

inc=radius/((npoints-2)/4);
xcircle1=[xc-radius:inc:xc+radius];
xcircle2=fliplr(xcircle1);
xcircle=[xcircle1 xcircle2];
```

```
ycircleAux=sqrt((radius.^2).*ones(size(xcircle1))-(xcircle1-
   xc*ones(size(xcircle1))).^2);
ycircle1=ycircleAux+(yc.*ones(size(ycircleAux)));
ycircle2=-1.*ycircleAux+(yc.*ones(size(ycircle1)));
ycircle=[ycircle1 ycircle2];
plot(xcircle,real(ycircle),color);
```

"randsample.m"

```
function y = randsample(n,k)
%RANDSAMPLE Random sampling, without replacement
%   Y = RANDSAMPLE(N,K) returns K values sampled at random, without
%   replacement, from the integers 1:N.
%
%   Copyright 1993-2002 The MathWorks, Inc.
%   $Revision: 1.1 $  $Date: 2002/03/13 23:15:54 $
%
% RANDSAMPLE does not (yet) implement weighted sampling.

if nargin < 2
    error('Requires two input arguments.');
end

% If the sample is a sizeable fraction of the population, just
% randomize the whole population (which involves a full sort
% of n random values), and take the first k.
if 4*k > n
    rp = randperm(n);
    y = rp(1:k);

% If the sample is a small fraction of the population, a full
% sort is wasteful.  Repeatedly sample with replacement until
% there are k unique values.
else
    x = zeros(1,n);                         % flags
    sumx = 0;
    while sumx < k
        x(ceil(n * rand(1,k-sumx))) = 1;    % sample w/replacement
        sumx = sum(x);                      % count how many unique elements so far
    end
    y = find(x > 0);
    y = y(randperm(k));
end
```

"ploteaTray.m"

```
% Only to use if .mat file is loaded
strColor=['r.';'g.';'m.';'c.';'k.';'ro';'go';'mo';'co';'ko';'r<';'g<';'m<';'c<';'k<'];
iterTotal=size(kTotal,1);
Cx=[];
Cz=[];
fila=1;
maxK=0;

for i=1:iterTotal
    k=kTotal(i,1);
    if k>maxK
        maxK=k;
```

```
    end

    % Saving values
    Cx(i-init+1,ITotal(fila:fila+k-1))=CTotal(fila:fila+k-1,1)';
    Cz(i-init+1,ITotal(fila:fila+k-1))=CTotal(fila:fila+k-1,2)';
    fila=fila+k;
end;

% Plotting the results
figure(1);
hold on; grid;
for i=1:maxK
    plot(Cx(:,i),Cz(:,i),strColor(i,:),'LineWidth',2);
end
```

VI. PRESUPUESTO

En esta sección se estima el importe de la ejecución del proyecto. Para ello se realiza un estudio agrupando los gastos en función de su origen.

1. Ejecución material

El presupuesto de ejecución material puede desglosarse en tres elementos:

- Coste de equipos.
- Coste del material utilizado.
- Coste de mano de obre por tiempo empleado.

Coste de equipos

Los distintos equipos que han sido utilizados en este trabajo son los siguientes:

EQUIPO	PRECIO
Ordenador Portátil	999 €
Impresoras	300 €
Total	**1.299 €**

Coste material

MATERIAL	PRECIO
Microsoft Windows XP profesional	135 €
Microsoft Office XP	200 €
Matlab 6.5	2.500 €
Material de oficina	150 €
Total	**2.985 €**

Coste por tiempo empleado

FUNCIÓN	Nº HORAS	€/HORA	TOTAL
Ingeniería	960	60 €	57.600 €
Mecanografiado	100	12 €	1.200€
		Total	**58.800€**

Coste total del presupuesto de ejecución material

PARTIDA	PRECIO
Coste de equipos	1.299 €
Coste Material	2.985 €
Coste por tiempo Empleado	58.800
Total	**63.084 €**

2. Gastos generales y beneficio industrial

Se incluyen los gastos generados por las instalaciones donde se ha realizado el proyecto más el beneficio industrial. Para estos conceptos se estima el 25 % del presupuesto de ejecución del proyecto.

El valor de los gastos generales y beneficio industrial asciende a **15.771 €.**

3. Presupuesto de ejecución por contrata

Este concepto incluye el presupuesto de ejecución material, los gastos generales y el beneficio industrial.

Presupuesto de ejecución material	63.084 €
Gastos generales y beneficio industrial	15.771 €
Total	**78855 €**

4. Horarios de redacción

Se ha calculado como el 7 % del presupuesto de ejecución material multiplicado por 0.9.

Presupuesto de ejecución material	**63.084 €**
Honorarios por redacción	**3.974 €**

5. Importe total del presupuesto

Presupuesto de ejecución por contrata	78.855 €
Honorarios por redacción	3.974 €
Subtotal	82.829 €
Total: Subtotal + 16% IVA	**96.081 €**

El importe total asciende a: **Noventa y seis mil ochenta y uno euros.**

Alcalá de Henares a 1 de Octubre de 2007.

Firmado: María Cabello Aguilar.

Ingeniero Técnico Industrial.

VII. BIBLIOGRAFÍA

[Almeida05] A. Almeida, J. Almeida, R. Araújo. “Real-Time of Moving Objects Using Particle Filters”, Institute for Systems and Robotics. Department of Electrical and Computer Engineering, Univserity of Coimbra, Coimbra, Portugal, 2005.

[Alsabti98] K. Alsabti, S. Ranka, V. Singh. "An efficient k-means clustering algorithm", Proceedings of the First Workshop on High Performance Data Mining at the International Parallel Processing Symposium (IPPS98), Orlando, 1998.

[Broddfelt05] J. Broddfelt. “Tracking multiple objects with Kalman filters”. Proyecto Final de Carrera, Universidad de Alcalá, Febrero 2005.

[Cerro07] J. Cerro Jiménez. “Comparativa teórica y empírica de métodos de clasificación aplicados a medidas tridimensionales de visión”. Proyecto Final de Carrera, Universidad de Alcalá, Julio 2007.

[Garcia01] J. García de Jalón, J.I. Rodríguez, A. Brazales. “Aprenda Matlab como si estuviese en primero”, Escuela Técnica Superior de Ingenieros Industriales, Universidad Politécnica de Madrid, 2001.

[Gordon93] N.J. Gordon, D.J Salmond, A.F.M. Smith. “Novel approach to nonlinear/non-gaussian bayesian state estimation”, IEE Proceedings Part F, Vol. 140, nº 2, pp: 107-113, Abril 1993.

[Kanungo02] T. Kanungo, N.S. Netanyahu, A.Y. Wu. “An efficient k-means clustering algorithm: analysis and implementation", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol 24, nº 7, July 2002.

[Karlsson02] R. Karlsson, “Simulation based Methods for Target Tracking”. Linköping Studies in Science and Technology, thesis no. 930. Division of Automatic Control & Communication systems, department of Electrical Engineering, Linköping universitet, Linköping, Sweeden, 2002. [Lesaux02] B. LeSaux and N. Boujemaa. "Unsupervised robust clustering for image database categorization", Proceedings of the Sixteenth International Conference on Pattern Recognition (ICPR02), Vol. 1, pp. 10259, ISBN: 0-7695-1695-X, Quebec, August 2002.

[MacCormick99] J. MacCormick, A, Blake. “A probabilistic exclusion principle for tracking multiple objects”, in Proc. of the Seventh IEEE International Conference on Computer Vision (ICCV99), vol. 1, pp. 572-578, Corfu, Septiembrer 1999.

[Marrón05] M. Marrón, M.A. Sotelo, J.C. García, D. Fernandez, D. Pizarro. "XPFCP: An extended particle filter for tracking multiple and dynamic objects in complex environments", Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS05), ISBN: 0-7803-9252-3, pp. 234-239, Edmonton, Agosto 2005.

[Marrón07] M. Marrón, M.A. Sotelo, J.C. García, J. Broddfelt. “Comparing improved versions of ‘K-Means’ and ‘Subtractive’ clustering in a tracking applications”, Proc. of the Eleventh International Workshop on Computer Aided Systems Theory, Extended Abstracts (EUROCAST07), ISBN: 978-84-690-3603-7, pp. 252-255, Las Palmas de Gran Canaria, Febrero 2007. [Maybeck79] P.S. Maybeck. “Stochastic models, estimation and control”, Academic Press, ISBN: 0-12-480701-1, Vol. 1, New York, 1979.

[Pao94] L. Pao. “Multisensor mutitarget mixture reduction algorithms for tracking”, AIAA Journal of Guidance, Control and Dynamics, Vol 17, No 6, 1207-1211, 1994.

[Rasmussen01] C. Rasmussen, G. D. Hager. “Probabilistic Data Association Methods for Tracking Complex Visual Objects”, in Proc. of the IEEE Transactions on Pattern Analysis and Machine Intelligence (IEEECS), vol. 23, no. 6, Junio 2001.

[Schulz01] D. Schulz, W. Burgard, D. Fox, A.B. Cremers. "Tracking multiple moving objects with a mobile robot", Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR01), ISBN: 0-7695-1272-0, Vol. 1, pp. 371-377, Hawaii, December 2001.

[Schulz03] D. Schulz, W. Burgard, D. Fox, A. B. Cremers. “People tracking with mobile robots using sample-based joint probabilistic data association filters”, International Journal of Robotics Research, Vol. 22, nº 2, pp: 99-116, Febrero 2003. [Smith05] K. Smith, D. Gatica-Perez, J.M. Odobez. “Using particles to track varying numbers of interacting people”, in Proc. of the Fourth IEEE Conference on Computer Vision and Pattern Recognition (CVPR05), pp. 962-969, San Diego, Junio 2005.

[Welch01] G. Welch, G. Bishop. “An introduction to the Kalman filter”, Course 8 Siggraph 2001, University of North Carolina, Chapel Hill, 2001.

[Wilson05] D.H. Wilson, C. Atkeson. “Simultaneous tracking & activity recognition (STAR) using many anonymous, binary sensors”, Proceedings of the Third International Conference on Pervasive Computing (PERVASIVE05), Lecture Notes in Computer Science, ISBN: 3-540-26008-0, Vol. 3468, pp. 62-79, Munich, May 2005.