Sistemas de localización avanzados en entornos interiores basados en tecnología UWB

Autor: Miguel Eguizábal Alonso

Director: Juan Chóliz Muniesa

Ponente: Ángela Hernández Solana

Curso 2009/2010

1 de Septiembre de 2010

ANEXOS A, B y C

Ingeniería Superior de Telecomunicación

Departamento de Ingeniería Electrónica y Comunicaciones

Centro Politécnico Superior

Universidad de Zaragoza

ANEXO A: Descripción de los algoritmos de localización

A.1 Trilateración

La trilateración es un método matemático para determinar las posiciones relativas de objetos usando la geometría de los triángulos y de las esferas. La trilateración usa las localizaciones conocidas de dos o más puntos de referencia, y la distancia medida entre el sujeto y cada punto de referencia. Para determinar de forma única y precisa la localización relativa de un punto en un plano bidimensional usando sólo trilateración, se necesitan al menos 3 puntos de referencia.

Como hemos comentado, para estimar la posición del target, necesitamos la medida de la distancia de tres anchors diferentes. Sí el número de medidas disponibles es inferior a tres, no podemos estimar la posición. Sí por el contrario, el número de medidas disponibles es superior a tres, ordenaremos las medidas de tal forma, que nos quedaremos con las medidas de los tres anchors más cercanos, es decir, con las tres medidas de distancia más pequeñas.

Las ecuaciones se van a formular teniendo en cuenta que una de las circunferencias tiene el centro en el origen y otra de las circunferencias tiene el centro en el eje x. Esta situación no es realista, pero se puede construir un nuevo eje de coordenadas en el cual el origen esté en el centro de la circunferencia 1 y el eje x sea la recta que une los centros C1 y C2.

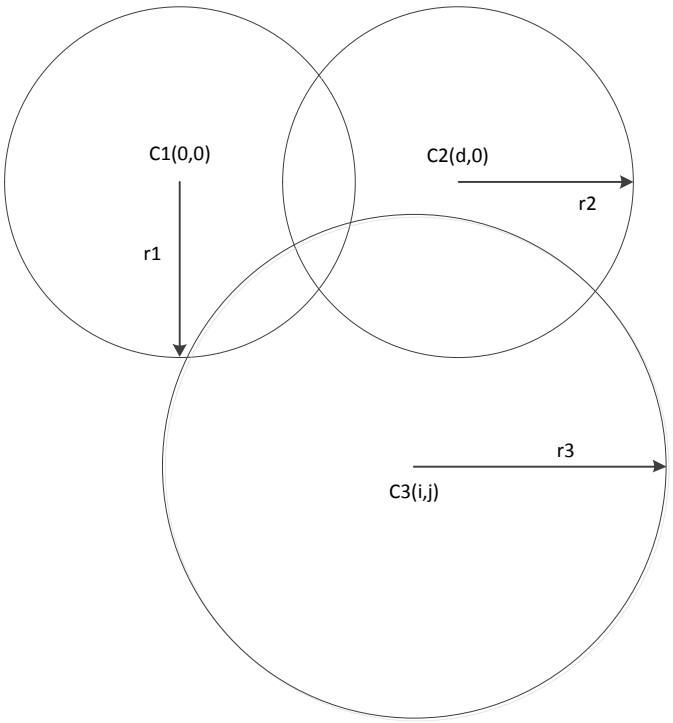

Figura A1. Esquema trilateración

Hay que comenzar formulando las ecuaciones de las 3 circunferencias:

$$r_1^2 = x^2 + y^2$$

$$r_2^2 = (x-d)^2 + y^2$$

$$r_3^2 = (x-i)^2 + (y-j)^2$$

Hay que encontrar el punto (x, y) que satisfaga las tres ecuaciones. En primer lugar, se le resta la segunda ecuación a la primera y se resuelve para x:

$$x = \frac{r_1^2 - r_2^2 + d^2}{2d}$$

Sustituyendo $y^2 = r_1^2 - x^2$ en la ecuación de la tercera circunferencia y resolviendo para y se obtiene:

$$y = \frac{r_1^2 - r_3^2 + i^2 + j^2}{2j} - \frac{i}{j}x$$

Con las dos ecuaciones se obtienen las coordenadas x e y.

A.2 Filtro de Kalman

El filtro de Kalman se dedica al problema de intentar estimar el estado $x \in \mathbb{R}^n$ de un proceso discreto en tiempo, gobernado por una ecuación diferencial lineal estocástica. Si la ecuación diferencial no es lineal, se hace referencia al filtro de Kalman Extendido (EKF), que linealiza la ecuación sobre la media actual y covarianza. En el simulador está implementado el filtro de Kalman Extendido.

La ecuación diferencial no lineal con la que trabaja el filtro es:

$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1})$$

Y la medida $z \in \mathbb{R}^m$ es:

$$z_k = h(x_k, v_k)$$

w_k y v_k son variables aleatorias que representan el ruido del proceso y de medida respectivamente. Se consideran independientes, de media cero y con una distribución de probabilidad normal: $p(w) \sim N(0, Q)$ y $p(v) \sim N(0, R)$. Las matrices de covarianza Q y R podrían cambiar en cada iteración. La función no lineal f relaciona el estado en el paso anterior k-1 con el estado en el paso actual k. Incluye como parámetros u_{k-1}, que es una entrada de control opcional, y el ruido del proceso. La función no lineal h relaciona el estado x_k con la medida z_k e incluye como parámetro el ruido de medida.

En la práctica no se conocen los valores individuales del ruido w_k y v_k en cada paso, sin embargo se puede aproximar el estado y la medida sin ellos, de la siguiente manera:

$$\tilde{x}_k = f(\hat{x}_{k-1}, u_{k-1}, 0)$$

$$\tilde{z}_k = h(\tilde{x}_k, 0)$$

Donde $\hat{x}_k$ es la estimación a posteriori del estado. El filtro hace una estimación a priori del estado, que es $\tilde{x}_k$, con el conocimiento del proceso del paso anterior a k. Y posteriormente hace una estimación a posteriori del estado con la medida z_k.

En el simulador el vector de estado incluye la posición p_k, y la velocidad v_k del target. Las ecuaciones de medida utilizadas son de distancias relativas a puntos de posiciones conocidas(anchors) p_i , $i = 1,...,M$. Así que se dispone de M ecuaciones de medida, donde M será el número de anchors utilizados. Las estimaciones del vector de estado y de la medida son:

$$\underbrace{\begin{pmatrix} \tilde{p}_k \\ \tilde{v}_k \end{pmatrix}}_{\tilde{x}_k} = \begin{pmatrix} I & T_s \cdot I \\ 0 & I \end{pmatrix} \underbrace{\begin{pmatrix} \hat{p}_{k-1} \\ \hat{v}_{k-1} \end{pmatrix}}_{\hat{x}_{k-1}}$$

$$\tilde{z}_k(i) = \mid p_i - \tilde{p}_k \mid$$

Para estimar el proceso hay que comenzar escribiendo las ecuaciones linealizadas:

$$x_k \approx \tilde{x}_k + A(x_{k-1} - \hat{x}_{k-1}) + Ww_{k-1}$$

$$z_k \approx \tilde{z}_k + H(x_k - \tilde{x}_k) + Vv_k$$

donde:

x_k y z_k son los vectores actuales del estado y de la medida

$\tilde{x}_k$ y $\tilde{z}_k$ son las estimaciones a priori de los vectores del estado y de la medida

$\hat{x}_k$ es la estimación a posteriori del vector de estado

w_k y v_k son las variables aleatorias que representan los ruidos de proceso y de medida respectivamente

A es la matriz Jacobiana de derivadas parciales de f respecto a x:

$$A_{[i,j]} = \frac{\partial f_{[i]}}{\partial x_{[j]}}(\hat{x}_{k-1}, u_{k-1}, 0)$$

W es la matriz Jacobiana de derivadas parciales de f respecto a w:

$$W_{[i,j]} = \frac{\partial f_{[i]}}{\partial w_{[j]}}(\hat{x}_{k-1}, u_{k-1}, 0)$$

H es la matriz Jacobiana de derivadas parciales de h respecto a x:

$$H_{[i,j]} = \frac{\partial h_{[i]}}{\partial x_{[j]}}(\tilde{x}_k, 0)$$

V es la matriz Jacobiana de derivadas parciales de h respecto a v:

$$V_{[i,j]} = \frac{\partial h_{[i]}}{\partial v_{[j]}}(\tilde{x}_k, 0)$$

Hay que tener en cuenta que aunque en la notación de las Jacobianas no se utilice el subíndice k, las matrices son diferentes en cada iteración del filtro de Kalman.

El error de predicción y el residuo de medida se definen de la siguiente forma:

$$\tilde{e}_{x_k} \equiv x_k - \tilde{x}_k$$

$$\tilde{e}_{z_k} \equiv z_k - \tilde{z}_k$$

En la práctica no se tiene acceso a x_k, ya que es el vector del estado actual. Sin embargo sí que se tiene acceso a z_k, que es el vector de medida actual y se puede utilizar para estimar x_k. Teniendo en cuenta esto, se puede reescribir el error de predicción y el residuo de medida:

$$\tilde{e}_{x_k} \approx A(x_{k-1} - \hat{x}_{k-1}) + \varepsilon_k$$

$$\tilde{e}_{z_k} \approx H\tilde{e}_{x_k} + \eta_k$$

ε_k y η_k representan unas nuevas variables aleatorias independientes de media cero y matrices de covarianza WQW^T y VRV^T respectivamente. Se puede pensar en utilizar el residuo de medida $\tilde{e}_{z_k}$ en un hipotético segundo filtro de Kalman para estimar el error de predicción $\tilde{e}_{x_k}$. Esta estimación $\hat{e}_k$ puede utilizarse para obtener la estimación a posteriori del vector de estado:

$$\hat{x}_k = \tilde{x}_k + \hat{e}_k$$

La ecuación del filtro de Kalman utilizado para estimar $\hat{e}_k$ es:

$$\hat{e}_k = K_k \tilde{e}_{z_k}$$

Si expresamos esta ecuación de otro modo, puede verse que realmente no es necesario ese hipotético segundo filtro de Kalman:

$$\hat{x}_k = \tilde{x}_k + K_k \tilde{e}_{z_k} = \tilde{x}_k + K_k (z_k - \tilde{z}_k)$$

La matriz K_k es la ganancia del filtro de Kalman y se calcula para minimizar la covarianza del error de la estimación a posteriori. El conjunto completo de ecuaciones del EKF, que engloban el proceso de estimación y el proceso de corrección, son las siguientes:

- Proceso de estimación

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_{k-1}, 0)$$

$$P_k^- = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T$$

- Proceso de corrección

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1}$$

$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - h(\hat{x}_k^-, 0))$$

$$P_k = (I - K_k H_k) P_k^-$$

$\hat{x}_k^-$ y P_k^- son estimaciones a priori. P_k es la matriz de covarianza del error de estimación:

$$e_k^- \equiv x_k - \hat{x}_k^-$$

$$e_k \equiv x_k - \hat{x}_k$$

$$P_k^- = E[e_k^- e_k^{-T}]$$

$$P_k = E[e_k e_k^T]$$

Con las ecuaciones del proceso de corrección, corregimos las estimaciones a priori del estado y de la covarianza con la medida z_k. Por lo tanto el modo de funcionamiento del EKF es el mostrado en la figura A2.

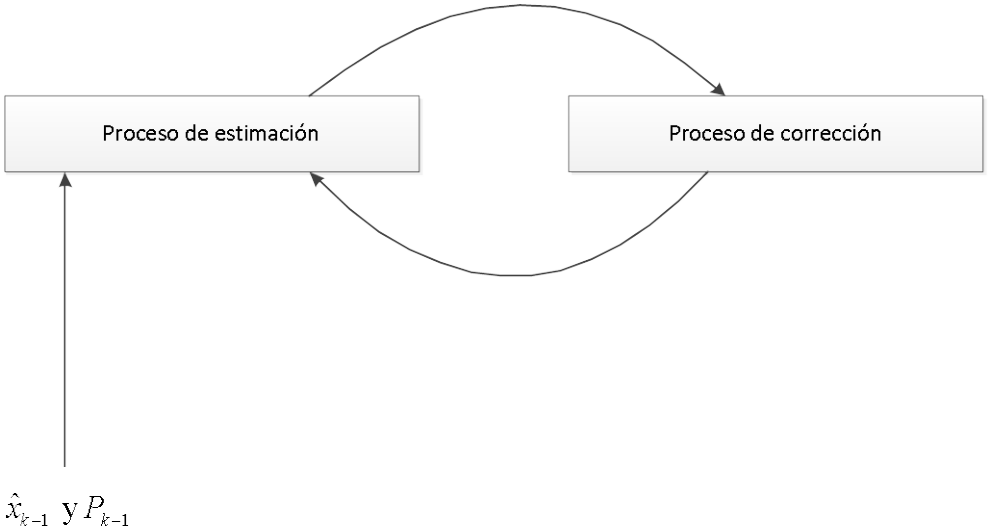

Figura A2. Procedimiento filtro de Kalman

A.3 Filtro de partículas

El filtro de partículas también es conocido como método Monte Carlo secuencial (SMC). Este filtro se utiliza para estimar modelos Bayesianos recursivos. El filtro de partículas se presenta como una alternativa al filtro de Kalman para aplicaciones en tiempo real. En las situaciones donde el modelo no es lineal o el ruido no es Gaussiano, es donde el filtro de partículas tiene más potencial.

Se consideran sistemas descritos por el modelo genérico de estado:

$$x_{t+1} = Ax_t + B_u u_t + B_f f_t$$

$$y_t = h(x_t) + e_t$$

donde:

x_t es el vector de estado

u_t son las entradas medidas

f_t son las fuerzas no medidas o errores

y_t son las medidas

e_t es el error de medida

Se asumen distribuciones independientes para f_t, e_t y x_0, con densidades de probabilidad conocidas p_{f_t}, p_{e_t} y p_{x_0} respectivamente.

El modelo de movimiento que se asume es:

$$\begin{pmatrix} p_{t+1} \\ v_{t+1} \end{pmatrix} = \underbrace{\begin{pmatrix} I & T_s \cdot I \\ 0 & I \end{pmatrix}}_{A} \begin{pmatrix} p_t \\ v_t \end{pmatrix} + \underbrace{\begin{pmatrix} T_s^2 / 2 \cdot I \\ T_s \cdot I \end{pmatrix}}_{B_f} f_t$$

Con este modelo obtenemos las relaciones $p_t = p_{t-1} + v_{t-1}T_s + a_{t-1}T_s^2 / 2$ y $v_t = v_{t-1} + a_{t-1}T_s$. La aceleración se modela como una variable aleatoria con distribución normal y de media cero.

Las ecuaciones de medida utilizadas son de distancias relativas a puntos de posiciones conocidas(anchors) p_j , $j = 1,...,M$. Así que se dispone de M ecuaciones de medida, donde M será el número de anchors utilizados:

$$h_j(p_t) = \| p_j - p_t \|$$

El algoritmo del filtro de partículas consta de los siguientes 5 pasos:

1. Inicialización

Se genera $x_0^i \sim p_{x_0}$, $i = 1,...,N$. Cada muestra del vector de estado se denomina partícula.

2. Actualización de la medida

Se actualizan los pesos con la probabilidad: $w_t^i = w_{t-1}^i p(y_t \mid x_t^i)$, $i = 1,...,N$. Donde y_t es la observación disponible en el instante t. Se dispone de varios modelos para estimar las probabilidades $p(y_t \mid x_t^i)$. Y se normalizan los pesos $w_t^i = w_t^i / \sum_i w_t^i$. Se realiza la estimación de la posición mediante la siguiente aproximación:

$$\hat{x}_t \approx \sum_{i=1}^{N} w_t^i x_t^i$$

3. Remuestreo

Se quitan N muestras con reemplazo del conjunto $\{x_t^i\}_{i=1}^N$, donde la probabilidad de quitar la muestra i es w_t^i. Luego se da valor a los pesos: $w_t^i = 1/N$. Sólo se remuestrea cuando el número efectivo de muestras es menor que un umbral:

$$N_{eff} = \frac{1}{\sum_i (w_t^i)^2} < N_{th}$$

Aquí, $1 \le N_{eff} \le N$, donde el límite superior se obtiene si todas las partículas tienen el mismo peso($1/N$) y el límite inferior cuando una partícula concentra toda la probabilidad. A

menudo se escoge como umbral $N_{th} = 2N/3$. El remuestreo se realiza para evitar que unas pocas partículas concentren toda la probabilidad.

4. Predicción

   Actualizamos las partículas:

$$x_{t+1}^{i} = Ax_t^i + B_f f_t^i,\ i = 1,...,N$$

5. Hacer $t = t+1$ e ir al paso 2.

A.4 MDS (MultiDimensional Scaling)

MDS es un método que proporciona una representación espacial de los datos. Dada una matriz que contenga todas las semejanzas-desemejanzas ∂_{ij} entre N puntos, MDS mapea esa configuración en un espacio tal que la diferencia entre ∂_{ij} y las distancias relativas d_{ij} es mínima. Se llama X a la matriz $[N \times \mu]$ que contiene las coordenadas de N objetos en un espacio de μ dimensiones y se escoge el conjunto $\{\partial_{ij}\}$ como el conjunto de distancias euclídeas entre dos objetos i y j del escenario $\{d_{ij}\}$, donde $d_{ij}^2 = (x_i - x_j)^T \cdot (x_i - x_j)$.

De esta forma el problema MDS puede resolverse algebraicamente como la solución de mínimos cuadrados de la matriz $B = X \cdot X^T$. La solución clásica es construir la siguiente matriz de Gram:

$$G = -\frac{1}{2} \cdot J \cdot D^{\circ 2} \cdot J$$

donde $J = I_N - 1_N 1_N^T / N$ es la "centering matrix", que es una matriz simétrica e idempotente, la cual cuando se multiplica por un vector tiene el mismo efecto que restarle la media de los componentes del vector a cada componente, con 1_N el vector unitario $[N \times 1]$, I_N la matriz identidad $[N \times N]$ y $^{\circ}$ indica el producto elemento por elemento. D es la matriz de distancias euclídeas($[D]_{ij} = d_{ij}$).

La matriz G es equivalente a B (para X centrada en el origen). Por lo tanto, la técnica MDS permite recuperar la localización Y de todos los nodos de la red a partir de la matriz de distancias euclídeas:

$$Y = [V]_{1:\mu} \cdot [\Lambda]_{1:\mu}^{1/2}$$

$$G = V \cdot \Lambda \cdot V^T$$

donde V contiene en sus columnas los vectores propios de G y Λ es una matriz diagonal cuyos elementos de la diagonal son los valores propios de G. Por lo tanto, para poder calcular

la matriz Y se debe obtener la matriz de Gram G y después aplicarle la descomposición en valores propios.

La configuración de los nodos que nos devuelve el algoritmo MDS, está mapeada en un sistema de coordenadas diferente al original. Para trasladar las localizaciones Y y obtener las localizaciones reales X , es necesario aplicar una transformación a Y llamada Procrustes. De tal forma que conociendo las coordenadas X_A de al menos $A > \mu$ anchors, la solución Y puede reorientarse mediante esta transformación, obteniendo X . La transformación Procrustes es una transformación geométrica lineal involucrando únicamente traslación, reflexión, rotación ortogonal y escalado.

A.5 Distance Contraction

Como se ha comentado anteriormente al explicar el modelo de ranging, la distancia estimada está sesgada debido al error introducido por el canal (multicamino, NLOS) que es siempre positivo. Una posible estrategia consiste en "contraer" las distancias estimadas para tratar de mitigar el sesgo de la estimación. Mediante el algoritmo de distance contraction se contraen las distancias estimadas, que posteriormente se utilizarán en un algoritmo de optimización como por ejemplo SMACOF para hallar la solución óptima.

Se dispone del conjunto de distancias medidas del target a cada anchor i: $\left\{\tilde{d}_i\right\}$. El primer paso del algoritmo es verificar la existencia de la región de viabilidad (feasibility region). Esta región de viabilidad se define como:

$$I \triangleq \left\{\hat{x} \mid \hat{d}_i \le \tilde{d}_i \forall i\right\}$$

donde $\hat{d}_i$ es la distancia del punto $\hat{x}$ a cada anchor i. Si la región de viabilidad no existe, no se puede aplicar el algoritmo de contracción de distancias.

Una vez verificada la existencia de la región de viabilidad, se pueden calcular las distancias contraídas $\overline{d}_i$. Para ello se evalúa la siguiente expresión para cada uno de los anchors:

$$\overline{x}_i = \arg\max_{\hat{x}\in I} (\tilde{d}_i - \hat{d})^2$$

Para cada anchor obtenemos el punto $\overline{x}_i$, que es el punto de tangencia entre la región de viabilidad y la circunferencia con centro la posición del anchor y con radio $\overline{d}_i$, que es la distancia del anchor al punto $\overline{x}_i$. Es necesario conseguir un punto inicial $\hat{x}_0$ dentro de la región, para que después el algoritmo sea capaz de encontrar el punto que maximiza la expresión anterior. Para calcular el punto inicial se utiliza la siguiente expresión:

$$\hat{x}_0 = \arg\min_{\hat{x}\in\mathbb{R}^n} \sum_{i=1}^{N_A} \max\left(0, \hat{d}_i - \tilde{d}_i\right)^2$$

donde N_A es el número de anchors utilizados en la medida.

En la figura A3 se muestra un ejemplo gráfico de este algoritmo. La región de viabilidad es la región remarcada con línea negra.

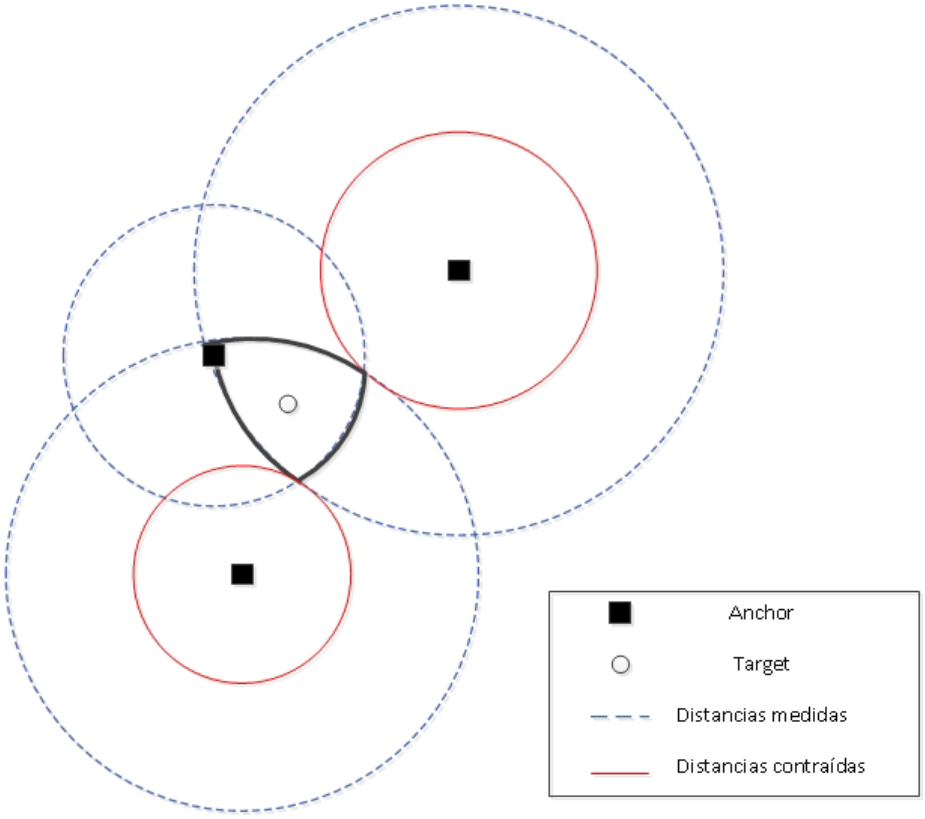

Figura A3. Distance Contraction

Por lo tanto los pasos que se llevan a cabo en este algoritmo son los siguientes:

1) Verificar la existencia de la región de viabilidad

$$I \triangleq \left\{ \hat{x} \mid \hat{d}_i \le \tilde{d}_i \forall i \right\}$$

2) Calcular las distancias contraídas $\overline{d}_i$

$$\overline{x}_i = \arg\max_{\hat{x} \in I} (\tilde{d}_i - \hat{d})^2$$

donde

$$\hat{x}_0 = \arg\min_{\hat{x} \in \mathbb{R}^n} \sum_{i=1}^{N_A} \max\left(0, \hat{d}_i - \tilde{d}_i\right)^2$$

3) Sustituir las distancias medidas $\tilde{d}_i$ por las distancias contraídas $\overline{d}_i$

4) Llevar a cabo un algoritmo de optimización, como por ejemplo SMACOF, para obtener la solución óptima

A.6 SMACOF

El algoritmo SMACOF sirve para optimizar la matriz de coordenadas de los anchors y del target, X de dimensiones $[n \times \mu]$, a partir de una solución inicial obtenida mediante otro algoritmo como MDS o distance contraction, de forma que la matriz de distancias derivada de la matriz X sea lo más parecida a la matriz de distancias medidas. Se quiere encontrar la matriz X, tal que $d_{ij}(X) \approx \partial_{ij}$, donde:

$$d_{ij}(X) = \sqrt{\sum_{s=1}^{\mu} (x_{is} - x_{js})^2}$$

El índice $s = 1, ..., \mu$ indica el número de dimensiones del espacio. Se define la función *stress* $\sigma(X)$ de la siguiente forma:

$$\sigma(X) = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} (\partial_{ij} - d_{ij}(X))^2$$

La matriz W es la matriz $n \times n$ de pesos w_{ij}, para ponderar las estimaciones en base a su variabilidad. Se asume sin pérdida de generalidad que $\sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \partial_{ij}^2 = 1$. Para minimizar la función de *stress* se deriva e iguala a cero, obteniendo la configuración X que la minimiza:

$$X = V^{\dagger} B(Y) Y$$

donde los elementos de la matriz $B(Y)$ son $b_{ij} = \begin{cases} -w_{ij} \delta_{ij} d_{ij}^{-1}(Y) & i \neq j \text{ y } d_{ij}(Y) \neq 0 \\ 0 & i \neq j \text{ y } d_{ij}(Y) = 0 \\ -\sum_{k \neq i} b_{ik} & i = j \end{cases}$ y $V^{\dagger}$ es la matriz pseudo-inversa “Moore-Penrose”.

En el simulador se implementa como un procedimiento iterativo, en el paso $t = 0$, se ajusta $Y = X^{(0)}$, donde $X^{(0)}$ es la configuración inicial. En cada iteración se calcula $X^{(t)}$ y después se obtiene $\sigma(X^{(t)})$ y se para de iterar si se cumple la condición: $\sigma(X^{(t)}) - \sigma(X^{(t-1)}) < \varepsilon$ o si se supera un determinado límite de iteraciones.

ANEXO B: Descripción del simulador

B.1 Introducción

El principal objetivo del simulador es evaluar las prestaciones que un sistema de localización UWB real puede proporcionar en entornos interiores. De cara a que esta evaluación sea realista, se deben utilizar las especificaciones tanto a nivel físico como de acceso a la red (topología de la red, estructura de la supertrama, duración de los slots) de un sistema UWB real, en concreto del diseñado dentro del propio proyecto EUWB, además de un modelo de estimación de distancias en interiores.

B.2 Escenario

El escenario que se plantea es el de seguimiento de dispositivos móviles en entornos interiores relativamente amplios tales como centros comerciales, aeropuertos, estaciones, centros de congresos, etc. Se tratan de escenarios relativamente grandes por lo que se deberán cubrir por varias estaciones fijas. La gestión de recursos de la red es crítica ya que serán usados por un elevado número de usuarios todos ellos con movilidad. Este escenario representado en el simulador tiene unas dimensiones *longitud* x *anchura* cuyo valor será seleccionado en el fichero de entrada del simulador.

Sobre ese escenario se despliega una picocelda UWB con un conjunto de nodos fijos dispuestos en cuadrícula separados por una distancia *d*. Por simplicidad se considera una única picocelda, con lo que no será necesario realizar handover intercelular. La picocelda dispone de N_a anchors, elementos fijos y de posición conocida, y N_m targets, elementos móviles a localizar.

Uno de los elementos fijos será el coordinador de la picocelda, que marca la sincronización de la supertrama y gestiona el acceso de los nodos a los recursos. El resto de nodos serán “esclavos” y pueden tener conexión directa con el coordinador (si están en cobertura) o a través de otros nodos hasta un número máximo de saltos N_{hops}.

En la picocelda se definen uno o más controladores de localización (LC), que realizan la función del cálculo de la posición de los elementos a localizar. Según la estrategia que se defina, esta función puede localizarse físicamente en uno o más nodos fijos (estrategia centralizada en la red), en los nodos móviles (estrategia centralizada en el móvil) o ser asumida dinámicamente por cualquier nodo de la red (estrategia distribuida).

B.3 Modelos implementados

Antes de analizar el simulador en detalle, se describen algunos de los modelos implementados.

B.3.1 Topología de la red

La topología de la picocelda es mallada-centralizada. El coordinador de la picocelda transmite tramas piloto (beacon frames) para la sincronización. Después se construye el árbol de distribución y se utiliza para transportar las tramas piloto y los comandos, que son enviados

desde el coordinador a cualquier nodo de la picocelula. Finalmente, la topología se extiende a un árbol mallado (meshed tree), permitiendo la transmisión fuera del árbol de tramas de datos y ranging, así como de las tramas de 'hello' que son enviadas periódicamente a todos los vecinos.

B.3.2 Estructura de la supertrama MAC

La estructura de la supertrama define la temporización de slots utilizados para cada propósito (descubrimiento de vecinos, comunicación, ranging...). La supertrama MAC se divide en slots temporales (timeslots) en los que se envían las distintas tramas (tramas piloto, tramas hello, tramas de datos, tramas de ranging, solicitudes de slots...).

La supertrama MAC se divide en los siguientes periodos:

- Beacon period, utilizado para alineamiento.
- Topology management period, usados para mandar las tramas hello.
- CFP (Contention Free Period), compuesto por slots GTS (Guaranteed Time Slots), que pueden utilizarse tanto para transmisión de datos como para ranging y un GTS Request Period, usado para solicitar la asignación de slots GTS.
- CAP (Contention Access Period), que se utilizaría para enviar comandos desde los nodos al coordinador de la celda.

B.3.3 Arquitecturas del sistema de localización

El simulador tiene implementadas varias arquitecturas del sistema de localización, que serán seleccionadas desde el fichero de entrada.

• **Centralizada en la red**. Uno o varios nodos fijos de la red actúan como controladores de localización. Esos nodos están prefijados.

• **Distribuida**. Cualquier nodo anchor puede asumir el papel de controlador de localización para un target dado y calcular el dato de posicionamiento. El anchor lo elige de forma dinámica el móvil.

• **Centralizada en el móvil**. La función de controlador de localización está implementada en los propios nodos target, por lo que es el propio móvil el que calcula su posición.

B.3.4 Modelo dinámico de los targets

Para simular el movimiento de los targets se define una velocidad máxima y mínima en el fichero de entrada. En el modelo estadístico, el simulador elige una velocidad aleatoria en ese rango, además de una dirección aleatoria. Pasado un intervalo de tiempo también definido en el fichero de entrada, se fijan una nueva dirección y velocidad aleatorias. En el modelo basado en paredes y rutas, el simulador elige aleatoriamente un nodo destino, con lo que la dirección queda fijada y escoge una velocidad aleatoria en el rango comprendido entre la velocidad máxima y mínima. Cuando llega al nodo destino, este nodo pasa a ser el nodo inicial y escoge nuevamente un nodo destino y una velocidad.

B.3.5 Modelo de ranging

Este modelo determina la distribución del error en la estimación de las distancias y se compone de un determinado sesgo y de un determinado nivel de error residual, que está relacionado con la resolución en la estimación del TOA y es independiente de la distancia. La distribución está definida en el simulador en el fichero ranging_model, salvo el error residual que se añade posteriormente y cuya varianza se puede seleccionar en el fichero de parámetros. Para el caso del modelo estadístico el sesgo es función de la distancia real y de la probabilidad de tener un determinado canal LOS/NLOS/NLOS2 (que a su vez también es función de la distancia). Para el modelo basado en paredes y rutas el sesgo es función de la distancia real y del tipo de canal(LOS/NLOS/NLOS2), que se determina comprobando el número de paredes que atraviesa el enlace.

B.3.6 Modelo de cobertura

El modelo de cobertura se utiliza para identificar los anchors "vecinos" de cada target con los que realizará el ranging. También para identificar las relaciones de vecindad entre los anchors y determinar el número de saltos necesarios para llegar al controlador desde cada nodo. A falta de mayor información sobre el comportamiento de los nodos se ha implementado un modelo básico, si la distancia es menor a 15 metros se considera en cobertura y si está más lejos no.

B.3.7 Algoritmo de seguimiento

En el simulador están implementados cinco algoritmos de seguimiento:

- **Trilateración:** Es un algoritmo no paramétrico que calcula la posición en base a la distancia estimada a tres anchors realizando cálculos geométricos.
- **Filtro de Kalman:** Es un método paramétrico que permite combinar la información de la dinámica del target con la de múltiples observaciones.
- **Filtro de partículas:** Es un algoritmo paramétrico que combina la información dinámica del target con la de múltiples observaciones.
- **LS-MDS:** Es un algoritmo no paramétrico que calcula la posición en base a la distancia estimada a varios anchors, mediante el algoritmo MDS. Utiliza el algoritmo SMACOF para optimizar la solución.
- **LS-DC:** Es un algoritmo no paramétrico que calcula la posición en base a la distancia estimada a varios anchors, mediante el algoritmo MDS. Utiliza los algoritmos Distance Contraction y SMACOF para optimizar la solución.

B.3.8 Modos de funcionamiento de ranging

La adquisición de la información de localización se hace a través del procedimiento de ranging entre los que el simulador implementa dos modos de funcionamiento.

▪ **Two way ranging:** El iniciador del procedimiento (target o anchor) transmite un ranging request a otro nodo, que estima el tiempo de llegada y envía un ranging response después de

un tiempo predefinido. El iniciador mide el tiempo de llegada de la respuesta y puede estimar el retardo de transmisión y la distancia entre nodos. Se necesitan dos slots.

▪ **Three way ranging:** Similar al two way ranging, pero en este caso se envían dos ranging responses para compensar la deriva del reloj (clock drift). Tanto el target como el anchor pueden iniciar el procedimiento, siendo el que ha iniciado el intercambio el que finalmente estima la distancia. Se necesitan tres slots, pero la estimación de distancia es mucho más exacta. Por defecto, se utilizará este método.

B.3.9 Adquisición y distribución de la información de localización

De cara a reducir la cantidad de recursos utilizados para la adquisición y distribución de la información de localización, se han implementado diversos modos.

- Solicitud de ranging single/multicast/broadcast: Se puede enviar una solicitud de ranging a un único anchor (single), o a un grupo de ellos (multicast) o a todos los nodos vecinos (broadcast).
- Respuesta de ranging single/multicast: Se puede enviar una respuesta de ranging única (single) inmediatamente después de recibir una solicitud, o por el contrario enviar una respuesta multicast después de recibir varias solicitudes de ranging de distintos iniciadores.
- Agregación de datos: Las distancias estimadas se pueden enviar por separado en distintos paquetes de datos o bien agruparse en un único paquete de datos (agregación de datos).

B.3.10 Método de selección de anchors

De cara a escoger los anchors más cercanos a un determinado target para realizar el posicionamiento, se han implementado diversos modos de selección.

- Selección por mínima distancia en base a conocimiento ideal: Selecciona los anchors más cercanos en base a la posición real de los targets.
- Selección por mínima distancia en base a las distancias estimadas con broadcast periódico: Realiza periódicamente actualizaciones con todos los anchors vecinos y escoge los más cercanos en base a las distancias estimadas, que se mantendrán hasta la siguiente actualización periódica.
- Selección por mínima distancia en base a la posición estimada del target: Selecciona los anchors más cercanos en base a la posición estimada de los targets.
- Selección por mínima distancia en base a la posición estimada del target con broadcast periódico: Similar al método anterior, pero incluyendo actualizaciones periódicas con todos los anchors vecinos.

B.4 Documentación del simulador

En este apartado se va a describir el funcionamiento del simulador. El simulador se ha desarrollado en C++, lenguaje orientado a objetos, lo que se adapta muy bien a la situación real, ya que cada clase se corresponderá con un elemento real (target, nodos fijos, location controller, red, etc...), y en ella se implementaran sus funcionalidades. Además, esto también

favorece a que el simulador sea más cercano al escenario real, ya que una clase sólo tiene su información o la que se le envía desde otras clases, que es lo que sucede en la situación real.

B.4.1 CSimulator

En esta clase se definen las variables globales del sistema y se implementa el programa principal del simulador. En el programa principal se ejecuta el bucle de simulación donde, a cada paso de simulación, se van llamando a las funciones que gestionan el escenario y las transmisiones.

B.4.2 CNetwork

Esta clase representa a la red, tiene acceso a todas las clases de los elementos que la forman como son los anchor nodes, target nodes, location managers y picocell manager. Contiene numerosas variables para almacenar la información relevante.

▪ CTarget_node* target_nodes[MAX_NUMBER_OF_TARGETS]: Mediante este vector se apunta a los targets.

▪ CAnchor_node* anchor_nodes[MAX_NUMBER_OF_ANCHORS]: Este vector permite acceder a los anchors.

▪CLocation_manager* location_managers[MAX_NUMBER_OF_LOCATION_MANAGERS]: Este vector apunta a los location managers.

▪ CPicocell_manager picocell_manager: Se utiliza para acceder al elemento que actúa como coordinador de la picocelda.

▪ double distance_between_anchors: En esta variable se almacena la distancia entre anchors, que determinará el número de anchors.

▪ int distance_between_location_managers: Este dato sólo se necesita en caso de utilizar la arquitectura centralizada en la red, y determina el número de location managers.

▪ int num_anchor_nodes: Almacena el número de anchor nodes.

▪ int num_target_nodes: Contiene el número de targets introducidos en el fichero parameters.

▪ int num_location_managers: Esta variable guarda el número de location managers.

Además de estas variables, la clase network contiene todos los datos introducidos en el fichero parameters.txt ya que esta clase es la que realiza la lectura de los parámetros. Esta clase realiza también la lectura de los escenarios y de las probabilidades para el movimiento de los targets, de los ficheros plan.txt y likelihood.txt respectivamente. A continuación, se explican las funciones implementadas en la clase CNetwork.

▪ CNetwork(void): Esta función es la creadora de la clase network.

▪ bool Init_network(): Se encarga de iniciar la red, se ocupa de leer los parámetros, los escenarios, crear los targets y los anchors, para ello llama a las siguientes funciones:

▪ bool Load_parameters(): Realiza la lectura del fichero parameters.txt.

▪ bool Load_plan(): Realiza la lectura del fichero plan.txt.

▪ bool Load_likelihood (): Realiza la lectura del fichero likelihood.txt.

▪ bool Load_ranging_model(): Realiza la lectura del modelo de ranging.

▪ void Distribute_anchors(): Se encarga de calcular las posiciones de los anchors.

▪ void Compute_neighbourhood_relations(): Calcula las relaciones de vecindad entre los distintos nodos, tanto entre los anchors como entre los anchors y los targets.

▪ void Distribute_location_managers(): En el caso de la arquitectura centralizada en la red, esta función calcula los anchors en que deben implementarse la función de location manager.

▪ void Assign_targets_to_location_managers(): En el caso de la arquitectura centralizada en la red, se encarga de asociar los targets a los distintos location manager existentes.

▪ void Calculate_number_of_hops(): Para la arquitectura centralizada en la red, calcula el número de saltos necesarios para transmitir un paquete desde cada anchor hasta el location manager más cercano.

▪ void Update_real_positions(): Llama a la función Update_real_position de cada target.

▪ void Update_location_stats(): Llama a la función Update_location_error de cada target.

▪ void Check_position_updates(): Se encarga de llamar a la función Check_position_update de cada target.

▪ void Update_assignation_targets_to_location_managers(): En la estrategia centralizada en la red, actualiza la asignación de los targets a los location managers.

▪ void Update_neighbourhood_relations(): Actualiza las relaciones de vecindad entre los targets y los anchors.

▪ void Collect_statistics(int n_tramas): Recoge los resultados de la simulación y los escribe en los ficheros de salida.

B.4.3 Picocell manager

Representa al coordinador de la picocelda, esta clase controla todo lo relacionado con la transmisión y el uso de slots. Para ello necesita una serie de variables y funciones.

▪ CAnchor_node** anchor_list: Permite acceder a los anchors.

▪ CTarget_node** target_list: Permite acceder a los targets.

▪ int num_target_nodes, int num_anchor_nodes: En estas variables se almacenan el número de targets y de anchors.

▪ ranging_request* pending_rangings_buffer[MAX_RANGING_BUFFER_SIZE]: Para simular el proceso de ranging, se ha creado un buffer donde se almacenan y se extraen los ranging request.

▪ location_info_packet* pending_transmissions_buffer[MAX_TRANSMISSIONS_BUFFER_SIZE]: Para simular el proceso de transmisión, se ha creado un buffer donde se almacenan y se extraen los paquetes de datos.

▪ int num_of_pending_rangings: Contiene el número de rangings pendientes.

▪ int num_of_pending_transmissions: En esta variable se almacena el número de transmisiones pendientes.

Además, una serie de variables recogen las estadísticas acerca del uso de recursos:

▪ int number_of_ranging_requests: Número de slots utilizados para la transmisión de tramas de solicitud de ranging.

▪ int number_of_ranging_responses: Número de slots utilizados para la transmisión de tramas de respuesta de ranging.

▪ int number_of_measurement_reports: Número de slots utilizados para la transmisión de tramas de datos con información de las distancias.

▪ int number_of_position_update: Número de slots utilizados para la transmisión de tramas de datos con la posición actualizada.

▪ int number_of_location_data_packets: Número de slots utilizados para la transmisión de tramas de datos, que es la suma de las dos variables anteriores.

▪ int total_number_of_slots_for_location: Almacena el número total de slots utilizados para la localización.

▪ double total_bytes_for_location: Similar a la variable anterior, pero en bytes.

▪ double total_time: Contiene el tiempo total de simulación.

▪ double total_time_for_location: Contiene el tiempo utilizado para la localización.

En cuanto a las funciones realizadas por la clase CPicocell_manager, tenemos las siguientes:

▪ CPicocell_manager(void): Crea el controlador de la picocelda.

▪ void Add_pending_ranging(ranging_request *pkt): Esta función es la encargada de almacenar una solicitud de ranging en la cola de tramas de ranging.

▪ void Add_pending_transmission(location_info_packet *pkt): Añade los paquetes de datos al a la cola de tramas de datos.

▪ ranging_request Check_pending_ranging(): Esta función sólo se utiliza en el caso del simulador MAC, sirve para consultar el request sin extraerlo de cara a comprobar si tenemos slots suficientes para procesarlo.

▪ location_info_packet Check_pending_transmission(): Esta función es similar a la anterior pero para los paquetes de datos.

▪ ranging_request Extract_pending_ranging(): Esta función extrae el primer paquete de ranging de la cola de tramas de ranging.

▪ location_info_packet Extract_pending_transmission(): Se encarga de extraer el primer paquete de datos de la cola de tramas de datos.

▪ void Process_transmission(): Se encarga de procesar los paquetes de datos que haya en la cola de tramas de datos. En el caso del simulador MAC realiza una comprobación de la cantidad de recursos disponibles antes de extraer el paquete del pending_transmissions_buffer.

▪ void Process_ranging(): Esta función procesa las solicitudes de ranging que haya en la cola de tramas de ranging, creando las tramas de respuesta correspondientes en función del modo seleccionado. Si se ha seleccionado el simulador MAC, antes de procesar el request comprueba si tiene slots suficientes para mandarlo.

▪ void Edit_pending_ranging(int number): Esta función se utiliza cuando no hay slots suficientes para mandar el request completo, pero si cabe una parte, esta se manda y mediante esta función se modifica el request del pending_rangings_buffer, para descontar lo ya mandado.

▪ void Update_neighbours(int *n_anchors): Realiza la actualización de los vecinos en el caso del simulador MAC, esta función es llamada en los slots de Topology management period (slots 13-15 de la trama MAC), cada slot está dividido en 4 subslots por lo que se actualizaran 4 nodos por slot, 12 por trama.

▪ void MAC_actualization(int n_slots): Se encarga de actualizar el tiempo y los slots necesarios para procesar un paquete ó parte de él, llama a la función Update_position.

▪ void Update_position(int n_slots): Se encarga de llamar a la función update_real_position para cada target.

▪ void Collect_stats(): Recoge los datos estadísticos relativos al uso de los recursos para presentarlos en el fichero resultados.

B.4.4 CLocation_manager

Esta clase representa a los controladores de localización, que son los elementos que realizan el cálculo de la posición, en función de las distancias estimadas. Dependiendo de la estrategia de localización utilizada, puede localizarse físicamente en uno o varios nodos fijos o en los targets.

Para poder definir este elemento se necesitan las siguientes variables.

▪ int lm_ID: Representa el identificador del location manager.

▪ CAnchor_node* associated_anchor: En el caso de que el controlador se localice en un nodo anchor, apunta al anchor correspondiente.

▪ CTarget_node* associated_target: En el caso de que el controlador se localice en un nodo target, apunta al target correspondiente.

▪ int num_targets, int num_anchors: Indica el número de targets y de anchors de la picocelda.

▪ int target_list[MAX_NUMBER_OF_TARGETS]: Vector donde se almacenan los identificadores de los targets asociados al controlador.

▪ target_info target_info_array[MAX_NUMBER_OF_TARGETS]: Contiene información relativa a cada uno de los targets asociados al controlador. La estructura target_info se define en esta clase y contiene toda la información relevante de un target.

▪ double anchors_position_x[MAX_NUMBER_OF_ANCHORS], double anchors_position_y[MAX_NUMBER_OF_ANCHORS]: En estos dos vectores se almacenan las posiciones de los anchors, que son fijas y conocidas.

▪ int num_anchors_for_positioning: Indica el número de anchors que se utilizarán para calcular la posición.

En cuanto a las funciones realizadas por esta clase, cabe destacar las siguientes.

▪ CLocation_manager(int ID): Esta función se encarga de crear el location manager.

▪ void Update_estimated_position(int targetID): Esta función es la encargada de llamar a la función Calculate_position, cuando ya tiene todas las medidas necesarias para el cálculo.

▪ void Calculate_position(int target_ID, bool trilaterate): En esta función se realiza el cálculo de la posición para un target determinado, para ello se utiliza el filtro de Kalman ó la trilateración, dependiendo de lo que se seleccione en el fichero parameters. La variable trilaterate indica si es la primera vez que se calcula la posición, en cuyo caso se utiliza obligatoriamente trilateración ya que el filtro de Kalman requiere disponer de una posición anterior.

▪ void Start_Kalman_filter(int targetID): Esta es la función se utiliza para inicializar el filtro de Kalman. Es llamada dentro de la función Calculate_position previamente a utilizar el filtro por primera vez.

▪ void select_anchors(int target_ID, int *anchor_list, double position_x, double position_y): Selecciona los anchors más cercanos a un target, tiene distintos métodos de funcionamiento implementados, que se seleccionan en parameters.

▪ void sort_anchor_list_by_distance(int target_ID, int *anchor_list, int list_size): Se encarga de ordenar una lista de anchors de menor a mayor distancia con respecto al target introducido.

▪ void Start_update(int target_ID): Inicia el proceso de actualización de la posición. Determina cuantos anchors se utilizarán para calcular la posición, lo que depende del número de anchors para posicionamiento y del número anchors en cobertura, comprobando además la necesidad de actualización broadcast periódica en el caso de los métodos de selección que la implementan. Se ejecuta la función select_anchors para que los elija y se llama a Start_ranging del target para que cree los request correspondientes.

▪ void Update_target_to_anchor_distances(double distances[5][MAX_NUMBER_OF_ANCHORS]): Elimina la distancia estimada más antigua y reordena las cuatro distancias estimadas almacenadas, para dejar hueco para la nueva estimación.

B.4.5 CAnchor_node

Este elemento representa a los nodos fijos distribuidos por el escenario, para ello necesitan una serie de variables que almacenen sus datos necesarios.

▪ int anchor_ID: Esta variable es el identificador del anchor.

▪ double position_x, double position_y: En estas variables se almacena la posición del anchor, que será fija durante toda la simulación.

▪ int num_of_neighbour_anchors: Aquí tenemos el número de anchors vecinos que tiene un anchor determinado.

▪ CAnchor_node* neighbour_anchors[MAX_NUMBER_OF_ANCHORS]: Este vector se apunta a los anchors vecinos de un anchor dado.

▪ double estimated_distances[MAX_NUMBER_OF_TARGETS]: En esta variable tenemos las distancias estimadas desde un anchor determinado a cada uno de los targets.

▪ int targets_for_location[MAX_NUMBER_OF_TARGETS]: En este vector se almacena el identificador de los targets que utilizan un anchor determinado para la localización.

▪ int num_targets_for_location: Contiene el número de targets que utilizan un anchor determinado para la localización.

▪ int num_ranging_available: Indica el número de solicitudes de ranging recibidas, lo que es necesario para implementar la mejora de respuesta multicast.

▪ int hops_to_location_manager[MAX_NUMBER_OF_LOCATION_MANAGERS]: Almacena el número de saltos que hay desde el anchor hasta cada location managers.

▪ CPicocell_manager* picocell_manager: Apunta al elemento que actúa como coordinador de la picocelda.

A continuación, se explican las funciones implementadas en esta clase.

▪ CAnchor_node(int ID): Esta función es la encargada de crear el anchor node.

▪ location_info_packet Create_position_report(int type, int target_ID, double pos_x, double pos_y, double speed_x, double speed_y): Esta función crea un paquete de datos para la transmisión de la posición actualizada.

▪ location_info_packet Create_measurement_report(int type, int num, int *targets): Esta función crea un paquete de datos para la transmisión de una o varias distancias estimadas, en el caso de que sean los anchors los encargados de iniciar el intercambio de ranging.

▪ ranging_request Create_ranging_request(int type, int num, int *targets): Esta función crea un paquete de solicitud de ranging, en el caso de que los anchors sean los encargados de iniciar el intercambio de ranging.

▪ void Start_ranging(): En el caso de que el intercambio de ranging sea iniciado por los anchors, esta función llama a create_ranging_request para crear los paquetes de solicitud de ranging en función del modo de adquisición seleccionado.

▪ void multicast_request_mac(int type, int num, int *targets): Se utiliza cuando se elige la opción multicast request en el simulador MAC, si el request no cabe en una trama se crean varios request en lugar de uno sólo.

B.4.6 CTarget_node

La clase CTarget_node representa los nodos móviles a localizar. Dispone de una serie de variables que almacenan sus datos e información relevante para poder simular su comportamiento.

▪ int target_ID: Representa el identificador del target.

▪ double real_position_x, double real_position_y: Variables que almacenan la posición real de los targets, utilizadas para calcular el error en la estimación.

▪ double estimated_position_x, double estimated_position_y: En estas variables se almacena la posición estimada del target calculada por el location controller.

▪ double speed, double direction, double direction_change_rate: En estas variables se almacenan la velocidad y la dirección (entre $-\pi$ y π) actuales del móvil.

▪ double max_speed, double min_speed, double direction_change_rate: En estos parámetros guardamos la información introducida por el fichero de entrada de parámetros acerca del modelo dinámico. En max_speed y min_speed almacenamos las velocidades máxima y mínima y direction_change_rate indica el tiempo que debe transcurrir para que el móvil escoja una nueva dirección y velocidad, en el caso del modelo estadístico del simulador.

▪ double estimated_speed_x, double estimated_speed_y: En estas variables se almacenan la velocidad estimada de cada target.

▪ double position_update_rate, double refresh_rate: En position_update_rate tenemos la tasa de actualización de la posición y en refresh_rate la tasa para las actualizaciones broadcast periódicas para los métodos de selección que las implementan.

▪ double time_since_last_update: En esta variable se va almacenando el tiempo que ha pasado desde la última actualización de la posición, cuando es igual ó mayor al position_update_rate, el target necesita una actualización de la posición, y cuando se realiza su valor se pone a cero.

▪ double time_since_last_change: Similar a la variable anterior pero almacena el tiempo desde su último cambio de dirección y velocidad.

▪ double time_since_last_refresh: Esta variable contiene el tiempo transcurrido desde la última actualización broadcast periódica y se pone a cero cuando se realiza.

▪ int num_anchors_for_location: Almacena el número de anchors utilizado para calcular la posición.

▪ int anchors_for_location[MAX_NUMBER_OF_ANCHORS]: Almacena el identificador de los anchors utilizados para calcular la posición.

▪ int num_of_neighbour_anchors: Contiene el número de anchors en cobertura del target.

▪ CAnchor_node* neighbour_anchors[MAX_NUMBER_OF_ANCHORS]: Este vector permite a cada target acceder a sus anchors vecinos.

▪ int number_of_hops_to_location_manager: Almacena el número de saltos necesarios para llegar al location manager.

▪ int update_status: Indica el estado en el que se encuentra el target, 0:Estado inicial, se fuerza una actualización inicial 1:La actualización está pedida 2: Se está recibiendo información de ranging 3:Actualizado.

▪ int num_ranging_available: Indica el número de distancias estimadas, necesario para implementar las mejoras de respuesta multicast y agregación de datos.

▪ double positioning_error, double sum_of_positioning_error, double sum_of_squared_positioning_error, double average_positioning_error, double variance_positioning_error, int number_of_positioning_proceses, int histogram_count[MAX_NUMBER_OF_CATEGORIES]: En estas variables se almacenan los datos relativos al error de posicionamiento para cada target, se calculan mediante la función Update_stats.

▪ int number_of_location_samples, double sum_of_location_error, double sum_of_squared_location_error, double average_location_error, double variance_location_error: En estas variables se calculan los datos relativos al error de seguimiento, mediante la función Update_location_error.

▪ int number_of_histogram_categories, int max_histogram_value: En estas variables se almacenan los datos relativos a los histogramas, se introducen mediante el fichero de entrada parameters.

▪ double estimated_distances[MAX_NUMBER_OF_ANCHORS]: Este vector almacena la distancia estimada a cada anchor vecino.

▪ CLocation_manager* associated_location_manager: Apunta al location manager asociado a este target.

▪ CPicocell_manager* picocell_manager: Apunta al elemento que actúa como coordinador de la picocelda.

Una vez conocemos las variables de esta clase, se analizan las funciones que realiza.

▪ CTarget_node(int ID, double maximum_speed, double minimum_speed, double dir_change_rate, double pos_update_rate, double v_refresh_rate, int hist_categories, int max_hist_value): La primera función es la que crea el nodo target

▪ double Estimate_distance(CAnchor_node* anchor): Calcula la distancia estimada al anchor introducido, en base a la distancia real mediante la aplicación del modelo probabilístico de ranging.

▪ void Update_real_position(int n_slots): En esta función se actualizan los contadores de tiempo (time_since_last_update, time_since_last_change, time_since_last_refresh) y calcula la nueva posición real. Para el modelo estadístico, en caso de que se haya superado el valor de direction_change_rate, se determinarán una nueva dirección y velocidad. Para el modelo basado en paredes y rutas, en caso de que se haya alcanzado el nodo destino, se determinará un nuevo nodo destino y una nueva velocidad.

▪ CAnchor_node* Pick_anchor_for_location_manager(): Cuando se utiliza una arquitectura distribuida, esta función selecciona el anchor que actuará como location manager.

▪ ranging_request Create_ranging_request(int type, int num, int *anchors): Esta función crea el paquete de solicitud de ranging, en caso de que el intercambio de ranging sea iniciado por los nodos target.

▪ location_info_packet Create_measurement_report(int type, int num, int *anchors): Esta función crea un paquete de datos para la transmisión de una o varias distancias estimadas.

▪ void sort_neighbours_by_distance(int* anchor_list): Ordena la lista de anchors vecinos de menor a mayor distancia respecto al target.

▪ void Select_anchors(): Selecciona los anchors más cercanos a un target, de acuerdo al método de selección escogido.

▪ void Check_position_update(): Comprueba si le toca actualizar la posición (time_since_last_update) y si lo necesita llama a start_update.

▪ void Start_update(): Inicia el proceso de actualización de la posición. Determina cuantos anchors se utilizarán para calcular la posición, lo que depende del número de anchors para posicionamiento y del número anchors en cobertura, comprobando además la necesidad de actualización broadcast periódica en el caso de los métodos de selección que la implementan. Se ejecuta la función select_anchors para que los elija y se llama a Start_ranging para crear las solicitudes de ranging correspondientes.

▪ void Start_ranging(): Esta función llama a create_ranging_request para crear los paquetes de solicitud de ranging en función del modo de adquisición seleccionado.

▪ void multicast_request_mac(int type, int num, int *anchors): Esta función se utiliza cuando se elige la opción multicast request en el simulador MAC, si el request no cabe en una trama se crean varios request en lugar de uno sólo.

▪ void Update_target_info(): Cuando se asigna un target a un determinado location controller, mediante está función se proporciona al location controller toda la información relativa al target.

▪ void Update_stats(): Calcula los datos relativos al error de posicionamiento.

▪ void Update_location_error(): Calcula los datos relativos al error de seguimiento.

B.4.7 cPlaneEKF

La clase cPlaneEKF implementa el filtro de Kalman. Las funciones más importantes que realiza son las siguientes:

▪ void makeProcess(): Se almacena en un vector temporal el nuevo vector de estado.

▪ void makeMeasure(): Se actualiza el vector de medidas.

▪ void makeA(): Función que construye la matriz A.

▪ void makeH(): Función que construye la matriz H.

▪ void makeV(): Función que construye la matriz V.

▪ void makeR(): Función que construye la matriz R.

▪ void makeW(): Función que construye la matriz W.

▪ void makeQ(): Función que construye la matriz Q.

B.4.8 pffuncs.cc

Este fichero fuente de C++, implementa el filtro de partículas. Este fichero implementa dos clases:

▪ cv_state: Almacena la posición y velocidad, tanto en el eje x como en el eje y, para cada partícula.

▪ cv_obs: Almacena la posición y la distancia medida de cada anchor.

Las funciones más relevantes se comentan a continuación.

▪ smc::particle<cv_state> fInitialise(smc::rng *pRng): Esta función inicializa las partículas obteniendo una posición y una velocidad aleatorias.

▪ void fMove(long lTime, smc::particle<cv_state> & pFrom, smc::rng *pRng): Obtiene una aceleración de forma aleatoria y actualiza la partícula que se le pasa como parámetro, actualizando su posición(con la velocidad y la aceleración) y su velocidad(con la aceleración).

▪ double logLikelihood(const cv_state & X): Calcula el logaritmo de la probabilidad para la partícula que se le pasa como parámetro.

▪ double integrand_mean_x(const cv_state&, void*): Calcula la posición en el eje x del target, a partir de las partículas.

▪ double integrand_mean_y(const cv_state&, void*): Calcula la posición en el eje y del target, a partir de las partículas.

B.4.9 CMDSalgorithm

La clase CMDSalgorithm implementa los algoritmos LS-MDS y LS-DC. Las funciones más relevantes que realiza se analizan a continuación.

▪ void Centralized_Algorithm(): Implementa el algoritmo LS-MDS. Esta función es llamada por el controlador de localización, en su función Calculate_position.

▪ void Centralized_Algorithm_EUWB(): Implementa el algoritmo LS-DC. Esta función es llamada por el controlador de localización, en su función Calculate_position.

▪ void MDSMAP(): Implementa el algortimo MDS.

▪CGeneralMatrix smacofEUWBW (CGeneralMatrix D, CGeneralMatrix Xanchors, CGeneralMatrix x0, CGeneralMatrix WLocal): Realiza el algoritmo de optimización SMACOF, para las distancias almacenadas en la matriz D y la posición almacenada en x0. En Xanchors se almacenan las posiciones de los anchors y en Wlocal se guarda la matriz de pesos.

▪CGeneralMatrix DistContrPos(CGeneralMatrix X0_partial, CGeneralMatrix WLocal, CGeneralMatrix Dlocal): Realiza el algoritmo Distance Contraction, para la matriz de distancias Dlocal. En Wlocal se encuantra la matriz de pesos y en X0_partial se encuantra la posición estimada del target y las posiciones de los anchors.

▪ void LINK_Weight(double gammaValue): Es la función encargada de calcular los pesos utilizados en la ponderación de las estimaciones. La variable gammaValue toma diferente valor según sea el WLS-MDS o el WLS-DC.

B.4.10 CGeneralMatrix

Esta clase se ha adaptado del desarrollo en C# realizado en el proyecto EUWB y contiene funciones para realizar operaciones con matrices. Las principales variables que utiliza son:

▪ int m: Número de filas.

▪ int n: Número de columnas.

▪ double **A: Array bidimensional, utilizado para almacenar el resultado de las operaciones.

En esta clase se implementan funciones para realizar operaciones básicas de matrices, como son sumas, productos, etc, así como operaciones más complejas como descomposiciones (descomposición de Cholesky, descomposición en valores propios, etc.).

B.4.11 matemat.cpp

El fichero fuente matemat.cpp incluye funciones matemáticas de propósito general. A continuación se enumeran dichas funciones.

▪ double uniforme(double min, double max): Devuelve un valor aleatorio en el intervalo dado con distribucion uniforme.

▪ double crop2PI(double ang): Devuelve el angulo indicado circunscrito al intervalo [-PI,PI].

▪ void cart2pol(double x, double y, double &mod, double &phase): Convierte las coordenadas cartesianas dadas a coordenadas polares.

▪ void pol2cart(double mod, double phase, double &x, double &y): Convierte las coordenadas polares dadas a coordenadas cartesianas.

▪ int rand_int(int min,int max): Devuelve un numero entero aleatorio dentro de un intervalo dado.

▪ double rand_exp(double lambda): Devuelve un valor con distribucion exponencial.

▪ double gaussian(double mean, double var): Devuelve un valor con distribucion gaussiana.

▪ double gamma(double shape, double scale): Devuelve una variable aleatoria con distribucion Gamma.

▪ double Hypot(double a, double b): Función que calcula la longitud de la hipotenusa de un triángulo rectángulo.

▪ void cutoff_point(double line1[4], double line2[4], double c_point[2]): Función que calcula el punto de corte entre dos rectas.

▪ int n_cutoff_point(double anchor_x, double anchor_y, double target_x, double target_y): Función que devuelve el número de puntos de corte de un enlace, entre target y anchor, con las paredes del escenario.

▪ double ValueOfTCDF(double x, int k): Función que devuelve el valor de la distribución t de Student. Esta función utiliza funciones adicionales, incluidas también en matemat.cpp.

B.4.12 Fichero de entrada de parámetros

En el fichero de entrada (Parameters.txt) se seleccionan los distintos parámetros a utilizar en la simulación, que se exponen a continuación.

PARÁMETROS GENERALES:

▪ Tipo de simulador: ideal (ranging y transmisión de datos instantánea e ilimitada) o adaptada a la MAC de EUWB (el número de slots está limitado y la temporización se adapta a la estructura de la supertrama MAC).

▪ Modelo de simulador: Estadístico o basado en paredes y rutas.

▪ Tiempo de simulación: Se introduce la duración de la simulación en segundos.

▪ Paso de simulación: Se puede elegir este parámetro, por defecto determinado por la duración de los slots de la trama MAC (3.6864 ms).

▪ Estrategia de localización: Este parámetro permite seleccionar entre las distintas arquitecturas explicadas anteriormente: centralizada en la red, distribuida, centralizada en el móvil.

▪ Iniciador de la actualización de la posición: Permite seleccionar quien inicia el proceso de actualización de la posición, la red o los móviles.

▪ Selección de anchors: Indica quien realiza la selección de anchors, la red o los móviles.

▪ Modos de selección de anchors: Mediante este parámetro seleccionamos el método de selección de anchors, 1:Ideal, 2: Distancias estimadas con broadcast periódico, 3: Posición estimada y 4: Posición estimada con broadcast periódico.

▪ Iniciador del ranging: Permite seleccionar quien inicia los intercambios de ranging, los nodos anchor o los nodos target.

▪ Tipo de ranging: Mediante este parámetro se selecciona el modo de funcionamiento del ranging, 2-way ranging o 3-way ranging.

▪ Agregación de datos: Con este dato seleccionamos si queremos utilizar agregación de datos ó no.

▪ Ranging Request: En este caso podemos seleccionar tres modos de funcionamiento, Single request, Multicast request y Broadcast request.

▪ Ranging Response: Se elige entre Single response ó Multicast response.

PARÁMETROS SUPERTRAMA MAC:

Mediante los siguientes parámetros seleccionamos las características de la supertrama, por defecto se utilizan los valores propuestos en el proyecto EUWB.

▪ Duración de la supertrama: El valor utilizado para este proyecto es de 200 ms

▪ Duración del slot: Se fija en 3.6864 ms

▪ Tamaño de paquete (bytes): El tamaño de los paquetes se fija en 160 bytes.

▪ Número total de slots en la supertrama: Se considera 53.

▪ Número de slots para el beacon period: Como indica el proyecto se utilizan 12 slots.

▪ Número de slots para el topology management period: Se le da un valor de 3 slots.

▪ Número de slots para comunicaciones de datos: Se utiliza un valor de 8 slots.

▪ Número de slots para ranging: Esta fijado en 12 slots.

▪ Número de slots para GTS request period: Se fija en 6 slots.

▪ Número de slots en CAP period: Tiene un valor de 12 slots.

NETWORK

A continuación se seleccionan los parámetros que definen las características de la red.

▪ Distancia entre anchors: La distancia entre los nodos anchor determina el número de anchors que cubrirán el escenario.

▪ Número de targets nodes: Selecciona el número de dispositivos a seguir.

▪ Distancia entre location managers: Permite seleccionar la distancia (en número de nodos anchor) que separa a los controladores de localización, lo que determina también su número.

▪ Tamaño del área x.

▪ Tamaño del área y.

NODOS MÓVILES

En este apartado se definen las características de los dispositivos a seguir.

▪ Velocidad máxima (m/s): Mediante la velocidad máxima y mínima se define el rango de la velocidad del móvil.

▪ Velocidad mínima (m/s).

▪ Tasa de cambio de dirección: Marca el tiempo en que la velocidad y la dirección permanece constante, transcurrido ese tiempo selecciona otra dirección y velocidad.

▪ Tasa de actualización de la posición (ms): Define el tiempo entre actualizaciones de la posición.

▪ Refresh rate (ms): Para los modos de selección con broadcast periódico, define el tiempo entre las actualizaciones broadcast periódicas.

▪ Refresh threshold (m): Para el modo de selección basado en distancias estimadas con broadcast periódico, este parámetro se refiere al umbral en metros para realizar una actualización por pérdida de cobertura.

ALGORITMO DE SEGUIMIENTO

Mediante estos parámetros seleccionamos las características del algoritmo de seguimiento.

▪ Algoritmo de seguimiento: Permite seleccionar el algoritmo de seguimiento a utilizar.

▪ Número de anchors utilizados para la localización.

ESTADISTICAS

▪ Número de categorías de histograma.

▪ Valor máximo para el histograma.

RANGING

▪ Modelo de ranging: Permite seleccionar el modelo de ranging a utilizar.

▪ Nivel de error residual de ranging: Este parámetro hace referencia a la varianza del error residual en la estimación de las distancias.

B.4.13 Ficheros de entrada de escenarios

Mediante los ficheros plan.txt se introducen las características de los escenarios en el simulador. Cuando se selecciona la distancia entre anchors en el fichero de parámetros, el simulador selecciona automáticamente el fichero plan.txt correspondiente a esa distancia. En estos ficheros se introduce la información de las paredes, los nodos y los segmentos:

- Paredes: En el caso de las paredes se introducen las coordenadas de los dos extremos de la pared (las paredes implementadas en el simulador sólo pueden ser rectas).
- Nodos: Para los nodos se introducen sus coordenadas y los identificadores de sus nodos vecinos.
- Segmentos: En los segmentos se indican los identificadores de los dos nodos que une.

B.4.14 Fichero de probabilidades para el modelo dinámico de los targets

Mediante el fichero likelihood.txt se introducen en el simulador las probabilidades para el modelo dinámico de los targets en el caso de utilizar el modelo basado en paredes y rutas. En este fichero se incluyen las probabilidades de que cuando un target llega al nodo destino siga recto, se dé la vuelta y vuelva al mismo nodo, gire a la izquierda o gire a la derecha.

B.4.15 Ficheros de salida: error y resources

El simulador tiene como salida dos ficheros donde se guardan los resultados de la simulación realizada.

En el fichero error.txt, se almacenan las estadísticas relativas al error en el posicionamiento, tanto a nivel global como para cada target. Contiene los siguientes datos:

▪ Número de procesos de posicionamiento.

▪ Error medio de posicionamiento: Este es el error de la posición cada vez que se actualiza y por tanto hace referencia al error del procedimiento de cálculo de la posición.

▪ Varianza del error de posicionamiento.

▪ Histograma del error de posicionamiento: En base al valor máximo y número de categorías definido en el fichero de parámetros.

▪ Número de muestras de localización.

▪ Error medio de seguimiento: Este es el error en la posición disponible en el móvil a cada paso de simulación y por tanto tiene en cuenta tanto el error de posicionamiento como el debido a que la posición no esté actualizada.

▪ Varianza del error medio de seguimiento.

En el fichero resources.txt, se guardan las estadísticas relacionadas con los recursos consumidos para cada simulación, los datos son los siguientes:

▪ Número de slots utilizados para la transmisión de paquetes de datos relativos a localización: Es la suma de los dos siguientes

▪ Número de slots utilizados para la transmisión de las distancias estimadas.

▪ Número de slots utilizados para transmisión de la posición actualizada.

▪ Número de slots utilizados para respuestas de ranging.

▪ Número de slots utilizados para respuestas de ranging.

▪ Número total de slots usados para la localización.

▪ Número de bytes usados para la localización.

▪ Tiempo total (s).

▪ Tasa de datos usada para localización.

▪ Tiempo total utilizado para localización (s).

▪ Porcentaje de tiempo usado para la localización.

B.4.16 Librerías

En este punto se citan las librerías utilizadas en el simulador:

▪ libKalman.lib: Es la librería utilizada para desarrollar el filtro de Kalman.

▪ smctcd.lib: Es la librería utilizada para implementar el filtro de partículas.

▪ AltaxoBase.dll y AltaxoCore.dll: La librería Altaxo se utiliza en el algoritmo SMACOF, para obtener la matriz pseudo-inversa.

▪ DotNumerics.dll: Esta librería se emplea en el algoritmo Distance Contraction, para las funciones de maximización y minimización.

ANEXO C: Resultados

C.1 Comparativa de algoritmos básicos

C.1.1 Modelo estadístico

En primer lugar se van a evaluar los algoritmos de localización para el modelo estadístico del simulador. La evaluación de los algoritmos de localización se va a realizar en función del número de anchors utilizados para la estimación de la posición.

En la figura C1, se muestra el error medio de posicionamiento de los algoritmos, para una distancia entre anchors de 10 metros, lo que significa que el área está cubierta por 36 nodos anchors, y para un error residual de ranging de $\sigma_n = 0.7$.

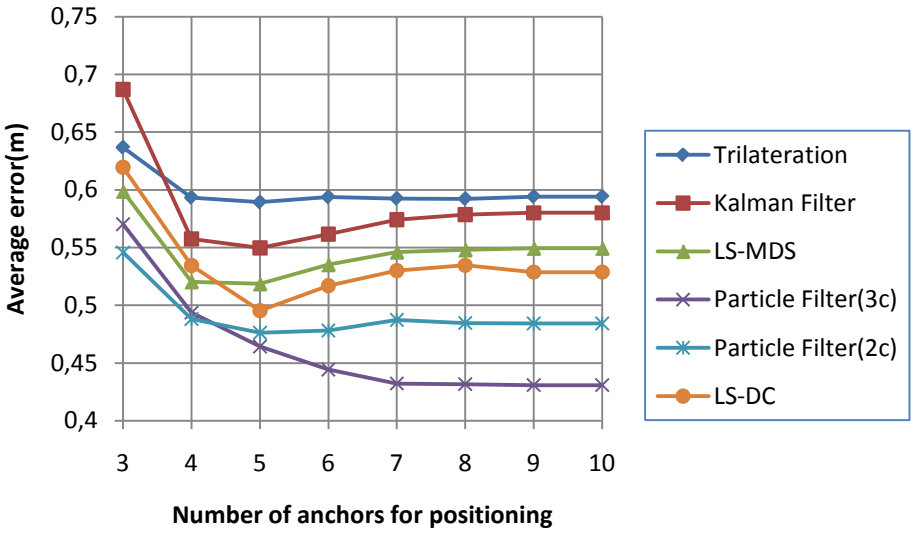

Figura C1. Error medio de posicionamiento. Distancia entre anchors = 10 m. σ_n = 0.7.

Como puede observarse en la figura C1, los mejores resultados se obtienen con el filtro de partículas. Como ya se explicó, para calcular la probabilidad de cada partícula se dispone de tres modelos, basados en una gaussiana, en la suma de dos gaussianas y en la suma de tres gaussianas respectivamente. Con la notación "3c" y "2c", se hace referencia al modelo usado, siendo "3c" el modelo basado en la suma de 3 gaussianas y "2c" el basado en la suma de dos gaussianas. No obstante, los resultados conseguidos con el filtro de partículas no son realistas, debido a que en el simulador las distancias estimadas se obtienen mediante un modelo de ranging estadístico y el filtro de partículas utiliza también un modelo estadístico del error de medida para obtener los pesos de las partículas. Además ese modelo se ha optimizado mediante simulaciones, de tal forma que su comportamiento es prácticamente similar al del modelo de ranging. En un sistema real, una caracterización tan precisa del modelo de error especifico del escenario requeriría unas fases de medida y de calibración muy costosas. Y el uso de un modelo genérico no proporcionará resultados tan buenos. Por lo tanto, los resultados del filtro de partículas se deberán considerar como una referencia más que como una implementación realista. El modelo de ranging estadístico tiene 3 componentes, por eso, el filtro de partículas utilizando el modelo de 3 componentes obtiene mejores resultados que utilizando solo dos componentes.

Los algoritmos LS-MDS, LS-DC y filtro de Kalman presentan una evolución similar, sin embargo los mejores resultados se obtienen con LS-MDS y LS-DC. En particular, con LS-MDS se obtienen mejores resultados para 3 y 4 anchors y con LS-DC para más de 4 anchors. En los tres algoritmos, el número óptimo de anchors es 5. Si se utilizan más anchors, los nuevos anchors estarán más lejos del target y tendrán mayor desviación de ranging, provocando que el error de posicionamiento aumente. Por último, la trilateración es el algoritmo que peores resultados ofrece y es independiente del número de anchors escogido, ya que únicamente se emplean los tres anchors más cercanos al target en el algoritmo.

Para todos los algoritmos hay un incremento del error cuando solo se utilizan tres anchors y el error se mantiene constante para más de 7 anchors, debido a que no es probable que el target esté en cobertura de más de 7 anchors.

Los resultados cuando se aumenta la distancia entre anchors a 12.5 metros, por lo que el área estará cubierta por 25 anchors, y se mantiene el error residual de ranging de $\sigma_n = 0.7$, se muestran en la figura C2.

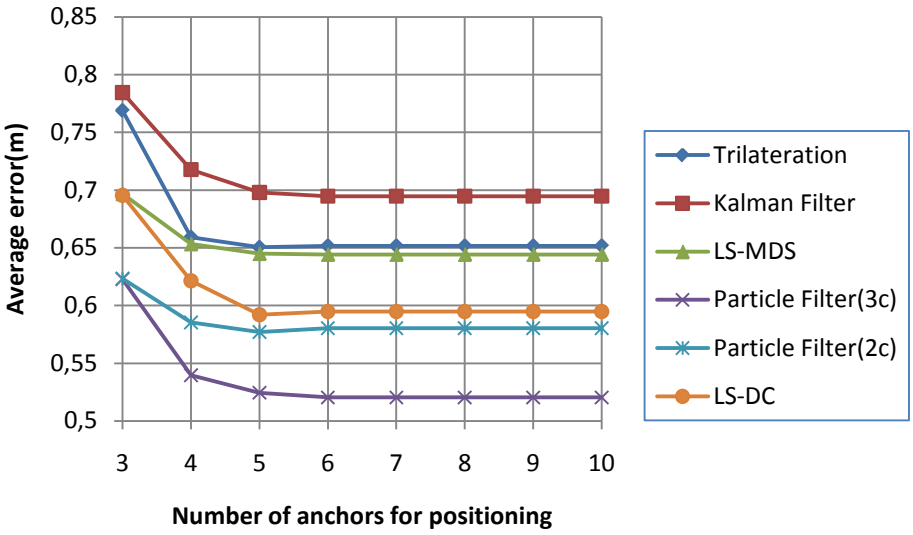

Figura C2. Error medio de posicionamiento. Distancia entre anchors = 12.5 m. σ_n = 0.7.

Debido a que la distancia entre anchors es elevada, los anchors utilizados para el posicionamiento estarán bastante lejos del target. El filtro de partículas presenta los mejores resultados ya que se beneficia de su preciso modelo del error de medida. El algoritmo LS-DC obtiene mejores resultados que el LS-MDS, ya que el algoritmo Distance Contraction permite compensar las desviaciones en las estimaciones cuando la mayoría de las distancias estimadas a los anchors presentan una desviación positiva. Por lo tanto, cuanto mayor sea la distancia entre anchors, mayor será la desviación de las distancias estimadas y mejores serán los resultados frente al uso únicamente del MDS, es decir, frente al algoritmo LS-MDS. La trilateración y el LS-MDS tienen un comportamiento muy similar mientras que, los peores resultados se obtienen con el filtro de Kalman, ya que su simple modelo de error Gaussiano no puede tratar con las elevadas desviaciones de las distancias estimadas para anchors muy alejados.

Para todos los algoritmos el error se mantiene constante para más de 5 anchors, ya que no es probable que el target tenga cobertura con más de 5 anchors. No se consideran distancias mayores entre anchors, ya que se podría perder la cobertura entre los nodos anchor.

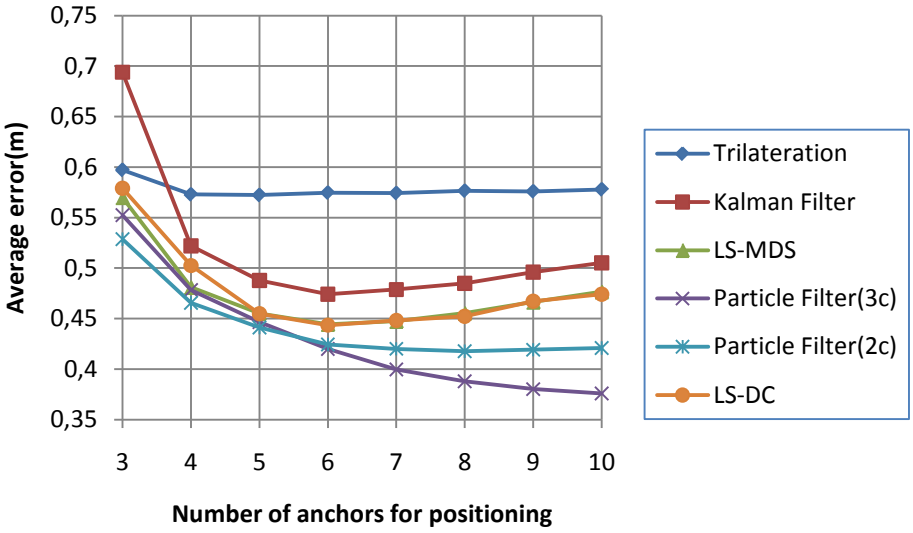

Figura C3. Error medio de posicionamiento. Distancia entre anchors = 8.33 m. σ_n = 0.7.

En esta situación los anchors escogidos para el posicionamiento están cercanos al target, por lo tanto las desviaciones de las estimaciones son menores. Como en los casos anteriores el filtro de partículas es el que mejor resultados obtiene. Sin embargo en este caso, como la desviación de las estimaciones es menor, el filtro de partículas utilizando el modelo de 2 componentes se aproxima bastante al modelo de 3 componentes, incluso presenta un error algo inferior en algunos casos. Al ser las desviaciones menores, el algoritmo Distance Contraction no funciona tan bien, de tal forma que los algoritmos LS-MDS y LS-DC obtienen resultados muy similares. El filtro de Kalman tiene un comportamiento similar al LS-MDS y al LS-DC y en esta situación el número óptimo de anchors aumenta a 6. Para un número de anchors inferior a 6, los algoritmos LS-MDS y LS-DC obtienen un error similar al filtro de partículas, ya que los anchors están cercanos. Si se utilizan más anchors, al estar más alejados, las elevadas desviaciones de las estimaciones hacen que el error del LS-MDS y LS-DC aumente. La trilateración presenta los peores resultados.

En la figura C4, se muestran los resultados para un distancia entre anchors de 7.14 m, el área está cubierta por 64 anchors, y para un error residual de ranging de $\sigma_n = 0.7$.

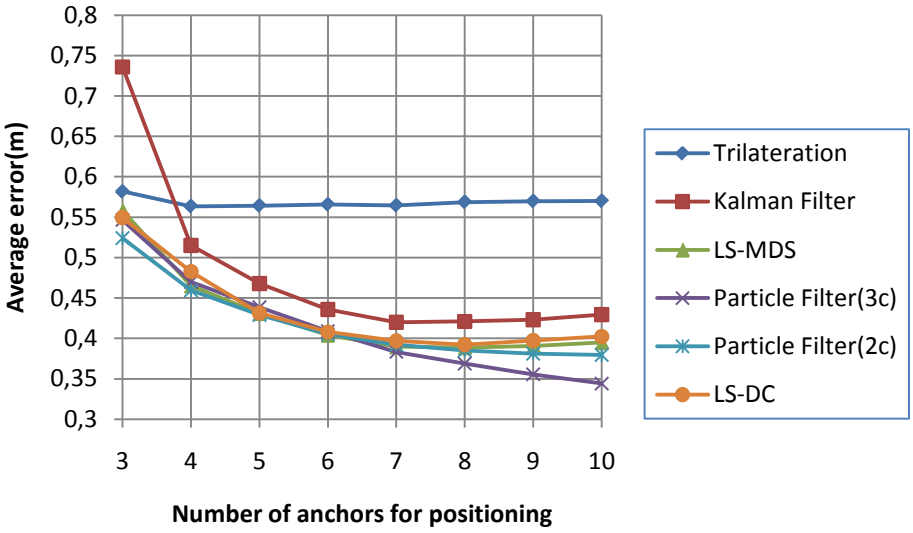

Figura C4. Error medio de posicionamiento. Distancia entre anchors = 7.14 m. σ_n = 0.7.

El número óptimo de anchors para el LS-MDS, LS-DC y filtro de Kalman aumenta a 7. El filtro de Kalman tiene un comportamiento muy similar al LS-MDS y LS-DC aunque con un error mayor. Los resultados del filtro de partículas con 3 y con 2 componentes son similares a los obtenidos para LS-MDS, LS-DC y filtro de Kalman para un número de anchors inferior a 7, mientras que para un número de anchors superior, como los anchors que se añaden están alejados del target, el filtro de partículas de 3 componentes obtiene los mejores resultados. La trilateración presenta los peores resultados.

En la figura C5 se muestran los resultados para un distancia entre anchors de 6.25 m, el área está cubierta por 81 anchors, y para un error residual de ranging de $\sigma_n = 0.7$.

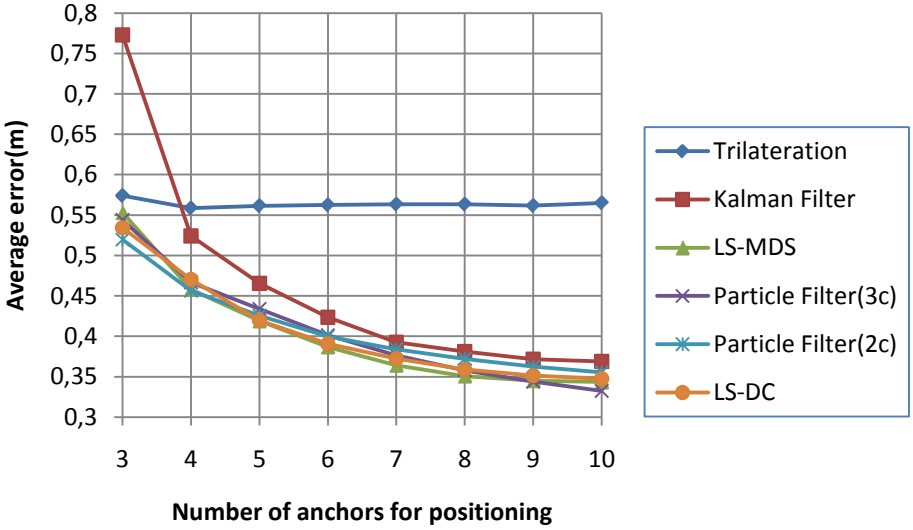

Figura C5. Error medio de posicionamiento. Distancia entre anchors = 6.25 m. σ_n = 0.7.

Todos los algoritmos menos el de trilateración presentan un comportamiento similar, aunque el filtro de Kalman con un error mayor. Esto se debe a que todos los anchors utilizados están cercanos al target. El número óptimo de anchors usados para el cálculo es 10. El algoritmo de trilateración presenta los peores resultados.

A modo de resumen, puede apreciarse que cuando la distancia entre anchors se reduce, el error de posicionamiento es menor. Los algoritmos más sensibles a la distancia entre anchors son el LS-MDS, LS-DC y el filtro de Kalman ya que son más sensibles a las desviaciones en las estimaciones. Mientras que el filtro de partículas, gracias a su modelo del error de medida, es capaz de compensar esas desviaciones. La trilateración únicamente muestra una leve mejoría cuando la distancia entre anchors disminuye. También hay que notar que cuando la distancia entre anchors disminuye, el número de anchors óptimo que proporciona la máxima precisión aumenta.

Ahora se muestran los resultados cuando el error residual de ranging disminuye a $\sigma_n = 0.3$, para unas distancias entre anchors de 10 y 7.14 m, en las figuras C6 y C7 respectivamente.

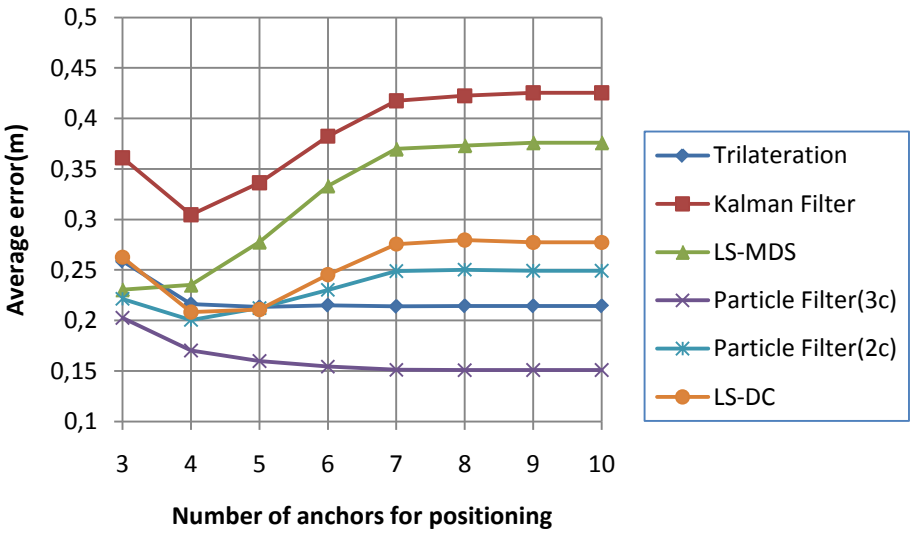

Figura C6. Error medio de posicionamiento. Distancia entre anchors = 10 m. σ_n = 0.3.

Como podría esperarse, el comportamiento de todos los algoritmos es mejor que en el caso de utilizar la misma configuración con un error residual de ranging de 0.7 y puede obtenerse un error medio de posicionamiento en torno a 15-30 cm. La mejora más remarcable es la del algoritmo de trilateración, que obtiene mejores resultados que el LS-MDS, LS-DC y filtro de partículas de 2 componentes para un número de anchors superior a 4. Esto significa que la trilateración requiere una elevada resolución en la estimación del TOA para proporcionar buenos resultados, ya que siempre utiliza tres medidas para calcular la posición y no puede beneficiarse de la diversidad de las medidas. Como el error residual ha disminuido, las desviaciones del ranging tienen mayor importancia y por ello los anchors cercanos proporcionan estimaciones mucho más precisas que anchors alejados. Como consecuencia, el número óptimo de anchors es de 3 para el LS-MDS y de 4 para el filtro de Kalman, LS-DC y filtro de partículas de 2 componentes, empeorando mucho el comportamiento cuando se aumenta el número de anchors utilizados en el cálculo de la posición.

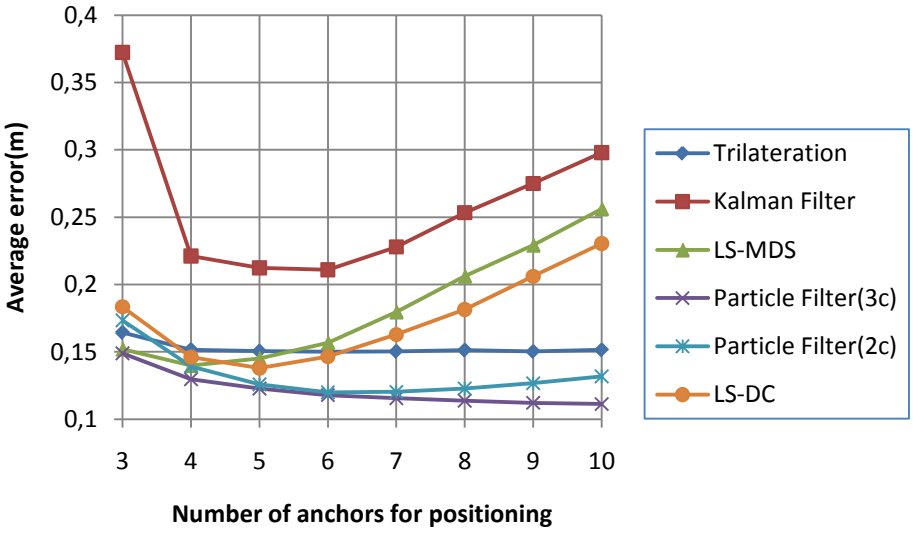

Figura C7. Error medio de posicionamiento. Distancia entre anchors = 7.14 m. σ_n = 0.3.

En la figura C7, la distancia entre anchors se reduce a 7.14 m. Para esta configuración el comportamiento de todos los algoritmos mejora, ya que los anchors seleccionados están más cercanos y proporcionan estimaciones muy precisas, ya que el error residual es bajo. El algoritmo LS-MDS obtiene resultados muy similares al filtro de partículas de 3 componentes para 3-4 anchors, obteniendo un mínimo del error medio de 14 cm para 4 anchors. El LS-DC tiene un comportamiento muy similar al LS-MDS, obteniendo mejores resultados para más de 4 anchors y obteniendo un mínimo del error medio de 14 cm para 5 anchors. El comportamiento de ambos filtros de partículas es muy similar, aunque el de 2 componentes obtiene un mayor error.

C.1.2 Modelo basado en paredes y rutas

En este punto se van a evaluar los algoritmos de localización para el modelo basado en paredes y rutas del simulador. Al igual que en el anterior apartado, la evaluación de los algoritmos de localización se va a realizar en función del número de anchors utilizados para el cálculo de la posición.

En la figura C8, se muestran los resultados para una distancia entre anchors de 10 m y un error residual de ranging de $\sigma_n = 0.7$. En la figura 5 se muestra el escenario modelado para esta configuración.

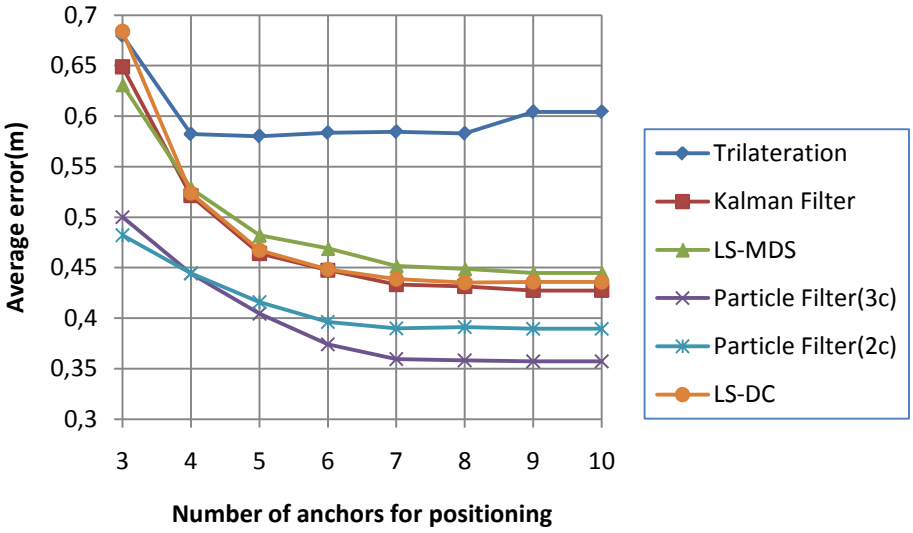

Figura C8. Error medio de posicionamiento. Distancia entre anchors = 10 m. σ_n = 0.7.

Los algoritmos LS-MDS,LS-DC y filtro de Kalman presentan un comportamiento similar, aunque el LS-MDS con un error mayor para 5 o más anchors. El filtro de partículas de 3 componentes obtiene mejores resultados que el de dos componentes para más de 4 anchors, ya que los anchors que se añaden están alejados y las desviaciones del ranging son mayores. La trilateración es el algoritmo con peores resultados, mientras que el filtro de partículas es el que ofrece los mejores resultados.

Para todos los algoritmos el error se mantiene constante para más de 7 anchors, debido a que no es probable que el target esté en cobertura de más de 7 anchors. Por lo tanto el número óptimo de anchors utilizados para el cálculo es 7.

Si se observa la figura C1, se puede ver que en el modelo estadístico para la misma configuración, el filtro de partículas obtiene mejores resultados respecto al resto de algoritmos que en el caso del modelo basado en paredes y en rutas. Esto se debe a que se ha modificado el modelo de ranging y ya no se calcula la probabilidad de que el enlace entre target y anchor sea LOS, NLOS o NLOS severo sino que se calcula el número de paredes existentes entre el target y los anchors utilizados en la medida, y en base al número de paredes se determina la componente de error que se utiliza. El modelo estadístico del error de medida del filtro de partículas sigue ofreciendo muy buenos resultados, aunque en este caso se diferencia algo más del modelo de ranging.

En la figura C9 aparecen los resultados si se aumenta la distancia entre anchors a 12.5 m y se mantiene el error residual de ranging $\sigma_n = 0.7$. En la figura 6 se puede observar el plano modelado, para esta distancia entre anchors.

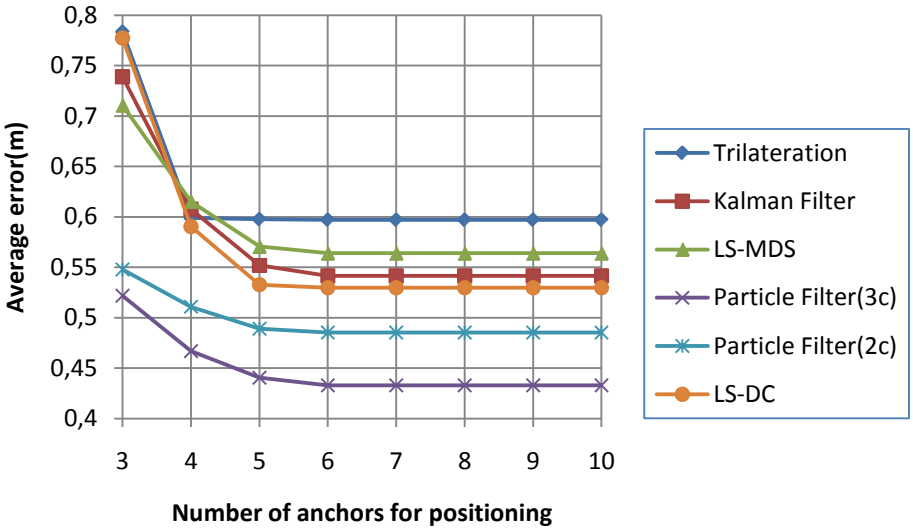

Figura C9. Error medio de posicionamiento. Distancia entre anchors = 12.5 m. σ_n = 0.7.

Ahora la distancia entre anchors es mayor, así que las desviaciones del ranging son mayores ya que los anchors están más alejados. Los algoritmos LS-MDS, LS-DC y filtro de Kalman presentan una evolución similar aunque el LS-MDS con un error mayor para más de 4 anchors. El LS-DC mejora más respecto al LS-MDS que en la configuración anterior, ya que como se explico el algoritmo DC funciona mejor cuando todas las distancias estimadas presentan un error positivo.

El filtro de partículas de 3 componentes es el que mejor resultados ofrece, ya que es capaz de compensar las grandes desviaciones del ranging para anchors alejados. El filtro de partículas de 2 componentes presenta una evolución similar aunque con un error bastante superior. La trilateración es el algoritmo que peores resultados ofrece.

Para todos los algoritmos el error se mantiene constante para más de 6 anchors, ya que no es probable que el target esté en cobertura de más de 6 anchors. Para esta configuración el número óptimo de anchors utilizados para el cálculo de la posición es 6.

Al igual que en el modelo estadístico no se consideran distancias mayores entre anchors, ya que se podría perder la cobertura entre los nodos anchor.

Observando las figuras C8 y C9 vemos que al aumentar la distancia entre anchors, el comportamiento de los algoritmos empeora. Como en el modelo estadístico los algoritmos más sensibles a la distancia entre anchors son el LS-MDS, el LS-DC y el filtro de Kalman. La trilateración únicamente empeora un poco cuando la distancia entre anchors aumenta.

Ahora se presentan los resultados para las 2 configuraciones anteriores, pero reduciendo el error residual de ranging a $\sigma_n = 0.3$. En la figura C10, se muestran los resultados para una distancia de 10 m entre anchors.

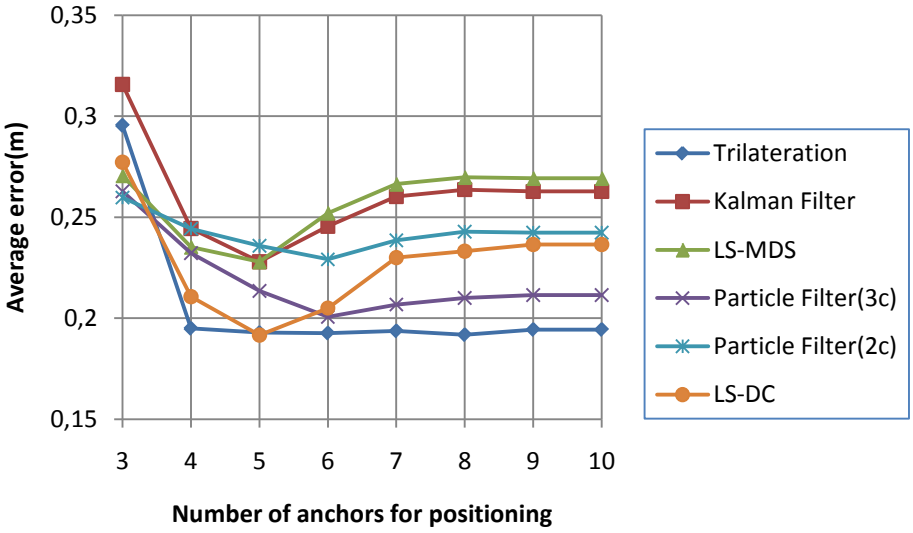

Figura C10. Error medio de posicionamiento. Distancia entre anchors = 10 m. σ_n = 0.3.

El comportamiento de todos los algoritmos es mejor que en la situación de utilizar la misma configuración para un error residual de ranging de 0.7. Como ya se vio en el modelo estadístico del simulador, como el error residual ha disminuido, las desviaciones del ranging tienen mayor importancia y por ello los anchors cercanos proporcionan estimaciones mucho más precisas que anchors alejados. Debido a esto, como la trilateración solo utiliza los tres anchors más cercanos, es el que mejor resultados ofrece para 4 o más anchors, ya que los anchors que se añaden proporcionan estimaciones mucho peores y hacen que el error de posicionamiento aumente, incluso para el filtro de partículas.

Los algoritmos LS-MDS, LS-DC y filtro de Kalman tienen una evolución similar, aunque el LS-DC ofrece mejores resultados. El número óptimo de anchors para los tres algoritmos es 5. Para los dos modelos del filtro de partículas el número óptimo de anchors es 6.

La figura C11 muestra el error medio de posicionamiento para una distancia entre anchors de 12.5 m y un error residual de $\sigma_n = 0.3$.

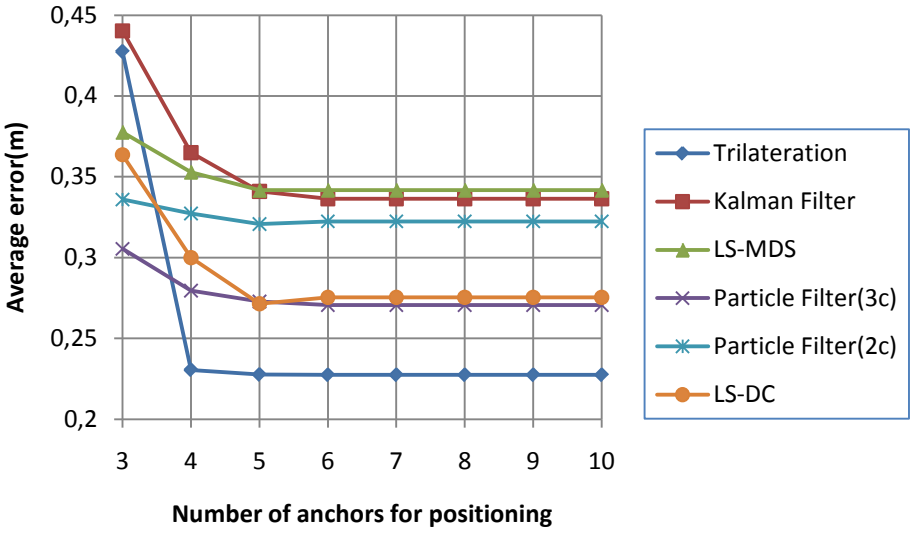

Figura C11. Error medio de posicionamiento. Distancia entre anchors = 12.5 m. σ_n = 0.3.

Ahora la distancia entre anchors es mayor, así que al añadir al cálculo de la posición anchors alejados, el error de posicionamiento empeora aún más que en la situación anterior, por ello la trilateración vuelve a ser el algoritmo que mejores resultados obtiene para 4 anchors o más. El número óptimo de anchors para el LS-MDS y el LS-DC es 5 y ambos algoritmos presentan una evolución similar, siendo el LS-DC el que mejores resultados ofrece, ya que el DC funciona mejor para distancias entre anchors elevadas. El número óptimo de anchors para los dos modelos del filtro de partículas y para el filtro de Kalman es 6.

C.1.3 Filtro de partículas

Como se ha comentado anteriormente, los parámetros del filtro de partículas tanto para el modelo de 3 componentes como para el de 2 componentes se optimizaron mediante simulaciones. Sin embargo, en un sistema real los parámetros se fijarán mediante una fase de calibración que no ofrecerá resultados tan óptimos como mediante simulación. Por ello, en este apartado se van a comparar los resultados obtenidos por el filtro de partículas de 2 componentes optimizado, con el filtro de partículas de 2 componentes sin optimizar. La optimización se realizó para situaciones con un error residual de ranging de $\sigma_n = 0.7$. En las figuras C12 y C13 se muestran los resultados para dos configuraciones, distancia entre anchors de 10 y 12.5 m, del modelo basado en paredes y rutas del simulador en función del número de anchors utilizados para el cálculo de la posición.

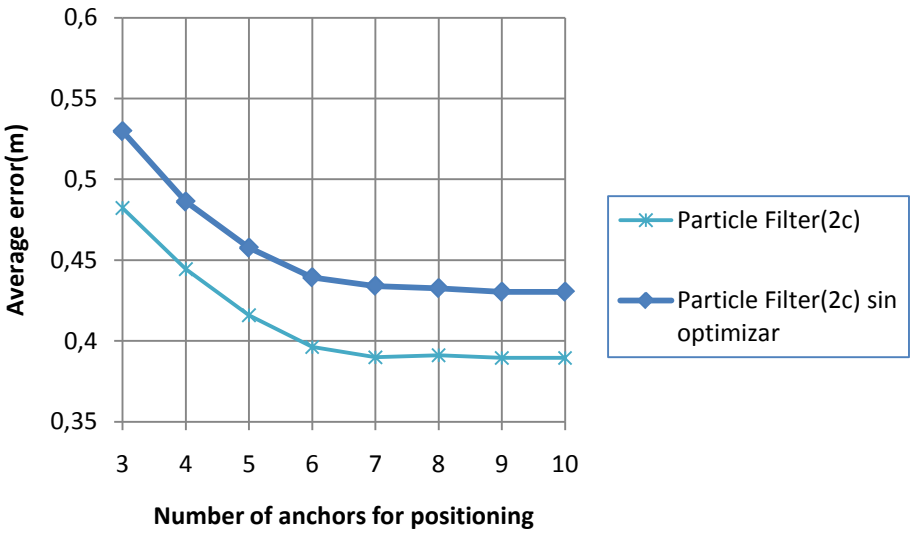

Figura C12. Error medio de posicionamiento. Distancia entre anchors = 10 m. σ_n = 0.7.

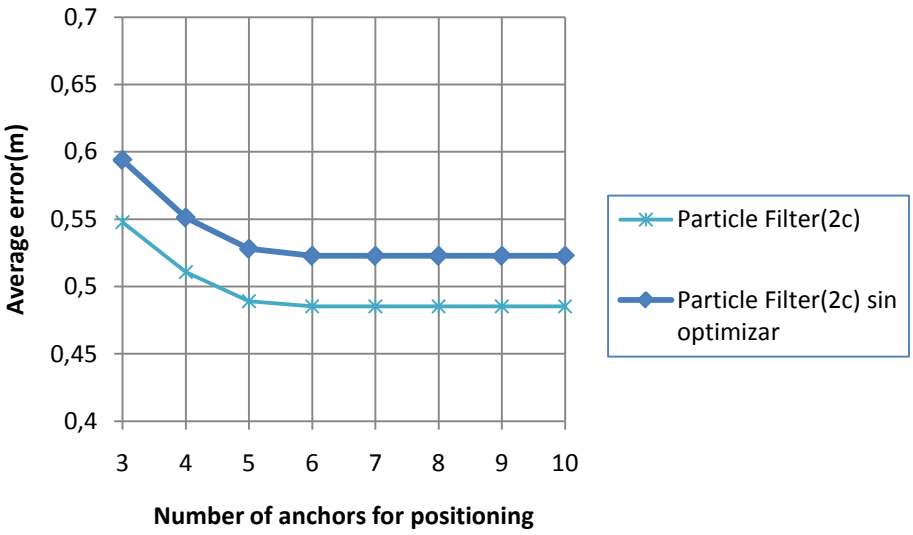

Figura C13. Error medio de posicionamiento. Distancia entre anchors = 12.5 m. σ_n = 0.7.

En las dos configuraciones la evolución de ambos filtros de partículas es la misma, pero el optimizado obtiene mejores resultados. Se puede observar que al aumentar la distancia entre los anchors el error aumenta, pero la diferencia entre los errores del filtro optimizado y del filtro sin optimizar se mantiene constante en torno a los 4 cm.

Por lo tanto, para disponer de un filtro de partículas perfectamente optimizado, se deberán optimizar los parámetros específicamente para el escenario en el que se pretenda utilizar. Como ya se comentó, para conseguir una buena optimización, se requerirían unas fases de medida y de calibración muy costosas. Y si se utiliza un modelo genérico sin optimizar, se pierde precisión respecto a un modelo optimizado como se ha observado anteriormente con las figuras C12 y C13.

C.2 Prefiltrado

En este punto se van a mostrar los resultados del prefiltrado de las estimaciones para la configuración de 10 m entre anchors y error residual de ranging $\sigma_n = 0.7$. Para evaluar el comportamiento de esta mejora se van a comparar diferentes estrategias de filtrado en función de la velocidad del target. Los resultados se muestran en la figura C14.

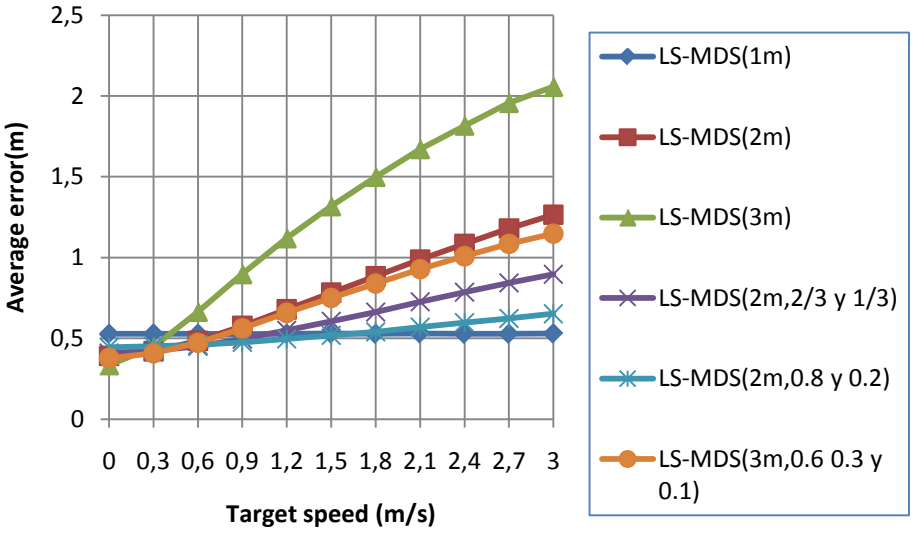

Figura C14. Error medio de posicionamiento. Distancia entre anchors =10 m. σ_n = 0.7.

En primer lugar se va a explicar la notación de la leyenda de la figura C14. El término 1m, 2m o 3m, se refiere al número de muestras que se utiliza para el filtrado. El caso 1m, significa que unicamente se utiliza la muestra actual, el caso 2m quiere decir que se utilizan la muestra actual y la muestra anterior, y el caso 3m significa que se utiliza la muestra actual y las dos anteriores. Si solo se nombra con 2m o 3m, quiere decir que se realiza la media entre las muestras. Mientras que si por ejemplo también se nombra con un 2/3 y 1/3, quiere decir que la muestra actual se pondera por 2/3 y la muestra anterior por 1/3.

Si el target se mueve a velocidades bajas, realizar un prefiltrado es beneficioso, llegando a reducir el error obtenido por el LS-MDS convencional, mientras que para velocidades altas, el prefiltrado empeora la situación, ya que hay mucha diferencia de distancia entre una muestra y la siguiente si el target se mueve muy rápido.

Una opción que funciona bastante bien es utilizando dos muestras y ponderando la actual por 0.8 y la anterior por 0.2, ya que obtiene mejores resultados para velocidades inferiores a 1.5 m/s. La opción de utilizar dos muestras pero ponderarlas con 2/3 y 1/3, obtiene mejores errores que la de 0.8 y 0.2 únicamente para velocidades inferiores a 0.4 m/s. Para velocidades superiores, el hecho de dar más valor a la muestra anterior hace que su comportamiento empeore.

Las opciones de realizar una media con dos muestras o utilizar tres muestras ponderandolas por 0.6, 0.3 y 0.1 obtienen resultados muy similares para velocidades inferiores a 1.5 m/s. Mientras que para velocidades superiores, la opción de dos muestras empeora ya que da menos peso a la muestra actual que la opción de tres muestras.

La peor opción es realizar la media con tres muestras, ya que la muestras actual y la muestra de dos iteraciones antes es muy diferente incluso para velocidades bajas. Y eso se ve en los resultados, ya que para velocidades superiores a 0.3 m/s el error medio se dispara.

Como conclusión podemos decir que el prefiltrado puede resultar beneficioso para velocidades inferiores a 1 m/s. Mientras que para velocidades superiores, una muestra y la

siguiente se diferencian demasiado, provocando un aumento del error medio de posicionamiento.

C.3 Ponderación

En este apartado se va a comprobar el funcionamiento de la adaptación de los algoritmos propuestos en PULSERS PHASE II (WLS-MDS) y EUWB (WLS-DC).

En la figura C15 aparecen los resultados para una distancia de 10 m entre anchors y un error residual de ranging de $\sigma_n = 0.7$.

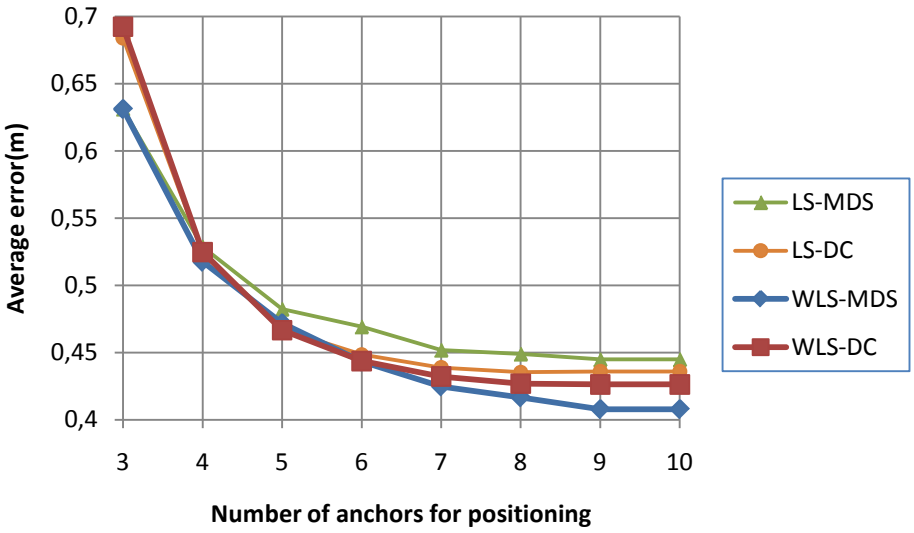

Figura C15. Error medio de posicionamiento. Distancia entre anchors = 10 m. σ_n = 0.7.

Primero vamos a analizar el algoritmo LS-MDS. La adaptación del algoritmo PULSERS PHASE II (WLS-MDS) funciona bastante bien para más de 4 anchors, consiguiendo reducir el error medio de posicionamiento mediante la ponderación de las distancias estimadas en base a su variabilidad. Los anchors que se añaden a partir de 4 anchors, son anchors alejados y por lo tanto sus estimaciones tienen una mayor variabilidad que anchors cercanos, por ello la ponderación es capaz de compensar esa variabilidad y mejorar la precisión.

Ahora analizamos el algoritmo LS-DC. La adaptación del algoritmo propuesto en EUWB (WLS-DC), obtiene los mismos resultados que el LS-DC salvo para más de 6 anchors que obtiene unos errores medios algo inferiores. En este algoritmo la ponderación de las distancias no funciona tan bien como para el caso del LS-MDS.

Aunque para esta configuración el algoritmo DC mejoraba respecto al uso únicamente del MDS (LS-MDS), los pesos se comportan mejor para el LS-MDS haciendo que el algoritmo que mejores resultados obtiene es el WLS-MDS.

En la figura C16, se pueden observar los resultados si se aumenta la distancia entre anchors a 12.5 m y se mantiene el error residual de ranging.

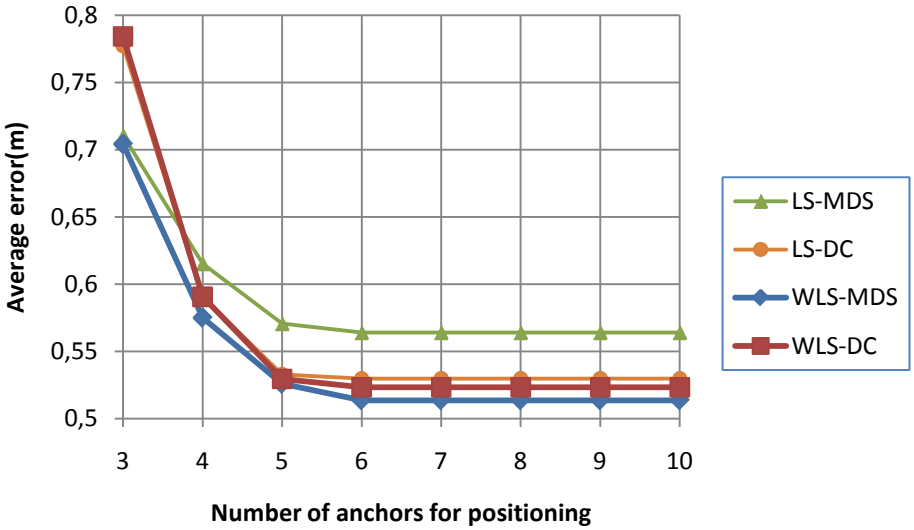

Figura C16. Error medio de posicionamiento. Distancia entre anchors = 12.5 m. σ_n = 0.7.

Al aumentar la distancia entre anchors todos los algoritmos se comportan peor, como era de esperar. El algoritmo WLS-MDS se comporta mejor que el algoritmo LS-MDS. A una distancia entre anchors tan grande, no es probable que el target esté en cobertura de más de 6 anchors, por eso el error se mantiene constante para más de 6 anchors.

El LS-DC y el WLS-DC obtienen los mismos errores para menos de 6 anchors, y para 6 o más el WLS-DC obtiene unos errores levemente inferiores.

Resumiendo se puede decir que la ponderación de las estimaciones en base a su variabilidad ha funcionado correctamente, sobre todo en el caso del algoritmo LS-MDS. La ponderación ofrece mejores resultados para distancias entre anchors elevadas, ya que la variabilidad de las estimaciones es mayor. Ahora vamos a observar los resultados para las dos configuraciones, pero reduciendo el error residual de ranging a $\sigma_n = 0.3$. En la figura C17 aparecen los resultados para una distancia entre anchors de 10 m.

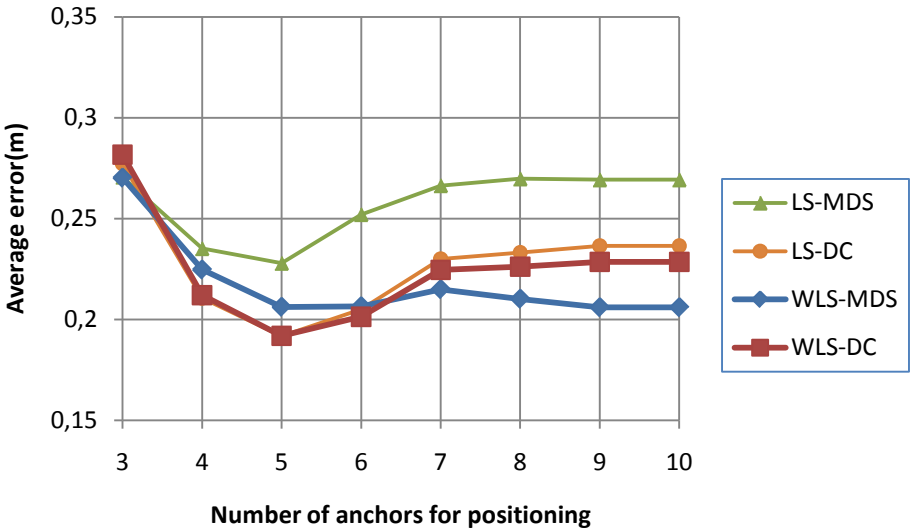

Figura C17. Error medio de posicionamiento. Distancia entre anchors = 10 m. σ_n = 0.3.

Al reducirse el error residual, las desviaciones del ranging son más importantes y los anchors cercanos ofrecen estimaciones mucho más precisas que los anchors alejados. Todos los algoritmos se comportan mejor que en las configuraciones con error residual de 0.7. El número óptimo de anchors para todos los algoritmos es 5. El WLS-MDS consigue reducir bastante el error respecto al LS-MDS.

El WLS-DC obtiene los mismos errores que el LS-DC para 5 o menos anchors, mientras que para más de 5 obtiene errores levemente inferiores.

En la figura C18 aparecen los errores para una distancia entre anchors de 12.5 y un error residual de ranging de $\sigma_n = 0.3$.

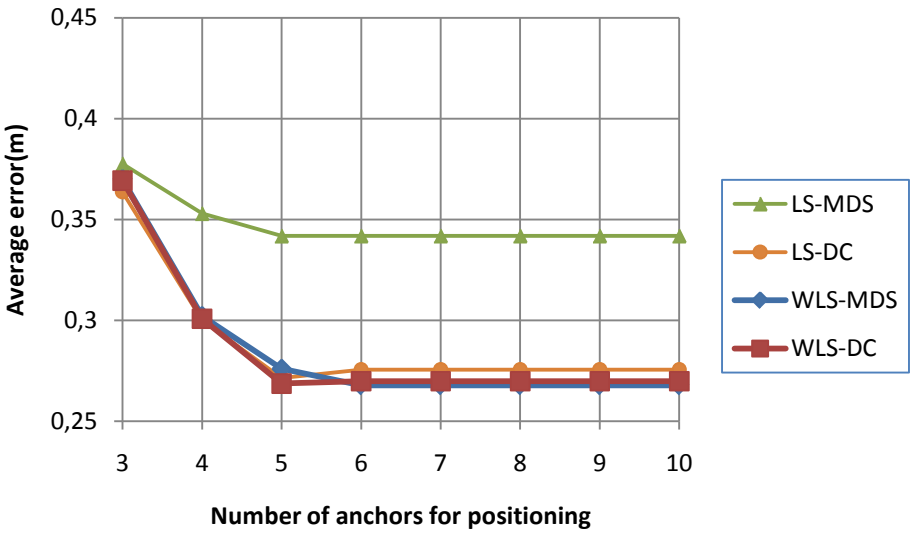

Figura C18. Error medio de posicionamiento. Distancia entre anchors = 12.5 m. σ_n = 0.3.

Como la distancia entre anchors es mayor, la variabilidad de las estimaciones es mayor y el WLS-MDS consigue reducir mucho el error de posicionamiento del LS-MDS. Como la distancia entre anchors es elevado, el LS-DC obtiene buenos resultados respecto al LS-MDS. EL WLS-DC obtiene los mismos errores que el LS-DC para 5 anchors o menos, y errores levemente inferiores para más de 5 anchors.

C.4 Uso de información geográfica

C.4.1 WLS-MDS y WLS-DC

En este apartado se va a comprobar el funcionamiento de la identificación de situaciones de NLOS para los algoritmos WLS-MDS y WLS-DC (WLS-MDS & NLOS y WLS-DC & NLOS). En estos algoritmos la identificación de situaciones NLOS consistía en comprobar si el enlace entre el target y los anchors utilizados estaba en situación de LOS o de NLOS comprobando si el enlace cruzaba alguna pared del plano. Si el enlace estaba en situación de NLOS, el peso correspondiente a ese enlace se multiplicaba por 0.5. Los algoritmos se van a comparar en función del número de anchors utilizados para estimar la posición, para dos planos en el modelo del simulador basado en paredes y rutas.

En la figura C19 aparecen los resultados para una distancia de 10 m entre anchors y un error residual de ranging de $\sigma_n = 0.7$.

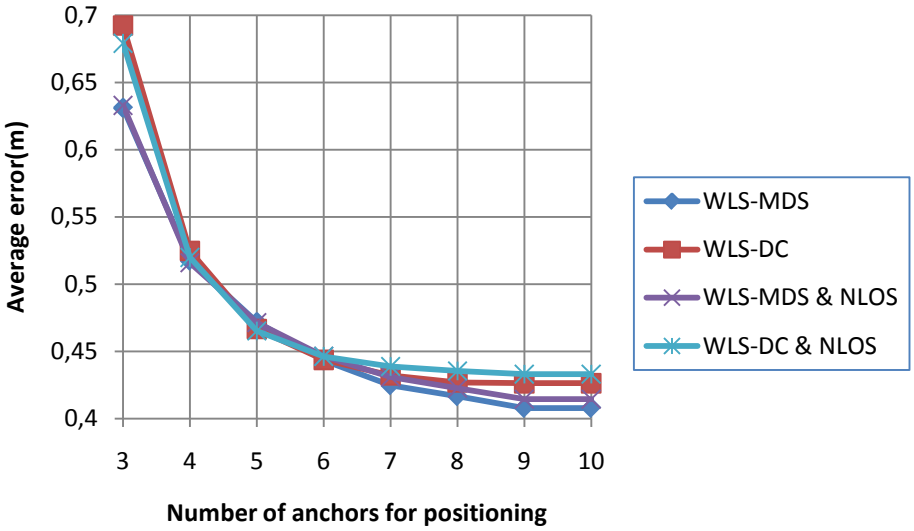

Figura C19. Error medio de posicionamiento. Distancia entre anchors = 10 m. σ_n = 0.7.

El algoritmo WLS-MDS & NLOS sigue la misma evolución que el WLS-MDS, obteniendo los mismos errores para 6 o menos anchors, aunque obtiene errores mayores para más de 6 anchors. La modificación de la identificación de situaciones NLOS no funciona como se esperaba, ya que se esperaba que funcionase un poco mejor que el WLS-MDS y es debido a que con la ponderación ya se está dando menos peso a situaciones de NLOS, ya que estas presentan mayor variabilidad que las situaciones de LOS, y al multiplicar el peso por 0.5, se está reduciendo el peso de las situaciones NLOS en exceso, provocando ese aumento del error respecto al WLS-MDS para un número de anchors superior a 6.

En cuanto al algoritmo WLS-DC & NLOS, tiene la misma evolución que el WLS-DC, obteniendo los mismos errores para 6 o menos anchors y errores algo superiores para más de 6 anchors. Ocurre lo mismo que para el caso del WLS-MDS, y es que al aumentar el número de anchors e introducir anchors que están en situaciones de NLOS, se reduce en exceso el valor del peso al multiplicarlo por 0.5 y se empeora el error medio respecto al WLS-DC.

En la figura C20, se pueden observar los resultados si se aumenta la distancia entre anchors a 12.5 m y se mantiene el error residual de ranging.

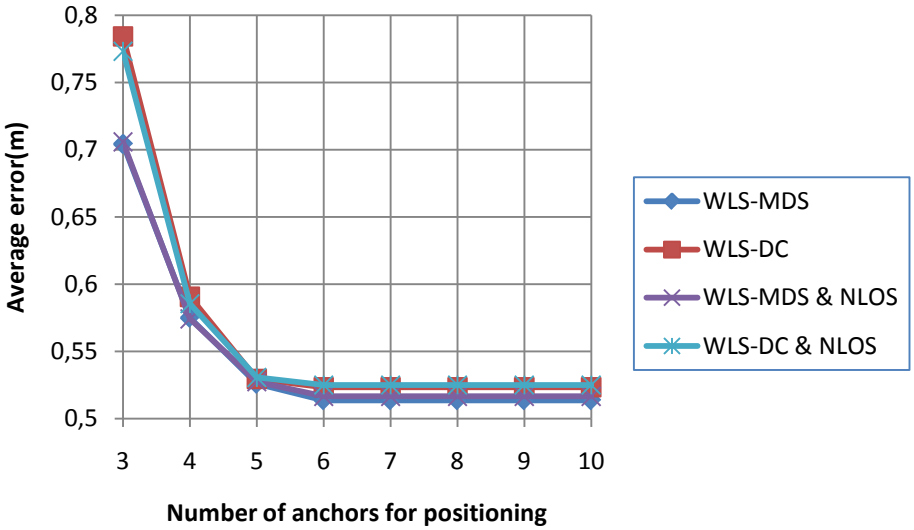

Figura C20. Error medio de posicionamiento. Distancia entre anchors = 12.5 m. σ_n = 0.7.

Al aumentar la distancia entre anchors todos los algoritmos se comportan peor, como era de esperar. El algoritmo WLS-MDS & NLOS obtiene prácticamente los mismos errores que el WLS-MDS. A una distancia entre anchors tan grande, no es probable que el target esté en cobertura de más de 6 anchors, por eso el error se mantiene constante para más de 6 anchors y por eso, al no tener en cobertura muchos anchors en situación de NLOS, los algoritmos WLS-MDS y WLS-MDS & NLOS tienen un comportamiento tan parecido. El WLS-DC y el WLS-DC & NLOS obtienen los mismos errores.

A modo de conclusión se puede decir que la identifiación de situaciones NLOS no ha funcionado como esperabamos, debido a que al multiplicar por 0.5 en los enlaces en NLOS se reduce en exceso el peso de estos enlaces. Ahora vamos a observar los resultados para las dos configuraciones, pero reduciendo el error residual de ranging a $\sigma_n = 0.3$. En la figura C21 aparecen los resultados para una distancia entre anchors de 10 m.

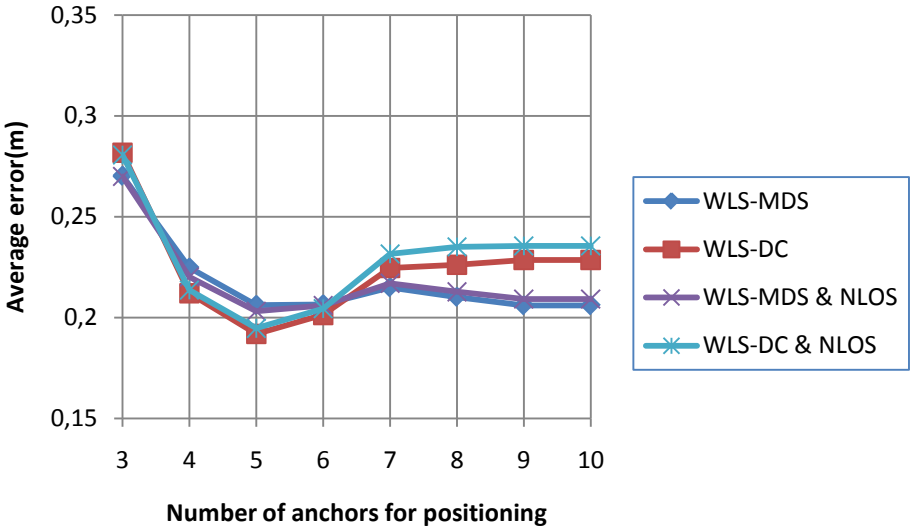

Figura C21. Error medio de posicionamiento. Distancia entre anchors = 10 m. σ_n = 0.3.

Al reducirse el error residual, las desviaciones del ranging son más importantes y los anchors cercanos ofrecen estimaciones mucho más precisas que los anchors alejados. Todos

los algoritmos se comportan mejor que en las configuraciones con error residual de 0.7. El número óptimo de anchors para todos los algoritmos es 5. El WLS-MDS obtiene errores practicamente idénticos a los obtenidos por el WLS-MDS & NLOS.

El WLS-DC & NLOS obtiene los mismos errores que el WLS-DC para 6 o menos anchors, mientras que para más de 6 obtiene errores levemente superiores.

En la figura C22 aparecen los errores para una distancia entre anchors de 12.5 y un error residual de ranging de $\sigma_n = 0.3$.

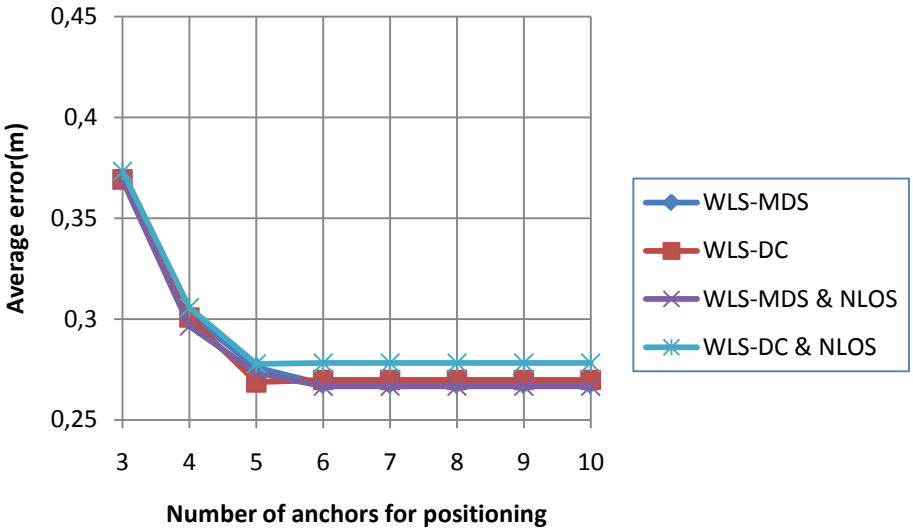

Figura C22. Error medio de posicionamiento. Distancia entre anchors = 12.5 m. σ_n = 0.3.

El WLS-MDS & NLOS obtiene errores idénticos a los obtenidos por el WLS-MDS. EL WLS-DC & NLOS obtiene los mismos errores que el WLS-DC para 4 anchors o menos, y errores algo superiores para más de 4 anchors.

C.4.2 Filtro de partículas

Ahora vamos a comparar los resultados obtenidos con el filtro de partículas de 3 componentes del modelo basado en paredes y rutas del simulador, con los obtenidos con detección de situaciones NLOS y utilización de información acerca de las rutas de los targets. La detección de situaciones NLOS en el filtro de partículas consistía en modificar el modelo de cálculo de la probabilidad de las partículas. Sin detección, los pesos LOS, NLOS y NLOS2 (severe NLOS), se calculan de forma estadística, mientras que con detección de situaciones NLOS, si no existen paredes en el enlace, el peso LOS toma valor uno, si hay una pared toma valor uno el peso NLOS y si hay dos o más paredes toma valor uno el peso NLOS2.

La utilización de información acerca de las rutas consistía en dos modificaciones. La primera consistía en dividir la probabilidad de la partícula entre 6 si estaba fuera de las rutas posibles. La segunda modificación consistía en que cuando el target se encontrase en un nodo, se modificaban las direcciones del 80% de las partículas de acuerdo a las probabilidades determinadas en el fichero likelihood para seguir recto, girar a la derecha, girar a la izquierda o dar media vuelta.

En la figura C23 se observan los resultados para una distancia entre anchors de 10 m y un error residual de ranging de 0.7. El algoritmo Particle Filter & NLOS & routes es el filtro de partículas con detección de situaciones NLOS y utilización de información acerca de las rutas.

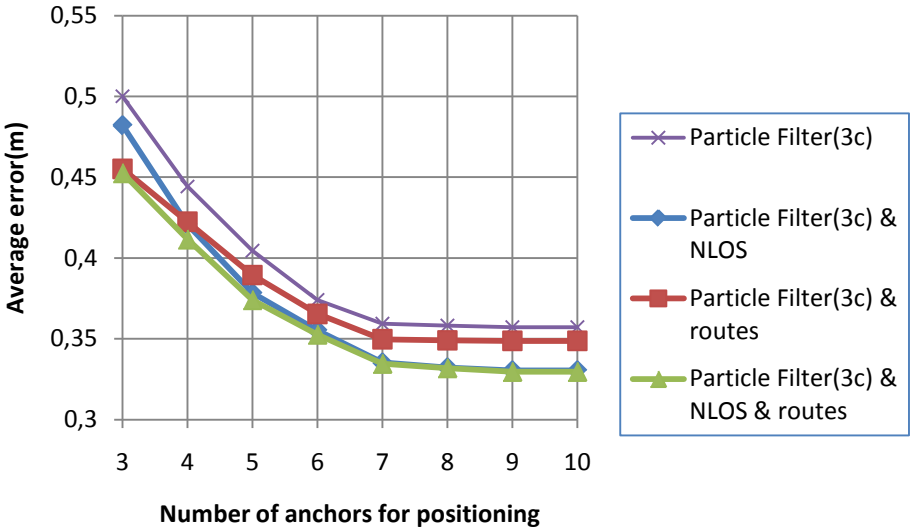

Figura C23. Error medio de posicionamiento. Distancia entre anchors = 10 m. σ_n = 0.7.

Los cuatro algoritmos siguen una evolución similar y es el filtro de partículas con ambas modificaciones el que mejor resultados ofrece. Viendo los resultados se puede concluir que la modificación que más mejora la precisión es la detección de situaciones NLOS, ya que obtiene errores inferiores para más de 3 anchors, que la utilización de información de las rutas. Además, para más de 5 anchors el filtro con detección de situaciones NLOS obtiene los mismos resultados que el filtro que combina ambas modificaciones. Es lógico que la detección de situaciones NLOS funcione mejor para un número de anchors elevado, ya que cuantos más anchors se utilicen, mayor es la probabilidad de que haya varios enlaces en situación de NLOS.

En la figura C24 aparecen los resultados si se aumenta la distancia entre anchors a 12.5 m.

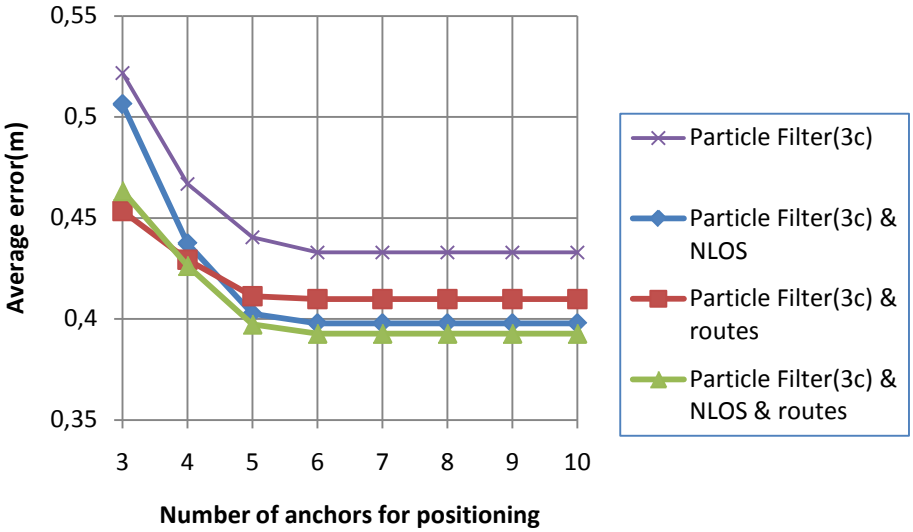

Figura C24. Error medio de posicionamiento. Distancia entre anchors = 12.5 m. σ_n = 0.7.

Al aumentar la distancia entre los anchors, los resultados obtenidos empeoran, ya que la variablidad de las estimaciones es mayor. La mejor opción es nuevamente el filtro de partículas que combina ambas modificaciones. La modificación de detección de NLOS funciona mejor que la utilización de información de las rutas para más de 4 anchors. El filtro con ambas modificaciones implementadas mejora la precisión respecto al filtro convencional aproximadamente en unos 4 cm, mientras que para la situación anterior, con una distancia de 10 m, mejoraba aproximadamente unos 3 cm.

Ahora se van a analizar los resultados para las dos configuraciones anteriores, pero reduciendo el error residual de ranging a 0.3. En la figura C25 se muestran los resultados para una distancia de 10 m entre anchors.

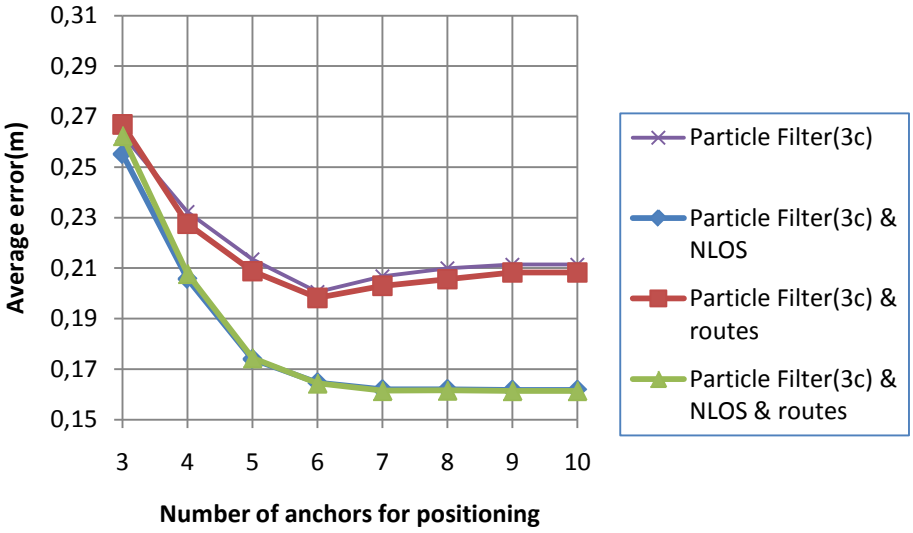

Figura C25. Error medio de posicionamiento. Distancia entre anchors = 10 m. σ_n = 0.3.

Como el error residual es inferior, los resultados obtenidos son mejores que en las situaciones anteriores. Como ahora las estimaciones de los anchors cercanos son mucho más precisas que las de los anchors alejados, el número óptimo de anchors para el filtro convencional y para el filtro con la modificación de las rutas es de 6, mientras que para los otros dos filtros es de 7. La modificación de las rutas apenas afecta, ya que los errores obtenidos con la modificación son prácticamente idénticos a los obtenidos con el filtro convencional.

Los mejores resultados se obtienen con el filtro con la modificación de NLOS y con el filtro que tiene implementadas ambas. Ambos filtros obtienen los mismos errores, ya que la modificación de rutas no afecta prácticamente. Cuanto mayor es el número de anchors, mejor funciona la modificación de NLOS, ya que es más probable que haya varios enlaces en condiciones de NLOS.

En la figura C26, se aumenta la distancia entre anchors a 12.5 m, manteniendo el error residual de ranging.

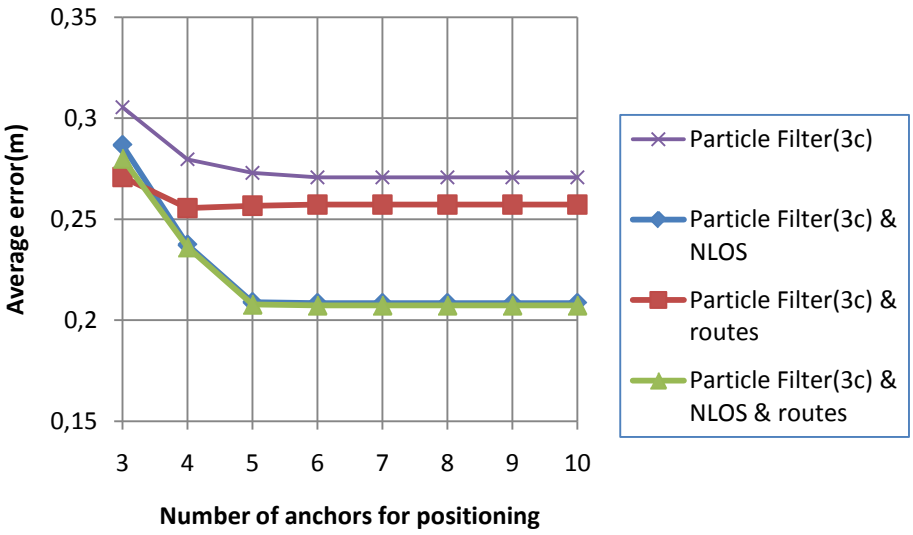

Figura C26. Error medio de posicionamiento. Distancia entre anchors = 12.5 m. σ_n = 0.3.

Como ya vimos en las anteriores situaciones, al aumentar la distancia entre anchors, las modificaciones tienen un mejor comportamiento. La modificación de las rutas consigue mejorar el comportamiento del filtro convencional, a diferencia de la situación anterior, con una distancia de 10 m. Los mejores resultados se obtienen nuevamente con el filtro con detección de NLOS y con el filtro con ambas modificaciones, los cuales obtienen los mismos errores. La única situación en la que no son la mejor opción es para 3 anchors, en la que el filtro con la modificación de las rutas obtiene un error algo inferior.

C.4.3 Filtro de Kalman

En este punto se van a observar los resultados de las modificaciones de detección de situaciones NLOS y de utilización de información acerca de las rutas de los targets para el filtro de Kalman. La detección de situaciones NLOS consiste en multiplicar por 1.5 el valor de covarianza cuando se construye la matriz de covarianza del ruido de medida R. En cuanto a la utilización de información acerca de las rutas, consiste en que cuando el target está en un nodo, se multiplica el valor de la aceleración por 5 cuando se construye la matriz Q. Se van a analizar dos configuraciones del modelo basado en planos y rutas del simulador, en función del número de anchors utilizados para el cálculo de la posición. En la figura C27 se muestran los resultados obtenidos para una distancia de 10 m entre anchors y un error residual de ranging de $\sigma_n = 0.7$.

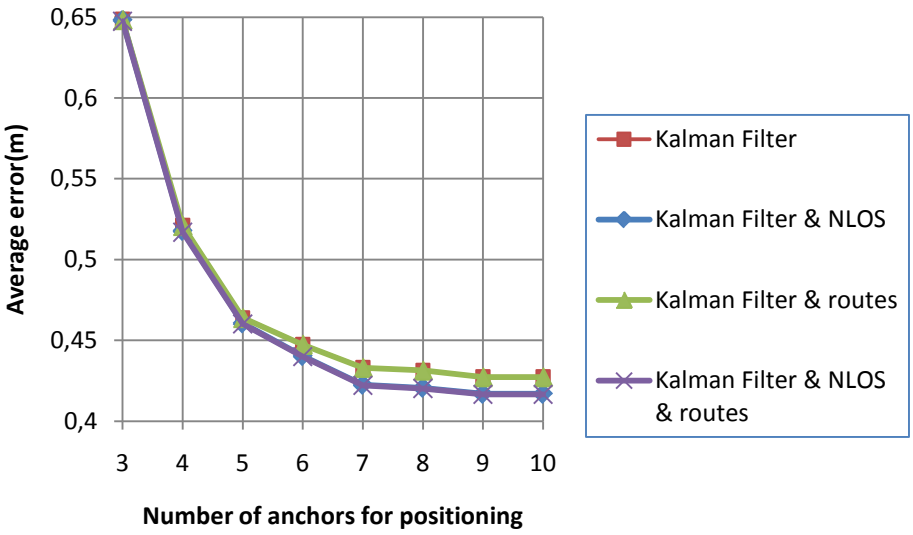

Figura C27. Error medio de posicionamiento. Distancia entre anchors =10 m. σ_n = 0.7.

Los cuatro algoritmos siguen una evolución similar, aunque para más de 5 anchors, el filtro con la modificación de detección de NLOS y el filtro con ambas modificaciones consiguen errores más pequeños. La modificación de las rutas no modifica apenas el funcionamiento del filtro de Kalman, ya que obtiene prácticamente los mismos errores que el filtro de Kalman convencional. La modificación de la detección de situaciones NLOS funciona mejor al aumentar el número de anchors, ya que aumenta el número de anchors que se encuentran en situación de NLOS. Como la modificación de las rutas apenas modifica el funcionamiento, el filtro con la modificación de NLOS obtiene los mismos resultados que el filtro con ambas modificaciones.

En la figura C28 aparecen los resultados obtenidos al aumentar la distancia ente anchors a 12.5 m, manteniendo el error residual de ranging.

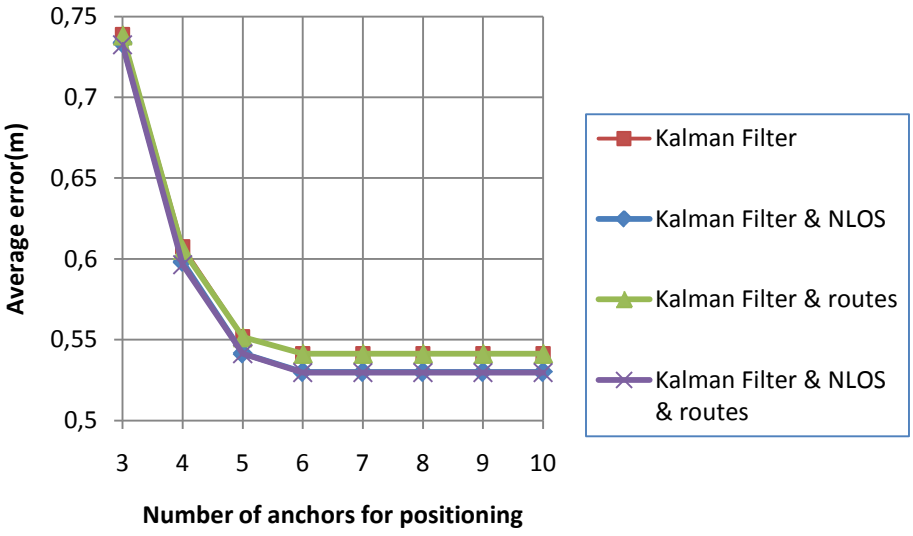

Figura C28. Error medio de posicionamiento. Distancia entre anchors =12.5 m. σ_n = 0.7.

La situación es idéntica a la configuración anterior, pero al aumentar la distancia entre anchors los errores obtenidos son mayores. Además, la modificación de detección de NLOS mejora el resultado del filtro convencional para más de 4 anchors, en lugar de para más de 5

anchors como en la configuración anterior. La modificación de las rutas tampoco aporta nada en esta configuración.

A continuación se presentan los resultados obtenidos para las dos configuraciones, pero disminuyendo el error residual de ranging. En la figura C29 se muestran los resultados para una distancia entre anchors de 10 m y un error residual de 0.3.

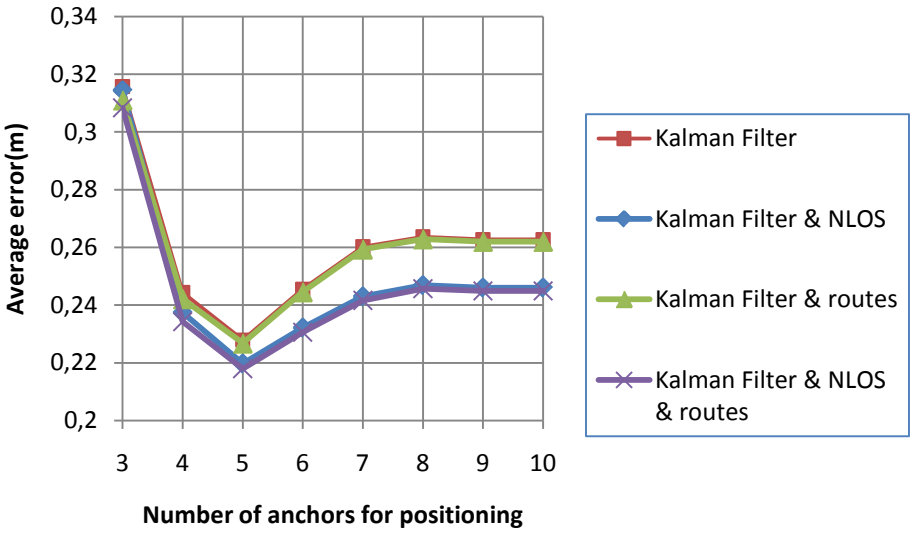

Figura C29. Error medio de posicionamiento. Distancia entre anchors =10 m. σ_n = 0.3.

Como se ha reducido el nivel de error residual, los anchors cercanos ofrecen estimaciones mucho más precisas que los anchors lejanos, por ello el número óptimo de anchors es 5 para todos los algoritmos, ya que añadiendo más de 5 anchors, se añaden anchors alejados y sus estimaciones tienen una gran desviación de ranging, provocando que el error medio de posicionamiento aumente. Al igual que en la misma configuración para error residual de 0.7 (figura C27), la modificación de las rutas no mejora el funcionamiento del filtro de Kalman convencional, obteniendo los mismos errores. La modificación de la detección de situaciones NLOS mejora el error del filtro de Kalman convencional para más de 3 anchors.

En la figura C30, pueden observarse los resultados al aumentar la distancia entre anchors a 12.5 m.

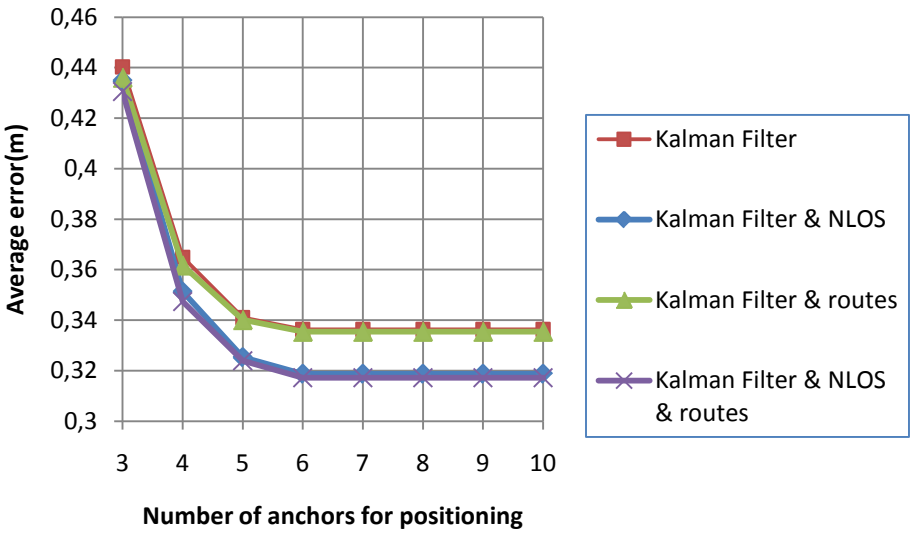

Figura C30. Error medio de posicionamiento. Distancia entre anchors =12.5 m. σ_n = 0.3.

Al aumentar la distancia entre anchors el error medio ha aumentado, pero al igual que en la situación anterior, la modificación de las rutas no mejora el error, mientras que la modificación de NLOS consigue mejorar el error del filtro de Kalman convencional para más de 3 anchors. Como la modificación de las rutas apenas afecta al funcionamiento, el filtro de Kalman con ambas modificaciones, obtiene los mismos errores que el filtro con detección de situaciones de NLOS.

A modo de resumen, se puede concluir que la mejora de utilización de información acerca de las rutas apenas modifica el funcionamiento del Filtro de Kalman y no consigue mejorar el error. Sin embargo, la modifiación de detección de situaciones NLOS consigue mejorar el funcionamiento del filtro de Kalman para las configuraciones de 10 y 12.5 m. Esta modificación funciona mejor para un número elevado de anchors. El filtro con ambas modificaciones implementadas obtiene los mismos resultados que el filtro con detección de situaciones NLOS, lo cual es lógico, ya que la modificación de las rutas no produce cambios en el comportamiento del filtro convencional.