Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería Industrial.

Máster Universitario en Automática y Robótica

Trabajo Guiado y Navegación de Robots

Localización y planificación de trayectorias

Guillermo Illana Gisbert
Alberto López Rodríguez
Pablo García Peris

Madrid, Enero de 2022

Índice

1. Aplicación escogida

La aplicación escogida es la de utilizar un robot para el transporte de objetos entre estancias de una misma casa. Este robot tiene como objetivo ayudar a personas con movilidad reducida a poder desenvolverse mejor en su propia casa, trayéndole los objetos que necesite en ese momento, como podría ser comida, ropa u otras necesidades.

El robot utilizado en el presente trabajo es un Pioneer 3-AT. Se trata de un pequeño robot de cuatro ruedas y cuatro motores ideal para para el funcionamiento en todo tipo de terrenos. Entre las características más destacadas del robot Pioneer 3-AT se encuentran un interruptor de parada de emergencia, codificadores de rueda y un microcontrolador con firmware ARCOS, así como el paquete de desarrollo de software Pioneer SDK. El robot utilizado puede verse en la figura 1.

Figura 1: Robot Pioneer 3-AT

Al robot base se le añade un total de 5 sensores de ultrasonidos instalados rodeando su estructura para conocer la distancia entre el robot y las paredes que le rodean y un telémetro láser utilizado para el conocimiento del ángulo del robot respecto a balizas instaladas en el entorno del mismo.

El entorno utilizado en el trabajo es una casa con 10 balizas de localización que pudo ser trasladada al software Apolo tras la realización de numerosas mediciones. El entorno integrado en Apolo se encuentra a una escala 2:1 respecto al entorno real. En la figura 2 puede observarse el entorno ya construido en Apolo. El mismo entorno fue reproducido en Matlab para realizar la estimación de las medidas de los sensores del robot en el mismo entorno. El entorno reproducido en Matlab puede observarse en la figura 3.

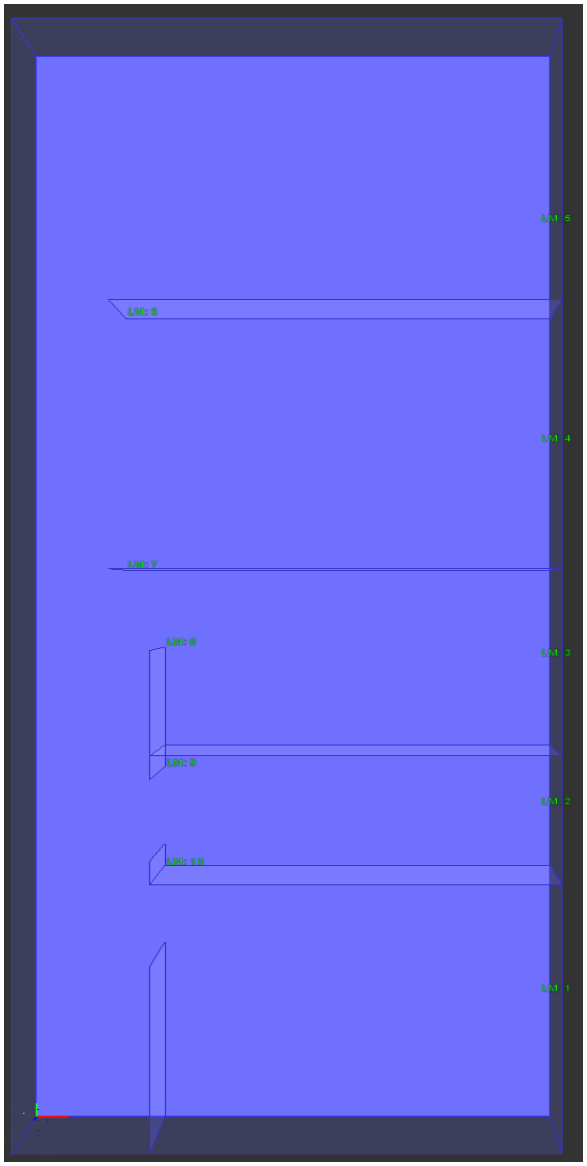

Figura 2: Representación del entorno de trabajo en Apolo con las balizas ya incorporadas (letras en verde).

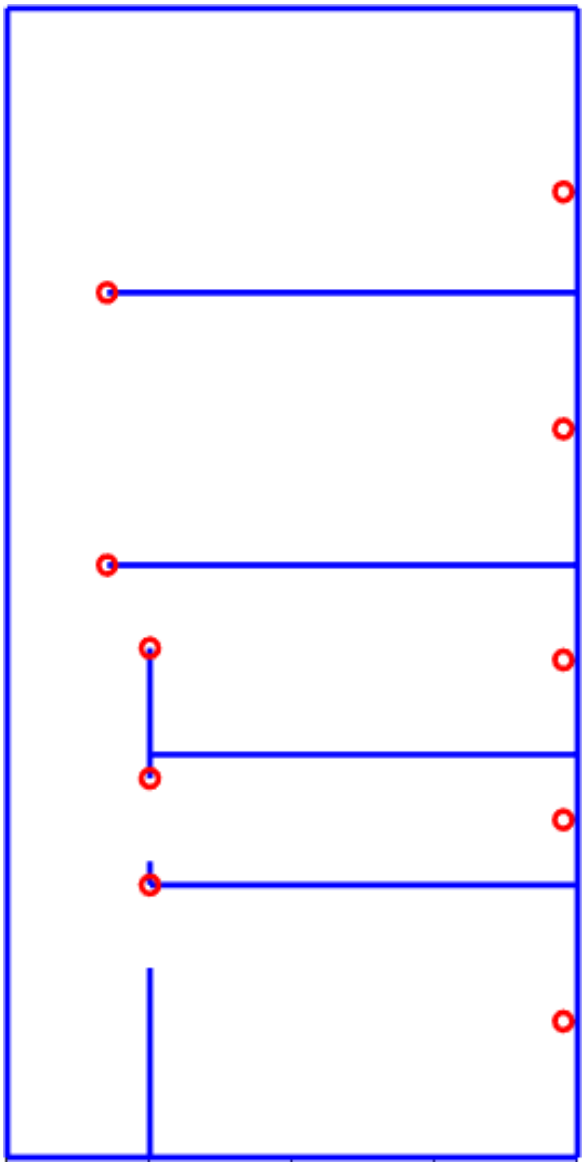

Figura 3: Representación del entorno de trabajo en Matlab con las balizas ya incorporadas (círculos rojos).

2. Calibración del sistema de locomoción

En esta sección, primero se detallarán las fórmulas utilizadas para la obtención de la posición estimada del robot en función de la odometría y, a continuación, los detalles de la matriz Q en función de las velocidades del robot.

2.1. Estimación de la posición en función de la odometría

La primera parte para calibrar el sistema de locomoción u odometría es saber cómo funciona.

Al obtener los valores de la odometría con *apoloGetOdometry*, la función devuelve tres valores: $\Delta x_o(k)$, $\Delta y_o(k)$ y $\Delta\theta_o(k)$, y no los posibles valores de la medición odométrica directa ($\Delta d_i(k)$, $\Delta d(k)$ o $\Delta\beta(k)$). De esta manera, no es necesario saber si se obtiene la medida mediante giro diferencial, o de Ackerman, o de alguna otra manera. Es más, el robot es capaz de girar sin necesidad de avance o retroceso, en el propio sitio.

Se comprueba experimentalmente que, si no se resetea la odometría (a cero) entre cada medición, se produce un error anormalmente elevado en la posición respecto a la posición real (obtenida con *apoloGetLocationMRobot*. Por este motivo, la mejor solución es resetear la odometría con *apoloResetOdometry(robotName, [0 0 0])* en cada iteración, y calcular la nueva posición del robot mediante las mediciones de la odometría. Las fórmulas que hemos obtenido y que parecen más acertadas al compararlas con la posición real son las siguientes:

$$\hat{x}(k+1) = \hat{x}(k) + \Delta x_o(k+1)cos(\theta(k)) - \Delta y_o(k+1)sin(\theta(k)) \tag{1}$$

$$\hat{y}(k+1) = \hat{y}(k) + \Delta x_o(k+1)sin(\theta(k)) + \Delta y_o(k+1)cos(\theta(k)) \tag{2}$$

$$\hat{\theta}(k+1) = \hat{\theta}(k) + \Delta\theta_o(k+1) \tag{3}$$

Donde $\Delta x_o(k+1)$ es el avance hacia delante del robot, $\Delta y_o(k+1)$ es el avance en la dirección perpendicular, y $\Delta\theta_o(k+1)$ es el giro del robot (yaw), todos éstos medidos por la odometría. Obsérvese cómo en estas ecuaciones se utiliza $\Delta\theta_0(k+1)$ en lugar de $\Delta\theta_0(k+1)/2$ (ecuaciones de la odometría vistas en clase), tal y como se ha comprobado experimentalmente. Matricialmente:

$$\begin{bmatrix} \hat{x}(k+1) \\ \hat{y}(k+1) \\ \hat{\theta}(k+1) \end{bmatrix} = \begin{bmatrix} \hat{x}(k) \\ \hat{y}(k) \\ \hat{\theta}(k) \end{bmatrix} + \begin{bmatrix} cos(\theta(k)) & -sin(\theta(k)) & 0 \\ sin(\theta(k)) & cos(\theta(k)) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \Delta x_o(k+1) \\ \Delta y_o(k+1) \\ \Delta\theta_o(k+1) \end{bmatrix} \tag{4}$$

2.2. Calibración y obtención de la matriz Q

Como es lógico, la varianza de las mediciones odométricas dependen de la velocidad del robot; cuanto más rápido avance o gire, mayor será el error acumulado.

Para obtener la varianza de la odometría, se mueve al robot 100 pasos con una velocidad lineal v y una velocidad angular w y se anota la medición de la odometría (Δx_o, Δy_o y $\Delta\theta_o$), tomándose la varianza S de cada magnitud. A continuación, se varía el valor de (v) y/o (w), y se vuelven a tomar 100 medidas y a calcular la varianza. Este proceso se repite para varias velocidades y velocidades angulares, buscando la relación entre v y w y las varianzas.

No es sorprendente que la desviación típica (raíz cuadrada de la varianza) de Δx_o es proporcional a la velocidad v, así como la de $\Delta\theta_o$ lo es de la velocidad angular w. Para la variable Δy_o es más complicado prever de qué depende exactamente y, tras una serie de experimentos, se descubre que se modela muy bien como directamente proporcional al producto de sus velocidades $v \cdot w$, de manera que, elevando al cuadrado, se obtienen las siguientes relaciones:

$$Var(\Delta x_o(k)) = q_{11} \cdot v^2(k) \tag{5}$$
$$Var(\Delta y_o(k)) = q_{22} \cdot (v(k)w(k))^2 \tag{6}$$
$$Var(\Delta\theta_o(k)) = q_{33} \cdot w(k)^2 \tag{7}$$
$$Cov(\Delta x_o(k), \Delta y_o(k)) = \sqrt{Var(\Delta x_o(k)), Var(\Delta y_o(k))} = q_{12} \cdot v^2(k)|w(k)| \tag{8}$$

Mediante el método de regresión lineal (mínimos cuadrados) es posible obtener estos coeficientes q_{ii}, que modelan bastante bien las varianzas. Las covarianzas se consideran nulas en general, salvo la correspondiente a Δx_o y Δx_o. Estas dos variables se comprueba que guardan un coeficiente de correlación unitario, es decir, mantienen una correlación directa y la covarianza es la media geométrica de las varianzas individuales. Todos los valores de q_{ij} se guardan en una matriz Q_{pu}. Una vez calculados estos coeficientes, la matriz Q de varianzas y covarianzas del ruido del proceso se puede representar como:

$$Q(k) = \begin{bmatrix} 2{,}3585 \cdot 10^{-7} v^2(k) & 2{,}0758 \cdot 10^{-9} v^2(k)|w(k)| & 0 \\ 2{,}0758 \cdot 10^{-9} v^2(k)|w(k)| & 1{,}827 \cdot 10^{-11} (v(k)w(k))^2 & 0 \\ 0 & 0 & 3{,}9139 \cdot 10^{-8} w(k)^2 \end{bmatrix}$$

En la figura 4 se muestran los distintos valores de la varianza de Δx_o respecto a v^2, representando la recta en rojo la expresión aproximada de la ecuación 5, comprobándose lo buena que es esa expresión. En las figuras 5, 6 y 7 se muestran también las comparaciones frente a los resultados reales de las ecuaciones 6, 7 y 8, respectivamente.

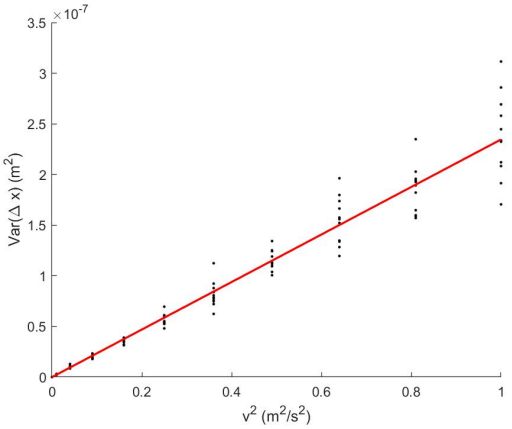

Figura 4: $Var(\Delta x_o)$ frente av^2

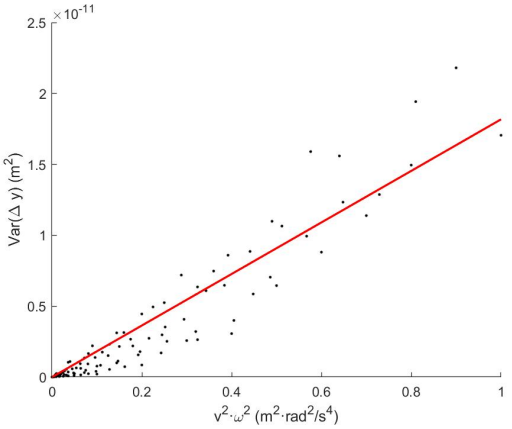

Figura 5: $Var(\Delta y_o)$ frente a $(vw)^2$

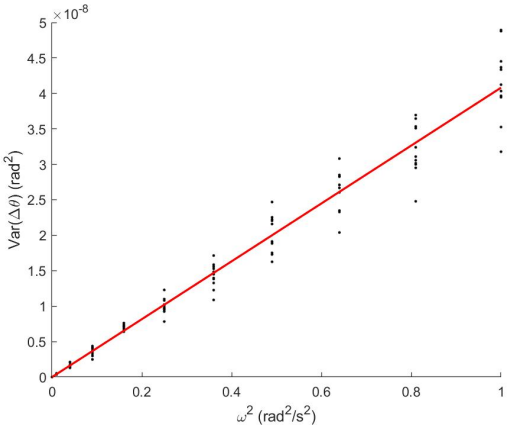

Figura 6: $Var(\Delta\theta_o)$ frente a w^2

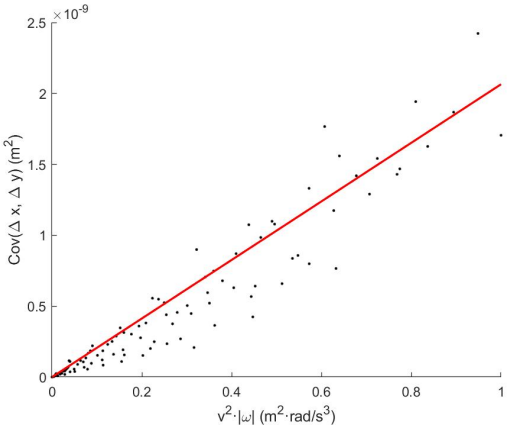

Figura 7: $Cov(\Delta x_o, \Delta y_o)$ frente a $v^2|w|$

3. Sensores utilizados y calibración

En este capítulo se describirán las clases creadas en Matlab que representan sensores de ultrasonidos y de telémetro láser, así como el número y la localización de los sensores utilizados. Por último, se explicará la construcción de la matriz R de covarianzas de la medida.

3.1. Programación de los sensores en Matlab

Los sensores se implementan en Matlab como una clase llamada "sensor", de la que derivan las clases "sensor_us" (sensor de ultrasonidos) y "sensor_ls" (telémetro láser). Estas clases están incluidas dentro de la clase "robot", que es básicamente un array de sensores unidos a un centro común. Este centro se toma como la posición del robot ($\bar{X} = [x, y, \theta]^t$), que es la posición que hay que estimar y respecto al cual se calcula todo (control, matrices jacobianas...).

El objeto del tipo "robot" tiene métodos para actualizar su posición y la de los sensores, estimar medidas (a través de las clases "sensor" que tiene asociadas), calcular el jacobiano y dibujarse en un plot de Matlab. Muchas de estas cosas las hace interaccionando con un objeto de la clase "entorno", que consiste en una colección de objetos "pared"(solo se consideran paredes verticales u horizontales) y "baliza".

Cada sensor viene determinado por 2 vectores de posición: la posición relativa del sensor respecto al centro del robot ($\bar{X}_{rel} = [x_{rel}, y_{rel}, \theta_{rel}]^t$), y la posición absoluta del sensor respecto al origen ($\bar{X}_{abs} = [x_{abs}, y_{abs}, \theta_{abs}]^t$). La posición relativa se mantiene siempre constante, siendo x_{rel} la proyección de la posición del sensor en la dirección delantera del robot; y_{rel}, la proyección en la dirección transversal; y θ_{abs} el ángulo (yaw) en el que apunta el sensor respecto a la guiñada (yaw) del robot.

En la figura 8 se muestran estas posiciones, donde el hexagrama negro es la posición central del robot y el asterisco tojo, la posición del sensor. El robot tiene una guiñada θ, mientras el sensor apunta a donde indica la línea punteada roja.

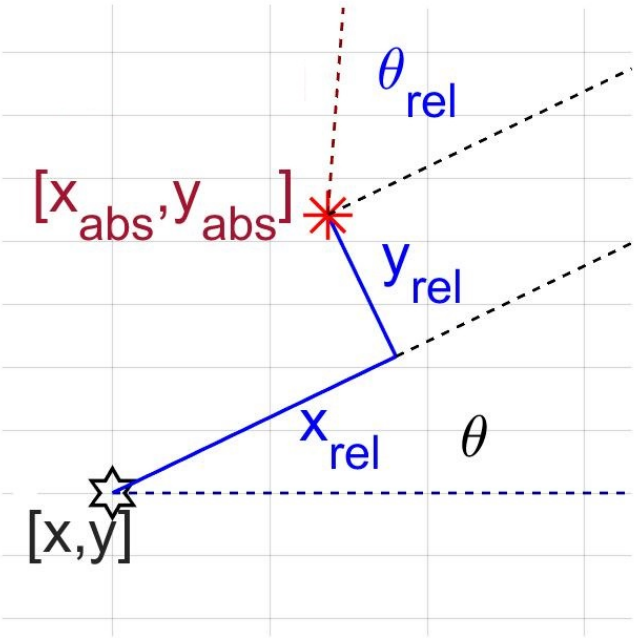

Figura 8: Posición absoluta y relativa del sensor respecto a la del robot

De esta figura, se pueden obtener las siguientes relaciones, que serán muy útiles en la estimación de las medidas y el cálculo del jacobiano en el filtro de Kalman:

$$x_{abs} = x + x_{rel}cos(\theta) - y_{rel}sin(\theta) \tag{9}$$
$$y_{abs} = y + x_{rel}sin(\theta) + y_{rel}cos(\theta) \tag{10}$$
$$\theta_{abs} = \theta + \theta_{rel} \tag{11}$$

Las cuales son muy parecidas a las ecuaciones 1, 2, 3 y, consecuentemente, 4, ya que se trata en el fondo de una matriz de rotación.

3.2. Sensores escogidos y utilización

Se utilizan cinco sensores de ultrasonidos y un telémetro láser en modo detección de balizas, contabilizando un total de 10 balizas. Debido a un mayor error en la medición de distancias, del telémetro láser solo se toman los datos de ángulo relativo frente a las balizas (este tratamiento de medidas se lleva a cabo en la función creada *GetLaserData*. La disposición de estos sensores utilizados se muestran en la figura 9.

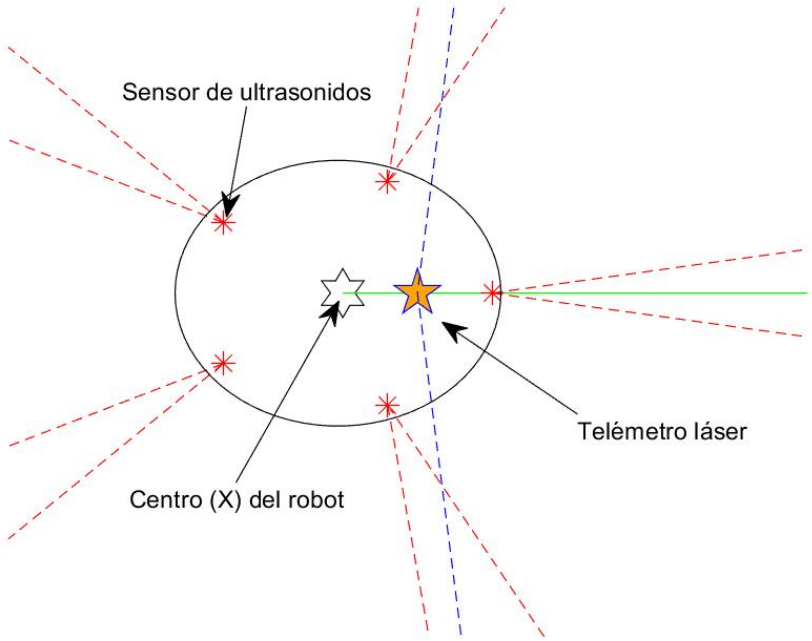

Figura 9: Distribución de los sensores utilizados respecto al centro del robot

Los cinco ultrasonidos son colocados en un círculo de radio 0.2 m con centro en la posición $\bar{X}$, centro del robot. El primer ultrasonido se posiciona apuntando en la misma dirección que el robot, en dicho círculo. Los siguientes se hallan en el círculo, equidistantes entre sí, con un ángulo respecto al centro del robot de $2\pi/5$, $4\pi/5$, $-4\pi/5$ y $-2\pi/5$ radianes, respectivamente. Es este mismo ángulo en el que apuntan los ultrasonidos.

Estos ultrasonidos tienen un ángulo del cono de 0.174533 rad (ángulo obtenido indagando en los ficheros del Apolo), y este cono se representa igualmente en la figura 9, en líneas discontinuas rojas. El ángulo de los ultrasonidos respecto al suelo es nulo, es decir, se colocan exactamente en horizontal. Los sensores tienen un rango máximo de tres metros

El telémetro láser ha sido elegido apuntando en la misma dirección que el robot, a una distancia de 0.1 m, tal y como se aprecia en la mencionada figura 9. El ángulo de visión del telémetro es de $\pm\pi/2$ rad, aunque en la imagen se muestra, en líneas azules discontinuas, ligeramente reducido para una mayor apreciación de la dirección de este sensor.

3.3. Calibración y obtención de la matriz R

Como la función *apoloGetAllultrasonicSensors* no genera ningún ruido en las medidas de los ultrasonidos, hemos considerado necesario añadirles ruido manualmente en pos de un buen funcionamiento del filtro de Kalman (ya que no tendría sentido aplicar el filtro de Kalman con medidas perfectas, además de que nula varianza implicaría que la matriz R fuera singular). Este ruido se modela con una distribución normal de media cero y desviación típica σ 0.03 (varianza $s^2 = 9 \cdot 10^{-4}$), que se suma a las medidas de los ultrasonidos en nuestra función *GetUltrasonicSensorsWithNoise*. La varianza de este ruido es del mismo orden de magnitud que la del sensor láser, y se ha elegido así para que no haya demasiada prevalencia de una medida sobre otra. Sin embargo, es varios órdenes de magnitud mayor que la del sistema de locomoción.

El telémetro láser, como ya se ha indicado, solo mide los ángulos respecto a las balizas. Este ya viene con un ruido aleatorio directamente de Apolo (*apoloGetLaserLandMarks*), por lo que no ha sido necesario alterar las mediciones lo más mínimo.

Para calibrar los sensores y obtener la matriz de varianzas y covarianzas de las medidas R, los ultrasonidos se disparan (sin moverse del sitio) los ultrasonidos 500 veces, y se obtiene la varianza de las medidas. Teóricamente debería ser la misma para todos los ultrasonidos (debido a la manera artificial de introducción ruido llevada a cabo), pero evidentemente hay pequeñas diferencias. El telémetro láser también se dispara 500 veces, observando siempre la misma baliza, y se calcula la varianza de las medidas. Esta varianza se considera que es la misma para cada baliza (lo cual es lógico), y se considera independiente de la distancia del sensor a la baliza.

La matriz R, tras la calibración es, teniendo en cuenta que hay 5 ultrasonidos (que se disparan todos a la vez) y 10 balizas:

$$R = \begin{bmatrix} \underbrace{\begin{matrix} 9 \cdot 10^{-4} & & 0 \\ & \ddots & \\ 0 & & 9 \cdot 10^{-4} \end{matrix}}_{\times 5} & & \\ & \underbrace{\begin{matrix} 3{,}2 \cdot 10^{-4} & & 0 \\ & \ddots & \\ 0 & & 3{,}2 \cdot 10^{-4} \end{matrix}}_{\times 10} & \end{bmatrix}$$

4. Algoritmo de localización

En este capítulo, se desarrollará el algoritmo de localización mediante el filtro de Kalman, incluyendo la obtención de la estimación de las medidas, las matrices jacobianas y gráficas comparativas.

4.1. Estimación de las medidas y de las matrices jacobianas

Una de las partes más importantes del EKF es la estimación de las medidas. Esta parte es necesaria para poder hacer la comparación y la corrección posteriores.

4.1.1. Telémetro láser

Para la utilización del sensor láser, se decidió únicamente utilizar los datos de ángulo entre el robot y las balizas, despreciando la medida de la distancia entre el robot y la baliza al no considerarse necesaria para la correcta localización del robot.

La obtención del ángulo entre el robot y cada una de las balizas (ϕ) se realiza a través de la siguiente función:

$$\phi = \arctan\left(\frac{P_y(i) - y_{abs}}{P_x(i) - x_{abs}}\right) - \theta_{abs} \tag{12}$$

Para la obtención del jacobiano de las medidas del sensor primero deben calcularse las derivadas parciales de la posición absoluta de los sensores en función de la orientación (θ) del robot:

$$\frac{\partial x_{abs}}{\partial \theta} = -x_{rel} \cdot \sin\theta - y_{rel} \cdot \cos\theta \tag{13}$$

$$\frac{\partial y_{abs}}{\partial \theta} = x_{rel} \cdot \cos\theta - y_{rel} \cdot \sin\theta \tag{14}$$

Finalmente, se calcula el jacobiano (matriz H) para cada baliza:

$$H = \begin{bmatrix} \frac{\partial \phi}{\partial x} & \frac{\partial \phi}{\partial y} & \frac{\partial \phi}{\partial \theta} \end{bmatrix} = \begin{bmatrix} \frac{P_y(i)-y_{abs}}{(P_x(i)-x_{abs})^2+(P_y(i)-y_{abs})^2} & \frac{-(P_x(i)-x_{abs})}{(P_x(i)-x_{abs})^2+(P_y(i)-y_{abs})^2} & \frac{\partial \phi}{\partial x}\frac{\partial x_{abs}}{\partial \theta} + \frac{\partial \phi}{\partial y}\frac{\partial y_{abs}}{\partial \theta} - 1 \end{bmatrix} \tag{15}$$

Siendo θ la orientación del robot, x_{rel}, y_{rel} las posiciones relativas del sensor respecto al centro del robot, x_{abs}, y_{abs} las posiciones absolutas del sensor en el plano, P_x, P_y, ϕ la posición absoluta y ángulo respecto al robot del sensor láser número (i). Este último elemento del jacobiano se pudo hacer así debido a que $\frac{\partial x}{\partial x_{abs}} = \frac{\partial y}{\partial y_{abs}} = 1$.

4.1.2. Sensor de ultrasonidos

Para estimar las medidas de los sensores de ultrasonidos se supone que el sensor siempre va a medir el punto más cercano dentro del cono de ultrasonidos. Para ello, hay que conocer cuál es dicho punto, que puede ser una de las siguientes opciones:

1. El punto formado por la intersección de la perpendicular que va desde el sensor hasta la pared con la propia pared, en caso de estar incluida esta línea perpendicular dentro del cono.

2. El punto intersección de la recta que sale del sensor con un ángulo de $\theta_{abs} \pm \delta$, siendo θ_{abs} el ángulo (yaw) en el que se orienta el sensor y δ, el ángulo del cono.

3. El punto extremo de una pared (una esquina), si esta se encuentra dentro del cono.

Una vez comprobados todos los casos, se escoge la medida que dé la distancia más corta, según la siguiente ecuación:

$$z = \sqrt{(x_{abs} - x_p)^2 + (y_{abs} - y_p)^2} \tag{16}$$

Donde x_{abs} e y_{abs} son la posición del sensor respecto al origen y x_p e y_p son la posición del punto de mínima distancia.

Todos los casos de distancia mínima enumerados se pueden observar en la figura 10, en la cual el hexagrama negro es el centro del robot, orientado en la dirección de la recta verde; los asteriscos en rojo son los sensores de ultrasonidos, con un cono delimitado por las rectar discontinuas rojas; los puntos morados, los puntos de mínima distancia dentro de cada cono; en azul se representan las paredes y, por último, los círculos rojos son las balizas (que no hay que tener en cuenta aquí.

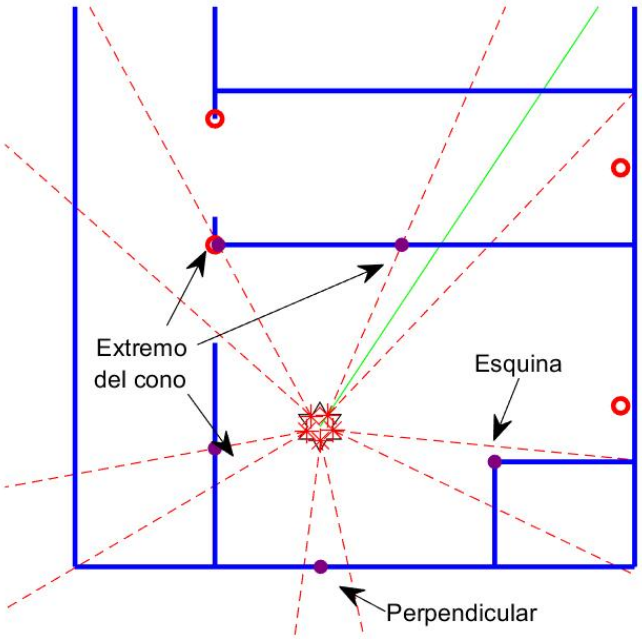

Figura 10: Puntos de las paredes más cercanos en el cono de los ultrasonidos

Todo esto se realiza en la función *estimar_medidas*, donde interaccionan los objetos "sensor" o "robot" con "entorno" (explicados en el capítulo 3.1), comprobando todos estos casos para cada ultrasonido y para cada pared. Esta función también calcula la matriz jacobiana, que para un solo sensor coincide con el gradiente (∇d). Debido a la gran variedad de casos que hay, en los cuales un movimiento en x, y o θ puede o no producir un cambio en la medida, el cálculo del jacobiano cambia mucho. Los casos posibles son los siguientes:

1. En una pared horizontal, un cambio en la dirección x no produce ningún cambio en la medida; para el resto de las variables, la derivada sí que toma otro valor:

$$H = \left[\begin{array}{ccc} 0 & -\frac{1}{sin(\alpha)} & \frac{1}{sin(\alpha)^2}(-x_{rel}sin(\alpha-\theta) + y_{rel}cos(\alpha-\theta) + (y_{abs}-y_p)cos(\alpha)) \end{array} \right] \quad (17)$$

   Donde α es el ángulo que forman el ultrasonido con el punto de mínima distancia ($theta_{abs} \pm \delta$, $\pi/2$ o $-\pi/2$) e y_p es la posición y de ese mismo punto. El resto de subíndices ya se han comentado en el apartado 3.1.

2. En una pared vertical, un movimiento paralelo al eje y no altera la distancia medida:

$$H = \left[\begin{array}{ccc} -\frac{1}{cos(\alpha)} & 0 & \frac{1}{cos(\alpha)^2}(-x_{rel}sin(\alpha-\theta) + y_{rel}cos(\alpha-\theta) - (x_{abs}-y_p)sin(\alpha)) \end{array} \right] \quad (18)$$

   Donde α es el ángulo que forman el ultrasonido con el punto de mínima distancia ($theta_{abs} \pm \delta$, 0 o pi) y x_p es la posición x de ese mismo punto.

3. En una esquina, la jacobiana es:

$$H = \left[\begin{array}{ccc} \frac{x_{abs}-x_p}{z} & \frac{y_{abs}-y_p}{z} & \frac{\partial\phi}{\partial x}\frac{\partial x_{abs}}{\partial\theta} + \frac{\partial\phi}{\partial y}\frac{\partial y_{abs}}{\partial\theta} \end{array} \right] \quad (19)$$

   Donde z es la medida tomada, y $\frac{\partial x_{abs}}{\partial\theta}$ y $\frac{\partial y_{abs}}{\partial\theta}$ se han calculado en las ecuaciones 13 y 14

4.1.3. Sistemas de locomoción

Las fórmulas de los sistemas de locomoción se vieron en el capítulo 2.1, concretamente en la fórmula 4. A partir de esta, es posible obtener las matrices jacobianas necesarias para obtener la matriz $P(k+1|k)$. Todo esto se realiza en la propia función *GetPositionFromOdometry*.

La matriz jacobiana respecto a la posición estimada es:

$$F_{\hat{X}} = \begin{bmatrix} 1 & 0 & -\Delta x_o sin(\hat{\theta}) - \Delta y_o cos(\hat{\theta}) \\ 0 & 1 & \Delta x_o cos(\hat{\theta}) - \Delta y_o sin(\hat{\theta}) \\ 0 & 0 & 1 \end{bmatrix} \quad (20)$$

La matriz jacobiana respecto a la odometría es la propia matriz de ecuación 4:

$$F_{\Delta\bar{X}} = \begin{bmatrix} cos(\hat{\theta}) & -sin(\hat{\theta}) & 0 \\ sin(\hat{\theta}) & cos(\hat{\theta}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (21)$$

4.2. Filtro de Kalman Extendido

Para la realización del filtro de Kalman extendido se realizó siguiendo el procedimiento descrito por Schutter et al. [1]. Dicho procedimiento puede verse reflejado en la figura 11.

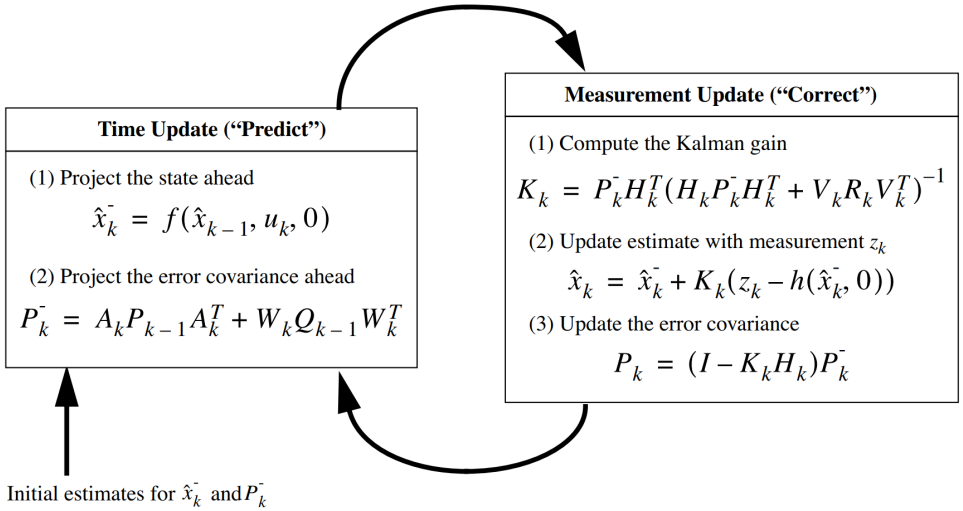

Figura 11: Esquema de aplicación del filtro de Kalman extendido. Fuente: Welch and Bishop [2]

La aplicación del filtro de Kalman extendido se realizó a través de los siguientes pasos realizados de manera iterativa:

1. **Obtención de la posición del robot a través de la odometría**: El primer paso a realizar consiste en la obtención de la posición del robot y su orientación (x, y y θ), junto con la matriz de varianzas P asociada. Para ello se llama a la función de Matlab *GetPositionFromOdometry*, que devuelve las medidas basándose en los datos obtenidos a través de la odometría del robot (mediante la función de Apolo *apoloGetOdometry*) y la posición previa del mismo. Este procedimiento ha sido descrito en el apartado 4.1.3 y viene definido por las siguientes ecuaciones:

$$\hat{x}_{k|k-1} = f(x_{k-1}, u_k) \tag{22}$$

$$P_{k|k-1} = F_{\hat{X}}^T \cdot P_{k-1} \cdot F_{\hat{X}} + F_{\Delta\bar{X}}^T \cdot Q_k \cdot F_{\Delta\bar{X}} \tag{23}$$

   Siendo $\hat{x}_{k|k-1}$ la posición y orientación del robot estimada mediante la odometría, $F_{\hat{\bar{X}}}$ el jacobiano respecto a $\hat{x}_{k|k-1}$, $F_{\Delta\bar{X}}$ el jacobiano respecto a la odometría del robot y Q_k la matriz de la varianza del ruido del proceso.

2. **Predicción de las nuevas medidas de los sensores**: El segundo paso a realizar llama a la función de Matlab *estimar_medidas*, que es un método de la clase robot. Esta función devuelve una estimación de las medidas de todos los sensores del robot ($\hat{Z}_k$) en función de la nueva posición del mismo obtenida en el paso anterior. También se obtiene la matriz jacobiana de todos los sensores (H_k). Este procedimiento ha sido descrito en los apartados 4.1.2 y 4.1.1.

3. **Obtención de las nuevas medidas de los sensores**: El tercer paso a realizar llama a las funciones de Matlab *GetUltrasonicSensorsWithNoise* y *getLaserData*, que devuelve una estructura con las medidas de los sensores de ultrasonidos con ruido gaussiano para simular sus errores en la medida y de los sensores láser respectivamente.

4. **Comparación entre predicción y estimación**: El cuarto paso a realizar compara las medidas de los sensores del robot obtenidas mediante Apolo y las estimaciones de las mismas realizadas a través del modelo realizado en Matlab.

$$\nu = Z_k - \hat{Z}_k \tag{24}$$

Siendo Z_k las medidas de los sensores del robot obtenidas mediante Apolo y $\hat{Z}_k$ la estimación de las medidas de los sensores modelada en Matlab.

Esta comparación devuelve el vector de las diferencias entre medidas (ν), eliminando las medidas que no cumplan ciertas características.

5. **Obtención de las matriz de la ganancia de Kalman (W_k)**: El quinto paso a realizar es obtener la matriz de la ganancia de Kalman (W_k) a través de los datos obtenidos previamente. Esta matriz de ganancias permitirá el ajuste del estado del robot y de la matriz de varianzas asociadas P al realizar la corrección del estado. La ecuación utilizada para obtener la ganancia de Kalman es la siguiente:

$$W_k = P_{k|k-1} \cdot H_k^T \cdot (H_k \cdot P_{k|k-1} \cdot H_k^T + R)^{-1} \tag{25}$$

6. **Corrección del estado ($X_{k|k}$) y de la matriz de varianzas ($P_{k|k}$)**: El paso final a realizar obtiene el vector del estado del sistema corregido ($X_{k|k}$) y la matriz de varianzas corregida ($P_{k|k}$) mediante la aplicación de la ganancia de Kalman a la diferencia entre las medidas de los sensores y las medidas teóricas modeladas (vector ν). Para realizar dicha corrección se aplican las siguientes fórmulas:

$$X_{k|k} = X_{k|k-1} + W_k \cdot \nu \tag{26}$$

$$P_{k|k} = (I - W_k \cdot H_k) \cdot P_{k|k-1} \tag{27}$$

Siendo I la matriz identidad, ν el vector de actualizaciones en la medida, W la matriz que representa la ganancia de Kalman.

4.3. Comprobación del efecto de cada parámetro

Se realiza una evaluación del efecto de modificar algún parámetro con relación al movimiento del robot. Para ello se realiza una simulación de una trayectoria predefinida del robot, sin hacer uso del planificador para poder realizar un gran número de simulaciones alterando el menor número de parámetros posible (a excepción del parámetro a estudiar).

4.3.1. Modificación de la matriz Q

En primer lugar se evalúa la influencia de la matriz Q en la trayectoria del robot. Se realizan un total de 6 simulaciones, multiplicando la matriz Q obtenida en la calibración del robot por 0.1, 10, 100, 1000, 10000, obtenieniendo los resultados que pueden observarse en las figuras 12, 13 y 14 representando la trayectoria estimada del robot, su trayectoria real y la distancia entre ambas respectivamente:

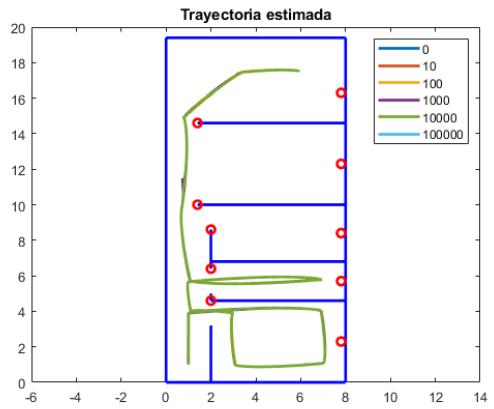

Figura 12: Trayectoria estimada del robot con diferentes valores.

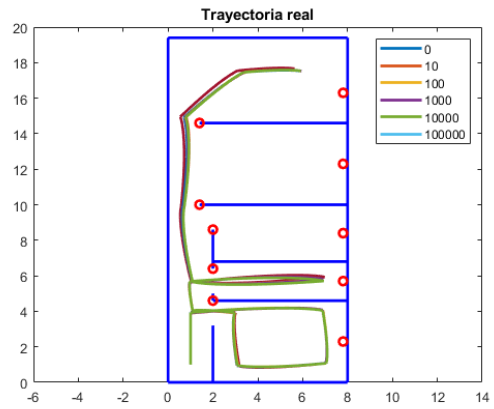

Figura 13: Trayectoria estimada del robot con diferentes valores.

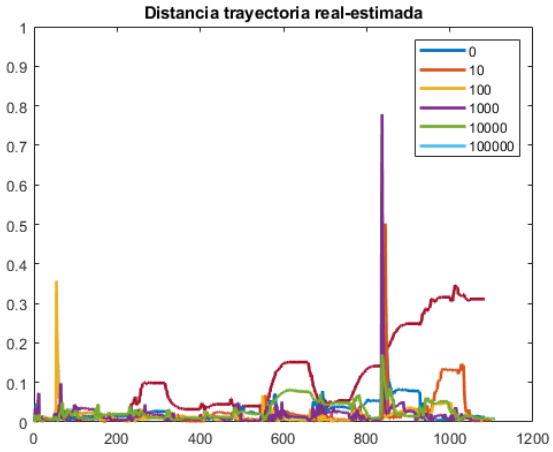

Figura 14: Distancia entre la trayectoria estimada y la trayectoria real a lo largo del tiempo.

Puede observarse como en los casos $Q = 1000 \cdot Q$ y $Q = 10 \cdot Q$ hay un aumento considerable en la distancia entre trayectoria estimada y la real en los momentos donde únicamente se detecta una o ninguna baliza, aumentando la incertidumbre del sistema y produciendo así un mayor error.

4.3.2. Modificación de la matriz R

En segundo lugar se evalúa la influencia de la matriz R en la trayectoria del robot. Se realizan un total de 7 simulaciones, multiplicando la matriz R (varianza en la medida de los sensores) obtenida en la calibración del robot por 0.1, 10, 100, 1000, 10000 y 100000. Se realiza una simulación más ya que la matriz R es de varianzas en vez de desviaciones típicas, teniendo menos efecto multiplicar por valores bajos. Los resultados que pueden observarse en las figuras 15, 16 y 17 representando la trayectoria estimada del robot, su trayectoria real y la distancia entre ambas respectivamente:

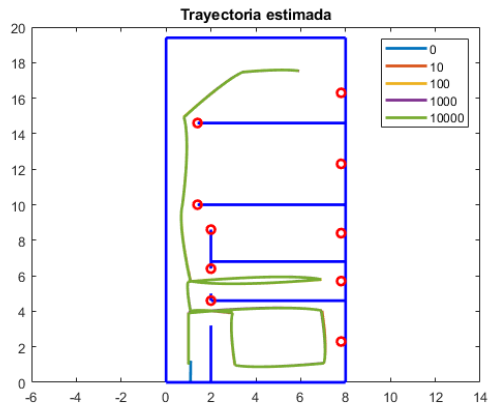

Figura 15: Trayectoria estimada del robot con diferentes valores.

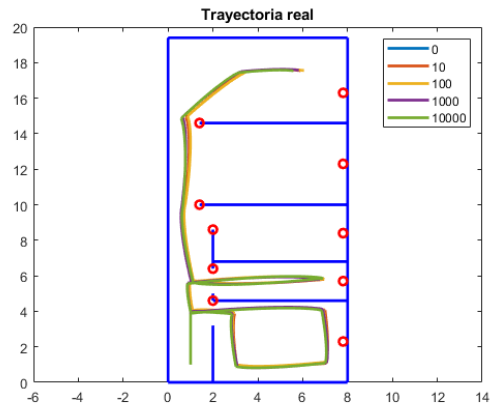

Figura 16: Trayectoria estimada del robot con diferentes valores.

Es destacable mencionar como en los casos ($Q = 10 \cdot Q$ y $Q = 1000 \cdot Q$), se produce un pico en la distancia entre trayectoria real y la estimada, .

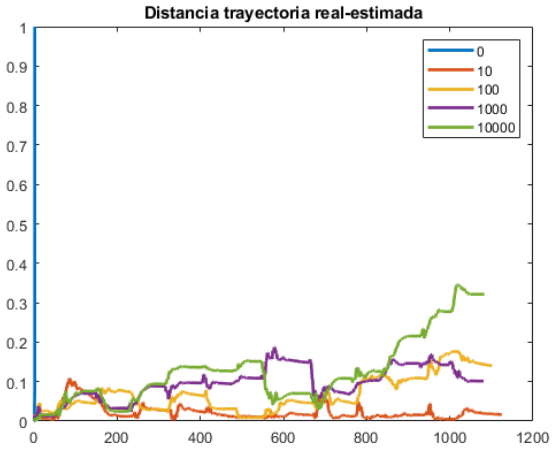

Figura 17: Distancia entre la trayectoria estimada y la trayectoria real a lo largo del tiempo.

Es destacable mencionar como en el primer caso ($R = 0$), la simulación no avanza ya que la diferencia entre la trayectoria estimada y la real tiende a infinito, evitando la convergencia del algoritmo. También es reseñable como la simulación con el mayor valor de R ($R \cdot 100000$), la distancia entre trayectoria estimada y real aumenta hasta 0.4 metros, siendo esta diferencia inadmisible en una aplicación real.

4.3.3. Evolución de la matriz P

Finalmente, se realiza un análisis de la evolución de la matriz P con el paso de las iteraciones. Para analizar su valor en la figura 18 puede observarse la evolución de la traza de la matriz a lo largo del tiempo.

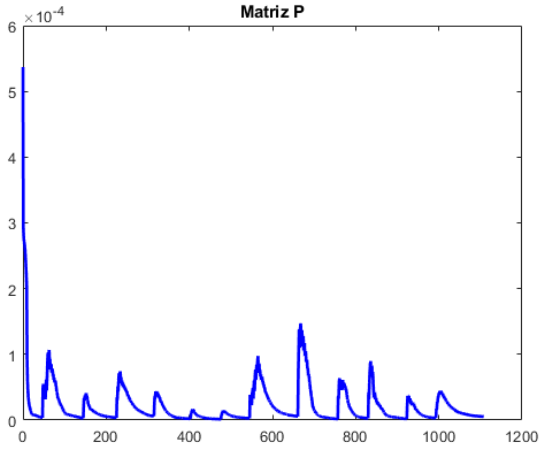

Figura 18: Evolución de la traza (norma de Frobenius) de la matriz P en la simulación realizada.

Puede observarse como empieza con un valor muy elevado que disminuye rápidamente. También pueden observarse varios picos de incertidumbre producidos por los momentos en los que el robot dejaba de ver alguna baliza, aumentando en gran medida la incertidumbre de su localización exacta. Dicha incertidumbre disminuía al detectar una nueva baliza y por la convergencia del algoritmo.

5. Algoritmo de control

La estrategia de control que se ha decidido seguir es la de avanzar en dirección al siguiente punto de destino. Los pasos a seguir son:

1. Girar hacia el siguiente punto.
2. Avanzar hacia el siguiente punto.
3. Si se aporta orientación del destino, orientarse.

Este método si bien es más lento tiene la ventaja de ser sencillo de programar y de mantener la cabeza del robot mirando al frente y en consecuencia los sensores.

5.1. Cálculos

Los cálculos realizados en este apartado son realmente sencillos. Únicamente se calcula el ángulo del siguiente objetivo como:

$$\theta_{objetivo} = \arctan\left(\frac{Y_{objetivo} - Y_{robot}}{X_{objetivo} - X_{robot}}\right) \tag{28}$$

Y la distancia como:

$$d = \sqrt{(X_{objetivo} - X_{robot})^2 + (Y_{objetivo} - Y_{robot})^2} \tag{29}$$

Siendo la posición del robot proporcionadas por el filtro extendido de Kalman.

5.2. Control P de las velocidades

A la hora de elegir las velocidades v y w con las que se mueve el robot se han obtenido buenos resultados con un simple control proporcional. De este modo las velocidades en las diferentes etapas del controlador se establecen de la siguiente manera:

1. Control ángulo:

$$v = 0 \tag{30}$$
$$w = Kp * (\theta_{objetivo} - \theta_{robot}) \tag{31}$$

2. Control de distancia:

$$v = Kp * distancia \tag{32}$$
$$w = 0 \tag{33}$$

3. Choque: velocidades aleatorias

$$v = -0{,}1 + (0{,}1 + 0{,}1). * rand(1, 1) \tag{34}$$
$$w = -0{,}1 + (0{,}1 + 0{,}1). * rand(1, 1) \tag{35}$$

5.3. Fases de la programación del controlador

En un primer intento se trató de hacer una programación del controlador de manera secuencial, de este modo, primero se giraba y una vez girado simplemente se avanzaba esperando llegar al objetivo. Lo cual en ocasiones no sucedía debido a los errores al controlar el robot. El primer esquema del código del controlador era el siguiente:

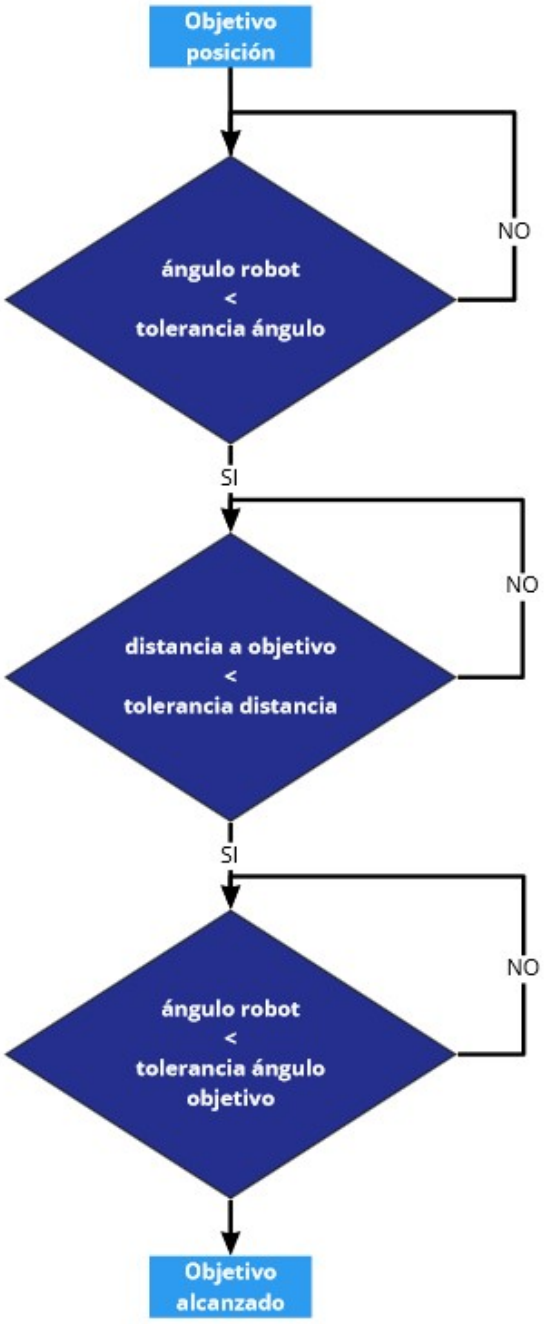

Figura 19: Primera versión del controlador.

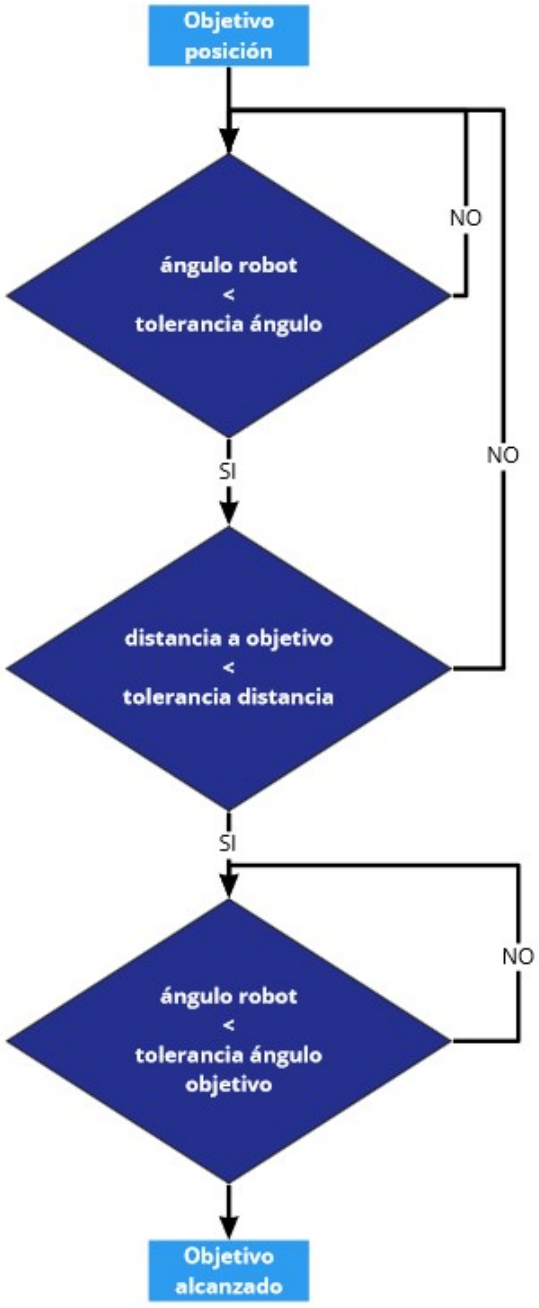

Figura 20: Segunda versión del controlador.

Como se ve en la figura 19 dado que las etapas eran secuenciales, una vez alcanzado el ángulo, al avanzar si no se recalculaba el ángulo y te desviabas, la distancia al objetivo nunca llegaba a ser 0.

Para arreglar este problema se cambió ligeramente la arquitectura comprobando así en todo

momento ángulo y distancia lo cual funcionó considerablemente mejor. El esquema de esta nueva versión se encuentra en la figura 20.

5.4. Control Reactivo

Dado que el antiguo controlador no era capaz de superar obstáculos, se ha implementado un controlador reactivo simple. Para decidir cual de los dos controles ha de ejecutarse, se ha añadido la figura del piloto.Este puede decidir cambiar al modo reactivo si sucede cualquiera de los dos supuestos siguientes:

- La distancia al sensor frontal es menor a un valor determinado.
- El robot se ha chocado.

Se han intentado utilizar diferentes estrategias de control reactivo, entre ellas, tratar de seguir el obstáculo usando los sensores de ultrasonidos, mover el robot de manera aleatoria tratando de que esquive el obstáculo por azar, tratar de modificar los puntos de la trayectoria ya calculada. Todos ellos sin mucho éxito.

Finalmente el control reactivo que mejor resultados ha proporcionado ha sido una secuencia de movimientos predefinida:

1. Marcha atrás.
2. Giro en una dirección (atendiendo a los sensores laterales de ultrasonidos).
3. Avance.

Uno de los principales problemas que se encontraron a la hora de implementar este control era que, dado que algunos puntos de la trayectoria calculada podían encontrarse bajo un obstáculo, a pesar de que el robot evitara el obstáculo, seguía intentando avanzar hacia él. Para evitar eso, si se dispara el control reactivo, se avanza un punto en la secuencia de puntos planificada en la trayectoria.

5.5. Pruebas de los parámetros de la tolerancia

Uno de los parámetros que más afecta al controlador es la tolerancia. Para aislar este problema del resto de los elementos, usamos en lugar de los valores de posición del filtro de Kalman los valores de posición exactos proporcionados por el simulador Apolo. Intuitivamente se llega al siguiente razonamiento:

- Valores laxos de tolerancia: menor precisión pero mayor rapidez. Valores extremadamente laxos hacen que no siga bien la trayectoria chocando con paredes.
- Valores estrictos de tolerancia: mayor precisión pero más lento. Valores demasiado estrictos hacen que no se puedan satisfacer las restricciones y nunca se llegue al objetivo.

A continuación se muestran algunas de las trayectorias seguidas por el robot al darle como consigna 4 puntos a los que ir:

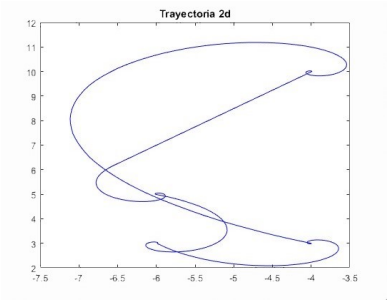

Figura 21: Tolerancia laxa.

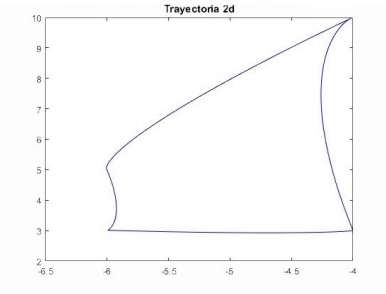

Figura 22: Tolerancia media.

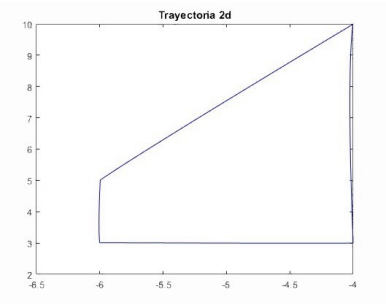

Figura 23: Tolerancia alta.

Como se puede ver en las figuras 21, 22 y 23 a medida que hacemos más restrictiva la tolerancia la trayectoria se asemeja más a una linea recta que es lo que nos interesa. Para valores más restrictivos el robot no consigue avanzar de la fase de orientación.

6. Algoritmo de planificación

Tras hacer una búsqueda de los distintos algoritmos de planificación disponibles en Matlab se decidió optar por el *plannerRRTstar()* (RRT*).

Este algoritmo se base en el RRT (Rapidly-exploring Random Tree) con la particularidad que una vez que encuentra la solución puede continuar optimizando. Esto hace que la solución tienda a ser óptima.

6.1. Creación del mapa planificador

Para poder funcionar el planificador necesita tener un mapa del entorno. Como ya se había creado un mapa del entorno para poder predecir las medidas de los sensores en el filtro de Kalman, se ha reutilizado con algunas particularidades:

- Dado que el mapa ha sido creado con unidades enteras, si se pasase el mapa del entorno, dado que las puertas son muy pequeñas, al crear el validador no hay hueco para que pase el robot.

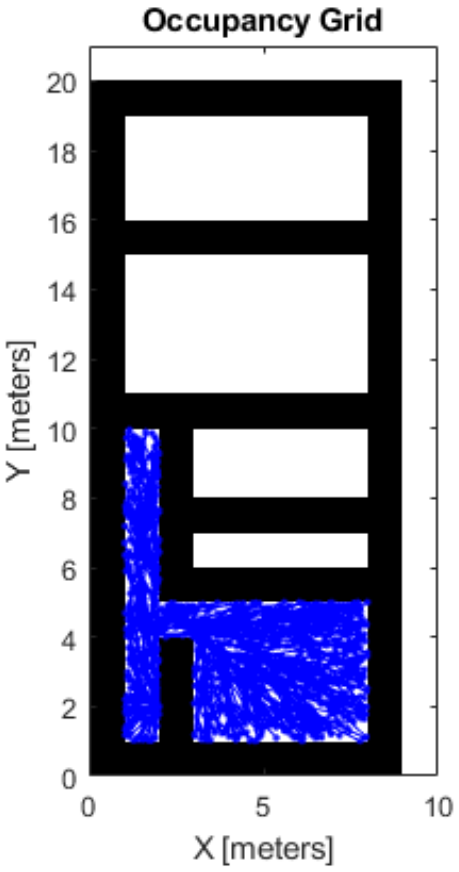

Figura 24: Primera versión del planificador.

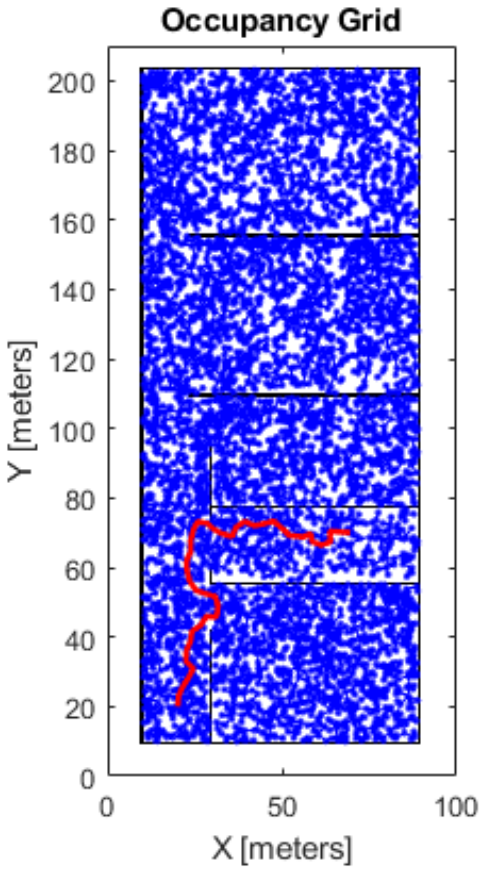

Figura 25: Segunda versión del planificador.

Es por ello que se ha incluido un parámetro de "resolución" el cual se encarga de aumentar los puntos que contiene el mapa para que, de este modo, el robot pueda encontrar el camino.

De este modo, el planeador ya es capaz de encontrar una solución. Sin embargo, dado que el robot no es un objeto puntual, para evitar que colisiones con las paredes se ha añadido un parámetro de "inflación".

Este parámetro lo que hace es añadir grosor extra a las paredes, de este modo estamos evitando que el planeador planifique sobre configuraciones en las cual haya colisión. El mapa resultante al aplicar estos dos parámetros es el siguiente:

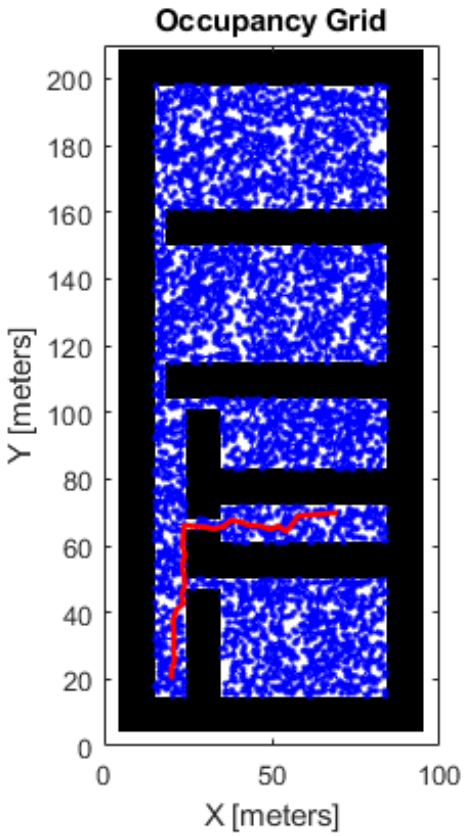

Figura 26: Planificador con resolución 0.1 e inflación 0.5.

De este modo el planeador logra encontrar una trayectoria que se encuentre lo suficientemente lejos de las paredes y así evitar colisiones.

En el planificador *plannerRRTStar()* para que siga optimizando la solución una vez encontrado el objetivo es necesario activar el parámetro:

$$planner.ContinueAfterGoalReached = true;$$

6.2. Planificación de trayectorias

Hasta este punto el planificador ha sido usado para navegar del punto A al punto B. Si queremos hacer una ruta pasando por los puntos A, B, C y D deberemos ejecutar la planificación para los diferentes puntos por pares. Es decir, primero calcularemos la trayectoria para los puntos A y B, luego partiendo de B, para B y C y por último para C y D.

De este modo iremos acumulando los puntos por los que debe ir pasando el robot en un vector que luego iremos dando al controlador.

7. Demostrador final

En este último apartado, se va a describir el uso del demostrador final.

7.1. Apolo

El primer paso y más obvio es abrir Apolo. En este trabajo se han utilizado dos posibles mapas: *TrabajoGuiado.xml*, que es el mapa principal, conformado por una casa con varias habitaciones y con balizas; y *TrabajoGuiado_con_Obstaculos.xml*, que es esa misma casa pero con un par de obstáculos.

Para observar la planificación y la localización mejor, se recomienda utilizas el primer mapa. El mapa con obstáculos es mejor para probar el control reactivo.

Si se quiere usar un mapa y luego otro, es preferible cerrar el mapa y luego cargar el siguiente, ya que con varios mapas abiertos a la vez es posible que no se ejecute correctamente.

7.2. Matlab

En Matlab existen cuatro scripts diferenciados, que se pueden ejecutar de manera independiente, aunque el archivo **MegaMain** ejecuta todos de manera secuencial:

1. **construccion_ent_bot**: Construye el objeto de tipo "entorno" en Matlab, que guarda información sobre las paredes (objeto "pared") y las balizas. También construye el objeto de clase "robot" que, como se ha indicado en la memoria, consiste en una posición a la que se le tienen acoplados un array de objetos "sensor" (incluyendo ultrasonidos o "sensor_us" y telémetro láser o "sensor_ls").

2. **Calibracion_odometria**: Calibra la odometría, generando la matriz *Q_pu*, que guarda para poder usarse más adelante sin necesidad de calibrar otra vez.

3. **Calibracion_sensores**: Calibra los sensores, mostrando gráficas y creando la matriz de covarianzas R, que guarda en memoria.

4. **Main_GyNR**: Es la parte del código importante. Carga los valores, previamente calculados, del robot, el entorno y la calibración. En este script, se deciden los puntos por los que queremos que vaya el robot, incluyendo el punto inicial (*trayectoria*), y se planifican las trayectorias a partir de estos (función *Planner*). También se incluye el bucle principal del programa, donde se deciden los movimientos del robot.

   Este bucle llama a otras funciones, como son el piloto (*Piloto*), el controlador normal o el reactivo en función de lo que diga el piloto (*Controller* y *ControllerReactive*, el movimiento del robot en Apolo (*apoloMoveMRobot*) y la estimación de la posición en el filtro de Kalman (*KalmanFilter*). Dentro de este filtro de Kalman, además, se llaman a las funciones que calculan la matriz Q en función de la velocidad (*MatrizQ*), que estiman la posición (*GetPositionFromOdometry*), que estiman la medidas y calculan la jacobiana (*robot.estimar_medidas*) y que obtienen las medidas (*GetUltrasonicSensorsWithNoise* y *GetLaserData*).

7.3. Ejecución del código

Si se ejecuta el script **MegaMain**, se realizará toda la calibración de la odometría y de los sensores, por lo que se verá al robot en APolo en una misma sala moviéndose por ella y haciendo muchas medidas. A continuación, aparecerán gráficas que muestran la relación lineal de las varianzas y covarianzas con distintas velocidades (vistas en el apartado 2.2), así como las mediciones de los sensores durante su calibración.

Durante la ejecución del propio **Main_GyNR**, se realizará la planificación entera mediante RRT* de la trayectoria, y se mostrarán los mapas planificadores y la trayectoria generada (vistos en el apartado 6).

A continuación, el robot se empezará a mover siguiendo las indicaciones dadas. Si el mapa tiene obstáculos, es posible que se quede atascado o tarde más en realizar el recorrido. El movimiento del robot se puede seguir en Apolo.

Una vez finalizado el recorrido (o transcurrido un tiempo lo suficientemente grande), aparecerán diversas gráficas mostrando el desempeño del robot. Estas gráficas serán similares a la de la figura 27, mostrando la velocidad lineal y angular de referencia, además de ciertas variables del controlador. Además, se mostrará una comparación de la trayectoria real y la estimada mediante el EFK, como son las figuras 28 (en un mapa sin obstáculos) y 29 (en un mapa con obstáculos).

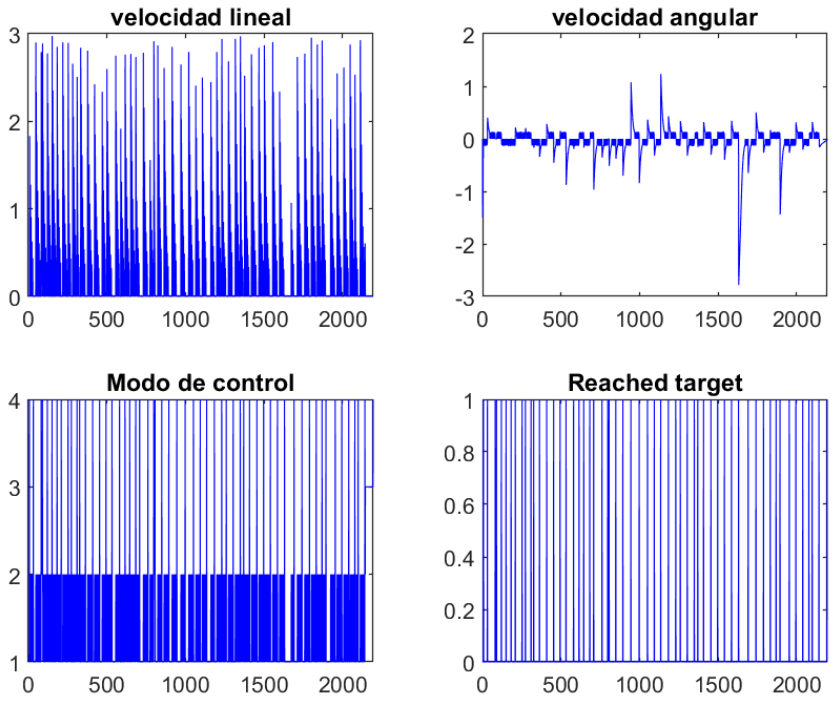

Figura 27: Gráficas de velocidades y otras variables de control

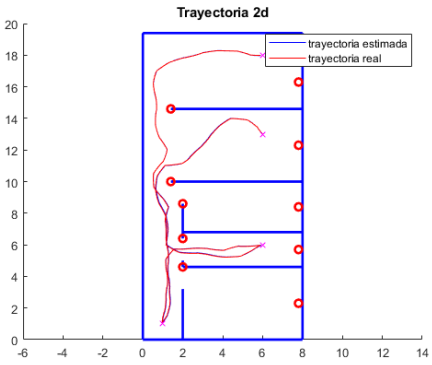

Figura 28: Trayectoria seguida por el robot en el mapa sin obstáculos

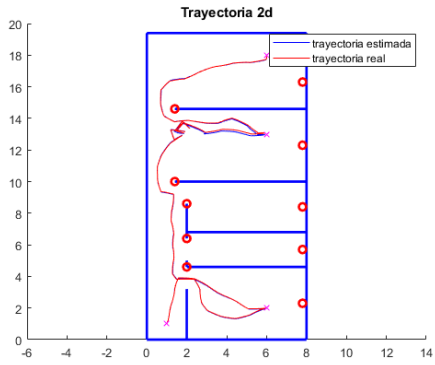

Figura 29: Trayectoria seguida por el robot en el mapa con obstáculos

8. Reparto de roles

El grupo está compuesto por tres integrantes, y aunque todos los integrantes han participado de manera activa en todos los apartados del trabajo, el reparto del trabajo se ha realizado de la siguiente manera:

1. **Guillermo Illana Gisbert**: Programación y diseño del filtro de Kalman extendido y del tratamiento y estimación de las medidas de telémetro láser. Edición de vídeo. Redacción de la memoria. Aporte de ideas.

2. **Alberto López Rodríguez**: Programación y diseño del algoritmo de control, del control reactivo y de la planificación. Redacción de la memoria. Aporte de ideas.

3. **Pablo García Peris**: Calibración de la odometría y los sensores. Programación y diseño de la estimación de las medidas de los ultrasonidos. Redacción de la memoria. Aporte de ideas.

9. Anexo: enlaces

El vídeo demostrativo que demuestra la integración del trabajo se puede ver en el siguiente enlace de Youtube:

https://www.youtube.com/watch?v=qidVsR9CFnE

El código se puede encontrar en su integridad, con todos los cambios hechos y versiones previas, en el siguiente repositorio de GitHub:

https://github.com/albertolr98/TrabajoGuiado

Referencias

[1] J. D. Schutter, J. D. Geeter, T. Lefebvre, and H. Bruyninckx. Kalman filters: A tutorial. 1999.

[2] G. Welch and G. Bishop. Welch & bishop , an introduction to the kalman filter 2 1 the discrete kalman filter in 1960. 1994.