IPD482 Guia 3

Patricio Carrasco O'Ryan
Ingeniería Civil Electrónica
Universidad Técnica Federico Santa María
Valparaíso, Chile
patricio.carrascoo@sansano.usm.cl

Sebastian Espinoza Toro
Ingeniería Civil Electrónica
Universidad Técnica Federico Santa María
Valparaíso, Chile
sebastian.espinozat@sansano.usm.cl

Tomás Bernal
Ingeniería Civil Electrónica
Universidad Técnica Federico Santa María
Valparaíso, Chile
tomas.bernal@sansano.usm.cl

***Abstract*—Este informe se centra en la simulación de un robot en entornos tanto conocidos como desconocidos, donde se ubican postes. Estos postes son posicionados manualmente por el usuario y el robot simulado es capaz de detectarlo mediante un LiDAR, por medio de una trayectoria que también se encuentra definida por el usuario.**

La situación se aborda desde distintos escenarios. En primer lugar se considera el robot ideal, teniendo en cuenta su pose, lo que detecta el LiDAR y sin conocimiento del entorno, de esta manera el objetivo es determinar la ubicación representativa de los postes.

En segundo se también se debe determinar la ubicación representativa de los postes, teniendo en cuenta su pose, pero el robot no es ideal, por lo que se sabe la posición real de cada poste para poder realizar una medición del error.

En tercer lugar se debe implementar un sistema de localización, teniendo como datos solamente los datos que arroja el LiDAR, la pose inicial del robot y el conocimiento previo de las ubicaciones de los postes.

En cuarto lugar, se solicita un sistema de localización solamente utilizando los lados del LiDAR, la pose inicial del robot, pero no hay conocimiento de los postes en el mapa.

El código desarrollado se encuentra disponible en https://github.com/P-coryan/Guia3_IPD482.

***Index Terms*—Robots móviles, LiDAR, , SLAM , Filtro de Kalman Extendido (EKF).**

En el archivo `simulador.m` se incluye una función llamada `DeteccionPostes`, la cual viene en blanco y es utilizada en todos los problemas de la guía. Por esto, crea el código a utilizar.

En cada iteración de la simulación se llama la función *DeteccionPostes* que detecta los postes visibles en cada instante. El rango máximo del LiDAR es de $10m$, por lo que se considera todos aquellos puntos que tengan un rango menor a $9.5m$, para hacer la distinción entre lo que esta 'viendo' el LiDAR y lo que no.

De cada punto observado se resta $\frac{\pi}{2}rad$ para que queden alineados con el eje local del robot. Los puntos se transforman a coordenadas cartesianas y se les realiza una traslación y rotación al eje de coordenadas globales utilizando los valores entregados por la pose del robot. Finalmente se tiene una matriz con todos los puntos globales observados en la iteración k, y se agrega a una matriz que almacena todos los puntos globales observados durante todas las iteraciones k del trayecto.

I. Pregunta 1

Como el objetivo es poder obtener una matriz M con la ubicación de todas las características puntuales, es decir los postes, y una matriz de covarianza para cada característica, es posible realizar la distinción de cada poste posterior a realizar el recorrido.

El robot utilizado en este caso es ideal, y no posee error de odometría, por lo que en la función `DeteccionPostes` se entrega como uno de los argumentos el robot *robot*.

Se crea la función `ClusteringNube(nubePtos, M, distCluster, cantCluster)`, que debe ser ejecutada luego de la simulación. `nubePtos` corresponde a la nube de puntos recolectada por el LiDAR durante la simulación, `M` es la ubicación de los postes designados al comienzo y `distCluster` y `cantCluster` son parámetros utilizados por la función `dbscan`. Las salidas de la función son `PMi, covNP, EPC, posteYgrupo, idx`.

Se utiliza el algoritmo `dbscan`, que clasifica en grupos la nube de puntos entregada. Cada grupo obtenido es la representación de un poste.

Por cada grupo, se toman sus puntos y mediante la función `convhull` (convex hull) se obtienen los vértices de la figura formada por los puntos. Se crea una matriz de $N \times N$ con las distancia entre todos los puntos, para obtener a los puntos que tengan la mayor distancia entre si. Se estima la posición del poste como el punto medio entre los dos vértices con la mayor distancia entre si.

El poste estimado se asigna al poste de la matriz M con el que tenga el menor error (es decir, el poste real más cercano al estimado). Se crea también una variable que identifica cada grupo a su poste correspondiente.

Finalmente se entrega la matriz `PMi` con las estimaciones de los postes y una matriz de covarianzas en coordenadas globales por cada poste, dada por

$$\begin{bmatrix} Vx & 0 \\ 0 & Vy \end{bmatrix} \tag{1}$$

donde Vx es la varianza de los puntos observados que componen al poste en el eje x y Vy es la varianza de los puntos observados que componen al poste en el eje y.

Ahora se procede a ejecutar la simulación. Se utiliza el entorno de la Figura 1. El robot utilizado es el ideal, y además el LiDAR se encuentra ubicado también en el robot ideal.

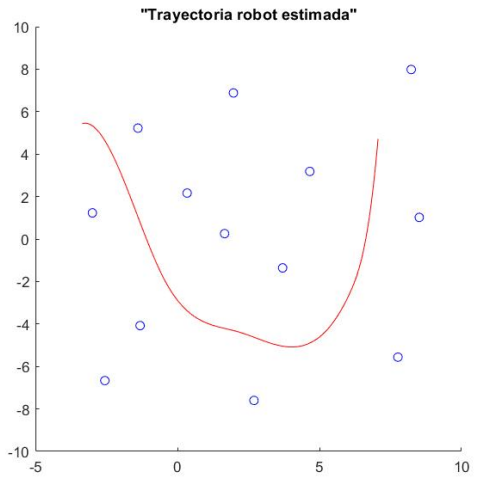


Fig. 1: Recorrido a utilizar

En la Figura 2 se observa la clasificación realizada utilizando el algoritmo de clustering de la nube de puntos captada por el robot.

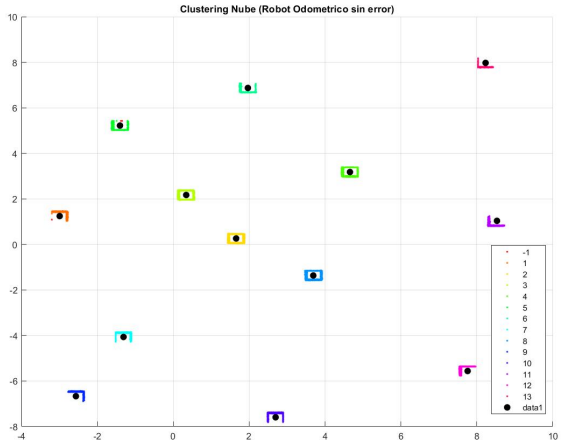


Fig. 2: Nube de puntos, clustering y estimaciones robot sin errores

La matriz de características se puede apreciar en la ecuación 2:

$$PM_i = \begin{bmatrix} -2.9896 & 1.2429 \\ 1.6576 & 0.2605 \\ 0.3428 & 2.1697 \\ 4.6552 & 3.1777 \\ -1.4013 & 5.2184 \\ 1.9675 & 6.8701 \\ -1.3085 & -4.0669 \\ 3.6893 & -1.3665 \\ -2.5629 & -6.6673 \\ 2.6970 & -7.5943 \\ 8.5272 & 1.0351 \\ 7.7546 & -5.5614 \\ 8.2267 & 7.9708 \end{bmatrix} \quad (2)$$

La distancia de error respecto a la ubicación real de los postes se puede apreciar en la Tabla I.

Poste	Error (en distancia)
1	0.0130
2	0.0060
3	0.0047
4	0.0082
5	0.0101
6	0.0092
7	0.0113
8	0.0154
9	0.0069
10	0.0072
11	0.0215
12	0.0084
13	0.0123

TABLE I: Errores en las estimaciones de las características con robot sin error

Se obtienen las siguientes matrices de covarianza de los postes estimados:

$$\begin{bmatrix} 0.0170 & 0 \\ 0 & 0.0140 \end{bmatrix} \begin{bmatrix} 0.0286 & 0 \\ 0 & 0.0222 \end{bmatrix} \begin{bmatrix} 0.0256 & 0 \\ 0 & 0.0210 \end{bmatrix} \begin{bmatrix} 0.0220 & 0 \\ 0 & 0.0166 \end{bmatrix} \begin{bmatrix} 0.0081 & 0 \\ 0 & 0.0148 \end{bmatrix} \begin{bmatrix} 0.0222 & 0 \\ 0 & 0.0179 \end{bmatrix} \begin{bmatrix} 0.0173 & 0 \\ 0 & 0.0089 \end{bmatrix} \begin{bmatrix} 0.0256 & 0 \\ 0 & 0.0185 \end{bmatrix} \begin{bmatrix} 0.0177 & 0 \\ 0 & 0.0118 \end{bmatrix} \begin{bmatrix} 0.0179 & 0 \\ 0 & 0.0089 \end{bmatrix} \begin{bmatrix} 0.0162 & 0 \\ 0 & 0.0163 \end{bmatrix} \begin{bmatrix} 0.0076 & 0 \\ 0 & 0.0166 \end{bmatrix} \begin{bmatrix} 0.0160 & 0 \\ 0 & 0.0066 \end{bmatrix} \quad (3)$$

Viendo entonces la Tabla I y las Figuras 1 y 2, es posible notar que la estimación realizada por el robot sin error es tanto precisa como exacta.

II. Pregunta 2

Ahora se realiza el mismo experimento que en la pregunta 1 en el entorno 1, pero con la diferencia de que el robot a utilizar es *robot2*, el cual no es ideal y posee errores odométricos. Cabe destacar que el LiDAR se encuentra ubicado en el robot sin error.

El algoritmo de Clustering y determinación de características es exactamente el mismo. Se utiliza el camino y entorno utilizado en la pregunta 1 (ver Figura **??**).

La nube de puntos clasificada por grupos y las estimaciones de los postes se pueden apreciar en la Figura **??**.

La matriz de características se puede apreciar en la ecuación **??**

$$\begin{bmatrix} -3.0735 & 1.4686 \\ 1.4118 & 0.2372 \\ 0.1338 & 2.1727 \\ 4.5400 & 3.1012 \\ -1.3879 & 5.2119 \\ 1.9920 & 6.8434 \\ -3.1672 & 0.9203 \\ 2.1031 & 6.7770 \\ -1.7513 & -4.0407 \\ 3.4216 & -1.4463 \\ -3.0839 & -6.5414 \\ 2.2255 & -7.6407 \\ 8.1836 & 0.9916 \\ 7.4697 & -5.5984 \\ 1.1946 & 6.8818 \\ 7.3553 & 8.3987 \\ -2.0184 & 5.0097 \end{bmatrix} \tag{4}$$

La distancia de error respecto a la ubicación real de los postes se puede apreciar en la Tabla II

Poste	Error (en distancia)
1	0.2497
2	0.2483
3	0.2068
4	0.1458
5	0.0118
6	0.0419
7	0.3552
8	0.1678
9	0.4381
10	0.2943
11	0.5426
12	0.4702
13	0.3305
14	0.2954
15	0.7752
16	0.9670
17	0.6615

TABLE II: Errores estimaciones robot con error

Se obtienen las siguientes matrices de covarianza

$$\begin{bmatrix} 0.0168 & 0 \\ 0 & 0.0121 \end{bmatrix} \begin{bmatrix} 0.0174 & 0 \\ 0 & 0.0197 \end{bmatrix}$$
$$\begin{bmatrix} 0.0155 & 0 \\ 0 & 0.0210 \end{bmatrix}$$
$$\begin{bmatrix} 0.0269 & 0 \\ 0 & 0.0352 \end{bmatrix} \begin{bmatrix} 0.0036 & 0 \\ 0 & 0.0160 \end{bmatrix}$$
$$\begin{bmatrix} 0.0095 & 0 \\ 0 & 0.0287 \end{bmatrix}$$
$$\begin{bmatrix} 0.0001 & 0 \\ 0 & 0.0129 \end{bmatrix} \begin{bmatrix} 0.0039 & 0 \\ 0 & 0.0022 \end{bmatrix}$$
$$\begin{bmatrix} 0.0216 & 0 \\ 0 & 0.0088 \end{bmatrix} \tag{5}$$
$$\begin{bmatrix} 0.0269 & 0 \\ 0 & 0.0263 \end{bmatrix} \begin{bmatrix} 0.0275 & 0 \\ 0 & 0.0120 \end{bmatrix}$$
$$\begin{bmatrix} 0.0432 & 0 \\ 0 & 0.0209 \end{bmatrix}$$
$$\begin{bmatrix} 0.0279 & 0 \\ 0 & 0.0909 \end{bmatrix} \begin{bmatrix} 0.0205 & 0 \\ 0 & 0.0581 \end{bmatrix}$$
$$\begin{bmatrix} 0.0185 & 0 \\ 0 & 0.0125 \end{bmatrix}$$
$$\begin{bmatrix} 0.0167 & 0 \\ 0 & 0.0062 \end{bmatrix} \begin{bmatrix} 0.0068 & 0 \\ 0 & 0.0166 \end{bmatrix}$$

Como era de esperar, la estimación en el robot ideal tiene errores notablemente menores respecto al robot con error (ver Tablas I y II).

Como es posible apreciar, la estimación de los postes no es del todo precisa, notando que estima otros postes muy cercanos a otros. Esto se debe a que el robot si posee error odométrico, el cual a medida que avanza en el trayecto extenso trazado, se aleja cada vez más con respecto al robot ideal.

Los datos del LiDaR son tomados desde la posición del robot ideal, lo que tiene mucho sentido ya que en la vida real el LiDaR estará donde el robot real este.

Posterior a esto, al ocupar la función DeteccionPostes creada, esta ocupará las posiciones del robot con error odométrico, lo que cambiará evidentemente las matrices de rotación y traslación para la nube de puntos recolectada.

Como ejemplo, al momento de comenzar el trayecto se captará una nube de puntos correspondiente a un poste 'n'. Al momento de ya casi finalizar el trayecto trazado, el robot con error odométrico se desviara significativamente, por ende, al captar nuevamente la nube de puntos del poste 'n', esta estará corrida con respecto al comienzo. Es por esto que, la función clusteringNube define ambas nubes de puntos captadas como postes distintos, que a fin de cuentas, corresponden al mismo poste.

Para corregir este problema es necesario agrupar las estimaciones con sus determinados postes (posible en el código con la variable `posteYgrupo` de la función `ClusteringNube`) y realizar un promedio ponderado (dando mayor ponderación a los puntos con menor error) entre

todos los puntos candidatos. De esta manera la cantidad de estimaciones será igual a la cantidad de postes. Esto puede mejorar el error, pero como se considerarán todos los grupos asociados a un mismo punto, debido al error, su covarianza aumentará.

III. Pregunta 3

A. Filtro de Kalman Extendido (EKF)

El filtro de Kalman extendido (EKF, por sus siglas en inglés) es una extensión no lineal del filtro de Kalman, utilizado para estimar el estado de un sistema no lineal a partir de mediciones que contienen ruido. EKF combina la idea de propagación de la media y la covarianza de un filtro de Kalman junto con la linealización de un sistema no lineal a través de una expansión de Taylor de primer orden. Las ecuaciones que corresponden al sistema a analizar poseen la siguiente estructura.

$$\begin{aligned} X_{k+1} &= f(X_k, u_k) + s_k \\ Z_k &= h(X_k, u_k) + l_k \end{aligned} \tag{6}$$

en donde X_k corresponde a los estados no medibles del sistema, mientras que Z_k corresponde a las salidas medibles del sistema. Por otro lado, las funciones $f()$ y $h()$ son funciones no lineales que describen la dinámica del sistema, mientras que s_k y l_k son ruidos gaussianos independientes de media cero y covarianza fija.

En el contexto de un robot con un LIDAR 2D que necesita localizarse, el EKF es de gran utilidad. En este caso, el mapa M del entorno del robot sera conocido y consta de una serie de características que el LIDAR puede detectar. El robot utiliza el EKF para fusionar la información de su sistema de navegación inercial y las mediciones del LIDAR para obtener una estimación precisa de su ubicación.

Para empezar a describir las matrices del sistema, debemos conocer las ecuaciones dinámicas del sistema. Se poseen las entradas conocidas $u_k = [V_k, W_k]^T$, del sistema. Además conocemos como el robot (x_v, y_v, θ_v) se debe mover con respecto a un eje de referencia global del mapa (con el periodo de muestreo Δ del sistema conocido), de la forma:

$$\begin{aligned} x_{v_{k+1}} &= x_{v_k} + \Delta V_k cos(\theta_{v_k}) \\ y_{v_{k+1}} &= y_{v_k} + \Delta V_k sin(\theta_{v_k}) \\ \theta_{v_{k+1}} &= \theta_{v_k} + \Delta W_k \end{aligned} \tag{7}$$

Además, debido a que las características del mapa (postes) son estáticas, es decir, no se mueven, podemos decir que las característica (en coordenadas globales) a medida que pasa el tiempo se modelan como $\begin{bmatrix} X_{1_{k+1}} & Y_{1_{k+1}} & ... & X_{n_{k+1}} & Y_{n_{k+1}} \end{bmatrix}^T = \begin{bmatrix} X_{1_k} & Y_{1_k} & ... & X_{n_k} & Y_{n_k} \end{bmatrix}^T$.

Por otro lado, para describir la ecuación de salida del sistema, tenemos que recordar que el LIDAR 2D nos entrega una nube de puntos de un poste, por ende, una vez aplicado la detección y el clustering explicado en la pregunta 1, se obtienen mediciones de postes de la forma $\begin{bmatrix} r_1 & \theta_1 & ... & r_n & \theta_n \end{bmatrix}^T$. Asi, asociando estos postes guardados en nuestros estados del sistema X_k en coordenadas globales, podemos describir las ecuaciones del sistema como:

$$\begin{aligned} X_{k+1} &= \begin{bmatrix} x_{v_{k+1}} \\ y_{v_{k+1}} \\ \theta_{v_{k+1}} \\ X_{1_{k+1}} \\ Y_{1_{k+1}} \\ \vdots \\ X_{n_{k+1}} \\ Y_{n_{k+1}} \end{bmatrix} = \underbrace{\begin{bmatrix} x_{v_k} + \Delta V_k cos(\theta_{v_k}) \\ y_{v_k} + \Delta V_k sin(\theta_{v_k}) \\ \theta_{v_k} + \Delta W_k \\ X_{1_k} \\ Y_{1_k} \\ \vdots \\ X_{n_k} \\ Y_{n_k} \end{bmatrix}}_{f(X_k, u_k)} + s_k \\ Z_k &= \begin{bmatrix} r_1 \\ \theta_1 \\ \vdots \\ r_n \\ \theta_n \end{bmatrix} = \underbrace{\begin{bmatrix} \sqrt{(X_1 - x_{v_k})^2 + (Y_1 - y_{v_k})^2} \\ atan2\left(\frac{Y_1 - y_{v_k}}{X_1 - x_{v_k}}\right) - \theta_{v_k} \\ \vdots \\ \sqrt{(X_n - x_{v_k})^2 + (Y_n - y_{v_k})^2} \\ atan2\left(\frac{Y_n - y_{v_k}}{X_n - x_{v_k}}\right) - \theta_{v_k} \end{bmatrix}}_{h(X_k, u_k)} + l_k \end{aligned} \tag{8}$$

Como se menciono anteriormente, el EKF trabaja con la linealización del sistema no lineal a través de una expansión de Taylor de primer orden. Es por esto que, las ecuaciones correspondientes al EKF necesita los Jacobianos de las funciones del estado $f()$ y de las salidas $h()$, con respecto al estado completo X_k. De esta forma podemos describir los Jacobianos $(\nabla F, \nabla H)$, mostrados en la figura 3.

El filtro ocupa estos Jacobianos obtenidos, para desarrollar dos pasos clave del filtro:

- **Las ecuaciones de predicción:** Predice la próxima estimación del estado y la covarianza basándose en el estado actual, con la dinámica del sistema conocida.
- **Las ecuaciones de actualización:** Actualiza la estimación del estado y la covarianza en función de la medición actual.

Prediccion:

$$\begin{aligned} \hat{X}_k &= f(\hat{X}_{k-1}, u_{k-1}) \\ P_{k/k-1} &= \nabla F_k P_{k-1} \nabla F_k^T + G_k Q_k G_k^T \end{aligned}$$

Actualización:

$$\begin{aligned} S_k &= \nabla H_k P_{k/k-1} \nabla H_k^T + R_k \\ K_k &= P_{k/k-1} \nabla H_k^T / S_k \\ P_k &= (I - K_k \nabla H_k) P_{k/k-1} \\ v_k &= Z_k - h(\hat{X}_{k/k-1}) \\ \hat{X}_k &= \hat{X}_{k/k-1} + K_k v_k \end{aligned} \tag{10}$$

Se debe notar que G_k corresponde al Jacobiano de $f(X_k, u_k)$ con respecto a las entradas u_k, para así multiplicar a la covarianza Q_k de la señal de entrada u_k (solo si hay), pero que en este caso particular, como son entradas deterministicas, $Q_k = 0$, por ende el factor $G_k Q_k G_k^T$ no afecta en nuestro caso en la etapa de predicción del EKF.

$$
\begin{aligned}
\nabla F_k = \frac{\partial f(X_k, u_k)}{\partial X_k} &= \begin{bmatrix} 1 & 0 & -\Delta V_k sin(\theta_{v_k}) & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & \Delta V_k cos(\theta_{v_k}) & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} dF_{xv}(X_{xv}) & 0 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & I \end{bmatrix} \\
\nabla H_k = \frac{\partial h(X_k, u_k)}{\partial X_k} &= \begin{bmatrix} \frac{-(X_1 - x_{v_k})}{r_1} & \frac{-(Y_1 - y_{v_k})}{r_1} & 0 & \frac{(X_1 - x_{v_k})}{r_1} & \frac{(Y_1 - y_{v_k})}{r_1} & \dots & 0 & 0 \\ \frac{(Y_1 - y_{v_k})}{r_1^2} & \frac{-(X_1 - x_{v_k})}{r_1^2} & -1 & \frac{-(Y_1 - y_{v_k})}{r_1^2} & \frac{(X_1 - x_{v_k})}{r_1^2} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots \\ \frac{-(X_n - x_{v_k})}{r_n} & \frac{-(Y_n - y_{v_k})}{r_n} & 0 & 0 & 0 & \dots & \frac{(X_n - x_{v_k})}{r_n} & \frac{(Y_n - y_{v_k})}{r_n} \\ \frac{(Y_n - y_{v_k})}{r_n^2} & \frac{-(X_n - x_{v_k})}{r_n^2} & -1 & 0 & 0 & \dots & \frac{-(Y_n - y_{v_k})}{r_n^2} & \frac{(X_n - x_{v_k})}{r_n^2} \end{bmatrix} \\
&= \begin{bmatrix} dH_{xv}(X1, Y1) & dH_f(X1, Y1) & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ dH_{xv}(Xn, Yn) & 0 & \dots & dH_f(Xn, Yn) \end{bmatrix}
\end{aligned} \tag{9}
$$

Fig. 3: Jacobianos de matrices $f(X_k, u_k)$ y $h(X_k, u_k)$.

El filtro de Kalman extendido se inicializa con las características ya conocidas de manera exacta, por ende, la covarianza inicial de estas características es igual a cero, agregándose a la diagonal en forma de 2x2:

$$
\hat{X}_0 = \begin{bmatrix} X_{v_0} \\ Y_{v_0} \\ \theta_{v_0} \\ X_1 \\ Y_1 \\ \vdots \\ X_n \\ Y_n \end{bmatrix} \qquad P_0 = \begin{bmatrix} 1 & 0 & 0 & \dots & \dots \\ 0 & 1 & 0 & \dots & \dots \\ 0 & 0 & 0.1 & \dots & \dots \\ \dots & \dots & \dots & 0 & 0 \\ \dots & \dots & \dots & 0 & 0 \end{bmatrix} \tag{11}
$$

B. Data association

En esta parte del algoritmo se realiza la asociación de datos en un filtro de Kalman extendido, asociando las mediciones del LIDAR con los estados estimados del robot y actualizando las matrices relevantes (H_k y R_k) y el vector de estimaciones de las mediciones del LIDAR. Esto es un paso crítico en el EKF, ya que asegura que las actualizaciones del filtro se basen en mediciones correspondientes al estado actual del robot. la descripción del algoritmo se encuentra a continuación:

Se inicializa H_K con ceros que representa la matriz jacobiana de la función de medición de las características observadas por el LIDAR en ese instante con respecto a los cambios de la pose del robot (posición y orientación) , por otro lado se inicializa R_K como una matriz cuadrada de puros ceros que representa la covarianza de las mediciones de cada características detectada en ese instante. También se inicializa una matriz llamada asociacionMed , la cual tiene los puntos medidos por el LIDAR en coordenadas globales y una columna extra con ceros.

for $t = 1$ **to** n **do**
  for $i = 1$ **to** *length*(*posteMed_G*(:, 1)) **do**
    threshold = CalculateDistance(Med(i), Predict(t)) ;
    if *threshold* < 0.6 **then**
      UpdateH_k(Med$(i, :)$, robot_hat$, t, i)$;
      UpdateR_k(i) ;
      UpdateEstimatedZ_k(robot_hat$, t, i, k)$;
      asociacionMed$(i, 3)$ = t ;
      break ;
    end
  end
end

Algorithm 1: Data Association en EKF

En el Algoritmo 1, se inicia con un bucle que itera sobre todas las características conocidas del mapa, M, que también están presentes en la estimación del vector de estado del sistema. Esta iteración utiliza un índice t. Anidado dentro de este bucle, hay otro bucle que itera sobre las características medidas en el instante actual por el LIDAR, utilizando un

índice i.

En cada iteración, se calcula la distancia euclidiana entre una característica 't' y todas las características medidas 'i'. Si alguna de estas distancias es menor que un umbral predefinido, se procede a asociar la característica conocida con la característica medida que cumple con este criterio de proximidad.

Dentro de la condición del umbral, se actualiza la matriz H_k de la siguiente manera: se calcula el jacobiano de la pose del robot con respecto al punto de medición i, y se evalúa en la matriz dH_{xv}. Este jacobiano se agrega a H_k. A continuación, se calcula el jacobiano de la medición i con respecto a la característica medida y se inserta en la columna correspondiente a i en H_k , asociando así el valor de la medición con la estimación del vector de estado en el filtro de Kalman.

Posteriormente, se agrega la covarianza de la medición 'i' a la matriz R_k. Esto se hace insertando la covarianza en la posición diagonal correspondiente a i en R_k.

Las estimaciones de las mediciones, $\hat{Z}_k$, se reordenan de acuerdo con las asociaciones establecidas con las estimaciones. Esto asegura que las mediciones y las estimaciones estén alineadas correctamente para el calculo del vector de innovación. Finalmente en asociacionMed se agrega una etiqueta diferente de cero a cada medición.

La funcionalidad de asociacionMed cuando se conoce las características de M, es eliminar las estimaciones anteriores de las características que ya no son visibles por el LIDAR , al tener una tercera columna con la etiqueta se puede coincidir entre lo que se mide y estima, caso contrario el valor se mantiene en cero si es que ya no hay coincidencias. Luego todas las mediciones que posean una etiqueta igual a 0, se asocian con las filas de H_K, R_K y $\hat{Z}_k$ para no ser consideradas en estas mismas.

Ejemplo de 3 características estimadas en $\hat{Z}_k$ y 2 mediciones de LIDAR Z_k

$$\hat{Z}_k = \begin{bmatrix} 1 & 9 \\ 3 & 3 \\ 4 & 6 \end{bmatrix} \tag{12}$$

$$Z_k = \begin{bmatrix} 1 & 9 \\ 4 & 6 \end{bmatrix} \tag{13}$$

Como se deja de ver un punto que esta presente en la estimación de la medición $\hat{Z}_k$, la matriz asociacionMed queda la siguiente forma :

$$asociacionMed = \begin{bmatrix} 1 & 9 & 1 \\ 3 & 3 & 0 \\ 4 & 6 & 2 \end{bmatrix} \tag{14}$$

en este caso solamente la fila 2 de asociacionMed es cero , por lo que esa es la fila de las otras matrices que se debe eliminar antes de hacer la actualización del vector estado con la matriz de innovación.

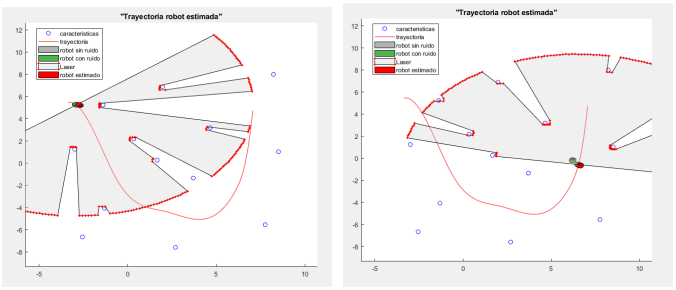
La matriz Jacobiana ∇H para este caso es:

$$\nabla H_k = \begin{bmatrix} dH_{xv}(X1,Y1) & dH_f(X1,Y1) & 0 & 0 \\ 0 & 0 & 0 & 0 \\ dH_{xv}(X3,Y3) & 0 & 0 & dH_f(X3,Y3) \end{bmatrix} \tag{15}$$

que al eliminar las filas correspondientes al estado que ya no se esta viendo, obtenemos las matrices de interés ocupadas en la etapa de actualización del EKF:

$$\nabla H_k = \begin{bmatrix} dH_{xv}(X1,Y1) & dH_f(X1,Y1) & 0 & 0 \\ dH_{xv}(X3,Y3) & 0 & 0 & dH_f(X3,Y3) \end{bmatrix}$$
$$\hat{Z}_k = \begin{bmatrix} 1 & 9 \\ 4 & 6 \end{bmatrix} \tag{16}$$

A continuación se muestra en la figura 4 la implementación del EKF para estimar la pose del robot a través de una trayectoria, logrando satisfactoriamente este propósito.



(a) Comienzo de la estimación de la Trayectoria.

(b) Final de la estimación de la Trayectoria.

Fig. 4: Estimación de la trayectoria del robot (robot rojo) mediante EKF, con conocimiento del ambiente.

IV. Pregunta 4

Para este caso como no se conoce las características inicialmente, se obtiene una primera medición del Laser, que estos primeros puntos captados (los cuales si poseen una covarianza global, ya que es una nube de puntos por poste), se agregan al vector de estados inicial al igual que la covarianza inicial. Estas covarianzas se agregan a la diagonal en forma de 2x2. De esta forma el estado inicial del filtro de Kalman para este caso es:

$$\hat{X}_0 = \begin{bmatrix} X_{v_0} \\ Y_{v_0} \\ \theta_{v_0} \\ X_1 \\ Y_1 \\ \vdots \\ X_n \\ Y_n \end{bmatrix} \qquad P_0 = \begin{bmatrix} 1 & 0 & 0 & \dots & \dots \\ 0 & 1 & 0 & \dots & \dots \\ 0 & 0 & 0.1 & \dots & \dots \\ \dots & \dots & \dots & R_{nx} & 0 \\ \dots & \dots & \dots & 0 & R_{ny} \end{bmatrix} \tag{17}$$

A. *Data association y Add feature*

De manera similar al Algoritmo 1, no hay distinción entre conocer todas las características del mapa M o no conocerlas. Cuando el mapa M no es conocido, la función del Filtro de Kalman Extendido (EKF) incorpora nuevas características que son medidas por el LIDAR. En este contexto, la función asociacionMed no solo se encarga de extraer las filas de las mediciones estimadas $\hat{Z}_k$ que ya no están presentes en las mediciones del LIDAR, sino que también extrae de Z_k las nuevas características medidas y las almacena en dos variables, add_feature para las características y add_CovFeature para las covarianzas de estas nuevas mediciones. Estas características se incorporan tras la actualización del vector de estado del filtro.

Ejemplo de 2 características estimadas en $\hat{Z}_k$ y 3 mediciones de LIDAR Z_k

$$\hat{Z}_k = \begin{bmatrix} \hat{z}_1 \\ \hat{z}_2 \end{bmatrix} = \begin{bmatrix} 1 & 9 \\ 4 & 6 \end{bmatrix} \tag{18}$$

$$Z_k = \begin{bmatrix} z_1 \\ z_3 \\ z_2 \end{bmatrix} = \begin{bmatrix} 1 & 9 \\ 3 & 3 \\ 4 & 6 \end{bmatrix} \tag{19}$$

Como hay un nuevo punto que esta presente en la medición Z_k , la matriz asociacionMed queda la siguiente forma :

$$asociacionMed = \begin{bmatrix} 1 & 9 & 1 \\ 3 & 3 & 0 \\ 4 & 6 & 2 \end{bmatrix} \tag{20}$$

en este caso solamente la fila 2 de asociacionMed posee etiqueta igual a cero, por lo que esa es la fila de las otras matrices (Z_k) que se debe eliminar antes de hacer la actualización del vector estado con la matriz de innovación. Se deben guardar las características de la nueva medición y sus covarianzas de la forma add_feature = Z_k(2,:) y add_CovFeature = covMed3.

La matriz Jacobiana ∇H_k para este caso es:

$$\nabla H_k = \begin{bmatrix} dH_{xv}(z1) & dH_f(z1) & 0 \\ dH_{xv}(z2) & 0 & dH_f(z2) \end{bmatrix} \tag{21}$$

Al eliminar de la medición Z_k la fila correspondiente del estado que no se conoce, podremos realizar correctamente la etapa de actualización del EKF:

$$Z_k = \begin{bmatrix} 1 & 9 \\ 4 & 6 \end{bmatrix} \tag{22}$$

Para agregar add_feature = Z_k(2,:) y add_CovFeature = covMed2, se debe hacer una vez que hemos actualizado el vector de estado. En el código, esto se realiza en la sección etiquetada como add_feat. Primero, se verifica si hay características nuevas para agregar, lo cual se hace comprobando si la variable add_feature no está vacía. Si hay características nuevas, se crea un bucle para transformar estas características al marco de referencia global utilizando la función g(), y se almacenan en add_featGLOBAL.

Luego, se expande la matriz de covarianza P, con la nueva covarianza de la nueva medición, de la forma:

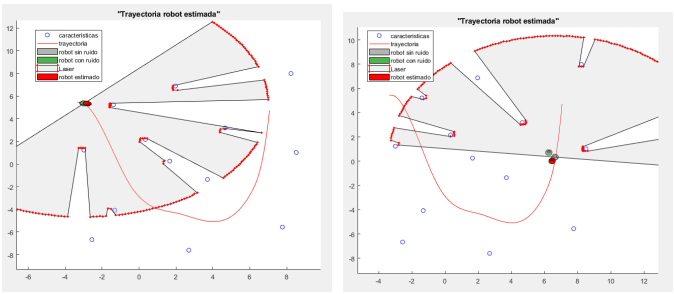
$$P = \begin{bmatrix} P & 0 \\ 0 & \text{covMed3} \end{bmatrix} \tag{23}$$

Se debe destacar que no hay problema al agregar directamente la covarianza de la nueva medición, ya que en el código implementado, esta covarianza ya esta obtenida mediante coordenadas globales.

Finalmente, las nuevas características transformadas a coordenadas globales se agregan al vector de estado $\hat{X}$ en la fila correspondiente al instante de tiempo actual.

$$\hat{X}_k = \begin{bmatrix} \hat{X}_k \\ X3 \\ Y3 \end{bmatrix} \tag{24}$$

A continuación se muestra en la figura 5 la implementación del EKF para estimar la pose del robot a través de una trayectoria, logrando satisfactoriamente este propósito.



(a) Comienzo de la estimación de la Trayectoria.

(b) Final de la estimación de la Trayectoria.

Fig. 5: Estimación de la trayectoria del robot (robot rojo) mediante EKF, sin conocimiento del ambiente.

V. Discusión

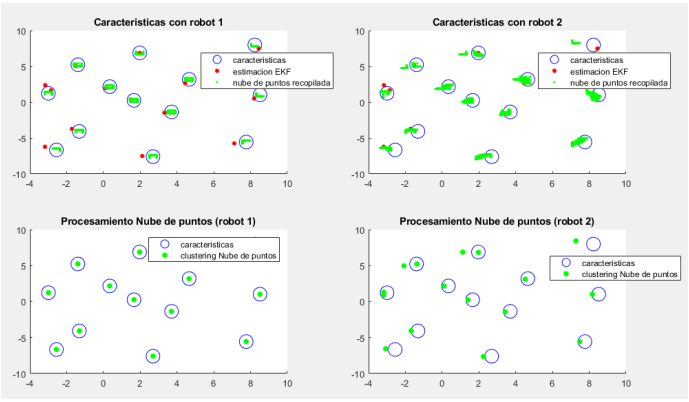


Fig. 6: Procesamiento de nube de puntos y estimaciones de EKF para robot basados en odometría pura y ruidosa (robot1 y robot2 respectivamente)

Al examinar la Figura 6, se observa una caracterización distintiva de la nube de puntos basada en dos escenarios

diferentes de odometría del robot: pura y ruidosa. Donde se utiliza odometría pura (denominada robot1), la estimación de la posición y orientación del robot muestra un error mínimo en comparación con las características originales del mapa. Esto indica una alta precisión en la localización del robot. Por otro lado, la que se basa en odometría con ruido (robot2), muestra un error significativamente mayor en la estimación. Esto sugiere que la presencia de ruido en la odometría puede tener un impacto adverso en la precisión de la localización del robot. Es importante destacar que la calidad de la odometría es un factor crítico en la eficacia de la localización y mapeo, y es necesario considerar métodos de filtrado o corrección de ruido para mejorar la precisión en entornos prácticos donde la odometría perfecta no es alcanzable.

VI. Conclusiones

La implementación del Filtro de Kalman Extendido (EKF) ha demostrado ser valiosa en la estimación precisa y en tiempo real de la posición y orientación de un robot, especialmente en entornos con incertidumbre. A pesar de su eficiencia computacional y robustez frente al ruido de medición, el EKF requiere una cuidadosa selección de parámetros y puede enfrentar limitaciones en entornos altamente no lineales. Su capacidad para adaptarse a entornos desconocidos al incorporar nuevas características es particularmente útil en aplicaciones de Mapeo y Localización Simultáneos (SLAM). Futuras investigaciones podrían explorar algoritmos más avanzados, como el Filtro de Partículas, Filtro de información y técnicas de optimización basadas en gráficos, para abordar las limitaciones del EKF en entornos complejos.