# Proyecto Final

Julio Sánchez 148221, Diego Amaya 149119, Gumer Rodríŋguez 149109, Gabriel Reynoso 150904

## I. Introducción

LOS objetivos en este proyecto fueron los siguientes, estimar el estado propio del robot con base en una entrada de visión, estimar el estado de un obstáculo móvil con ayuda de un sensor y hacer que el robot de seguimiento a dicho obstáculo. Para estas tareas se utilizaron las herramientas Gazebo (como simulador) y ROS como interfaz y control. Los algoritmos se implementaron en C++. Para lograr la primera tarea su utilizó una bag proporcionada por el profesor, se discretizaron estados y utilizó un filtro de histogramas para lograr la estimación. La segunda tarea se dividió en dos partes, la primera fue estimar la pose (estado) del obstáculo utilizando los datos proporcionados por el LiDAR y un filtro de Kalman, y la segunda fue seguir a dicho obstaculo movil, se implementó un controlador que permite seguir a dicho objeto a una velocidad y a una distancia.
El equipo definió líderes de tareas para optimizar la comunicación y el funcionamiento del mismo, para la primer tarea los encargados fueron Diego y Gabriel, para la segunda tarea los encargados fueron Gumer y Julio, para la última parte de la implementación los 4 integrantes colaboraron de manera activa. En lo concerniente al marco teórico, Julio fue el encargado de la primera parte, Diego de la segunda, Gumer del primer punto de la tercera y Gabriel de los dos puntos restantes. También se definieron tareas como revisión de código (Gumer), administración del repositorio en Github (Diego), edición del video (Gabriel y Julio) y armado del documento final (Gabriel y Julio).

## II. Marco Teórico

### Parte 1.1:

Las lecturas del LiDAR arrojan una nube de puntos con la cual podemos obtener la distancia y el ángulo promedio del obstáculo con respecto al robot. Para este problema no es necesario contar con la rotación del obstáculo, por lo que no hay matriz de rotación. Dicho lo anterior y dado que el eje x apunta al frente de nuestro auto, la representación de la pose del obstáculo con respecto al robot es la siguiente:
$X_{obstáculo} = distanciapromedio * cos(angulo_{promedio})$
$Y_{obstáculo} = distanciapromedio * sin(angulo_{promedio})$

### Parte 1.2:

El sistema de control es retroalimentado con las lecturas del LiDAR. Las lecturas son procesadas y entregadas al filtro de Kalman, el cual estima el estado del obstáculo. El vector de estado del obstáculo nos proporciona la posición y la velocidad en X y en Y. A partir de estas velocidades se calcula la magnitud de la velocidad y posteriormente se evalúa: si la distancia al obstáculo es considerablemente grande, se utiliza un control proporcional a la distancia, la cual multiplica a la magnitud de la distancia y ese resultado es el valor que entrega el sistema de control. En cambio, si la distancia al obstáculo es considerablemente chica, se elimina el control proporcional y el sistema de control entrega la magnitud de la velocidad, la cual es la misma que la velocidad del obstáculo. Con este sistema no sólo se logra igualar la velocidad del obstáculo, sino que también se logra acercarse al obstáculo antes de igualar la velocidad.

### Parte 2.1:

El LiDAR es un dispositivo que permite sensar el medio mediante un haz láser pulsado, la distancia al objeto se determina midiendo el retardo de ida y vuelta desde su emisión hasta la recepción de la señal reflejada. Para hacer la extracción de líneas y caminos es común utilizar otro método que provea datos a parte del LiDAR, como alguna imagen con procesamiento. El LiDAR genera una nube de puntos al escanear el medio ambiente y se reciben datos crudos, lo que sigue es clasificar los puntos y poder identificar rasgos del entorno.[3]
Una vez teniendo la nube de puntos, se pueden identificar aspectos específicos, como obstáculos, líneas, etc. En este caso en específico nos interesan las líneas que trazan un camino para poder seguirlo, por lo tanto, nos interesan los bordes, la pendiente, el ángulo y las dimensiones de la carretera. Esto se puede conocer de manera aproximada si conoces las líneas que caracterizan el camino. Se puede utilizar el algoritmo de Hough para extracción de líneas, sin embargo, el algoritmo previamente conocido era aplicado en un espacio de dos dimensiones, pero aquí se trata con puntos en el espacio, por lo que se hace una extensión de la transformada para aplicarla en el espacio. La transformada pretende identificar figuras geométricas primitivas como líneas, círculos, etc. Para 3D se busca identificar planos en vez de líneas que contengan el mayor número de puntos. De aquí se pueden extraer líneas o curvas para después analizar la información y obtener los parámetros del objeto.

**Algorithm 1: 3D Hough-transform for plane detection**

1. $X\_min = \min(X)$; $Y\_min = \min(Y)$; $Z\_min = \min(Z)$
2. $X\_max = \max(X)$; $Y\_max = \max(Y)$; $Z\_max = \max(Z)$
3. Calculation of: $Dis\_min$; $Dis\_max$
4. $\theta$ = from 0 to 360, step = $\theta\_step$; $n\_\theta = \text{length}(\theta)$
5. $\varphi$ = from -90 to +90, step = $\varphi\_step$; $n\_\varphi = \text{length}(\varphi)$
6. $n\_\rho = 2 * (Dis\_max - Dis\_min) / \rho\_step$
7. $\rho$ = from $Dis\_min$ to $Dis\_max$; step = $\rho\_step$
8. $\theta\_mat(n\_\varphi, n\_\theta) = [\theta\ \theta\ \theta \ldots \theta]' * \pi/180$
9. $\varphi\_mat(n\_\varphi, n\_\theta) = [\varphi\ \varphi\ \varphi \ldots \varphi] * \pi/180$
10. $H(n\_\theta, n\_\varphi, n\_\rho) = 0$
11. $ratio = (n\_\rho - 1)/(\rho(n\_\rho) - \rho(1))$
12. for $k = 1$ to length($X$)
13. $\rho\_mat = \cos(\varphi\_mat) * \cos(\theta\_mat) * X(k) + \ldots$
    $\cos(\varphi\_mat) * \sin(\theta\_mat) * Y(k) + \sin(\varphi\_mat) * Z(k)$
14. $\rho\_indix = \text{round}(ratio * (\rho\_mat - \rho(1) + 1))$
15. for $i = 1$ to $n\_\varphi$
16. for $j = 1$ to $n\_\theta$
17. $H(j, i, \rho\_index(i, j)) = H(j, i, \rho\_index(i, j)) + 1$
18. next $j$ ; next $i$ ; next $k$

Figure 1. Pseudocódigo de la transformada de Hough

Las entradas son $\theta$, $\phi$ and $\rho$ que son los incrementos a los que crecen las variables para el arreglo acumulador, llamados $\theta_{step}$, $\phi_{step}$ y $\rho_{step}$. La nube de puntos 3D es representada por puntos X, Y y Z.[2]

En este algorítmo $Dis_{min}$ and $Dis_{max}$ son las distancias entre el origen y las dos extremidades de la nube de puntos, calculados en la línea 1 y 2 del código. H es una matriz en tres dimensiones, $\theta_{mat}$, $\phi_{mat}$ and $\rho_{mat}$ son matrices de dos dimensiones. El resultado es la matriz H que contiene la representación de la nube de puntos dentro del espacio parametral. Cada uno de los puntos da una superficie sinusoidal del espacio parametral. Se detectan picos en la matriz H, lo que significa que es un plano que contiene a muchos puntos.

## Parte 2.2:

Una imagen puede ser vista como tres matrices en las que se tienen diferentes componentes, como la intensidad de rojo, verde y azul, u otros formatos. Para poder identificar un camino se puede utilizar una imagen provista por una cámara, de ésta se pueden obtener los parámetros geométricos del camino. Para hacer un procesamiento que nos permita reconocer las características del camino, debemos hacer ciertas suposiciones que regulen algunas de las variables que pueden ser influyentes al momento de procesar: se asume que la luz es uniforme, es decir, que no hay grandes sombras que afecten al procesamiento, que se distinguen de manera aceptablemente definida los colores de cada uno de los elementos presentes en la imagen, y por último, que no hay reflejos o efectos visuales, como se pueden presentar en un camino, que interfieran directamente con la cámara. Tomadas estas suposiciones, procedemos a definir el procesamiento.[5] Se tomará como método principal de filtrado la segmentación de color, utilizando como espacio de cromaticidad el HSV, ya que nos permite filtrar colores como el negro y excluir colores con mayor saturación, que están comúnmente en los caminos. Se tienen que ajustar los parámetros de matiz, saturación y valor de tal manera que se filtren los colores objetivo. Si se toma en cuenta que los carriles están separados por una línea continua, se pueden diferenciar los carriles como dos bloques y se e obtiene una imagen binarizada en la que, en teoría sólo se encontraría el camino, dividido en dos carriles. Aquí es más fácil encontrar los bordes del camino utilizando un filtro diferencial o un filtro de Canny. Una vez obtenidos los bordes, podemos utilizar un algoritmo como la transformada de Hough y detectar líneas, las cuales serán los bordes del camino. Se arrojan los puntos que componen a esas líneas con coordenadas como pixeles dentro de la imagen.[6]

Una vez que se tienen los pixeles, es momento de saber cuál es su localización en el espacio. Conocemos el proceso de proyección de perspectiva, en el cual se pierde la profundidad, por lo que experimentalmente el proceso inverso puede resultar no muy preciso. Sea K la matriz intrínseca de la cámara, que se encarga de hacer el escalamiento, la traslación y el ajuste en 2D. [R|t] es la matriz extrínseca de la cámara, que se encarga de la rotación y traslación en 3D, es decir, describe cómo es el mundo con respecto a la cámara. Los otros factores que juegan un papel en la transformación son: C es el centro de la cámara en el mundo, P es el punto en 3D en el mundo, c es la proyección del punto P en el plano de la imagen, w es el factor de escalamiento usado en la transformación de 3D a 2D, â=[a b c] que es el vector normal al plano. La ecuación del plano se denota como ax+by+cz+d=0. El procedimiento para hacer la transformación es como sigue:

$$wp = K[R|t]P$$
$$= K[RP + t]$$
$$= KR[P - C] \quad \text{(Note: Camera center, } C = -R^{-1}t) \quad \text{---------- (1)}$$

Using (1),

$$w[KR]^{-1}p + C = P \quad \text{---------- (2)}$$

$P$ is a point on the plane, $\Rightarrow \hat{n}.P + d = 0$

$$\Rightarrow \hat{n}.(P - C) + d + \hat{n}C = 0$$
$$\Rightarrow \hat{n}.(w[KR]^{-1}p) + d + \hat{n}C = 0$$
$$\Rightarrow w = \frac{(-d - \hat{n}C)}{\hat{n}.([KR]^{-1}p)} \quad \text{---------- (3)}$$

Substituting (3) in (2),

$$\boldsymbol{P = \frac{(-d - \hat{n}C)}{\hat{n}.([KR]^{-1}p)}[KR]^{-1}p + C}$$

Figure 2. Pseudocódigo de la transformada de Hough

## Parte 3.1:

El objetivo del filtro de Kalman consiste en calcular un estimador lineal, insesgado y óptimo del estado de un sistema en t con base en la información disponible en t-1, y actualizar, con la información adicional disponible en t. Para este proyecto estimaremos la posición $(x, y)$ del obstáculo, así como su velocidad en x y y, todo con respecto a nuestro automóvil. El problema general a

resolver es: dado un modelo del sistema y mediciones con ruido, estimar el estado del sistema X.

$$\hat{\boldsymbol{x}}\langle k+1|k\rangle = \boldsymbol{F}\hat{\boldsymbol{x}}\langle k\rangle + \boldsymbol{G}\boldsymbol{u}\langle k\rangle$$

$$\hat{\boldsymbol{P}}\langle k+1|k\rangle = \boldsymbol{F}\hat{\boldsymbol{P}}\langle k|k\rangle\boldsymbol{F}^T + \hat{\boldsymbol{V}}$$

El filtro tiene dos etapas. La primera es la predicción del estado basada en el estado anterior y las entradas aplicadas. Donde x gorro es la estimación del estado (4x1), P gorro la estimación de la covarianza (4x4) y F describe la dinámica del sistema.

$x = [x, y, dx, dy]^T$

El vector u es la entrada al sistema y la matriz G describe como las entradas son reflejadas en los estados del sistema. Desconocemos u, por lo que Gu<k> será cero. V es el ruido (gaussiano), representado por una una matriz identidad (4x4) multiplicada por una constante.

$$\boldsymbol{\nu}\langle k+1\rangle = \boldsymbol{z}\langle k+1\rangle - \boldsymbol{H}\hat{\boldsymbol{x}}\langle k+1|k\rangle$$

Tenemos que introducir nueva información obtenida de los sensores, conocida como innovación que es la diferencia entre lo que los sensores miden y lo que se había predicho ($v < k+1 >$). La matriz H describe como los estados del sistema son mapeados a las salidas del observador, y z es la medición de los sensores.

El segundo paso del filtro de Kalman es la actualización, la cual utiliza la ganancia de Kalman para mapear la innovación al estado predicho en forma de corrección, buscando optimizar la estimación acercándolo a lo observado en los sensores.

$$\boldsymbol{K}\langle k+1\rangle = \hat{\boldsymbol{P}}\langle k+1|k\rangle\boldsymbol{H}^T\underbrace{\left(\boldsymbol{H}\hat{\boldsymbol{P}}\langle k+1|k\rangle\boldsymbol{H}^T + \hat{\boldsymbol{W}}\right)}_{S}^{-1}$$

$$\hat{\boldsymbol{x}}\langle k+1|k+1\rangle = \hat{\boldsymbol{x}}\langle k+1|k\rangle + \boldsymbol{K}\langle k+1\rangle\boldsymbol{\nu}\langle k+1\rangle$$
$$\hat{\boldsymbol{P}}\langle k+1|k+1\rangle = \hat{\boldsymbol{P}}\langle k+1|k\rangle - \boldsymbol{K}\langle k+1\rangle\boldsymbol{H}\hat{\boldsymbol{P}}\langle k+1|k\rangle$$

Es importante señalar que la incertidumbre ahora está disminuida, ya que el segundo término se resta de la covarianza predicha. El término indicado por S es la covarianza estimada de la innovación y proviene de la incertidumbre en el estado y de la covarianza de la medición del ruido.

Utilizamos el filtro de Kalman para determinar la posición de nuestro obstáculo con los datos obtenidos del sensor tipo LiDAR. Para ello definimos nuestro sistema con las matrices:

$$F = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Las condiciones iniciales utilizadas fueron:

$$X = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$P = \begin{bmatrix} 0.1 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix}$$

## Parte 3.2:

El filtro de histograma es la herramienta más sencilla para representar beliefs de forma continua. Dividimos el dominio $dom(x_t)$ en $n$ espacios disjuntos $b_0, ..., b_{n-1}$ de tal forma que la unión de todos los espacios sea igual al dominio, esto es $\cup_i b_i = dom(x_t)$. A continuación, definimos un nuevo estado $x_{t*} \in 0, ..., n-1$ donde el estado $x_{t*} = i$ si solo si $x_t \in b_i$. Como el estado $x_{t*}$ tiene un espacio finito de estados discretos, podemos utilizar el filtro de Bayes discreto para calcular $bel(x_{t*})$. Este filtro nos da la probabilidad para cada $b_i$ que el estado $x_t$ este en ese $b_i$ dado. El filtro de Bayes toma como entradas el belief previo, la acción comandada y la información de los sensores, así el filtro entrega un nuevo belief que vuelve a ser retroalimentado. Es importante mencionar que a mayor cantidad de espacios $b_i$ más precisa la aproximación, pero esto lleva a un mayor costo computacional. También el estado $x_{t*}$ asume la propiedad Markoviana que la probabilidad del estado actual depende únicamente del estado anterior, esto solo funciona si el ambiente es estático y no sufre cambios drásticos en el tiempo de la prueba. [1]

Para implementarlo en este proyecto, se consideraron 8 estados posibles en los que pudiera estar el automóvil. El procesamiento de imágenes indica las líneas de los carriles y mediante código se le asignan probabilidades a los distintos estados posibles, con base en estos, se actualiza la probabilidad de que el carro esté en uno u otro estado. Para lograr esta actualización se toma en cuenta únicamente qué líneas son visibles en un momento dado, pudiendo modificar las probabilidades con cada recepción de datos.

## Parte 3.3:

Para que el robot aprenda a mantenerse en su carril utilizando el método de aprendizaje por refuerzo asumimos lo siguiente: El robot cuenta con una cámara o sensor para ver la carretera en todo momento, además cuenta con un buen procesamiento para mapear dichas líneas en un plano cuyo origen sea el robot. También identificamos 6 estados posibles: estar fuera del carril por la derecha, estar fuera del carril por la izquierda, estar sobre la línea derecha, estar sobre la línea izquierda, estar un poco a la derecha y estar un poco a la izquierda. A los estados se le añaden dos acciones: girar a la izquierda o girar a la derecha. Para definir en que estado nos encontramos

utilizamos la información obtenida de nuestros sensores para encontrar la distancia a las dos líneas observadas, esta distancia se utiliza para identificar el estado en que robot se encuentra y con base en eso definir nuestro GOAL que en este caso sería que ambas distancias (a la línea derecha e izquierda) sean iguales. En el siguiente diagrama se puede observar una representación gráfica de nuestros estados y las transiciones dependiendo de la acción realizada.[4]

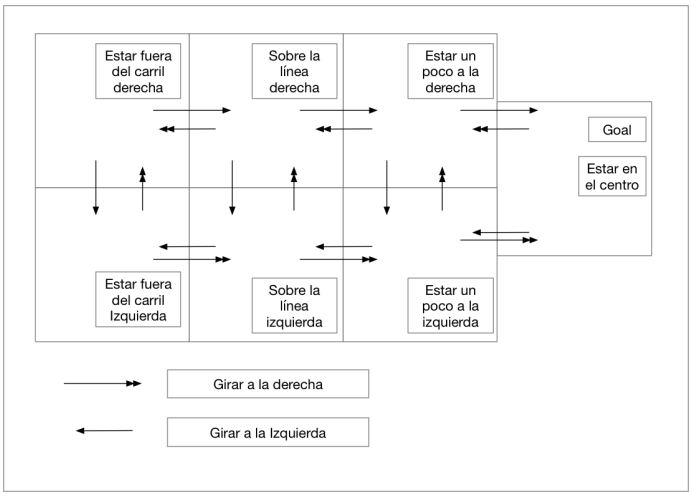


Figure 3. Grid World robot

Por último, se le asigna un mayor factor de descuento en la función valor de cada estado dependiendo que tan cerca esté del GOAL, así el robot aprenderá que acción deberá elegir para maximar su recompensa dado el estado en el que se encuentra.

## III. Esbozo de la solución

### Parte 1:

Primero se definieron los ocho estados posibles: Izquierda Carretera,Sobre la línea izquierda, Carril Izquierdo, Centro de la carretera, Carril Derecho, Sobre la línea derecha, Derecha Carretera y Desconocido . Todos estos contenidos en un arreglo con esa misma configuración. A continuación definimos distintos comportamientos para la probabilidad de estar en cada uno de esos estados, también definidos como arreglos, donde el arreglo seleccionado depende directamente de las líneas observadas en un ciclo determinado.

Se estableció una bandera para cada nube de puntos que define una recta (izquierda, centro y derecha), y son actualizadas cada vez que se reciben datos, si una bandera no cambia se sabe que no se vió esa línea. Una vez que se determina qué líneas fueron vistas se consideran casos. Al entrar a un caso específico, se hace la multiplicación uno a uno de la probabilidad actual con la del posible estado y es normalizada para que la suma sea uno, esto determina el estado para la siguiente iteración. Este estado es publicado en el tópico /estadoEstimado y se reinician a falso las banderas.

El arreglo con los estados es el siguiente:

IzqCar | LinIzq | CenIzq | Cent | CenDer | LínDer | DerCar | Desc

La tabla de asignación de arreglo de probabilidad dependiente en la línea observadas es como sigue:

| Left | Center | Right | Arreglo de Probabilidad |
|---|---|---|---|
| 0 | 0 | 0 | desconocido (a) |
| 0 | 0 | 1 | derecha (b) |
| 0 | 1 | 0 | centro (c) |
| 0 | 1 | 1 | centroDer (d) |
| 1 | 0 | 0 | izquierda (e) |
| 1 | 0 | 1 | centro (c) |
| 1 | 1 | 0 | centroIzq (f) |
| 1 | 1 | 1 | centrado (g) |

Y dichas probabilidades son las siguientes:

| [ ] | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| a | 0.015 | 0.015 | 0.015 | 0.015 | 0.015 | 0.015 | 0.015 | 0.895 |
| b | 0.015 | 0.015 | 0.015 | 0.015 | 0.2 | 0.225 | 0.5 | 0.015 |
| c | 0.015 | 0.015 | 0.05 | 0.825 | 0.05 | 0.015 | 0.015 | 0.015 |
| d | 0.015 | 0.015 | 0.015 | 0.1 | 0.725 | 0.1 | 0.015 | 0.015 |
| e | 0.5 | 0.225 | 0.2 | 0.015 | 0.015 | 0.015 | 0.015 | 0.015 |
| f | 0.015 | 0.1 | 0.725 | 0.1 | 0.015 | 0.015 | 0.015 | 0.015 |
| g | 0.015 | 0.015 | 0.25 | 0.425 | 0.25 | 0.015 | 0.015 | 0.015 |

### Parte 2:

El LiDAR entrega 360 mediciones de distancia por segundo, una para cada ángulo dado en radianes. Dentro de estas 360 mediciones, algunas corresponden al chasis del propio coche ya que el sensor se encuentra debajo del techo. Para poder encontrar la distancia correspondiente al obstáculo, se recorre el arreglo de datos y se ignoran aquellos datos menores a 0.2 (los cuales corresponden a la carrocería) y los que no sean infinito (mediciones del LiDAR que no corresponden a ningún objeto). Cada que se encuentra una medición que cumple con estas dos condiciones (mayor a 0.2 y menores a infinito) se van sumando. Al terminar de recorrer el arreglo de datos del LiDAR, se calcula la distancia promedio: $Distancia_{promedio} = distancia_{acumulada}/n$. De igual forma para el ángulo: $Angulo_{promedio} = ángulo_{acumulado}/n$.

Procesar los ángulos es un poco diferente: para obtener el valor del ángulo en la iteración actual se multiplicaba el valor de dicha iteración con la distancia angular entre mediciones, la cual nos la entrega el LiDAR. A ese producto se le suma el acumulado de ángulos y se actualiza. Sin embargo, existe un problema con la lectura de los ángulos: debido a que el LiDAR recorre $2\pi$ radianes y para cada radian entrega una medida, si promediamos radianes cercanos a 0 y radianes cercanos a $2\pi$ (los cuales hacen referencia a medidas del obstáculo en frente del robot) estamos obteniendo un valor promedio cercano a $\pi$, lo que nos estaría diciendo que está a las espaldas del robot y no en frente. Para resolver este problema pusimos una

condición en la que si se obtienen mediciones antes de la iteración 30 (aproximación debido al tamaño del robot) se activa una bandera. Posteriormente, si se encuentra una medida después de la iteración 330 y la bandera está activa, a ese radian se le resta 2 pi para obtener el valor respecto al centro pero en sentido negativo y de esta forma poder promediar correctamente.

Para calcular el tiempo entre cada medición (deltaT), se usa la librería de ROS para calcular el tiempo transcurrido entre cada medición, ya que dicha medida que supuestamente nos debería de entregar el LiDAR siempre es 0. Sin embargo, por simplicidad lo fijamos en 1, dado experimentalmente obtuvimos que el sensor arrojaba datos una vez por segundo.

Obtenidas la distancia promedio y el ángulo promedio, calculamos sus correspondientes coordenadas x y y con la formula mencionada en la parte 1.1 del marco teórico.

Ya con las coordenadas $x$ y $y$ del obstaculo, se procedió a implementar el filtro de Kalman. Para ello, se hicieron dos métodos, predicción() y actualización(), para ejecutar las dos etapas del filtro. Nos ayudamos de variables tipo matriz implementado las operaciones matriciales correspondientes sin utilizar librerías externas.

Después de filtrar procedemos a publicar el estado en un tópico llamado /pose objetivo, de tipo geometry msgs::Twist, utilizando las variables linear.x y linear.y para las coordenadas, y las variables angular.x y angular.y para las velocidades lineales del objetivo.

### Parte 3:

Como fue sugerido en el documento de las especificaciones del proyecto, se utilizó un programa muy parecido al control del proyecto 2 pero con varias modificaciones para adaptarlo al problema actual: seguir al obstáculo. En el control del proyecto 2 se tenía al "mundo" como referencia, mientras que en este caso el robot es la referencia. También, se cambio al esquema de control "ir a un punto", dado que no nos importa la pose, solo llegar a las coordenadas deseadas, se optó por un control proporcional. Por ultimo, nos subscribimos al tópico resultante del filtro de Kalman para obtener el punto de destino.

Como el robot es la referencia, sus coordenadas son (0,0) y debido a que sólo queremos seguir al obstáculo (llegar a cierto punto en cada iteración) y no queremos llegar a una pose deseada, el control es muy sencillo, las salidas son la velocidad (v) y el ángulo de giro (gamma). Las ecuaciones de control se muestran acontinuación:

$$\gamma = k\gamma * atan(y_d/x_d) * 180/\pi * sign$$

$$v = k\rho * sqrt(x_d * x_d + y_d * y_d)$$

Donde k$\gamma$ y k$\rho$ son las ganancias del ángulo y velocidad respectivamente. $x_d$ y $y_d$ representan la posición del obstáculo y sign nos dice si el obstáculo está a las espaldas o al frente del robot.

Tanto gamma como se publican en los tópicos de control manual de nuestro automóvil.

## IV. Experimentos

### Parte 1:

Para probar el código, se corre la bolsa de ROS y se imprime el estado al que entró la condición al ver las banderas. En otra terminal se hace echo al tópico estadoEstimado, donde se puede ver que converge al estado centrado. Además comparamos el estado impreso por nuestro código contra lo que podemos observar en las imágenes contenidas en la bolsa de la detección de las tres líneas.

### Parte 2:

Dado que el objetivo de esta parte es obtener el estado de del otro automóvil utilizando el LiDAR y el filtro de Kalman, nos decidimos por dos experimentos para probar su correcta funcionalidad: El primero consiste en verificar la detección del estado del obstáculo dentro del rango del LiDAR, es decir, dentro del perímetro del láser. Para ello, dejamos fijo nuestro automóvil y le damos una velocidad al obstáculo para que salga de los limites del sensor. En el segundo experimento probamos la detección correcta del estado del obstáculo alrededor de los 360 grados de nuestro automóvil. Dejamos fijo nuestro robot y damos instrucciones al obstáculo para hacer un circulo alrededor de éste.

### Parte 3:

El objetivo de esta parte es seguir al obstáculo móvil, por lo que consideramos necesarios tres experimentos para verificar una completa funcionalidad. El primer experimento es simple, seguir al obstáculo en linea recta para probar que el automóvil iguala velocidad y se detiene cuando llega a estar muy cerca para evitar colisiones. Para el segundo experimento planteamos un reto mas completo, seguir al obstáculo en múltiples direcciones en una trayectoria mas larga, para lo cual debemos dirigir al otro robot a través de la publicación de tópicos en la linea de comandos. Planeamos una ruta en forma de serpiente. El objetivo del tercer experimento fue escapar de la persecución, dado que el LiDAR se encuentra dentro del automóvil y el láser choca con las paredes de la carrocería, tenemos cuatro zonas o ventanas ciegas. Dando instrucciones de trayectorias muy forzadas al obstaculo, tratamos de llevarlo a una de estas zonas y utilizarla como ruta de escape.

El link de Google Drive con el video que muestra el funcionamiento es:

https://drive.google.com/open?id=1wyU1Z710jF6q_1p19PAoiZ1Q8jpiKJ3T

## V. Conclusiones

Un coche autónomo es un sistema muy complejo, sujeto a muchas variables que pueden producir errores aleatorios, por esto es necesario crear un sistema que pueda tomar en cuenta este tipo de perturbaciones y no perder su ubicación o entrar en una singularidad. El tener entradas de sensores exteroceptivos puede proveer de información precisa sobre el entorno, sumado con un sistema probabilístico que de manera retroactiva haga estimaciones sobre el estado actual, resulta muy robusto. Un robot está en continua retroalimentación entre su modelo del mundo, lo que lee del mundo y lo que pasa en el mundo, por lo tanto, entre más interacción tengan esas facciones, el modelo puede ser más preciso y converger de manera más rápida a un valor, con una certidumbre bastante alta.

Tuvimos resultados muy satisfactorios en todas las partes del proyecto:
1. Se hace una estimación coherente del estado de un auto con el filtro de histogramas
2. Se detecta con mucha exactitud la posición de un obstaculo, menos en las ventanas ciegas del LiDAR.
3. Nuestro auto es capaz de seguir un obstaculo móvil de una manera muy fluida y en cualquier trayectoria, siempre y cuando, éste no se escape por una zona ciega del LiDAR.
Sin embargo, hay muchas cosas que se pueden mejorar todavía. Para la primera parte, seria bueno ver lo que hay dentro de cada nube de puntos, procesar dicha información y utilizarla para tener mejores estimaciones en cada ciclo. En cuando a la segunda y tercera parte, proponemos una primer mejora al modelo del automóvil en Gazebo, donde el LiDAR tenga una visión completa del entorno, es decir, que no sea bloqueado por la misma carrocería del vehículo, ya que esto nos ocasionó algunos problemas de detección como se puede observar en los experimentos realizados. Como segundo paso, o paso alternativo, implementar un algoritmo que trate de recuperar al obstáculo cuando este sale del rango visible utilizando las estimaciones que ya tenemos del filtro de Kalman.

Hacer un coche autónomo se ha vuelto una meta a alcanzar en la industria automotriz actual, y por lo que podemos ver, hay escenarios que no se pueden controlar del todo, pero conforme avanza la tecnología y se actualizan las técnicas sobre estimaciones y mediciones, se incrementa la posibilidad de dotar de robustez a un carro autónomo, la cual puede ser implementada en cualquier otro sistema, como pueden ser los drones

## VI. Bibliografía

[1] Daniel. (2017). Robot Localization II: The Histogram Filter. 18/12/2017, de deep ideas Sitio web: http://www.deepideas.net/robot-localization-histogram-filter

[2] Fayez Tarsha-Kurdi, Tania Landes, Pierre Grussenmeyer. Hough-Transform and Extended RANSAC Algorithms for Automatic Detection of 3D Building Roof Planes from Lidar Data. ISPRS Workshop on Laser Scanning 2007 and SilviLaser 2007, Sep 2007, Espoo, Finland. XXXVI, pp.407-412, 2007. [En línea] Disponible: https://halshs.archives-ouvertes.fr/halshs-00264843/document

[3] A. Miraliakbari, M. Hahn, H. Arefi, J. Engels. EXTRACTION OF 3D STRAIGHT LINES USING LIDAR DATA AND AERIAL IMAGES. Stuttgart, Alemania. [En línea] Disponible: http://www.isprs.org/proceedings/XXXVII/congress /4pdf/250.pdf

[4]Carlos Greg. (2015). Aprendizaje por refuerzos. 19/12/2017, de UBA Sitio web: http://www.dc.uba.ar/materias/aa/2015/cuat2/refuerzos.

[5]Kowdle Adarsh, Back projection-2D points to3D, Chen lab, Cornell University, [En línea] Disponible: http://chenlab.ece.cornell.edu/people/adarsh/publications /BackProjection.pdf

[6]Simek Kyle, Dissecting the Camera Matrix, Part 3: The Intrinsic Matrix, Sightations, Agosto 13, 2013. [En línea] Disponible: http://ksimek.github.io/2013/08/13/intrinsic/

[7] Peter Corke. Robotics, Vision and Control. Springer, 2011