Universidad de Buenos Aires
Facultad De Ingeniería
Año 2022 - 1er Cuatrimestre

TP especial Vigilancia y exploración.
Robótica Móvil(86.48)

2do. Cuatrimestre
Curso: MAS

INTEGRANTES:
Fuentes Acuña, Brian Alex - #101785
<bfuentes@fi.uba.ar>

Índice

Resumen

Se abordan los desafíos de **Vigilancia** y **Exploracion** en donde se implementa el algoritmo necesario en Matlab 2020a, haciendo uso del ToolBox Robotics para facilitar la implementación y no reinventar la rueda.
En el problema de vigilancia el robot debe partir de una posición aleatoria dentro del un mapa dado y moverse hasta un punto A, esperar tres segundos, luego ir a un punto B. para ello se divide el problema en localización, planeamiento de la ruta y el control de ruta para ello se a utilizado localización de monte Carlo (MCL), Hybrid A* y un control PD respectivamente.
En el problema de exploración el robot se encuentra totalmente deslocalizado y solo posee información de un lidar, este debe moverse y mapear el entorno, para ello se hace uso del algoritmo de graph-slam.
Todos estos se probaran en un simulador al cual se le carga el siguiente mapa de ocupacion.

Figura 1: Mapa de prueba.

1. Desafío 1: Vigilancia

El desafío de vigilancia consiste primero en ubicar al robot dentro de un mapa, luego generar una trayectoria hacia una meta para asi aplicar un control que en conjunto con el algoritmo de ubicación va siguiendo la trayectoria hasta la meta. entonces los tres pilares de este desafío residen en:

- Localizacion(Localizacion de monte carlo, MCL)
- Planeamiento de ruta (Hybrid A*)
- Controlador (Proporcional)

Para ello se procede a explicar cada parte por separado.

1.1. Localización (MCL)

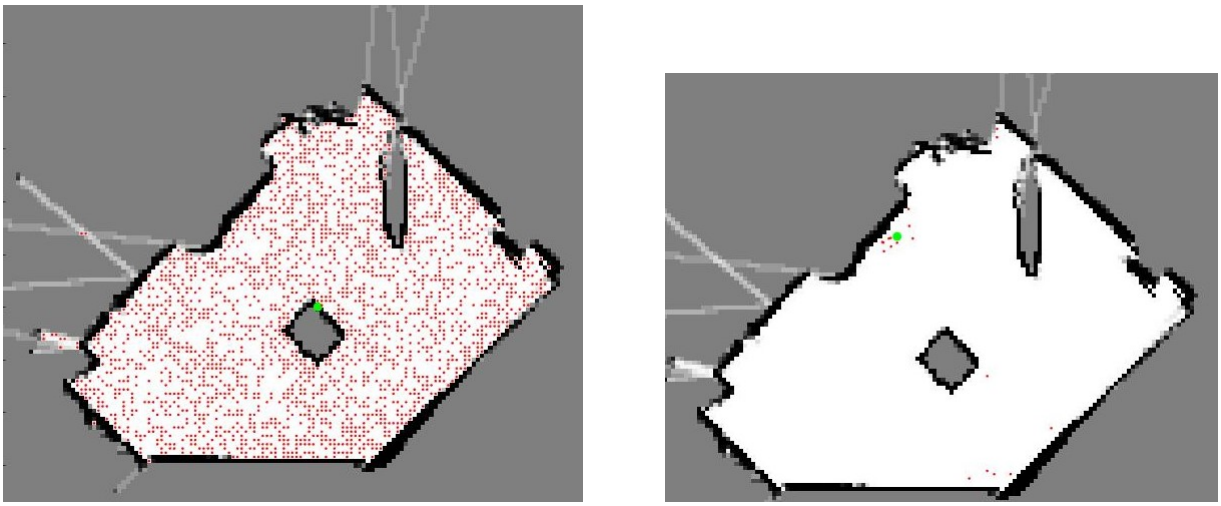
Se tiene el mapa de la Figura 1 en donde hay mucho espacio y salvo por los objetos en medio y arriba el mapa no parece tener caracteristicas que se puedan aprovechar como landmarks, asi que las tecnicas de localizacion con landmarks vistas no son posibles de aplicar, ademas de que por lo general estos se aplican cuando se dispone de una camara.
Por ello solo nos queda el filtro de particulas con poses desconocidas, asi entonces se decide usan la localizacion de monte carlo el cual usa un filtro de particulas para estima la pose y es posible configurar la distribucion inicial de particulas facilmente. una brebe descripccion del MCL seria:

"Para localizar el robot, el algoritmo MCL utiliza un filtro de partículas para estimar su posición. Las partículas representan la distribución de los estados probables del robot. Cada partícula representa un posible estado del robot. Las partículas convergen alrededor de una sola ubicación a medida que el robot se mueve en el entorno y detecta diferentes partes del entorno mediante un sensor de rango. El movimiento del robot se detecta mediante un sensor de odometría. "

Las partículas se actualizan en este proceso:

1. *Las partículas se propagan según el cambio de pose y el modelo de movimiento especificado.*
2. *A las partículas se les asignan pesos en función de la probabilidad de recibir la lectura del sensor de rango para cada partícula. Esta lectura se basa en el modelo de sensor.*
3. *En función de estos pesos, se extrae una estimación del estado del robot en función de los pesos de las partículas. El grupo de partículas con mayor peso se utiliza para estimar la posición del robot.*
4. *Finalmente, las partículas se vuelven a muestrear en función del valor especificado. El remuestreo ajusta las posiciones de las partículas y mejora el rendimiento al ajustar la cantidad de partículas utilizadas. Es una característica clave para adaptarse a los cambios y mantener las partículas relevantes para estimar el estado del robot.*

Por supuesto mientras más partículas mejor será la estimación a costa del costo computacional por esta vez se deja un mínimo de 500 partículas y un máximo de 5000 para que cuando el robot este deslocalizado use el rango máximo de partículas y estas irán disminuyendo a medida que se localiza al robot dentro del mapa.



(a) Particulas iniciales: 5000 (b) Particulas finales 501

Figura 2: Variacion de particulas 500 a 5000

En la figura 2-a se puede ver la posición inicial estimada del robot como un punto verde dentro del mapa, y alrededor una distribución de partículas uniformemente distribuidas dentro esto para lograr inicialmente la localización global del robot, si se tuviera alguna estimación inicial del robot en el mapa se usaría una inicialización con distribución normal alrededor de la estimación. En la figura 2-b se puede observar como las partículas fueron disminuyendo hasta encontrar el mejor estado marcado en verde, por supuesto todavía hay diversidad de partículas puesto que se nota como del lado opuesto todavía hay partículas alejadas de la media, esto es las características similares del entorno de mediciones, irán desapareciendo a medida que el robot se mueva.

1.2. Planeamiento de ruta (Hybrid A*)

Teniendo en cuenta que se da 3 minutos para completar el desafio de vigilancia uno puede pensar que quiza lo que masp uede tardar sea el generador de trayectorias (path plannig) por lo que se requiere uno que sea rapido incluso si se debe recalcular la trayectoria en algun moment (no fue el caso) entonces se descarta el algortimo Djstra por que es un algortimo lento a comparacion de A* ademas de que se tiene el mapa junto con el dato de la meta, entonces A* es la mejor opccion dado que como se vio en los anteriores TPs es el mas rapido y aficas, pero A* por si solo tiene un problema y es que las rutas que genera pueden ser demasiado directas, es decir que puede haber cambios bruscos de direccion y por lo general entra en juego las restricciones no holonomicas es decir que el robot podria no poder generar las velocidades necesarias, en nuestro caso se tienen limitaciones de velocidad pero esto iguamente se soluciono interpolando puntos para obtener los necesarios en un determinado tiempo y que este no supere las velocidades maximas o recomendadas. Pero hay un problema y es que en un cambio brusco de direccion el robot practicamente debera parar y reorientarse por lo que perdera tiempo en la re-orientacion por lo general son deseables trayectorias suaves y es ahi donde entra hybrid A* que puede generar distintas direcciones de desplazamientos(primitivas) dentro de una o mas grillas es decir posibles caminos para tomar el de menor costo y ademas estas primitivas seran curvas suaves que mejoraran la ruta hacia la meeta.
El algoritmo Hybrid A es una extensión del algoritmo A* regular para superar sus deficiencias* En la figura 3 se puede ver la comparacion entre A* y Hybrid A*, notar como disminuyen la cantidad

de vecinos , esto hace que sea mas rapido en la ejecucion.ademas su pose esta ligado a la grilla pero puede tener una posicion por asi decirlo con mayor resolucion gracias a la implementacion de las primitivas.

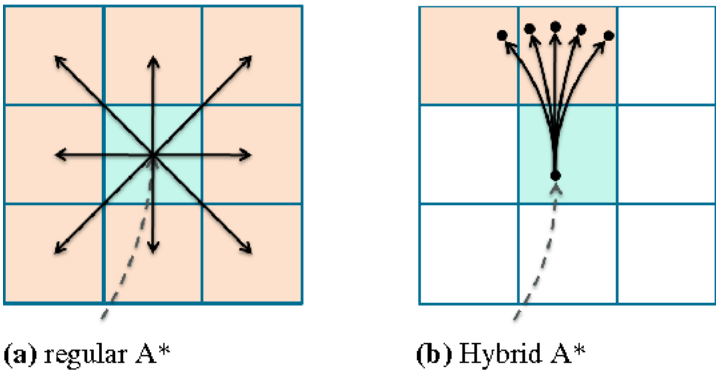
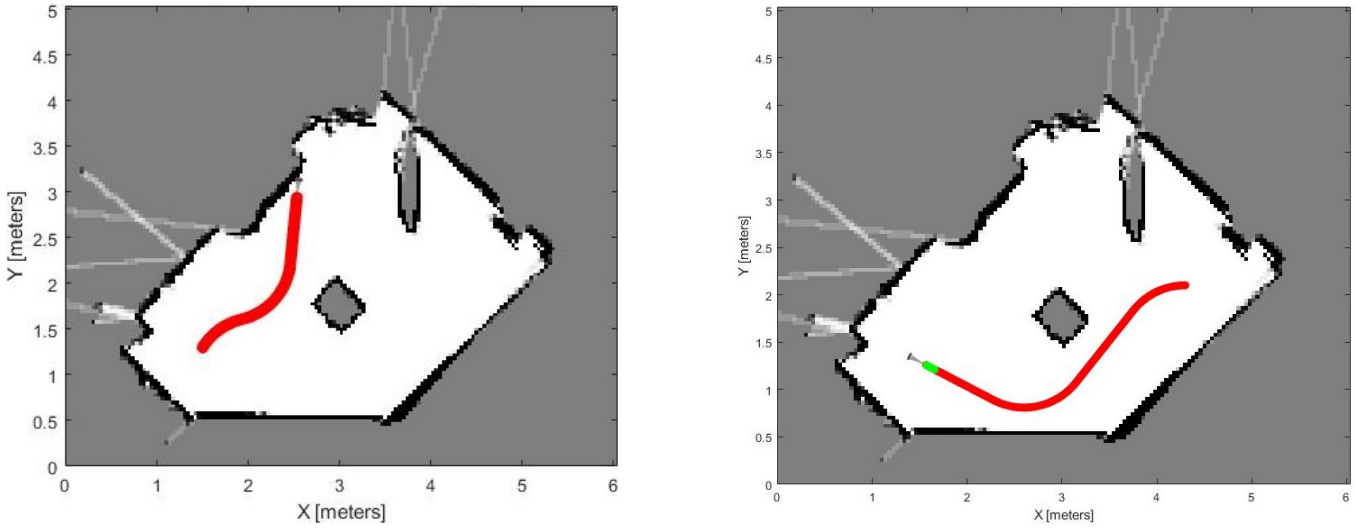


Figura 3: Hybrid A*

Por supuesto que Hybrid A* no es perfecto, a diferencia de A* este no necesariamente te da la curva optima, pero si una curva que intenta ser suave, dado hybrid A* para generar su primitivas tiene en cuenta una longitud y un radio estas restricciones pueden hacer que para llegar a la meta el robot tenga que retroceder para reorientarse. entonces si uno quiere evitar este tipo de problemas debe prestar especial atención al Angulo inicial y final de la trayectoria dado que de este depende la correcta implementación, por ejemplo si el robot inicialmente esta orienta a $pi/2$ y la meta a $-pi/2$ la trayectoria generada tendrá dos curvas concatenadas que generan un pico en la trayectoria, esto tira abajo la ventaja de hybrid A* por sobre A* e incluso si se la usa puede que el robot tenga que ir en retro para cumplir con el ángulo de llegada.
Para solucionar este problema se implementó una función que reorienta al robot haciendo que apunte hacia la meta, en realidad mientras la meta este entre $+-\pi/2$ ya es suficiente para generar una ruta suave.
Esto último fue aplicado y si obtuvieron las siguientes rutas planeadas con Hybrid A*.



(a) Ruta hacia el puntoA (b) Ruta hacia el puntoB

Figura 4: Rutas generadas por Hybrid A*

Notar que en la figura 4-a el robot puede ver a la meta perfectamente por lo que la ruta optima puede ser una simple línea recta y de hecho esto puede hacerse así simplemente restringiendo la reorientación. pero si se observa la figura 4-b puede que al restringir la reorientación entonces la ruta generada al no estar dentro del campo de visión se puedan generar picos debido al radio de giro elegido.

1.3. Controlador

Finalmente queda la etapa de control, pero habiendo tantos controles, PID, por realimentacion de estados, LQR, controles no lineales etc... la pregunta es ¿Cual deberia usar y por que? ¿entre mas complejo el control mejor?.
Al final lo mas simple es mejor y relamente lo que se está haciendo es el control de posción del robot para que siga la trayectoria deseada por lo que lo mejor seria arrancar con algo facil e ir mejorandolo de ser necesario.
Entonces sabiendo que se trabaja con un robot diferencial, se tiene el modelo del mismo:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}cos(\theta) & 0\\ sen(\theta) & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}v\\ w\end{bmatrix} \tag{1}$$
$$\dot{X} = J\dot{Q}$$

entonces lo que uno quiere es minimizar el error entre una posción deseada X_D y la verdadera pose X_{real} entonces se define el error como:

$$X_e = X_D - X_{real}$$

derivando:

$$\dot{X}_e = \dot{X}_D - \dot{X}_{real}$$

despejando:

$$\dot{X}_{real} = \dot{X}_D - \dot{X}_e$$

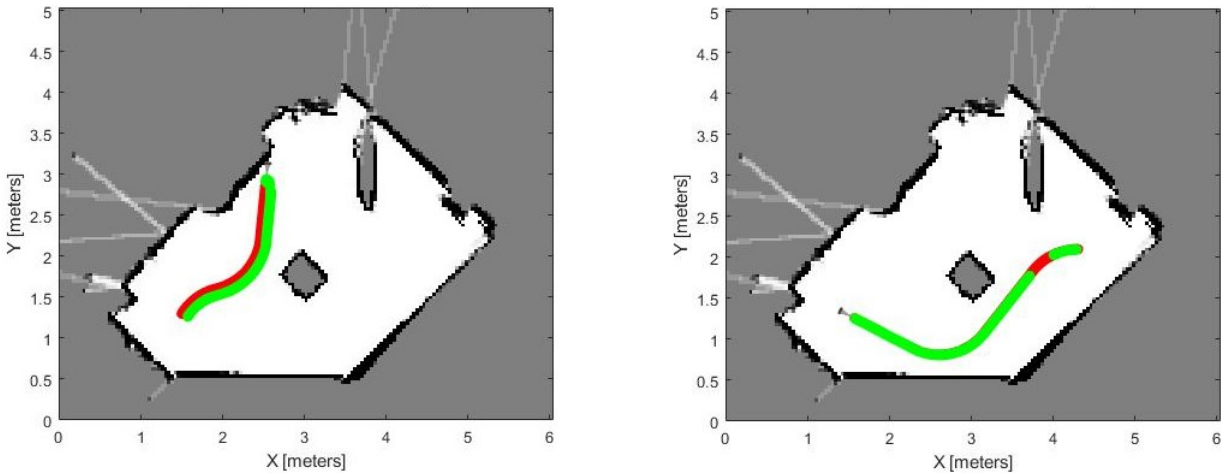
luego se reemplaza en (1):

$$\dot{X}_D - \dot{X}_e = J\dot{Q} -> \dot{Q} = J^{-1}(\dot{X}_D - \dot{X}_e)$$

ahora bien por simplicidad se asume $\dot{X}_d = 0$ en principio por que la pose deseada es constante. luego queda ver como obtener $\dot{X}_e$ para ello hay que hacer un analisis de estabilidad, y como J no es lineal se usa la estabilidad de Lyapunov para ello se busca una funcion candida de Lyapunov $V = \frac{Xe'.Xe}{2}$ en donde $\dot{V} = X'_e\dot{X}_e < 0$ entonces se propone $\dot{X}_e = -K.X_e$ por lo que ahora teniendo en cuenta que J es singular y no cuadrada la inversa no existe, entonces se usará la pseudoinversa (se podria correr virtualmente el lidar respecto de robot para que almenos J de 2x2 sea inversible, pero estaria muy cerca de la singularidad). Tambien se hace uso de las velocidades deseadas dado que ahora mismo no se estará haciendo el control para un solo punto sino para toda una trayectoria, entonces como primera aproximacion se usa:

$$\dot{Q} = pinv(J(\theta))(\dot{X}_{deseado} + K.X_e) \tag{2}$$

y asi finalmente se tiene un control PD que gracias al jacobiado se puede controlar las velocidades de comando y asi poder controlar las posiciones.
En la siguiente figura se puede ver cómo funciona este control (rojo: trayectoria deseada, verde: posiciones del robot), en principio para partir desde una posición inicial hacía en punto A lo hace bastante bien pero le cuesta seguir la trayectoria con cero error, esto es propio de un PD y además hay que sumarle que la trayectoria deseada es en realidad la trayectoria estimada por el filtro de partículas. aun así llega a la meta con un error tolerable (0.1m aprox) luego al llegar al punto A el robot espera 3 segundos sin hacer nada pero en ese tiempo el filtro sigue funcionando y luego empieza a reorientarse hacia la meta, con velocidad lineal nula, esto hace que las partículas estén realmente representen la verdadera pose y finalmente esto se ve reflejado en el seguimiento de trayectoria en la figura 5-b donde el seguimiento es perfecto (se borra parte de la verdadera pose del robot para que se vea el solapamiento).



(a) Control hacia el puntoA (b) Control hacia el puntoB

Figura 5: Control de posición PD

1.4. Resultados

Finalmente, luego del ejecutar todo el algoritmo se vio como el robot sigue la trayectoria bastante bien y evita colisiones, para este caso se decidió no convolucionar el mapa dado que si el robot pueda estar inicialmente en cualquier parte del mapa entonces un caso extremo como el mostrado en las imágenes haría imposible la generación de la trayectoria dado que al luego de ser localizado la posición en la que el robot podría dejar de ser valida. y se produciría fallo en la generación de rutas. entonces finalmente el robot llego a la meta exitosamente en dos minutos y 50 segundos como se ve en la figura 6. Este tiempo es aceptable, pero puede mejorarse dado que las velocidades lineales fueron en promedio fueron menores a las recomendadas, además de que la trayectoria se genera usando un tiempo esperado, un minuto en cada caso, puede mejorarse.

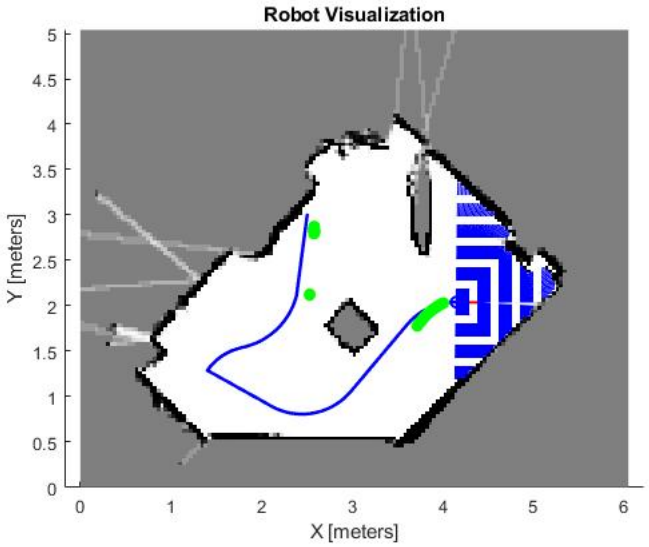


Figura 6: Vigilancia

2. Exploración

El problema de exploración es básicamente un problema de SLAM pero hay muchas formas de SLAM para ser implementadas de entre ellas las vistas son full-slam, fast-slam pero en los ejemplos estas se resolvieron utilizando landmarks, ósea que de alguna forma se tenía definido, en este caso si bien el mapa es conocido en la simulación, no se lo usa como tal sino que solo la información dada por el robot es usada, odometría y ranges de un sensor lidar.
Entonces full slam por lo general guarda los mapas de todas la partículas esto tiene un elevado costo computacional y fast slam requiere fue implementado con poses conocidas las que además con landmarks esto se ayuda bastante, en este caso no se tiene nada de esto pero podría usarse SLAM con scan-maching para mejorar la odometria, ósea usar ICP, pero esta técnica permite obtener una rotación y traslación relativa, ¿entonces por qué no usar esto para generar el mapa? bueno de hecho son más confiables las mediciones del lidar que las de la odometria, de alguna forma se puede generar una odometria virtual simplemente con dos escaneos simultáneos.
Bueno resulta que esto alguien ya lo pensó y se llama Graph-Slam

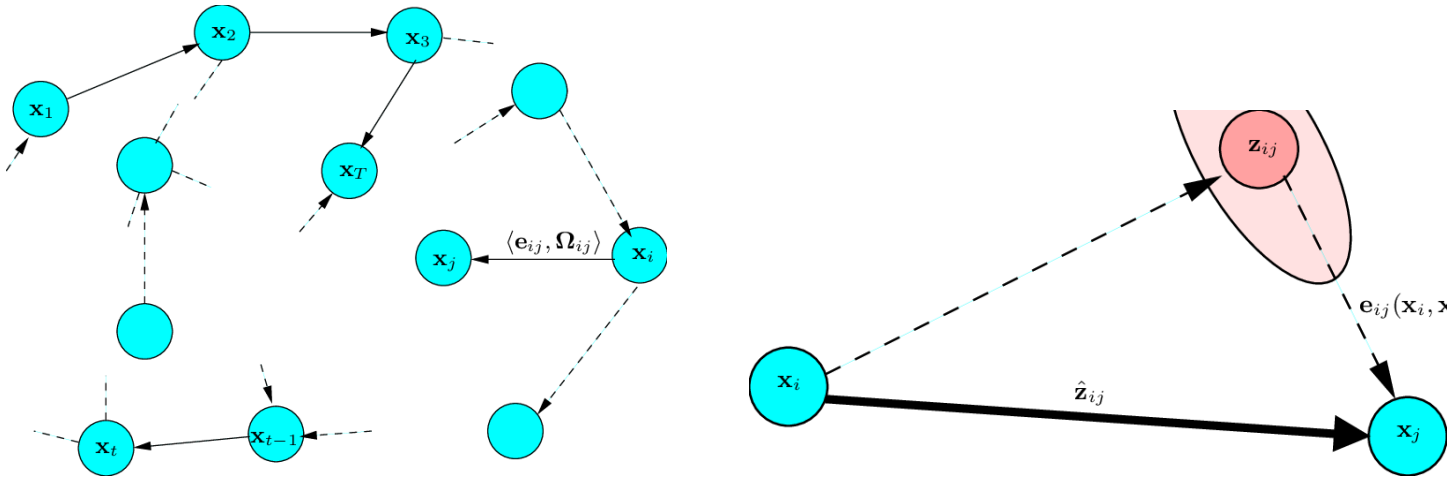
2.1. Graph-Slam

Ün enfoque SLAM basado en grafos construye un problema de estimación simplificado al abstraer las mediciones del sensor sin procesar. Estas medidas sin procesar se reemplazan por los bordes(edges) en el gráfico que luego se pueden ver como "medidas virtuales". "

Entonces para resumir Graph-Slam es un problema de slam como los ya vistos en donde ahora cada POSE del robot es representada como un nodo X_i y cada posición del nodo está conectada por un .edge"(son las flechas en la figura 7) este edge representa la odometria virtual entre nodos, es decir cuánto se desplazó un nodo respecto al anterior. Por supuesto por cada nodo hay una medición del entorno, el mapa, y estas se van superponiendo a medida que se suman los nodos y asi se forma el mapa. La ventaja de esto es que como se muestra en la figura 7-b uno puede corregir minimizar los errores porque si bien las mediciones de un lidar son más confiables no son ideales, e_{ij} es el error cometido y está definida como $e_{ij}(x_i, x_j) = z_{ij} - \hat{z}_{ij}(x_i, x_j)$ donde $\hat{z}_{ij}(x_i, x_j)$ es la observación esperada y z_{ij} es la verdadera observación.
y ya está, en principio es todo lo que hay que saber para entender el funcionamiento dado que después solo queda minimizar el error de la siguiente forma:

$$\begin{aligned} F(x) &= \sum_{<i,j>\in C} e_{ij}^T \Omega_{ij} e_{ij} \\ X* &= armin_x F(x) \end{aligned} \tag{3}$$

donde Ω_{ij} es la matriz de informacion virtual entre el nodo i y el j.



(a) Representacion de poses Graph-Slam (b) Aspectis de edge entre nodos

Figura 7: Grap-Slam

En la figura 7: El edge se origina debido a z_{ij}, para la posicion relativa entre dos nodos es posible computar la medicion esperada z_{ij}
Un edge se caracteriza completamente por su función de error $e_{ij}(x_i, x_j)$ y por la matriz de información Ω_{ij} (inversa de la matriz de covarianza) de la medida que da cuenta de su incertidumbre
Graph-Slam se puede entender por dos partes, la del front-end y back-end, en front-ent se construyen el grafo el cual consiste en obtener las poses estimadas, los edges(arcos) mediciones. mientra que en la parte back-end se obtimiza ese grafo haciendo uso de las restricciones y asi macheando distintas partes del grafico mapa total. En este caso se hace uso del Graph-Slam online, es decir que a medida que se crean los nodos se van macheando de ser posible.

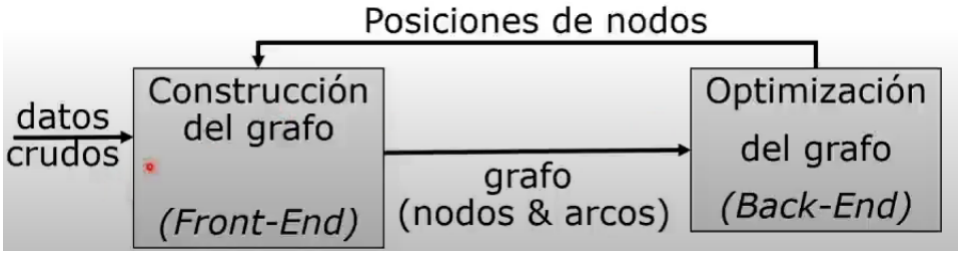


Figura 8: Back-end y Front-End

Nota: lo interesante de esto es que lo único necesario para agregar nueva información al grafo encontrar alguna manera de agregar la restricción, esto hace que sea más flexible el agregar nueva información de distintos sensores.

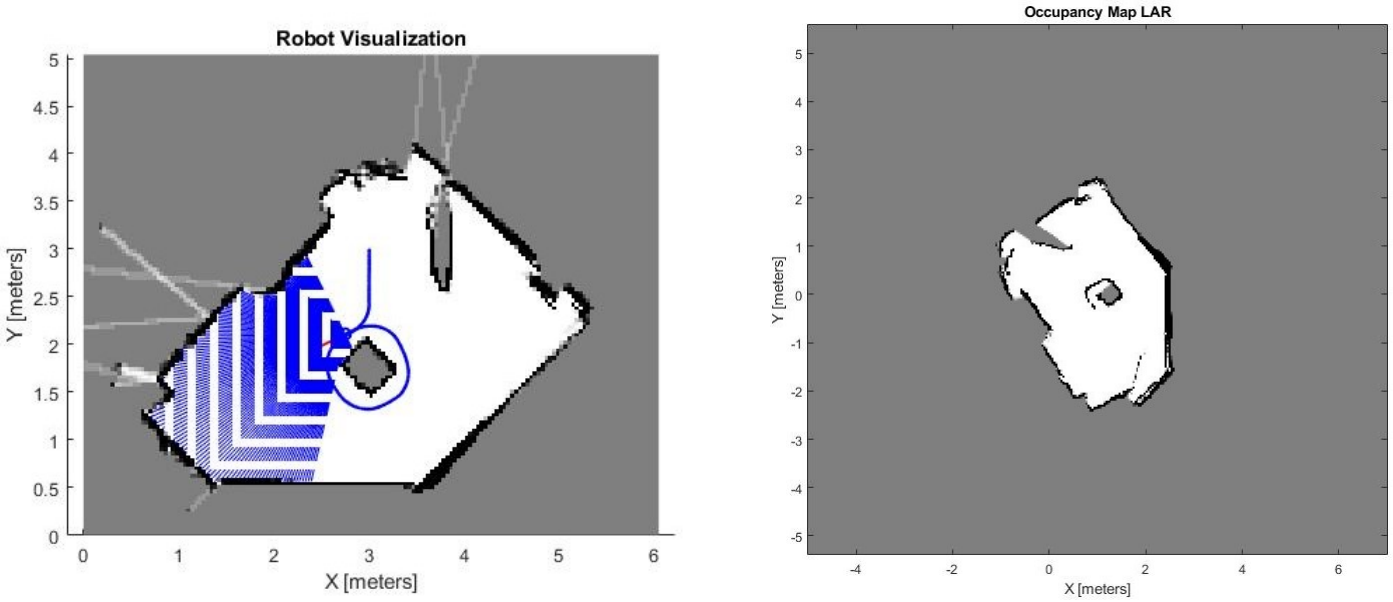
2.2. Navegacion

Ahora resuelto el problema de SLAM queda otro problema implícito el cual se lo trata aparte por su inesperada importancia.
Uno puede pensar el problema de SLAM como si fuese un ciego que tiene un bastón muy largo y

puede censar muchas veces el medio y en base a eso se arma un mapa mental del entorno.
Bueno lo cierto es que el lidar hace algo similar pero el çiego"debe moverse para generar el mapa mental, de nada sirve censar mi alrededor si no me muevo, entonces lo que hace es en principio moverse desde una posición inicial el cual será su cero, y por lo general una persona aun queriendo moverse recto termina moviéndose en círculos así que es lo que el algoritmo *super_blind* va a hacer solo que al detectar demasiada cercanía con las paredes se moverá en sentido contrario y siempre tendrá preferencia por ir a lugares desconocidos, es decir a donde el rango es máximo, de esa manera suele evitar estancarse en esquinas muy pronunciadas.

2.3. Resultados

En la figura 9-a se puede ver cómo es que el algoritmo *super_blind* hace uso de la información dada y en base a ello se mueve, tal como lo haría un ciego.
En la figura 9-b se puede ver el mapa generado usan Graph-Slam online.



(a) Trayectoria realizada (b) Mapa obtenido por Grap-Slam

Figura 9: Resusltados: Graph-Slam

Notar que los macheos no son perfectos, esto es debido juntamente al scan-maching, en donde si se tiene demasiado desplazamiento entre mediciones el macheo puede ser erróneo o tardar demasiado, entonces el problema acá es que la velocidad angular es muy grande, esto puede ajustarse según el entorno y es una cuestión por mejorar.
también es cierto que hay zonas del mapa que no fueron exploradas, es una mejora a hacer y es juntamente ir a zonas donde el mapa no fue explorado, pudiendo hacerse algún análisis de entropía o simplemente darle la preferencia de ir a zonas donde la probabilidad sea 0.5, ósea total desconocimiento.

3. Conclusiones

Los algoritmos planteados pueden hacerse tan complejos como uno quiera, pero esto no garantiza que funcione mejor, a priori cuanto más simple mejor y luego se irá mejorando.
Hybrid A* es un algoritmo muy rápido por lo general tarda 100ms en generar la trayectoria por lo que se lo podría usar repetidamente en caso de necesitar recalcularla. a su vez la calidad de la trayectoria es ve comprometida por las orientaciones iniciales y finales.
El controlador PD si bien es muy simple cumple con lo pedido así que no será necesario cambiarlo de momento, aunque siempre se puede agregar una parte integral. Si bien Graph-Slam es lo más reciente, ya se pueden encontrar bastantes implementaciones y trabajos de este, ofrecer buenos resultados y si se le quiere agregar información de otros sensores la única dificultad recae en entrar las restricciones necesarias.
Encontrar un algoritmo que permita movernos en un entorno desconocido es muy complejo dado que hay demasiados casos particulares, pero por lo general haciendo analogías a situaciones de la vida cotidiana ayuda bastante a resolverlo, la mejor opción es ver como lo resolvió alguien en las condiciones que se encuentra el robot, por ejemplo el ciego.

4. Bibliografia

J. Petereit, T. Emter, C. W. Frey, T. Kopfstedt and A. Beutel, .ªpplication of Hybrid A* to an Autonomous Mobile Robot for Path Planning in Unstructured Outdoor Environments,ROBOTIK 2012; 7th German Conference on Robotics, 2012, pp. 1-6.

G. Grisetti, R. Kümmerle, C. Stachniss and W. Burgard, .ª Tutorial on Graph-Based SLAM,ïn IEEE Intelligent Transportation Systems Magazine, vol. 2, no. 4, pp. 31-43, winter 2010, doi: 10.1109/MITS.2010