Introducción a la Robótica Móvil

Trabajo Práctico Final

Carolina Lucía Gonzalez

24 de febrero de 2023

El objetivo del presente trabajo es desarrollar un sistema para un robot omnidireccional concreto que le permita seguir una trayectoria precisa, unificando así los conceptos estudiados durante la materia (odometría, control a lazo cerrado y filtro de Kalman, entre otros). La presentación del mismo consiste en el presente informe, que documenta el trabajo realizado, y en una serie de archivos, que conforman la implementación.

1. Introducción

Un robot móvil es aquel que tiene la capacidad de desplazarse por su entorno. Podemos clasificarlos de varias maneras, pero en este trabajo estamos interesados en la relación entre los grados de libertad totales y los grados de libertad controlables. De esta manera, podemos distinguir entre dos tipos: *holonómicos* y *no holonómicos*. Los holonómicos (también llamados *omnidireccionales*) son aquellos en los que el número de grados de libertad controlables es igual al de grados de libertad totales, o dicho de otra manera, pueden desplazarse en cualquier dirección sin la necesidad de alcanzar previamente una orientación determinada, lo cual permite realizar trayectorias muy diversas y esto es de especial utilidad para aplicaciones como el transporte de carga. Por otro lado, los no holonómicos suelen ser más sencillos. Un ejemplo de ellos es el clásico auto con tracción en sus dos ruedas traseras, que presenta limitaciones en el movimiento como el problema de estacionar en paralelo.

Algunas aplicaciones de robots holonómicos (y en particular, de un estilo similar al usado en este trabajo) incluyen desde sillas de ruedas motorizadas [5], transporte carga [1], exploración de entornos peligrosos o difíciles [4], hasta incluso jugar al fútbol [3]. Y aunque las aplicaciones sean muy distintas, podemos decir que las principales problemáticas a resolver en cada caso son esencialmente las mismas, entre las que podemos mencionar:

- ¿A qué velocidad deben girar las ruedas del robot para que éste alcance una determinada velocidad?
- ¿A qué velocidad debe moverse el robot para poder seguir una determinada trayectoria?
- ¿Cómo se puede ajustar la velocidad en base a la pose que tiene el robot?
- ¿Cómo integramos la información provista por los sensores?

Luego por supuesto podemos agregar más preguntas a lista, relacionadas específicamente con la aplicación (por ejemplo, cómo se comunica con los demás robots en el campo de juego), pero la base es muy similar a la que estudiaremos en este trabajo. Por eso, más allá del valor educativo, este proyecto es interesante por ser de hecho factible aplicarlo en la vida real.

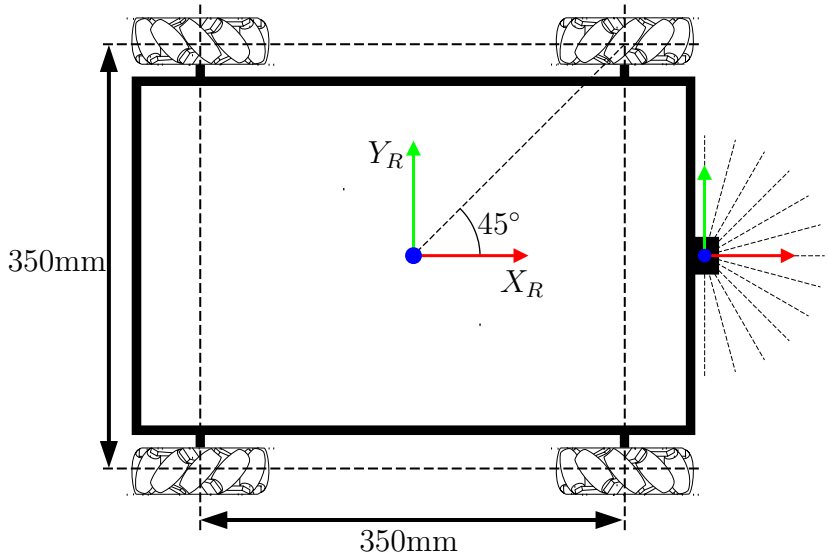


Figura 1: Esquema del robot utilizado.

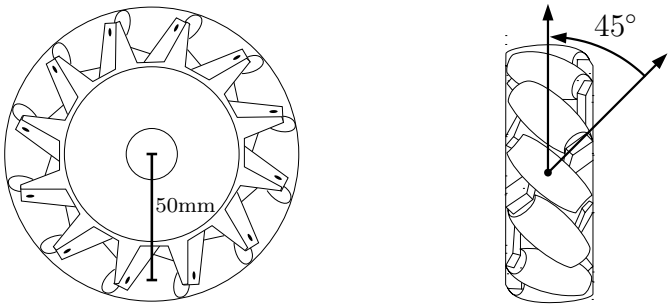


Figura 2: Rueda Mecanum de 50mm de radio.

1.1. Características del robot utilizado

El robot con el cual trabajaremos se mueve sobre el plano, es holonómico y cuenta con 4 ruedas fijas especiales y un sensor láser, dispuestos como se muestra en la Figura 1. La distancia entre los ejes de las ruedas es 350mm, el radio de cada rueda es de 50mm y la resolución de su correspondiente encoder es de 500 ticks por revolución.

Las ruedas del robot, llamadas *Mecanum* [6], poseen pequeños rodillos colocados alrededor del diámetro externo y montados a 45°, como podemos observar en la Figura 2. Los rodillos no son sensados ni actuados, interactúan de manera pasiva con el suelo pero transmiten una porción de la fuerza ejercida por el eje principal de la rueda en dirección al ángulo definido por ellos. Esta configuración genera cuatro fuerzas debido a la dirección y velocidad de rotación de cada rueda, que, al integrarse en una única fuerza total, permiten el desplazamiento en cualquier dirección en el plano. Esta característica, además de su diseño compacto y alta capacidad de carga, las hace muy atractivas para diversas aplicaciones de carácter médico, militar, industrial y educativo [1], aunque por otro lado presentan ciertos problemas prácticos debido a que tienen mayor predisposición al *patinaje* [8, 10] y mayor sensibilidad a las irregularidades del suelo [1].

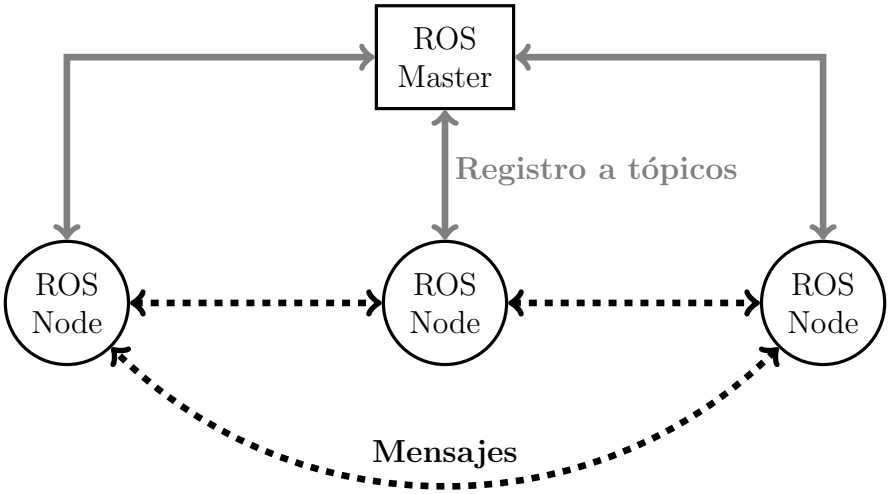


Figura 3: Comunicación de nodos en ROS.

1.2. Entorno y metodología de trabajo

Para el desarrollo de este trabajo se utilizaron distintas herramientas, descriptas a continuación.

- **ROS:** El *sistema operativo de robots* (*Robot Operating System, ROS*) es un entorno de trabajo para escribir software relacionado con robótica. Consiste en una colección de herramientas, bibliotecas y convenciones cuyo fin es simplificar la tarea de crear comportamientos de robot complejos y robustos transversales a una gran variedad de plataformas robóticas.

  Los sistemas basados en ROS están constituidos por módulos que se comunican a través del envío y recepción de mensajes (ver Figura 3). Existe un módulo especial llamado *master* y los demás módulos se llaman *nodos*. El master provee información de conexión entre nodos, de manera que puedan transmitir *mensajes* entre ellos. Los nodos deben registrarse a *tópicos* a través del master, informándole en cuáles publicarán y de cuáles leerán mensajes. Una vez hecho esto, los nodos forman una conexión directa *peer-to-peer* entre ellos. Los nodos son procesos de un único propósito (por ejemplo, drivers de sensores, procesamiento de imágenes, localización) que son compilados, ejecutados y manejados de manera individual. Los tópicos representan un canal anónimo de comunicación entre nodos y están conformados por un nombre y el tipo de mensajes que transmiten.

- **rviz:** Es una herramienta de ROS que permite la visualización conjunta de datos de sensores y muchos otros mensajes de ROS. Por ejemplo, permite ver las lecturas de un sensor láser y la pose del robot. Es también particularmente útil para comparar la trayectoria que se le pide seguir al robot con la que éste realmente sigue.

- **catkin:** Es el *build system* oficial de ROS. Extiende CMake con scripts de Python para manejar dependencias de paquetes.

- **V-REP:** Es un simulador de robots de propósito general con un entorno de desarrollo integrado, basado en una arquitectura de control distribuido: cada objeto/modelo puede ser controlado individualmente mediante un script embebido, un plugin, nodos ROS, etc. Permite modelar y simular comportamientos de sensores y mecanismos, además

de combinarlos para construir sistemas complejos. Para este trabajo contamos con una simulación del robot y los postes con el fin de poder visualizar el comportamiento del robot y realizar distintas pruebas.

En concreto, este trabajo se divide de la siguiente manera. Primero, en la Sección 2 hacemos un repaso de marcos de referencia, un concepto básico necesario para el resto del trabajo. En la Sección 3 estudiamos la relación entre la velocidad (lineal y angular) del robot y las velocidades de cada rueda, dando las ecuaciones de *cinemática inversa* y *cinemática directa*. Continuamos en la Sección 4 con la generación de trayectorias *suaves*, que luego extendemos en la Sección 5 con *control a lazo cerrado*, esto es, cómo realizar cambios a la velocidad del robot para ajustar su recorrido en base a una predicción de la pose (en principio, la provista por las ecuaciones de la Sección 3). En las secciones 6 y 7 refinaremos la estimación de la pose del robot utilizando un sensor láser y postes ubicados en posiciones fijas del mapa, y esta nueva estimación se utilizará en el control a lazo cerrado para el seguimiento de la trayectoria. Concluimos con una explicación más bien técnica del sistema desarrollado (Sección 8) y algunos comentarios finales (Sección 9). La implementación se encuentra en los archivos adjuntos a este informe.

2. Marcos de referencia

Podemos pensar al robot como un cuerpo rígido que se mueve sobre el plano (recordemos que si bien el robot con el que estamos trabajando vive en un espacio tridimensional, éste sólo se puede mover en un plano). En cada instante, el robot se encuentra en un cierto punto (x, y) y con una cierta orientación θ, lo que denominaremos *pose*. Cada pose debe estar referenciada a un sistema de coordenadas.

Un *marco de referencia* (o simplemente *marco*) i en el plano consiste en un origen O_i y un par de vectores ortonormales $\widehat{x}_i$, $\widehat{y}_i$ que conforman una base. Vamos a considerar dos marcos particulares: uno relativo al *mapa* (o *mundo*, llamado M) y otro relativo al robot (R). El marco relativo al robot tendrá su origen en el centro del robot y vectores X_R y Y_R como se muestra en la Figura 1.

En ocasiones tenemos poses expresadas en un marco y queremos conocerlas en referencia al otro. Para esto se debe proceder de la siguiente manera. Comenzaremos explicando cómo recalcular las coordenadas de un punto cuando los marcos tienen el mismo origen. Observemos la Figura 4 donde P es un punto cualquiera del plano, que vamos a suponer, sin pérdida de generalidad, que tiene con coordenadas polares (r, α) en el marco A. Luego, las coordenadas cartesianas de P en el marco A, (x_A, y_A), están dadas por las siguientes ecuaciones:

$$\begin{aligned} x_A &= r \cdot \cos(\alpha) \\ y_A &= r \cdot \sin(\alpha) \end{aligned}$$

Entonces P tiene coordenadas polares $(r, \alpha+\theta)$ en el marco B y sus coordenadas cartesianas (x_B, y_B) son tales que:

$$\begin{aligned} x_B &= r \cdot \cos(\alpha + \theta) \\ &= r \cdot (\cos(\alpha) \cdot \cos(\theta) - \sin(\alpha) \cdot \sin(\theta)) \\ &= r \cdot \cos(\alpha) \cdot \cos(\theta) - r \cdot \sin(\alpha) \cdot \sin(\theta) \\ &= x_A \cdot \cos(\theta) - y_A \cdot \sin(\theta) \end{aligned}$$

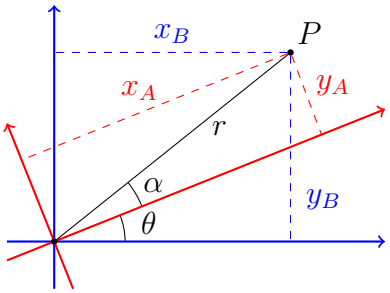


Figura 4: Un punto P en referencia a un marco A (en rojo) y a otro B (en azul), donde ambos tienen el mismo origen y forman un ángulo θ.

$$\begin{aligned} y_B &= r \cdot \cos(\alpha + \theta) \\ &= r \cdot (\cos(\alpha) \cdot \sin(\theta) + \sin(\alpha) \cdot \cos(\theta)) \\ &= r \cdot \cos(\alpha) \cdot \sin(\theta) + r \cdot \sin(\alpha) \cdot \cos(\theta) \\ &= x_A \cdot \sin(\theta) + y_A \cdot \cos(\theta) \end{aligned}$$

Ahora bien, si los marcos no poseen el mismo origen, simplemente hay que sumar la diferencia entre los orígenes. Respecto a la orientación de una pose, si ésta es φ en el marco A entonces en el marco B es $\varphi + \theta$.

3. Modelo cinemático y odometría

El primer paso será determinar qué leyes rigen el movimiento del robot. Llamemos v_x y v_y a sus velocidades lineales en x y en y respectivamente, ω_z a su velocidad angular, r al radio de las ruedas (sabemos que es el mismo en todas ellas), l_x a la mitad de la distancia entre los ejes de las ruedas frontales (y traseras), l_y a la mitad de la distancia entre los ejes de las ruedas izquierdas (y derechas), y ω_i, con $i \in \{1, 2, 3, 4\}$, a la velocidad de cada rueda.

Por un lado, dadas las dimensiones del robot, su velocidad lineal y su velocidad angular, queremos determinar la velocidad de cada rueda. Esto se conoce como *cinemática inversa*. Por otro lado, también buscamos las ecuaciones de *cinemática directa*, es decir, dadas las velocidades de las ruedas y las dimensiones del robot, queremos determinar la velocidad lineal y angular del mismo. Siguiendo las ideas de Taheri et al. [9], podemos obtener las ecuaciones de cinemática directa e inversa, respectivamente, para el robot en cuestión:

$$\begin{cases} v_x = (\omega_1 + \omega_2 + \omega_3 + \omega_4)\,\dfrac{r}{4} \\ v_y = (-\omega_1 + \omega_2 + \omega_3 - \omega_4)\,\dfrac{r}{4} \\ \omega_z = (-\omega_1 + \omega_2 - \omega_3 + \omega_4)\,\dfrac{r}{4(l_x + l_y)} \end{cases} \qquad \begin{cases} \omega_1 = \dfrac{1}{r}\,(v_x - v_y - (l_x + l_y)\omega_z) \\ \omega_2 = \dfrac{1}{r}\,(v_x + v_y + (l_x + l_y)\omega_z) \\ \omega_3 = \dfrac{1}{r}\,(v_x + v_y - (l_x + l_y)\omega_z) \\ \omega_4 = \dfrac{1}{r}\,(v_x - v_y + (l_x + l_y)\omega_z) \end{cases}$$

¿Por qué es necesario conocer estas ecuaciones? El robot sólo puede moverse haciendo que los ejes de sus ruedas giren a cierta velocidad. De esta manera, si queremos que el robot se mueva en x con una velocidad de $10m/s$, aplicando las ecuaciones de cinemática inversa podemos indicarle a qué velocidad debe girar cada rueda. Otra habilidad que tiene el robot es contar vueltas de encoder, las cuales nos sirven para calcular la velocidad *real* de giro de las ruedas (que sabemos que puede diferir de la esperada por varios motivos, como roce con el suelo u obstáculos en el camino). Entonces, dadas las velocidades reales de las ruedas,

v_x	v_y	ω	ω_1	ω_2	ω_3	ω_4	ω'_1	ω'_2	ω'_3	ω'_4	v'_x	v'_y	ω'
0.1	0	0	2	2	2	2	2.01	2.01	2.01	2.01	0.1	0	0
-0.1	0	0	-2	-2	-2	-2	-2.01	-2.01	-2.01	-2.01	-0.1	0	0
0	0.1	0	-2	2	2	-2	-2.01	2.01	2.01	-2.01	0	0.1	0
0	-0.1	0	2	-2	-2	2	2.01	-2.01	-2.01	2.01	0	-0.1	0
0	-0.25	0	5	-5	-5	5	4.78	-4.78	-4.78	4.78	0	-0.24	-0.01
0	-1	0	20	-20	-20	20	10.30	-12.57	-10.30	12.82	0	-0.57	0.03
0	0	0.5	-3.5	3.5	-3.5	3.5	-3.52	3.52	-3.52	3.52	0	0	0.49
-0.25	-0.25	0	0	-10	-10	0	0	-8.80	-8.80	0	-0.22	-0.22	0.01

Tabla 1: Comparación de las velocidades esperadas en las ruedas $(\omega_1, \omega_2, \omega_3, \omega_4)$ según las ecuaciones de cinemática inversa contra las velocidades reales ($\omega'_1, \omega'_2, \omega'_3, \omega'_4$ velocidades de las ruedas, v'_x, v'_y, ω' velocidad lineal y angular del robot), dada las velocidades lineales y angulares asignadas al robot (v_x, v_y, ω). Las unidades son m/s para las velocidades lineales y rad/s para las angulares.

Δt	v_x	v_y	ω	x	y	θ	x'	y'	θ'
15	0	-0.1	0	0.01	-1.31	0.01	-0.01	-1.28	-0.02
30	0	-0.1	0	0.04	-2.80	0.03	-0.06	-2.74	-0.05
4	0	-1	0	0.26	-2.24	0.14	-0.65	-1.72	-0.47
5	0	0	0.5	0	0	2.40	0	0	2.37
15	-0.25	-0.25	0	-2.31	-2.32	0.01	-2.29	-2.23	-0.03
10	-0.5	0	1	0.52	-0.60	-1.68	0.50	-0.51	-1.63

Tabla 2: Pose estimada (x, y, θ) contra pose real (x', y', θ'), dada la velocidad asignada al robot (v_x, v_y, ω) y durante un cierto tiempo Δt, expresadas en el marco *odom*. Las unidades son: s, m/s, rad/s, m, rad, según corresponda.

podemos aplicar las ecuaciones de cinemática directa para estimar las velocidades v_x, v_y, ω_z reales del robot. Si además calculamos $v_x\Delta t$, $v_y\Delta t$ y $\omega_z\Delta t$, obtenemos los desplazamientos en el marco del robot cada cierto tiempo Δt, que también podemos transformar a cualquier otro marco (ver Sección 2). Conociendo la pose inicial y acumulando los desplazamientos, podemos estimar la pose actual del robot (lo que se conoce como *odometría*), la cual expresaremos en un marco con origen en la pose inicial del robot (marco *odom*).

A partir de varias pruebas, confeccionamos la Tabla 1, donde, dadas diferentes consignas de velocidades lineales y angulares del robot, se muestran las velocidades que cada rueda debería tener según las ecuaciones de cinemática inversa en comparación con las velocidades *reales* de las ruedas del robot. Como podemos notar, hay una pequeña diferencia, y esto se debe a diferentes factores, como el roce con el suelo y sobre todo el patinaje de las ruedas. Notemos que, justamente, velocidades menores producen menos error.

Por otro lado, la Tabla 2 muestra la pose calculada con el método explicado anteriormente contra la pose real, dada la velocidad lineal y angular del robot y el tiempo transcurrido, suponiendo que comienza en $(0, 0, 0)$. Observemos que, si bien son similares, no coinciden exactamente. La diferencia entre ambas crece con el tiempo, ya que los errores de estimación se van acumulando. Tengamos en cuenta también que, al asignar una velocidad, el robot no puede ejercerla inmediatamente y le tomará un tiempo acelerar hasta alcanzarla, pero por simplicidad de los cálculos estamos desestimando esto.

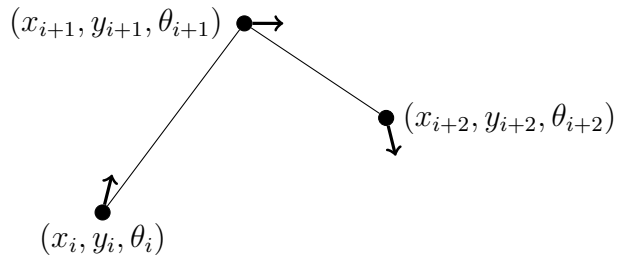


Figura 5: Interpolación lineal entre tres waypoints.

4. Generación de trayectorias

Un *camino* es una lista ordenada de poses por las que queremos que pase el robot, mientras que una *trayectoria* es un camino con restricciones de tiempo para cada pose. Cada elemento de estas listas se llama *waypoint*.

Con el objetivo de darle más autonomía al robot, ahora nos gustaría que sea capaz de seguir trayectorias. A partir de puntos en el plano, cada uno con una orientación y un tiempo en el cual el robot debe estar allí, queremos calcular la velocidad lineal y angular del robot en cada instante. En particular, debemos determinar cómo se mueve el robot *entre* los waypoints.

Consideremos una trayectoria de $n + 1$ waypoints, nombrados de 0 a n. Notaremos (x_i, y_i, θ_i) y t_i, respectivamente, a la pose y al tiempo del waypoint i, con $i \in \{0, \ldots, n\}$. Dado que estamos trabajando con un robot omnidireccional, podemos estudiar cada dimensión (x, y, θ) por separado. En lo que sigue sólo explicaremos una solución para x, pero la misma debe aplicarse también tanto a y como a θ (realizando los reemplazos correspondientes).

Necesitamos describir una curva que una dos waypoints consecutivos de la trayectoria. Una solución sencilla es suponer un recorrido rectilíneo y considerar una velocidad constante allí (interpolación lineal, como se aprecia en la Figura 5). El problema con esta idea es que en cada waypoint la velocidad puede necesitar cambiar bruscamente, situación que en una aplicación real no puede ocurrir. Por lo tanto, sería deseable un recorrido *suave* en este sentido, es decir, que la velocidad sea continua. Teniendo esto en cuenta, vamos a proponer el siguiente polinomio $x(t)$, a partir de cual (derivando) podemos calcular la velocidad $v(t)$:

$$\begin{aligned} x(t) &= a_0 + a_1(t - t_i) + a_2(t - t_i)^2 + a_3(t - t_i)^3 \\ v(t) &= a_1 + 2a_2(t - t_i) + 3a_3(t - t_i)^2 \end{aligned}$$

Para calcular a_0, a_1, a_2 y a_3 vamos a resolver el siguiente sistema:

$$\begin{aligned} x_i &= a_0 + a_1(t_i - t_i) + a_2(t_i - t_i)^2 + a_3(t_i - t_i)^3 \\ x_{i+1} &= a_0 + a_1(t_{i+1} - t_i) + a_2(t_{i+1} - t_i)^2 + a_3(t_{i+1} - t_i)^3 \\ v_i &= a_1 + 2a_2(t_i - t_i) + 3a_3(t_i - t_i)^2 \\ v_{i+1} &= a_1 + 2a_2(t_{i+1} - t_i) + 3a_3(t_{i+1} - t_i)^2 \end{aligned}$$

Simplificando obtenemos inmediatamente $a_0 = x_i$ y $a_1 = v_i$. Reescribimos entonces el sistema con estos datos:

$$\begin{aligned} a_2(t_{i+1} - t_i)^2 + a_3(t_{i+1} - t_i)^3 &= x_{i+1} - x_i - v_i(t_{i+1} - t_i) \\ 2a_2(t_{i+1} - t_i) + 3a_3(t_{i+1} - t_i)^2 &= v_{i+1} - v_i \end{aligned}$$

Restando $\frac{t_{i+1} - t_i}{3}$ veces la segunda ecuación a la primera y simplificando, obtenemos a_2 y

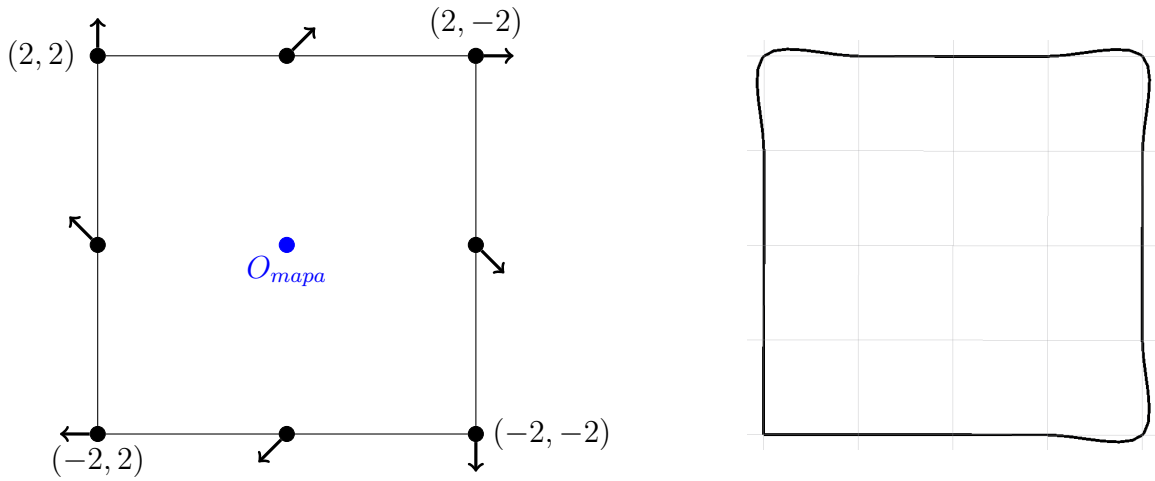


Figura 6: Trayectoria ideal requerida (a) vs. trayectoria suave generada a partir de tomar 17 waypoints de la trayectoria ideal (b). Se marcan algunos waypoints para ejemplificar la orientación.

partir de éste podemos calcular también a_3. En síntesis:

$$
\begin{aligned}
a_0 &= x_i \\
a_1 &= v_i \\
a_2 &= \frac{3(x_{i+1} - x_i)}{(t_{i+1} - t_i)^2} - \frac{2v_i + v_{i+1}}{t_{i+1} - t_i} \\
a_3 &= \frac{-2(x_{i+1} - x_i)}{(t_{i+1} - t_i)^3} + \frac{v_i + v_{i+1}}{(t_{i+1} - t_i)^2}
\end{aligned}
$$

A partir de estas ecuaciones surge una nueva pregunta: ¿cuál es la velocidad instantánea v_i en cada waypoint i? Una forma razonable y sencilla de definirla es la siguiente: al principio y al final es 0 porque podemos suponer que el robot empieza y finaliza su recorrido en reposo, y en un waypoint i, con $i \in \{1, \ldots, n-1\}$, es la velocidad promedio a la que debe ir el robot entre el waypoint anterior y el siguiente, esto es, $v_i = \frac{x_{i+1} - x_{i-1}}{t_{i+1} - t_{i-1}}$. Notemos que ahora sí, finalmente, contamos con todos los elementos para calcular la velocidad lineal y angular del robot en cada instante.

En concreto, para este trabajo se solicitaba una trayectoria cuadrada de 4 metros de lado centrada en el origen de coordenadas del mapa, de tal forma que el robot mantuviese siempre una orientación "opuesta al centro", como se ve en la Figura 6a. En la Figura 6b se muestra la curva generada, usando el método del polinomio antes descripto, a partir de los waypoints creados.

5. Control a lazo cerrado

Ya contamos con las herramientas suficientes para indicarle al robot los comandos de velocidad para poder seguir una determinada trayectoria. Sin embargo, al poner el robot en marcha notamos que no sigue exactamente dicha trayectoria. Esto se debe a que el robot no siempre se encuentra donde debería, como mencionamos en la Sección 3. Para solucionar este problema, en esta sección veremos cómo lograr que el robot ajuste su velocidad en base a una predicción de su pose cuando recorre la trayectoria.

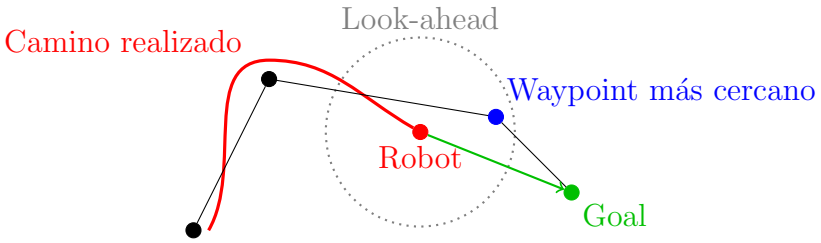


Figura 7: Definición del siguiente goal en el algoritmo pure pursuit.

5.1. Seguimiento *Pure Pursuit*

Si hiciéramos que el robot simplemente corrija su posición para coincidir con el waypoint entonces éste oscilaría constantemente, porque nunca lograría esta posición exactamente. En su lugar, vamos asumir que el robot no siempre estará en la posición correcta pero que podemos lograr que converja a "la siguiente". Entonces vamos a ver ahora cómo elegir la siguiente posición a la cual el robot debe moverse en función de la posición actual estimada.

El algoritmo de "siga la zanahoria" (*pure pursuit*) fue diseñado originalmente como un método para calcular el arco necesario para que un robot vuelva a su ruta [2]. Este algoritmo calcula los comandos de velocidad necesarios para mover al robot desde su pose actual a un cierto objetivo (*goal*) que se encuentre más adelante en la trayectoria y a una cierta distancia (*look-ahead*), iterando hasta encontrar el punto final del camino.

Recordemos los pasos del algoritmo vistos en clase:

- Determinar la pose actual del robot.
- Si el robot está lo suficientemente cerca del waypoint final:
  - Terminar.
- Si no:
  - Encontrar el waypoint más cercano al robot (recorriendo todos los waypoints y midiendo la distancia euclídea).
  - A partir de este punto, definir el goal como el siguiente waypoint que se encuentre a al menos la distancia look-ahead definida.
- Transformar el goal a las coordenadas del robot.
- Realizar el control a lazo cerrado hacia el goal.
- Calcular la nueva pose del robot.

5.2. Controlador proporcional

En cada instante hay una diferencia (*error*) entre la pose actual (o estimada) del robot (x, y, θ) y la pose objetivo (goal), que podemos notar $e = (d_x, d_y, d_\theta)$, expresada en el marco

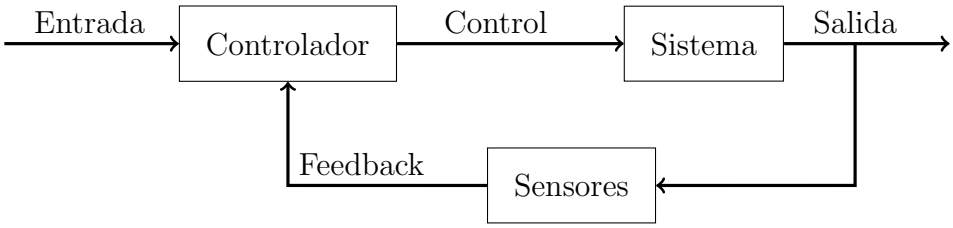


Figura 8: Esquema de control a lazo cerrado.

del robot. Vamos a utilizar un *controlador proporcional* K, es decir, la salida del controlador es proporcional al error e:

$$u = \begin{pmatrix} v_x \\ v_y \\ \omega \end{pmatrix} = K \cdot e = \begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} \\ K_{2,1} & K_{2,2} & K_{2,3} \\ K_{3,1} & K_{3,2} & K_{3,3} \end{pmatrix} \cdot \begin{pmatrix} d_x \\ d_y \\ d_\theta \end{pmatrix}$$

Dado que estamos trabajando con un robot omnidireccional, podemos suponer que la matriz K es diagonal. Entonces:

$$\begin{aligned} v_x &= K_{1,1} d_x \\ v_y &= K_{2,2} d_y \\ \omega &= K_{3,3} d_\theta \end{aligned}$$

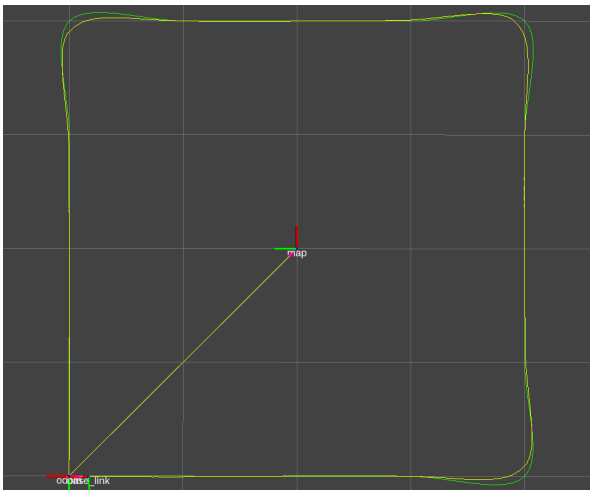
Como queremos acercarnos al goal, requerimos $K_{i,i} > 0$ para todo $i \in \{1, 2, 3\}$.

5.3. Elección de las constantes involucradas

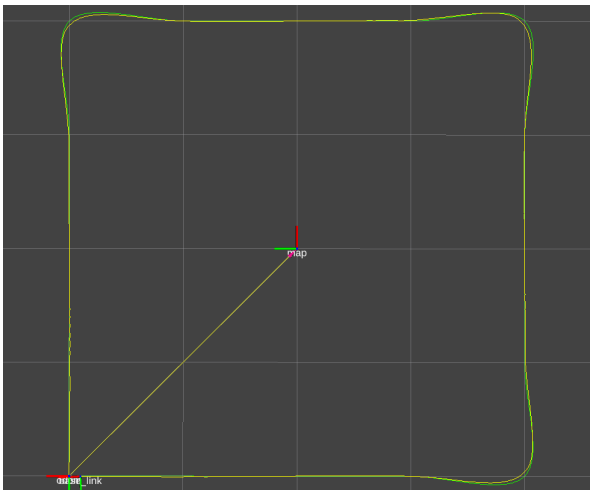
Para la elección del look-ahead, sabemos que una distancia corta haría oscilar más al robot, mientras que una mayor sería más suave pero esquivaría más waypoints. Se graficaron las trayectorias seguidas por la estimación dada por odometría (para tener una idea de cuál sería el movimiento ideal del robot) con valores de look-ahead 0,05, 0,1 y 0,2 (Figura 9), eligiendo K en cada caso de modo que las velocidades del robot sean similares. A partir de esto podemos fijar entonces a 0,1 como distancia look-ahead, ya que genera una buena curva sin tanta oscilación.

Para fijar los valores de K, en principio probamos con los mismos valores para cada $K_{i,i}$, luego veremos si es necesario considerarlos diferentes. Comenzaremos estudiando qué valores dan una buena convergencia de la estimación dada por odometría. Sabemos que mayores valores de las constantes implican una convergencia más rápida, pero sólo hasta cierto punto ya que también implican una mayor velocidad pero los actuadores del robot tienen un límite. Por lo tanto, probamos diferentes valores hasta notar que se volvía inestable (el camino "tiembla" un poco más, ver Figura 10, y también en los comandos de velocidad se pueden observar mayores cambios entre unos y otros). Concluimos que nuestro mejor valor es $K_{i,i} = 2$. Por otro lado, no se observó ninguna mejora al cambiar solamente uno de estos valores.

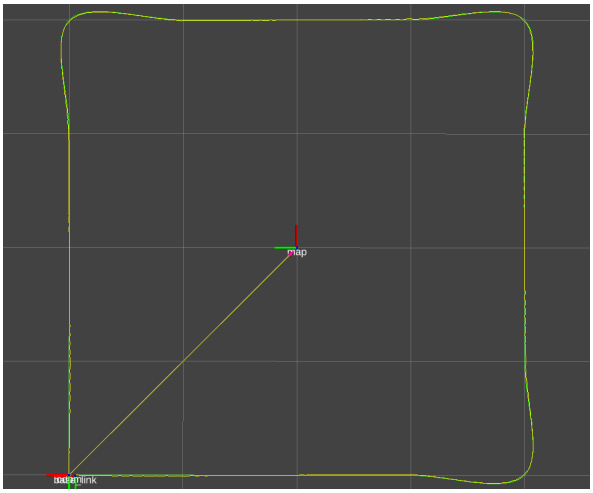
Como vimos en la Sección 3, a mayor velocidad mayor será el error de la estimación odométrica, y además este error se acumula en el tiempo. De poco sirve un controlador que tenga una buena convergencia teórica si luego el robot sigue una trayectoria completamente distinta. Esto es realmente un problema para nuestro caso. Por lo tanto, para obtener un robot que siga correctamente la trayectoria, deberíamos bajar las constantes. El inconveniente aquí es que se vuelve mucho más lento y además nunca logra ajustarse muy bien la trayectoria real. Hay que encontrar entonces un punto intermedio. Teniendo esto en cuenta, se decidió fijar $K_{i,i} = 1$ (ver Figura 11).



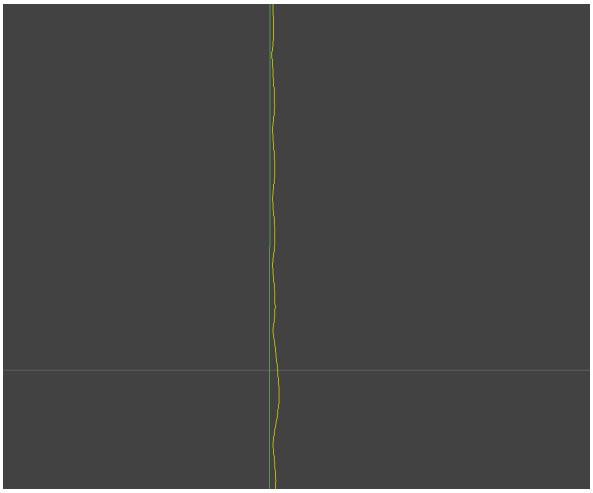
(a) Look-ahead = 0,2, $K_{i,i} = 1$.



(b) Look-ahead = 0,1, $K_{i,i} = 2$.

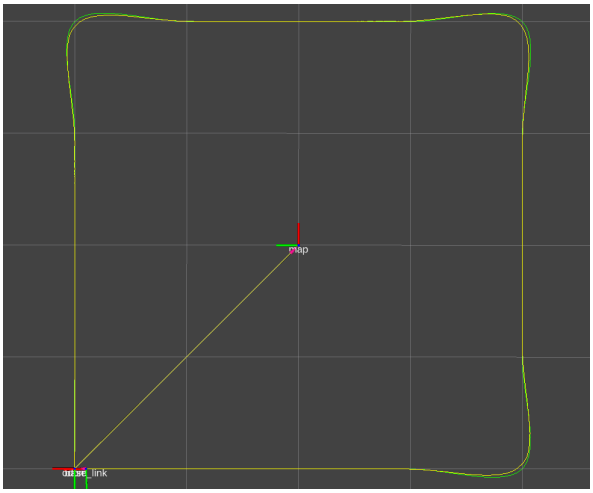


(c) Look-ahead = 0,05, $K_{i,i} = 4$.

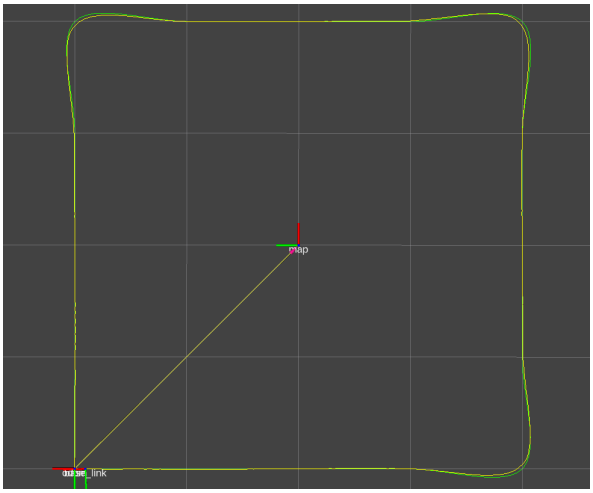


(d) Look-ahead = 0,05, $K_{i,i} = 4$. Acercamiento.

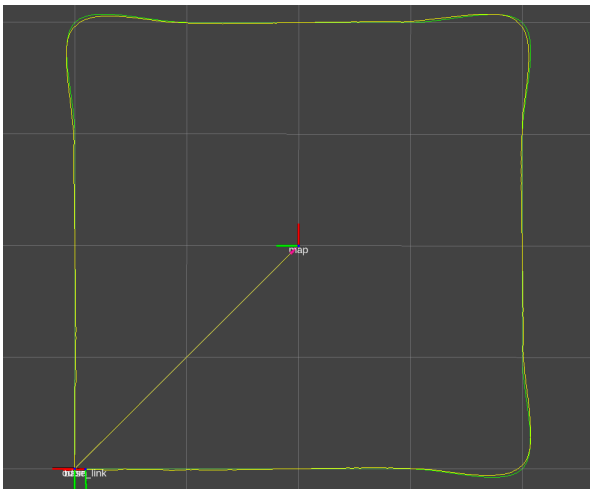
Figura 9: Diferentes valores de look-ahead. Se hizo un acercamiento para $0,05$ para observar mejor la "ondulación" en la trayectoria.



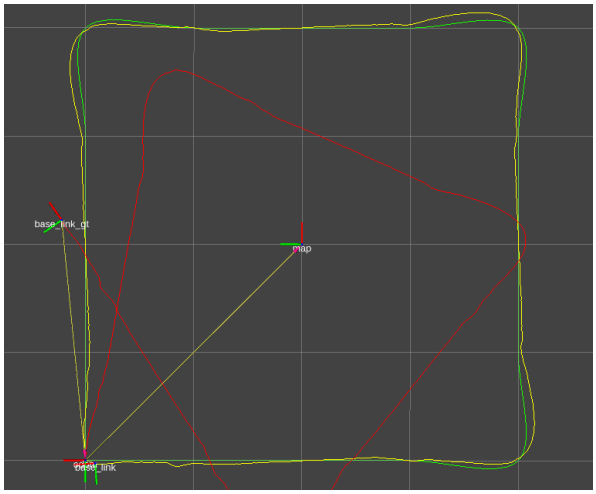
(a) $K_{i,i} = 1$



(b) $K_{i,i} = 2$

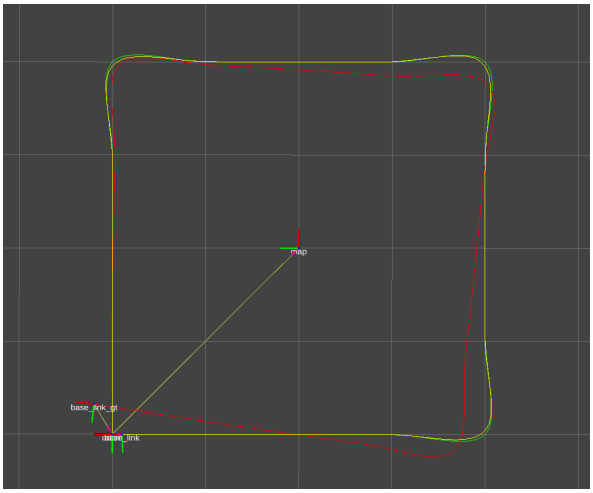


(c) $K_{i,i} = 3$

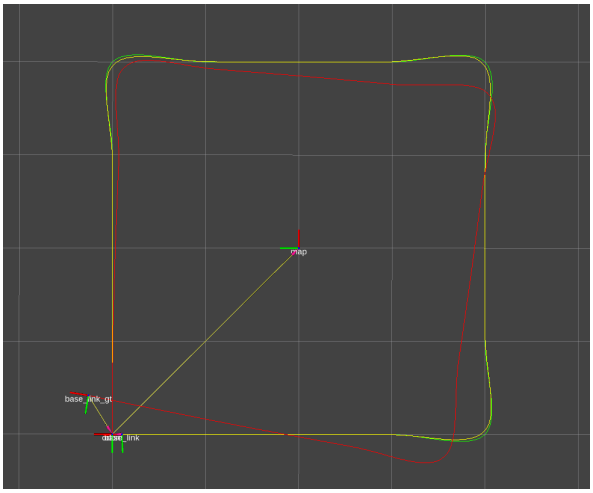


(d) $K_{i,i} = 10$

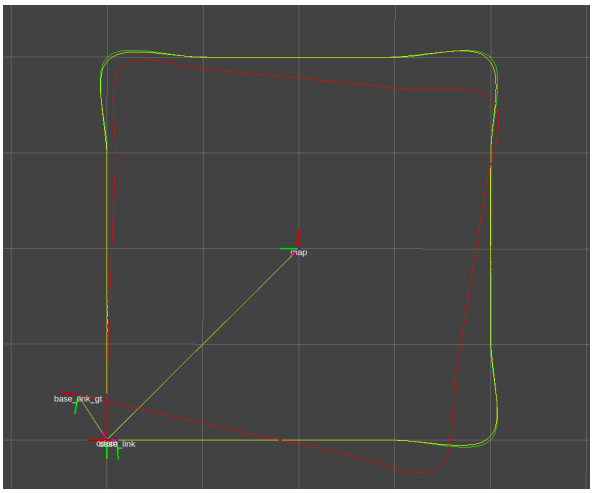
Figura 10: Análisis de distintos valores de las constantes de control. En verde, el camino a seguir. En amarillo, el camino dado por la estimación de la pose según odometría. Se incluye la última imagen sólo para ilustrar más claramente el efecto de la inestabilidad.



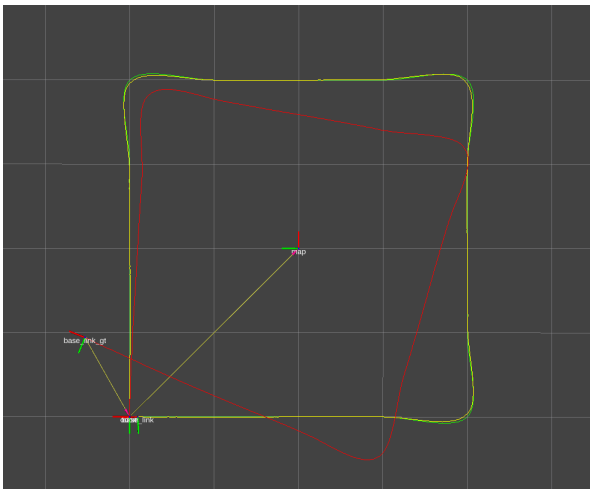
(a) $K_{i,i} = 0{,}25$



(b) $K_{i,i} = 0{,}5$



(c) $K_{i,i} = 1$



(d) $K_{i,i} = 2$

Figura 11: Análisis de distintos valores de las constantes de control. En verde, el camino a seguir. En amarillo, el camino dado por la estimación de la pose según odometría. En rojo, el realizado en realidad.

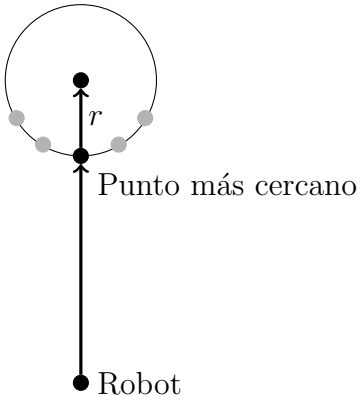


Figura 12: Determinación del centro de un poste de radio r mediante las mediciones del láser.

6. Localización basada en EKF

Dado que la predicción odométrica de la pose no es muy precisa, en esta sección estudiaremos cómo integrarla con la información provista por el láser del robot y los postes del mapa para dar una mejor aproximación de la pose.

6.1. Estimación de la ubicación según la información del láser

Recordemos que el robot cuenta con un láser. Las mediciones del mismo consisten en pares (ρ, ϕ) con las coordenadas polares en el marco del láser de los puntos detectados, cada una de las cuales podemos transformar fácilmente a coordenadas polares en el marco del robot (ya que contamos con ${}^{R}T_{L}$).

Estos puntos detectados se pueden agrupar según el poste al que pertenecen. No obstante, a priori no sabemos a qué poste pertenece cada punto. Lo que sí sabemos es que los puntos forman *clusters*, es decir, grupos de puntos que se encuentran mucho más cerca entre ellos que de los demás. Podemos decir que cada cluster representa un poste detectado. Para saber cuál es cada uno de estos postes, primero debemos encontrar sus centros y luego compararlos con los centros conocidos de los postes dados por el mapa.

Existen ecuaciones para determinar de manera exacta el centro de un círculo dado un conjunto de al menos tres puntos en él. Dado que son un tanto complicadas y que en nuestro proyecto no requerimos extrema precisión (como sí la podríamos requerir si, por ejemplo, estuviésemos desarrollando un robot con aplicaciones médicas, o que manipule objetos muy pequeños), por simplicidad vamos a utilizar otro método para calcular los centros de los postes. Vamos a suponer que contamos con suficiente información del láser como para haber detectado el punto de la circunferencia del poste que se encuentra más cerca del robot (sabemos que esto no siempre ocurre, pero siempre se encuentra un punto lo suficientemente cercano a este). De esta manera, el centro se obtiene simplemente extendiendo el vector que une el centro del robot con este punto tantas unidades como el radio del poste, como podemos observar en la Figura 12.

Para determinar cuál es el poste real del mapa que se corresponde con el centro calculado, podemos simplemente buscar entre todos ellos cuál es el más cercano. Por supuesto, aquí estamos suponiendo que conocemos la pose inicial del robot y que además éste se mueve *suvamente*, es decir, que nunca sufrió ningún cambio de pose demasiado brusco (esto podría ocurrir si, por ejemplo, una persona lo mueve intencionalmente).

Conocer cuáles deberían ser las coordenadas reales de un poste y a qué distancia se

encuentra el robot (esto viene directamente de la medición del láser) nos permite estimar muy fácilmente la pose real del robot.

6.2. Filtro de Kalman Extendido (EKF)

Gracias al láser obtenemos una secuencia de mediciones en el tiempo con las que podemos estimar la pose del robot. Utilizaremos el Filtro de Kalman Extendido (*Extended Kalman Filter, EKF*) para derivar una estimación de la pose del robot a partir de estas observaciones. Además, también vamos a analizar cómo se propaga la incertidumbre de la pose a medida que avanza. Vamos a utilizar el siguiente modelo:

- Estado: $\vec{x} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$. Es la pose estimada del robot en el tiempo, en referencia al mapa.
- Entradas de control: $\vec{u} = \begin{pmatrix} v_x \\ v_y \\ \omega \end{pmatrix}$. Son las velocidades dadas por odometría.
- Mediciones: $\vec{z} = \begin{pmatrix} \rho \\ \phi \end{pmatrix}$. Son coordenadas polares en el marco del robot.
- Ruido del actuador: $\vec{w} = \begin{pmatrix} w_x \\ w_y \\ w_\theta \end{pmatrix}$, donde $w \sim \mathcal{N}(0, Q)$.
- Ruido del sensor: $\vec{v} = \begin{pmatrix} v_\rho \\ v_\phi \end{pmatrix}$, donde $v \sim \mathcal{N}(0, R)$.
- Modelo de movimiento: $f(\vec{x}, \vec{u}, \vec{w}) = \begin{pmatrix} x + v \cdot \Delta t \cdot \cos(\theta + \alpha) + w_x \\ y + v \cdot \Delta t \cdot \sin(\theta + \alpha) + w_y \\ \theta + \omega \cdot \Delta t + w_\theta \end{pmatrix}$ donde $v = \sqrt{v_x^2 + v_y^2}$ y $\alpha = \text{atan2}(v_y, v_x)$. Es el nuevo estado del robot en base al anterior. Se calcula simplemente transformando el desplazamiento al marco del mapa.
- Modelo de sensado: $h(\vec{x}, \vec{v}) = \begin{pmatrix} \sqrt{(p_x - x)^2 + (p_y - y)^2} + v_\rho \\ \text{atan2}(p_y - y, p_x - x) - \theta + v_\phi \end{pmatrix}$. Son las coordenadas polares, en el marco del robot, de un poste cuyas coordenadas cartesianas, en el marco del mapa, son (p_x, p_y).
- Jacobianos: se obtienen derivando las matrices anteriores.

$$A = \frac{\partial f(\vec{x}, \vec{u}, \vec{w})}{\partial \vec{x}} = \begin{pmatrix} 1 & 0 & -\sin(\theta + \alpha) \cdot v \cdot \Delta t \\ 0 & 1 & \cos(\theta + \alpha) \cdot v \cdot \Delta t \\ 0 & 0 & 1 \end{pmatrix}$$

$$W = \frac{\partial f(\vec{x}, \vec{u}, \vec{w})}{\partial \vec{w}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$H = \frac{\partial h(\vec{x}, \vec{v})}{\partial \vec{x}} = \begin{pmatrix} -\frac{p_x - x}{\sqrt{(p_x-x)^2+(p_y-y)^2}} & -\frac{p_y - y}{\sqrt{(p_x-x)^2+(p_y-y)^2}} & 0 \\ \frac{p_y - y}{(p_x-x)^2+(p_y-y)^2} & -\frac{p_x - x}{(p_x-x)^2+(p_y-y)^2} & -1 \end{pmatrix}$$

$$V = \frac{\partial h(\vec{x}, \vec{v})}{\partial \vec{v}} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

- Matrices de covarianza:

  - Covarianza de $\vec{v}$: $R = \begin{pmatrix} \sigma_\rho^2 & 0 \\ 0 & \sigma_\phi^2 \end{pmatrix} = \begin{pmatrix} 0{,}001 & 0 \\ 0 & 0{,}000076 \end{pmatrix}$

    La matriz de covarianza R se determina a partir de las desviaciones estándar dadas por la precisión de los instrumentos de medición, en este caso el láser. Así, $\sigma_\rho = 0{,}1$ (en metros) y $\sigma_\phi = 0{,}5 \cdot \frac{\pi}{180}$ (desvío de 0,5 grados). Esto no se aclara en el enunciado, pero vamos a asumir que es el mismo tipo de sensor que el visto en uno de los talleres.

  - Covarianza de $\vec{w}$: $Q = \begin{pmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{pmatrix} = \begin{pmatrix} 0{,}04 & 0 & 0 \\ 0 & 0{,}04 & 0 \\ 0 & 0 & 0{,}04 \end{pmatrix}$

    Para determinar Q, primero asumimos que x, y, θ son independientes (por esto los valores 0 en la matriz) y luego realizamos el siguiente experimento: asignamos una cierta velocidad al robot durante un cierto tiempo y observamos su pose, repetimos varias veces esto y luego tomamos la desviación estándar de los valores obtenidos en x, y y θ. Obtuvimos los siguientes valores: $\sigma_x = 0{,}023$, $\sigma_y = 0{,}032$, $\sigma_\theta = 0{,}045$. Estos valores son un tanto pequeños ya que la simulación no genera demasiado ruido. Por lo tanto se decidió tomar mayores valores para reflejar una mayor incertidumbre que podría darse en una aplicación real. Entonces fijamos $\sigma_x = \sigma_y = \sigma_\theta = 0{,}2$ (esto es, entre cinco y diez veces más que la simulación).

  - Covarianza inicial: $P = \begin{pmatrix} 0{,}1 & 0 & 0 \\ 0 & 0{,}1 & 0 \\ 0 & 0 & 0{,}1 \end{pmatrix}$

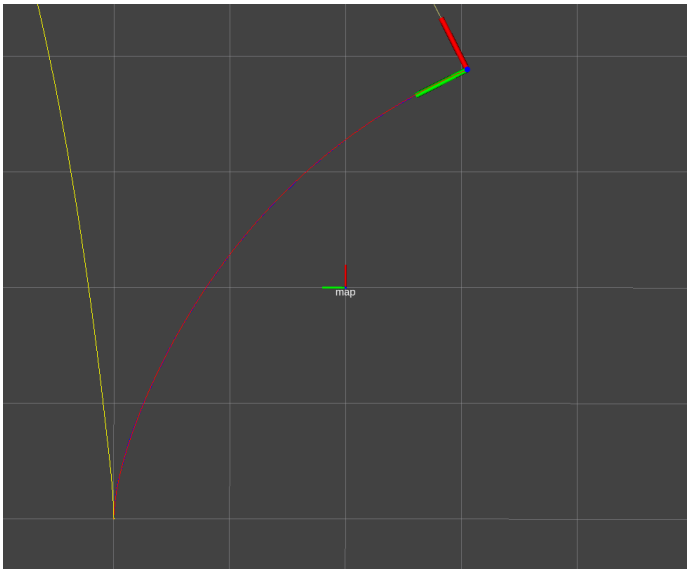
    Aquí también consideramos valores más bien altos, pero ya con el objetivo de poder apreciar cómo mejora la certeza de la pose con la aplicación del método.

6.3. Análisis de los resultados

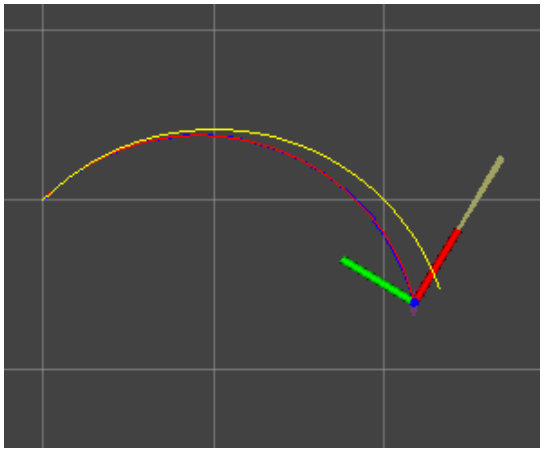
Movimos el robot de diferentes maneras, como se muestra en la Figura 13. Se puede ver que la estimación de la pose dada con este nuevo método prácticamente coincide con la pose real cuando detecta postes, incluso en los casos donde la estimación por odometría falla gravemente. En el caso (c) se giró el robot hasta perder de vista los postes y luego se continuó moviéndolo. Vemos que esta estimación es similar a la odométrica hasta que detecta nuevos postes y entonces automáticamente ajusta su estimación más cerca de la pose real.

En la Figura 14 podemos apreciar la elipse de covarianza a lo largo de una trayectoria, observándose un claro aumento cuando se dejan de percibir los postes y luego una disminución notable al encontrar al menos dos de ellos. En particular, en la Figura 14d vemos cómo aumenta la incertidumbre (ya que cuenta con un solo poste) y se reduce en la Figura 14e al encontrar otro poste (aunque pierda de vista el anterior, tiene más información dada la estimación anterior). Esta misma situación ocurre también en la Figura 15. Podemos concluir que se confía mucho más en la información dada por el láser que en la dada por odometría. De hecho, cuando la elipse crece se puede notar que también aumenta la diferencia con la pose real (ver Figura 14f).

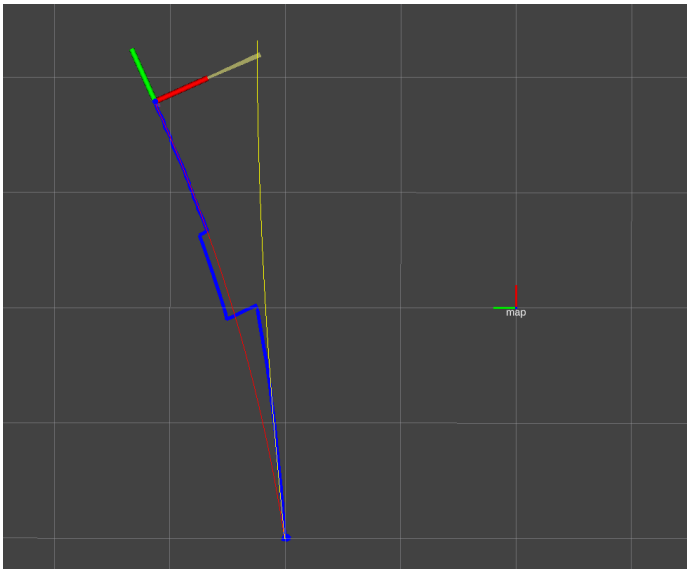
La elipse crece tan rápidamente debido a que las matrices de covarianza elegidas tienen valores altos (justamente para mostrar cómo mejora la certeza con el láser). En conclusión, la nueva estimación de la pose no sólo es casi idéntica a la pose real sino que además nos brinda mayor certeza cuando observamos que la elipse es pequeña (caso que ocurre cuando se detectan al menos dos postes).



(a)

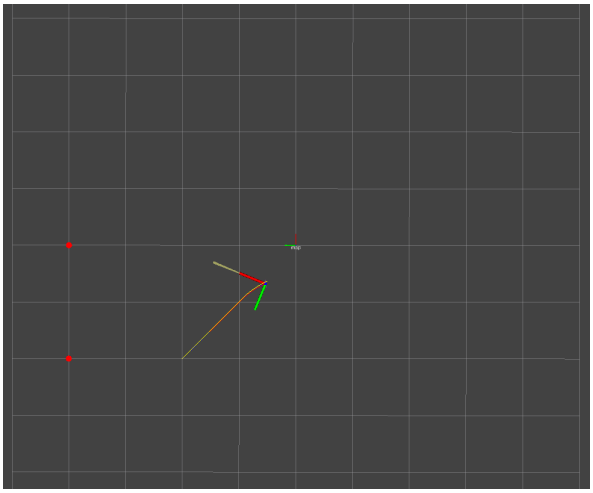


(b)

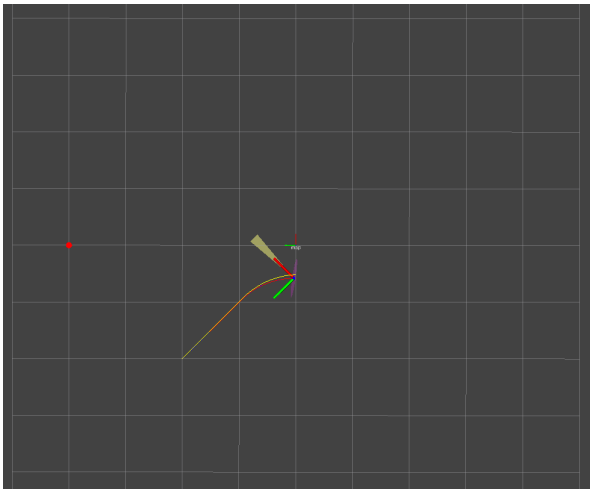


(c)

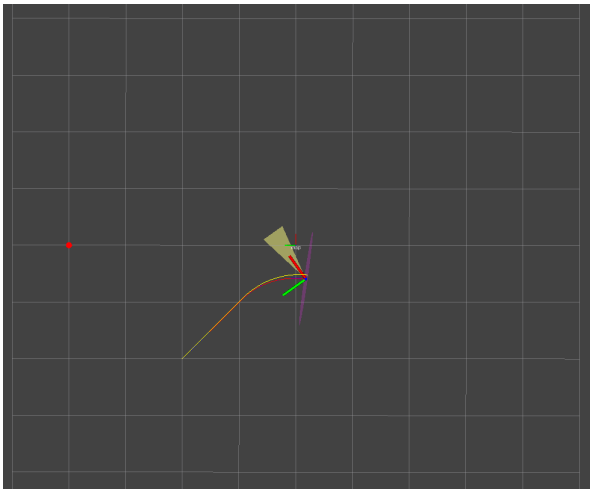
Figura 13: Comparación de la pose real (rojo) y las estimaciones dadas por odometría (amarillo) y EKF (azul).



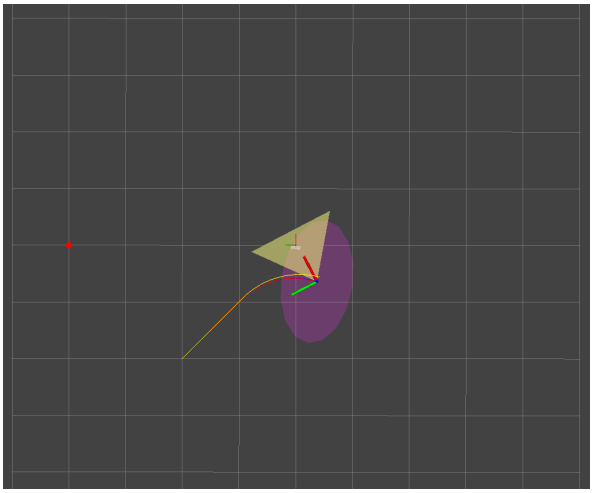
(a)



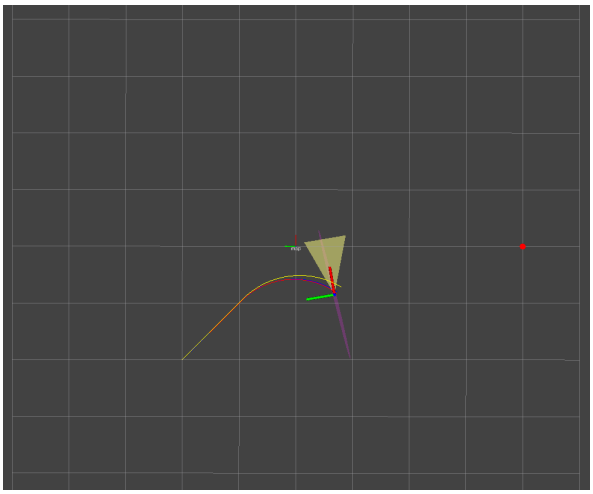
(b)



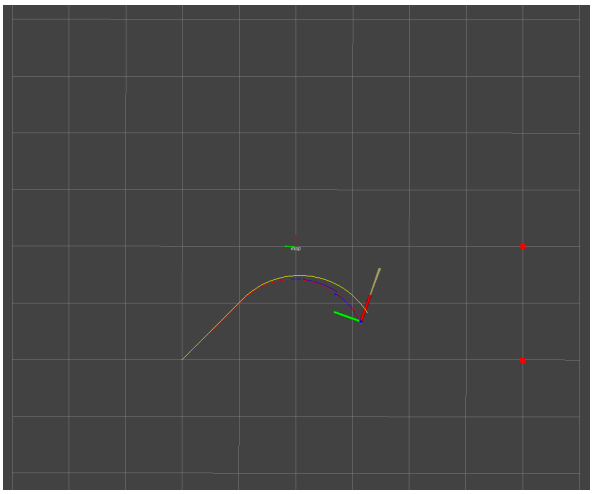
(c)



(d)

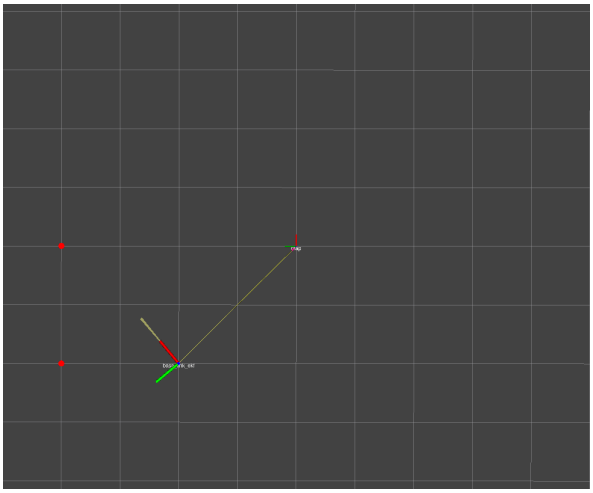


(e)

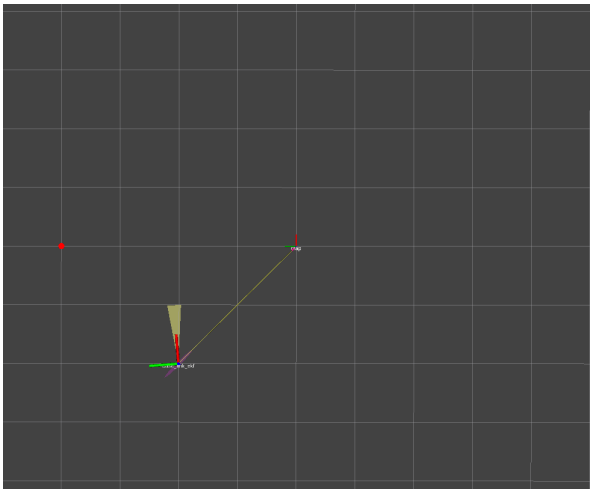


(f)

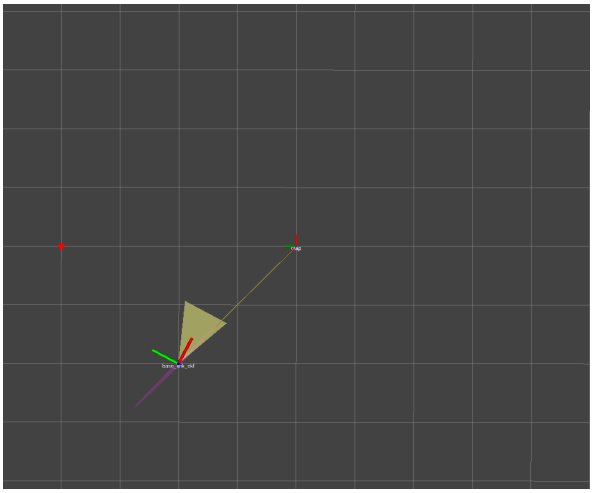
Figura 14: Elipse de covarianza durante una trayectoria en la que se pierden de vista los postes.



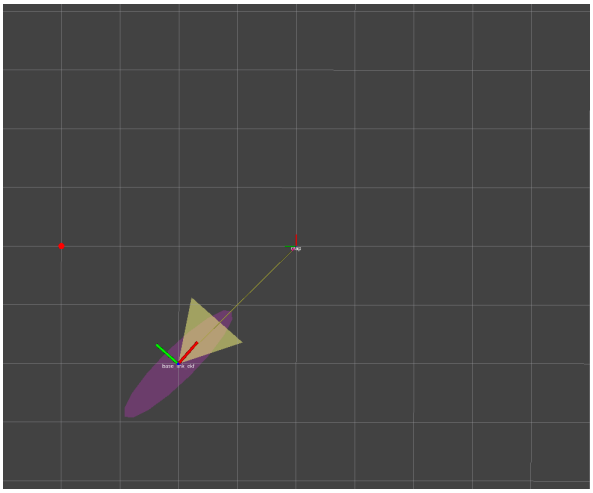
(a)



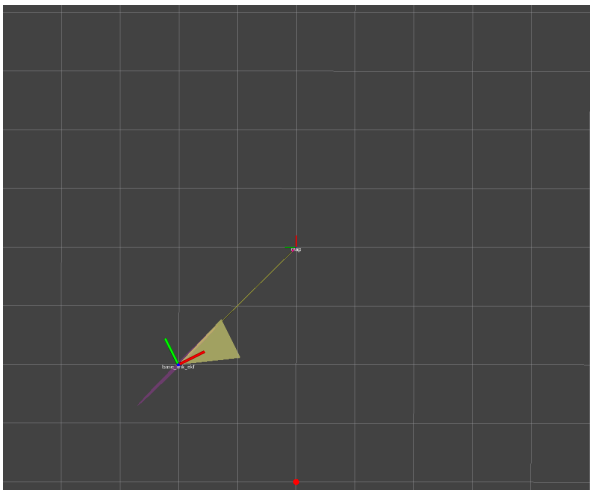
(b)



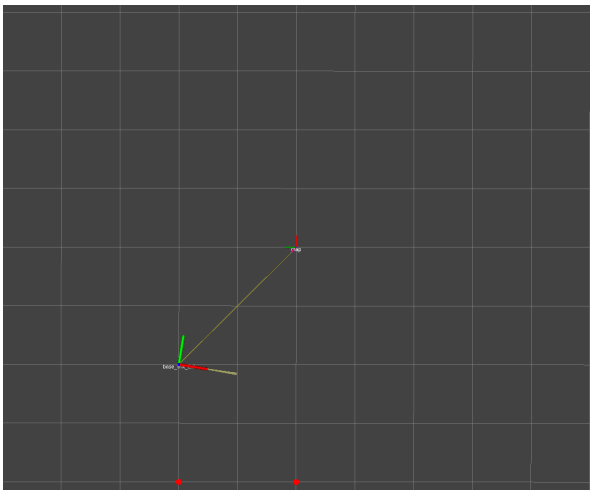
(c)



(d)

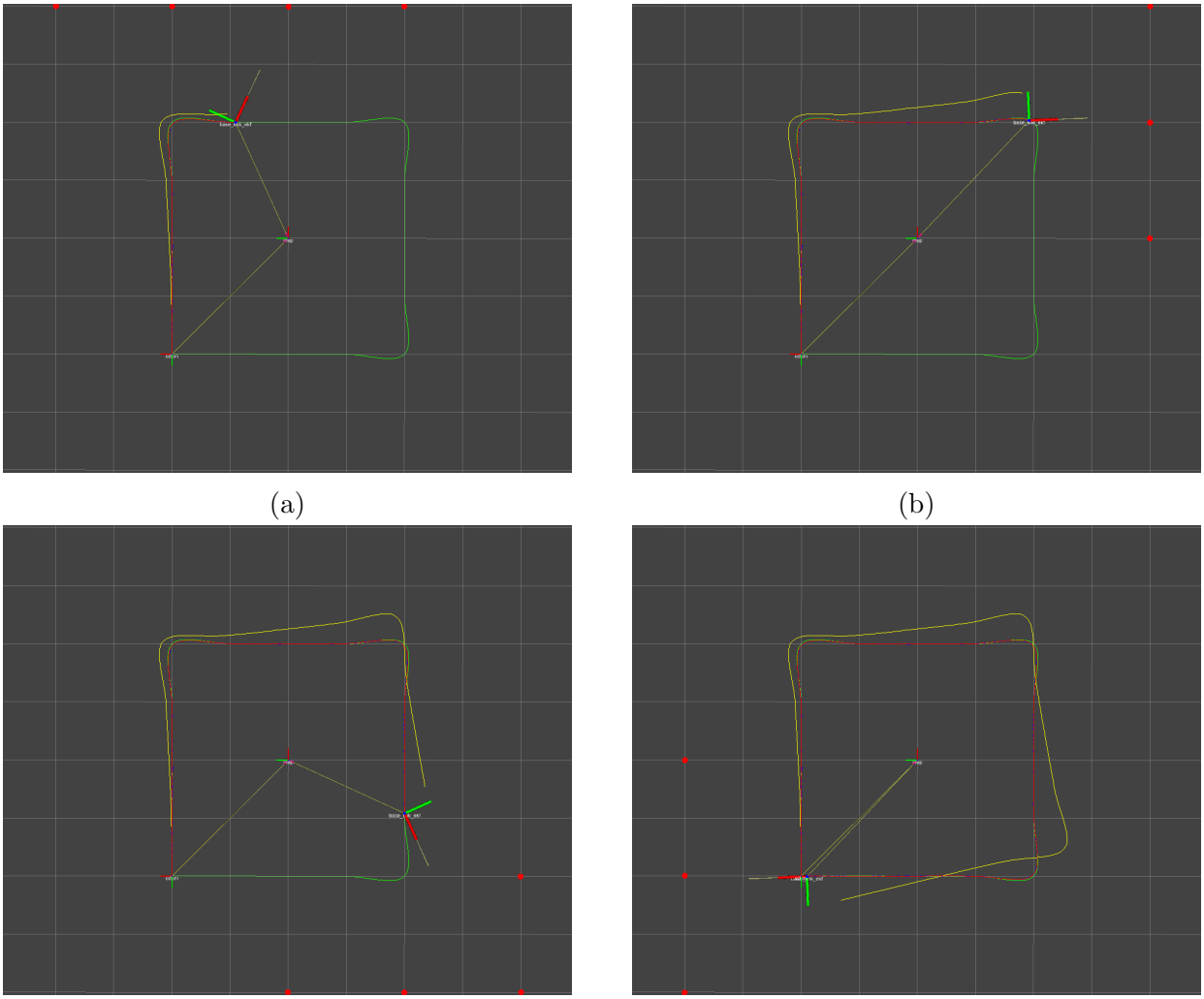


(e)



(f)

Figura 15: Elipse de covarianza durante una trayectoria en la que se pierden de vista los postes.



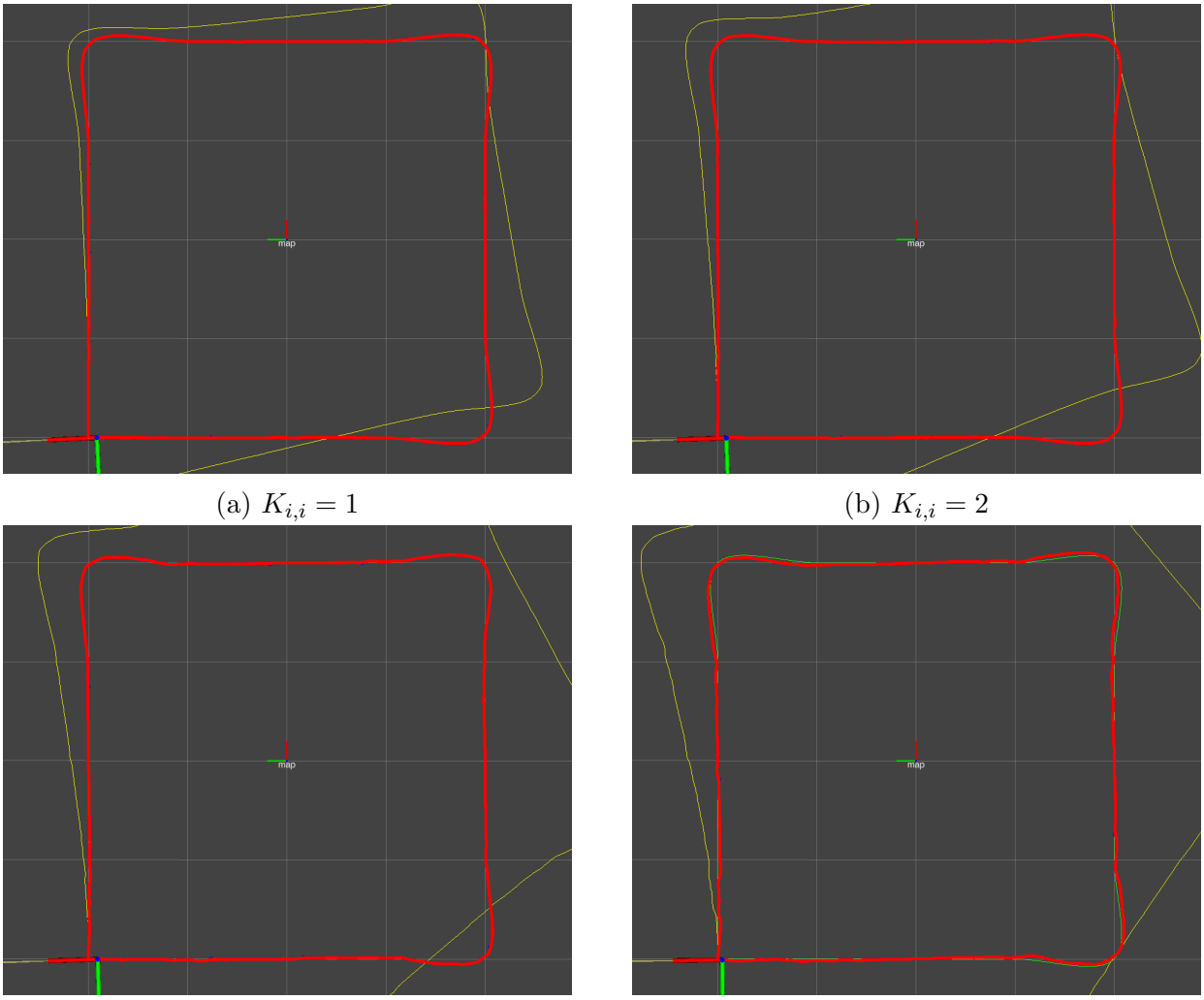
(a) (b) (c) (d)

Figura 16: Comparación del camino a seguir (verde) con los caminos descriptos por: la posición real (rojo), la estimación de la pose según EKF (azul), la estimación de la pose según odometría (amarillo). Los círculos rojos corresponden a los postes detectados.

7. Seguimiento de trayectorias a lazo cerrado utilizando localización basada en EKF

Para concluir este proyecto, veremos qué ocurre al utilizar la predicción basada en EKF como feedback del control a lazo cerrado para el seguimiento de trayectorias (en lugar de la provista por odometría como usamos en la Sección 5). Nuestra primera impresión es que recorre casi perfectamente la trayectoria (Figura 16). Esto se debe a que la predicción de la pose es muy buena ya que todo el tiempo puede detectar al menos tres postes (notemos que la elipse de covarianza es prácticamente nula).

Dado que ahora la estimación de la pose es considerablemente mejor, podemos proponer valores mayores de las constantes del controlador proporcional (para evitar los problemas explicados en la Sección 5.3). Realizamos nuevamente los experimentos para ajustar las constantes de control. Como se puede apreciar en la Figura 17, $K_{i,i} = 1$ y $K_{i,i} = 2$ producen buenos resultados mientras que con $K_{i,i} \geq 3$ ya se empiezan a observar inestabilidades. Por lo tanto, podemos fijar $K_{i,i} = 2$ (que coincide con el primer valor encontrado en la Sección 5 y es por supuesto mayor al valor final fijado considerando la trayectoria real en ese caso).



(a) $K_{i,i} = 1$ (b) $K_{i,i} = 2$

(c) $K_{i,i} = 3$ (d) $K_{i,i} = 4$

Figura 17: Análisis de distintos valores de las constantes de control. En verde, el camino a seguir. En amarillo, el camino dado por la estimación de la pose según odometría. En azul, el camino dado por la estimación de la pose según EKF. En rojo, el realizado en realidad. Algunos de estos son imperceptibles por estar cubiertos por el camino rojo.

8. Sistema desarrollado

Se siguió un esquema muy similar al utilizado durante la cursada, incluso reutilizando parte de los archivos. En la carpeta `src` se encuentran definidos los nodos, cada uno en una carpeta diferente (excepto los que son para tareas de testeo, llamados loggers, los cuales no explicaremos). Básicamente definimos un nodo por cada sección de este trabajo:

- `odom`: modelo cinemático, odometría (Sección 3). Define el nodo `robot_odometry`. Recibe comandos de velocidad mediante mensajes de tipo `geometry msgs/Twist` por el tópico `/robot/cmd_vel`. Publica la transformación $odom \rightarrow base\_link$ (${}^{odom}T_R$) por el tópico `tf`, y también la pose y la velocidad como mensajes de tipo `nav_msgs/Odometry` por el tópico `/robot/odometry`. También se encuentra el nodo `loggerOdom`.

- `lazo_abierto`: generación de la trayectoria y posibilidad de realizar un seguimiento a lazo abierto (Sección 4). Define los nodos `trajectory_generator` (generación de la trayectoria) y `trajectory_follower` (sólo para poder realizar pruebas sin tener lazo cerrado definido, luego es reemplazado por `closed_trajectory_follower`).

- `lazo_cerrado`: control a lazo cerrado (Sección 5). Incluye el algoritmo pure pursuit y el controlador proporcional. Define el nodo `closed_trajectory_follower`. Se publican comandos de velocidad de control mediante mensajes de tipo `geometry_msgs/Twist` por el tópico `/robot/cmd_vel`. La implementación de pure pursuit usa como base la vista en clase pero con el agregado de la variable `previous_closest` para recordar la última pose más cercana del arreglo de waypoints, y sólo buscamos entre los más cercanos al anterior. Esto se hizo porque la posición inicial coincide con la final y esto traía problemas para comenzar/terminar la trayectoria, ya que se confundían estos puntos (o bien no podía arrancar porque consideraba que ya estaba lo suficientemente cerca del waypoint final, o volvía a empezar porque definía como goal el waypoint inicial en lugar del final). También se encuentra en nodo `logger`.

- `laser`: manejo de la información provista por el láser y detección de postes (Sección 6.1). Define el nodo `landmark_detector`. Publica *landmarks*, en el marco del robot, mediante mensajes de tipo `robmovil_msgs/LandmarkArray` por el tópico `/landmarks`.

- `robmovil_ekf`: estimación de la pose utilizando la información del láser y el filtro de Kalman extendido (Sección 6.2). Define el nodo `localizer`. Recibe los mensajes de tipo `nav_msgs/Odometry` publicados por el tópico `/robot/odometry`. Publica la transformación $map \rightarrow base\_link\_ekf$ (${}^{map}T_{ekf}$) por el tópico `tf`. También se encuentra en nodo `loggerEKF`.

En las figuras 18 y 19, generadas con `rqt_graph`, se pueden ver gráficamente las interacciones entre los nodos principales, mediante mensajes en los diferentes tópicos, que acabamos de describir en el texto. En la implementación también se incluyen otros archivos ya definidos durante la materia, los cuales no fueron modificados:

- `vrep`: contiene el archivo con el entorno simulado para V-REP.
- `ros-vrep-util`: define el nodo para tomar el reloj de la simulación en V-REP.
- `kfilter`: implementación del filtro de Kalman extendido.
- `rviz_plugin_covariance`: plugin de covarianza para rviz.
- `robmovil_msgs`: definiciones de mensajes, como se explica en el enunciado del trabajo.

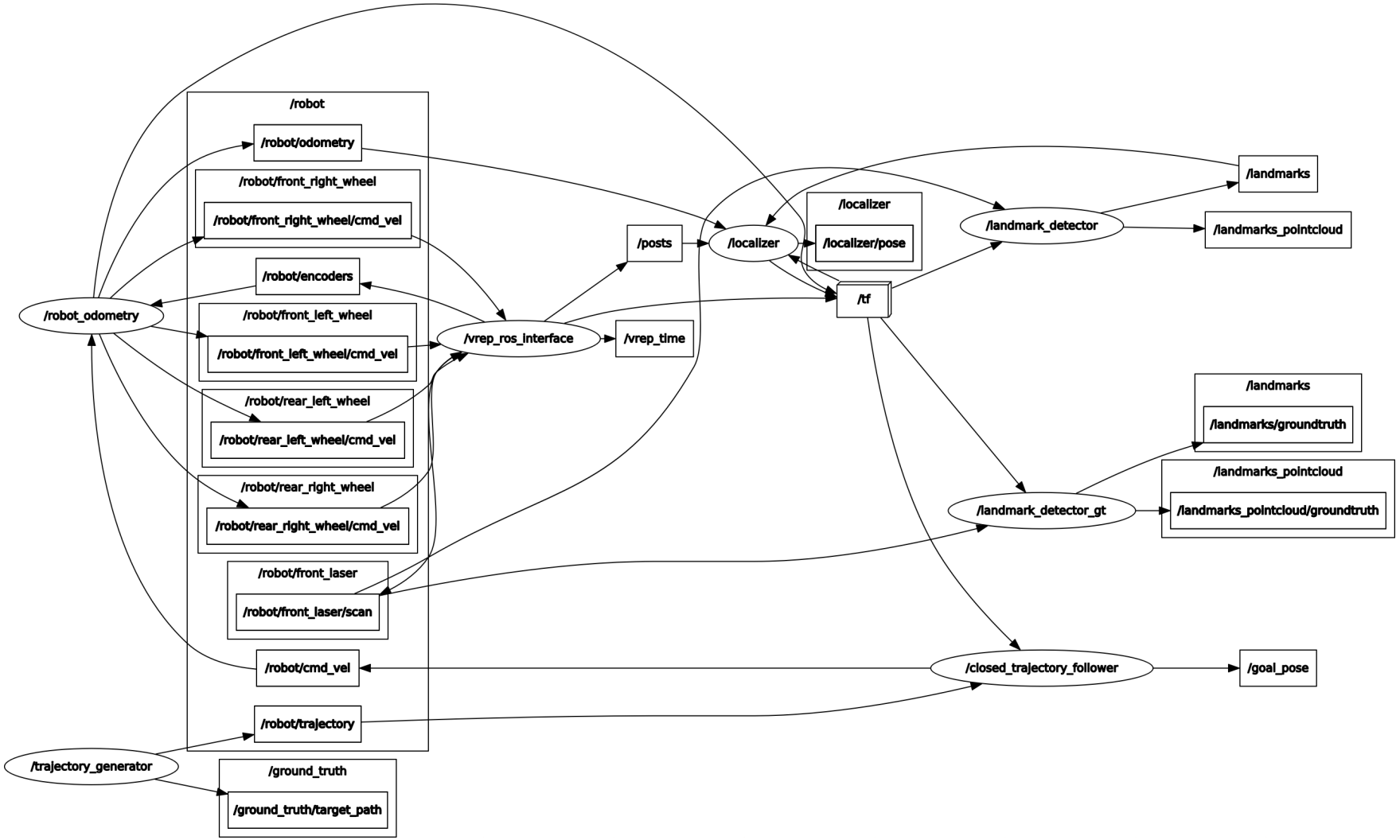


Figura 18: Esquema de subscripción a tópicos de los nodos principales.

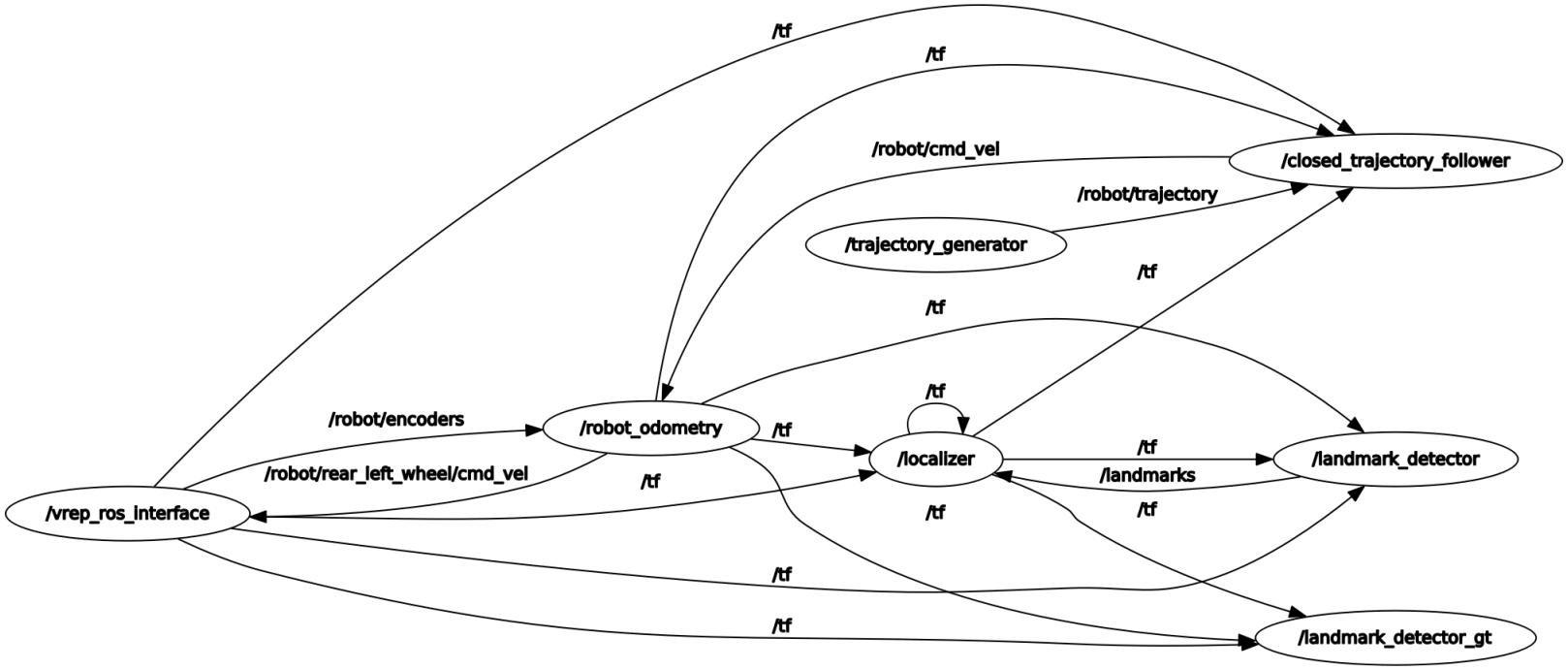


Figura 19: Comunicación de mensajes de los nodos principales.

Para su ejecución debemos proceder de igual forma que en los talleres. Esto es, primero compilar el código con `catkin build` en donde se encuentra la carpeta `src` y luego configurar las variables de entorno del workspace con `source devel/setup.bash`. Iniciar el ROS Master mediante `roscore` (en otra terminal). Abrir el entorno de simulación `vrep/omni_ekf.ttt` con V-REP. Opcionalmente, abrir rviz con la configuración `rviz_config_tp.rviz` provista. Para lanzar los nodos, hacer `roslaunch robmovil_ekf robmovil_ekf_lazo_cerrado.launch` (o cualquiera de los otros archivos launch incluidos en las distintas carpetas para probar las versiones parciales del trabajo).

Un detalle de la implementación, un tanto técnico y engorroso, que sería bueno corregir es el siguiente. Entre los archivos de lazo cerrado hay uno (`tf_utils.hpp`) que define funciones para obtener transformaciones. Estas funciones son bastante útiles y se desea utilizarlas también en los archivos de algunos de los demás nodos. Dado que no compilaba correctamente ninguna de las formas que se intentaron para mover este archivo a otra ubicación más razonable (teniendo en cuenta que siempre es deseable tener un código modular, que separe lo más posible las diferentes funcionalidades), se optó por hacer que los demás nodos utilicen el archivo en esa misma carpeta (esto implicó cambiar sus archivos `CMakeLists.txt` y `package.xml`). Queda como trabajo futuro estudiar catkin más a fondo para corregir esta pequeña desprolijidad.

9. Conclusiones

Presentamos un robot capaz de seguir trayectorias utilizando información provista por un sensor láser. Estudiamos las ecuaciones de cinemática directa e inversa, que relacionan la velocidad del robot con las velocidades de las ruedas, y a partir de esto pudimos dar la estimación odométrica. También utilizamos el sensor láser para proveer otra estimación de la pose del robot, con la ayuda del filtro de Kalman extendido. Como vimos, estimar la pose del robot es útil para poder seguir correctamente trayectorias. La estimación utilizando sólo los contadores de vueltas de las ruedas no es lo suficientemente buena en comparación a la utilización del sensor láser. En general, los métodos que utilicen más información "estable" del entorno (como los postes fijos) serán mejores para estimar la pose. Por otro lado, generamos una trayectoria suave que el robot debe seguir aplicando el algoritmo pure pursuit, dando además los comandos de velocidad que muevan adecuadamente al robot. Estas velocidades se ajustan en función de la pose actual estimada del robot.

A pesar de que se hicieron algunas simplificaciones de cálculos y suposiciones para facilitar la tarea, el resultado fue un robot logra obtener una buena estimación de su posición y logra seguir adecuadamente una trayectoria planteada. Este proyecto podría ser utilizado en la vida real sin mayores modificaciones (más allá de las propias que requiera la aplicación).

Se podrían realizar aún más experimentos para ajustar mejor las constantes del sistema. Es importante destacar que en este trabajo no se buscó optimizar estos valores lo mejor posible, sino simplemente mostrar un panorama general de la situación para comprender qué fenómenos ocurren y por qué.

Uno de los problemas encontrados es que si tuviésemos que hacer que el robot pase forzosamente por determinados lugares entonces el algoritmo pure pursuit tal como lo vimos en clase no es el más adecuado, ya que no logra manejar correctamente trayectorias que tengan *ciclos*, es decir, que pasen varias veces por un mismo punto, ni tampoco trayectorias que piden pasar por puntos muy cercanos en momentos diferentes (en estos casos podría ocurrir que el robot "saltee" gran parte del camino).

Otros problemas incluyen no considerar la aceleración, tanto en la generación de la trayectoria (se podría utilizar un polinomio de grado 5, como el explicado en [7, Capítulo 9.2])

como en la estimación de la posición (durante los experimentos se pudo observar, por ejemplo, que tomaba aproximadamente 0,1s en pasar de 0 a 0.25m/s en v_x), y también utilizar un modelo muy sencillo de control (existen controladores llamados PID que consideran no sólo el controlador proporcional sino también integral y derivativo [7, Capítulo 11.3]). Para este caso no es realmente necesario realizar estas mejoras porque no requerimos demasiada precisión, pero sí podría serlo para otra aplicación.

Referencias

[1] F. Adascalitei and I. Doroftei. Practical applications for mobile robots based on mecanum wheels - a systematic survey. *Romanian Review Precision Mechanics, Optics and Mechatronics*, pages 21–29, 01 2011.

[2] R. C. Coulter. Implementation of the pure pursuit path tracking algorithm. Technical Report CMU-RI-TR-92-01, Carnegie Mellon University, Pittsburgh, PA, January 1992.

[3] J. Gonçalves, J. Lima, and P. G. Costa. Real time tracking of an omnidirectional robot - an extended kalman filter approach. In J. Filipe, J. Andrade-Cetto, and J. Ferrier, editors, *ICINCO 2008, Proceedings of the Fifth International Conference on Informatics in Control, Automation and Robotics, Robotics and Automation 2, Funchal, Madeira, Portugal, May 11-15, 2008*, pages 5–10. INSTICC Press, 2008.

[4] N. Houshangi and T. Lippitt. Omnibot mobile base for hazardous environment. In *Engineering Solutions for the Next Millennium. 1999 IEEE Canadian Conference on Electrical and Computer Engineering (Cat. No.99TH8411)*, volume 3, pages 1357–1361 vol.3, 1999.

[5] T.-V. How. *Development of an anti-collision and navigation system for powered wheelchairs*. University of Toronto (Canada), 2010.

[6] B. E. Ilon. Wheels for a course stable selfpropelling vehicle movable in any desired direction on the ground or some other base, Apr. 8 1975. US Patent 3,876,255.

[7] K. M. Lynch and F. C. Park. Modern robotics: Mechanics, planning, and control. Cambridge Univeristy Press, 2017.

[8] K. Nagatani, S. Tachibana, M. Sofne, and Y. Tanaka. Improvement of odometry for omnidirectional vehicle using optical flow information. In *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113)*, volume 1, pages 468–473 vol.1, 2000.

[9] H. Taheri, B. Qiao, and N. Ghaeminezhad. Kinematic model of a four mecanum wheeled mobile robot. *International Journal of Computer Applications*, 113(3):6–9, March 2015.

[10] L. Xie, C. Scheifele, W. Xu, and K. A. Stol. Heavy-duty omni-directional mecanum-wheeled robot for autonomous navigation: System development and simulation realization. In *2015 IEEE International Conference on Mechatronics (ICM)*, pages 256–261, 2015.