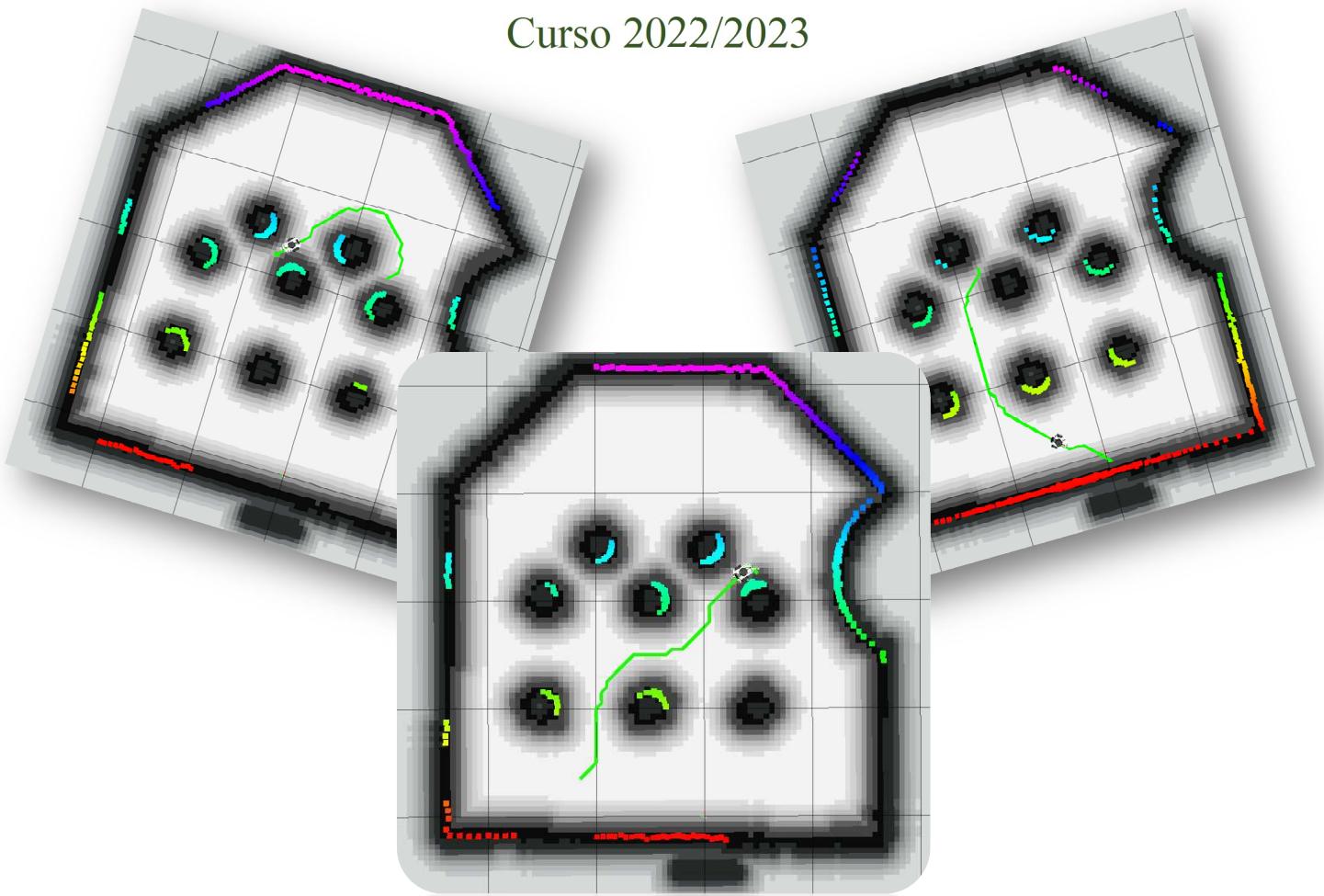


# MEMORIA PROYECTO: ALPHABOT PLANNER

*Memoria de trabajo desarrollado y resultados obtenidos en simulación de la planificación de caminos y control de un robot móvil en configuración diferencial*

Control y Programación de Robots – 4ºGIERM

Curso 2022/2023



## AUTORES:

ARTURO RENATO ÚBEDA QUILÓN  
SERGIO LEÓN DONCEL  
ISAAC RODRÍGUEZ GARCÍA  
ÁLVARO GARCÍA LORA

## CONTENIDOS

<b>0. INTRODUCCIÓN.....</b>	<b>2</b>
<b>1. LOCALIZACIÓN.....</b>	<b>3</b>
<b>1.1. MÉTODO 1 .....</b>	<b>5</b>
<b>1.2. MÉTODO 2 .....</b>	<b>6</b>
<b>1.3. MÉTODO 3 .....</b>	<b>6</b>
<b>1.4. COMPARACIÓN DE MÉTODOS.....</b>	<b>7</b>
<b>2. CONTROLADOR ALTO NIVEL .....</b>	<b>8</b>
<b>2.1. TÉCNICA USADA: PURE-PURSUIT .....</b>	<b>8</b>
<b>2.2. EXPERIMENTOS Y RESULTADOS OBTENIDOS.....</b>	<b>10</b>
<b>3. PLANIFICADOR DE CAMINOS .....</b>	<b>12</b>
<b>3.1. ARQUITECTURA DEL MAPA DE COSTES .....</b>	<b>12</b>
<b>3.2. ALGORITMO PLANIFICACIÓN: DIJKSTRA .....</b>	<b>13</b>
<b>3.3. EXPERIMENTOS Y RESULTADOS OBTENIDOS .....</b>	<b>15</b>
<b>4. SISTEMA COMPLETO.....</b>	<b>18</b>
<b>4.1. EXPERIMENTOS Y RESULTADOS OBTENIDOS .....</b>	<b>19</b>
<b>5. AMPLIACIONES FUTURAS.....</b>	<b>22</b>
<b>6. REFERENCIAS .....</b>	<b>23</b>

## 0. INTRODUCCIÓN

En la presente memoria se recogerán las explicaciones del proyecto realizado durante el curso sobre la simulación en Gazebo de un robot móvil con ruedas en configuración diferencial desarrollado en ROS.

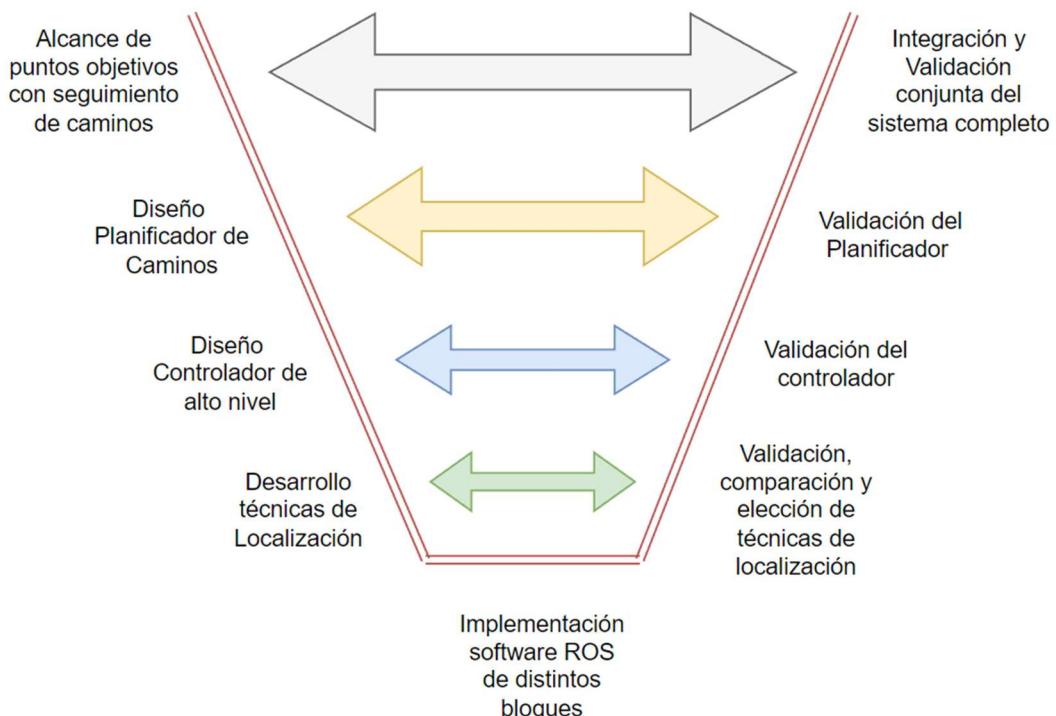
Se detallará el trabajo desarrollado en cuanto a localización, control de alto nivel y planificación de caminos con posibilidad de extensión a planificación de trayectorias.

Comentaremos las decisiones de diseño tomadas y mostraremos los resultados experimentales obtenidos en diferentes condiciones, desacoplando cada una de las partes y el sistema general.

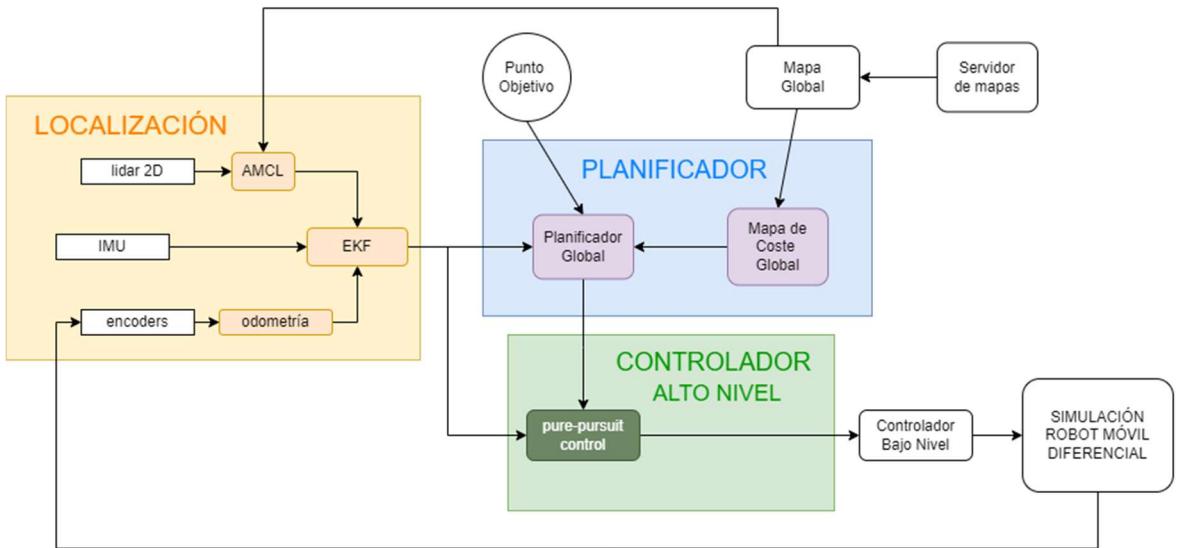
Los objetivos que nos hemos fijado consisten en ser capaces de aplicar diferentes técnicas de localización, implementar alguna de las técnicas de control de alto nivel vistas en la asignatura y generar caminos mediante planificación a partir de mapas de coste obtenidos del entorno de simulación.

Partiremos para ello de un paquete de robot diferencial existente del cual nos quedaremos únicamente con la parte de modelado y control de bajo nivel del mismo. Añadiremos sensores para aplicar distintas técnicas de localización y desarrollaremos nodos propios para conseguir los objetivos fijados.

Podemos seguir una metodología en V como arquitectura de diseño del sistema:



En el siguiente esquema se muestra de forma resumida el funcionamiento del sistema que se pretende alcanzar:



## 1. LOCALIZACIÓN

Planteamos la localización del robot como resultado de la fusión sensorial de los sensores que lleva a bordo: IMU, Lidar y medidas de velocidad. En este trabajo, se presentan tres métodos de localización basados en la fusión mediante un EKF, implementada a través del paquete “robot\_localization” de ROS:

- **Odometría por modelo cinemático + IMU + AMCL (lecturas del Lidar):** para poder introducir las lecturas del Lidar al EKF, el paquete “robot\_localization” requiere un tipo de dato concreto. Hemos usado el paquete AMCL de ROS para generar a partir de las lecturas del lidar el tipo de dato adecuado.

Aunque no es un esquema del todo correcto (estaríamos introduciendo la salida de un estimador a la entrada de otro), lo compararemos con otros métodos de localización. Sin embargo, es un esquema usado, hemos encontrado algunos ejemplos de otros proyectos de investigación en los que se usa un esquema similar:

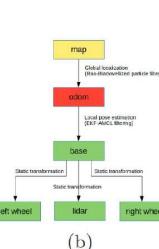
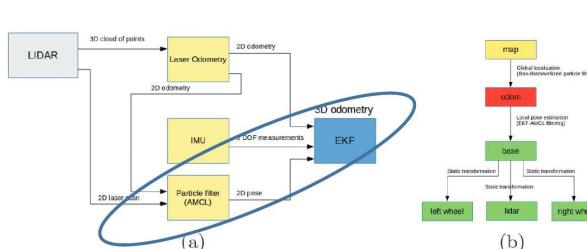


Fig. 1: Developed hybrid AMCL-EKF pose estimation: Combined AMCL-EKF technique for 3 DOF local pose (a) and connections between coordinate frames in global localization approach

[https://link.springer.com/chapter/10.1007/978-3-030-20131-9\\_279](https://link.springer.com/chapter/10.1007/978-3-030-20131-9_279) => Página 6

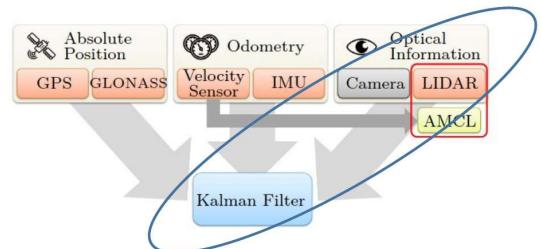
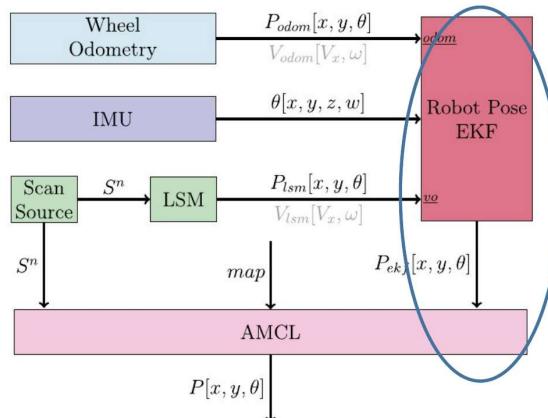


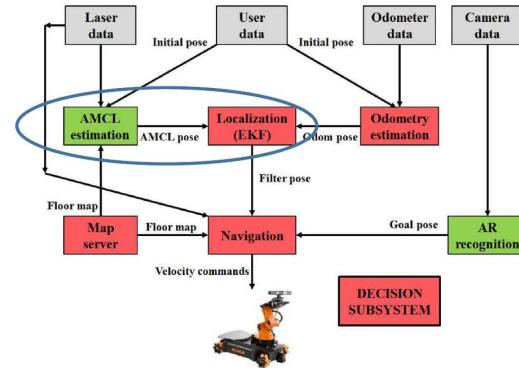
Figure 2. Sketch of available sensors used for localization. All sensors shown in red are included in our setup. This paper focuses on the LIDAR and AMCL, highlighted with a red frame.

<https://pdfs.semanticscholar.org/40ad/b1ceb2679bc61f03b0fdaf5f5bbf4802169cb.pdf> => Página 3



<https://link.springer.com/article/10.1007/s00170-022-08883-0>

<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7251505> => Página 4



<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7251505> => Página 4

- **Odometría por modelo cinemático + IMU + AMCL (lecturas del Lidar) + Odometría por lidar:** sobre el método anterior, se procesa la información del lidar con dos métodos diferentes que proporcionan enfoques de la misma (no se trata de usar la misma información dos veces) y tratar así de mejorar el método anterior.
- **Odometría por modelo cinemático + Odometría por lidar + IMU, separados del AMCL:** se fusionan todos los sensores incrementales (miden las variaciones de pose entre instantes y se suman a la pose anterior) para dar lugar a una odometría filtrada.

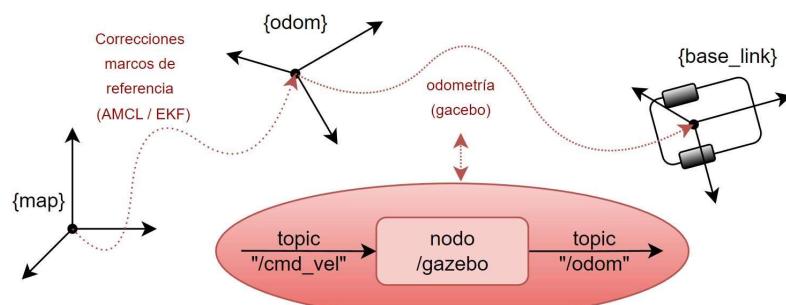
Por otro lado, el AMCL se encarga de corregir las derivas de la odometría filtrada.

Por último, contamos con un cuarto método, que sería emplear directamente el Ground Truth del simulador (disponible añadiendo un plugin de gazebo en el modelo URDF del robot), que nos proporciona la pose exacta del robot en cada instante. Este es el método que empleamos para realizar pruebas de funcionamiento e integración, con vistas a trabajar de forma desacoplada.

```
<!-- Added Ground Truth plugin -->
<gazebo>
  <plugin name="p3d_base_controller" filename="libgazebo_ros_p3d.so">
    <alwaysOn>true</alwaysOn>
    <updateRate>50.0</updateRate>
    <bodyName>base_link</bodyName>
    <topic>cmd_gazebo</topic>
    <state>pose</state>
    <topicName>pose</topicName>
    <frameName>map</frameName>
    <x><offset>0 0 </offset></x>
    <y><offset>0 0 </offset></y>
    <z><offset>0 0 </offset></z>
  </plugin>
</gazebo>
```

Forma final del plugin ground\_truth dentro de alpha\_bot\_description/urdf/include/robot.gazebo.xacro

Los tres primeros métodos anteriormente descritos tratan de mejorar la localización del robot frente a resolver este problema tan sólo con la odometría basada en el modelo cinemático del robot, usando el AMCL/EKF para corregir sus derivas:



Según la norma REP 105 de ROS, la transformación entre el marco de referencia fijo del mundo (“map”) y el marco de referencia móvil solidario al robot (“base\_link”) no se calcula directamente, sino que se realiza en dos etapas, usando un marco de referencia intermedio, “odom”:

- odom -> base\_link: esta transformación se publica usando solamente el modelo cinemático del robot: el nodo de gazebo se suscribe al topic “/cmd\_vel” (en nuestro trabajo, “/alpha\_bot/mobile\_base\_controller/cmd\_vel”) y haciendo uso del modelo URDF del robot, es el único que publica la pose estimada del robot en el topic “/odom” (en el trabajo, “/alpha\_bot/mobile\_base\_controller/odom”), como podemos apreciar en esta imagen.

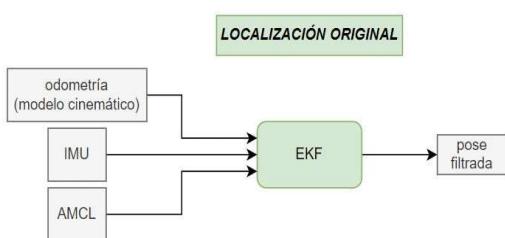


El uso de un marco de referencia intermedio se debe a que los métodos de localización incrementales (odometría) dan lugar en general a variaciones suaves, mientras que las correcciones de métodos absolutos provocan saltos, que pueden dificultar el diseño de controladores y otros algoritmos. Por tanto, se realizan estos diseños sobre el marco de referencia odom, teniendo siempre en cuenta que deben realizarse correcciones sobre dicho marco de referencia.

- map -> odom: tanto el AMCL como el EKF pueden realizar las correcciones sobre la odometría. En los dos primeros métodos, la corrección la realizará el EKF, que aglutina todos los sensores, y en el segundo, la corrección la hará el AMCL, para corregir las derivas de todas las fuentes de odometría (métodos puramente incrementales) fusionadas por el EKF.

Antes de comentar los resultados obtenidos, queremos destacar la presencia de un error en la localización que afectará a todos los métodos que hemos comentado anteriormente (excepto al de Ground Truth): la odometría mediante el modelo cinemático introduce aleatoriamente un gran error, semejante al que podría producirse si el robot “patinara” durante un intervalo de tiempo considerable. Sin embargo, se han tomado los datos en conjuntos de 5 experimentos para cada método de localización, que no contengan este error para poder compararlos en las mismas condiciones.

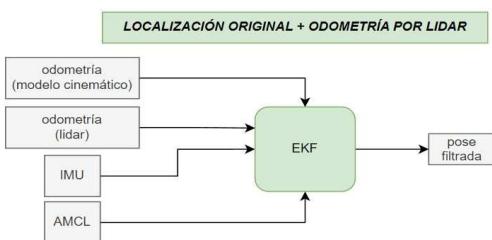
### 1.1. MÉTODO 1



Este es el método que empleamos originalmente para la localización del robot. En este caso, la transformación odom -> base\_link, que representa la odometría por modelo, será publicada por Gazebo, mientras que el EKF realizará las correcciones de “odom” publicando la transformación “map” -> “odom”. Se han obtenido los siguientes resultados:

Magnitud/Parámetros	Media de los errores medios	Desviación típica de los errores medios	Media de los errores máximos	Desviación típica de los errores máximos
Error en posición (cm)	8.097	0.235	13.107	0.645
Error en orientación (°)	-0.847	0.884	4.340	1.325

## 1.2. MÉTODO 2

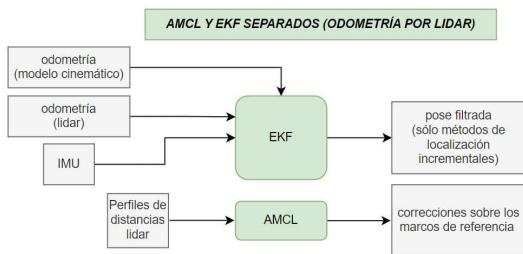


Añadimos el nodo “laser\_scan\_matcher” del paquete “scan\_tools” de ROS, que realiza un cálculo incremental de la pose a partir de las lecturas del lidar (odometría mediante lidar), tratando de mejorar el esquema anterior.

Resultados obtenidos:

Magnitud/Parámetros	Media de los errores medios	Desviación típica de los errores medios	Media de los errores máximos	Desviación típica de los errores máximos
Error en posición (cm)	8.1173	0.415	13.584	0.386
Error en orientación (°)	-0.879	0.377	4.043	0.374

## 1.3. MÉTODO 3



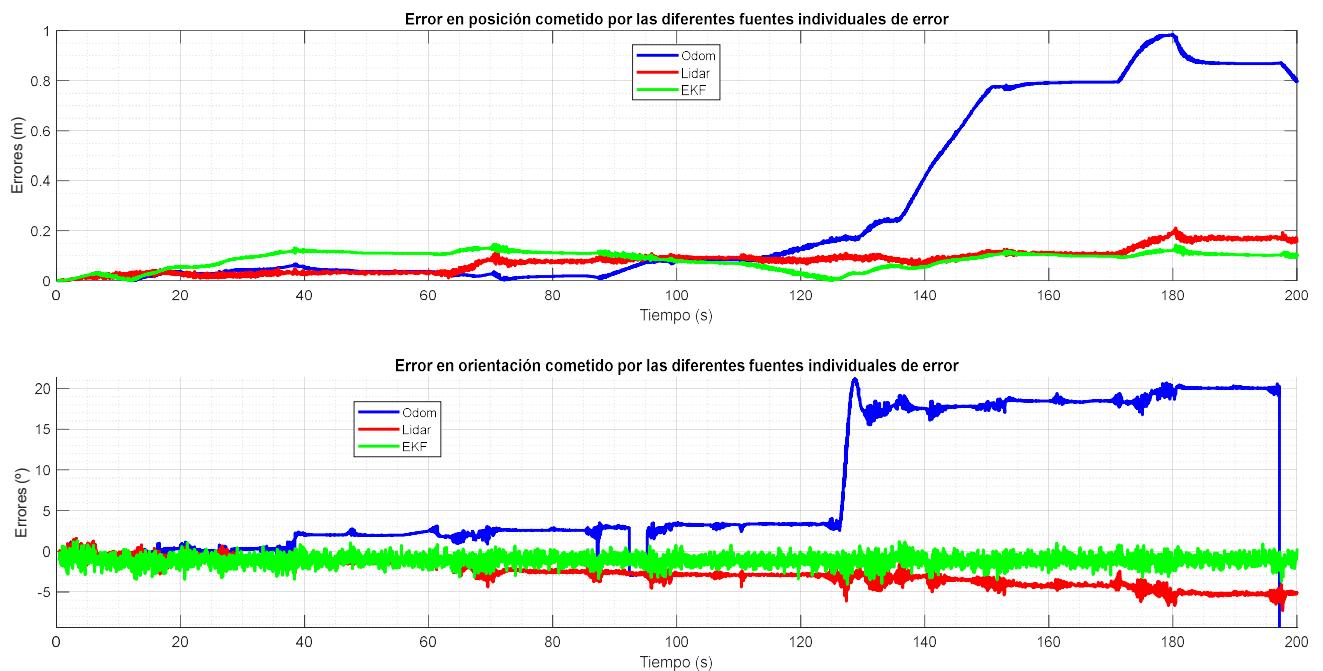
En este método, evitamos introducir la salida de un estimador en la entrada de otro separando sus funciones: el AMCL corregirá a todas las fuentes de odometría fusionadas por el EKF.

Resultados obtenidos:

Magnitud/Parámetros	Media de los errores medios	Desviación típica de los errores medios	Media de los errores máximos	Desviación típica de los errores máximos
Error en posición (cm)	7.259	1.344	13.324	2.349
Error en orientación (°)	-0.427	1.480	5.218	1.215

#### **1.4. COMPARACIÓN DE MÉTODOS**

Mientras la adición de la odometría láser al planteamiento original no proporciona cambios significativos, la separación entre EKF y AMCL sí consigue reducir los errores medios y máximos en general. Sin embargo, la respuesta de la odometría por láser ayuda a corregir las derivas de la odometría por modelo cuando tiene lugar un deslizamiento: podemos comprobar que la odometría por láser sigue manteniendo un error similar al instante anterior a la deriva (como es lógico porque el lidar no ha compartido esa deriva), y como la IMU tampoco indica cambios de velocidad tan acusados, la odometría filtrada en el EKF no tiene una variación tan significativa, a pesar de la deriva de la odometría por modelo:



## 2. CONTROLADOR ALTO NIVEL

### 2.1. TÉCNICA USADA: PURE-PURSUIT

Nos hemos basado en la técnica de control de alto nivel de persecución pura vista en la asignatura. A partir de ella hemos hecho una serie de adaptaciones particulares para nuestra configuración y hemos planteado algunas simplificaciones para su implementación.

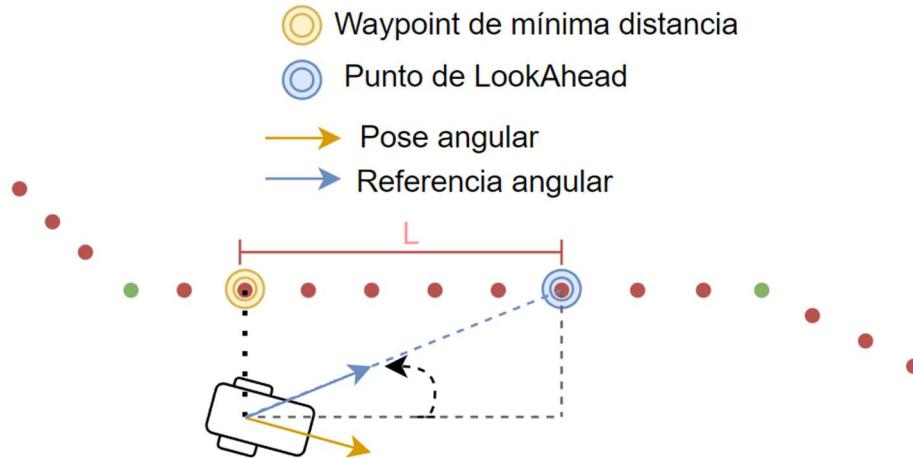
En primer lugar, dado que para el funcionamiento del sistema completo es previsible la llegada de un conjunto de waypoints de distinto tamaño por parte del planificador, durante el desarrollo del nodo asociado al controlador se han generado conjuntos de puntos ficticios pertenecientes a una trayectoria para trabajar. El objetivo es comprobar el funcionamiento del controlador de forma independiente al resto de bloques.

Un punto importante a destacar es el hecho de que el vector de waypoints vendrá ordenado en cuanto a requerimientos de paso se refiere, es decir, los puntos iniciales serán aquellos a los que el robot deberá dirigirse en primer lugar en el recorrido deseado y viceversa.

Como paso previo al uso del conjunto de waypoints originales, se realiza una interpolación lineal entre cada uno de ellos y el siguiente. De esta forma contaremos con un vector de waypoints extendido. Las orientaciones de cada nuevo punto intermedio se calculan de tal forma que sigan la dirección de la recta que une los puntos originales sucesivos.

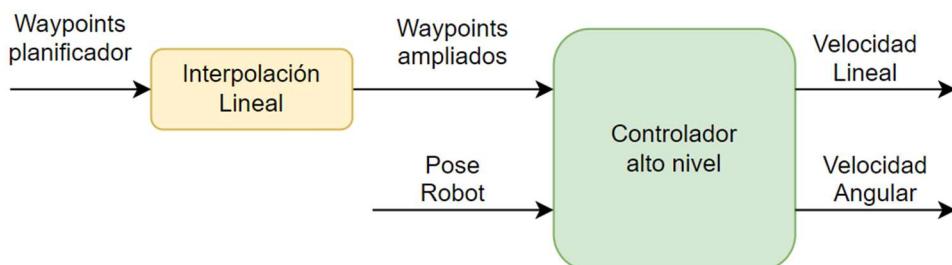
En resumen, los pasos que hemos adaptado en cuanto a la técnica de control Pure-Pursuit para lograr el funcionamiento de este bloque del sistema han sido:

- En cada instante de muestreo hallamos cual es el waypoint de mínima distancia respecto a la posición actual. Esto se consigue recorriendo el conjunto de puntos intermedios ampliado y calculando el módulo del vector que une la pose actual con cada uno de ellos, seleccionando en cada momento el que presente una distancia menor.
- A partir del punto anterior, teniendo en cuenta la consideración de que estarán ordenados en el sentido de avance deseado, tomaremos como punto de lookahead aquel que se encuentre a una distancia de lookahead ( $L$ ) por delante de dicho waypoint de mínima distancia. Lo llamativo es que se ha decidido finalmente tomar el valor de  $L$  en términos de números de puntos en lugar de distancia. Esto es así para evitar problemas de no seguimiento y saltos de tramos en los casos en los que el camino dado presentara cruces o aproximaciones notables sobre sí mismo.
- Conseguimos generar una referencia de ángulo para la reorientación del vehículo aplicando relación trigonométrica sencilla entre la pose actual y el punto de lookahead al cual se pretende ir en cada momento.



- La velocidad lineal, será constante hasta que se acerque de forma considerable al objetivo final, momento en el cual pasa a controlarse en función de la distancia a éste para una llegada suave en la parada, evitando de esta forma sobreoscilaciones indeseadas alrededor del punto final. En el caso de extensión a seguimiento de trayectorias esta velocidad pasaría a ser variable con vista a cumplir las restricciones temporales establecidas.
- La velocidad angular se calcula en función la diferencia existente entre la referencia de ángulo y el ángulo de orientación de la pose instantánea del robot (error en ángulo) y el tiempo deseado en la corrección de dicho error.
- Para facilitar la tarea al controlador y conseguir un seguimiento más oportuno del camino se realiza un primer giro con velocidad lineal nula en el que el vehículo se orienta a favor del primer punto de lookahead calculado.
- Típicamente la referencia usada es la local al robot, transformando la pose instantánea de coordenadas respecto al punto inicial que tomamos como global. Sin embargo, en nuestro sistema usamos coordenadas globales ya que nos resulta más simple a la hora de trabajar en las simulaciones ya que los waypoints y pose objetivo vendrán dados en las mismas coordenadas globales.

Por tanto, el esquema de entradas y salidas del nodo desarrollado para el controlador descrito es el siguiente:



## 2.2. EXPERIMENTOS Y RESULTADOS OBTENIDOS

Estudiamos el comportamiento ante variación de la velocidad lineal fija de movimiento. Se pretende conocer de forma experimental un rango de velocidades lineales de operación válido, tanto para seleccionar una velocidad de trabajo como para tener una idea de las velocidades máximas y mínimas que podríamos usar de cara al desarrollo del seguimiento de trayectorias.

Hemos realizado una serie de experimentos de seguimiento de caminos bajo las mismas condiciones variando las velocidades lineales constantes de movimiento. Los resultados obtenidos de errores máximos y medios cometidos obtenidos se recogen en la siguiente tabla:

Velocidad Lineal [m/s]	Error máximo [cm]	Error medio [cm]
0.10	14.79	2.45
0.12	8.28	2.08
0.14	10.51	2.06
0.16	12.26	2.28
0.18	12.49	2.32
0.20	32.36	12.09

Para velocidades inferiores a 0.12 m/s el movimiento resulta muy lento y el error cometido comienza a incrementarse. Trabajar con velocidades cercanas a 0.20 m/s y superiores supone cometer demasiado error respecto al camino, pudiéndose producir colisiones con los obstáculos del mapa.

Experimentalmente hemos decidido que la velocidad nominal del robot será de 0.15 m/s, llegando a una solución de compromiso entre rapidez en completar el recorrido y reducción de errores cometidos.

Llevamos a cabo un análisis de la influencia en cuanto a funcionamiento de la distancia de lookahead seleccionada. La idea que se pretende es la de seleccionar una L de garantías para la velocidad lineal concreta escogida de 0.15 m/s.

Distancia Lookahead [nº waypoints]	Error máximo [cm]	Error medio [cm]
3	11.79	2.14
5	10.44	2.08
7	15.79	2.75
9	22.52	3.79
11	15.38	5.82

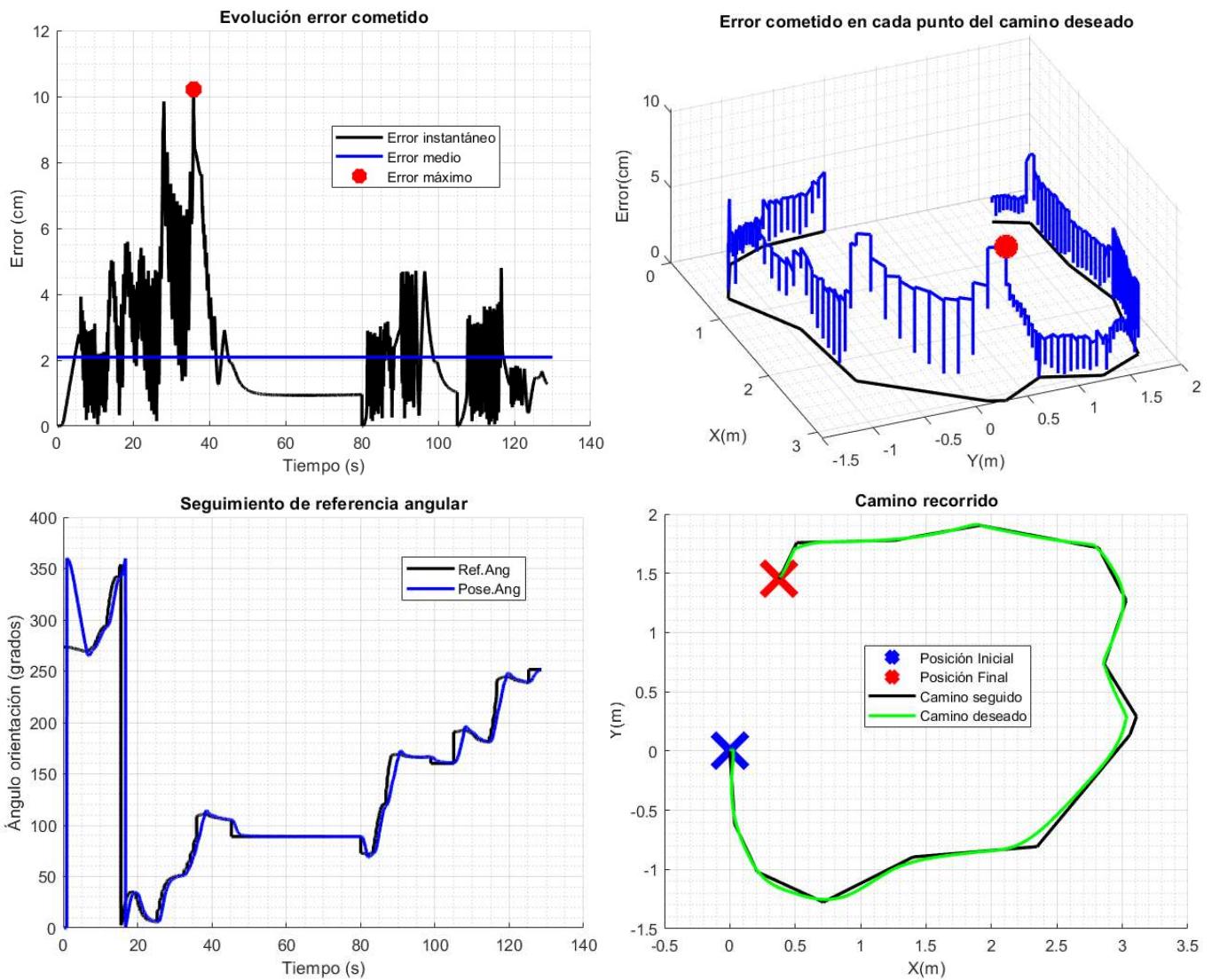
Trabajaremos con una distancia de lookahead de 5 waypoints del conjunto ampliado por delante del correspondiente al de mínima distancia. Experimentalmente observamos que con dicho valor el vehículo describe curvas más suaves en el seguimiento del camino de tal forma que comete un menor error, en general, en comparación con valores inferiores donde el robot trata de alcanzar puntos variables más cercanos puesto los supera o con valores superiores donde no se ajusta tanto al camino particular deseado.

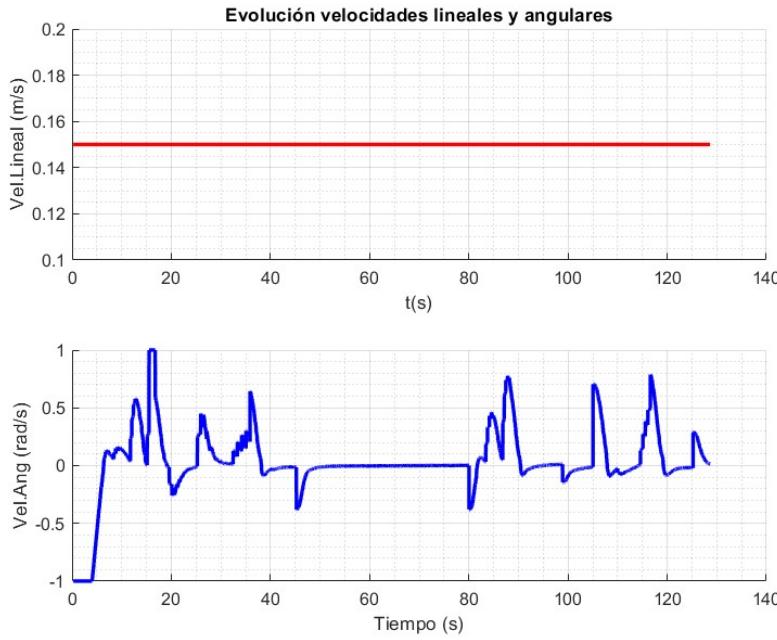
Una vez seleccionado L y velocidad lineal de trabajo realizamos 20 replicaciones de un experimento para dar una métrica de los resultados previsibles de funcionamiento del controlador. Cada experimento consiste en hacer un recorrido alrededor de un mapa con obstáculos, simulando la llegada de distintas rachas de waypoints pertenecientes a diversos caminos, con tiempos de separación variables.

Los resultados obtenidos en cuanto a valores medios y variabilidad de las pruebas quedan resumidos a continuación:

Media de los errores máximos obtenidos [cm]	10.33
Desviación típica de los errores máximos obtenidos [cm]	0.096
Media de los errores medios obtenidos [cm]	2.10
Desviación típica de los errores medios obtenidos [cm]	0.0089

Mostramos una serie de gráficas de un experimento representativo.





### **3. PLANIFICADOR DE CAMINOS**

#### **3.1. ARQUITECTURA DEL MAPA DE COSTES**

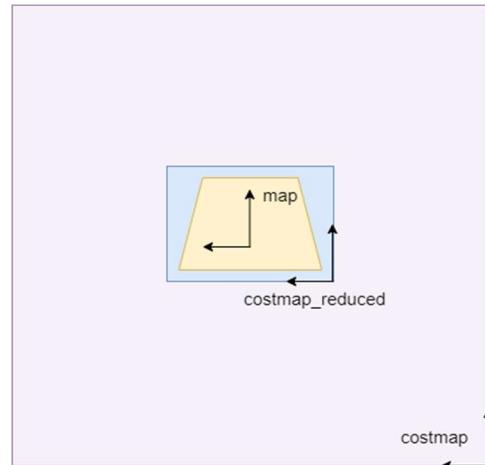
Como se ha comentado anteriormente, se va a implementar un algoritmo de planificación de caminos. Este algoritmo se basará en la información de un mapa de costes global, por lo que no tendrá en cuenta posibles modificaciones del mapa en tiempo de ejecución, sino que será tomado como estático. El planificador ha sido programado por nosotros mismos, pero para ello necesitamos hacer uso de un mapa de costes.

En este caso, el mapa de costes lo obtendremos del paquete oficial de move\_base de ROS, que incluye un nodo de costmap\_2d. Este nodo publica, entre otras cosas, el mapa de costes global en el topic “/move\_base/global\_costmap/costmap”. Por tanto, nos suscribimos a ese topic para obtener el mapa de costes correspondiente. Los parámetros han sido configurados en el archivo costmap\_common\_params.yaml para una resolución de 0.05 m/pixel, con un radio de inflación de 0.5 metros. Sin embargo, hemos observado que el costmap obtenido es de un tamaño muy superior al de nuestro mapa, por lo que la mayor parte de éste está indefinida (toma valor -1), ocupando una gran cantidad de memoria innecesaria y haciendo la planificación más costosa de lo que debería. Para arreglar esto, se propuso calcular la ‘bounding box’ del mapa del entorno dentro del mapa de costes, obteniendo por tanto un mapa de costes reducido al tamaño real del mapa, que es el que usaremos para realizar la planificación.

Además de esto, hay que tener en cuenta que tanto las posiciones objetivo como la pose actual del robot, que se obtienen de los topic ”/move\_base/goal” y ”/odometry/filtered”, respectivamente, están dadas en el marco de referencia del mapa, “map”, que se encuentra en la posición inicial del robot al comenzar el experimento. Para poder trabajar la planificación las necesitamos en el mismo sistema de referencia que el mapa de costes reducido.

Sabiendo que el sistema de referencia “map” está en el centro del mapa de costes, calculamos el punto inferior derecho de este en función de “map” como la altura y anchura dividida entre dos, y multiplicada por la resolución. Como lo que nos interesa no es el origen del mapa de costes del nodo `costmap_2d`, sino el reducido una vez calculada la ‘bounding box’ del mapa del entorno, calculamos a partir de las coordenadas origen del mapa de costes las del punto inferior derecho del mapa de costes reducido. Por tanto, teniendo la traslación entre el origen del mapa de costes reducido y el marco de referencia “map”, en el cual nos dan la posición del robot y el objetivo, podemos transformarlos para obtener estos dos puntos respecto al origen del mapa de costes reducido.

Una vez calculados los puntos de la trayectoria mediante el algoritmo elegido, se deben “antitransformar” para pasarlo del sistema del mapa reducido al marco “map” de nuevo, y publicarlos en un topic “/PlannedPath” para que el controlador pueda interpretarlos y controlar al robot para que siga el camino calculado, tal y como se ha explicado en el punto anterior.



### 3.2. ALGORITMO PLANIFICACIÓN: DIJKSTRA

Como se ha realizado el algoritmo de planificación desde cero, por mayor simplicidad en el código, se ha decidido usar el algoritmo de Dijkstra para calcular el camino de mínimo coste hasta el punto objetivo.

El algoritmo de Dijkstra es uno de los más simples y conocidos para la obtención de un camino de mínima distancia o coste. Se usa tanto en diagramas de flujo, comunicaciones y protocolos de red como en planificación de caminos, como es nuestro caso. Éste consiste en dividir el mapa en nodos, y asignar un coste a cada uno de ellos, que es lo que se ha hecho en el apartado anterior.

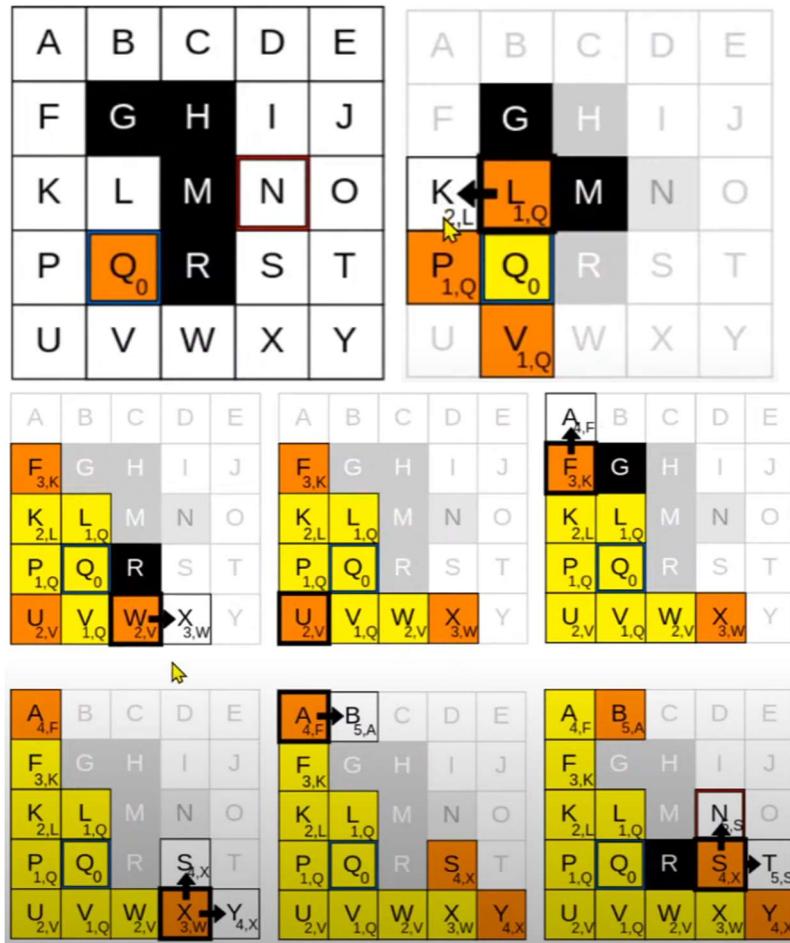
A cada nodo se le asignan, además, un identificador, un nodo previo (*parent*) y un peso (*g\_cost*), que consiste en el mínimo coste acumulado desde el nodo inicial. El algoritmo se ejecuta cada vez que se recibe una nueva pose objetivo usando *rviz*, y se ha dividido principalmente en dos fases. Los pasos a seguir son los siguientes:

- Marcar el nodo inicial con coste 0 y añadirlo a la *open\_list*, que es una lista de nodos cuyo peso está pendiente de ser actualizado.

Mientras que la *open\_list* no esté vacía:

- Buscar el nodo con menor peso de la lista.
- Se añade a la *closed\_list* para marcar que el nodo ya tiene actualizado su peso como el menor posible.

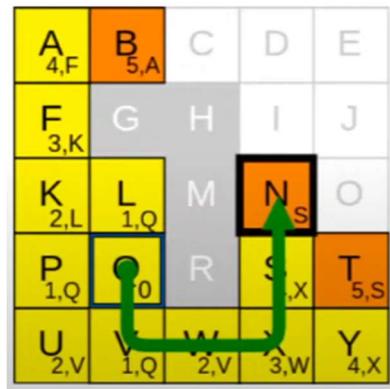
- Si el nodo encontrado es el objetivo, podemos pasar a la siguiente fase. Si no, buscamos los nodos vecinos a éste. Se nos pueden presentar tres casos para cada vecino:
  - o Si está en la *closed\_list*, se ignora, ya que su peso ya es mínimo.
  - o Si está en la *open\_list*, vemos si su peso se puede disminuir, actualizando su *g\_cost* y su nodo *parent*.
  - o Si no está en ninguna lista, se da valor a su peso, se toma como nodo previo o *parent* al nodo actual y se añade a la *open\_list*.



Para asignar pesos a los nodos vecinos, se tiene en cuenta tanto el coste del nodo a actualizar como si éste se encuentra en diagonal o es correlativo al nodo actual. En caso de ser vecinos en diagonal se asignará al peso el valor del peso del nodo que se va a tomar como ‘padre’, sumando el coste del nodo que se actualiza + 14. En caso contrario, es decir, si el vecino se encuentra en sus 4 direcciones principales, se actualizará de igual forma pero sumando 10 en vez de 14. Esto se ha hecho para permitir desplazamientos en diagonal, pero asignándole un peso mayor que si se realiza en perpendicular. Cabe destacar que los nodos con un peso de 100 (es decir, el *lethal\_cost*, que es el asignado a los obstáculos) nunca serán añadidos a la *open\_list*, y por tanto su peso se mantendrá infinito.

Una vez que la *open\_list* está vacía o hemos alcanzado el nodo final, se puede pasar a la segunda fase, que consiste en obtener los puntos de la trayectoria completa. Para ello se recorren los nodos del final al inicial, viendo cuál es el ‘padre’ de cada uno, y se guardan sus identificadores en un vector *path\_id*.

Lo único que quedaría sería obtener el vector de poses que define la trayectoria, tomando los identificadores de *path\_id* en sentido opuesto (para que empiece por el inicial y termine en el objetivo), pasar de posiciones *x* e *y* en número de celdas a metros (multiplicando por la resolución), y calcular su orientación de referencia como el ángulo entre el nodo siguiente y el actual en cada caso.

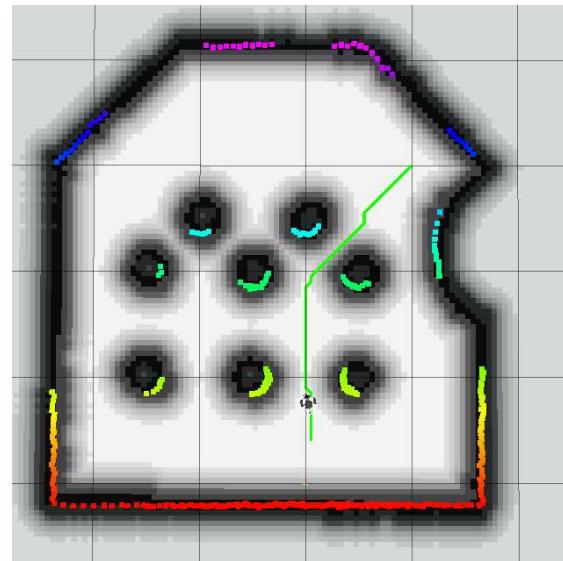


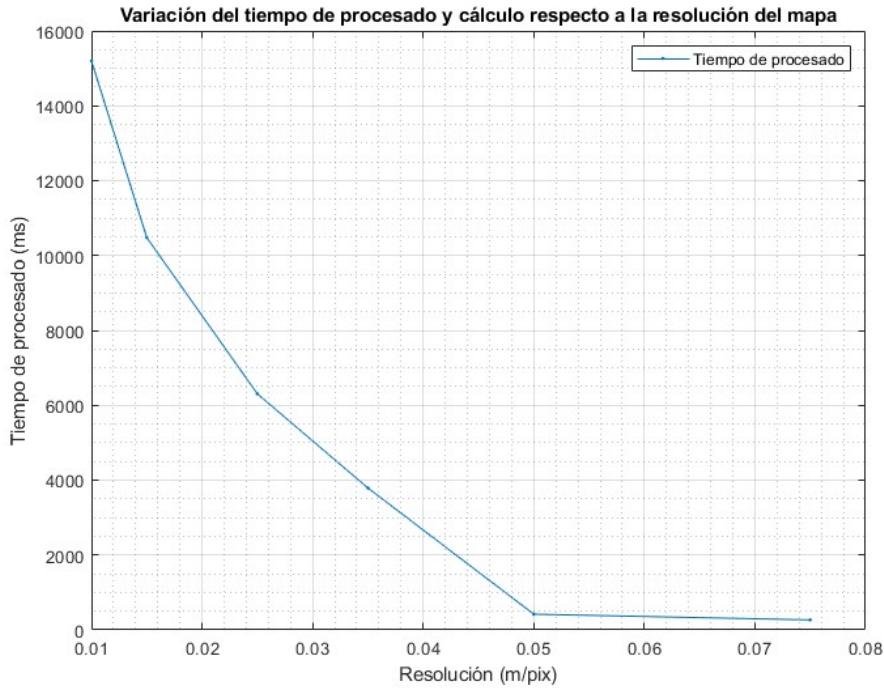
### 3.3. EXPERIMENTOS Y RESULTADOS OBTENIDOS

Para la correcta comprobación del funcionamiento del planificador se han realizado una serie de experimentos, en los que se han variado algunos parámetros del mapa, como la resolución o factor de escalado.

En primer lugar, se ha comprobado que la trayectoria se genera de forma adecuada y el robot es capaz de seguir los puntos proporcionados, llegando al punto final. Los caminos no atraviesan los obstáculos, y en caso de pedirse una posición objetivo inalcanzable, no se ofrece ninguna trayectoria.

Se ha modificado la resolución del mapa, que era inicialmente de 0.05 m/pixel. Se han realizado varias simulaciones para un punto objetivo similar, y se observa que el camino generado no varía; sin embargo, si calculamos el tiempo que tarda en actuar el planificador, es decir, el tiempo que tarda desde que recibe la posición objetivo, en calcular los puntos del camino y enviar el mensaje, podemos deducir que a mayor resolución (menos metros por pixel), más tarda en ejecutarse la planificación, ya que tendrá que recorrer más puntos del mapa.



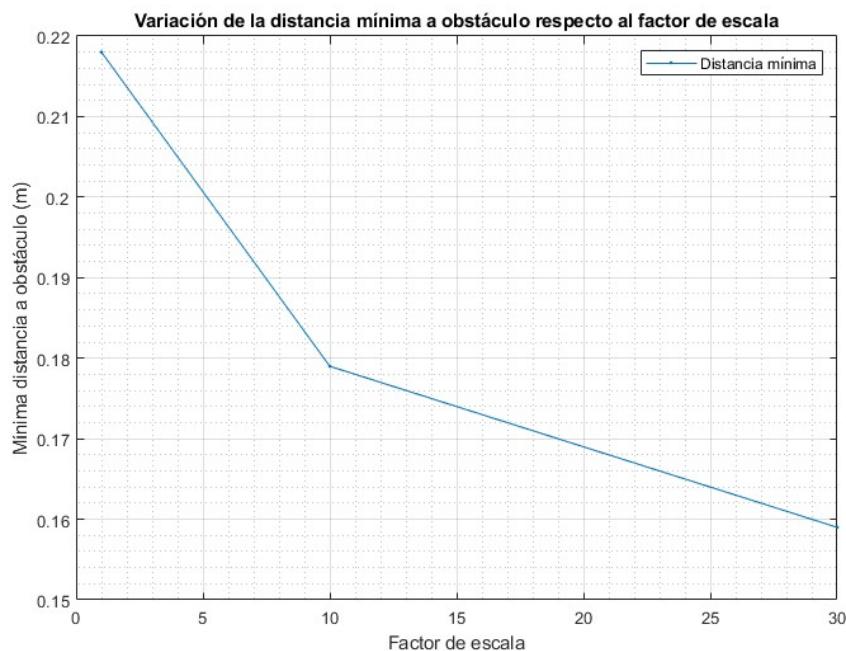


En segundo lugar, trabajando con la resolución de 0.05 m/pixel, se han hecho varios experimentos variando el factor de escala, con un valor inicial de 10. Este factor influye en la inflación de los obstáculos, de modo que a menor factor de escala ésta es más suave (es decir, la inflación se aleja más del obstáculo), y viceversa. Por tanto, se ha tenido en cuenta la distancia mínima a la que se acerca el robot a un obstáculo con los datos del lidar. Además, se puede observar cómo el camino calculado puede variar, recorriendo más distancia, pero asegurándose que el robot no se acerque demasiado a los obstáculos:

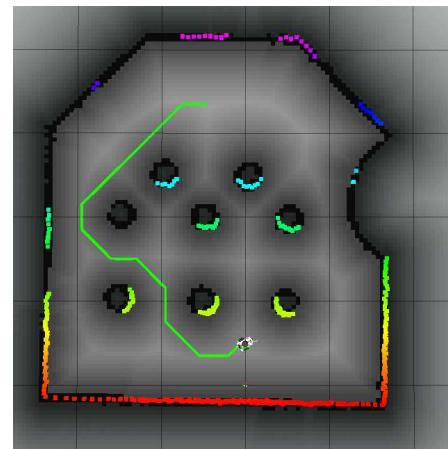




Factor de escala: 10



Finalmente, se ha experimentado con el radio de inflación, inicialmente de 0.5 metros, y aumentándolo considerablemente hasta 10 metros, dejando el factor de escala en 1. Se ha observado cómo el mapa completo tomaba un coste bastante grande, por lo que colocando el *lethal\_cost* a 50, no era capaz de encontrar un camino. Sin embargo, al volver a subir el *lethal\_cost* a 100, ya sí podemos obtener un camino hasta el punto objetivo, a pesar de tener que pasar por zonas con un alto coste.

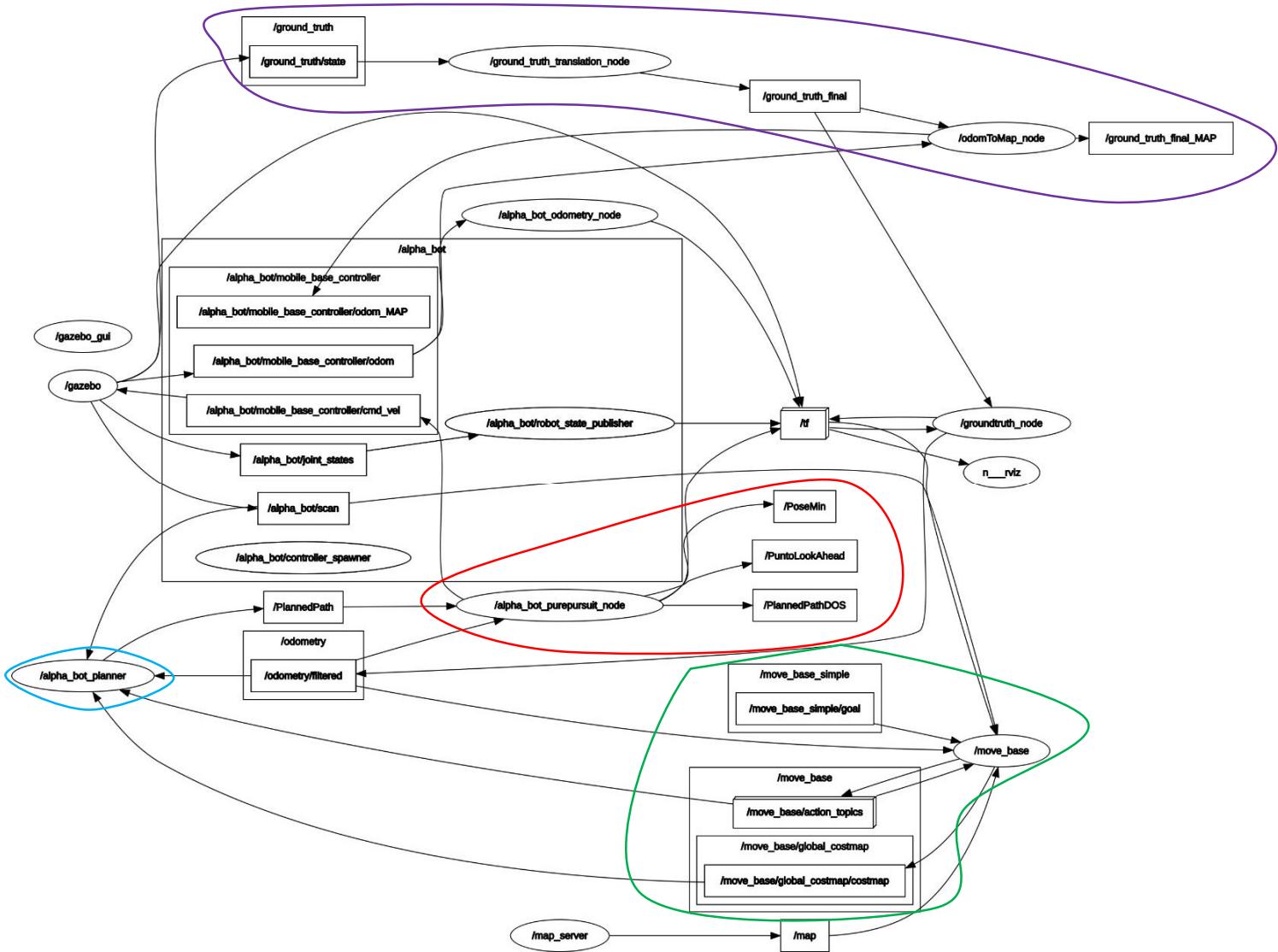


#### 4. SISTEMA COMPLETO

Una vez desarrollados y testeados los distintos bloques que componen el sistema completo, siguiendo la metodología en V planteada, nos encontramos en disposición de llevar a cabo la integración y validación del sistema completo.

Recapitulando, contaremos con distintos métodos de localización estudiados. El planificador de caminos será el desarrollado, basado en algoritmo de Dijkstra. Ambos proporcionarán las entradas de pose actual y conjunto de waypoints del camino planificado al controlador de alto nivel que será la adaptación de la técnica de persecución pura detallada. La salida del controlador de alto nivel hacia el controlador de velocidad de bajo nivel será la velocidad lineal y angular.

El esquema final del sistema resultante de la integración es:



Aquí se pueden ver todos los nodos activos en la ejecución, así como los tópicos utilizados mediante los cuales se transmite la información contenida en mensajes, para así comunicarlos y sincronizar sus funcionamientos.

Tal y como puede verse, gran parte del sistema está hecha por nosotros:

-En **azul** podemos ver como el planificador se implementa en un único nodo, el cuál ejecuta el cálculo de camino una vez reciba una nueva goal por parte de move\_base (el mecanismo de sincronización usado ha sido por callbacks). Tras encontrar camino (es posible que el planificador se dé cuenta de que no es posible, en cuyo caso no devuelve mensaje), se envía el conjunto de puntos al control de alto nivel.

-En **rojo** encontramos el paquete de controlador, el cuál una vez recibe mensajes por el tópico “/PlannedPath” comprueba si se ha recibido una nueva ruta, para actuar en consecuencia reajustando referencias para los cálculos con la nueva información y publicando mensajes hacia el controlador de bajo nivel a través de /cmd\_vel.

-En **morado** vemos los nodos implementados para la adaptación de la información recibida por el plugin añadido en Gazebo.

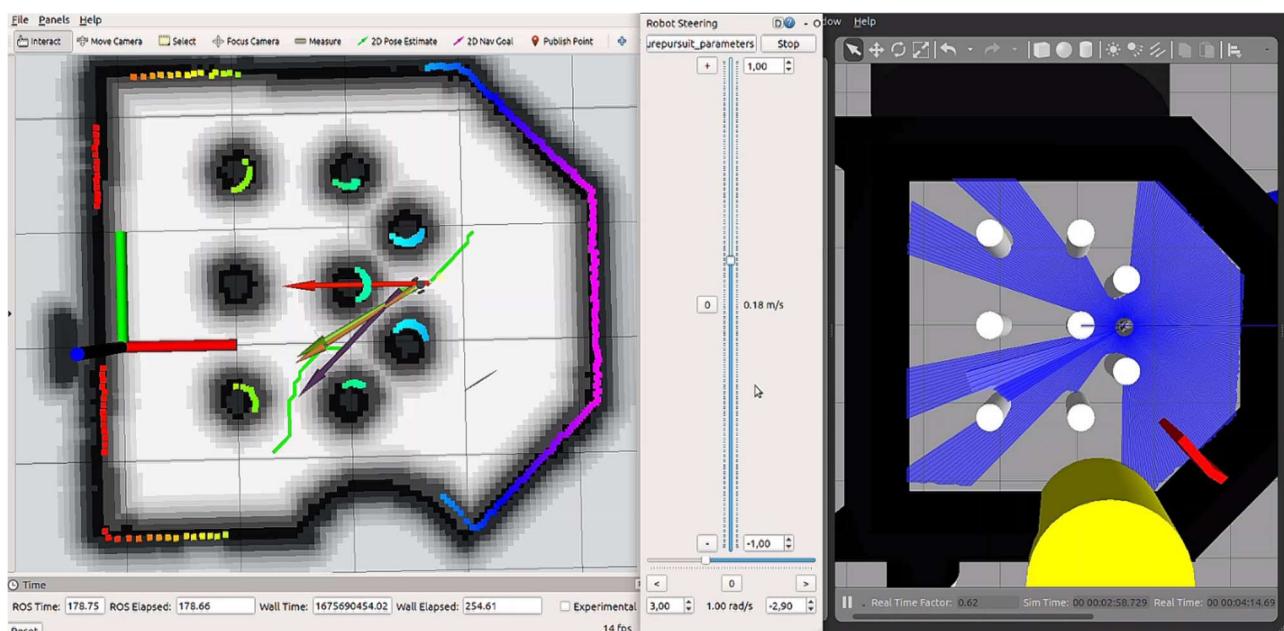
Los paquetes externos que se utilizan son, por tanto: **move\_base** (para aprovechar el plugin de mapa de coste global como entrada del planificador), el servicio de Gazebo (que publica los resultados de aplicar un motor de físicas al modelo urdf del robot) y el **server\_map** (previamente generado con **gmaping**).

#### **4.1. EXPERIMENTOS Y RESULTADOS OBTENIDOS**

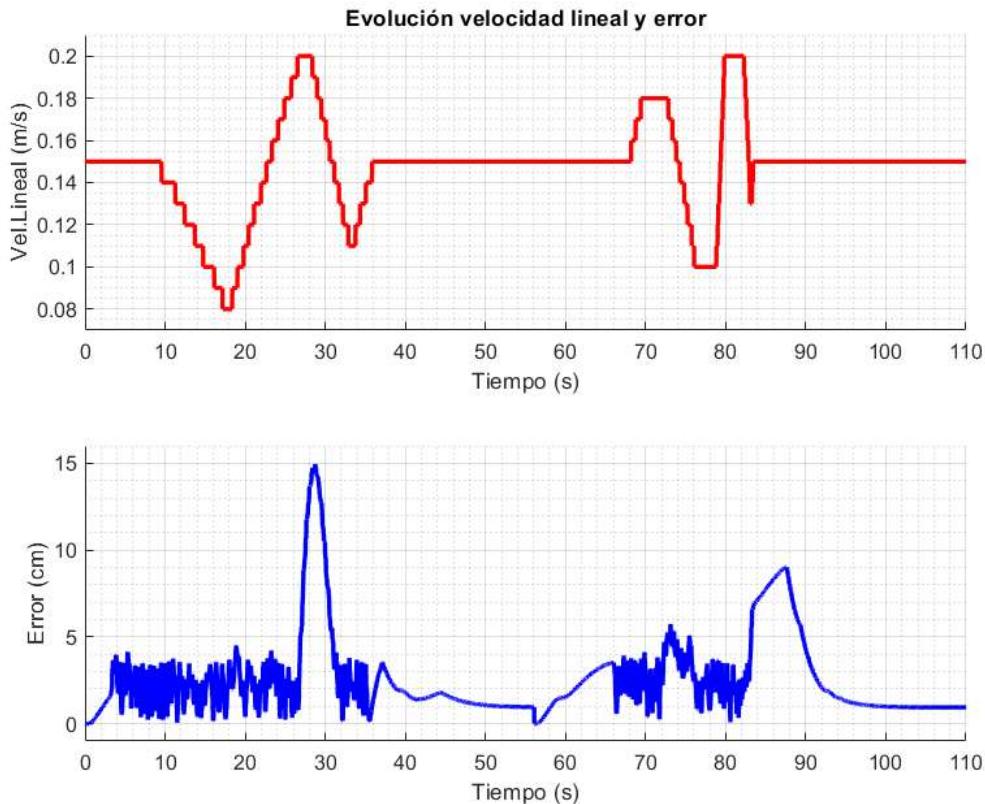
Llevamos a cabo una serie de experimentos con objeto de valorar el funcionamiento completo del proyecto en simulación.

- Cambio velocidad lineal en tiempo de ejecución

Realizamos un experimento consistente en realizar cambios en la velocidad lineal de desplazamiento de salida del controlador con el objetivo de medir la estabilidad y rango de velocidades de operación aceptable en cuanto a desviación del camino planificado. Esto es interesante en cuanto a la ampliación a seguimiento de trayectorias donde la velocidad pasará a ser variable.

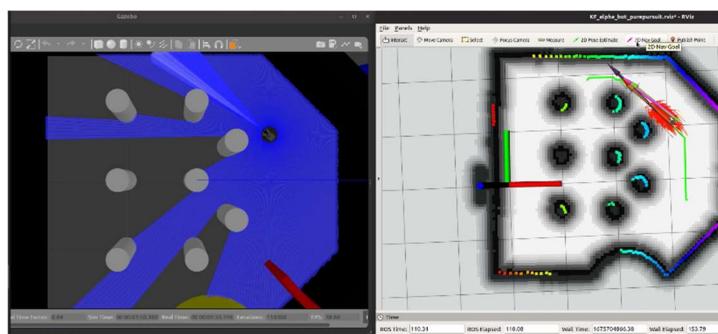


En torno a la velocidad nominal con la que estamos trabajando modificamos por tanto la velocidad comentada y podemos observar como se comete una mayor desviación del camino y por tanto un mayor error. Esto se da con un cierto retraso, el cual corresponde al tiempo que existe entre el momento en el que se realizan los cambios hasta que la desviación respecto al camino que se sigue se hace palpable.



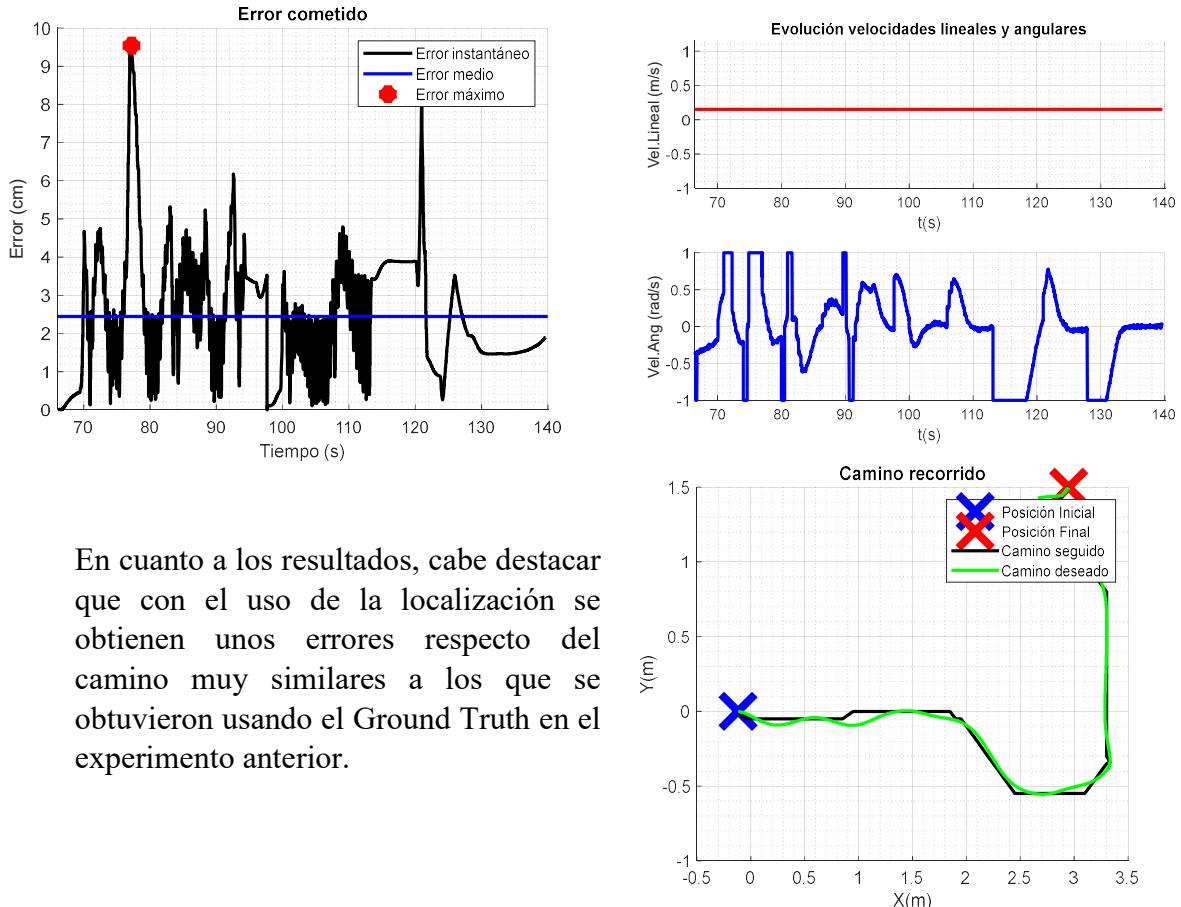
- Integración con tercer método de localización: experimento buscando puntos objetivo con el planificador; parámetros del controlador: velocidad lineal = 0.15 m/s, distancia de Look Ahead = 5 waypoints por delante

Con los parámetros fijos especificados, realizaremos un experimento similar al anterior. Sin embargo, en lugar de usar ahora el Ground Truth como medio de localización, usaremos el tercer método de localización explicado (AMCL separado de un EKF que fusiona todos los métodos de localización incrementales):



Experimento accesible en: <https://www.youtube.com/watch?v=76DFZr4jUll>

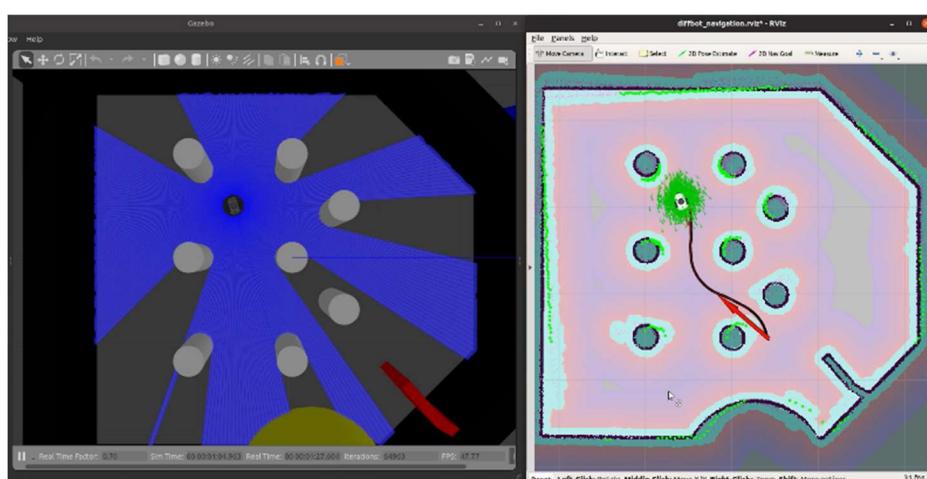
Se adjunta vídeo, pues nos parece interesante comentar que durante el experimento se ha realizado una prueba adicional: antes de que el robot alcanzara su pose objetivo, la hemos cambiado (minuto 1:20). Se puede apreciar que el robot se detiene, se recalcula la trayectoria, y el robot busca ahora su nuevo objetivo.

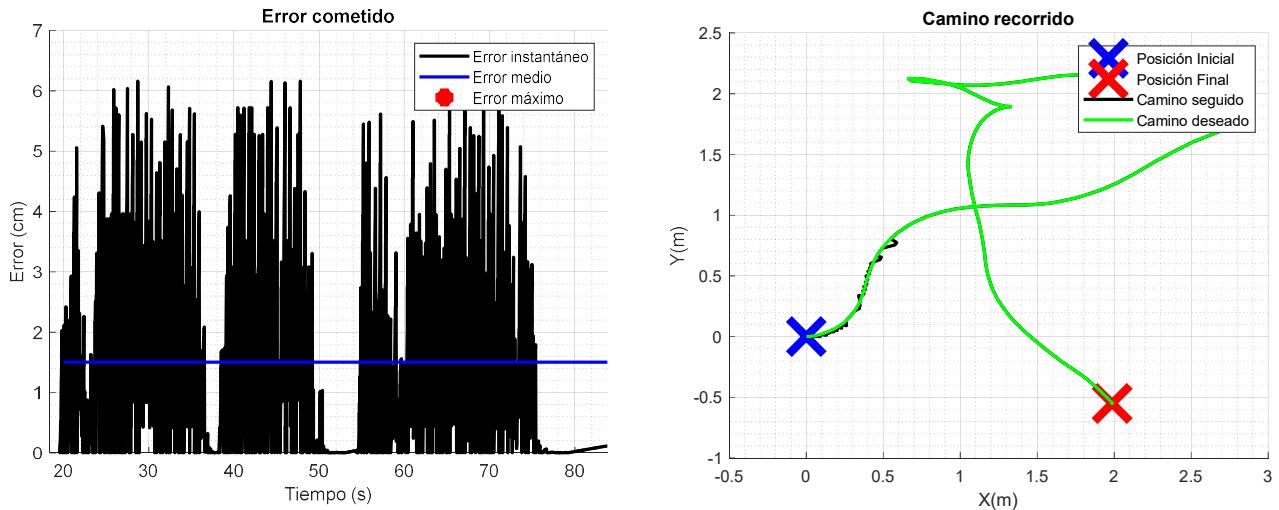


En cuanto a los resultados, cabe destacar que con el uso de la localización se obtienen unos errores respecto del camino muy similares a los que se obtuvieron usando el Ground Truth en el experimento anterior.

- Comparación de errores con el paquete original

Usamos el paquete <https://github.com/ros-mobile-robots/diffbot> original que hemos usado (modificado) en nuestro trabajo, y sometemos a este robot a un experimento similar a los dos anteriores, con el fin de contrastar su precisión en la planificación y seguimiento de caminos respecto al nuestro:





Con el paquete de `move_base` en el `diffbot` original, que aglutina las funciones de planificación y seguimiento de caminos con código en ROS, se consiguen unos errores inferiores respecto a los paquetes desarrollados por nosotros.

También cabe destacar el hecho de que el camino de este paquete se va recalculando a medida que el robot se desplaza (nuestro camino es fijo), por tanto el camino se acomoda a la posición del robot, disminuyendo el error en la simulación.

## 5. AMPLIACIONES FUTURAS

La mejora inmediata que nos planteamos es la ampliación a un planificador de trayectorias.

De cara a una aproximación a esta mejora, nos gustaría conocer como de bueno es el controlador a distintas velocidades de movimiento. De esta forma podríamos saber que restricciones temporales son realistas para nuestro robot el rango de velocidades de operación posibles disponibles para cumplirlas.

En este aspecto, hemos incluido la posibilidad de variación de la velocidad en tiempo de ejecución y comenzado a desarrollar pruebas con idea de satisfacer restricciones temporales, así como forzar al sistema hasta el fallo.

Además, en el apartado correspondiente al controlador de alto nivel hemos visto como se ha realizado experimentos con diferentes velocidades lineales fijas de movimiento.

Por tanto, el siguiente paso sería sustituir dicha velocidad fija por un control de la misma asegurando la llegada del robot en tiempos específicos al punto objetivo final o incluso a puntos intermedios de la trayectoria.

Una cuestión a tener en cuenta es la relación existente entre la velocidad lineal y la distancia de lookahead. Experimentalmente, hemos visto que a mayores velocidades requeridas el comportamiento mejora si se incrementa esta distancia y viceversa. Por ello, habría que tener en cuenta la posibilidad de hacer la distancia variable en función de las velocidades necesarias en cada instante de cara a mejorar el funcionamiento.

Por otro lado, podríamos mejorar la tarea de planificación incluyendo además un mapa de costes local que tenga en cuenta posibles modificaciones del mapa en tiempo de ejecución. De esta forma se podría alcanzar la replanificación para evitación de obstáculos dinámicos.

## 6. REFERENCIAS

- [1] “Diffbot: differential drive robot”: <https://github.com/ros-mobile-robots/diffbot>
- [2] “Wiki ROS”: <http://wiki.ros.org/>
- [3] A. Ollero, “Robótica. Manipuladores y robots móviles” (2001)
- [4] “Algoritmo de Dijkstra”:  
[https://foro.elhacker.net/programacion\\_cc/algoritmo\\_de\\_dijkstra\\_paso\\_a\\_paso-t427371.0.html](https://foro.elhacker.net/programacion_cc/algoritmo_de_dijkstra_paso_a_paso-t427371.0.html)
- [5] “Costmap”: [http://wiki.ros.org/costmap\\_2d](http://wiki.ros.org/costmap_2d)
- [7] “Extended Kalman Filter (EKF)”: <https://automaticaddison.com/extended-kalman-filter-ekf-with-python-code-example/>
- [8] “Odometry for Wheeled Robots”: <https://www.freedomrobotics.com/blog/tuning-odometry-for-wheeled-robots>
- [9] “Build a Custom Robot in ROS | URDF”:  
<https://www.youtube.com/watch?v=JFpg7vG8NxE>
- [10] “Pure-Pursuit-Controller”: <https://es.mathworks.com/help/robotics/ug/pure-pursuit-controller.html>
- [11] “ROS REP of Coordinate Frames”: <https://www.ros.org/reps/rep-0105.html>