# Seguimiento de paredes mediante el algoritmo Split & Merge

Adrián Fernández Llenderrozos y Daniel Castro Gómez

En este documento vamos a comentar de forma detallada el desarrollo que hemos necesitado, para la realización del comportamiento de seguimiento de paredes usando el algoritmo de Split & Merge usando el robot Turtlebot. Por último, vamos comentaremos los resultados que hemos obtenido durante las pruebas y las conclusiones que hemos podido obtener.

## Introducción

Inicialmente vamos a comentar un poco el trabajo que tenemos que realizar para poder obtener el comportamiento que deseamos.

En esta ocasión para poder realizar el comportamiento contamos con el simulador Gazebo, en el cual vamos a simular el robot Turtelbot, para poder hacer el desarrollo más fácil.

El comportamiento que tenemos que desarrollar es un comportamiento, con un robot Turtelbot, para el seguimiento de paredes usando la unión de los datos medidos por un sensor Lidar, que se encuentra instalado en la parte superior del robot, y el algoritmo de Split & Merge. Este algoritmo se encargará de analizar los datos del sensor para obtener las paredes que están el entorno del robot. Después es necesario que el comportamiento elija que pared va a seguir y que pared se encuentra delante del robot.

Vamos a realizar varias pruebas para comprobar si el comportamiento que hemos desarrollado es un comportamiento robusto y ver los resultados que obtenemos. En esta ocasión vamos a realizar pruebas en simulador, pero también vamos a realizar pruebas en el robot real, de esta manera vamos a comprobar la transferencia del comportamiento al mundo real

A continuación, vamos a comentar alguna de las limitaciones que podemos encontrar a la hora del desarrollo del comportamiento y que es necesario solucionar para poder obtener el resultado que deseamos.

- Lectura de las medidas del sensor

Es necesario leer los datos medidos por el sensor en cada instante para la realización de esto vamos a usar las posibilidades que nos proporciona ROS.

- Conversión de coordenadas polares a coordenadas cartesianas

El resultado que nos proporciona el sensor es la distancia medidas cada un cierto incremento de ángulo hasta completar los 360 grados. Se pueden correlacionar con las coordenadas polares medias desde el robot, como posición 0,0 del sistema. Para poder usar esta información, en funcionalidades siguientes, es necesario convertirlas en coordenadas cartesianas.

- Algoritmo Split & Merge

Una de las limitaciones que tenemos es el desarrollo del algoritmo Split & Merge. Este algoritmo se encarga de analizar las coordenadas cartesianas, para la generación de diversos segmentos, los cuales describen la forma del entorno en el que se encuentra el robot.

- Selección de segmentos

Después de la generación de los diferentes segmentos es necesario elegir dos segmentos uno será el segmento el cual el robot va a seguir y el otro segmento será un segmento que se encuentra en posición vertical en frente del robot con el cual robot se puede cochar en el avance.

- Cálculo de velocidades

Para poder seguir las paredes es necesario calcular la velocidad lineal y la velocidad angular que es necesario para cada una de las situaciones.

# Desarrollo

En este apartado vamos a comentar la implementación que hemos desarrollado para poder realizar el comportamiento que deseamos buscar. Como hemos comentado contamos con diferentes limitaciones, por lo cual este apartado lo vamos a dividir en esas diferentes limitaciones.

**Lectura de las medidas del sensor**

Para obtener estos datos vamos a suscribirnos al topic que proporciona el sensor Lidar. El mensaje que tenemos es un mensaje de tipo *sensor_msgs/LaserScan* el cual cuanta con varios parámetros, pero para esta ocasión es importante los valores Range, los cuales nos indican la distancia medida por cada uno de los incrementos del ángulo.

**Conversión de coordenadas polares a coordenadas cartesianas**

Como comentamos, las medias realizadas por el sensor laser se pueden relacionar con coordenadas polares, donde el origen es el robot. Para el uso de esta información en funcionalidades siguientes es necesario transformarlos en coordenadas cartesianas, para la realización de esto vamos a transformar cada una de las coordenadas con la siguiente formula.

$$x = DistanciaMedida * \cos(\theta)$$

$$y = DistanciaMedida * sen\ (\theta)$$

$$Cordenada = [x\ , y\ ]$$

El ángulo se incrementará $\Delta\theta$ desde 0 hacia 2π, el valor del incremento depende del sensor el cual está instalado en el robot. Por lo cual es necesario configurarlos para cada robot en el que usemos la función.

**Split & Merge**

Esta función es una de las más importantes para la realización del comportamiento, este algoritmo es encargado de segmentar las coordenadas medias para describir el entorno que lo rodea. Este método se diferencia principalmente en tres funciones principales y en varias funcionalidades complementarias, para obtener distancias entre un segmento y un punto, el ángulo entre dos segmentos o plotear los segmentos generados.

Ahora vamos o comentar de forma matemática las funciones que realizamos en cada una de las funciones del algoritmo.

- **Cálculo de la distancia**

Para calcular la distancia entre un segmento y un punto. Primeramente, es necesario calcular la proyección del punto en el segmento que

1. Vector director del segmento y vector desde el punto : $v = (x_1 - x_0\ , y_1 - y_0)$
2. Vector desde el punto inicial del segmento al punto : $w = (\ x_q - x_0\ , y_q - y_0\ )$
3. Proyección del punto externo sobre el segmento : $proy = \frac{v*w}{\|v\|^2} = \frac{v*w}{(x_1 - x_0)^2 + (y_1 - y_0)^2}$
4. Punto proyectado sobre el segmento : $p = p_0 + proy * v = = (x_0 + proy * (x_1 - x_0)\ , y_0 + proy * (y_1 - y_0)$

Después se calcula la distancia entre el punto y la proyección, $d = \sqrt{(x_q - x_p)^2 + (y_q - y_p)^2}$

- **Cálculo del ángulo entre segmentos**

Para calcular el ángulo que forman dos segmentos entre si cuando se cortan se realiza con los vectores directores de ambos segmentos con la siguiente formula: $\cos(\theta) = \frac{v_1 * v_2}{\|v_1\| \|v_2\|}$

- **Split**

La función Split se encarga de dividir el conjunto de puntos en diferentes segmentos utilizando el criterio de la distancia máxima de los puntos al segmento actual. Si dicha distancia ($dist_{max}$) supera un umbral ($d_{th}$), el segmento se subdivide. Esto se gestiona mediante una pila que almacena los índices inicial y final de cada

segmento pendiente de procesar. Al finalizar, la función devuelve una lista de segmentos que cumplen los criterios.

- **Merge**

La función merge une segmentos consecutivos si el ángulo entre ellos ($\theta$) es menor que un umbral ($\theta$). Si no hay segmentos previos, se añade directamente a la lista final. Si $\theta < a_{th}$, se fusionan en un nuevo segmento; en caso contrario, el segmento actual se agrega como independiente. Devuelve la lista de segmentos unidos que cumplen este criterio.

- **Purge**

La función purge elimina los segmentos que no cumplen con ciertos criterios mínimos de calidad en términos de número de puntos o en longitud.

La función recorre la lista de segmento que se crearon comprobando si cada uno cumple ambos criterios, el resultado final que obtenemos es la lista de segmentos depuradas compuesta por aquellos que cumples con los umbrales definidos

**Selección de segmentos**
Estas funciones seleccionan dos tipos de segmentos diferentes de la lista de segmentos calculados.

- Segmento para seguir:

En esta ocasión buscamos un segmento el cual se encuentra a la derecha del robot, traduciendo a coordenadas, estamos buscando un segmento el cual cuente con la mayoría de las coordenadas y negativas. Para hacer filtrarlo, primeramente, calculamos las ecuaciones de la recta al cual pertenece el segmento y después seleccionamos los segmentos los cuales el corte con el eje y sea negativo y además que sea un segmento horizontal. Después de este filtro seleccionamos el segmente con menor distancia al robot

- Segmento enfrente:

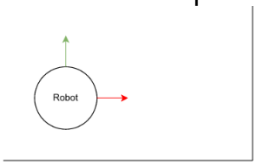
En esta ocasión buscamos el segmento el cual se encuentra perpendicular al avance del robot. Traduciendo esto se corresponde a un vector vertical y que se encuentra en la parte positiva del eje X. Dentro de los segmentos que cumples estas condiciones se selecciona el segmento más cercano respecto al robot.

**Cálculo de velocidades**
El último paso que tenemos que realizar es el cálculo de las velocidades necesarias para realizar el comportamiento que deseamos a partir de los datos que hemos calculado. Para la realización hemos definido una serie de estados dependiendo de diferentes criterios.

En esta ocasión los estados que hemos definido son los siguientes:

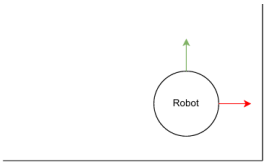
- Estado 0: Cuando el robot se encuentra siguiendo una pared

En esta ocasión el robot detecta segmento a seguir y además si no encontramos un segmento enfrente o la distancia a la pared es menor a una distancia definida. Como podemos ver en el siguiente diagrama.




- Estado 1: Cuando se encuentra una pared enfrente

En esta ocasión el robot sigue teniendo una pared a la derecha, pero encuentra una pared en frente a una distancia menor que un umbral



- Estado 2: El robot deja de detectar la pared

En esta ocasión el robot deja de detectar la pared a la cual estaba siguiente anteriormente.


A continuación, vamos a comentar las velocidades que son necesarias en cada uno de los estados que hemos comentado anteriormente.

- Velocidades para Estado 0

Como hemos visto en este estado queremos seguir la pared que se encuentra a la derecha del robot a una distancia de seguridad para no colisionar con ella. Para esto hemos definido las siguientes velocidades.

Hemos decidido que la velocidad lineal sea constante, en cambio, para la velocidad angular hemos implementado una combinación de controladores PD para mantener la distancia a la pared y para que el robot se mantenga con orientación 0 respecto al segmento.

La fórmula que describe la velocidad angular es la siguiente:

$$w = k * PD(Distancia) + (1 - k\ )PD(Angulo)$$

La variable k pondera la importancia que tiene la velocidad para mantenerse a la distancia o la velocidad para alinear el robot con el segmento.

- Movimiento en el estado 1

Como hemos visto en este estado queremos esquivar la pared que se encuentra en frente, por lo cual para eso es necesario que giremos hacia la izquierda, Para esto hemos definido las siguientes velocidades.

Hemos decidido que la velocidad varie conforme a la distancia que está de la pared de enfrente para que sea proporcional a la distancia y facilite el giro sin chocarse. Se calcula con la siguiente ecuación $v = kv * DistanciaX$. En cambio, la velocidad angular será constate positiva para que pueda girar hacia la dirección necesaria

- Movimiento en el estado 2

Como hemos observado en este estado, el objetivo es girar hacia la derecha debido a que el robot ha superado la pared. Las dos posibles situaciones son las siguientes:

1. Esquina exterior: El robot se encuentra en una esquina, pero por el borde exterior. En este caso, el robot debe girar a la derecha para continuar siguiendo la otra pared que forma la esquina.
2. Pared única: Ocurre cuando el robot sigue una pared que termina sin conectarse con ninguna otra. Aquí, el robot debe realizar un giro de casi 180 grados hacia la derecha para poder continuar siguiendo la misma pared, pero por el otro lado.

En este caso, hemos decidido que ambas velocidades sean constantes. La velocidad angular será negativa para permitir que el robot gire en la dirección correcta

## Resultados

En este apartado vamos a comentar los resultados que hemos obtenido durante las pruebas que hemos realizado, para la validación del comportamiento que hemos desarrollado. Primeramente, comentaremos los resultados que obtuvimos en simulación y por últimos comentaremos los resultados que hemos obtenido con el robot real.

**Simulación**

Las primeras pruebas que hemos realizado es un mapa sencillo, el cual está compuesto por un cuadrado con el cual podemos obtener los tres estados diferentes si iniciamos el robot en posiciones distintas. Los

resultados que hemos obtenido los podemos ver en los siguientes videos: Video recorriendo el mapa por dentro , Video recorriendo el mapa por fuera

Como podemos ver en los videos el robot consigue recorrer los circuitos correctamente, pero como podemos ver cuando el robot en ambos casos se encuentra con únicamente una pared a la derecha a al cual va a seguir, correspondiente al estado 0, el robot es capaz de avanzar de forma estabilizada a una distancia de la pared. Cuando se encuentra cono una pared en frente a una distancia menor, estado 1, vemos que el robot es capaz de girar hacia la derecha para volver a encontrarse en el estado 0. También cuando deja de detectar la pared a la que estaba siguiendo, estado 2, el robot es capaz de girar a la derecha para seguir recorriendo el mapa.

Pero durante las pruebas que hemos realizado hemos encontrado varios errores, los cuales los podemos ver en los siguientes videos: Video error por dentro , Video error por fuera

- El error por dentro se produce debido al comportamiento del PD ya que al intentar estabilizar el error para ir a la distancia objetivo choque contra la pared cercana lo que da el error. Este error también se puede producir cuando el robot entra en el estado desde una lejana distancia que al compensar tanto se desvía contra la pared.
- El error por fuera se produce cuando el robot nos está bien centrado en la distancia objetivo, es decir, si el robot lleva tiempo siguiendo la pared y se posiciona en una distancia a la pared cercana a la que buscamos hace el giro perfectamente, pero si por alguna razón no se aproxima el giro puede llegar a fallar y chocar con la pared.

Ambos errores dependen parcialmente de la configuración del controlador PD. Por ello, se ajustan los valores para minimizar las oscilaciones, es decir que el robot no se aleje demasiado de la distancia objetivo, pare intentando maximizar el tiempo de asentamiento sea el menor posible para que se pueda estabilizar en pequeñas distancias.

La otra prueba que vamos a realizar va a ser una prueba con un mapa más complejo para ver cómo se comporta el robot en esta ocasión. El comportamiento que el robot ha realizado lo podemos ver en el siguiente video : Video mapa 2

Como podemos ver en el video el robot a sido capaz de conseguir recorrer todo el mapa pero como podemos ver la trayectoria que recorre el robot no esta tan refinada como en el caso anterior, debido que en cuando nos encontramos en el estado 0, siguiendo una pared, no contamos con el tiempo suficiente para poder estabilizar la distancia, como consecuencia cuando giramos en las esquinas, los giros no son tan buenos debido que en el comportamiento que hemos definido las velocidades no son adaptativas a la información que tenemos medidas. Esto lo que implica que en un número de veces el robot no es capaz de recorrer todo el mapa debido que puede chocarse con la pared.

Otra cosa que podemos hacer es que el robot es capaz de recorrer un mapa en el cual cuenta con paredes oblicuas, esto es debido que lo detecta como una pared delante entonces decide girar y seguir a la pared inclinada , debido que no es necesario que la pared de enfrente sea completamente inclinada. Si que va tenemos un cierto un cierto nivel de inclinación máximo, debido que si superamos el valor máximo el filtro no la detectara.

**Robot real**

Para la implementación de este algoritmo en el Turtelbot real, primeramente, tuvimos que adaptar los valores intrínsecos del algoritmo Split&Merge, que corresponde con el funcionamiento del algoritmo y además los que corresponde con el sensor laser, debido que son diferentes al usado en simulación. Ademas nos fue necesario adaptar el funcionamiento para que funcionara en dicho robot, debido que es necesario procesar los datos en local a causa de incompatibilidades de versión y únicamente es obligatorio enviarle al robot los valores de velocidad necesarias en cada caso.

En esta ocasión las prueba que hemos realizado son similares a las realizadas en simulación para poder comprobar cómo se transfiere el comportamiento al mundo real. Los resultados que hemos obtenido se pueden ver en los siguientes videos, en los cuales ponemos el robot en diversas situaciones para comprobar como sigue la pared. Video1 , Video2

Como podemos ver en los videos los resultados que hemos obtenido son muy similares a los del simulador. Donde el robot no calcula correctamente la trayectoria óptima para cada una de las situaciones. Ademas, vemos que el entorno en los que hemos hechos las pruebas no sencillo debido que cuenta con números obstáculos que dificultan el cálculo del algoritmo y además cuenta con paredes pequeñas que como hemos visto en simulación que en el estado de seguir pared le cuesta estabilizarse a la distancia deseada.

Pero hemos demostrado que el uso del comportamiento que hemos diseñado en simulación se puede usar en el mundo real, con la adaptación de varios parámetros propios del robot y además es necesario adaptar al funcionamiento propio del robot que estamos usando.

# Conclusiones

Las conclusiones que hemos podido obtener después del desarrollo del comportamiento y de las diversas pruebas que hemos realizado tanto en simulación, para la validación del comportamiento, como haciendo las pruebas en el robot real son las siguientes.

A la hora del desarrollo del comportamiento hemos visto que es bastante sencillo debido que, a partir de los datos medidos por el sensor, podemos realizar un mapeado del entorno que rodea el robot y que acompañado del algoritmo Split&Merge, nos permite discretizar el entorno para solo tener en cuanta los datos que necesitamos.

El control del robot es más complejo debido que es necesario que las velocidades sean las correctas para cada uno de los casos para la que la trayectoria calculada sea la óptima. Para realizar esto es necesario implantar un controlador para cada una de las situaciones, que en esta ocasión no lo hemos realizado debido al aumento de complejidad que tenemos.

En respecto a los resultados que hemos obtenido son lo que esperábamos debido que al no contar con el controlador como comentábamos, es complicado que la trayectoria sea la correcta en cada uno de los casos. Pero a pesar de eso hemos obtenido unos resultados aprovechables para la validación del comportamiento.

A la hora del transferir el comportamiento al mundo real, a pesar de que inicialmente es necesario un ajuste propio a la plataforma que estamos usando en cada caso, el comportamiento principal obtiene resultados similares a los simulados.

Como se ha observado en este archivo, este proyecto es un prototipo que busca explorar el comportamiento del sistema al ser implementado en un entorno real. Sin embargo, existen diversas oportunidades para mejorar su rendimiento mediante la incorporación de nuevas tecnologías. Algunas de las mejoras propuestas son las siguientes:

- Selección de los segmentos más eficientes

Actualmente, la selección de segmentos se basa en las posiciones de los puntos extremos. Se podrían integrar nuevas funcionalidades que optimicen este proceso, logrando así una segmentación más precisa y eficiente.

- Implementación de un controlador para cada estado

Para mejorar la trayectoria generada en cada situación, es esencial el uso de un controlador. Se podría diseñar un sistema que ajuste las velocidades de manera matemática para cada caso específico o emplear técnicas de aprendizaje automático para adaptarse dinámicamente a las condiciones del entorno.

- Uso de estimadores de posición

Aunque en este prototipo no es necesario debido a la baja carga computacional, si el algoritmo Split & Merge requiriera más tiempo para calcular los segmentos, sería útil estimar la posición del robot en función de la distancia recorrida por las ruedas. Esto se podría lograr mediante el uso de algoritmos como los filtros de Kalman, permitiendo un comportamiento más fluido y continuo.

Las conclusiones finales que hemos obtenido es que el comportamiento que hemos realizado es bueno, existen elementos los cuales se pueden mejorar, pero aumentaría la complejidad de este trabajo que en esta ocasión no lo hemos visto oportuno.