Imagen que contiene Logotipo

Descripción generada automáticamente![Logotipo, nombre de la empresa

Descripción generada automáticamente]()

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES

MÁSTER EN AUTOMÁTICA Y ROBÓTICA

GUIADO Y NAVEGACIÓN DE ROBOTS

TRABAJO FIN DE ASIGNATURA

Autores:

Álvaro Benito Oliva

Germán Andrés Di Fonzo Caturegli

Juan José Jurado Camino

Tutores:

Fernando Matía

Miguel Hernando

Paloma de la Puente

Madrid, \_\_ de \_\_\_\_\_ de 2020

Tabla de contenido

[1 Introducción, objetivos y reparto de roles 3](#_Toc57811829)

[2 Planteamiento del problema 3](#_Toc57811830)

[3 Incertidumbre del sistema de locomoción 3](#_Toc57811831)

[4 Sistema de percepción 3](#_Toc57811832)

[5 Algoritmo de localización 3](#_Toc57811833)

[6 Algoritmos de control 3](#_Toc57811834)

[7 Algoritmo de planificación de trayectorias 3](#_Toc57811835)

[8 Demostrador final 3](#_Toc57811836)

[9 Conclusiones y observaciones 3](#_Toc57811837)

# Introducción, objetivos y reparto de roles

# Planteamiento del problema

# Sistema de locomoción

# Sistema de percepción

# Estimación del estado

# Control

## Algoritmos de seguimiento de trayectorias

Para llevar a cabo el movimiento del robot a lo largo de una trayectoria generada por un planificador, es necesario programar un controlador que vaya siguiendo los distintos puntos de la trayectoria. El sistema de control utilizado se denomina “*Pure Pursuit Algorithm*” y está implementado el Matlab, lo cual facilita su implementación en el trabajo.

Sin entrar en mucho detalle, el algoritmo mencionado consta de las siguientes etapas:

1. En primer lugar, se obtiene la pose del vehículo a través del algoritmo de localización que se ha programado previamente.
2. A continuación, se busca el punto en la trayectoria que se desea seguir, que se encuentra a una distancia determinada “*look-ahead distance*”. La determinación de este punto se hace a partir de cálculos geométricos.
3. Posteriormente, se calcula la velocidad de giro necesaria para llegar al punto deseado, finalizando la iteración.

Dado que se debe especificar una velocidad de avance constante, el algoritmo de control trabaja únicamente con la velocidad de giro. De esta manera, existen dos parámetros que se pueden modificar para mejorar el comportamiento del controlador: la distancia “*look-ahead*” y la velocidad de avance.

## Implementación del algoritmo pure pursuit

La programación de este algoritmo es sencilla ya que existe una clase en Matlab denominada “*controllerPurePursuit*” que contiene los métodos para la estimación de la velocidad angular que debe tener el robot.

Los parámetros que hay que definir para inicializar el controlador son: la trayectoria, la velocidad angular máxima, la velocidad de avance del robot y la distancia “*look-ahead*”. Además, hay que especificar el error máximo permitido, ya que el algoritmo programado no llega al punto deseado, sino que se queda en una zona del entorno.

Se ha generado una trayectoria sinusoidal que debe seguir el robot y se ha comprobado el funcionamiento del algoritmo variando los parámetros de velocidad lineal y distancia look-ahead, obteniéndose las trayectorias mostradas en las **FIGURAS**

Gráfico

Descripción generada automáticamenteGráfico

Descripción generada automáticamenteHistograma

Descripción generada automáticamente con confianza mediaGráfico

Descripción generada automáticamente

*Trayectorias simuladas para distintos parámetros del controlador*

Como se puede observar, la elección de los valores del controlador es esencial para su correcto funcionamiento. Por otro lado, si se escoge una velocidad de avance elevada, el control se dificulta a la hora de realizar giros bruscos, por lo que es conveniente mantener una velocidad de avance limitada.

## Arquitectura de control reactiva

# Planificación de trayectorias

## Selección del algoritmo de planificación

Para poder llegar desde un punto inicial hasta un punto final sin chocar con ningún obstáculo en el camino, es necesario generar una serie de puntos en el entorno que determinen la trayectoria que debe seguir el robot para lograr su objetivo.

Existen múltiples algoritmos para obtener los puntos deseados, entre los cuales destacan los planificadores discretos, los planificadores basados en combinatoria y los planificadores basados en muestreo. Entre todas las opciones, se ha escogido un algoritmo RRT que construye árboles de exploración para encontrar la trayectoria deseada.

El método RRT está basado en muestreo y está incluido en Matlab, facilitando su implementación para el trabajo. Además, presenta la gran ventaja de trabajar en el espacio de acciones en vez de en el espacio de estados, manteniendo las restricciones cinemático-dinámicas (kinodynamics) del robot no holonómico con el que se trabaja. Esto último asegura que los puntos generados por la trayectoria se pueden alcanzar por el robot móvil.

Sin embargo, este algoritmo presenta el problema de que no se tiene la orientación inicial del robot, pudiendo dar lugar a un inicio de la trayectoria no realizable desde la pose inicial. Este problema se puede solucionar desde la etapa de control, pero sería deseable que el planificador tuviera en cuenta esta situación y pudiera planificar una trayectoria posible para cualquier orientación inicial.

## Implementación del algoritmo RRT

Para llevar a cabo la planificación, en primer lugar se debe importar el mapa del entorno de trabajo, que se ha obtenido a partir de la planta del restaurante. A continuación, dado que el robot tiene una anchura determinada y el planificador trabaja con puntos discretos, es necesario ensanchar los obstáculos para evitar que el robot colisione con ellos. Este ensanchamiento toma como valor la mitad del ancho del robot. Después de esto, se genera un mapa de ocupación de celdillas y se escala para coincidir con las dimensiones del entorno.

Una vez establecido el mapa, se genera un espacio de estados [x, y, θ] utilizando la función de Matlab “*stateSpaceSE2*”. Dicho espacio de estados contiene la física involucrada en el movimiento del robot basándose en distancias euclídeas e interpolaciones lineales para calcular traslaciones y rotaciones.

Por otro lado, hay que establecer un elemento denominado “*validador*” que se encarga de detectar las posibles colisiones que se produzcan durante la generación de la trayectoria. Este objeto comprueba la validez de cada estado generado por el planificador con una resolución determinada por el usuario.

Tras esto, se crea el planificador a partir del espacio de estados y del validador. Además, se debe establecer la distancia máxima que tendrán las aristas de los árboles generados y el error permitido en el punto final planificado respecto al punto final deseado.

Finalmente, se ejecuta el planificador y se obtiene la trayectoria de puntos que llevan al robot móvil desde la posición inicial hasta la posición final.

## Resultados de la planificación

Tras un proceso iterativo de ajuste, se ha obtenido los parámetros mostrados en la **TABLA**

|  |  |
| --- | --- |
| Dimensiones del mapa de ocupación | [0, 0] – [16, 16] m |
| Ancho del robot | 0.497 m |
| Distancia de detección de colisión | 0.001 m |
| Máximo tamaño de arista del árbol de búsqueda | 0.2 m |
| Error máximo permitido (Distancia entre el punto final planificado y el deseado) | 0.02 m |

Pese a que el algoritmo necesita un menor tiempo de ejecución si se escogen valores mayores de tamaño máximo de arista del árbol, se ha decidido elegir un valor de 0.2 m para reducir la posibilidad de que el robot quede atrapado debido a sus restricciones cinemático-dinámicas.

Para comprobar el funcionamiento del planificador, se han generado varias trayectorias a partir de distintos puntos de inicio y de destino. Sin embargo, como se ha explicado en el apartado de control, el sistema de referencia del mapa de ocupación y el del entorno en Apolo no coinciden, por lo que hay que llevar a cabo las transformaciones de coordenadas necesarias con las funciones “*apolo2map*” y “*map2apolo*”.

Los pruebas realizadas para los parámetros escogidos se muestran en la **TABLA**, en la cual se han representado todas las coordenadas en el sistema de referencia del mapa para una mejor visualización.

|  |  |  |  |
| --- | --- | --- | --- |
| Nº Prueba | Pose Inicial [m, m, rad] | Punto final deseado [m, m] | Tiempo de ejecución (s) |
| 1 | [8, 8, pi/2] | [5 10] | 0.04 – 0.07 |
| 2 | [10.7, 0.5, 0] | [5 10] | 0.05 – 0.12 |
| 3 | [10.7, 15.5, 0] | [10.7, 0.5] | 0.13 – 0.19 |

La representación gráfica de las trayectorias planificadas así como de los árboles de búsqueda resultantes del algoritmo RRT se pueden observar en las **FIGURAS**.

Imagen que contiene Forma

Descripción generada automáticamenteImagen que contiene Forma

Descripción generada automáticamente

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

*Trayectorias generadas para las pruebas 1,2 y 3 de la* ***TABLA***

# Demostrador final

# Conclusiones y observaciones