Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Puebla

Implementación de robótica inteligente *(gpo 502)*

Evaluación 7.2 Planeación de trayectorias

Jhonatan Yael Martínez Vargas – A01734193

Dr. Alfredo García Suárez

Fecha: 17 mayo 2023

Como parte de esta evidencia se tenia que diseñar una estrategia de planificación de trayectorias para un robot móvil, por lo cual para poder hacerlo decidí basarme en el código proporcionado por profesor en una de las sesiones previas, el código que se utiliza para resolver esta evaluación fue seleccionado por que desde mi punto de vista es el más completo, ya que toma en cuenta los parámetros físicos del robot, cuenta con un sistema de control integrado y además simula el uso de un sensor lidar, lo cual hace mucho más preciso el control de movimiento del robot en el entorno.

A continuación, se explican las partes que componen al código, comenzando por la primera parte la cual estable los parámetros fiscos del robot.

```
%% Define Vehicle
R = 0.05;                  % Wheel radius [m]
L = 0.18;                  % Wheelbase [m]
dd = DifferentialDrive(R,L);
```

Figura 01 – Parámetros físicos del robot móvil

Y posteriormente los parámetros necesarios para la simulación

```
%% Simulation parameters
sampleTime = 0.025;              % Sample time [s]
tVec = 0:sampleTime: 162;        % Time array

initPose = [-0.6725402329641; 1.7308359108713; 0];            %
pose = zeros(3, numel(tVec));    % Pose matrix
pose(:,1) = initPose;

% Define waypoints
waypoints = [-0.6725402329641, 1.7308359108713; -0.6308178475905,

% Create visualizer
viz = Visualizer2D;
viz.hasWaypoints = true;
```

Figura 02 - Parámetros de la simulación

Observando la figura 02 se pude ver que existe un vector llamado “waypoints”, este vector tiene la función de almacenar las coordenadas (x , y) que va tener que seguir el robot como parte de su trayectoria.

Para esta evaluación se pido que la trayectoria a seguir fuera una representación de nuestro propio rostro, por lo que el proceso para extraer cada uno de los puntos fue el siguiente:

Figura 03 – Imagen base

Tomando como referencia la figura 03, se posicionaron puntos de manera manual sobre los detalles mas descados del rostro, haciendo uso del software en línea de geogebra, obteniendo así los siguientes resultados.

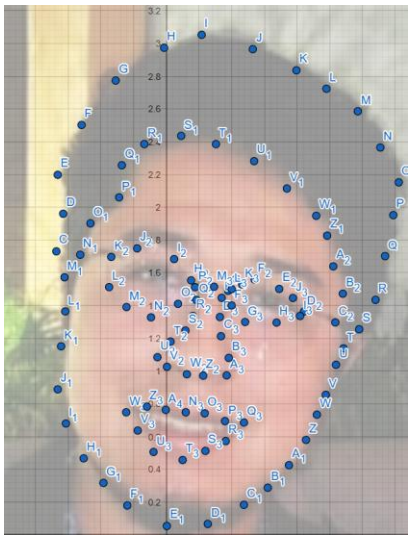

Figura 04 – Puntos destacados

Posteriormente lo que se hizo fue agrupar todos estos puntos en una lista, exportar sus valores numéricos y finalmente limpiarlos para darles el formato necesario para poder ser ingresados en Matlab, dando como resultado el contenido del vector "waypoints" como se mencionó anteriormente.

Finalmente, el código contiene el siguiente apartado

```
%% Pure Pursuit Controller
controller = controllerPurePursuit;
controller.Waypoints = waypoints;
controller.LookaheadDistance = 0.1;
controller.DesiredLinearVelocity = .10;
controller.MaxAngularVelocity = 2.0;
```

Figura 05 – Parámetros del controlador

Para poder hacer que el robot siga los puntos definidos previamente ajustaron los valores:

- “LookaheadDistance” el cual establece el rango de visión del sensor lidar.
- “DesiredLinearVelocity” el cual define la velocidad lineal máxima.
- “MaxAngularVelocity” el cual se encarga de indicar cual es la máxima velocidad angular.

Lo que se puede observar es que los primeros dos valores son muy pequeños, esto se debe a que la distancia entre los puntos es corta, por lo cual tener valores grandes en esto parámetros implicaría un sistema mas inestable al momento de querer seguir los puntos.

En el caso de la velocidad angular, esta se define con un valor mayor a los parámetros previos, esto con la finalidad de que las vueltas sen precisas y cortas, generando así la apariencia de crear bordes más limpios, obteniendo a si los siguientes resultados.

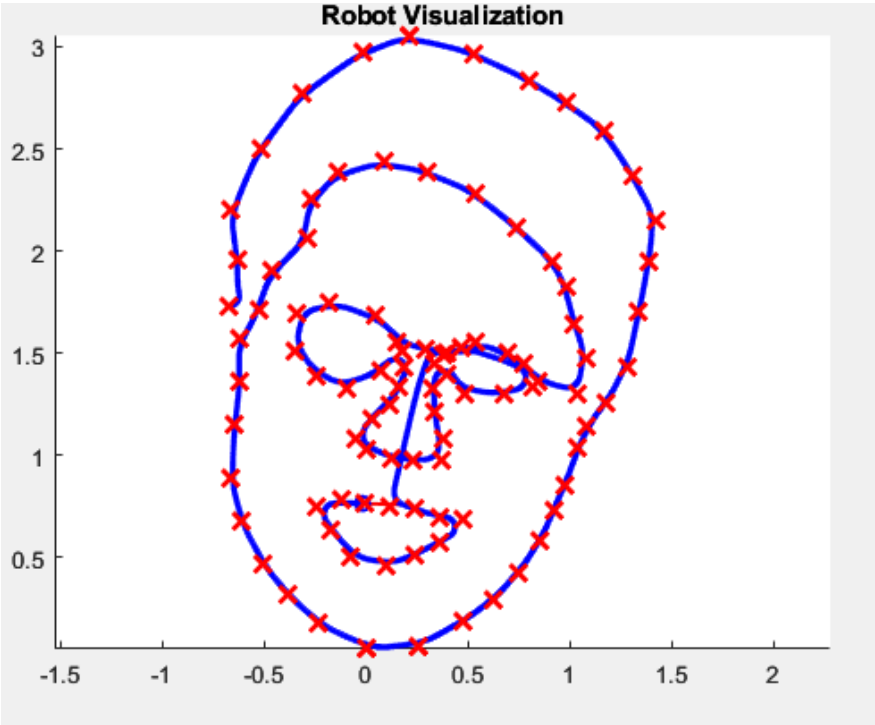

Figura 06 – Resultados finales