Fundamentación de robótica (Gpo 101) Martes 06 de Mayo del 2023
Antonio Silva Martínez
A01173663

Actividad 6 (SLAM de LiDAR)

El seguimiento de waypoints es una técnica fundamental en el campo de la robótica, ya que permite a los robots navegar de manera precisa y autónoma a través de entornos complejos. En la ingeniería robótica, el seguimiento de waypoints es esencial para el diseño y desarrollo de sistemas robóticos eficientes y efectivos. Los waypoints son puntos específicos en un entorno que un robot debe alcanzar durante su trayectoria de movimiento. El seguimiento de estos puntos es vital para lograr un movimiento suave y preciso del robot y para garantizar que este pueda cumplir con las tareas y objetivos para los que fue diseñado. En este sentido, el seguimiento de waypoints es una herramienta crucial para el desarrollo de soluciones robóticas avanzadas y eficientes. En este reporte de ingeniería, se analizarán y evaluarán diferentes técnicas de seguimiento de waypoints en robótica, con el objetivo de comprender su importancia y aplicaciones en el diseño y desarrollo de robots autónomos.

Explicación del código;
Este código es un ejemplo de cómo implementar un algoritmo de seguimiento de trayectorias para un robot diferencial que utiliza los algoritmos de Pure Pursuit y Vector Field Histogram (VFH) para seguir una serie de waypoints y evitar obstáculos. El código utiliza un modelo matemático del robot para controlar su movimiento y se simula el comportamiento del robot en un entorno representado por un mapa. En la simulación, se establece la posición inicial del robot y los puntos de referencia que debe seguir, los cuales se definen como los waypoints. La trayectoria de movimiento del robot se actualiza en cada iteración del bucle de simulación y se utiliza un lidar para obtener mediciones de distancia y evitar obstáculos. En resumen, el código simula la navegación autónoma de un robot diferencial que utiliza una combinación de algoritmos para seguir una serie de waypoints y evitar obstáculos en su camino.

```
%% EXAMPLE: Differential Drive Path Following
% In this example, a differential drive robot navigates a set of waypoints
% using the Pure Pursuit algorithm while avoiding obstacles using the
% Vector Field Histogram (VFH) algorithm.
%
% Copyright 2019 The MathWorks, Inc.
%% Simulation setup
% Define Vehicle
R = 0.1;                % Wheel radius [m]
L = 0.5;                % Wheelbase [m]
dd = DifferentialDrive(R,L);
```

```
% Sample time and time array
sampleTime = 0.1;              % Sample time [s]
tVec = 0:sampleTime:120;       % Time array
% Initial conditions
initPose = [1;1;-90];          % Initial pose (x y theta)
pose = zeros(3,numel(tVec));   % Pose matrix
pose(:,1) = initPose;
% Load map
close all
load exampleMap
% Create lidar sensor
lidar = LidarSensor;
lidar.sensorOffset = [0,0];
lidar.scanAngles = linspace(-pi/2,pi/2,51);
lidar.maxRange = 2;
% Create visualizer
viz = Visualizer2D;
viz.hasWaypoints = true;
viz.mapName = 'map';
attachLidarSensor(viz,lidar);
%% Path planning and following
% Create waypoints
waypoints = [1,1; 1,2; 1,3; 1,4; 2,1; 2,2; 2,3; 2,4; 3,1; 3,2; 3,3; 3,4; 4,1; 4,2; 4,3; 4,4];
% Pure Pursuit Controller
controller = controllerPurePursuit;
controller.Waypoints = waypoints;
controller.LookaheadDistance = 0.2;
controller.DesiredLinearVelocity = 0.3;
controller.MaxAngularVelocity = 30;
% Vector Field Histogram (VFH) for obstacle avoidance
vfh = controllerVFH;
vfh.DistanceLimits = [0.05 3];
vfh.NumAngularSectors = 36;
vfh.HistogramThresholds = [5 10];
vfh.RobotRadius = L;
vfh.SafetyDistance = L;
vfh.MinTurningRadius = 0.25;
%% Simulation loop
r = rateControl(1/sampleTime);
for idx = 2:numel(tVec)

    % Get the sensor readings
    curPose = pose(:,idx-1);
    ranges = lidar(curPose);

    % Run the path following and obstacle avoidance algorithms
    [vRef,wRef,lookAheadPt] = controller(curPose);
    targetDir = atan2(lookAheadPt(2)-curPose(2),lookAheadPt(1)-curPose(1)) - curPose(3);
```

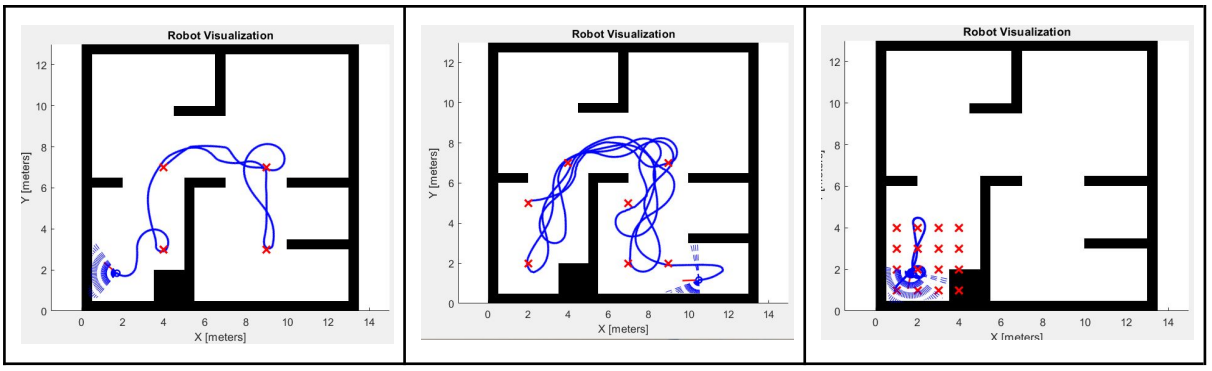
```
    steerDir = vfh(ranges,lidar.scanAngles,targetDir);
    if ~isnan(steerDir) && abs(steerDir-targetDir) > 0.1
        wRef = 0.5*steerDir;
    end

    % Control the robot
    velB = [vRef;0;wRef];                % Body velocities [vx;vy;w]
    vel = bodyToWorld(velB,curPose);  % Convert from body to world

    % Perform forward discrete integration step
    pose(:,idx) = curPose + vel*sampleTime;

    % Update visualization
    viz(pose(:,idx),waypoints,ranges)
    waitfor(r);
end
```

Resultados:
Los resultados obtenidos se pudieron dar gracias a que se tuvieron que ajustar valores como lo son la distancia de detección del muro, la velocidad angular, el tiempo de muestreo, entre otros.



Conclusión

En resumen, el código proporciona una base sólida para la implementación de sistemas de navegación autónoma en aplicaciones de robótica, lo que puede ser de gran utilidad para muchos proyectos y aplicaciones en el campo de la robótica, ya que se simula el movimiento del robot en un ambiente representado por un mapa y utiliza un sensor lidar para medir las distancias a los obstáculos cercanos y evitar colisiones. En este ejemplo, se muestra cómo se puede utilizar un modelo matemático del robot para controlar su movimiento, y cómo se pueden combinar diferentes algoritmos para lograr una navegación autónoma y segura.