Instituto Tecnológico y de Estudios Superiores de Monterrey

TE3002B.502

Implementación de robótica inteligente (Gpo 101)

Semestre: febrero - junio 2023

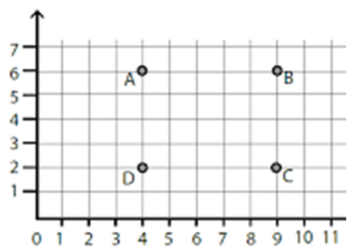
Actividad 6 (SLAM de Lidar)

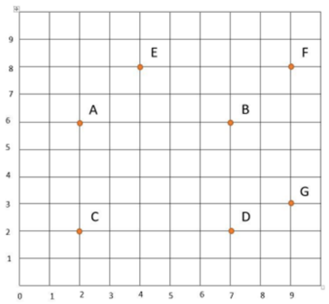
Alumno:

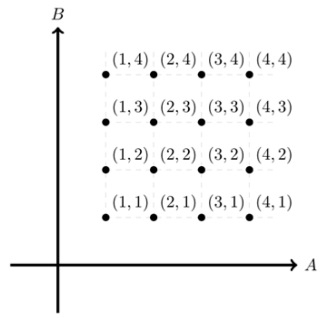
Daniel Ruán Aguilar A01731921

Profesor: Dr. Alfredo García Suárez

1. **En esta actividad primero se Implementa el código requerido para generar el seguimiento de los siguientes waypoints de forma aleatoria.**







Primera trayectoria

Para esta primera trayectoria, se definieron las coordenadas en un vector y con la función randperm() se selecciona una secuencia aleatoria para generar el seguimiento de los waypoints de forma aleatoria.

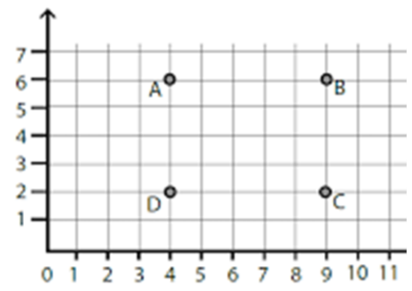
Además se cambió el tiempo final de simulación con 18 s y la posición inicial (initPose) dependiendo de donde se busca que parta la simulación.

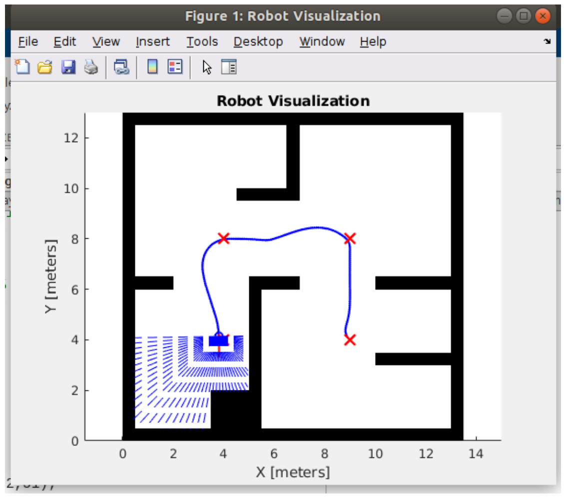
```
%% Simulation setup
% Define Vehicle
R = 0.1;                        % Wheel radius [m]
L = 0.5;                        % Wheelbase [m]
dd = DifferentialDrive(R,L);

% Sample time and time array
sampleTime = 0.1;              % Sample time [s]
tVec = 0:sampleTime:18;        % Time array

% Initial conditions
initPose = [9;4;3/4*pi];            % 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 = 5;
% Create visualizer
viz = Visualizer2D;
viz.hasWaypoints = true;
viz.mapName = 'map';
attachLidarSensor(viz,lidar);
```

```matlab
%% Path planning and following

% Create waypoints
coordenadas = [9 4; 9 8; 4 8; 4 4];

% Se genera una secuencia aleatoria de coordenadas (waypoints)
n = size(coordenadas, 1);
indices_aleatorios = randperm(n);

% Se indexa el vector de coordenadas con los índices aleatorios para %obtener
una trayectoria aleatoria
trayectoria_aleatoria = coordenadas(indices_aleatorios, :);
waypoints = [initPose(1:2)';
            trayectoria_aleatoria];

% Pure Pursuit Controller
controller = controllerPurePursuit;
controller.Waypoints = waypoints;
controller.LookaheadDistance = 0.5;
controller.DesiredLinearVelocity = 0.75;
controller.MaxAngularVelocity = 1.5;

% 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
```

Resultado:





Segunda trayectoria

Para esta simulación se cambiaron más parámetros, la simulación termina con 48 s, se declaró un lidar.maxRange de 4, un controller.LookaheadDistance de 0.5, controller.DesiredLinearVelocity de 0.75 y controller.MaxAngularVelocity en 4.5.

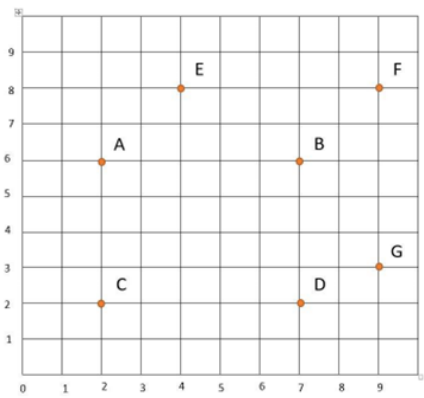
Aparte se declararon nuevos waypoints para formar la trayectoria requerida en un orden aleatorio.

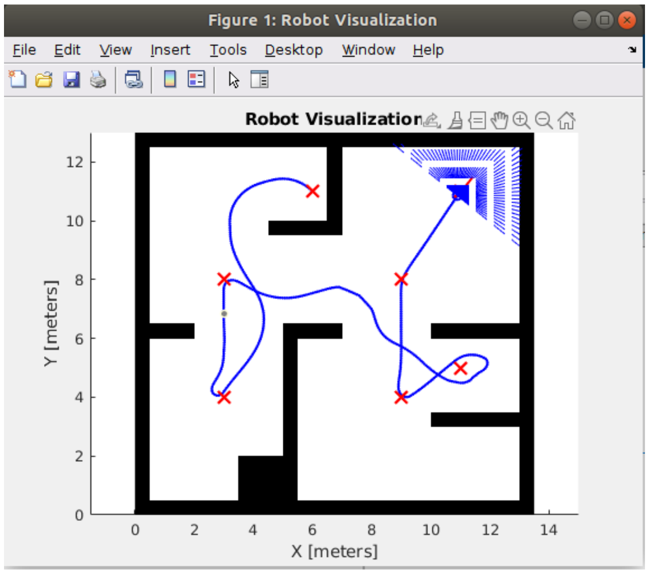
```
%% Simulation setup
% Define Vehicle
R = 0.1;                        % Wheel radius [m]
L = 0.5;                        % Wheelbase [m]
dd = DifferentialDrive(R,L);

% Sample time and time array
sampleTime = 0.1;              % Sample time [s]
tVec = 0:sampleTime:48;        % Time array

% Initial conditions
initPose = [6;11;3/4*pi];            % 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 = 4;
% Create visualizer
viz = Visualizer2D;
viz.hasWaypoints = true;
viz.mapName = 'map';
attachLidarSensor(viz,lidar);

%% Path planning and following
% Create waypoints
waypoints = [initPose(1:2)';
            6 11;
            3 4;
            3 8;
            11 5;
            9 4;
            9 8;
            11 11];

% Pure Pursuit Controller
controller = controllerPurePursuit;
controller.Waypoints = waypoints;
controller.LookaheadDistance = 0.5;
controller.DesiredLinearVelocity = 0.75;
controller.MaxAngularVelocity = 4.5;

% 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
```

Resultado:





Tercera trayectoria

En esta tercera se declaró un tiempo final de 16.5, para que lograra recorrer la distancia tan corta entre cada waypoint se declaró un lidar.maxRange de 0.5 y con el lidar.scanAngles linspace de (-pi, pi).

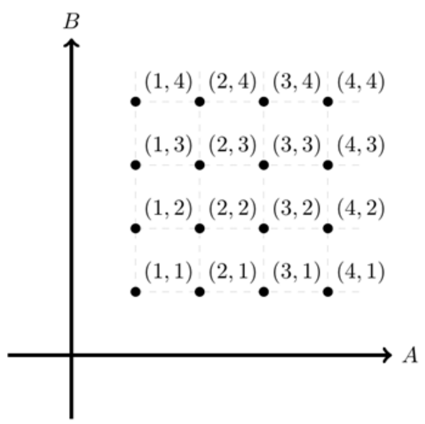
Posteriormente se declararon los waypoints y se aumentó la velocidad angular.

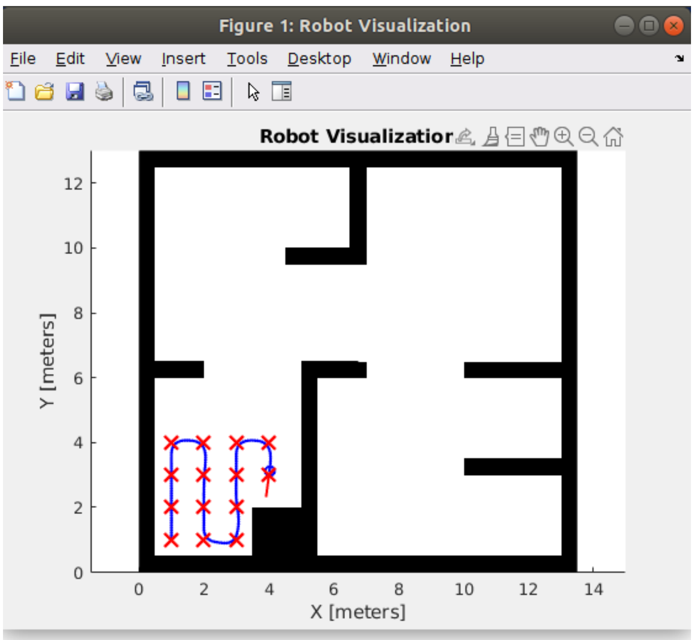
```
%% Simulation setup
% Define Vehicle
R = 0.1;                        % Wheel radius [m]
L = 0.5;                        % Wheelbase [m]
dd = DifferentialDrive(R,L);
```

```matlab
% Sample time and time array
sampleTime = 0.1;              % Sample time [s]
tVec = 0:sampleTime:16.5;        % Time array
% Initial conditions
initPose = [1;1;pi/2];            % 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,pi,100);
lidar.maxRange = 0.5;
% Create visualizer
viz = Visualizer2D;
viz.hasWaypoints = true;
viz.mapName = 'map';
attachLidarSensor(viz,lidar);
%% Path planning and following
% Create waypoints
waypoints = [initPose(1:2)';
            1 1;
            1 2;
            1 3;
            1 4;
            2 4;
            2 3;
            2 2;
            2 1;
            3 1;
            3 2;
            3 3;
            3 4;
            4 4;
            4 3];

% Pure Pursuit Controller
controller = controllerPurePursuit;
controller.Waypoints = waypoints;
controller.LookaheadDistance = 0.5;
controller.DesiredLinearVelocity = 0.75;
controller.MaxAngularVelocity = 4.5;
% 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
```

Resultado:





2. Ahora se implementa el código requerido para generar el seguimiento de los siguientes waypoints de forma secuencial: (1, 2), (2, 10), (11, 8), (8, 2), (8, 8) y (1, 2)

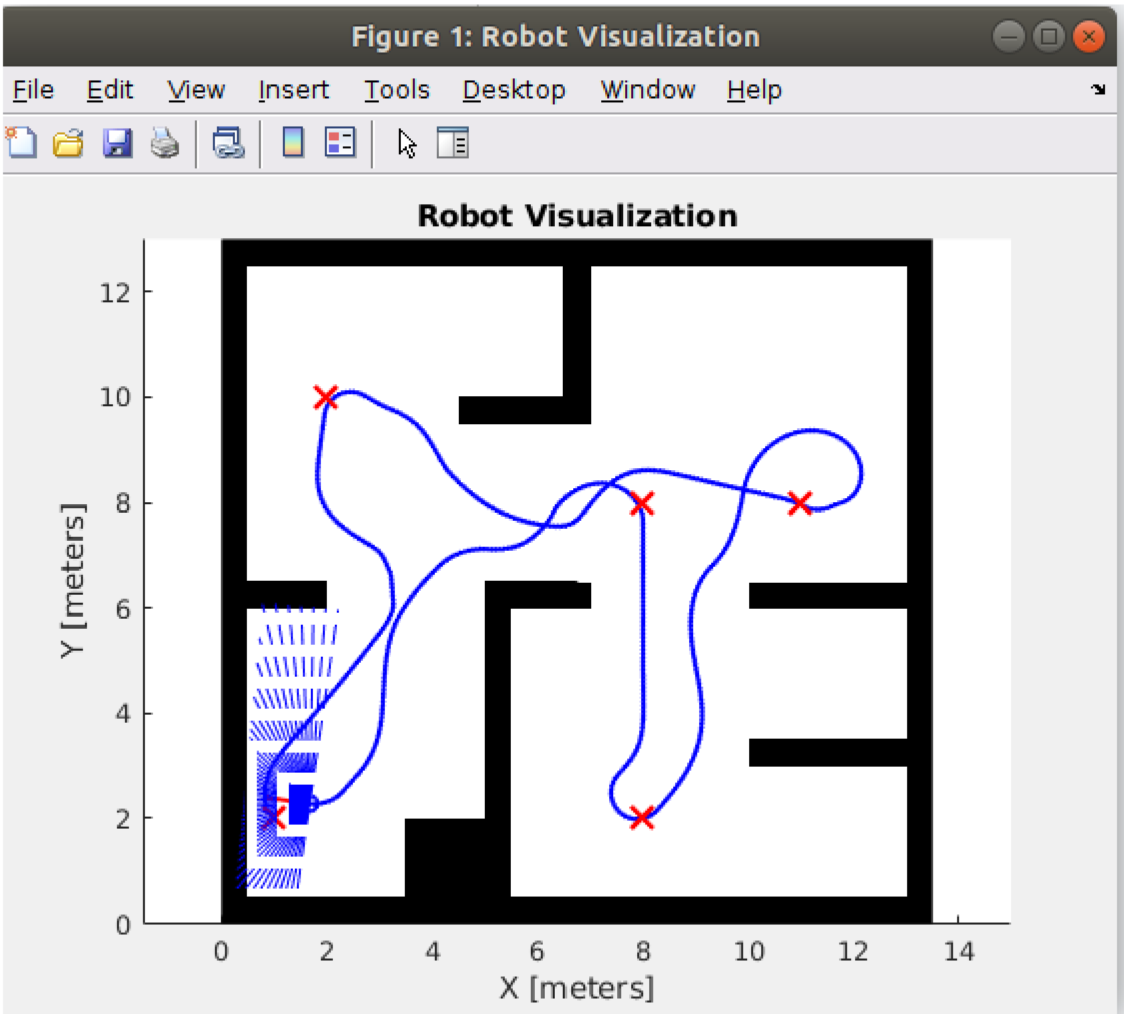
Para partir de la coordenada (1, 2), se cambia su posición inicial en initPose con un ángulo óptimo para empezar la trayectoria, también se cambia el tiempo final de la simulación con 64 para que de tiempo de realizar todo el recorrido y regresar a la posición inicial.

Código de trayectoria

```
%% 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:64;         % Time array

% Initial conditions
initPose = [1;2;3/4*pi];             % 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 = 5;

% Create visualizer
viz = Visualizer2D;
viz.hasWaypoints = true;
viz.mapName = 'map';
attachLidarSensor(viz,lidar);

%% Path planning and following
% Create waypoints
waypoints = [initPose(1:2)';
             1 2;
             2 10;
             11 8;
             8 2;
             8 8;
             1 2];
```

```matlab
% Pure Pursuit Controller
controller = controllerPurePursuit;
controller.Waypoints = waypoints;
controller.LookaheadDistance = 0.5;
controller.DesiredLinearVelocity = 0.75;
controller.MaxAngularVelocity = 1.5;
% 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
```

Resultado.



Como se puede ver, se realiza el recorrido desde la posición inicial (1,2), pasa por todos los puntos (2, 10), (11, 8), (8, 2), (8, 8) y regresa al (1,2) sin chocar con los bordes negros del “exampleMap”.