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

% Sample time and time array
sampleTime = 0.05;              % Sample time [s] (más preciso)
tVec = 0:sampleTime:470;         % Time array (más tiempo para maniobrar)

% Initial conditions
initPose = [8;4;pi/2];             % Initial pose (x y theta)
pose = zeros(3,numel(tVec));    % Pose matrix
pose(:,1) = initPose;

% Load map

%complexMap       41x52               2132  logical
%emptyMap         26x27                702  logical
%simpleMap        26x27                702  logical
%ternaryMap      501x501            2008008  double

close all
load complexMap

% Create lidar sensor
lidar = LidarSensor;
lidar.sensorOffset = [0,0];
lidar.scanAngles = linspace(-pi,pi,360);     % Menos resolución para mayor robustez
lidar.maxRange = 5;  %2                      % Rango suficiente sin sobrever
obstáculos

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

%% Path planning and following

% Create waypoints
waypoints = [initPose(1:2)';
             4,8;
             3,14;
             13,14;
             23,13;
             21,9;
             16,4;
             21,9;
             23,13;
```

```
            13,14;
            3,14;
            4,8;
            8,4];

% Pure Pursuit Controller
controller = controllerPurePursuit;
controller.Waypoints = waypoints;
controller.LookaheadDistance = 0.4; %0.1          %
controller.DesiredLinearVelocity = 0.3; %0.3
controller.MaxAngularVelocity = 20;

% Vector Field Histogram (VFH) for obstacle avoidance
vfh = controllerVFH;
vfh.DistanceLimits = [0.05 3];
vfh.NumAngularSectors = 900;
vfh.HistogramThresholds = [5 10];
vfh.RobotRadius = L;
vfh.SafetyDistance = L;
vfh.MinTurningRadius = 0.1;

%% 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

    % Check if goal is reached
    goal = waypoints(end,:);
    distToGoal = norm(pose(1:2,idx) - goal');
    if distToGoal < 0.2
        break;
    end

    % Perform forward discrete integration step
```

```
    pose(:,idx) = curPose + vel*sampleTime;

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