SWROB2
Exercise 4.2
Mandatory

Submitted by:

Study nr	Name	Study line
202005180	Nicolaj Meldgaard Pedersen	E
202105443	Johannes Baagøe	E
201270449	Anders Sandø Østergaard	EP
201905293	Daniel F. Borch Olsen	E

Århus Universitet

March 6, 2024

Contents

1 Exercise: Reactive navigation and path planning

1.1 Objective

The goal is to understand and be using Probabilistic Roadmap (PRM) in combination with the turtlebot.

1.2 Connection to the TurtleBot3 from powershell

For making the connection to the turtlebot we are connecting to the WiFi

- **ssid:** turtlebot
- **password:** turtlebot3

from powershell type:

ssh ubuntu@192.168.72.251, password: turtlebot

1.3 Starting ROS on turtlebot from powershell

from powershell type

roscore

1.4 Connection to TurtleBot3 from Matlab

For setting the ros environment variable and setting the IP on the host (turtlebot):

```
setenv('ROS_MASTER_URI','http://192.168.72.251:11311')
```

For setting the IP on the local machine

```
setenv('ROS_IP','192.168.72.220')
```

The following command is closing existing connection to be ensure that when the user is connection the robot isn't connected to anyone else

```
rosshutdown();
```

This command will be doing the initialization of the connection between ROS and Matlab

```
rosinit('http://192.168.72.251:11311','NodeHost','
   192.168.72.220');
```

Matlab script for init

```
setenv('ROS_MASTER_URI','http://192.168.72.251:11311')
setenv('ROS_IP','192.168.72.220')
rosshutdown();
rosinit('http://192.168.72.251:11311','NodeHost','
   192.168.72.220');
```

1.5 Probabilistic Roadmap Algorithm (PRM)

The Probabilistic Roadmap Algorithm (PRM) is a method developed for navigating relatively large maps. To start, we create the PRM object as follows:

```
prm = PRM(shannon)
```

Following the object creation, we initiate the planning process by specifying the number of points in the roadmap:

```
prm.plan('npoints', 150)
```

Through this process, 150 roadmap nodes are created. Each point is connected to its neighbor by a direct line path that does not intersect any obstacles, thereby constructing a network encompassing the entire map.

We can visualize the points by typing

```
prm.plot()
```

as we can see on the figure then there are dots and lines on our map, where the dots are representing the points and the lines are obstacle free-path between the points.

2 Road map in Shannon

2.1 Overview of the Code

This code demonstrates a process for simulating and navigating a robot within a virtual environment using the Robot Operating System (ROS) and specific robotics algorithms. It encompasses several steps from creating an environmental map, defining obstacles, planning a route, and finally steering the robot to follow the route. Below is what occurs in each of the key sections:

2.2 Creating a Binary Occupancy Map

A binary occupancy map (`binaryOccupancyMap`) is created to represent an environment measuring 6 by 4 meters with a resolution of 10 cells per meter. This map is used to define areas accessible to the robot and those blocked by obstacles, initially set up without any obstructions.

2.3 Defining Walls and Obstacles

A numerical array (`walls`) matching the map's size is created to represent the locations of walls and obstacles. Values are set to 1 to indicate the presence of an obstacle in specific cells (e.g., walls around the edge and three 'tables' as obstacles in the environment). This

information is transferred to the occupancy map, rendering these areas inaccessible to the robot.

2.4 Inflating the Map

The map is "inflated" by enlarging the obstacle sizes on the map to account for the robot's size. This ensures the robot does not attempt to navigate too close to an obstacle, thereby risking collision.

2.5 Creating a Probabilistic Roadmap (PRM)

A probabilistic roadmap method is utilized to generate a route for the robot. This involves creating a network of randomly generated points in free space, which are then connected with paths that avoid obstacles. The method uses 500 points to cover the map, attempting to find a feasible route from a start to an end position.

2.6 Navigating the Route

The route found using the PRM is employed along with a pure pursuit algorithm to steer the robot. Pure pursuit is a guidance algorithm that continually adjusts the robot's speed and direction to follow a route consisting of a series of waypoints. The algorithm adjusts the robot's linear and angular velocities based on the robot's current position and the desired route.

2.7 Position and Velocity Updates

The robot's position is continually updated using feedback from its sensors (in this case, position and orientation from a tf topic in ROS). Speed control commands are sent to the robot based on calculations from the pure pursuit controller, until the robot reaches its goal within a defined tolerance.

2.8 Visual Representations

Various visual representations of the environment, the robot's planned route, and eventually the actual path the robot attempts to follow are plotted throughout the code. This includes:

- The initial occupancy map before inflation.
- The occupancy map after inflation, showing how the "sizes" of obstacles have been increased.
- The probabilistic roadmap with the planned route.
- Updates on the robot's position and route adjustments as it follows the planned path.

These figures provide visual feedback on how the robot interacts with its environment, navigates around obstacles, and ultimately attempts to reach its goal.

```
%% Create a binary occupancy Map
% Here we create a map that is 10x10 meters with 1 square per
    meter.
map = binaryOccupancyMap(6,4,10)

%% Create a numeric array with matching size
walls = zeros(40,60);

% Left wall
walls(1:40,1) = 1;
% Right wall
walls(1:40,60) = 1;
% Top wall
walls(1,1:60) = 1;
% Bottom wall
walls(40,1:60) =1;

% Creating obstacles
%Table 1
walls(1:30, 10:20) = 1;

%Table 2
walls(35:40,27:60) = 1;

%Table 3
walls(7:27,40:60) = 1;

setOccupancy(map,[1,1],walls,'grid')
show(map)

%% Inflate the map to account for the size of the robot.
% We are inflating the obstacles by the radius of the robot
   and the robot
% is 20cm so by 0.1m.
inflate(map, 0.2);
show(map)

%% Creating a Proberlistic Roadmap
%Generating a probabilistic roadmap with 500 points
PRM = mobileRobotPRM(map,500);
show(PRM)

%% Adding a start and goal and calculating a path
%Defining start/end locations as measured IRL
startLocation = [0.5 3.5];
endLocation = [5.5 3.65];
```

```
%Generating the path using the points in the probabilistic
   road map
path = findpath(PRM,startLocation,endLocation);
show(PRM)

%% Implementing the purepersuit algorithm from last exercise
%Creating publishers and subscribers
vel_pub = rospublisher('/cmd_vel');
ang_sub = rossubscriber('/imu');
pos_sub = rossubscriber('/tf');

%Reset the pos
reset_pub = rospublisher('/reset');
send(reset_pub,rosmessage(reset_pub));

%Setting the initial position of the robot (should be ~0,0
   wince we reset
%the robot)
pos = update_pos(pos_sub)

%Defining the controller and all it's parameters
controller = controllerPurePursuit;
controller.DesiredLinearVelocity = 0.2;
controller.MaxAngularVelocity = 2;

%We found that the robot is relatively stable with a
   LookAheadDistance of 0.4
controller.LookaheadDistance = 0.4;

%The path is defined as the points found by the PRM
controller.Waypoints = path;

%defining the radius, endpoint and initializing
   distanceToGoal
goalRadius = 0.1;
robotGoal = path(end,:);
distanceToGoal = norm(pos - robotGoal)

while(distanceToGoal > goalRadius)
  %We start the loop by updating the position of the robot
  robotCurrentPose = update_pos(pos_sub);

  % Compute the controller outputs, i.e., the inputs to the
   robot
  [v, w] = controller(robotCurrentPose);

  %Then we update the angular and linear velocities based on
   the
```

```
  %controller outputs.
  update_vel(v,w,vel_pub)

  %At last we check the distance to goal for whether we have
   reached the
  %end of the path.
  distanceToGoal = norm(robotCurrentPose(1:2) - path(end, :)
   ')
end

function [output] = update_pos(pos_sub)
  %We start by getting the transform data from the tf topic
  receive(pos_sub,10);

  %the x and y data is = to the x and y of the tf topic
  pos.x = pos_sub.LatestMessage.Transforms.Transform.
   Translation.X;
  pos.y = pos_sub.LatestMessage.Transforms.Transform.
   Translation.Y;

  %We use the function quat2angle to transform the angle from
    quaternion to
  %yaw in radians
  quat.X = pos_sub.LatestMessage.Transforms.Transform.
   Rotation.X;
  quat.Y = pos_sub.LatestMessage.Transforms.Transform.
   Rotation.Y;
  quat.Z = pos_sub.LatestMessage.Transforms.Transform.
   Rotation.Z;
  quat.W = pos_sub.LatestMessage.Transforms.Transform.
   Rotation.W;

  [roll,pitch,yaw] = quat2angle([quat.X,quat.Y,quat.Z,quat.W
   ]);

  pos.theta = yaw;

  %In the end we output the data and add the initial position
    of the robot to
  %x and y
  output = [pos.x+0.5; pos.y+3.5; pos.theta];
end


function [true] = update_vel(v,w,vel_pub)
  %Very simple function. We get a new linear and angular
   velocity from the
  %controller and output it to the cmd_vel topic.
```

```
  twistmsg = rosmessage(vel_pub);

  twistmsg.Angular.Z = w;
  twistmsg.Linear.X = v;

  send(vel_pub,twistmsg);
end
```

Following figures Binary Occupancy Map, Inflated Binary Occupancy Map, Proberlistic Roadmap and Proberlistic Roadmap with path was plottet in Matlab

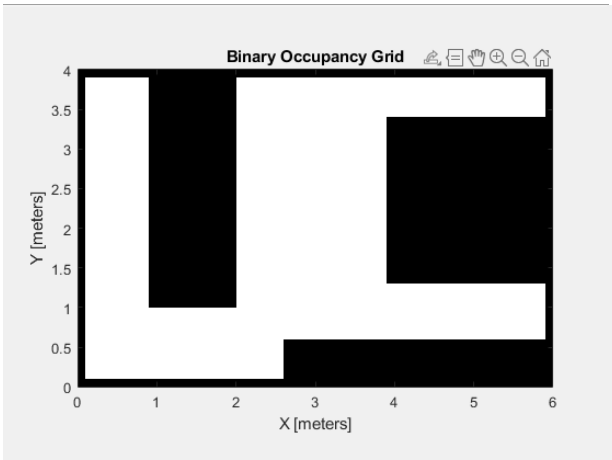

Figure 1: Binary Occupancy Map

Figure 2: Inflated Binary Occupancy Map

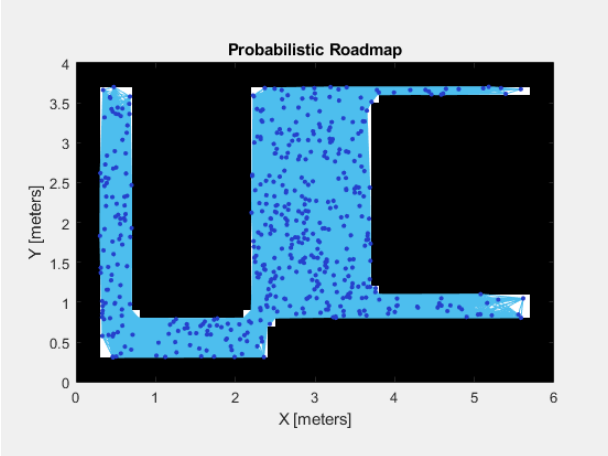

Figure 3: Proberlistic Roadmap

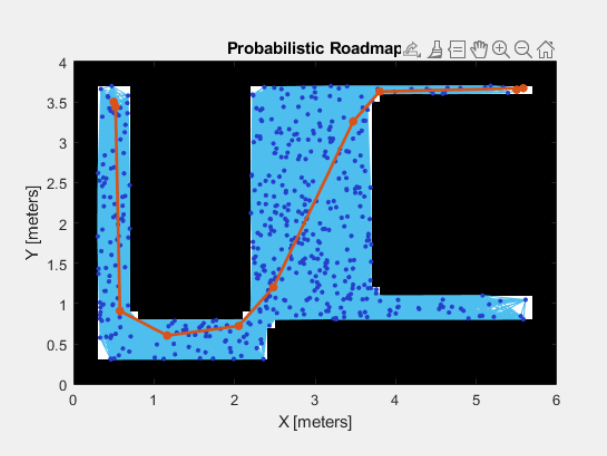

Figure 4: Proberlistic Roadmap with path