Humanoid Robots
Exercise Sheet 10 - RRT and Inverse Reachability Maps

Note: You have to submit your solutions before the lecture next Thursday.

Exercise 17 (10 points)

Consider a robot in a 2D environment represented by a grid map. Our objective is to find a path between a start cell and a goal cell. The robot is initially located at the start cell’s center and it can move to any of the eight neighboring cells’ centers (if they are not occupied). Use RRTs to find a possible path for the robot.

Exercise steps:

a) Implement the method `getRandomNode` that returns a random grid cell in the map, with the following conditions:
   - The cell must be free of obstacles.
   - The cell must not already belong to the tree that will be expanded in the current step.
   - The random generation should be biased towards the goal of the current tree, i.e., the method should return the goal node instead of a random node with a probability of 10%.

   Hint: Use the standard C function `rand()` that returns a random integer between 0 and `RAND_MAX`.

b) Implement the method `distance` that computes the Euclidean distance between two given cells.

c) Implement the method `getClosestNodeInList` that returns the grid cell with the smallest distance to the current node from a list of cells.

d) Implement the method `getNeighbors` that returns a list of neighbors of the current cell that are not occupied and have not already been expanded in the current tree.
e) Implement the method `tryToConnect` that checks for each neighbor whether it is already contained in the list of expanded nodes of the other tree. If this is the case, then we have found a connection between the two trees and the method should return the neighbor as the connection node. Otherwise, the method should return `NULL`.

f) Implement the method `addNearestNeighbor` that determines which neighbor is the nearest with respect to the random node and adds that neighbor to the tree.

g) Have a look at how the method `extendClosestNode` combines the above methods (you don’t have to implement anything here).

h) Implement the method `constructPath` that reconstructs the path from the start node to the goal node once the trees have been connected.

i) Implement the method `planPath` which grows the trees from the start and goal nodes and returns the complete path, using the RRT-connect algorithm on slide 33.

The Gnuplot script in `scripts/plot.gp` and the animation in the Wiki show an animation of the node expansion and the final path found by your implementation.
Exercise 18  (10 points)

In this exercise, we will compute reachability maps and inverse reachability maps for the robot arm with three links from exercise 14:

![Robot arm with three joints](image)

Figure 1: Robot arm with three joints

Reachability map

Compute the reachability map of the endeffector with respect to the robot’s base according to the following steps:

a) Implement the method `sampleConfiguration` for choosing a random configuration of the robot’s joints \((q_0, q_1, q_2)\) and return the configuration as a vector. The joint angles should be sampled uniformly within the following bounds:

\[
\begin{array}{c|c|c}
\text{joint} & \text{min} & \text{max} \\
\hline
q_0 & 0 & \frac{\pi}{2} \\
q_1 & -\pi & +\pi \\
q_2 & -\pi & +\pi \\
\end{array}
\]

b) Compute the following measurement of manipulability for a given joint configuration in `computeManipulability()`:

\[
\text{score} := 1 - \frac{1}{4\pi} \left(|4q_0 - \pi| + |q_1| + |q_2| + |\epsilon|\right) \tag{1}
\]

(This manipulability measurement favors configurations where \(q_0\) is near 45 degrees and penalizes configurations with pointed angles between links.)

c) Implement the method `computeRM` that computes the reachability map of the endeffector by iterating the following steps:

- Sample a joint configuration.
- Compute the end effector pose with the `forwardKinematics` method.
- Check that the end effector is above the ground (i.e., \(\epsilon_y > 0\)). If the end effector collides with the ground, then sample again. (You don’t have to perform other checks such as self-collisions, object collisions, etc. in this exercise.)
• Compute the manipulability score for the configuration.
• Add the configuration to the reachability map with the `addToRM` method.

With the Gnuplot script in `scripts/plot-rm.gp` or in the Wiki you can inspect the resulting reachability map. The robot’s base is located at the origin and the colors represent the manipulability of the possible end effector poses.

**Inverse reachability map**

For the sake of simplicity, we assume that the robot arm is mounted on a wheeled base so that we do not have to deal with stance feet and swing feet.

The inverse reachability map indicates suitable positions where the base of the robot has to be located so that the gripper can reach the desired object. Using this map, the robot can first drive to a suitable place near the table and then grasp the drink.

d) Implement the method `computeIRM`. The voxels of the reachability map are given to the method as an argument. For each configuration stored in the reachability map, add an entry to the inverse reachability map as follows:

• Compute the end effector pose from the configuration’s joint angles with the `forwardKinematics` method.
• Convert the end effector pose to a homogeneous transformation matrix, find the inverse, and convert the inverted transformation matrix back to the pose of the robot’s base \((b_x, b_y, b_θ)\). The result is the pose of the base expressed in the coordinate system of the gripper.
• Add the base pose together with the joint angles and the manipulability score to the inverse reachability map by calling the `addToIRM` method.

With the Gnuplot script in `scripts/plot-irm.gp` and the plot in the Wiki you can inspect the resulting inverse reachability map. The robot’s gripper is located at the origin and the colors represent the possible base poses that are suitable for reaching the object in the coordinate frame of the gripper.

Deadline: Thursday, 15 July 2021, 8:30 am