Human-Oriented Robotics Winter term 2013/2014
Prof. Dr. Kai Arras, Social Robotics Lab University of Freiburg
Lab instructors: Timm Linder, Luigi Palmieri, Billy Okal Department of Computer Science

Exercise 11: Rapidly-Exploring Random Trees

Submission: Send your solution to `palmieri@informatik.uni-freiburg.de` until February 12, 2014 with subject line "`[exercises] Sheet 11`". All files (Matlab scripts, exported figures, handwritten notes in pdf/jpg format) should be put into a single zip file named `lastname_sheet11.zip`.

You will need to download the Matlab frame `RunRRT.m` with functions `checkcollision.m`, `extend.m`, `diffdrivekinematic.m` and the environment file `environment.txt`.

Exercise 11.1: Getting Started, Plot the Environment

In this exercise we solve a motion planning problem for a differential drive robot using a popular and flexible method called rapidly-exploring random tree (RRT) algorithm.

a) In the Matlab frame `RunRRT.m` load the environment file `environment.txt`, it contains a set of circular shaped obstacles in a `x,y,radius`-syntax on each row. Plot these obstacles using either the `drawellipse` or the `fill` command.

   In the frame, you find several predefined parameters such as the size of the robot and the goal region, the dimensions of the C-space, as well as the initial and goal pose of the robot. Plot a robot at the initial pose (using `drawrobot`) and plot the circular goal region.

Exercise 11.2: Rapidly-Exploring Random Tree Algorithm

We now implement the RRT algorithm, a popular method for robot motion planning that grows a tree into the C-space toward the goal.

a) **Initialize the tree**. We suggest to use an array of struct for the tree named `tau`. Each array entry is a vertex of `tau` with fields `id` (a unique identifier), `pid` (the identifier of the parent vertex), and `pose` (the 3×1 configuration (x, y, θ)). In addition, each vertex has the fields `edgeq` (an $n \times 3$ matrix which holds the sequence of poses by which the robot moved from the parent vertex to the current vertex) and `edgeu` (a 1×2 vector of control inputs $\mathbf{u} = (v, \omega)$ that caused the movement.

   Initialize the tree with $\mathbf{q}_{init}$, assign an identifier of 1 and an identifier of 0 to the parent vertex. The other fields may be initialized with an empty matrix `[]`.

b) **Sample and check new configuration**. Set up the main loop which terminates only when the robot has reached the goal region. As a first step, generate a random configuration $\mathbf{q}_{rand}$ by uniformly sampling in the configuration space. Implement a function `qrand = sampleconfiguration(climits)` that takes the limits of the C-space in x, y, and θ (values given in the frame) and returns a sampled configuration `qrand`. Then, check if the newly created configuration is in the free space using the provided function `checkcollision`.

c) **Find nearest vertex**. If the new configuration has no collision, proceed by finding the vertex of the tree which is closest to the new configuration. Implement a function `qnear = findnearestvertex(tau,qrand)` which takes the tree and a random configuration as input and returns the closest tree vertex $\mathbf{q}_{near}$ in an Euclidean sense. **Note**: we are only interested in nearness in x and y, not in θ.

d) **Extend the tree**. Once $\mathbf{q}_{near}$ has been found, we extend the tree toward $\mathbf{q}_{rand}$. In the frame you find the function `extend.m` which depends on the function `diffdrivekinematic.m`. They generate motions that are kinematically feasible, i.e. motions that a wheeled robot can actually carry out. Call the extend function as shown in the frame, it will return the new vertex $\mathbf{q}_{new}$.

   Then, we need to check if the motion toward $\mathbf{q}_{new}$ is collision-free. Implement a function `collision = checkedgecollision(qnew,robotradius,obs)` for this purpose. It shall iterate over the sequence of poses in `qnew.edgeq` and verify that none of these poses cause a collision. Again, consider only x and y of the poses. If no collision occurs save $\mathbf{q}_{new}$ in the tree and plot the edge. Make sure to properly copy all information from $\mathbf{q}_{new}$ into the new tree vertex including a unique identifier.

e) **Goal check**. Check if $\mathbf{q}_{new}$ is in the goal region (considering only x and y), and if so, terminate the algorithm. Finally, extract the final path and the correspondings controls from the tree. Write a function `[p,u] = extractpath(qnew,tau)` that, starting at $\mathbf{q}_{new}$, traverses `tau` along the parent-child relations until the first vertex has been reached and accumulate the pose sequences (from `edgeq`) and the controls (from `edgeu`) into two matrices.

Exercise 11.3: Plots and Discussion

a) Plot the path into the figure by using `drawrobot` so to differentiate the tree from the final path. In a second figure plot the controls: v, the translational robot velocity (first column), and ω, the angular robot velocity (second column).

   By looking at the last figure, can you find a problem with the velocity profiles along the path?

b) You are free to play around, change parameter values, change goal and obstacle locations or plot more information.