CS 4610/5335: Robotic Science and Systems (Spring 2021) — Robert Platt
Northeastern University — Due XXX, 2021

# HW 4: Motion Planning, Sample Based

Please remember the following policies:

- Submissions should be made electronically via the Canvas. Please ensure that your solutions for both the written and/or programming parts are present and zipped into a single file.
- Solutions may be handwritten or typeset. For the former, please ensure handwriting is legible.
- You are welcome to discuss the programming questions (but *not* the written questions) with other students in the class. However, you must understand and write all code yourself. Also, you must list all students (if any) with whom you discussed your solutions to the programming questions.

## Sampling-based Motion Planning

Download the starter code from Piazza (`hw4.zip`). In this file, you will find:

- `hw4_motion.m`: Main function. Call `hw4_motion(<questionNum>)` with an appropriate question number (0–5) to run the code for that question. Do not modify this file! (Feel free to actually make changes, but check that your code runs with an unmodified copy of `hw4_motion.m`.)
- `M0.m` – `M5.m`: Code stubs that you will have to fill in for the respective questions.
- `check_collision.m, check_edge.m`: Helper functions to perform collision checking.

In this part of the assignment, you will implement two sampling-based motion planning algorithms, the probabilistic roadmap (PRM) and the rapidly exploring random tree (RRT). A 4-DOF 2-link arm has been defined, together with a spherical obstacle. The cylinders depicting the arm's links should not collide with the obstacle, but the cylinders located at the joints are just markers visualizing the axes of rotation and do not contribute to potential collisions.

M0. **2 points.** Familiarize yourself with the provided code in `hw4_motion.m` and the robot that has been defined. Use the code in `M0.m` to visualize the robot in various configurations. The code in `hw4_motion.m` that calls `M0` also checks whether the configuration is within joint limits, and whether the configuration is in collision.

Look at the code in `check_collision.m` and `check_edge.m`. Explain what each function is doing to check for collisions, for individual configurations and for configuration-space line segments respectively. Identify and describe two potential issues in the collision-checking algorithm (but do not try to fix them).

M1. **1 points.** Despite the issues you may have identified in M0, we will proceed with the provided collision checking functions. Given the provided joint limits, generate uniformly distributed samples from configuration space. The configurations should be within the joint limits, but they can be in collision.

M2. **2 points.** Implement the PRM algorithm to construct a roadmap for this environment. Your graph should contain `num_samples` collision-free configurations within joint limits. For each vertex, consider the nearest `num_neighbors` neighbors, and add an edge if the segment between them is collision-free. Edges are specified using the output `adjacency` matrix, where `adjacency(i,j)` is the distance between vertices $v_i$ and $v_j$ if an edge exists between them, and is 0 otherwise. Note that the adjacency matrix should be symmetric (edge between $v_i$ and $v_j$ implies edge between $v_j$ and $v_i$), and that each vertex may have $\leq$ `num_neighbors` neighbors (if some edges are in collision) or $\geq$ `num_neighbors` neighbors ($v_j$ may be closest to $v_i$, but there may be other vertices closer to $v_j$).

*Hint:* The next question relies on the output of `M2.m`, which may take a while to compute. To avoid re-computing it in future questions, we have provided functionality to save it in your MATLAB workspace, and pass it in on future calls:

```
[samples, adjacency] = hw4_motion(2);
hw4_motion(3, samples, adjacency);
```

M3. **2 points.** Use the roadmap to find a collision-free path from the start configuration to the goal configuration. You will need to appropriate points to get "on" and "off" the roadmap from the start and goal respectively. *Useful functions*: `shortestpath`

M4. **2 points.** Implement the RRT algorithm to find a collision-free path from the start configuration to the goal configuration. Choose appropriate hyperparameter values for the step size and frequency of sampling $q_{\text{goal}}$. Remember to traverse the constructed tree to recover the actual path.

M5. **2 points.** The path found by RRT likely has many unnecessary waypoints and motion segments in it. Smooth the path returned by the RRT by removing unnecessary waypoints. One way to accomplish this is to check non-consecutive waypoints in the path to see if they can be connected by a collision-free edge.

M6. **0 points.** If M4 and M5 have been implemented correctly, this question should require no further implementation. Watch your algorithm work on a more challenging motion planning problem. Can you make it even harder?