# Homework 2:
# Motion Planning

E. Todorov, CSE P590

Due May 29, 2014

## Problem statement

In this assignment you will implement a motion planning method based on probabilistic roadmaps to find and animate collision-free paths for a 7 degree of freedom robotic arm that must move through obstacles. See pages 25-41 of *Approximate Optimization* lecture notes for an overview of the approach.

## Part 1: Find valid arm states

In this part of the assignment, you will create a collection of randomly-generated collision-free arm states to be used in your path planner. To help you, we provided *generateSample* function in the starter code that will generate a 7-dimensional vector defining a random arm state. To test validity of a particular state, we provided *isValidState* function that will determine if the state has collisions with obstacles.

Using these two functions, generate a collection of valid arm states. It is up to you to decide how many samples you need. In the starter code, you are also given special state vectors $q_{\text{init}}^{1,2,3}$ and $q_{\text{goal}}$, which are three initial states and goal state, respectively. Make sure all these vectors are included in your collection.

Within the Markov Decision Process (MDP) framework described in the lecture notes, this collection is the discrete set of states for the MDP.

## Part 2: Find valid state transitions

Once you generated a collection of valid states, the next step is to connect these states via valid, collision-free transition paths. You must implement *isValidPath($q_a$,$q_b$)* function in the starter code, which given two state vectors $q_a$ and $q_b$ generates 10 in-between interpolated states and tests that all of them are valid (collision-free). Interpolation can be done by simply interpolating each of 7 degrees of freedom independently.

Instead of checking all possible pairs of states for valid transitions (which will become intractable as the number of samples becomes large), you will only consider $K$ closest can-

didates for each state. "Closest" in this case means standard L2 norm between state vectors $||q_a - q_b||$. It is up to you to decide how many neighbors $K$ you will need.

At this point, each state will have at most $K$ nearby states that it can transition to without colliding with any obstacles during the transition. These transitions define the discrete action space of your MDP. Note that some unlucky states may not have $K$ (or any) valid transitions.

# Part 3: Find shortest paths

Having a set of valid states and transitions, you will find shortest paths from each of the three initial states to the goal state we have provided. Let the distance of a path be the sum of distances between state vectors in that path: $dist(q^{1,..,M}) = \sum_{t=1}^{M-1} ||q^m - q^{m+1}||$. Three initial state vectors $q_{\text{init}}$ and goal $q_{\text{goal}}$ are provided in the starter code.

It is up to you find a way of computing shortest paths. One way is to treat the set of valid states as vertices of a graph, set of valid transitions as undirected edges of the graph and use your favorite graph-based shortest path algorithm. Another way is to treat the problem as a deterministic MDP and use value iteration algorithm described on page 9 of *Markov Decision Processes* lecture. In the discrete, deterministic, non-discounted case, value iteration from the lecture simplifies to (changing notation from $x$ to $q$):

$$v^{n+1} = T[v^n]$$
$$T[v(\bullet)](q) = \min_{q' \in N(q)} \{l(q, q') + v(q')\}$$

Where $q$ is one of the valid states, $N(q)$ is the set of $K$ nearby valid transitions for each state, and $l(q, q')$ is the distance between two states. The shortest path is then found by starting at an initial state and choosing transitions to neighbors that have the lowest value.

When you generate the shortest paths, record the distances for each of the three paths.

# Part 4: Animate paths

Once you generated the three shortest paths, play them back in mujoco server by following in succession the transitions you found. For each transition, animate a succession of 10 interpolated in-between states, just as you did when checking for valid paths. You can use provided *mjSetState* function to directly set the arm to a specified state vector (see starter code for an example of how this is done). Animate three shortest paths in succession.

# Starter code

On the Catalyst assignment page you will find a package containing Mujoco physics simulation server and a starter Visual Studio project containing a Mujoco client that you will extend.

After compiling MujocoClient, start *Mujoco.exe* (the server). You will see simulation display window. Then start *MujocoClient.exe*. The client will load a hand model we provided and set the arm to a randomly generated state configuration. Read starter code and comments for more details. Your task is to extend the starter code to complete the four assignment parts.

When generating state samples and transitions, the rendering in mujoco server may slow down the generation. To speed this process up during testing, you may want to press $F2$ and turn off *Geom 0* rendering.

# What to submit

Submit an executables solving each of the four parts of the assignment along with your source code. The executable should finish by playing back the three shortest paths you found. Additionally, submit a short writeup (pdf or txt) listing the best distances for the three shortest paths you found along with the number of generated samples $N$ and neighbors $K$ you used. What happens to distance of the paths as $N$ goes from small to large? As $K$ goes from small to large?