

CS 221, Autumn 2009

Problem Set #1 Programming Assignment

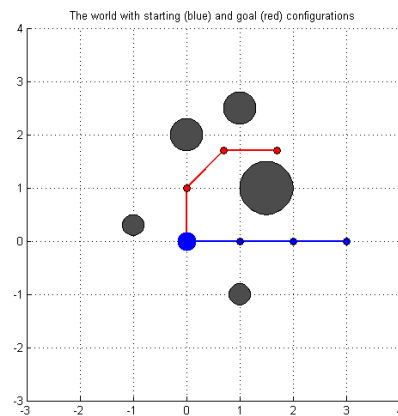
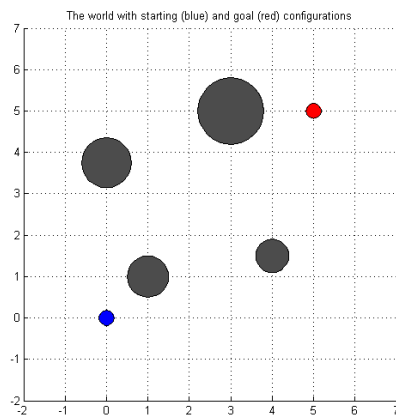
Due by 9:30am on Tuesday, October 13. Please see the course information page on the class website for late homework submission instructions. SCPD students can also fax their solutions to (650) 725-1449. We will not accept solutions by email or courier.

Programming part (30 points)

In this programming assignment, we will be working with two simulated robots (a simplified spherical robot and an n -jointed arm) in various workspaces. The goal of the assignment is to get the robot to move to different configurations given the constraints of the workspace. You will implement several search algorithms and compare their performance.

The robots and workspace

To get started, you must first understand the environment in which your robot will work. The first type of a robot is a small sphere free to move in a two-dimensional world. The second type is an n -jointed robotic arm (n may vary) anchored to a fixed location, also in a two-dimensional world. Both robots must move from an initial configuration to a goal configuration without colliding with any of the spherical obstacles. The spherical robot's configuration is its two-dimensional location, whereas the arm's configuration is its n -dimensional vector of joint angles. Here are the sample worlds with initial and goal configurations for the two robots:



Note that the worlds we provide you with are just sample worlds and not the only worlds in which your robots will exist. In the first type of world, the robot will always be spherical, but its size (and initial/goal configurations) may vary. In the second type of world, the robot may have different numbers of links of different lengths. In both worlds, the location and sizes of the obstacles may vary, but workspace obstacles will always be spherical. Your code should be general enough to work in any of these worlds.

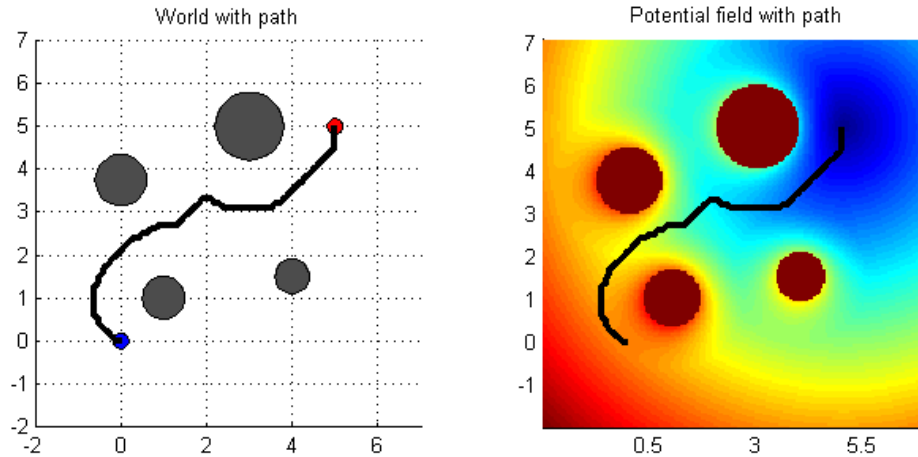


Figure 1: *Left.* The blue dot corresponds to the starting position of the robot, and red is the goal position. The black line is the path obtained using greedy moves down the potential field. *Right.* The potential field for this scenario is represented via a heat map (red = high potential field value, blue = low value).

Potential fields

For the spherical robot, the configuration space is similar to the workspace except with the obstacles dilated by the radius of the robot. A simple way to navigate this space is via a potential field. A potential field assigns a scalar value (which is analogous to a cost) to every point in the configuration space. The purpose of the potential field is to guide the robot to the goal via greedy moves going downhill in the potential field.

An example potential field is shown in Figure 1. A starting and goal configuration for a spherical robot is shown, plus the path obtained by navigating downhill on the potential field.

Probabilistic Roadmaps

A potential field is a simple way to guide a robot through a configuration space, but it has limitations, as you will discover in the programming assignment. It is also difficult to create a potential field when the configuration space is complex. This is the case for the n -jointed robot, where the n -dimensional configuration space is particularly complex. One method of working around this problem is using the probabilistic roadmap technique. We start by generating m random points in the freespace (or, rather, we generate random points in the configuration space and discard those that are not in the freespace until we obtain m points in the freespace). We calculate which pairs of points can be connected within freespace and create edges between them as shown in Figure 2. Finally, using one of several different search algorithms, we search for a path from the start node to the goal node.

Your robot will be provided with m random points in its configuration space. Based on these points, your robot must decide how to move between the different points so that it best reaches the goal. Remember that all these points are in the configuration space and not in the work

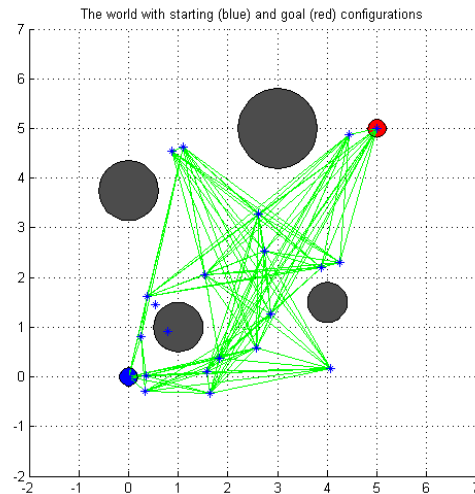


Figure 2: Probabilistic roadmap for the spherical robot.

space. Therefore the “shortest distance” between two points in the configuration space simply means the path that causes the robot to move itself, or move its joints the least, not the path that will cause the end of the robotic arm, say, to move the shortest distance. Keep this in mind when you see the visualizations of the robot performing your searches.

The code infrastructure

The Matlab files for this assignment are available at <http://cs221.stanford.edu/code/pa1.zip>. It will unpack into a folder called `cs221_pa1`. The two topmost-level files are `pa1_matlab_a.m`, which handles the potential field-related part of the assignment, and `pa1_matlab_b.m`, which handles the probabilistic roadmap-related part. The workspaces these files run on are described in XML files. The names of these world description files begin with either “sphere” or “arm,” depending on which type of robot they are for.

Both `pa1_matlab_a.m` and `pa1_matlab_b.m` are separated into a few sections, each with detailed comments explaining their function. The first section of both files returns the `World` object, constructed by parsing the XML world configuration file. This object will always contain the members `World.robot`, which describes the characteristics and starting configuration of the robot; `World.goal`, which specifies the goal configuration; and `World.obstacles`, which defines the spherical obstacles in the world. **You do not need to modify any of the code** in these two high-level functions¹. Rather you only need to implement certain functions that they call. These specific functions are described below.

¹The only exception is to switch between the different worlds you will need to change the argument to the `readAndDisplayWorld` function. See the code for details.

Potential Fields

The file `pa1_matlab_a.m` is for navigating a robot through a workspace by the means of potential fields. This section is relevant for only the spherical robot, not the robotic arm, and is meant to be run on the worlds `sphereWorld0.xml`, `sphereWorld1.xml` and `sphereWorld2.xml` only.

We provide all the code for reading in the world, displaying it, and conducting the search. What you need to implement is the function `getPotentialValue.m`. What this function does is, for the given point in the configuration space `config`, it returns the value of the potential field for it. We have provided a search function that will poll this function to make its movement decisions. You must construct the potential field such that the search will find the goal while steering clear of any obstacles. The search function always moves towards the lowest local value of the potential field, choosing from the N, S, E, W, NE, NW, SE, SW directions.

Probabilistic Roadmaps

The file `pa1_matlab_b.m` is for navigating the robot via a search on a probabilistic roadmap. This is relevant to both the spherical robot and the robotic arm, so it can be run on all of the provided world files.

Here the `World` object will have two extra members: `World.Landmarks` and `World.Connectivity`. The first contains the set of landmarks in freespace that we will use to build the roadmap. It is in the format of an n -by- m matrix, with n the dimension of the configuration space, and m the number of points (so the points lie along columns). The robot's starting configuration will always be in the first column of this matrix (or `World.Landmarks(:,1)` in Matlab) and the goal configuration will be in the last column (or `World.Landmarks(:,end)`).

`World.Connectivity` is the connectivity matrix between the configuration space points (containing a 1 at location i, j and j, i if those two points are mutually reachable). To find all points reachable from a point i , you can use the Matlab command `find(World.Connectivity(:,i))`.

The code you must implement here is the four search functions: `bestFirstSearch.m`, `AstarSearch.m`, `uniformCostSearch.m`, and `AstarInadmissible.m`. Each function must return a path, which is a vector of column indices into the landmarks matrix `World.Landmarks`. Since the robot's starting configuration is always at index 1, and the goal configuration is always the last column in this matrix, the first and last entries of any valid path must be 1 and m , respectively.

To implement your searches, you will need a priority queue. For this we have provided a Matlab implementation of a heap. (A heap is a standard way to implement a priority queue; ask if you're not sure what this is.) See the file `heapExample.m` for demonstrations on how to use this.

The task

This assignment consists of several parts. It is possible to use the same code to complete the majority of these parts, so be sure to read the entire assignment before beginning.

1. [9 points] Using `pa1_matlab_a.m`, implement the potential field evaluation function in `getPotentialValue.m`. If done correctly, the search should find the goal in `sphereWorld0.xml` and `sphereWorld1.xml`. In general, a potential field that accomplishes this will possess the following characteristics:
 - (a) Have an attractive potential that's decreasing towards the goal. For this assignment, the attractive potential will be just the Euclidean distance from the center of the robot to the goal point.

- (b) Be infinite for any invalid positions, such as where the robot collides with an obstacle. Use Matlab's `inf` constant to represent infinity.
- (c) Attach a penalty to being too near to any obstacles so as to steer the robot around them. For each obstacle, let x be the closest distance from the edge of the robot to the edge of that obstacle. If $x > 1$ the obstacle has no effect on the robot. Otherwise, the obstacle's repulsive potential is defined as

$$\frac{1}{(1+x)^2} - \frac{1}{4} \quad (1)$$

While this equation may seem obscure, the intuition is relatively simple. In order to properly visualize the heatmap we chose not to let the repulsive potential approach infinity as the robot gets close to obstacles. This is why we use $\frac{1}{(1+x)^2}$ instead of $\frac{1}{x^2}$. (Remember, however, that it is important to set the value of *invalid* positions explicitly to `inf`.) Also, for this assignment we want the obstacles to only have a local effect on the robot. Thus we only consider obstacles at most 1 unit distance away from the robot and define our repulsive function to decay smoothly to reach a value of 0 when $x = 1$. The total repulsive potential is defined as the sum of the repulsive potentials exerted by each of the obstacles.

Figure 1 shows this potential field on the `sphereWorld0.xml` and the corresponding path found using greedy search. You can use it to verify your results. Note that one of the parameters we did not specify is the tradeoff between the attractive and the repulsive potentials, i.e. the parameter α where

$$\text{Potential value} = \text{Attractive potential} + \alpha \times \text{Repulsive potential} \quad (2)$$

You should experiment with the different values of α , and find a value that works well for `sphereWorld0.xml` and `sphereWorld1.xml`. (Hint: There is a range of values of α that will work, but in the potential field shown in Figure 1, we used $1 \leq \alpha \leq 10$.) What value did you choose?

Remember, you only need to make the potential function work for the **spherical robot**; making a valid potential function for the robot arm will be significantly more difficult. Once your search works on `sphereWorld0.xml` and `sphereWorld1.xml`, using the same value of α , run it on `sphereWorld2.xml`. What happens?

2. [7 points] Using `pa1_matlab_b.m`, implement best first search in the configuration space. At every search step, the algorithm will choose the point from the fringe that will bring it closest to the goal and test if that point is the goal. If it is the goal, the search routine is done. If it is not the goal, the children of that point should be added to the set of fringe nodes. You must also obey the following guidelines:

- (a) Use the heuristic derived from the Euclidean distance from a point to the goal.²

²Even though the robot arm configuration is defined in terms of angles of the joints, the *Euclidean distance* between any two configurations $C = (\theta_1, \theta_2, \dots, \theta_n)$ and $\hat{C} = (\hat{\theta}_1, \hat{\theta}_2, \dots, \hat{\theta}_n)$ is still defined as $\sqrt{\sum_i (\theta_i - \hat{\theta}_i)^2}$. The code we provide enforces that none of the arm joints will ever complete a full circle, so all angles will always stay in the range $[-\pi, \pi]$.

- (b) Never expand a node twice, even if you are expanding it along a second search path different from the one on which it was first expanded. Be careful with this. A node may be added to the queue twice but should be expanded only the first time you pop it off the queue. Again, you are encouraged to make use of the provided heap implementation to make your task easier.³

Your search implementation must be general, and so work for **both** the spherical robot **and** the robotic arm. Run your code on all six provided worlds.

3. [**3 points**] Implement uniform-cost search for PRMs. At every search step, the agent will choose the point from the fringe that will result in the smallest total travel cost from the start node to the current node along the current path as measured by the Euclidean distance. All other guidelines are the same as above.
4. [**7 points**] Implement A* search for PRMs. At every search step, the agent will choose the point from the fringe that will result in the smallest expected travel cost from the start node to the goal as measured by the Euclidean distance. Use Euclidean distance as the heuristic for the distance from the current node to the goal. All other guidelines are the same as above.
5. [**2 points**] Implement A* search with an inadmissible heuristic, which you can use for testing purposes. Use the inadmissible heuristic $100 \cdot h$ where h is the heuristic derived by measuring the Euclidean distance from a state to the goal.
6. [**2 points**] In a few sentences, compare the performance of the different searches you have run. How do the four PRM searches compare in terms of the speed of the algorithm and the optimality of the solutions? In this context, *optimal* refers to how short a distance the arm moved in configuration space (not workspace) given the set of points provided. What happens when you introduced the inadmissible heuristic?

Deliverables

The programming assignment can be done in groups of up to 3 students. Please turn in just one copy per group. Remember, the **only** files you should modify to complete this assignment are the potential field function and the four search functions. Your code should be clear and commented. You should print out and turn in the following:

1. `getPotentialValue.m`
2. `bestFirstSearch.m`
3. `uniformCostSearch.m`
4. `AstarSearch.m`
5. `AstarInadmissible.m`
6. Any other code (e.g., helper functions) that you write

³Note that the lecture notes use a different way than this of handling repeated states. In the lecture notes, we assumed that the priority queue implicitly deals with repeated states by only keeping around the copy of the state which has the lowest priority. However, this is not the case for the queue implementation provided here, so it's up to you to explicitly check every node before expanding it to ensure it has not been expanded before.

7. For the potential fields search, turn in printouts of the output figures (the ones similar to Figure 1) generated by the search for the spherical worlds `sphereWorld1.xml` and `sphereWorld2.xml`. No need to turn in your solution for `sphereWorld0.xml`.
8. For the PRM search, turn in printouts of the four-tile figure (showing all search results) for all the different worlds, except for `sphereWorld0.xml`. There should be five printouts. This is what your results for `sphereWorld10.xml` should look like:

