CSCI–4190 Introduction to Robotic Algorithms, Spring 2002
Assignment 1: out January 18, due February 8

Motion planning

In this assignment, you will implement a simple motion planner using an approximate cell decomposition approach with a quadtree representation. There will also be some written questions regarding your implementation and on related motion planning topics.

Your program should do the following:

- Read the problem description from files.
- Display the workspace obstacles and boundary.
- Create and display the configuration space representation of the world.
- Plan a path for the robot from the given start to goal position. The result of each iteration of this motion planner as well as the planned path should be displayed.

More details on these items appear in the following sections.

I am only supporting the CS department SunOS (UNIX) computing environment for this assignment. Your program must compile and run on this platform. I will be supplying support code for:

- reading the problem description from files
- geometric representations and operations (based on the CGAL computational geometry library)
- graphical display (based on GLUT and OpenGL)

Details of the support code and the written questions will be available in a forthcoming handout. The remainder of this handout will describe the basic algorithms you will need to implement for this assignment.

Assumptions

We will make the following assumptions in this assignment:

- The robot cannot rotate, so its configuration may be specified by the two dimensional point (x, y).
- The robot will be modeled as a convex polygon.
- The world boundary will be convex (but not necessarily rectangular).
- All obstacles will be modeled as convex polygons, and they may overlap.

Creating the configuration space representation

Since both the robot and the obstacles are convex, we can use an algorithm based on the convex hull procedure to generate the configuration space obstacle. If we assume that the reference point on the robot is the origin of the robot frame, then we can compute the configuration space obstacle by the following algorithm:

- Let $\mathcal{P}$ be a list of points, initially empty
- For each vertex of the obstacle, $\vec{O}$ with respect to the world frame:
  - For each vertex of the robot, $\vec{R}$ with respect to the robot frame:
    - Add the point $(\vec{O} - \vec{R})$ to $\mathcal{P}$
- Compute the convex hull of the points in $\mathcal{P}$

The CGAL library provides support for representing points and polygons and for computing the convex hull (among other things).

Note that the world boundary is different than an obstacle, so you will need to create its configuration space representation differently.

Creating a quadtree representation of the configuration space

A quadtree is a data structure useful for representing spatial information because it can provide increased resolution where it is necessary. The concept behind quadtrees extends to arbitrary dimensions; in three dimensions, they are called octrees, and in m-dimensional space, they are called 2^m-trees.

A quadtree is a tree; each node is either a leaf node or has four child nodes. Each node is associated with a rectangular region of the configuration space. These rectangles are aligned with the coordinate axes to make geometric computations simple. For purposes of representing configuration space a rectangular cell will be labeled:

- "empty" if it is free of configuration space obstacles,
- "full" if it is contained entirely within a configuration space obstacle, or
- "mixed" if it intersects part of a configuration space obstacle.

We start with a single rectangle which contains the region of configuration space of interest. If a rectangle is "mixed," we divide it equally into four subrectangles which will be associated with the four child nodes. Thus "empty" and "full" rectangles can be leaf nodes of a quadtree at any depth, but "mixed" rectangles can only be leaf nodes at the maximum depth (i.e. they have not yet been subdivided).

The objective in motion planning with approximate cell decompositions is to find a *channel* of adjacent cells (i.e. rectangles) from the start cell (i.e. the cell containing the start configuration) to the goal cell. Ultimately, we want to find a channel that consists of only "empty" cells; such a channel is called an "E-channel."

For this assignment, start with the following algorithm to find an "E-channel."

- Create a depth 0 quadtree, i.e. a single rectangle containing the region of configuration space that is of interest.
- Repeat:
  - If the start or goal configuration is in a "full" cell, then return failure.
  - If the start and goal configurations are not in "empty" cells, then subdivide all "mixed" cells,
  - Otherwise (i.e. the start and goal configurations are in "empty" cells)
    - Search for an E-channel connecting the start to the goal cell.
    - If found, return success.
    - Otherwise, subdivide all "mixed" cells.

  Until maximum quadtree depth is reached.
- Return failure.

You may improve upon this algorithm in your implementation if you wish.

Creating an adjacency graph

Quadtrees are a hierarchical (tree) representation of space; to search for an E-channel, we want an adjacency graph — a graph in which each rectangle (from the leaf nodes of the quadtree) is a node and there is an edge between the nodes of adjacent rectangles.

Assume that adjacent rectangles must have an edge in common, i.e. we do not consider diagonal connectivity. Note that a large rectangle may have multiple smaller subrectangles adjacent to it on any given side. It will probably be easiest to create the adjacency graph and create the quadtree representation of the configuration space at the same time.

Searching the adjacency graph

Use the A* search algorithm to search the adjacency graph to find the shortest path from the start to the goal configuration. I suggest you use the following formulation:

1. Let OPEN be a list initially containing the start node
2. Let CLOSED be a list, initially empty
3. If OPEN is empty then return failure
4. Remove the node on OPEN with minimum $f()$, let this node be N
5. Add N to the CLOSED list
6. If N is the goal node, the return success
7. For each child (i.e. neighbor) N' of N:
    a. If N' is on the OPEN list, then update its $f()$ value if necessary
    b. If N' is on the CLOSED list, then do nothing
    c. Otherwise, add N' to the OPEN list
8. Go to step 3

A* is essentially a best-first heuristic search, where "best" means minimum estimated cost to reach the goal. This cost is represented by the $f()$ value of a node:

$$f(n) = g(n) + h(n)$$

where $g(n)$ is the cost to reach node n from the start node and where $h(n)$ is an estimate of the distance remaining to reach the goal from n. The above algorithm assumes that the heuristic function $h()$ is *admissible* and *monotonic*. An admissible heuristic will never overestimate the distance to the goal; most admissible heuristics are monotonic. (If the heuristic is not admissible and monotonic, then step 7 must be changed so that if N' is on the CLOSED list, it must be removed put back on the open list if the new $f()$ value is lower than the old $f()$ value.)

You should use the straight line distance from the node to the goal as your heuristic. This heuristic is admissible because the actual robot path cannot be shorter than the straight line distance. Any heuristic (in particular straight line (or Euclidean) distance) that obeys the triangle inequality is monotonic.

Generating a path

After you have found an E-channel, you must still generate a path for the robot. There are a number of ways to generate a path, some simpler than others and some that produce a shorter path than others. In general, these approaches produce a path consisting of a sequence of line segments.

You may use any approach you wish to generate a path. Here is one of the simplest: go from the start configuration to the center of its cell, then go to the centers of each cell in the channel until you get to the goal cell, and then go from the center of the goal cell to the goal configuration.