600.436 Algorithms for Sensor Based Robotics: Project Final Report

Dynamic Motion Planning

Anand Malpani
anandmalpani@jhu.edu

Vishwa Parekh
vishwaparekh@jhu.edu

Abstract

In an ideal dynamic motion planning environment, we have static and moving obstacles. We can assume that we have complete information about position of the static obstacles and no information whatsoever about the moving obstacles. The moving obstacles can be of any size, and can move in any random direction, with any random velocity. In this report, we propose a method to plan the motion of a robot in such an environment using sampling based motion planning and Potential Functions.

Problem

1. **Problem Statement:**
   In this setting, in addition to static obstacles O_1, O_2, ...,O_n, there are several moving obstacles M_1,..., M_k. The objective is to compute a path from an initial configuration to a goal configuration that not only avoids collisions with the static obstacles but also with the moving obstacles.
   Here we assume that size and position of the static obstacles is known and nothing is known about the moving obstacles. The robot has a laser sensor which can sense obstacles closer than a threshold distance at regular intervals from 0 to 2π.

2. **Description of the environment:**
   The environment consists of the following entities:
   - *Static Obstacles*
     - Irregular polygons (concave or convex) with vertices specified in a *static_obstacles.txt* file (passed in as a command line argument)
     - The polygons are formed using triangles so that OpenGL can draw the concave obstacles.
     - Color: Black

   - *Moving obstacles*
     - Circular shaped of randomly selected and initialized through code values
     - Color: Red
     - Number of obstacles - specified as a command line argument (default = 3)
     - Start Position, Radius, Direction and Speed - Start Position, Radius, Direction and Speed values are randomly generated and assigned for every obstacle.
     - Start Position is checked for non conflicting position with other obstacles (static and movign both) and the robot's position.
     - These obstacles have a random motion and they move in their heading directions which are set during initialization. On encountering other obstacles (static and moving) they change their direction depending on their changeTheta parameters which change the heading direction by this specific amount.
     - At any point in time, robot is not aware of any information on this kind of obstacles.

- *Robot*
    - Shape: Circular
    - Color: Blue
    - Radius: Randomly generated through code
    - Start Position: Randomly generated through the code, checked for non conflicts with goal and position of the obstacles and can be changed during execution through user events.

- *Goal*
    - Shape: Circular region
    - Color: Green
    - Radius: Randomly generated through code.
    - Position: Randomly generated through the code, checked for non conflicts with the position of the obstacles and can be changed during execution through user events.

Methods

<u>Step 1</u>: Sampling the points in the configuration space to create the Probabilistic Road Map (PRM)

The first step of our approach is to sample the configuration space into a number of valid sample configurations. We use Probabilistic Roadmap (PRM) Motion Planning to sample the points. For this, we randomly generate a point in the available configuration space. Then, we check by placing the robot at the point and considering collision with all the obstacles. After this is verified, we store this point into our array/vector/list of points in the configurations space. We keep doing this until we are able to sample 'N' (a threshold set in the code) number of points.

In order to check for collision with the obstacles,

```
for every obstacle,
      for every line segment of the obstacle,
      Equation of line is (y-y1)/(x-x1) = (y2-y1)/(x2-x1).
      Using this equation, 'm' points on the line are used to check if they lie
      within the circle formed by robot at that point. For this we check:
```

$$dist(\text{pt.on line, robot\_center}) > \text{radius of robot}$$

<u>Step 2</u>: Forming a connected graph using the points generated in step 1 i.e. connecting the PRM

First, we add the goal to the list of vertices generated above. Now for every vertex in this list, we try and form an edge with 'K' nearest vertices to this particular point. In order to form an edge we check whether a straight line drawn between the two points collides with any static obstacle or not. After this we check by considering robot at 'k' points along the edge and check whether the robot collides with any static obstacle at these 'k' points. When both these conditions are satisfied, an edge between the two points is added to the graph.

Step 3: Calculating possible paths from any point generated in Step 1 to goal using Dijkstra's algorithm.

We start with the goal position.
First we mark all the vertices as unvisited vertices and give each of them a weight - MAXINT and give the goal a weight of 0. A list of parent vertices is also maintained which is initialized with MAXINT as the parent of all the vertices.

```
Current Vertex is set to be the goal.
Repeat until list(unvisited vertices) is empty.
    For all unvisited neighbours (v1) of the current vertex, their weight is
    computed as per:  weight(current vertex) + length(edge between current vertex
    and v1) and is assigned to the neighbour v1. Update the parent of v1 to current
    vertex.
    Put current vertex into the list of visited vertices.
    From the remaining unvisited vertices the one with the least weight is made the
    current vertex.
After this is done, we get a list of weights for all the vertices and every vertex has
its parent, through which it can reach the goal i.e. given any particular vertex, we
just need to keep going to each vertex's parent to reach the goal.
```

Step 4: Connect the current position of the robot to the optimal point on the PRM.

For every vertex in the graph, calculate `distance_robot_goal = distance(robot, vertex) + weight of vertex`.
Now pick the smallest `distance_robot_goal` which provides us a collision free path between robot and goal via this vertex (could be the goal vertex as well). The condition check for the collision between the robot and this vertex is the same as done in step 2.

Once the PRM has been setup and the nearest vertex to the current robot position has been found on the PRM, it's time to move the robot towards the goal in an optimal manner. Thus we call our `moveRobot()` routine which is described below.

Step 5: Move the robot in the environment using the PRM and Potential Functions in conjunction.

We can have robot in any of the three scenarios:

- Moving from one vertex to another on the PRM.
- Moving obstacles are in the range of the sensor; hence the robot is using potential functions to move away from these sensors.
- Robot is in transit motion from current point (not on the graph) towards the optimal vertex found on the PRM.

Different flags are used to move from one state to another. We enter directly into the transit stage with the robot as we are off the PRM initially and then we start moving towards the optimal point on the PRM.
Flags used - `isPotential`, `inTransit`
When the robot is moving according to the Potential function `isPotential` is set.
When the robot is moving according to the Transit function `isTransit` is set.
Otherwise, robot follows Dijkstra's path and moves on the PRM

1. **Move along Dijkstra's path**

   We set the next vertex as the parent of the current vertex. In our case, the edges can be quite long. For this we divide the path into steps of 0.1 units each and take one step at a time. At every step, we check if we are close to the next vertex and on reaching closer than a threshold value we set the next vertex as the current one and get its parent for getting the next path. After every step, we also check using laser scanner, whether any moving obstacle is in range of the sensor. If any moving obstacle is found, `isPotential` flag is set.

2. **Moving obstacles are in the range of the sensor, hence the robot is using potential functions to move away from these sensors.**

   Since we are using potential functions to move away from moving obstacles, we will not reach a local minima, unless there actually is no way out of that particular position. This is a reason why we chose to implement potential functions to move away from moving obstacles.

   Here, for a range of 20 angles along the circle, laser scanner is used to find the distance of the closest obstacle along that direction. After this, if the distance is lesser than a 'maximum' threshold, we calculate repulsive delta change in position as:

   $$delta = ((1/\max\_threshold) - 1/(dist(obstacle, robot))) * 1/(dist(obstacle, robot)^2 * grad(robot, obstacle)$$

   Using the above formula, we calculate the repulsive delta change in position for all the angles at which an obstacle (static or moving) is found. An upper bound is place on this repulsive delta change so that the robot does not take large jump steps while getting repelled from the obstacles.

   Now, there are situations where the robot is stuck in a crunch situation where moving obstacles (more than 1) are charging into it. In order to come out safely from this kind of situations, we develop an artificial goal in an open direction, the direction in which robot is not being repelled by any obstacle (obtained while reading the laser sensor), and then set the attractive delta change in position in this direction as same as the value of the magnitude of total repulsive delta change.

   Now while moving away from obstacle, if our robot has moved away from all the moving obstacles, we change the mode to transit mode where the robot has to move towards the next vertex through which it can reach the goal in the minimum number of steps.

3. **Robot is in transit motion from current point (not on the PRM) towards an optimal vertex on the PRM**

   We find this point as explained in Step 4. We divide the path into small steps of 0.1 units each and at each point on the edge we check for moving obstacles within the laser sensor range and revert back to the potential function mode (if any moving obstacles are found). After reaching the specified vertex on the PRM, we switch to the Dijkstra's mode and move on towards the goal by looking up the parent vertices.

The above is a step-wise description of the algorithm that we have implemented for the problem statement introduced before. A brief description of the experiments we ran this algorithm on are listed.

Experiments and Results

Experiment 1: Check whether the complete environment has been setup properly with all the entities and that their respective properties are being observed as described earlier in this report.
In this experiment we checked:

- whether static obstacles are being drawn correctly, i.e the image that comes up in the graphics interface is the image we expect the program to show.
- The goal is placed into the environment with no collision with the static obstacles.
- Robot is placed into the environment with no collision with the static obstacles and goal.
- To check whether the moving obstacles are placed not on top of each other or any other static obstacle or the robot and whether the randomness is correctly introduced and whether they behave correctly when colliding with other obstacles.

In order to check these condition, the environment was run a number of times, using 3 different scenes and number of moving obstacles ranging from 3 to 15, and checked for correctness.
Results: After vigorous testing on various kinds of possibilities and improvements, we got our environment ready for the experiment.

Experiment 2: To check if the edges are being formed correctly.
Results: The edges were being formed properly and with enough space for the whole robot to move without touching any static obstacle

Experiment 3: To check Dijkstra's algorithm functioning.
Results: We checked Dijkstra's by printing out the parent, weight for every node and its Euclidean distance to its connected nodes, their weights, etc. and then checked if the robot was moving along an expected path or not. We found that Dijkstra's was working correctly.

Experiment 4: To check the transiting function. The robot was initialized at different places in the environment.
Results: The robot would go the optimal vertex that would take it to the goal throught the shortest path possible. This was confirmed by putting robot in positions where the goal was directly reachable and the optimal vertex selected was the goal itself.

Experiment 5: To check Potential Functions. We checked for potential function and generated different kinds of local minima for the robot.
1. Two obstacles move towards the robot in opposite directions and their repulsive potentials cancel each other.
2. For multiple obstacles moving towards robot, trying to push the robot into the corner or concavity
Results: In these cases, the artificial goal creation which generated the attraction gradient force that was useful in providing the way out for the robot. There were cases in which the robot could not get away from such a situation where it was surrounded by obstacles from all directions. The robot after trying to sneak out using artificial goals, gives up in such cases.

Finally, for testing all the pieces of the code together we had three static obstacle scenarios created. These were sufficient for rigorous testing as we had all the other available options of setting the robot start position, goal start position and the number of moving obstacles. We tested for many different scenarios in this manner. Also, as the initialization would randomize the robot and goal positions, many possible scenarios were created and tested.

Discussion

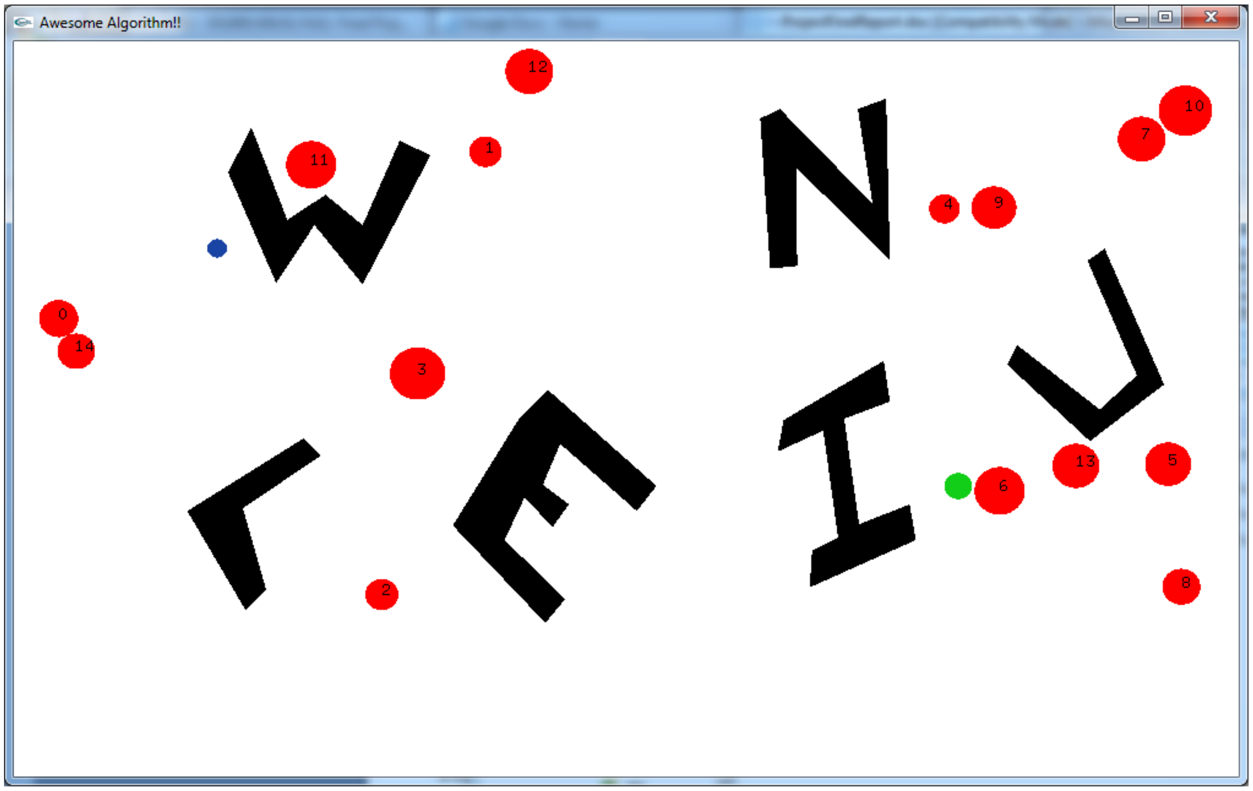
As the entire algorithm consisted of various parts we went step-by-step implementing everything. During each step of implementation we came across various challenges.
While setting-up the environment we started off with checking only for intersection of the circular entities with the edges of the polygons and forgot to consider the case that the circular bodies could be completely enclosed within the polygons. Initially, we were also checking for bounding box and polygon intersection instead of circle-polygon intersection which was implemented later on.
As PRM had been implemented in Assignment 4 incorporating it was trivial and didn't require much of an effort.
The Dijkstra's implementation, however, was tricky as we realized that we weren't storing the parent vertices initially and using the least weight vertex in the neighbourhood of the current vertex to move towards the goal. This mistake was realized when our robot started oscillating between two vertices indefinitely. After this, we stored parent vertices and were able to solve this bug.
During testing the algorithm we also noticed that during the transit scenario, the robot would many a times choose the closest vertex on the PRM which would take it away from the goal initially. This was happening because we were using the euclidean distance of the vertices only and not their weights assigned by the algorithm. Finally, we modified our approach by using a summation of the weight of the vertex (an indication of the distance from the goal) and the euclidean distance from current position of the robot. This approach also helped us in smoothening our path to the goal.

When implementing potential functions, we were under the assumption that because of moving obstacles, we will never run into a situation of getting into a local minima whenever a path would be possible. While working around and experimenting with our code, we found out that in some cases, the total repulsive potential due to multiple obstacles would cancel out each other, and these obstacles would slowly keep coming towards the robot and then as soon as they touch the robot, the robot jumps a huge distance. In order to counter this, we implemented an artificial goal/attractive potential which would act in the direction where there are no obstacles repelling the robot. We here assume that this would lead robot to an obstacle free region which would then provide it with an opportunity to reach the goal. This worked well in most cases except for the cases where the robot chooses a direction which it thinks is free of obstacles, but leads it into concavity of some static obstacle. In these cases, sometimes the robot escapes, and sometimes it gets trapped surrounded by moving obstacles on one side and static on the other. We are planning to make this dynamic part of the solution more effective by combining our potential function and transiting function to deal with moving obstacles.

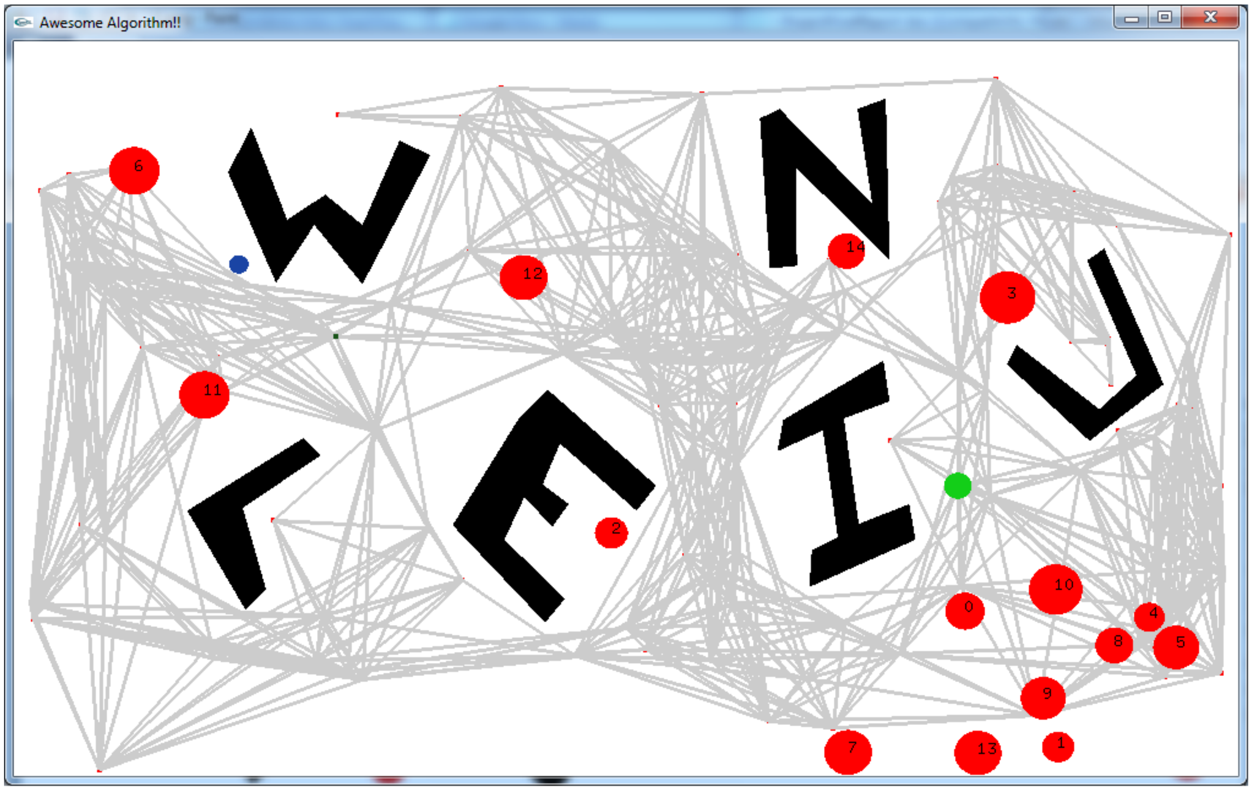
Conclusion

To conclude with, we feel that combining PRM with potential function was very effective in solving motion planning in dynamic environment. Since using PRM, gave us an efficient and quick solution to planning the motion for robot in a static environment. Implementing potential functions over this to deal with moving obstacles reduced the probability of robot being stuck at a point in a local minima because of obstacles constantly in motion.

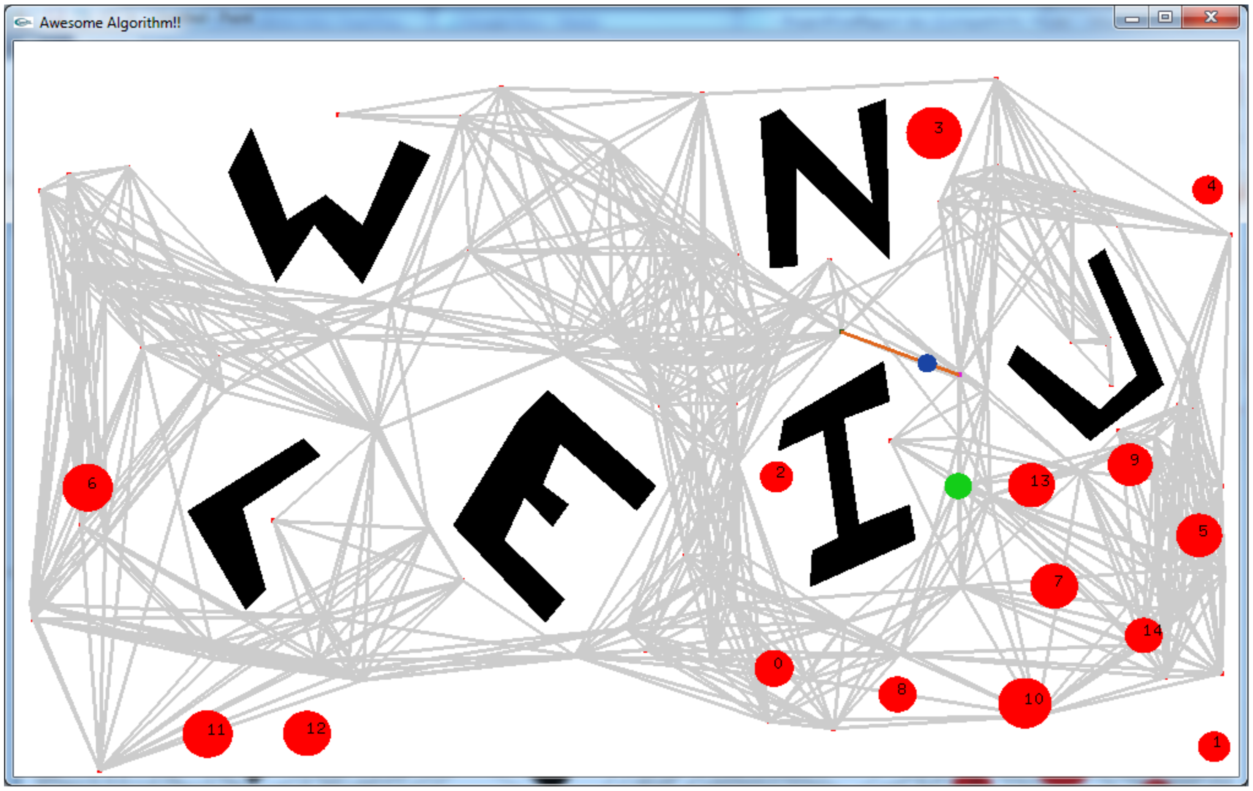
Figures



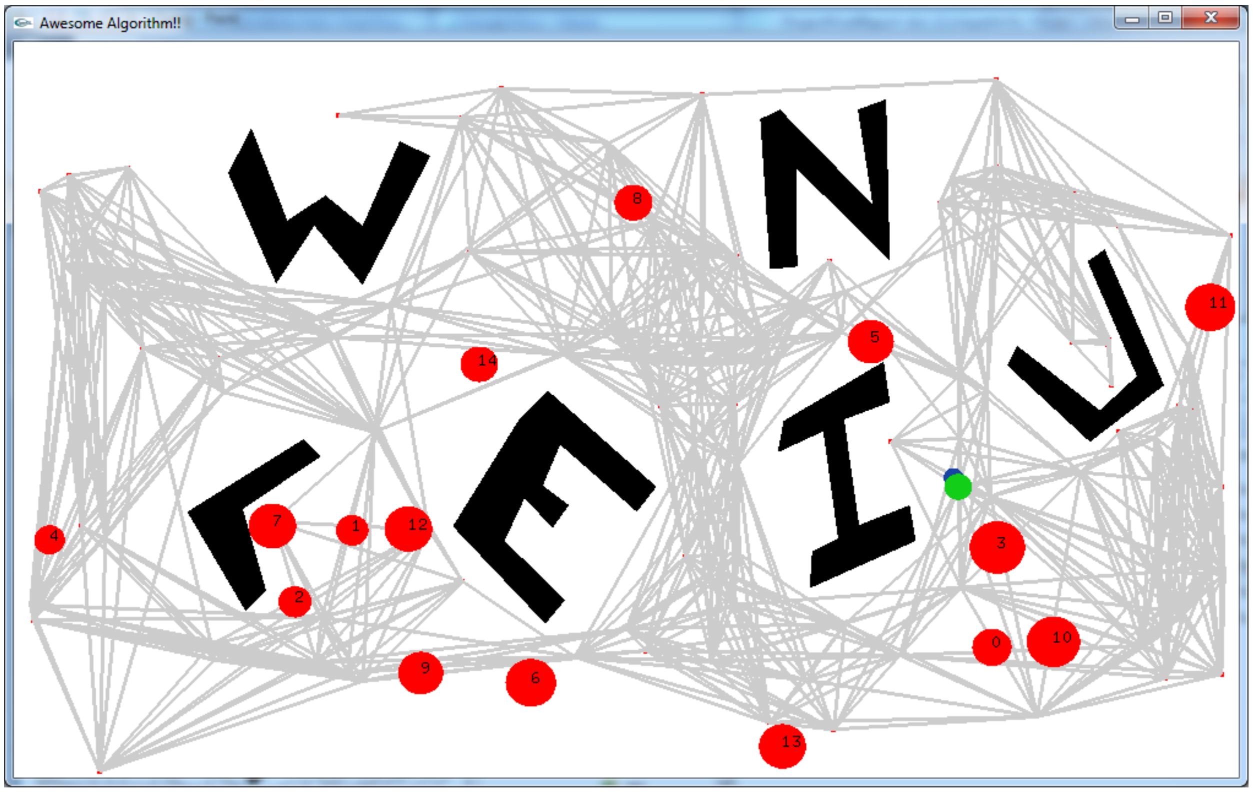
Initial Configuration
(blue circle - robot, green circle – goal)
(red circle - moving obstacles, black polygon – static obstacle)



PRM Graph
(edges in gray, vertices in dull red)



Moving on Dijkstra's path



Reached Goal Region

Resources:

http://ugrad.cs.jhu.edu/~amalpani/sensor/sensor.html