# Assignment 3: Path planning and object recognition

ETH Zurich

Due: Monday, 02.12.2013 at 16:00

You are outside the building, safe, but something is not right. You check your pockets and you look around. Something is missing, but you cannot remember what it is. ”What could I have lost?” You trail back your thoughts. You cannot remember, but you want to make sure that you don't leave anything important behind. Unfortunately, it's too dangerous for you to go back into the building yourself. Instead, you want your robot to search the building and report what it sees. Can you make sure that your robot does the job properly and come back out of the building before the building collapses?

## 1 Path planning

### 1.1 Background

Path planning is the method for producing a continuous motion that connects a start configuration and a goal configuration while avoiding collision with known obstacles. Obstacles are often described in a 2D or 3D map while the path is described in robot's configuration space. In mobile robot path planning with a differential drive robot, we often assume that the robot is holonomic and has a point mass. This simplifies the configuration space to be identical to 2D projection of physical space if the obstacles in the space is inflated by the radius of the robot to compensate for the change in the robot representation.

One way to search for a path is by applying a graph search alogrithm. To do so, the configuration space must be converted into a graph, and a popular method is decomposing the space into grid cells. Each cell is then a node in the graph, and two neighboring cells form an edge in the graph. The connectivity between two neighbouring cells can be either four-connected, in which a node forms an edge with the node's top, down, left, and right neighboring nodes, and eight connected, in which the node forms additional four edges to its diagonal neighbors.

A* search algorithm [1] is a best-first search algorithm which explores the path with the lowest expected total cost first. The expected total cost $f(n)$ of a node $n$ is computed by

$$f(n) = g(n) + \epsilon * h(n), \tag{1}$$

where $g(n)$ is the path cost to $n$ from the starting node, $h(n)$ is the heuristic cost to the goal from $n$, and $\epsilon$ determines the weight between $g(n)$ and $h(n)$. In turn, $g(n)$ is

$$g(n) = g(n') + c(n, n'), \tag{2}$$

that is the path cost $g(n')$ of $n$'s neighboring node $n'$, and the traversal cost $c(n, n')$ of getting from $n'$ to $n$. The traversal cost $c(n, n')$ can be computed by the Euclidean distance between two nodes $n$ and $n'$ or approximated by the Manhattan distance. Heuristic cost $h(n)$ is often computed as the Euclidean distance between $n$ and the goal node, and the optimal path is found when $\epsilon = 1$.

Using the expected cost $f(n)$, A* algorithm, show in Algorithm 1, works as follows: Given a starting node $n_{start}$ and a goal node $n_{goal}$, the algorithm initializes the search by setting

$g(n_{start}) = 0$ and adding the neighboring nodes of $n_{start}$ to a heap $S_{open}$. $S_{open}$ contains all the neighboring nodes of the visited nodes $S_{closed}$. Among the nodes in $S_{open}$, the algorithm searches for the next node $n_{next}$ with the smallest expected cost $f_{cost}(n_{next})$ to the goal and repeatedly adds the neighboring nodes of $n_{next}$ to $S_{open}$. When the next node $n_{next}$ is the goal node $n_{goal}$ or all the reachable nodes $S_{open}$ in the space have been searched, then the algorithm terminates its search. If the algorithm reaches the goal node $n_{goal}$, we can build the optimal path $P_{optimal}$ by trailing back the previous nodes until we reach the goal.

**Algorithm 1:** A* search algorithm [1]

```
Data: $n_{start}$: the starting node of the robot
      $n_{goal}$: the goal node
      $G = (V, E)$: the graph of nodes $V$ and edges $E$
Result: $P_{optimal}$: the optimal path from $n_{start}$ and $n_{goal}$
begin
    $S_{closed} = \emptyset$
    $S_{open} = \{n_{start}\}$
    $g_{cost}(n_{start}) = 0$
    $f_{cost}(n_{start}) = g_{cost}(n_{start}) + h_{cost}(n_{start}, n_{goal})$
    $n_{start}^{previous} = \varnothing$
    $P_{optimal} = \emptyset$
    while $S_{open} \neq \emptyset$ or $n_{goal} \notin S_{closed}$ do
        $\{n_{next} \mid n_{next} \in S_{open},\ f_{cost}(n_{next}) \leq f_{cost}(n)\ \forall n \in S_{open}\}$
        $S_{open} = S_{open} - \{n_{next}\}$
        $S_{closed} = S_{closed} + \{n_{next}\}$
        if $n_{next} = n_{goal}$ then
            $n_{current} = n_{goal}$
            while $n_{current} \neq \varnothing$ do
                $P_{optimal} = \{n_{current}\} + P_{optimal}$
                $n_{current} = n_{current}^{previous}$
        else
            forall $\{n_{neighbor} \in V \mid \exists (n_{next}, n_{neighbor}) \in E\}$ do
                if $n_{neighbor} \notin S_{open}$ then
                    $g_{cost}(n_{neighbor}) = g_{cost}(n_{neighbor}) + d(n_{neighbor}, n_{next})$
                    $f_{cost}(n_{neighbor}) = g_{cost}(n_{neighbor}) + h_{cost}(n_{neighbor}, n_{goal})$
                    $n_{neighbor}^{previous} = n_{next}$
                    $S_{open} = S_{open} + \{n_{next}\}$
                else
                    $g_{cost,temp}(n_{neighbor}) = g_{cost}(n_{next}) + d(n_{neighbor}, n_{next})$
                    $f_{cost}(n_{neighbor}) = g_{cost}(n_{neighbor}) + h_{cost}(n_{neighbor}, n_{goal})$
                    if $f_{cost,temp}(n_{neighbor}) < f_{cost}(n_{neighbor})$ then
                        $g_{cost}(n_{neighbor}) = g_{cost,temp}(n_{neighbor})$
                        $f_{cost}(n_{neighbor}) = g_{cost}(n_{start}) + h_{cost}(n_{start}, n_{goal})$
                        $n_{neighbor}^{previous} = n_{next}$
end
```

### 1.2 Task

Write an A* path planner algorithm. You can either extend your localization package or create a new package.

**Note**: You do not have to be able to localize to plan a path. You will be given a starting position and goal positions. Start the assignment by writing a path planner that can find a path between two points on an occupancy grid map. Make sure that you can plan a path between two points before extending it to find the shortest path visiting n points or controlling the robot to follow the path.

## 2 Grading

You will be given a map as a png file and three destinations within the map. The task is for the robot to plan the optimal path to visit these goal locations then come back to the starting location.

### 2.1 In-class demonstration (30 points)

- Compute and visualize the optimal path in RViz (10 points)
  - Able to find and display the optimal path between two nodes (6 points)
  - Able to find and display the shortest path visiting all nodes (4 points)
- Follow the optimal path (5 points per path)
  - Maximum deviation from the ideal path (2.5 points)
    - 2.5 points for the team with the smallest deviation
    - for every additional 0.5$cm$, -0.25 points
  - Number of peaks (2.5 points)
    - 2.5 points for the team with the fewest peaks (A peak has more than 1$cm$ deviation from the ideal path)
    - for every additional peak: -0.25 points

### 2.2 Software quality (30 points)

- Choice of abstraction and relations (9 points)
- Correctness of implementation (12 points)
- Extendibility and reusability (6 points)
- Comments and documentation, including "README" (3 points)

## References

[1] Hart, P. E., Nilsson, N. J., Raphael, B. 1968. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". IEEE Transactions on Systems Science and Cybernetics SSC4 4 (2): 100107.