Visibility Graph

Part I: Building Configuration Space

I have used the Star Algorithm to compute the 2-dimensional configuration space $Q$ starting from the workspace $W$ and the robot shape $R$. If we assume that the robot and the obstacles in the workspace are convex polygons, the Star Algorithm computes the obstacles in the workspace using the normal vectors of each side of the polygons. I have used the Matlab function $\text{norm} = \text{c}\_\text{norm}(\text{tang}, \text{dir})$ that returns the normal vector for each side of a polygon. The function is the same used in the Homework #3.

The algorithm sorts the sides of the robot and of one obstacle by increasing angles of normal vectors. Thereafter it builds the new convex polygon adding all the sides of the robot and of the obstacle in the calculated sequence. I define the vertex with the higher $x$-value as reference point on the robot and the vertex with the lower $x$-value as reference point on the obstacle. Since the robot’s reference point can touch the obstacle reference point by a given robot’s configuration $q$, the corresponding vertex of the obstacle in the configuration space must be at the position $q$.

Since the obstacles in the configuration space can intersect, I have implemented an algorithm to find all the possible intersections. I have written a new Matlab function $[v, X] = \text{check\_int}(V, V_f)$ that checks whether two edges in the plane intersect (parameter $v = 1$), overlie (parameter $v > 1$) or do not have any common points (parameter $v = 0$). Thereafter, I can define new obstacles in the configuration space $Q$ that have at most one common edge, which can never belong to the Visibility Graph.

When the program starts, push the button “Draw Obstacle” to draw an obstacle and set with the left mouse button the vertices of the obstacles within the white area (robot workspace $W$). To end the process click with the right mouse button somewhere. To
Part II: Building and searching the Visibility Graph

In order to generate the Visibility Graph I have implemented an algorithm that connects all vertices in the configuration to all other vertices to which they have line-of-sight. In order to check if the line-of-sight does not intersect any edge, I use the Matlab function 
\[ [v, X] = \text{check\_int}(V, V_f) \]. Moreover, in order to check that a line-of-sight does not completely lie inside an obstacle I have used the implemented Matlab function 
\[ \text{inside} = \text{vertex\_check}(X1, Y1, X2, Y2) \], that checks if an edge lies inside or outside an angle defined by two other edges.

After building the Visibility Graph, I have implemented the A* algorithm to search the shortest path between the start configuration and the goal configuration. I have
implemented a Matlab function  

```
[path, error] = astarv(edg, q_start, q_goal)
```

where if the parameter `error` is 1 this means that there is no path connecting the start and the goal configuration (as shown in the picture below).

For the A* algorithm the following functions are defined:

- `star(n)`: represents the set of vertices that have line-of-sight with \( n \);
- `c(n_1, n_2)`: represents the length of the line-of-sight connecting the vertices \( n_1 \) and \( n_2 \);
- `g(n)`: represents the total length of back pointer from \( n \) to \( q_{start} \);
- `h(n)`: represents the heuristic cost function which returns the estimated cost of shortest path from \( n \) to \( q_{goal} \); I have used the Euclidean distance;
- `f(n)=g(n)+h(n)`: is the estimated cost of shortest path from \( q_{start} \) to \( q_{goal} \) via \( n \).

The flow of the A* algorithm is the same as in Homework #4.

I have implemented this algorithm because it was very interesting working on finding intersections between obstacles in the configuration space. In order to register a movie of the path planning, there is a check box “Movie” that has to be checked before pushing the button “Calculate Path”.