ASEN 5519 - ALGORITHMIC MOTION PLANNING
HOMEWORK #3
JOE MICELI
23OCT2020

PROBLEM 1

(a) Complete Planning Algorithm - An algorithm that is guaranteed to find a path if one exists. I a path does not exist, the complete algorithm will exit in failure (it is guaranteed to not produce a false positive).

(b) Optimal algorithm - An algorithm that produces the "best" solution (in this case, path) based on the metrics used to evaluate each solution. Examples of an optimum could be shortest ~~[illegible]~~ path, minimum energy required, ~~[illegible]~~ fastest time to find a path, etc.

(c) A wavefront planner is complete only wrt to the discretization used to obtain the wavefront. This is because discretization of a space will unavoidably cause some loss of the space. So if any portion of ~~all~~ the valid path(s) in a space are lost during discretization, the algorithm will not be complete.

~~Similarly, wavefront planners are not always optimal but could potentially be used to find an optimal path if the C-space it was operating in was designed to account for the metrics being used to evaluate an "optimal solution." An example~~

A wavefront planner is not an optimal planning algorithm. It is not guaranteed to produce the best solution if multiple exist. In these cases, ~~[illegible]~~ cells ~~[illegible]~~ have multiple equally valued neighbors and a wavefront planner randomly selects one of them. The resulting path lengths will be equal but this could have drastic real-world differences on the system (i.e. energy required by each node).

PROBLEM 2

(a)

$$\nabla U(q) = \nabla U_{att}(q) + \nabla U_{rep}(q)$$

$$\nabla U_{att} = \begin{cases} \zeta (q - q_{goal}) & , \quad d(q, q_{goal}) \leq d^*_{goal} \\ \dfrac{d^*_{goal}\, \zeta (q - q_{goal})}{d(q, q_{goal})} & , \quad d(q, q_{goal}) > d^*_{goal} \end{cases}$$

$$\nabla U_{rep} = \sum \nabla U_{rep_i}$$

$$\nabla U_{rep_i} = \begin{cases} \eta \left(\frac{1}{Q^*} - \frac{1}{d_i(q)} \right) \frac{1}{d_i^2(q)} \nabla d_i(q) & d_i(q) \leq Q_i^* \\ 0 & d_i(q) > Q_i^* \end{cases}$$

$$d_i(q) = \min_{c \in QO_i} d(q, c)$$

$$\nabla d_i(q) = \frac{q - c}{d(q, c)}$$

Algorithm:

while $\nabla U(q(i)) > \varepsilon$:

  $q(i+1) = q(i) + \alpha(i) \nabla U(q(i))$

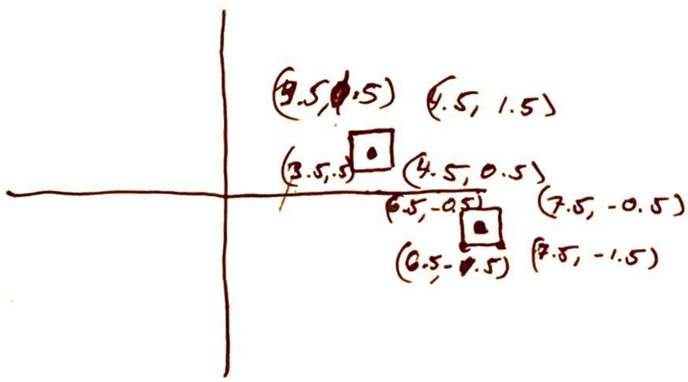
  $i = i+1$

end while

[Psuedo-Code]

Obstacles:

- Need to specify vertices
- Method of returning the point on the obstacle closest to current config, q_i
  - ↳ Could use primitives to generate list of boundary points
    - ↳ Could check every boundary point for the closest one to current configuration
- Centered @ (4, 1), (7,-1) lengths of 1




```
GetDistanceToObstacle( q )
 mindistance = GetDistance(q, boundery_points[0])
for point in boundery_points
    distance = GetDistance(q, point)
    if (distance < min_distance)
        min_distance = distance
        min_point = c
```

Get Boundary Points (

```
for (min_x ≤ x ≤ max_x )
  boundary_points.push_back( x, P1(x))
for (min_y ≤ y ≤ max_y)
  b_p.push_back( P2(x), y )
for (min_x ≤ x ≤ max_x)
  b_p.push_back ( x, P3(x))
for (min_y ≤ y ≤ max_y)
  b_p. push_back ( P0(y), y)
```

i) PLEASE SEE ATTACHED IMAGES/CODE

ii) I chose d^*_{goal} as 5.0 which is a relatively large value for the workspace size but I wanted the attractive portion of the gradient to become large early on so obstacles ~~could~~ nearby would not cause the robot to get stuck.

Similarly, I wanted the robot to be "allowed" to get close to obstacles so I set Q_i^* to 1.0 for both $i=1$ and $i=2$.
Trial and error ensured that the values did not produce a local minimum.

iii) PLEASE SEE ATTACHED IMAGES/CODE

iv) 9.87916 m is the length of the path.

v.) No, different values for d^*_{goal} and Q_i^* would almost always produce different paths & path lengths. These values impact how the robot behaves as it approaches an obstacle and how it avoids the obstacle. It can also produce local minima that prevent the robot from ever getting to the goal.

[IMPACT OF PARAMETERS]

$\varepsilon = 0.25$ → Fixed in problem statement

$\zeta = 0.1$

$\eta = 0.1$

$d^*_{goal} = 5.0$

$Q^* = 1.0$

Gradient is 0 but robot stops 2.5m away from goal

↳ Local minimum!

$\zeta = 2.0$

$\eta = 0.5$

$d^*_{goal} = 5.0$

$Q^* = 1.0$

Robot is able to get close to obstacles w/out getting stuck because attractive gradient is strong enough

(b)

i) The values I settled on were obtained after dozens of trials. The algorithm is extremely sensitive to changes in any parameter: $\alpha, \varepsilon, \zeta_0, \eta, d^*_{goal}, Q^*$

Ultimately, I was not able to find parameters that ~~produced~~ "collision free paths" for either workspace. My approach was to increase Q_i^* so that more obstacles would influence the gradient (even if it was far away). I chose a lower value for d^*_{goal} because I wanted obstacle avoidance to take presidence over achieving the goal. In workspace 1, this worked for avoiding obstacles but the robot became stuck in a local minimum.

In Workspace 2, the robot achieved the goal but collided w/ obstacles along the way. This planning problem was particularly challenging because the initial conditions produced a gradient descent direction that sent the robot straight into an obstacle.

[The exact values chosen for either workspace are listed in the README.txt provided with the code submission]

ii.) PLEASE SEE ATTACHED IMAGES

iii) PLEASE SEE ATTACHED README

iv.) No, both of these planning problems were very sensitive to changes in d^*_{goal} and Q_i^*. Slight modifications to either parameter drastically changes the potential function and therefore, the path produced from a gradient descent algorithm.