RBE550: Motion Planning

Project 4: Rigid Body Motion Planning

Student 1: Jeel, Chatrola
Student 2: Dev, Soni

Theoretical Questions:

Solution 1

(a) **RRT***

- The Main Idea behind RRT* is to Modify RRT algorithm such that as we sample more and more points the path generated gets closer to optimal path. (Asymptotically Optimal)
- In RRT we just sample points and perform collision check before adding them to our graph, Meanwhile in RRT* we add additional metrics/check to improve the performance of our algorithm
- **Cost Metric** - In RRT*, we keep track of a cost to reach a given node in order to use this information later on to improve the path and explore nodes that can lead to be better paths.
- **Node Rewiring Step** - In RRT*, we add a rewiring step added after the standard extension step. During the rewiring step, the algorithm identifies neighbors within the defined radius and re-configures the tree to use the lowest-cost paths.
- **Tree Rewiring Step** - In RRT*, when we add a new node we check neighbouring nodes, if cost of reaching the new node is lower than the current cost of reaching the node we rewire the tree. (This ensures that as the tree continues to grow we get closer to optimal path)

(b) **Informed RRT***

- Informed RRT* is an extension of RRT* algorithm that aims to improve RRT* by using information about the problem domain to bias the exploration towards the goal and, in some cases, find a solution faster. Because RRT* finds optimal path from the initial state to every state in the planning domain which is inefficient for single query problems.
- Informed RRT* works just like a RRT* until it finds a solution, but after that it samples from a subset from an admissible heuristic.
- **Biased Exploration:** Informed RRT* incorporates heuristics to guide the exploration. This bias allows the algorithm to focus its efforts on areas of the configuration space that are more likely to lead to the goal, potentially speeding up the search process.
- **Tree Size**: In RRT*, the tree expands uniformly in all directions. Informed RRT* focuses the tree expansion towards the goal, leading to a smaller tree size, which can improve the algorithm's performance, especially in high-dimensional spaces.
- **Adaptability:** Informed RRT* can be adapted to the specifics of the problem by choosing the right heuristics and information for a given problem. This adaptability can make it more effective in cases where you have domain knowledge that can be leveraged.
- **RRT* vs Informed RRT*** Even thought Informed RRT* is an improvement/extension of RRT* we cannot concluded that it is strictly better than RRT*. Because its performance is highly dependent on factors/parameters like the quality of heuristics and the nature of the problem.
- Cases where RRT* performs better than Informed RRT* -
  - Case where we don't have additional information except the planning problem as cited by the Informed RRT* paper *"In situations where they provide no additional information (e.g., when the informed subset includes the entire planning problem), Informed RRT* is equivalent to RRT*".*[1]
  - Case when our heuristic is not very accurate, let's say it may underestimate the cost of reaching the goal from some states in the narrow passage. This could cause Informed RRT* to discard those states, even though they could potentially lead to a better solution, whereas in RRT* there is no heuristic and it explores the space equally well so it will find a better solution.

[1] J. D. Gammell, S. S. Srinivasa and T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," IROS 2014.

Solution 2

(a)
- Configuration Space $\implies SE(2)$ or $\mathbb{R} \times S^1$
- Control Space $\implies u_l, u_r \in \mathbb{R}$
- State Space $\implies x, y, \theta \in \mathbb{R}$

(b) Euler Approximation for a differential drive robot given $(x, y, \theta) = (0, 0, 0)$ and control input as $(u_l, u_r) = (1, 0.5)$ for 1 second.
So,

$$\begin{bmatrix} \frac{x_{t+1} - x_t}{dt} \\ \frac{y_{t+1} - y_t}{dt} \\ \frac{\theta_{t+1} - \theta_t}{dt} \end{bmatrix} = \begin{bmatrix} \frac{r}{2}(u_l + u_r)\ cos(\theta) \\ \frac{r}{2}(u_l + u_r)\ sin(\theta) \\ \frac{r}{L}(u_r - u_l) \end{bmatrix} \implies \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \begin{bmatrix} \frac{r}{2}(u_l + u_r)\ cos(\theta)(dt) + x_t \\ \frac{r}{2}(u_l + u_r)\ sin(\theta)(dt) + y_t \\ \frac{r}{L}(u_r - u_l)(dt) + \theta_t \end{bmatrix}$$

Now, dt = 1 sec and t = 0 and t+1 = 1, we get

$$\begin{bmatrix} x_1 \\ y_1 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} \frac{r}{2}(1 + 0.5)\ cos(0)(1) + 0 \\ \frac{r}{2}(1 + 0.5)\ sin(0)(1) + 0 \\ \frac{r}{L}(0.5 - 1)(1) + 0 \end{bmatrix} = \begin{bmatrix} \frac{r}{2}(1.5) \\ 0 \\ \frac{r}{L}(-0.5) \end{bmatrix} = \begin{bmatrix} 0.75r \\ 0 \\ -0.5\frac{r}{L} \end{bmatrix}$$

This Actually leads to a incorrect answer because of large step size in Euler approximation. If we want to improve the accuracy the step size needs to be very small compare to range of time we want to compute. In this case it should be around 0.1.

Solution 3

We assume that given (x, y, θ) for a given robot we can compute vertices A,B,C,D using rotation matrices. Additionally we assume that obstacles are given as a set where each obstacle's vertices are known in (x, y) pairs.

Algorithm 1 Collision Checking Algorithm using AABBs

function COLLISION_CHECKING_2D_ROBOT(robot_location, obstacles)

robot_location $\rightarrow (x, y, \theta)$ ▷ Robot's position
robot_vertices $\rightarrow (A, B, C, D)$ ▷ Each vertices are defined with a (x,y)
Obstacle_set $= \{Obs_1, Obs_2, Obs_3, ..., Obs_n\}$ ▷ Obstacle Set with n obstacles
$Obs_i = \{a_1, a_2, a_3, ...a_m\}$ ▷ Obstacle i with m vertices each vertex defined with a (x,y)

$r^x_{robot} = [max(A_x, B_x, C_x, D_x) - min(A_x, B_x, C_x, D_x)]/2$ ▷ radius of the AABB in x axis for the robot
$r^y_{robot} = [max(A_y, B_y, C_y, D_y) - min(A_y, B_y, C_y, D_y)]/2$ ▷ radius of the AABB in y axis for the robot
$C_{robot} = (min(A_x, B_x, C_x, D_x) + r^x_{robot}\ ,\ min(A_y, B_y, C_y, D_y) + r^y_{robot})$ ▷ Centre of AABB for the robot

$r^x_{obs_i} = [max(a_{1x}, a_{2x}, ...) - min(a_{1x}, a_{2x}, ...)]/2$ ▷ Radius of AABB in x axis for the i^{th} obstacle
$r^y_{obs_i} = [max(a_{1y}, a_{2y}, ...) - min(a_{1y}, a_{2y}, ...)]/2$ ▷ Radius of AABB in y axis for the i^{th} obstacle
$C_{obs_i} = (min(a_{1x}, a_{2x}, ...) + r^x_{obs_i}\ ,\ min(a_{1y}, a_{2y}, ...) + r^y_{obs_i})$ ▷ Centre of AABB for the i^{th} obstacle

Collision_Matrix = [ ] ▷ To track with obstacles is the robot colliding with

for all i in Obstacle_set **do**
  if $(C^x_{robot} - C^x_{obs_i} > r^x_{robot} + r^x_{obs_i})$ or $(C^y_{robot} - C^y_{obs_i} > r^y_{robot} + r^y_{obs_i})$ **then**
    Collision_Matrix[i] = 0 ▷ Collision not detected (denoted as 0)
  else
    Collision_Matrix[i] = 1 ▷ Collision detected (denoted as 1)

Programming Component:

Solution 1 -

Solution 2 -