NORTH SOUTH UNIVERSITY

Introduction to Robotics

CSE495A

Home Work 3

AL IMRAN

2212602642

Ans to the Que NO. 1

ⓐ A* requires more computational resources Because

① Constructing agrid produces a graph whose size grows exponentially with respect to the dimension of the configuration space

ⓘⓘ A* must perform search over this graph and its perfromance depends on the resolution and finer resalution causes the graph to become very heavy. (Large).

Thus A* has high computational load because it must maintain and Update a priority queue of many graph nodes.

~~because~~

On the other hand

① In differential flatness method generates trajectories using simple algebraic mappings.

ⓘⓘ It is sufficient to specify a differentiable trajectory for the flat outputs, and the remaining state variables and control inputs can be analytically determined.

This requires much less computation because no search is performed.

Therefore, A* requires more computational resources.

(b) For obstacle avoidance I would like to use A* algorithm. Obstacles are represented explicitly in a grid where cells are free or forbidden (C_{free} and C_{obj}) and search explores adjacent free cells. The A* algorithm expands nodes according to cost and huristic, producing a collision free geometric path.

On the other hand, Differential flatness does not incorporate obstacles. It only ensure that state and inputs can be recovered from flat outputs and does not include collision information.

Theredor, A* is used for obstacle avoidance because it explicitly handles forbidden regions, while differential flatness does not.

(c) For obstacle free 3D space I would like to use Differential flatness.

For flat systems (like quadrotors) all feasible trajectories can be written as functions of the flat output and these mappings are analytic allowing easy trajectory calculation. The flat outputs allow constructing smooth trajectories without solving a optimal control problem, making the method computationally simple. And another importent thing is the space is obstacle free so the differential flatness is applicable.

onthe other hand in A* algorithm, the approach is resolution dependent and the graph grows bigger exponentially with the number of DOFs. Our problem is in 3D space so that the graph will be much bigger then 2D space. Thus A* is computationall heavy, gives non smooth paths. Since its an obstacles free space its dosen't use full potential of A* algorithms.

Ans to the Que No. 2

From book we get

example 3.2.1

$$\dot{V} = -k_1 \rho^2 \cos^2\alpha - k_2 \alpha^2$$

and, $k_1, k_2 > 0$

we can see that k_1 and k_2 must be positive and in $k_1 \rho^2 \cos^2\alpha$, ρ and $\cos\alpha$ is squared so they must be positive. so that $-k_1\rho^2\cos^2\alpha$ must be negative. We can also see that in $k_2\alpha^2$, α is squared so they also be positive. So that $-k_2\alpha^2$ will be negative. adding 2 negative number results a ngative nuber which proves that $\dot{V} < 0$.

For examples.

let $k_1 = 2$	let $k_1 = 2$	let $k_1 = 2$	let $k_1 = 2$
$k_2 = 1$	$k_2 = 1$	$k_2 = 1$	$k_2 = 1$
$\rho = -2$	$\rho = 2$	$\rho = -2$	$\rho = 2$
$\alpha = -30^\circ$	$\alpha = 30^\circ$	$\alpha = 30^\circ$	$\alpha = -30^\circ$
$\dot{V} = -2\times(-2)^2(\cos(-30))^2 - 1\times(-30)^2$	$\dot{V} = -2\times(2)^2(\cos(30))^2 - 1\times(30)^2$	$\dot{V} = -2(-2)^2(\cos(30))^2 - 1\times(30)^2$	$\dot{V} = -2(2)^2(\cos(-30))^2 - 1\times(-30)^2$
$\dot{V} = -906$	$\dot{V} = -906$	$\dot{V} = -906$	$\dot{V} = -906.$
$\dot{V} = -906 < 0$	$\dot{V} = -906 < 0$	$\dot{V} = -906 < 0$	$\dot{V} = -906 < 0.$

Ans to the Que No 3

Given, $V = \frac{1}{2}x_1^2 + \frac{1}{4}x_2^4$

Here, (i) $V(x_1, x_2) = 0$ for $x_1 = 0$ and $x_2 = 0$

$V = \frac{1}{2}0^2 + \frac{1}{4}0^4 = 0$

(ii) $V(x_1, x_2) > 0$ for $x_1 \neq 0$ and $x_2 \neq 0$

Since x_1 and x_2 will be squared so it will always be positive. so $V(x_1, x_2)$ will be greater then 0.

(iii) $V = \frac{1}{2}x_1^2 + \frac{1}{4}x_2^4$

Given,

$\dot{x}_1 = -x_1 + x_2^3$

$\dot{x}_2 = -x_2 + u$

$\dot{V} = \frac{1}{2} \cdot 2 \cdot x_1 \cdot \dot{x}_1 + \frac{1}{4} \cdot 4 \cdot x_2^3 \cdot \dot{x}_2$

$\dot{V} = x_1 \dot{x}_1 + x_2^3 \cdot \dot{x}_2$

$\dot{V} = x_1(-x_1 + x_2^3) + x_2^3 \cdot (-x_2 + u)$

$= -x_1^2 + x_1 x_2^3 - x_2^3 x_2 + x_2^3 u$

$= -x_1^2 + x_1 x_2^3 - x_2^4 + x_2^3 u$

$= -(x_1^2 + x_2^4) + \boxed{x_1 x_2^3 + x_2^3 u}$

To stabilite the system, we choose a control input u such that $\dot{V}$ is negative definite. By selecting u to cancle the positive term $x_1 x_2^3$, we make $\dot{V}$ negative dethniate.

$x_1 x_2^3 + x_2^3 u = 0$

$x_2^3(x_1 + u) = 0$

For $x_2 \neq 0$, the required control is $u = -x_1$

so that

we can find $$\dot{V} = -(x_1^2 + x_2^4)$$

and control is $u = -x_1$ (Am).

Ans to the Que No. 4

A^* is a label correcting algorithm that is a modified version of Dijkstra's algorithm. The step by step algorithm is:

(i) Initialization: Set the cost of arival $C(q) = \infty$ and $f(q) = \infty$ for all vertices
for initial vertex q_I, set $C(q_I) = 0$, $f(q_I) = h(q_I) \rightarrow$ heuristic
Add q_I to the set Q.

(ii) Select vertex: While Q is not empty, choose the vertex q in Q with the smallest value of $f(q)$.

(iii) Goal check: if $q = q_G$ then return the path

(iv) Exploring neighbor: Remove q from Q
For each neighbor q' of q, compute the tentative cost $\tilde{c}(q') = C(q) + c(q, q')$
if $\tilde{c}(q') < c(q')$ then update
$q'.\text{parent} = q$
$c(q') = \tilde{c}(q')$
$f(q') = c(q') + h(q')$
if q' is not already in Q, add it.

(v) If the goal is never selected, return failure.

cons of A* algorithms:

(i) Resolution dependent:

→ Not guaranteed to find solution if grid resolution is not small enough.

(ii) Limited to simple robots:

→ Grid size is exponential in the number of DOFs.

(iii) A* Depends on having a heuristic that is a positive Understimate. if the heuristic is poor, performance drops.

(iv) The number of grid cells grows quickly, leading to large memory use and slow search.

(v) Assumes the environment is Known, Changing environments (obstacles) require replanning.

Ans to the Q No. 5

Probabilistic Roadmap is the easiest because it is conceptually quite similer to combinatorial planner from the grid based planning. To achive this the steps are:

(i) set n, n = numbes of nodes to sample
For example n = 50 or n = 100

(ii) Sample a node q_i from a probability distribution

(iii) Collision checker: check
If $q_i \in C_{free}$ ⇒ Keep the node
if $q_i \in C_{obj}$ ⇒ dis card

(iv) Repeat (ii) and (iii) Untill we have n nodes. Buid the graph using these nodes.

(v) Run A^* on this graph to find minimum cost path.

The cons of the method are:

(i) The downside of PRM is that finding good solutions may require a large number of samples n to sufficiantly cover the configuration space

(ii) Having too many samples will require a lot of queries of the collision checker, which may be costly

(iii) it can not determine whether a solution does not exist.

(iv) it works only fixed environment. Its not usable in Dynamic environment

Ans to the QueNo. 6

In robotics, there are many cases that the obstacles are Dynamic. In this type of environment Rapidly exploring Random tree planning method are used. To achive this the steps are:

(i) Sample a node q_i

(ii) Check if $q_i \in C_{free} \Rightarrow$ Keep.

  if not discard.

(iii) Creat edge and move along the edge

(iv) check progress to goal

(v) Repeat steps (i), (ii), (iii), (iv) Untill we reached the goal.

Cons of this method:

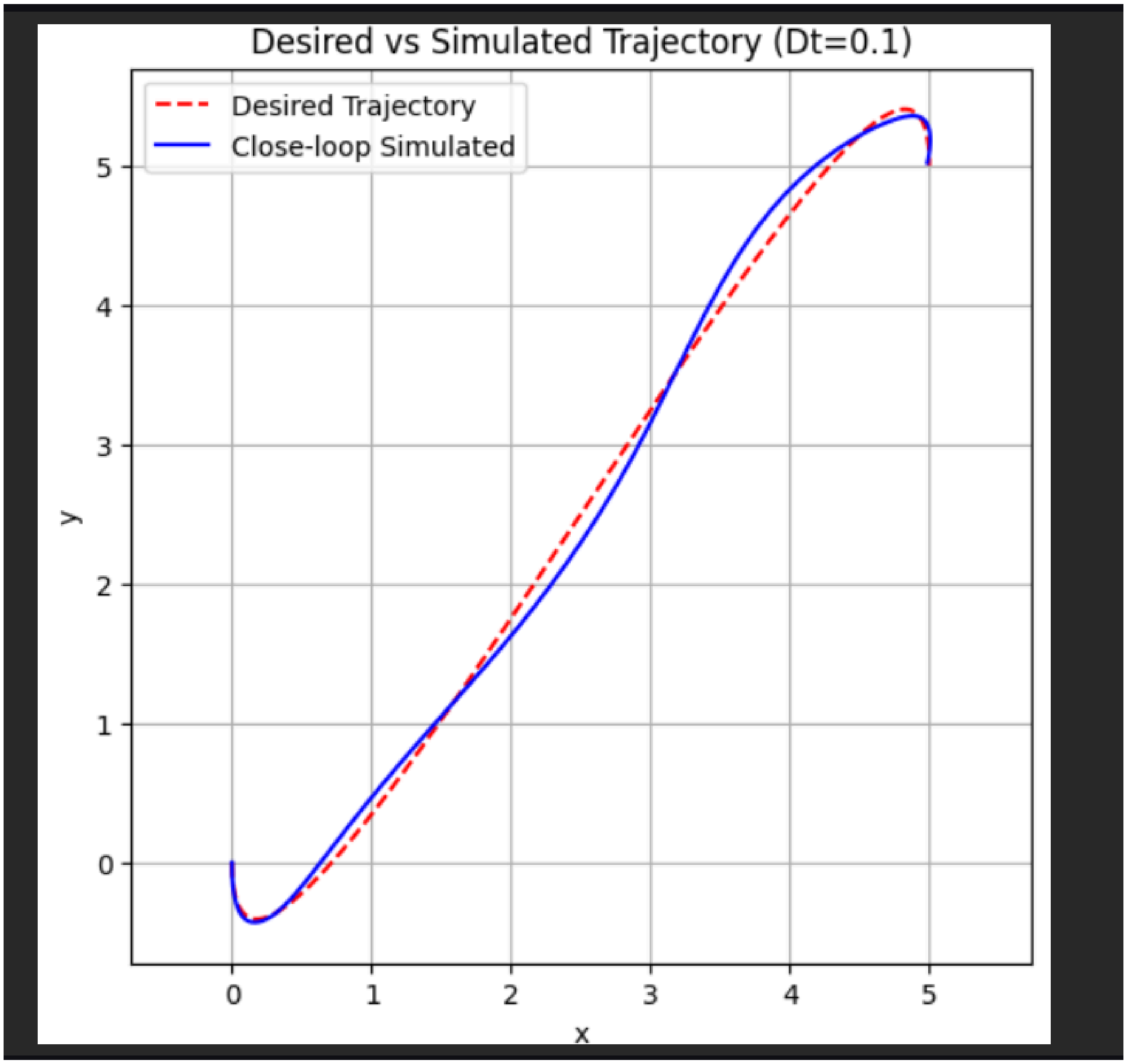
(i) RRT can be arbitrarily bad with non-negligible probability with respect to optimal path cost.

(ii) the path quality is poor then PRM.

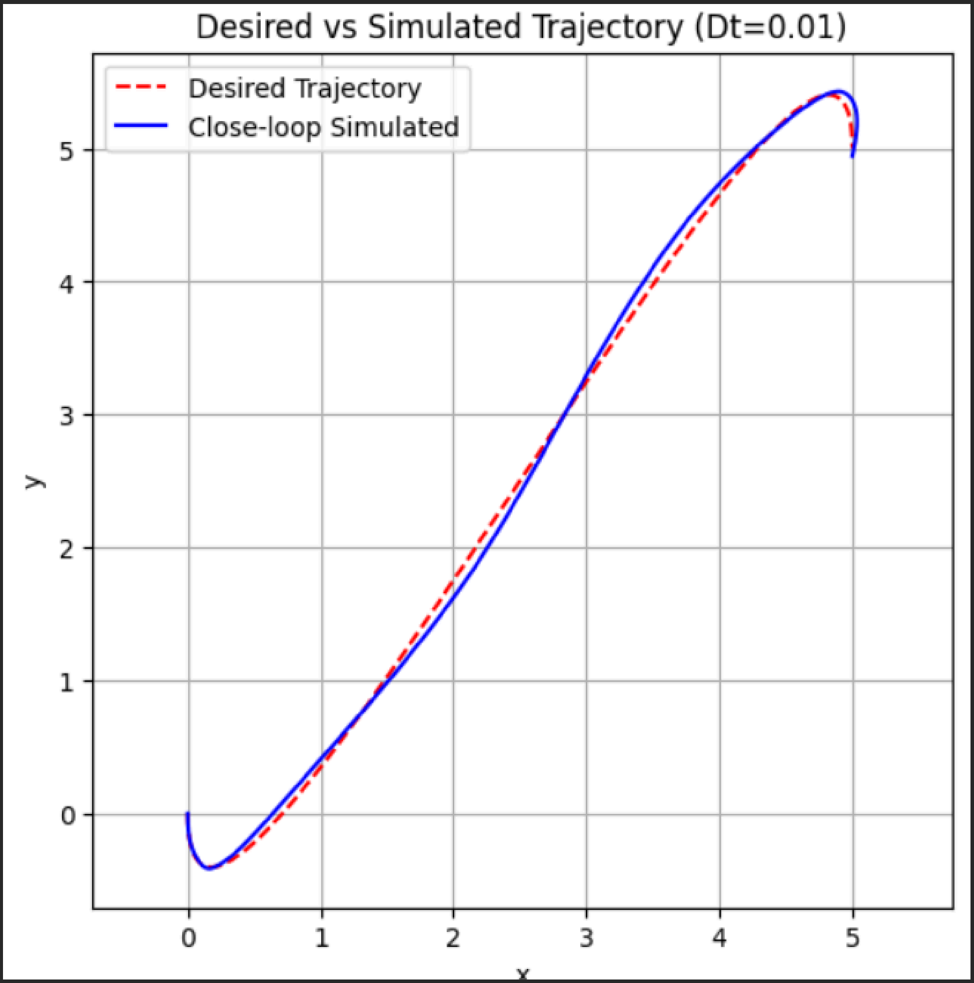
(iii) RRT finds a feasible path quickly but it traps itself by disallowing new better paths to emerge.

Answer to the question no 7

HW3_que7_dt_.1.ipynb



HW3_que7_dt_.01.ipynb


Desired vs Simulated Trajectory (Dt=0.01)
Desired Trajectory
Close-loop Simulated
y
x