North South University

CSE 495: Introduction to Robotics

Summer 2023

Section: 1

Home Work 2

Submitted By:

Name: Tahiat Hakim Himel

ID: 2013075042

Date: 5/11/2023

Submitted To:

Dr. Shahnewaz Siddique

Department of Electrical & Computer Engineering

North South University

Answer to question NO-01

a

Differential Flatness Method: Usually utilizes less computing power as it concentrates on determining trajectories that provide smoother and more effective pathways by satisfying system dynamics. It doesn't require a protreacted search procedure.

A* method: Generally require more computational resources, especially in complex environments, as it searches a discrete grid or graph representation of the environment for an optimal path.

b

For obstacle avoidance, we would use the A* method. It is specially designed to find collision-free paths in environment with obstacles. It considers discrete obstacle representations and efficiently avoids collisions.

C

For movement in an obstacle-free 3D space, the differential flatness method is more appropriate, It generates smooth and feasible trajectories that satisfy system constraints without the need for complex pathfinding algorithms.

Drawbacks of using A* in obstacle-free space is it may introduce ~~unnes~~ unnecessary complexity and computational overhead when obstacles are absent.

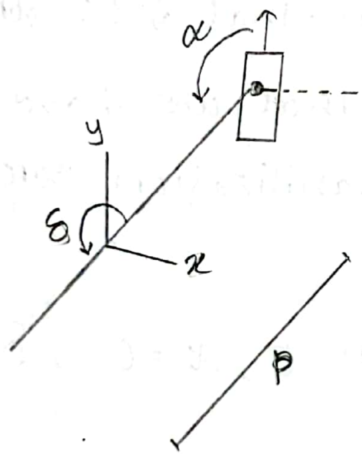
Drawbacks of using differential flatness in obstacle-filled space is Differential Flatness doesn't handle obstacle directly, so it may lead to collisions or require additional collision avoidance strategies.

Answer to question No - 2

Pose stabilization :

Given that,
$$\dot{x}(t) = v(t) \cos\theta(t)$$
$$\dot{y}(t) = v(t) \sin\theta(t)$$
$$\dot{\theta}(t) = \omega(t)$$

~~we~~ where control input are v and ω.
In polar co-ordinates, pose stabilization of a unicycle robot.



where dynamic equations are,

$$\dot{p}(t) = -v(t) \cos\alpha(t)$$
$$\dot{\alpha}(t) = \frac{v(t) \sin\alpha(t)}{P(t)} - \omega(t)$$
$$\dot{\delta}(t) = \frac{v(t) \sin\alpha(t)}{P(t)}$$

By expressing the dynamics in polar form, a Lyapunov stability now easily can performed Lyapunov function,

$$V(\rho, \alpha, \theta) = \frac{1}{2}\rho^2 + \frac{1}{2}(\alpha^2 + k_3\delta^2)$$

for closed loop control law,

$$v = k_1 \rho \cos\alpha$$

$$\omega = k_2\alpha + k_1 \frac{\sin\alpha \cos\alpha}{\alpha}(\alpha + k_3\delta)$$

where, $k_1, k_2, k_3 > 0$

Now, we have to show that $\dot{V} < 0$ for the chosen control inputs. For that we have to show that system is stabilization for lyapunov function,

1) $V(0,0,0) = 0$ for $\rho = 0, \alpha = 0, \delta = 0$

2) $V(\rho, \alpha, \delta) > 0$ for $\rho \neq 0, \alpha \neq 0, \delta \neq 0$ if $k_3 > 0$

3) $\dot{V}(\rho, \alpha, \theta) = \rho\dot{\rho} + \alpha\dot{\alpha} + k_3\delta\dot{\delta}$

$$= -k_1\rho^2\cos^2\alpha - k^2\alpha^2$$

Answer to question No-3

Given nonlinear system,

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

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

using lyapunov function $V = \frac{1}{2}x_1^2 + \frac{1}{4}x_2^4$

Now, we have to find control u for stablize the system,

Choose, $V(x, u) = \frac{1}{2}x_1^2 + \frac{1}{4}x_2^4$

Now, if a system is given $\dot{x} = f(x, u)$, there exists a function $V(x, u)$ such that,

1) $V(x, u) = V(0, 0)$ for $x = 0$ and $u = 0$

2) $V(x, u) > 0$ for $x \neq 0$, $u \neq 0$

3) $\dot{V}(x, u) < 0$ for $x \neq 0$, $u \neq 0$

then, $\dot{x} = f(x, u)$ is stable, then we will design controller u such that conditions 1, 2, 3 are met.

For given Lyapunov function. $X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

1) $X = 0$ then $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\therefore V(0, 0) = 0$

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

$\therefore V(x_1, x_2) > 0$ $[x_1 \neq 0$ and $x_2 \neq 0]$

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

$= x_1(-x_1 + x_2^3) + \dot{x}_2^3 (x_2 + U) < 0$

[for $x_1 \neq 0, x_2 \neq 0$ if $U \neq 0$]

$\frac{d}{dt}\left(\frac{1}{2}x_1^2 + \frac{1}{4}x_2^4\right)$

$= \frac{d}{dt}\left(\frac{1}{2}x_1^2\right) + \frac{1}{4}\frac{d}{dt}(x_2^4)$

$= \frac{1}{2} \cdot 2x_1 \frac{d}{dt}(x_1) + \frac{1}{4} \cdot 4x_2^3 \frac{d}{dt}(x_2)$

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

So,

$x_1(-x_1 + x_2^3) + x_2^3(-x_2 + U) < 0$

$\Rightarrow -x_1^2 + x_1 x_2^3 - x_2^4 + x_2^3 U < 0$

$\therefore U < \dfrac{x_1^2 - x_1 x_2^3 + x_2^4}{x_2^3}$ will result in a stable system where control U used for stabilization.

Answer to question NO-4

A* grid-based search algorithm: A* algorithm is a 2D based shortest path planning algorithm which is also known ~~as~~ as a lebel correcting algorithm. It is a modified version of Dijkstra's algorithm. In Dijkstra's algorithm the goal ~~vector~~ vertex q_G is not taken into account, potentially leading to wasted effort in case where the greedy choice makes no ~~program~~ progress towards the goal.

A* : Minimize cost of arrival and cost to go.
So, A* prioritized based on cost of arrival $c(q)$ plus an approximate cost to go $h(q)$. This provides a better estimate of the total quality of a path than just using the cost of arrival alone.

A* algorithm: $f(q) = c(q) + h(q)$
here, $f(q)$, $c(q)$ and $h(q)$ are respectively estimated cost, cost of arrival and minimum cost to go.

Data $\longrightarrow q, q_G, G$
Result $\rightarrow$ Path

$q_1 \rightarrow$ Initial vertex
$q_G \rightarrow$ Goal vertex
$G \rightarrow$ Graph

Here,
$c(q) = \infty$, $f(q) = \infty$, $\forall q$;
here all the vertex q, cost of arrival and estimated cost are infinite.

$c(q_1)=0.\ f(q_1) = h(q_1)$

for intial vertex q_1, cost of arrival are zero,

For that estimated cost are equal to minimum cost to go.

$Q = \{q_1\}$

while Q is not empty

$q = \arg\min\ q' \in Q\ f(q')$

if $q = q_G$

return path

Q.remove(q)

for $q' \in \{q' \mid (q, q') \in E\}$ do

$\tilde{c}(q') = c(q) + c(q, q')$

if $\tilde{c}(q') < c(q')$ then

q'.parent $= q$

$c(q') = \tilde{c}(q')$

$f(q') = c(q') + h(q')$

if $q' \notin Q$ then

Q.add(q')

return failure.

The algorithm respectively show that enter starting node in graph. Then if the graph is empty, it return failure.

The select node from graph where smallest path value and if node = goal, return path.

Expend node and generate succession, compute g+h for each node

The attach next node to back to point then again go to the step for empty graph.

<u>Cons</u>:

1) In this method if the grid resolution is not enough small, there is no gurantee to find solution.

2) Limitation to simple robots where grid size is exponential in the number of DOFS

Answer to question No 5

Probabilistic Road Map (PRM) : Probalistic Road Algorithm is conceptually quite similar to combinational planners. In particular, the PRM algorithm also generated a topological graph G called a roadmap where the vertics are configurations q in free part of configuration space free and edge connect the vertices. The key insight of the PRM algorithm is that a complete characterization of the free configuaration space can be avoided by sampling configuration q at random then using a collision checker to validate if q ∈ c free.

The algorithm follows -

1) Randomly sample n configurations q_i from the configuration space.

2) Query a collision checker for each q_i to determine if q_i ∈ c free then it is removed from the sample set.

3| Create a graph $G = (V, E)$ with ~~vectore~~ vertics from the sampled configuration $q_i \in C_{free}$. Define a radius r and create edges for every pair vertices q and q' where,

i| $||q - q'|| \leq r$

ii| the straight line path between q and q' is also collision free.

Cons:

1| Insuffician sampling may result in wrong path

2| PRM can be computationally intensive, especially in high-dimensional spaces.

Answer to question NO-6

Rapidly-exploring Random Tree (RRT): RRT is used for single query problem where it grows a tree and rooted at the start configuration and embedded in C_{free}.

This RRT algorithm solves the single query problem by incrementally sampling and building the graph, starting at the initial configuration q, until the goal configuration q_G is needed.

In general, the RRT algorithm begins by initialize a tree $T = (V, E)$ with a vertex at the initial configuration. At each iteration it follows,

1) Randomly sample a configuration $q \in C$
2) Find the vertex $q_{near} \in V$ that is closet to the sample configuration q.

3| Compute a new configuration q_{new} that lies on line conecting q_{near} and q such that the entire line from q_{near} to q_{new} is contained in the free configuration space C_{free}.

4| Add a vertex q_{new} and an edge (q_{near}, q_{new}) to the Tree T.

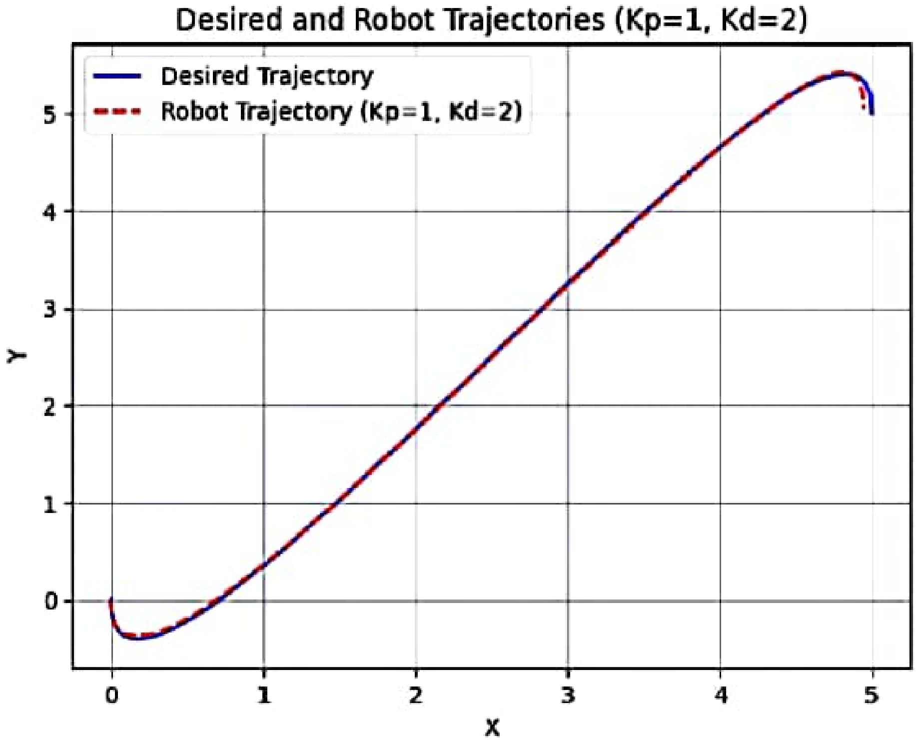
Cons:

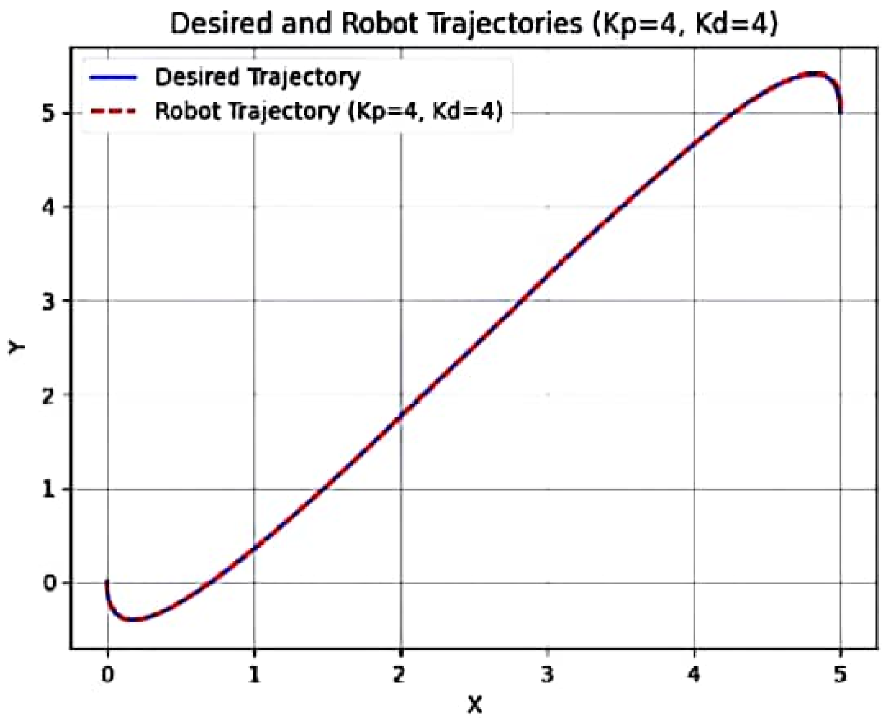
1| RRT can be arbitrary bad with non-negligible probability

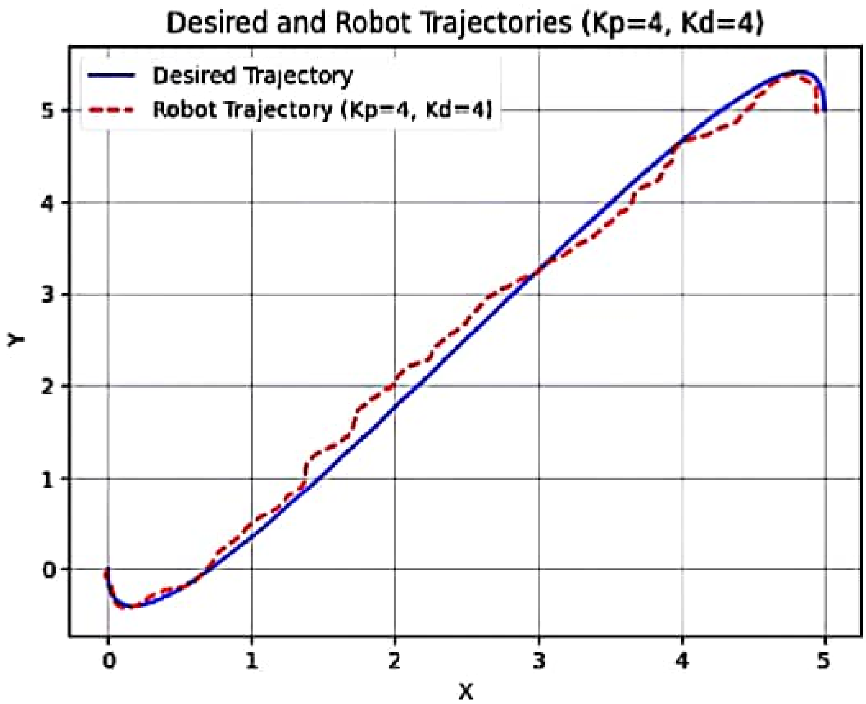
2| The generate suboptimal in only terms of length.

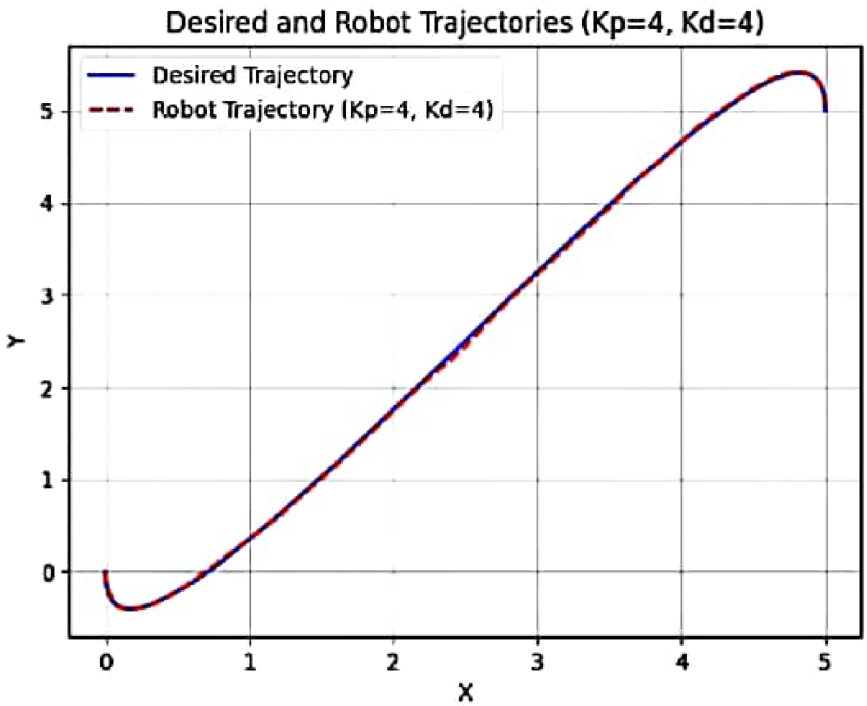
3| Difficulty for handing dynamic obstucles.

<u>Answer to question no - 7</u>






Desired and Robot Trajectories (Kp=4, Kd=4)
Desired Trajectory
Robot Trajectory (Kp=4, Kd=4)
X
Y


Desired and Robot Trajectories (Kp=4, Kd=4)
Desired Trajectory
Robot Trajectory (Kp=4, Kd=4)
X
Y