Ans to the Ques No 1

(a)

The differential flatness method involves solving differential equations that describe the dynamics of the system. The differential flatness method is suited for continious time control and trajectory tracking. It needs to execute in real time.

On the other hand, A* method is used for planning paths. It used to searching through a grid to find the right ppath. The computational complexity of A* depends on the size of the search space, the presence of obstacles. and

Comparing those method, the differential flatness method requires more computational resources due to the need to solve complex differential equations. A* method is only about searching paths and it computational demand depends on the size of grid. Therefore, differential flatness ~~more~~ requires ~~more~~ more resources than A* algorithm.

(b)

Differential flatness models the system as a flat output system, which enables the generation of smooth trajectories. A* is a search based path planning which excels ~~in~~ in finding optimal paths avoiding obstacles.

Comparing those method, A* method would be better for obstacle avoidance. A* method has the ability to find the optimal path, balance path optinality

③

and comutational cost. The differential flatness method is suited for continious time system and effective for non grid environment. A* method can find path that cover long distances and navigate through intricate obstacles, making it ideal when the robot must plan routes over extended areas. A* will be best choice for obstracle avoidance

ⓒ

I would use differential flatness method, for movement in obstacle free 3d space.

For movement in 3d space, which was obstacle free, there is no need for grid-based path planning. As the space are obstacle frees the smoothness and accuracy is important. Differential flatness are perfect for continious environments allowing smoother and efficient path planning. In space without obsticles, differential flatness method simplify the process. The differential flatness method allows for the generation of precise control inputs.

Finally a space without obstacles, the path planning process is simplified with differential flatness.

As I didn't choose A* method,

Drawbacks of A* method:

(1) A* method in open 3d space. will require unnecessary computation as there was no obstacle to avoid.

(2) It's a grid based approach where the environment divided into nodes. But for 3d system, it was ~~more~~ inefficient.

(3) A* method will consume more memory resources and take more time.

(4) A* method resulting more computational time without benefits for 3d space which was obstacle free.

Differential method offers the advantage of continious and smooth control and more efficient for Obstack free 3d environment. A* is not the best option for discrete path planning.

Ans to the Ques No 2

Unicycle robot model represented by,

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

Now, expressing in polar coordinate,

$$\rho = \sqrt{x^2+y^2}$$

$$\alpha = \operatorname{atan2}(y, x) - \theta + \pi$$

$$\delta = \alpha + \theta$$

polar coordinates dynamics equation,

$$\dot{\rho}(t) = -v(t)\cos\alpha(t)$$

$$\dot{\alpha}(t) = \frac{v(t)\sin\alpha(t)}{\rho(t)} - \omega(t)$$

$$\dot{\delta}(t) = \frac{v(t)\sin\alpha(t)}{\rho(t)}$$

Performing Lyapunov stability analysis,

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

Considering closed-loop control

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

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

$k_1, k_2, k_3 > 0$

now taking ~~tani~~ darivative on v,

$$\dot{v} = \rho \dot{\rho} + \alpha \dot{\alpha} + k_3 \delta \dot{\delta}$$

$$= \rho [- v \cos\alpha] + \alpha \left[\frac{v \sin\alpha}{\rho} - \omega \right] + k_3 \delta \frac{v \sin\alpha}{\rho}$$

$$= - k_1 \rho^2 \cos^2\alpha - k_2 \alpha^2 \ \cancel{\cos} < 0$$

for $\rho \neq 0, \ \alpha \neq 0, \ \delta \neq 0$

Ans to the Ques No 3

Lyapunov stability theory:

Given a system $\dot{x} = f(x, u)$, ~~iff~~

iff there exists a function $V(x, u)$ such

that,

① $V(x, u) = 0$ for $x = 0$

② $V(x, u) > 0$ for $x \neq 0$

③ $\dot{V}(x, u) < 0$ for $x \neq 0$

Then the system is stable

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

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

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

Hence, $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

for

① $x = 0$ then $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ for that,

$V = \frac{1}{2} x_1^2 + \frac{1}{4} x_2^4 = \frac{1}{2} 0 + \frac{1}{4} 0 = 0$

$\therefore$ for, $x=0$, $V(0,0)=0$

② for, $x_1 \neq 0$ and $x_2 = 0$

$$V(x_1,x_2) = \frac{1}{2}x_1^2 + \frac{1}{4}x_2^4$$
$$= \frac{1}{2}x_1^2 > 0$$

for, $x_1 = 0$ and $x_2 \neq 0$

$$V(x_1,x_2) = \frac{1}{2}x_1^2 + \frac{1}{4}x_2^4$$
$$= 0 + \frac{1}{4}x_2^4$$
$$= \frac{1}{4}x_2^4 > 0$$

for, $x_1 \neq 0$ and $x_2 \neq 0$

$$V(x_1,x_2) = \frac{1}{2}x_1^2 + \frac{1}{4}x_2^4 > 0$$

③ $\dot{V}(x_1,x_2) = \frac{d}{dt}\left(\frac{1}{2}x_1^2\right) + \frac{d}{dt}\left(\frac{1}{4}x_2^4\right)$

$$= \left(\frac{1}{2}\cdot 2x_1 \frac{d}{dt}x_1\right) + \frac{1}{4}\cdot 4x_2^3 \frac{d}{dt}x_2)$$
$$= x_1\dot{x}_1 + x_2^3\dot{x}_2$$

~~from~~ $= x_1(-x_1+x_2^3) + x_2^3(-x_2+u)$

$= -x_1^2 + x_1x_2^3 - x_2x_2^3 + ux_2^3$

~~the system~~ $= -x_1^2 + x_1x_2^3 - x_2^4 + ux_2^3$

to stabilize the system,

$-x_1^2 + x_1x_2^3 - x_2^4 + ux_2^3 < 0$

$\Rightarrow ux_2^3 < x_1^2 - x_1x_2^3 + x_2^4$

$\Rightarrow u < \dfrac{x_1^2 - x_1x_2^3 + x_2^4}{x_2^3}$ ~~the condition~~

thats the condition to stabilize the system.

Ans to the Ques No 4

A* is a label correcting algorithm that is modified version of Dijkstra's algorithm. In Dijkstra's algorithm the goal vertex q_G is not taken into account, potentially leading to wasted ~~effort~~ in cases where the greedy choice make no progress towards the goal.

While ~~A*~~ Dijkstra algorithm only prioritizes a vertex q based on its cost-of-arrival $c(q)$, A* prioritizeds 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 is grid-based approach which is simple, and fast.

A* algorithm,

$f(q) = c(q) + h(q)$; where, $f(q)$, $c(q)$ and $h(q)$ are respectively estimated cost, cost of arrival and minimum cost to go.

Data : q_I, q_G, G

$q_I \rightarrow$ Initial/start vertex,

$q_G \rightarrow$ Goal vertex

$G \rightarrow$ ~~Goap~~ Graph

Result: path

$c(q) = \infty$, $f(q) = \infty$, $\forall q$

for all vertex q, cost of arrival and cost ~~of~~ estimated cost are infinite.

$c(q_I) = 0$, $f(q_I) = h(q_I)$

for initial vertex q_I, cost of arrival are zero and estimated cost and minimum cost to go are equal.

$Q = \{ q_I \}$

The vertex only contain Initial vertex

While Q is not empty do

$q = \arg\min_{q' \in Q} f(q')$

if $q = q_G$ then

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

return failure

The algorithm shows,

① Enter starting node in graph

(ii) If the graph is empty, return failure

(iii) Select node from graph which has smallest value and if node = goal, return path.

(iv) Expend node and generate succession. Compute (g+h) for each node

(v) attach next node, to back point

(vi) go to (ii)

Cons of A* Algorithm:

(i) Resolution dependent! In that method, 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.

Ans to the Ques No 5

Probabilistic Road Map (PRM) is conceptually quite similar to combinatorial planneres. The PRM algorithm also generates a topological graph G called a roadmap where the vertices are configurations q in the free part of configuration space C_{free} and edges connected to the vertices.

The Algorithm

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

2. Query a collision checker for each q_i to determine if $q_i \in C_{free}$ If $q_i \notin C_{free}$ then It is removed from the sample set.

3. Create a Graph $G = (V, E)$, with vertices from the sampled configurations $q_i \in C_{free}$.

Define a radius r and create edges for every pair of vertices q and q' where (i) $||q - q'|| \le r$ and (ii) the straight line path between q and q' is collision free.

PRM is a multi-query planner, which generates a roadmap (graph) G embedded in the free space.

Cons of PRM

(1) Requires a large number of samples n to sufficiently cover configuration space.

(2) Insufficient sampling may result in wrong path.

~~(3) Required large number of samples to sufficien~~

(3) PRM can be computationally intensive, especially in high-dimentional space.

Ans to the Ques No 6

Rapidly-exploring random tree (RRT) is a single query planner, which grows a tree T, rooted at the start configuration s, embedded in C_{free}.

The RRT algorithm solves problem by incrementally sampling and building the graph, starting at the initial configuration q_I, until the goal configuration q_G is reached.

RRT ~~$~~ algorithm begins by initializing a tree, $T = (V, E)$ with a vertex at initial configuration. At each iteration the RRT algorithm then performs the following steps:

1. Randomly sample a configuration $q \in C$.

2. Find the vertex $q_{near} \in V$ that is closest to the sampled configuration q.

3. Compute a new configuration q_{new} that

lies on the line connecting q_{near} and q such that the entire line from q_{near} to q_{new} is ~~contt~~ contained in the free configuration space C_{free}.

4. Add a vertex q_{new} and edge to the tree.

After a iteration only a single point is sampled and potentially added to the tree.

Cons of RRT:

① ~~RRT~~ can be arbitary bad with non-negligible probability.

② According to this method, generate sub optimal in only terms of length

③ Difficult for handling dynamic obstacle.