Muhammad Rayan Alam 18311006 42
CSE495 Sec 01 Summer-2023
HW-02 Deadline - 06th NOV 2023

Answer-01

Differential Flatness Method: This method exploits the inherent flatness property of dynamic system allowing the trajectory problem to be simplified. It can be computationally less demanding compared to other methods because it formulates the trajectory planning problem in terms of flat outputs, and solving for these outputs can be more straightforward than directly solving for the trajectory.

A*-Method: The A*-algorithm is a pathfinding and graph traversal algorithm that is commonly used in Robotics for path planning. It explores the possible paths and selects the one with the lowest cost.

The computational resources required by A* depend on factors like the complexity of the environment, the size of grid or space being explored and the heuristic used to guide the search. In complex environments, A* may require significant computational resources.

① (b) For obstacle avoidance, the A* method is often more suitable. A* is designed for pathfinding in environment with obstacles and can adapt to dynamic changes in environment. It explores the space efficiently and can find a collision-free path by considering the cost associated with each potential path. Differential flatness may not inherently address obstacle avoidance, as it focuses more on the system's dynamic properties

① ⑥ Movement in Obstacle-free 3D space: In an obstacle-free 3D space, the choice between the two methods depends on the specific characteristics of the problem and the desired trajectory. Differential flatness might be advantageous ~~of the problem~~ in this case because it can provide smooth and feasible trajectories that adhere to the dynamic constraints of the system. However, it's crucial to consider the specific requirement and constraints of the application

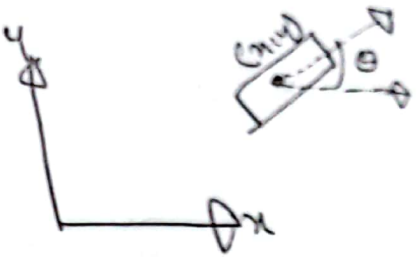
Drawbacks:

① Differential flatness Method Drawback:

This method may not be applicable to all types of dynamic systems, and finding a suitable coordinate transformation for differentiable flatness can be challenges in some cases. Additionally, it might not handle complex environments with obstacles as effectively as methods designed specifically for path planning.

A* - Method Drawbacks: While A* is effective for obstacle avoidance, it's main ~~back~~ drawback is that it may become computationally expensive in large and complex environments. The efficiency of A* depends on the heuristics used and the configurations of the environment and in other cases, it may not scale well

Ques 02: Pose Stabilization: Consider a robot that is modeled by the unicycle robot model (differential drive robot model) represented graphically in the fig 3.[illegible]



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

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

$$\dot{\theta}(t) = \omega(t)$$; where the control inputs are the robot speed v and rotational rate ω.

Show that $\dot{V} < 0$ for the chosen control inputs (equation 3.11 in the notes).

Eq 3.11 is closed-loop control law:

$$v = k_1 \rho \cos\delta$$

$$\omega = k_2\delta + k_1 \frac{\sin\delta\cos\delta}{\delta}(\delta + k_3\delta)$$ where $k_1, k_2, k_3 > 0$

continued

Show all the steps in the math

Soln:

Lyapunov function V is defined as:

$$V(x,y,\theta) = \frac{1}{2}(x^2+y^2) \quad ; \dot{V}<0$$ (dealt with it later)

or, $\dot{V} = \frac{\partial V}{\partial x}\dot{x} + \frac{\partial V}{\partial y}\dot{y} + \frac{\partial V}{\partial \theta}\dot{\theta}$ [After derivation]

or, $= x\cdot\dot{x} + y\cdot\dot{y} + 0\cdot\dot{\theta}$

$= x(v\cos\theta) + y(v\sin\theta)$

Side-Note

Given:

$\dot{x} = v\cos\theta$

$\dot{y} = v\sin\theta$

$\dot{\theta} = \omega$

Also,

$\frac{\partial V}{\partial x} = x, \frac{\partial V}{\partial y} = y, \frac{\partial V}{\partial \theta} = 0$

$$= \left(x - \frac{k_1 \sin\theta\cos\theta}{\theta + k_2 d}\right) v\cos\theta + \left(y + \frac{k_1 \sin\theta\cos\theta}{\theta + k_2 d}\right) v\sin\theta$$

∴ we can analyse the expression and show that V is negative definite. If $\dot{V}<0$, then the control input $\omega = k_2 a + \frac{k_1 \sin(a)\cos(\theta)}{a + k_2 d}$

As for $-1<v<1$, $\frac{k+\sin\theta\cos\theta}{\theta+k_3 d}$ is always positive

of $\sin\theta$ & $\cos\theta$, ~~we can~~ v is also always less than or

equal to zero

$\therefore \dot{V}$ is negative definite, indicating that the

chosen control input $\omega = \frac{k_2 a + k_1 \sin(\alpha)\cos(\epsilon)}{a+k_3 d}$

will drive the robot towards the origin.

✓Code in seperate file

Answer 03

Lyapunov function:

$$V = \frac{1}{2}x_1^2 + \frac{1}{4}x_2^4 \quad \text{[given]}$$

Also Given,

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

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

$$\text{or, } \frac{dV(x)}{dt} = \frac{\partial V}{\partial x_1}\underbrace{\frac{dx_1}{dt}}_{=\dot{x}_1} + \frac{\partial V}{\partial x_2}\frac{dx_2}{dt}$$

$$= \left(\frac{2}{2}x_1^{2-1} + 0\right)(\dot{x}_1) + \frac{4}{4}x_2^3(\dot{x}_2)$$

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

To make the system from unstable

$$\frac{dV}{dt} \geq -\alpha V \geq \frac{dV}{dt}$$

$$\text{or, } \frac{1}{2}x_1^2 + \frac{1}{4}x_2^4 \geq x_1^2(-x_1 + x_3) + x_2^3(-x_2 + u)$$

[$\because \alpha = 1$ to stable the system]

$$\text{or, } \frac{1}{2}x_1^2 + \frac{1}{4}x_2^4 + x_1^2x_1 - x_1^2x_3 + x_2^3x_2 \geq u$$

$$\text{or, } -1 \geq u.$$

$$\therefore u \leq -1$$

Ans

Answer 04: A* - grid based search Algorithm

```
open = []
closed = []
#①Start the node
open.append(Start)

while open:
    #② find the node with least fvalue of open
    current = min(open, key = lambda node: node.f)
    #③Remove current from the open list
    open.remove(current)
    #④ Add current to the closed list
    closed.append(current)
    #⑤ Check if the current node is the goalStat
    if current == goal:
        #⑥ Reconstruct the path for the start
        Node to goal node
        path = []
```

continued

```
while current:
    path.append(current)
    current=current.parent
path.reverse()
return path

#Generate all possible successors of current
for successor in successors:
    #Calculate the g value
    successor.g=current.g+distance(current,successor)
    #Calculate the h value
    successor.h=manhattan_distance(successor,goal)
    #Calculate the f-value
    successor.f=successor.g+successor.h
```

Continued

#check if a node with the same position as successor is already in the open list & has a lower f-value

if successor in open and successor.f<successor.f:

continue

#Add successor to the open list

open.append(successor)

Rest of the code is in attached jupyter Notebook.