Pseudospectral motion planning for autonomous obstacle avoidance

AS5580: Pseudo-Spectral Methods for Optimal Control

Kishore M (AE21B036)

Contents

List of Figures

List of Tables

Abstract

This project investigates the use of the pseudo-spectral method for motion planning in autonomous vehicles to achieve obstacle avoidance with an optimal path. The method was initially applied to an unmanned ground vehicle (UGV) with steering and velocity control. One of the key challenges was the mathematical representation of obstacles. Various methods, including P-norm and normalized P-norm, were tested, but hard constraints proved unsuitable for environments with densely clustered obstacles. To address this, soft constraints were introduced, and different functions were analyzed to identify the most effective approach for both sparse and clustered obstacles.

The study was then extended to a point mass model of an unmanned aerial vehicle (UAV), where the planned path's feasibility was verified using differential flatness equations. For a more realistic UAV model, control feasibility was ensured through two separate approaches: direct dynamic equations and differential flatness. This model included controls for thrust, lift coefficient, and bank angle. Finally, the interpolated controls from the optimal solution were used to perform a trajectory simulation using the Runge-Kutta-Fehlberg(RK45) method.

1 | Introduction

Motion planning is a fundamental task for autonomous vehicles. Over the years, various design approaches have led to the development of specific algorithms to solve unique challenges. For instance, the artificial potential function method has been widely used for the motion planning of unmanned ground vehicles (UGVs) and robotic manipulators. However, this method often struggles with ensuring the vehicle reaches its goal and managing complex environmental constraints.

To address these limitations, recent advancements have emphasized sampling-based planning techniques, such as probabilistic road-maps, rapidly exploring random trees, and expansive space trees. These methods use probabilistic approaches to connect the initial configuration to the goal, enabling a better chance of success and creating initial feasible paths. However, they generally neglect vehicle dynamics, requiring additional path-following control to satisfy motion constraints. Nonholonomic constraints thus play a significant role in ensuring path-following systems meet physical limitations.

Optimal control theory offers a natural framework for motion planning but has traditionally been challenging to apply due to issues like dimensionality and complexity, particularly in obstacle-cluttered environments. Recent breakthroughs in computational optimal control have overcome many traditional challenges. For example, the use of pseudo-spectral methods enabled real-world solutions like the zero-propellant maneuver on the International Space Station. This project applies pseudo-spectral methods to develop motion planning algorithms for autonomous vehicles. These methods account for nonlinear dynamics, obstacle-rich environments.

This study focuses on generating optimal trajectories for various autonomous vehicles while modeling obstacles of different sizes, shapes, and configurations as path constraints. The framework is tested across multiple platforms, including ground vehicles, and aerial vehicles, demonstrating its versatility and effectiveness. The unified framework of optimal control and pseudo-spectral methods enables motion planning for diverse problems, ensuring the optimality of computed trajectories.

2 | Vehicle Models

2.1 | Unmanned ground Vehicle(UGV)

This model has three degrees of freedom: two for motion in the plane and one for in-plane rotations. The schematic of the vehicle is shown below. This model is referred from

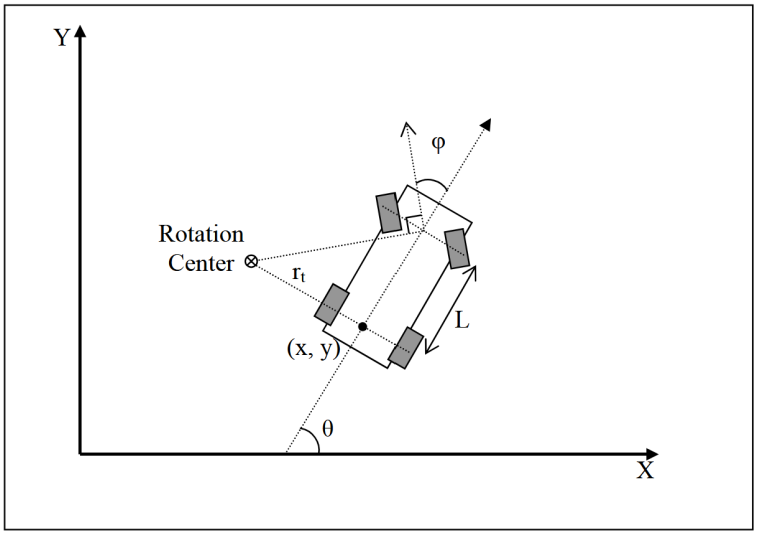


Figure 2.1: Schematic of Unmanned ground vehicle

State variables,

$$X = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

Control variables,

$$U = \begin{bmatrix} v \\ \phi \end{bmatrix}$$

where,

(x, y) = Position of the center of the rear axle.

θ = Heading angle of the vehicle with respect to x-axis.

v = Velocity of the vehicle.

ϕ = Steering angle with respect heading direction.

L = distance between the forward and rear axles.

Kinematic equations,

$$\dot{X} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \frac{v}{L}\tan\phi \end{bmatrix}$$

Control Constraints

$$v : 0 \leq v \leq v_{max}$$
$$\phi : -\phi_{max} \leq \phi \leq \phi_{max}$$

2.2 | Unmanned Aerial Vehicle(UAV) without dynamics

This model uses a point-mass approach with six degrees of freedom, without considering dynamics. There are three state variables and three control variables. The state variables represent three spatial degrees of freedom. The three control variables are the vehicle velocity, the flight path angle and the heading angle.

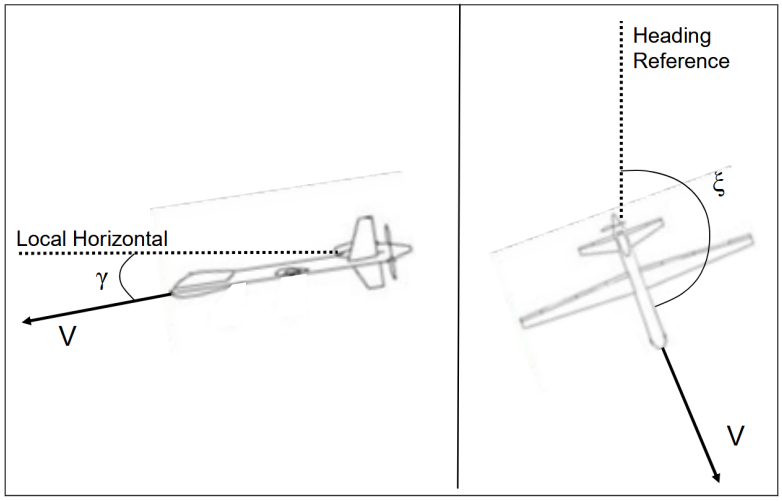


Figure 2.2: Schematic of UAV (with only Kinematics)

State Variables,

$$X = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Control Variables,

$$U = \begin{bmatrix} v \\ \gamma \\ \psi \end{bmatrix}$$

where,

(x, y, z) = Coordinates of UAV in 3D.

v = Velocity of UAV.

γ = Flight Path angle with respect to local horizon.

ψ = Heading angle with respect to Reference direction.

Kinematic equations,

$$\dot{X} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} v\cos\gamma\cos\psi \\ v\cos\gamma\sin\psi \\ v\sin\psi \end{bmatrix}$$

Control Constraints,

$$v : v_{min} \leq v \leq v_{max}$$

$$\gamma : -\gamma_{max} \leq \gamma \leq \gamma_{max}$$

$$psi : -\psi max \leq \xi \leq \psi_{max}$$

2.3 | Unmanned Aerial vehicle(UAV) with Dynamics

This model is adapted from [3] and includes UAV dynamics and controls such as thrust, bank angle, and coefficient of lift. The flight model contains translations only and no rotations, as the rotational dynamics are assumed to be significantly faster than the translational dynamics. The UAV parameters used is referred from [2] and is given below

Parameters	Value
m	4.5 kg
b	3 m
S_{ref}	0.473 m^2
AR	19.0
C_{D0}	0.0173
C_{D1}	-0.0337
C_{D2}	0.0517
$C_{L_{max}}$	1.1

Table 2.1: UAV parameters referred from [2]

State Variables,

$$X = \begin{bmatrix} x \\ y \\ z \\ v \\ \gamma \\ \psi \end{bmatrix}$$

Control Variables,

$$U = \begin{bmatrix} T \\ C_L \\ \mu \end{bmatrix}$$

Kinematic equations,

$$\dot{X}' = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} v\cos\gamma\cos\psi \\ v\cos\gamma\sin\psi \\ v\sin\psi \end{bmatrix}$$

2.3.1 | Dynamics of UAV

Forces acting on the UAV are thrust, Lift and Drag. These are functions of only thrust control and velocity. Lift and drag can be expressed as:

$$L = \frac{1}{2}\rho S v^2 C_L$$

$$D = \frac{1}{2}\rho S v^2 C_D$$

where, C_D is related to C_L by $C_D = C_{D0} + C_{D1}C_L + C_{D2}C_L^2$

The equations of motion of UAV are given by,

$$m\dot{v} = T - D - mg\sin\gamma$$

$$mv\dot{\psi}\cos\gamma = L\sin\mu$$

$$mv\dot{\gamma} = L\cos\mu - mg\cos\gamma$$

where,

(x, y, z) = Earth fixed coordinate system.

v = velocity of the UAV.

γ = Flight path angle with respect to local horizon.

ψ = Heading angle of flight with respect to reference, Here reference is y axis.

μ = Bank angle which is resulting control from rudders.

T = Thrust.

C_L = Coefficient of lift resulting control from ailerons.

2.3.2 | Differential Flatness equations

Imagine a nonlinear system of the form:

$$\dot{x} = f(x, u)$$

$$y = g(x, u)$$

This system is said to be differentially flat when there exist a set of outputs z such that the system outputs y, inputs u and states x can be expressed as function of the flat output z and its derivatives with respect to time.

The flight model above is differentially flat with $z = (x, y, z)$ being the flat output vector. The remaining state and control variables can be obtained as functions of the flat output z and its derivatives.

$$v = \sqrt{\dot{x}^2 + \dot{y}^2 + \dot{z}^2}$$

$$\psi = \arctan(\frac{\dot{x}}{\dot{y}})$$

$$\gamma = \arcsin{(\frac{\dot{z}}{v})}$$

$$\mu = \frac{\dot{\psi}\cos\gamma}{\dot{\gamma} + g/v\cos\gamma}$$

$$C_L = \frac{mv\dot{\gamma} + mg\cos\gamma}{1/2\rho S_{ref} v^2 \cos\mu}$$

$$T = m\dot{v} + D + mg\sin\gamma$$

3 | Pseudo-Spectral Formulation

3.1 | Optimal Control Problem

State and Control Variables:

- X: State variable U: Control variable

  Objective: Minimize Bolza cost function $g(X, U)$

$$g(X, U) = \phi(X(T)) + \int_0^T L(X(t), U(t), t)\, dt$$

  Subject to Constraints:

- **Dynamics:** $\dot{X} = f(X, U, t)$
- **Obstacles:** $h(X, t) \geq 0$
- **Bounds:** $X_{\min} \leq X \leq X_{\max}, \quad U_{\min} \leq U \leq U_{\max}$
- **Boundary Conditions:** $X(0) = X_0, \quad X(T) = X_f$

Shown above is the general optimal control problem formulation. The states, controls and dynamics changes for different vehicle models. The obstacles here is represented as separate hard constraint but can also be represented with the cost function as soft constraint. We see later which is better. Bounds are majorly used for limiting controls. Boundary conditions are the initial and target states.

Using the the above formulation the path planning can be done using uniform discretization and trapezoid integration method but it is computationally very expensive. This is observed in solving optimal control for 1D block problem in the figure 3.1.

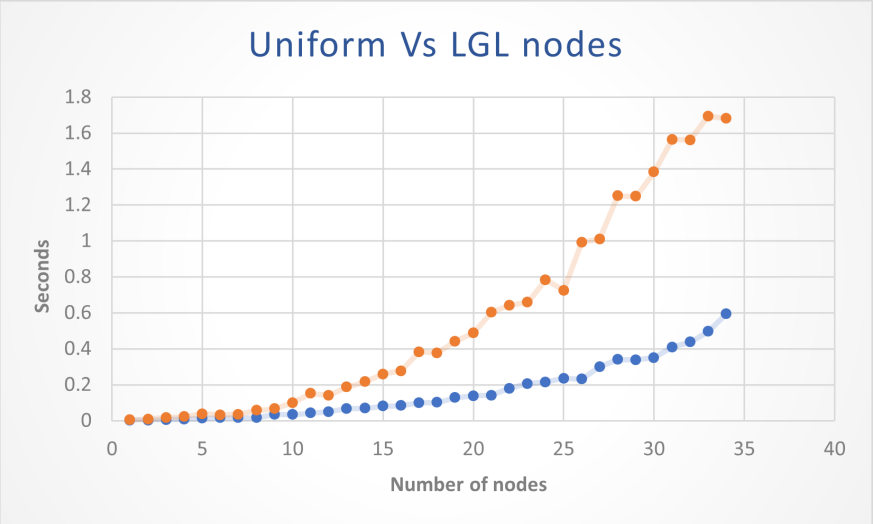


Figure 3.1: Pseudo-Spectral Vs Uniform discretization

So, Pseudo-Spectral methods known for its exponential convergence can be used. This can also be observed from the figure 3.1.

3.2 | Pseudo-Spectral Method

We take the Legendre PS method as an illustrative method while noting that the results are applicable to other PS methods as well. In the Legendre PS method, we approximate $X(t)$ by Nth order Lagrange polynomials $X_N(t)$ based on interpolation at the shifted Legendre–Gauss–Lobatto (LGL) quadrature nodes, that is

$$X(t) \approx X_N(t) = \sum_{k=0}^{N} X_N(t_k)\phi_k(t)$$

where $\phi_k(t)$ are the Lagrange interpolating polynomials, $t_k = t_f/2(\tau_k + 1)$, τ_k are the LGL collocation points.

3.2.1 | Differential Matrix

We need approximation for derivative of the Nth order polynomial for enforcing kinematic and dynamic equations. this can be achived using the differential Matrix.

$$
\begin{aligned}
X_N(t) &= \sum_{k=0}^{N} X_N(t_k)\phi_k(t) \\
\Rightarrow \dot{X}_N(t) &= \sum_{k=0}^{N} X_N(t_k)\dot{\phi}_k(t) \\
\text{derivative at jth node} \Rightarrow \dot{X}_N(t_j) &= \sum_{k=0}^{N} \dot{\phi}_k(t_j) X_N(t_k) \\
\Rightarrow \dot{X}_N(t_j) &= \sum_{k=0}^{N} D_{jk} X_N(t_k)
\end{aligned}
$$

So, we can obtain a differentiation matrix such that when this matrix is multiplied with vector of Lagrange interpolant values at node points gives the vector of its derivatives at node points. The D matrix is calculated using the node points as follows.

Finding the D matrix,

$$
\begin{aligned}
\Rightarrow D_{ij} &= \dot{\phi}_j(t_i) \\
\phi_j(t) &= \frac{\prod_{k\neq j}(t - tk)}{\prod_{k\neq j}(t_k - t_j)} \\
(w_j = \frac{1}{\prod_{k\neq j}(t_k - t_j)}) \Rightarrow \phi_j(t) &= w_j \prod_{k\neq j}(t - tk) \\
\Rightarrow \dot{\phi}_j(t) &= w_j \sum_{m=0}^{N} \prod_{k\neq j,m} (t - tk) \\
\Rightarrow \dot{\phi}_j(t_i) = D_{ij} &= w_j \sum_{m=0}^{N} \prod_{k\neq j,m} (t_i - tk)
\end{aligned}
$$

Pseudo Code for D matrix given the nodes

Algorithm 1 Computation of Differentiation Matrix D

Require: nodes $= \{x_0, x_1, \ldots, x_{n-1}\}$ (LGL or other nodes)
Ensure: D (Differentiation matrix)

Initialize $n \leftarrow$ length(nodes)
Initialize $D \leftarrow$ zero matrix of size $(n \times n)$
function WEIGHTS(nodes)
    for each $j \in \{0, 1, \ldots, n-1\}$ **do**
        Compute $w_j \leftarrow \frac{1}{\prod_{\substack{k=0 \\ k \neq j}}^{n-1} (x_j - x_k)}$
    end for
    return w
end function
$w \leftarrow$ WEIGHTS(nodes)
for each $i \in \{0, 1, \ldots, n-1\}$ **do**
    for each $j \in \{0, 1, \ldots, n-1\}$ **do**
        if $i \neq j$ **then**
            $D[i, j] \leftarrow w[j] \cdot \frac{1}{x_i - x_j}$
        else
            $D[i, i] \leftarrow -\sum_{\substack{k=0 \\ k \neq i}}^{n-1} \frac{w[k]}{x_i - x_k}$
        end if
    end for
end for
return D

Python Code for D matrix

```python
import numpy as np


def D_matrix(nodes):
    n = len(nodes)
    wj = lambda j: 1/np.product(nodes[j]-np.delete(nodes,j,0))
    D = np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            idxs = [m for m in range(len(nodes))]
            idxs.remove(j)
            D[i,j] = wj(j)* np.sum([np.product(nodes[i]-np.delete(nodes,[j,m])
   ) for m in idxs])

    return D
```

D_matrix.py

3.2.2 | Gauss Quadrature

Gauss quadrature is chosen to approximate the integral terms in cost function if any. So, Integration of a function is calculated as follows,

$$\int_{t_o}^{t_f} F(X(t), U(t))dt = \frac{t_f}{2}\int_{-1}^{1} F(X(\tau), U(\tau))d\tau$$
$$\approx \frac{t_f}{2}\sum_{k=0}^{N} w_k F(X(\tau_k), U(\tau_k))$$

where, w_k are the gauss lobatto weights of quadrature. It is calculated by the formula,

$$w_k = \frac{2}{N(N+1)} \times \frac{1}{[\phi_N(\tau_k)]^2}$$

4 | Obstacle Representation

4.1 | Problem beyond Computational Complexity

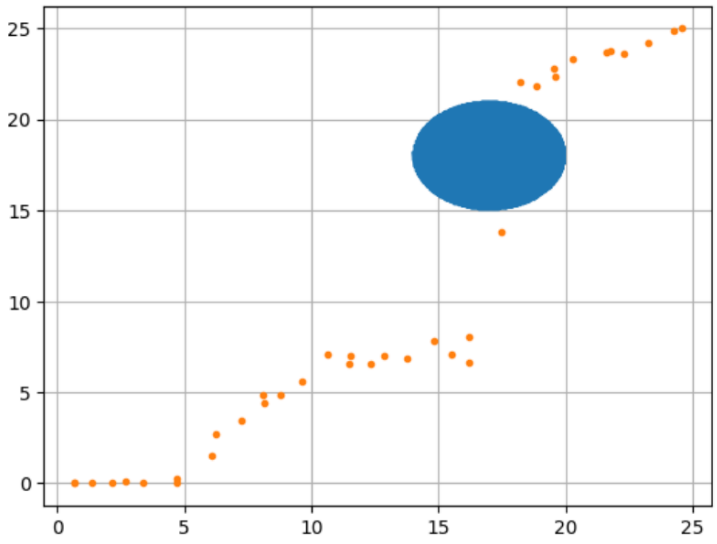


Figure 4.1: Infeasible and Suboptimal path problem

The above discussed Pseudo-Spectral method solves the problem of high computational complexity but there is some other problem which lead to infeasible problem and often resulting in suboptimal and infeasible paths like the one shown in figure 4.1 . This is a path output for UGV with obstacle represented using p-norm.

Many factors can be causing this problem. For example: global optimization may be the issue, we can use HP-FEM method; Maximum velocity is large so it jumps over the obstacle, velocity can be decreased; sparse node distribution in xy plane, can increase the number of collocation points. All these is tried out and nothing worked. The major problem here is the p-norm representation of obstacle as hard constraints. So, In this section, different Possible solution to this is proposed.

4.2 | Initial Obstacle representation

Initially the obstacles was represented by using p-norms. P-norm can be used to create generic shapes such as diamonds, circles, ellipses,squares, and rectangles. Figure 4.2 illustrates how the p-norm can be manipulated to create a diamond, a circle, and a square obstacle. Ellipses and rectangles are simply extensions of the circle and square, respectively.

$$h(x,y) = \|\frac{x-x_c}{a}, \frac{y-y_c}{b}\|_p - c$$

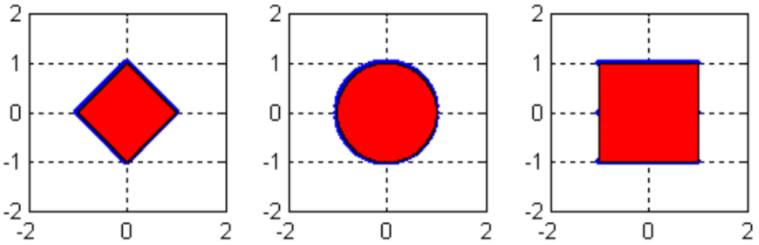


Figure 4.2: Obstacles using p-norm

This method is referred from [1] but as we can see from figure 4.1 this method leads to undesired solutions.

4.2.1 | Normalized P-Norm

While P-norm gives infeasible and suboptimal solutions, The normalized version of this using the logarithm of P-norm works magically! but only for sparse obstacle environment.

$$h(x,y) = \log(\|\frac{x-x_c}{a}, \frac{y-y_c}{b}\|_p - c)$$

But why does this work? Here is my explanation. difference between the p-norm and its normalized lies in the hessian of the two. Any NLP solver use this Hessian to evaluate the step size at each iteration. If the hessian of a constraint is large then the step size would be small and if the hessian is small then the step size would be large. Given below is the plot of the hard constraints by p-norm and normalized counterpart visuallized in 3D.

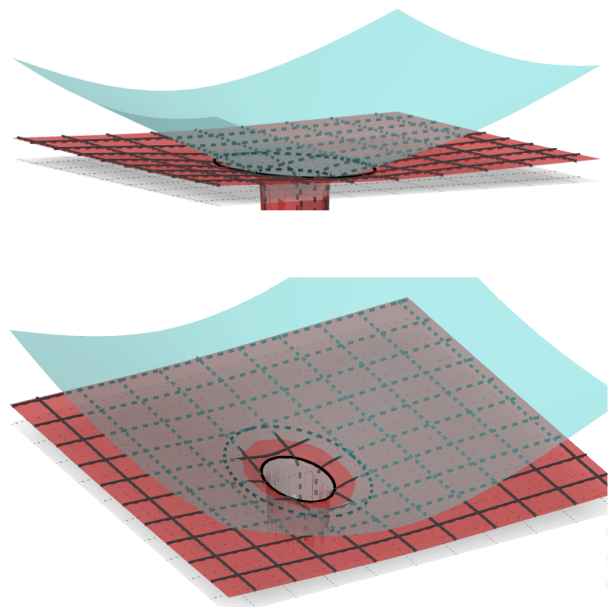


Figure 4.3: Normalized p-norm(in red) Vs P-norm(in green)

As we can see from the figure 4.3, the p-norm graph is more steep than that of Normalized P-norm. This might be the reason for working of normalized P-norm.

But this works only with two obstacle environment or sparse obstacle environment. With clustered obstacles, Again its the same problem of infeasible problem or suboptimal solutions.

What is the Major problem with this? The major problem is representing obstacles as hard constraints. Doing So makes the feasible region non convex which is a major issue in NLP problems. Figure illustrates non-convex and convex feasible regions. So, lets take out the hard constraints and represent obstacle as soft constraints with the cost function. This preserves the convexity of the feasible region.

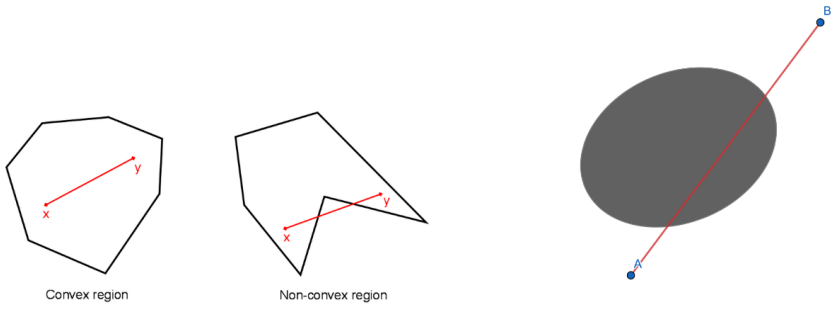


Figure 4.4: Convex Vs Non-Convex feasible region

4.3 | obstacles as soft constraints

Soft constraints in NLP are functions added to the objective function. These functions are chosen such that the constraints are satisfied when these functions are minimized. These should be chosen carefully so that optimality isn't compromised. Different such functions are tried out that are as follows.

4.3.1 | Inverse of square of the p-norm

$$h(x,y) = \frac{1}{(\|\frac{x-x_c}{a}, \frac{y-y_c}{b}\|_p - c)^2}$$

This function is minimized outside the obstacles but there is also an local minimum inside the obstacle. So, the path might go through the obstacle. This can be observed from the figure

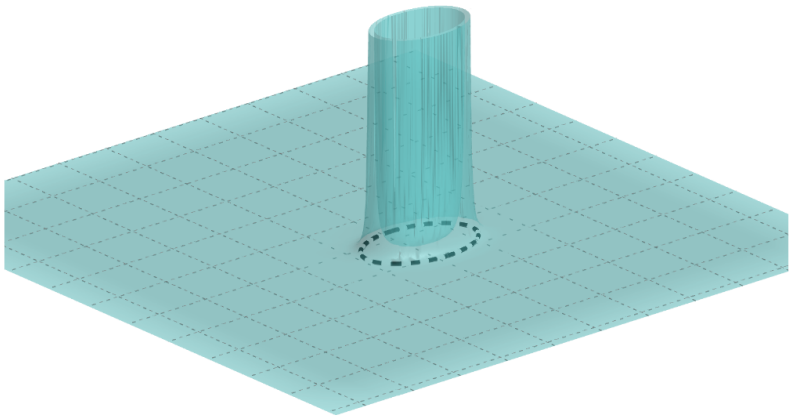


Figure 4.5: Inverse of square of the p-norm

4.3.2 | Gaussian Distribution

$$h(x,y) = \sum_{i=1}^{m}(\frac{K}{\sqrt{2\pi\sigma}} \cdot \exp(\frac{-\left(\|\frac{x-x_c}{a}, \frac{y-y_c}{b}\|_2 - c\right)}{2\sigma^2}$$

The basic idea behind this function is to fit the 2D Gaussian distribution over the obstacle. This method introduces two more decision variable that has to be given before solving. These variables are K and σ these variables introduces trade off between optimality and safety. If K is small then soft constraint will be compromised in minimizing the objective function. If K is large then optimality will be compromised in minimizing the soft constraint. Visual of this function is shown in figure 4.6.

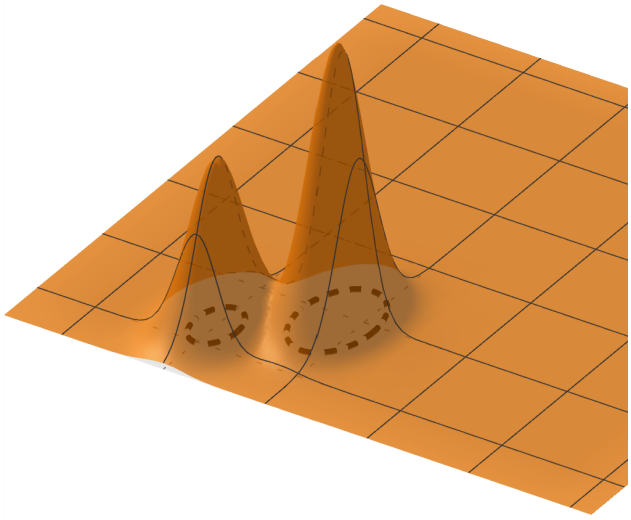


Figure 4.6: Gaussian distribution soft constraint

4.3.3 | Custom Soft constraint

$$H(x,y) = \left(\left(e^{e^{-h_1(x,y)}} - 1\right)^p + \left(e^{e^{-h_2(x,y)}} - 1\right)^p + \left(e^{e^{-h_3(x,y)}} - 1\right)^p + \dots\right)^{\frac{1}{p}}$$

Here $h(x,y)$ represent the p-norm constraint previously seen. This soft function constraint has all the desired properties for a soft constraint. This was referred from [1]. As we can see from the figure , it goes to infinity as we go near and inside the obstacle and also doesn't have any extra decision parameters.

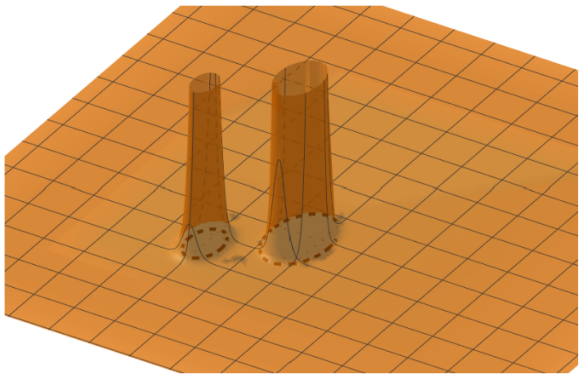


Figure 4.7: Custom Soft constraint

5 | Code Routine

This section describes the key steps involved in implementing the optimization problem for motion planning using the pseudospectral method. The problem is formulated and solved using CasADi with IPOPT as the solver. The following steps summarize the implementation:

1. **Define the Optimization Problem:**
   - Initialize the optimization problem using `opti = ca.Opti()`.

- Set the number of collocation nodes N and calculate the Legendre-Gauss-Lobatto (LGL) nodes and weights using the `lglnodes()` function.
- Compute the pseudospectral differentiation matrix D using the `D_matrix()` function.

2. **Define Variables:**
   - Declare state variables and Control variables.
   - Declare the final time t_f as an optimization variable.
3. **Set Bounds:**
   - Apply bounds on control variables using the *opti.subject_to*(*opti.bounded*()) method.
   - Ensure that all control variables remain within their feasible ranges.
4. **Define Boundary values:**
   - Constrain the initial and target values of the state variables using *opti.subject_to*(*opti.bounded*()) method.
5. **Add Dynamics Constraints:**
   - Use the pseudospectral differentiation matrix D to enforce the dynamics constraints using *opti.subject_to*() method.
6. **Define the Objective Function:**
   - Define the objective function with the optimality term (like final time) and the soft constraint function if required.
   - For this the `opti.minimize(objective)` method.
7. **Solver Configuration:**
   - Set solver options such as tolerance (`tol`), maximum iterations (`max_iter`), and constraint violation tolerance (`constr_viol_tol`).
   - Choose IPOPT as the solver using `opti.solver('ipopt', p_opts, s_opts)`.
8. **Provide Initial Guess:**
   - Set initial guesses to aid convergence.
9. **Solve the Problem:**
   - Use `opti.solve()` to find the optimal solution.
   - Extract and store the optimized variables such as x, y, θ, v, ϕ, w, and t_f.

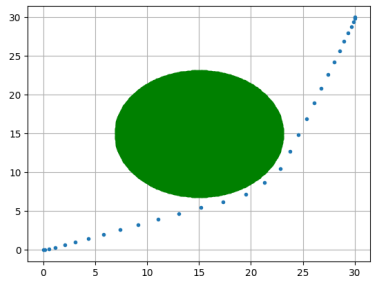
The code routine is modular and can be adapted to other vehicle dynamics by modifying the dynamics constraints and bounds accordingly.

6 | Results and Observation

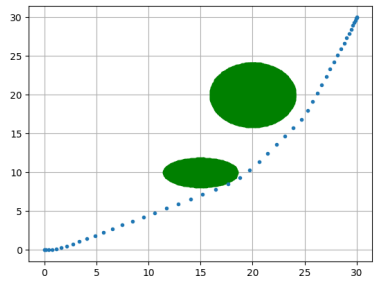
6.1 | For UGV

6.1.1 | Using Normalized P-norm obstacle representation

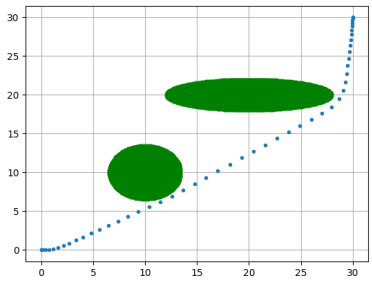
The following obstacle maps are generated with random parameters using 2-norm. Initial position for all is taken as (0,0) without loss of generality.



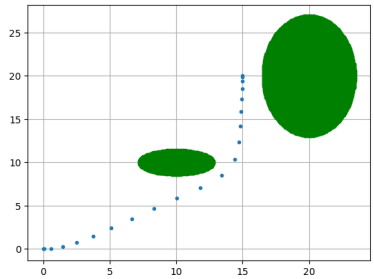
((a)) target=(30,30); Time optimal solution



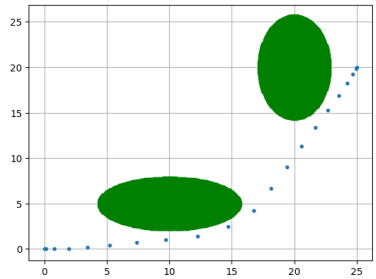
((b)) target = (30,30); Time optimal solution



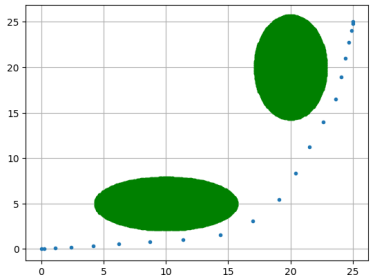
((c)) target = (30,30); Time optimal solution



((d)) target = (15,20); Time optimal solution



((e)) target = (25,20); Time optimal solution

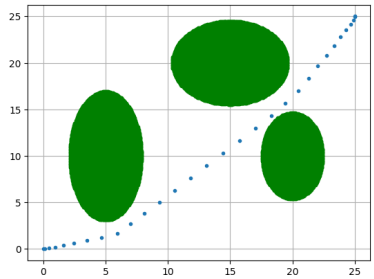


((f)) target = (25,25); Minimal control solution

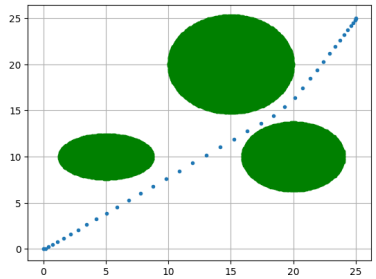
Figure 6.1: Results using normalized P-norm obstacles

6.1.2 | Using Soft constraint

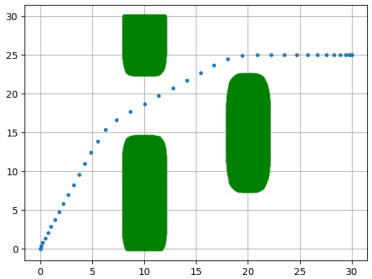
The following results generated by representing the obstacles as soft constraints.



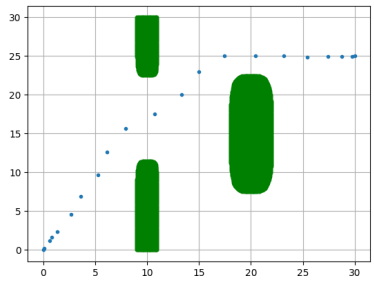
((a)) target=(25,25);Custom soft constraint Time optimal solution



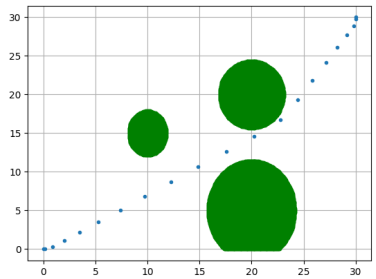
((b)) target=(25,25);Custom soft constraint; Time optimal solution



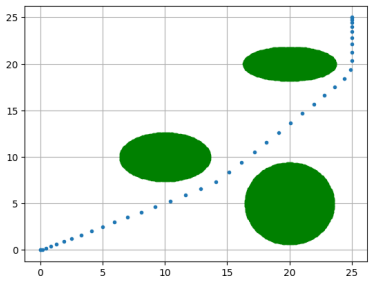
((c)) target=(30,25);Custom soft constraint; Time optimal solution



((d)) target=(30,25);Inverse of p-norm square soft constraint; Time optimal solution



((e)) target=(30,30);Gaussian soft constraint; Time optimal solution



((f)) target=(30,30);Custom Soft constraint; Time optimal solution

Figure 6.2: Results using normalized P-norm obstacles

6.1.3 | In Clustered environment

Finally to validate the code, I have made a custom clustered obstacle environment with narrow gaps in between them. The result for it is shown in figure 6.3

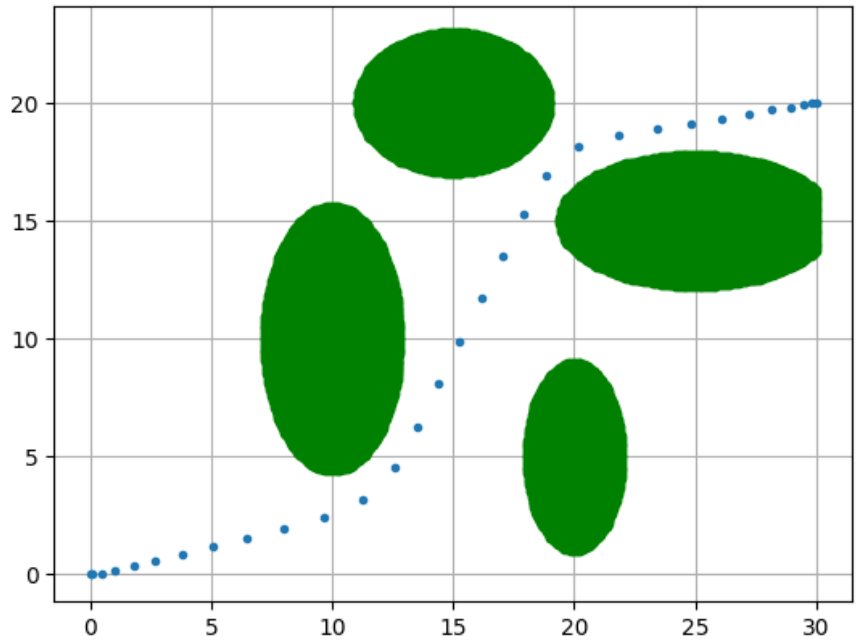
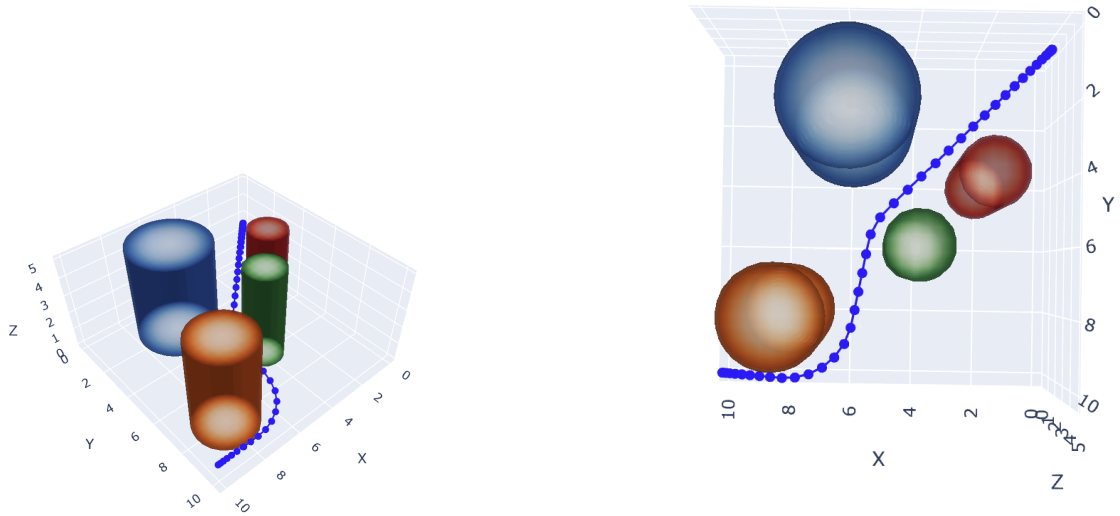


Figure 6.3: Clustered environment with narrow gaps; Time optimal solution using Custom soft constraint function

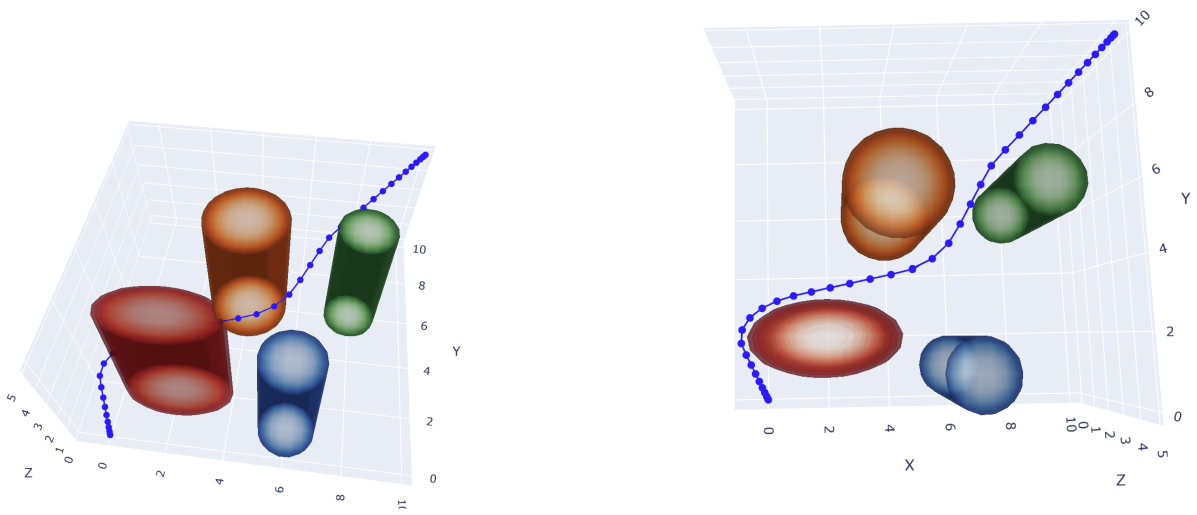
6.2 | For UAV without dynamics

For UAV only the custom soft constraint is used. The obstacles are in 2D but it is constrained at every height. The control constraints are $V_{max} = 5$ and $\gamma_{max} = \pi/4$ and $\psi_{max} = \pi$ these are referred from [1].



((a)) Bird View **((b))** Top View

Figure 6.4: Initial = (0,0,2.5); Target = (10,10,1)



((a)) Bird View ((b)) Top View

Figure 6.5: Initial = (0,0,0); Target = (10,10,5)

6.2.1 | Controls required for path in figure 6.5

Controls for the path shown in figure 6.5 is estimated using the UAV parameters in table 2.1.

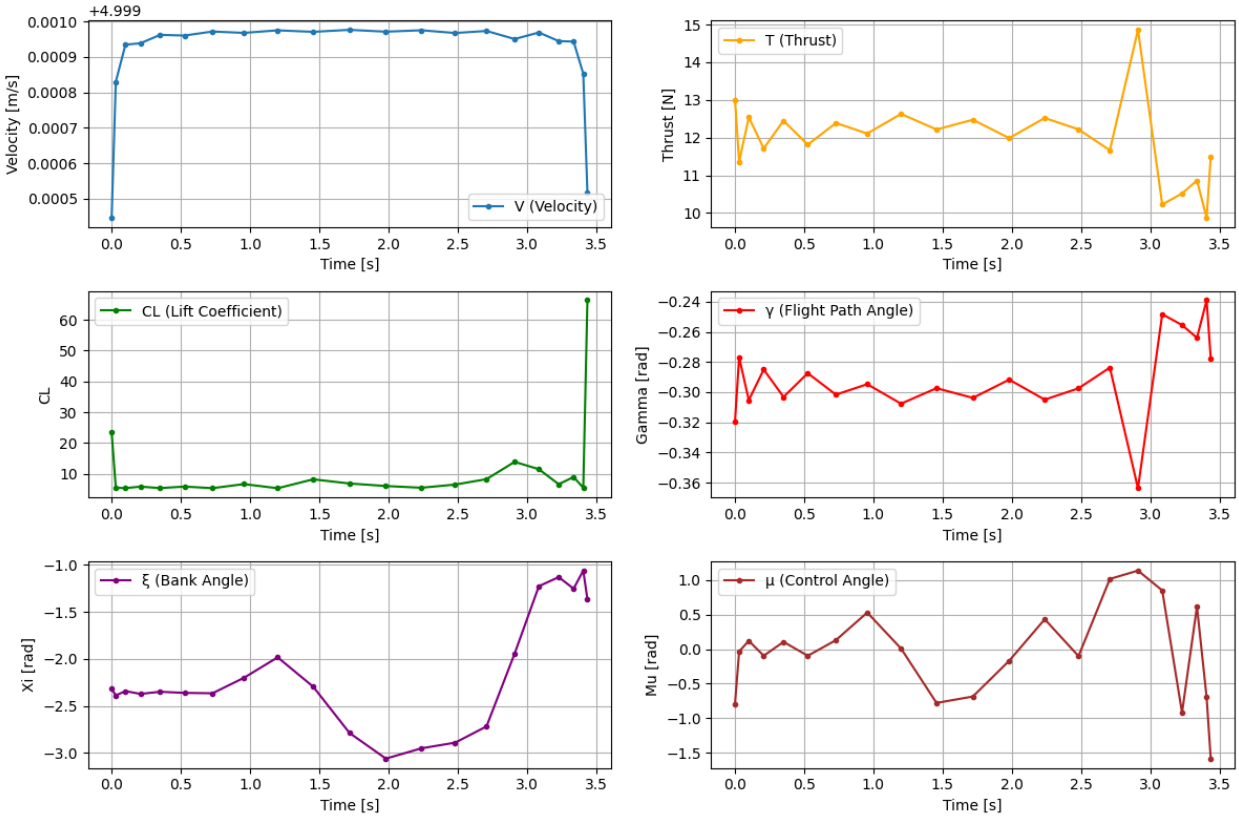
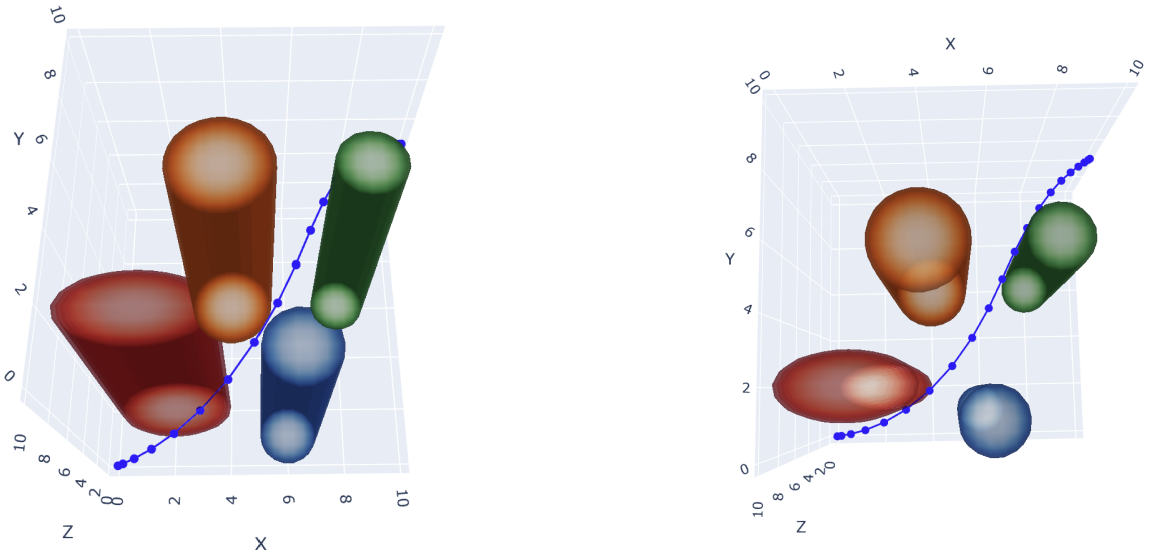


Figure 6.6: Controls required for path in figure 6.5

6.3 | For UAV with dynamics

The parameters used for this is as shown in table 2.1. Apart from that some other parameters used are,

$$T_{max} = 60N; \mu_{max} = \pi/3; \gamma_{max} = \pi/4; \xi_{max} = \pi$$



((a)) Bird View ((b)) Top View

Figure 6.7: Initial = (0,0,0); Target = (10,10,5); UAV with dynamics

6.3.1 | Controls for the Path in figure 6.7

The controls is found out using the differential flatness equations.

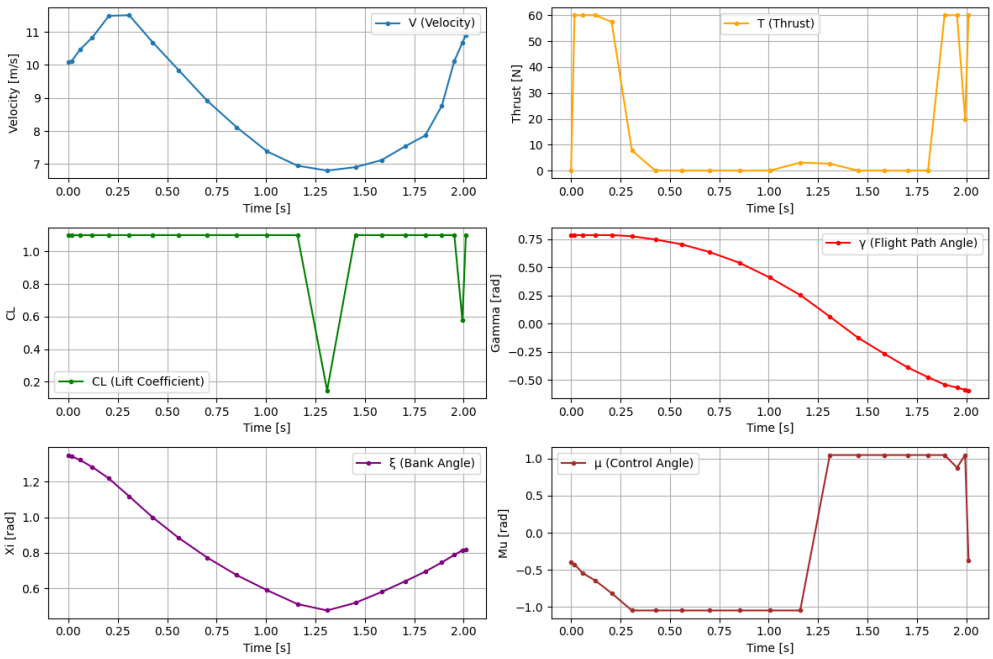
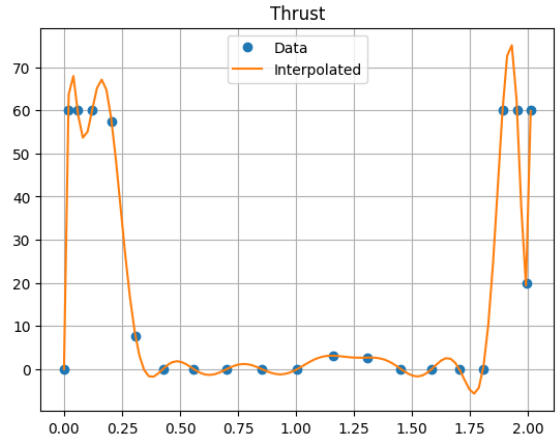


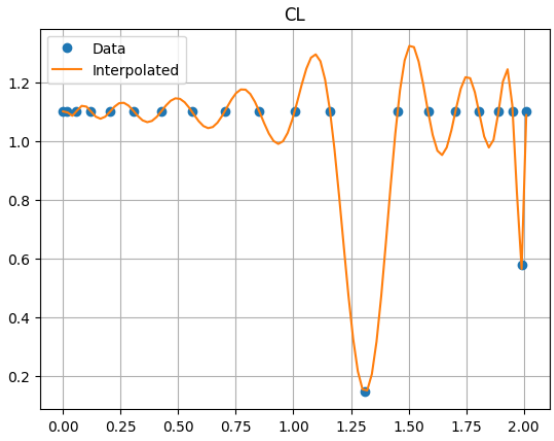
Figure 6.8: States and controls for Path in figure 6.7

6.3.2 | Trajectory simulation

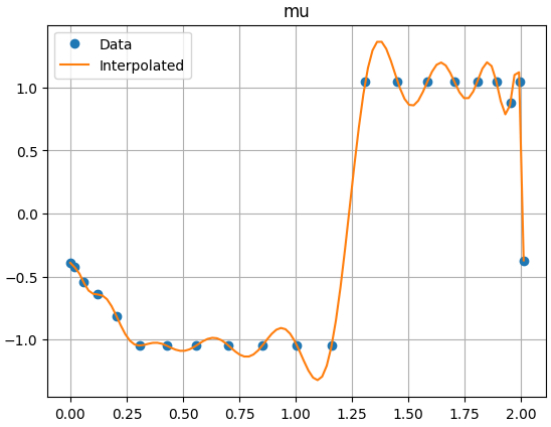
The above solution is validated by Trajectory simulation using RK45. The controls are thrust (T), Coefficient of Lift and Bank angle. These controls are obtained as function of time using global Lagrange interpolation of the controls shown in figure 6.8. Interpolated controls are shown in figure



((a)) Thrust (in N)



((b)) Coefficient of lift



((c)) Bank angle (in radians)

Comparing Trajectories

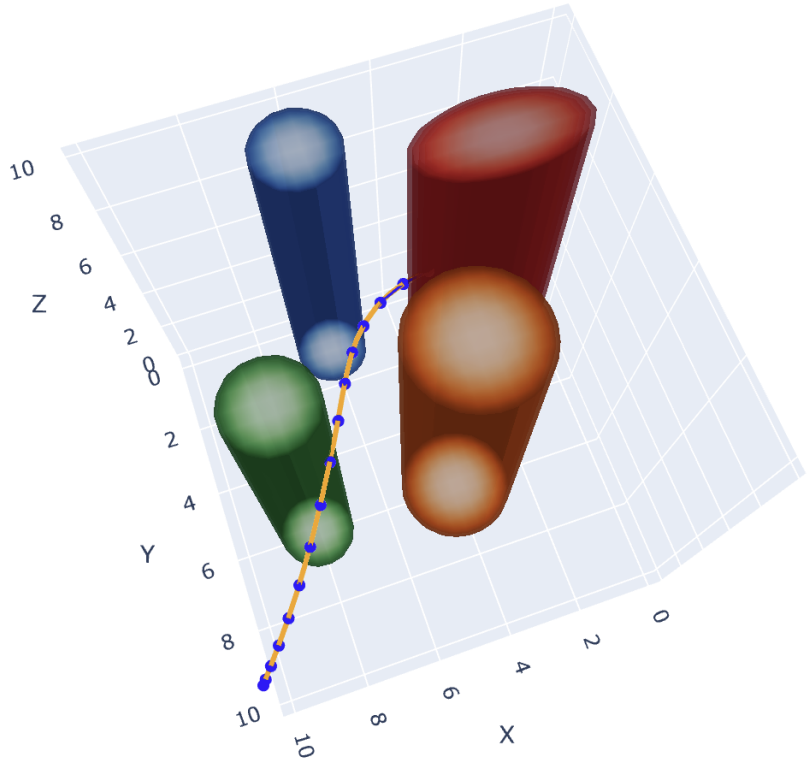


Figure 6.10: Trajectory comparison Bird view

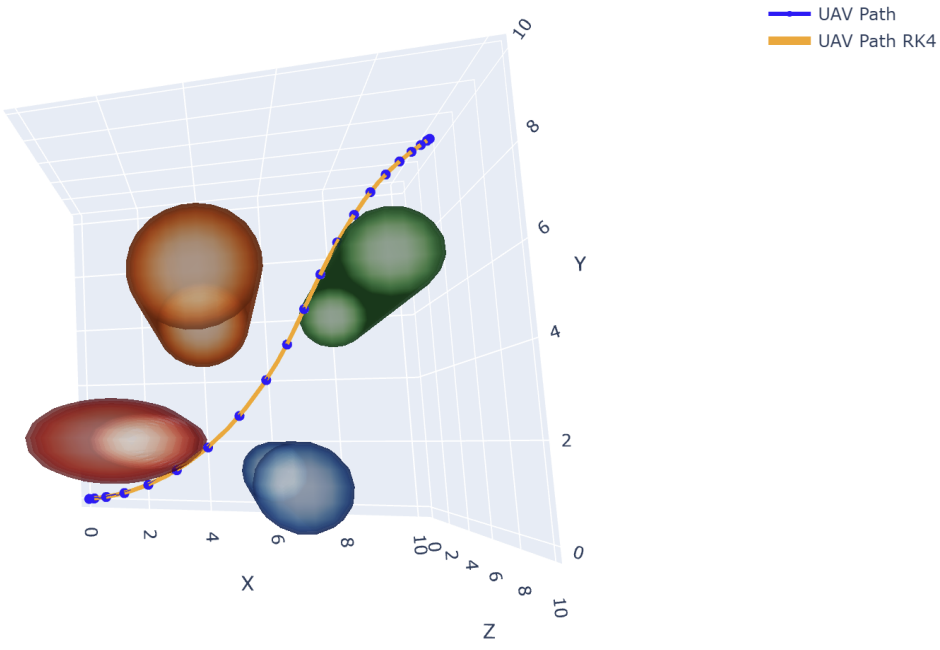


Figure 6.11: Trajectory comparison Top view

7 | Conclusion

A modular method for motion planning with autonomous obstacle avoidance has been developed using the pseudo-spectral method. The approach has been validated in diverse environments, including randomly generated obstacle fields and clustered environments. Furthermore, the method has been successfully tested on both unmanned ground vehicles (UGVs) and unmanned aerial vehicles (UAVs). Real-world applicability was demonstrated through trajectory simulations of UAVs, incorporating realistic dynamics to ensure the robustness of the approach.

8 | Further Works

8.1 | Verification of Optimality

The optimality of the computed solutions can be rigorously verified using analytical conditions, such as the Pontryagin's Maximum Principle.

8.2 | Adaptive Collocation Point Refinement

The barycentric form of Lagrange interpolation could be utilized to develop algorithms for adaptive refinement of collocation points. This enhancement could improve computational efficiency and accuracy, especially in scenarios with complex or highly dynamic obstacle fields.

References

[1] Laird-Philip Ryan Lewis. "Rapid motion planning and autonomous obstacle avoidance for unmanned vehicles". MA thesis. Monterey, CA; Naval Postgraduate School, 2006-12.

[2] C. A. Toomer Markus Deittert Arthur Richards and Anthony Pipe. "Engineless Unmanned Aerial Vehicle Propulsion by Dynamic Soaring". In: *JOURNAL OF GUIDANCE , CONTROL , AND DYNAMICS* (2012).

[3] Ying Celia Qi Yiyuan J. Zhao. "Minimum fuel powered dynamic soaring of unmanned aerial vehicles utilizing wind gradients". In: *Optimal Control Applications and Methods* (2004).