Collision Free Trajectory Optimization for Control of Fixed-Wing UAVs using Graph of Convex Sets

6.8210 Underactuated Robotics - Final Project May 2024

Steve Carter Feujo Nomeny
Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

***Abstract*—In this project, a collision-free trajectory is generated for a fixed-wing unmanned aerial vehicle (UAV) navigating an obstacle-dense environment. The chosen UAV has flat outputs, allowing the trajectory to be represented by Bézier curves, which ensure that the paths remain collision-free within the designated operational regions. These regions are convex partitions and a subspace of the available search space for potential paths. Once the trajectory is calculated in the flat output space, it is executed by reconstructing the original fixed wing's non-flat outputs and playing it back.**

I. Introduction

Fixed-wing Unmanned Aerial Vehicles (UAVs) rely on trajectory optimization to determine their flight paths. This powerful technique enables us to address critical tasks such as collision avoidance while simultaneously identifying the optimal route. However, these tasks often give rise to complex optimization problems. To tackle these challenges, I intend to use Graph of Convex Set (GCS) Trajectory Optimization. By parameterizing the trajectory using Bezier curves and defining convex regions that connect the start and end points, we transform the problem. Specifically, we shift from dealing with a hard nonconvex optimization (where the goal is to stay outside regions defined by obstacles) to a convex optimization (where the objective is to remain within a set of given convex regions that do not contain the obstacles).

The paper is divided into sections and structured in the following: section II will introduce the concept of Graph of Convex Sets for trajectory optimization and explain what problem is being solved. Moving on to section III, the partitioning of the free configuration space using Iris Clique cover will be presented. Following is section IV which explains the states of the delta-wing aircraft and it's differential flat properties. Section V presents the results and highlights some weaknesses of the implementation. Lastly is section VI and VII concluding the topic and suggesting future work on the implementation respectively.

II. Graph of convex set trajectory optimization

The concept of the Graph of Convex Sets (GCS) for trajectory optimization introduces a novel approach to managing the complexities inherent in planning collision-free trajectories in cluttered environments [8]. Figure 1 displays an overview of the GCS framework. Traditional methods often struggle with the nonconvex nature of spaces populated with obstacles. The GCS method circumvents these challenges by decomposing the robot's configuration space into overlapping convex sets, each representing a region free from obstacles, Figure 1a). This decomposition allows the trajectory optimization problem to be reformulated as a Shortest Path Problem (SPP) on a graph where nodes corresponding to these convex sets and edges represent feasible transitions between them as seen in Figure 1b).

Each convex region in the GCS framework is associated with a Bézier curve, enhancing the method's ability to handle complex trajectory shapes and ensuring that the trajectories are smooth and feasible within the defined safe regions. This is done using the control points of the Bezier curves and constraining them to lie within their region. In Figure 1c) the orange dots are the control points with the blue lines being the corresponding Bezier curve. This parameterization not only facilitates the smooth transition of trajectories from one convex set to another, as seen in Figure 1d) but also allows for the precise control of the trajectory's shape, duration, and dynamics, leading to highly optimized and feasible paths.

As mentioned in [4], this graph-based approach facilitates the use of efficient Mixed-Integer Convex Programming (MICP) techniques, substantially tightening the convex relaxations typically necessary in trajectory optimizations. By mapping each convex set to a node in a graph and connecting these nodes with edges that represent feasible trajectory segments, the trajectory design problem is simplified to finding the most efficient path through these nodes. This approach significantly improves computational efficiency and offers a robust framework for quickly resolving high-dimensional trajectory planning problems, as demonstrated in this project.

This algorithm allows us to generate feasible trajectories with velocity constraints and path continuity constraints up to a desired order. In this project the Bezier curves had an order of 7 (8 knot points), and path continuity constraints were enforced up to order 4 (snap).

III. Convex region partition

A. IRIS

As aforementioned, GCS trajectory optimization needs a convex representation of the free space and is given that through a set of convex regions. How these regions are generated is through an algorithm called IRIS (Iterative Regional

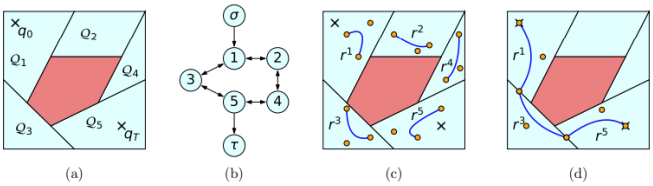


Fig. 1: The formulation of the GCS Trajectory optimization. The collision-free motion-planning problem is formulated as an SPP in GCS [4].

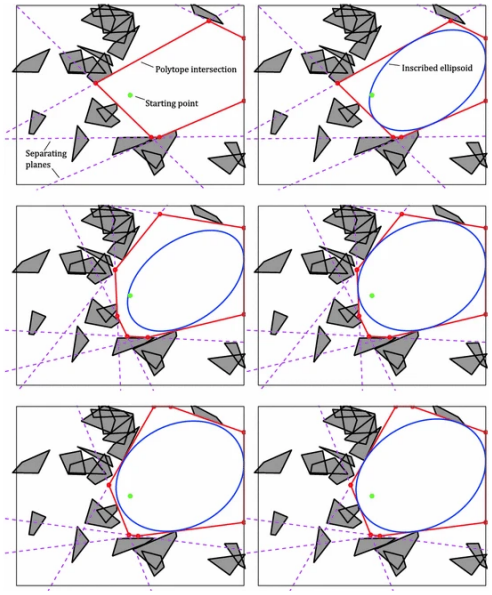


Fig. 2: Demonstration of one inflation of an IRIS region in an obstacle-filled environment [1].

Inflation by Semidefinite programming). It is a method for quickly computing large polytopic and ellipsoidal regions of obstacle-free space through a series of convex optimizations [1]. The algorithm inflates a convex region based on a given sample, well illustrated in Figure 2.

To cover most of the free configuration space the IRIS algorithm needs to run multiple times with different seed points. Depending on how well the seed points are chosen, the resulting Iris regions can be abundant and not as "well defined", which will slow down the GCS trajectory optimization solver. A clever way to choose seed points is by using a method called Iris Clique Cover [10].

B. IRIS Clique Cover

The Iris clique coverage method represents a significant advancement in robot configuration space approximation using convex sets. The algorithm is visualized in Figure 3. This technique utilizes a visibility graph where nodes represent collision-free sample points, and edges connect nodes that are directly visible to each other without any obstructions. In essence, the visibility graph contains subgraphs, or "cliques,"

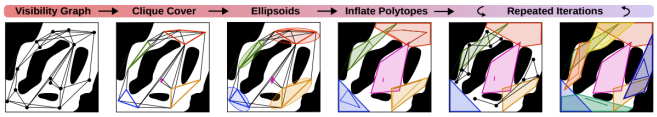


Fig. 3: Sketch of the Iris clique coverage as presented in [1].

where each clique is a fully connected subset of nodes [10]. These cliques are instrumental in approximating contiguous and convex portions of the configuration space.

The main strategy of the Iris clique coverage method involves first constructing a visibility graph by sampling points within the free configuration space of the robot, followed by identifying cliques within this graph, as shown in the first four figures of Figure 3. Each clique, by virtue of its fully connected nature, suggests a potential convex subset of the configuration space. The points within each clique are then used to direct the inflation of convex polytopes in the IRIS algorithm, which are large, full-dimensional, and encapsulate the sampled points [10]. Figure 4a shows parts of the cliques created using the Iris Clique Cover method.

The process of clique identification and polytope inflation is repeated iteratively, with each iteration refining the coverage of the configuration space by adding new samples from regions not adequately covered in previous steps, last 2 figures of Figure 3. This iterative refinement continues until the union of all inflated polytopes achieves satisfactory coverage of the configuration space. In this project, 2 iterations proved to be sufficient to achieve reasonable coverage of the configuration space ($\sim 90\%$) as seen in Figure 4b. The efficiency of this method, particularly in terms of the number of polytopes required and the computational time, marks a significant improvement over alternative method like randomly selecting seed points and passing this directly to IRIS. By leveraging the structural insights provided by the visibility graph and its cliques, the Iris clique coverage method achieves a more precise and computationally efficient approximation of the robot's configuration space [10].

IV. Model and Differential Flatness

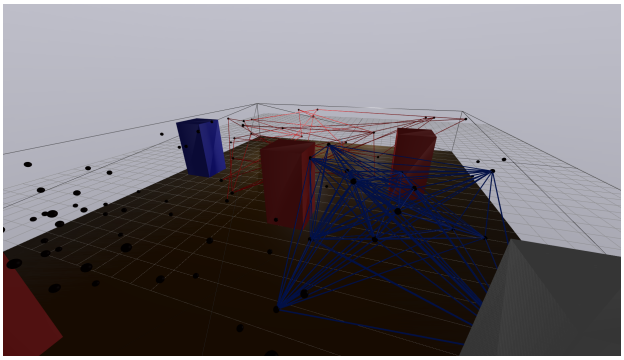
A. Model

The model of the fixed wing is retrieved in its entirety from [6], and the following are a few definitions and assumptions needed to fully understand it.

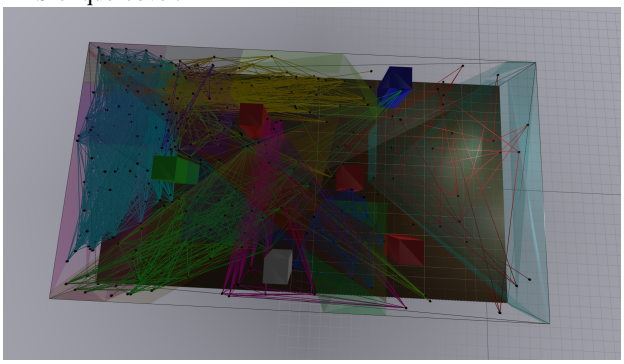
As in [6], no wind, no disturbance, constant gravity, and flat and fixed earth is assumed.

Furthermore, the world frame $\mathcal{W}$ is defined with axis z^w pointing upwords. The inertial frame $\mathcal{I}$ is defined as the world frame with the z axis flipped $180°$ around axis x^w pointing downwards wrt. the world frame. The aerodynamic frame $\mathcal{A}$ of the aircraft has its origin at the center of mass and its axis x^a points towards the aircraft's aerodynamic speed v_a.

As in [6] a delta-wing aircraft configuration is used, Figure 5, this is a convertible aircraft and can thus fly in hover mode. However, it will only be used in horizontal flight in



(a) Part of the cliques created from the visibility graph in the IRIS clique cover.



(b) The cliques and regions after Iris clique cover is done running with 500 sampled points. We observe that most of the configuration space is covered by the regions.

Fig. 4: Two stateg of Iris clique cover

this project. The mass of the aircraft is m = 1.56 kg, the body frame $\mathcal{B}$ has the origin at the center of mass, and the aircraft has a North-East-Down (NED) orientation. This means that the its axis x^b points towards the nose of the aircraft, y^b points out of the right wing and z^b points into the belly of the aircraft, as illustrated in Figure 5.

The delta-wing consists of two counter-rotating propellers with rotation speed denoted ω_l and ω_r for the left and right motor respectively. Additionally, δ_{el} and δ_{er} are the deflections of the left and right elevons.

The delta-wing has an aerodynamic angle of attack α, and an aerodynamic side-slip angle β.

From [6], the orientation of frame $\mathcal{A}$ wrt. frame $\mathcal{B}$ is given by

$$^B\mathbf{R}^A = \mathbf{R}_y(-\alpha)\mathbf{R}_z(\beta)$$

where the rotation $\mathbf{R}_x(\phi)$, $\mathbf{R}_z(\theta)$, $\mathbf{R}_y(\psi)$ are elementary rotation matrices in terms of three successive rotations ϕ, θ, ψ called euler angles around the axis x,y, and z.

$$\mathbf{R}_x(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix}$$

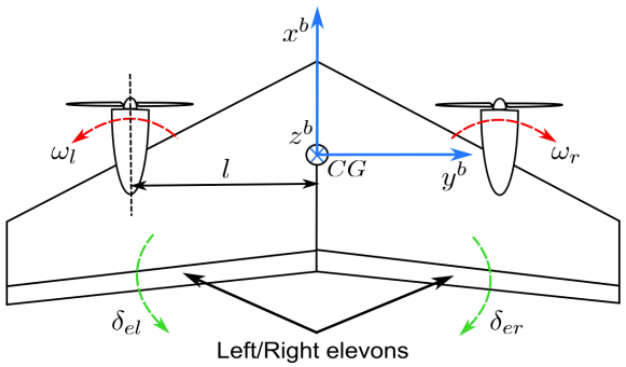


Fig. 5: Simplified model the delta-wing aircraft. It has two counter-rotating propellers and, two elevons [6].

$$\mathbf{R}_y(\psi) = \begin{pmatrix} \cos\psi & 0 & \sin\psi \\ 0 & 1 & 0 \\ -\sin\psi & 0 & \cos\psi \end{pmatrix}$$

$$\mathbf{R}_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Furthermore, the euler angles describing the orientation of the aerodynamic frame $\mathcal{A}$ wrt the inertial frame $\mathcal{I}$ are defined as χ, γ, and μ. Given this, the orientation of frame $\mathcal{A}$ wrt. frame $\mathcal{I}$ is given by

$$^I\mathbf{R}^A = \mathbf{R}_z(\chi)\mathbf{R}_y(\gamma)\mathbf{R}_x(\mu)$$

The angular velocity of the body frame $\mathcal{B}$ relative to inertial frame $\mathcal{I}$ and projected in the body frame $\mathcal{B}$, is:

$$^I\Omega^B_B = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

The state of the system together with its inputs are are thus fully defined by

$$\mathbf{x} = [x^o, y^o, z^o, v_a, \beta, \alpha, \chi, \gamma, \mu, p, q, r]$$
$$\mathbf{u} = [\delta_{el}, \delta_{er}, \omega_l, \omega_r]$$

where x^o, y^o, z^o denotes the position of the center of mass wrt. to the inertial frame $\mathcal{I}$.

B. Differential Flatness

Differential flatness theory states that if a system has a set of flat outputs, then it is possible to determine the complete state and input trajectory from these flat outputs and a finite number of their derivatives. Given a desired trajectory in terms of the flat outputs, one can compute the corresponding full state and input trajectory [8], [6]. As in [6] the flat outputs chosen are the position of the center of gravity of the delta-wing and its angle of attack.

$$\boldsymbol{\sigma} = [x^o, y^o, z^o, \alpha]$$

In this project, the angle of attack has been fixed to $\alpha = 0$. Similar to [6], the side slip angle has been assumed to be $\beta = 0$.

The full proof of the selected flat outputs indeed being differentially flat can be found detailed in [6]. However, the states of interest, being the position of center of mass and the euler angles χ, γ, μ, are necessary for visualization. Thus, for the sake of completeness the explicit expression of these states are given as:

$$x(t) = \boldsymbol{\sigma}[0]$$
$$y(t) = \boldsymbol{\sigma}[1]$$
$$z(t) = \boldsymbol{\sigma}[2]$$

The position of the center of mass is already a flat output.

$$v_a(t) = \sqrt{x^2 + y^2 + z^2}$$
$$\gamma(t) = \arcsin\left(\frac{\dot{z}}{v_a}\right) \tag{1}$$
$$\chi(t) = \arctan\left(\frac{\dot{y}}{\dot{x}}\right) \tag{2}$$

The derivatives of these states are necessary to compute μ, and are defined as:

$$\dot{v}_a(t) = \frac{\dot{x}\ddot{x} + \dot{y}\ddot{y} + \dot{z}\ddot{y}}{\sqrt{\dot{x}^2 + \dot{y}^2 + \dot{z}^2}} \tag{3}$$
$$\dot{\gamma}(t) = -\frac{1}{v_a\sqrt{1 - \left(\frac{\dot{z}}{v_a}\right)^2}}\left(\ddot{z} - \frac{\dot{z}\dot{v}_a}{v_a}\right) \tag{4}$$
$$\dot{\chi}(t) = \frac{\ddot{y}\dot{x} - \ddot{x}\dot{y}}{\dot{x}^2 + \dot{y}^2} \tag{5}$$

Finally from Equation 3-5 we have,

$$\mu(t) = \arctan\left(\frac{v_a\, m\cos(\gamma)\dot{\chi}}{v_a\, m\dot{\gamma} + \cos(\gamma)gm}\right)$$

V. Experimental results

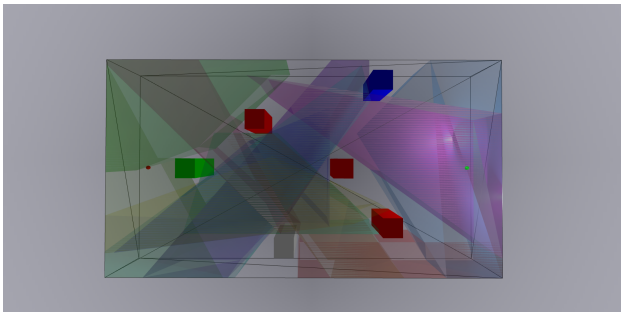
A. Implementation

The project was implemented in Drake[9] using Python. GcsTrajectoryOptimization was used with the commercial solver Mosek[5] and the IRIS clique coverage was implemented from scratch. Furthermore, the delta wing model was taken from Solidworks community [2] and converted to an urdf file. The obstacle-filled environment was inspired by [3]. The full code can be found in this GitHub repository. A video presentation of the project can be found here.

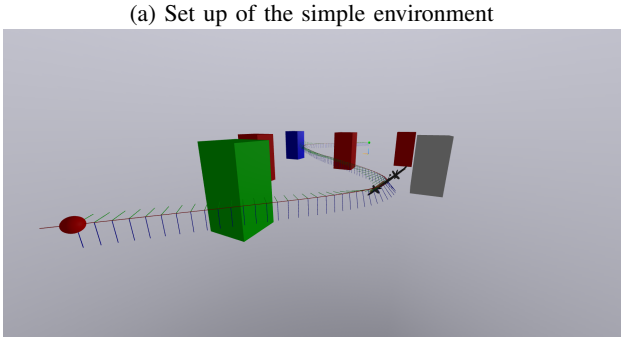
The simulation is running on a Lenovo Yoga Slim 7 with a 2,74GHz AMD Ryzen 7 CPU. As aforementioned the Bezier curves in each convex region were tested with order 7 with path constraint order up to order 4. The number of regions was kept below 20. Any increase significantly worsened the solve time from order of minutes to order of 10s of minutes. The mass (m=1.56kg), gravity (g=9.81m/s), and all other additional constant was taken from [6]. Additionally, the IRIS regions were generated by producing 100 random points and running the IRIS clique cover algorithm twice. This seemed to give the best result in terms of speed.

B. Demo 1: Simple Environment

A simple forest-like environment with a start in green and a goal in red was set up, Figure 6a. The Iris clique cover algorithm generated 12 regions, resulting in a reasonable trajectory solve time of 160s. An image of the delta-wing traversing the trajectory can be seen in Figure 6b.



(a) Set up of the simple environment



(b) The solution of the trajectory optimization in the simple envrionment

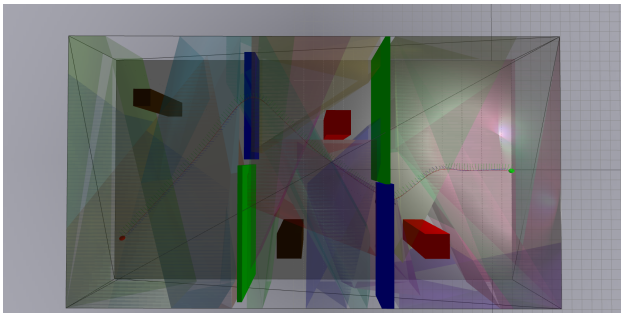
Fig. 6: Trajectory in simple Environment

C. Demo: Complex trajectory

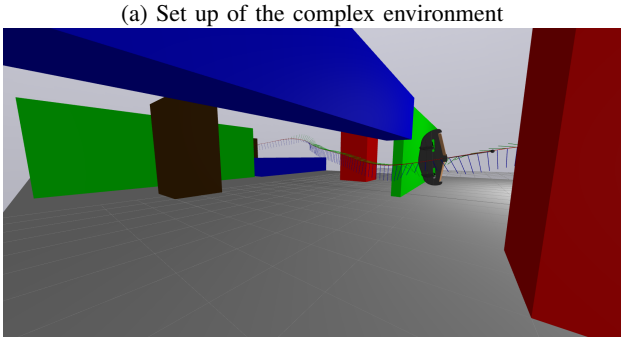
Moving on, a more complex environment was constructed as seen in Figure 7a. However, the number of iris regions from Iris Clique Cover increased to 15. This is a good indication of the underlying non-convexity of the free space as the generated regions are simply too small to cover enough of the free space and we need more of them. Consequently, the solve time for the trajectory optimization in GCS increased to 230s which is a significant increase in time from the simple environment case (43%), considering that only 3 more regions were added. As you can see in Figure 7c, the delta-wing is capable of traversing the environment without collision.

D. Weaknesses

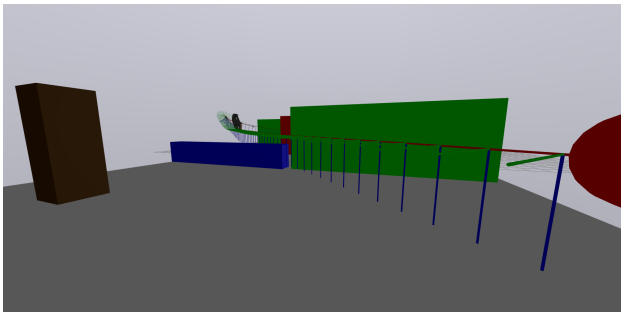
As discussed above increase in regions does increase the trajectory solve time. This is mainly due to the added edges between the regions increasing the size of the SPP that trajectory optimization has to solve. This poses an issue when dealing with highly dense obstacle-filled and/or cluttered



(a) Set up of the complex environment



(b) The delta-wing is able to traverse the trajectory without collision



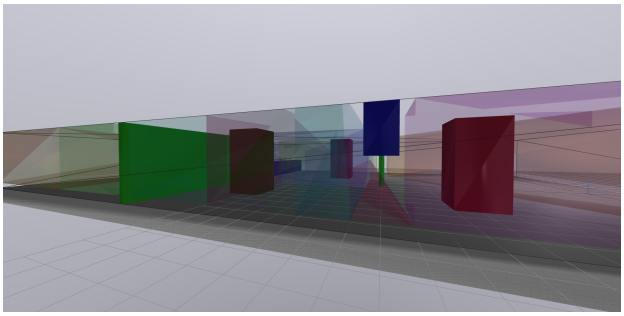
(c) The nature of the trajectory yields steep turns for the delta-wing

Fig. 7: Trajectory in Complex Environment

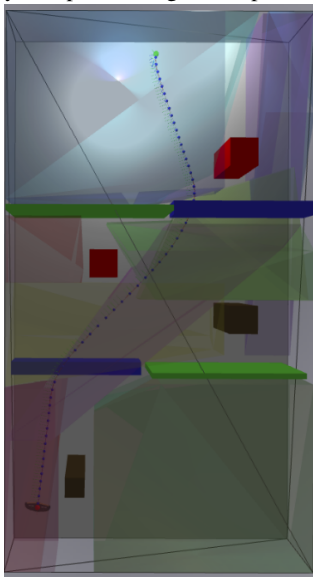
environments. To remedy this one would need a smarter way to partition the free space in a set of convex regions.

It is also worth noting that during sharp turns the plane is going to roll violently. Figure 7b is a good representation of this. We observe that the left wingtip is close to being in collision, but manages to avoid it because the obstacles were scaled with a margin equal to half the wingspan of the delta-wing. Although this might sound like a good solution, it also has the side effect of eliminating the possibility of finding trajectories in areas with slim rectangular windows in the wall or lower ground clearance on the obstacles. One could think that constraining the orientation in these cases could help, but that would only add non-convex constraints in the flat output space rendering the whole optimization non-convex.

When computing the convex regions, unless one computes to 100% coverage of the configuration space, there will always be regions and points that are unable to be reached. How large these nonreachable regions are, depends on how successful the region partitioning is. When running Iris clique cover only once in the complex environment, it yields a non-feasible optimization with 8 regions where the union of regions containing the goal and the start do not intersect as observed in Figure 8a, or that the goal is in an unreachable region as in Figure 8b. Increasing the number of iterations in the cliques cover algorithm did solve the issue, but did also increased the total number of regions and in turn increased the solve time. It turned out that running the algorithm twice was the sweet spot as it produced 15 regions compared to 21 regions and a very high solve time. Recall that increasing the number of regions from 12 to 15 increased the solve time by 45%.



(a) The two regions on either side of the wall are not intersecting and no trajectory can pass through this space.

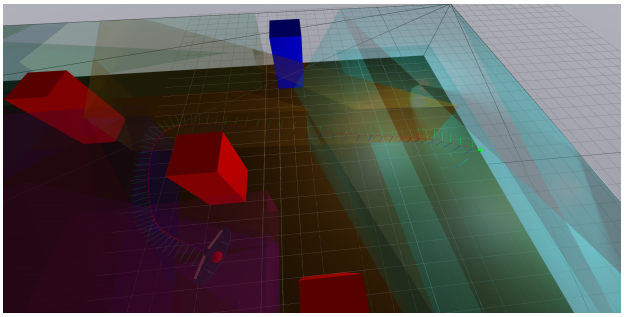


(b) The green region in the lower right corner is unreachable by the start and any trajectory to this region is unfeasible.

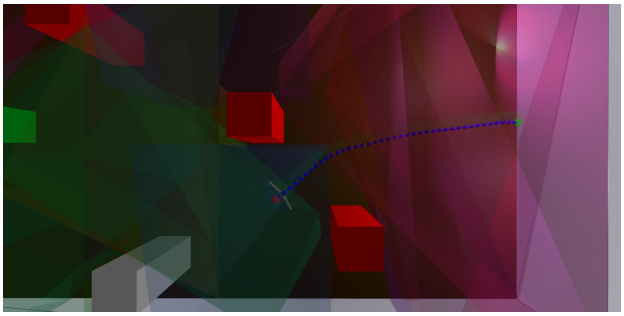
Fig. 8: Non-reachable areas in the configuration space

Given that a goal that is reachable from the start node is chosen, the solution is still not guaranteed to be the globally optimal path to the target unless this path is covered by a subset of the regions. Figure 9 is a good representation of such a problem. The goal, in red, is reachable from the start, but in Figure 9a the delta wing is forced to take a different path around one obstacle to reach the it. In Figure 9b the delta-wing can simply go straight as there now is a region connecting the

previously unconnected regions in Figure 9a. It is worth noting that both these trajectories are optimal, in the sense that they truly minimise the objective posed with the constraint they are subjected to. However, only the trajectory in Figure 9b minimized the path to the goal in a global sense (where the whole free configuration space is available for search). As mentioned this is because this global solution is included in the region partition in Figure 9b.



(a) The goal (in red) can only be reached if the delta-wing flies around the red obstacle.



(b) A different set of regions which now covers a direct path from the start to the goal and the trajectory optimization correctly picks this solution.

Fig. 9: Significance of the global shortest path solution being covered by a subset of the convex regions

VI. Conclusion

This project successfully demonstrated the application of the Graph of Convex Set (GCS) method for optimizing collision-free trajectories for fixed-wing UAVs in complex environments. By strategically decomposing the UAV's configuration space into convex sets using Iris Clique cover and mapping these onto a graph structure, we effectively transformed a traditionally nonconvex optimization challenge into a tractable convex optimization problem. The use of Bézier curves within each convex region as well as the use of the models flat output ensured that the trajectories were not only feasible but also adhered to the dynamic constraints of fixed-wing flight. Weaknesses regarding the reachability of the whole configuration space and truly global optimum were highlighted, as well as comments on the solve time of the different trajectories proving that increasing complexity rapidly increased solve times in a nonlinear fashion. Hence this implementation might not be mature enough for most real-world applications.

VII. Future work

During the project, I did not have time to properly implement the dynamics of the delta-wing to be able to simulate it (since Drake[9] does not support aerodynamic simulation) and thus was unable to test the Finite Horizon Linear Quadratic Regulator (FHLQR). For future work, this would be interesting to check out as well as testing out other controllers. [6] proposes using a PID controller since the latitudinal and longitudinal dynamics of the aircraft are decoupled. However, it would be interesting to test with more optimization-based controllers like the Iterative LQR (iLQR) used in an MPC fashion.

Furthermore, it would be interesting to try to implement more aggressive manoeuvres as has been done for other UAVs [7]. This could potentially be leveraged to tackle the challenge of the delata-wing not being able to fly through slim rectangular windows.

Lastly, exploring the possibility of partitioning the free configuration space more cleverly, for instance by exploiting the known structure of the scene to generate a minimal number of regions covering the whole space. However, it is worth noting here that this might only be feasible in an environment made of polytopes.

References

[1] Robin Deits and Russ Tedrake. "Computing large convex regions of obstacle-free space through semidefinite programming". In: *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*. Springer. 2015, pp. 109–124.

[2] Dx. *DeltatWin*. URL: https://grabcad.com/library/deltatwin-1.

[3] Bernhard Paus Graesdal. *Collision-Free Mixed-Integer Planning for Quadrotors Using Convex Safe Regions*. URL: https://github.com/bernhardpg/collision-free-mixed-integer-planning-for-uavs/tree/master/models.

[4] Tobia Marcucci et al. "Motion planning around obstacles with convex optimization". In: *Science robotics* 8.84 (2023), eadf7843.

[5] Mosek. *Mosek Otimization software*. URL: https://grabcad.com/library/deltatwin-1.

[6] Monika Pasquali. *Modeling and differential flatness based control for a convertible aircraft*. 2023.

[7] Ezra Tal, Gilhyun Ryou, and Sertac Karaman. "Aerobatic Trajectory Generation for a VTOL Fixed-Wing Aircraft Using Differential Flatness". In: *IEEE Transactions on Robotics* 39.6 (2023), pp. 4805–4819. DOI: 10.1109/TRO.2023.3301312.

[8] Russ Tedrake. *Underactuated Robotics. Algorithms for Walking, Running, Swimming, Flying, and Manipulation*. 2023. URL: https://underactuated.csail.mit.edu.

[9] Russ Tedrake and the Drake Development Team. *Drake: Model-based design and verification for robotics*. 2019. URL: https://drake.mit.edu.

[10] Peter Werner et al. “Approximating robot configuration spaces with few convex sets using clique covers of visibility graphs”. In: *arXiv preprint arXiv:2310.02875* (2023).