Documentation

Design and Architecture

I have implemented the solution using two different algorithms for the detection of conflicts.
I made the program modular with different files to handle different functions for both the methods:

1. Data Loader (data_loader.py)
   This file handles all JSON file operations. It loads the flight data and the primary drone data and checks for the format of the entered data.
2. Spatial Analyzer (spatial_analyzer.py)
   This file handles geometric calculations, path analysis and spatial conflict detections. It performs the operations – interpolation, position sampling, and spatial conflict detection.
3. Temporal Analyzer (temporal_analyzer.py)
   It handles time based conflicts i.e. when drones will be at the same position within a time threshold.
4. Conflict Detector (conflict_detector.py)
   It handles data from both spatial and temporal analyzers, combines their results and provides formatted conflict reporting.
5. Visualization Engine (visualization_engine.py)
   It handles all visualization tasks – 3D static plots, interactive Plotly visualizations and animations.

For the second method implementation, the code for temporal and spatial analysis are not implemented separately but implemented as single file spatial_analyzer.py only.

The control and flow of data in the program is as mentioned below:

1. The data that is entered in the JSON files is read and a count of the number of drones is displayed for error detection in reading the files.
2. The conflict detector creates an empty list to store the conflicts and extracts the waypoints and drone ids from the flight data.
3. Temporal and spatial conflict analysis is done and their results are stored.
4. A final list of conflicts is made by the conflict_detector.
5. A detailed report of the conflicts is then created with the type of conflict, timestamps indicating the points and the drones involved in the conflicts.

Spatial and Temporal Checks

FOR METHOD 1

Spatial Conflict: A spatial conflict is marked when two drones are within the safety threshold, given by the constant drone_radius that can be modified according to the drone size.
I have used the below algorithm to detect spatial conflicts

- Both the flights are sampled at 1 second intervals
- For each timestamp, the closest position between the drones is found
- Euclidean distance between the positions is calculated
- If the distance is less than drone_radius then it is marked as a spatial conflict
- In order to avoid spamming of conflicts, they are reported only if they occur after a time period of 2 seconds

Temporal Conflict: A temporal conflict is marked when two drones will occupy the same position within the safety window, given by the constant time_threshold that can be modified according to your requirements.
I have used the below algorithm to detect temporal conflicts

- Positional samples are generated for both the drones

- Find positions within 1m of each other (same location)
- Check if the time difference between the two positions is less than time_threshold
- If both the conditions are met then it is marked as a temporal conflict
- They are reported only if they occur after a time period of 1 second.

FOR METHOD 2

The flight path of each of the drones can be broken into small segments formed by connecting consecutive waypoints. Thereafter, the problem can be reduced to finding the minimum distance between two line segments. If we check for the points at the minimum distance for spatial and temporal conflicts we can cover most of the possible test cases.

Now, this is purely a vector algebra problem. I searched on math stackexchange and found an algebraic solution to this problem (https://math.stackexchange.com/questions/846054/closest-points-on-two-line-segments). I used ChatGPT to convert the majority of the solution into python code. There were several bugs though and some inaccuracies in the posted answer. Hence, a good understanding of the solution is required to correct them.

The method involved in the solution is shortly described here: -

Let the end points of the two line segments be (P1, P2) and (Q1, Q2) respectively. Then the line segments can be represented parametrically as:

$P = P1 + s (P2 - P1)$
$Q = Q1 + t (Q2 - Q1)$

Where the parameters s and t lie between 0 and 1. The line that joins the two points on these lines, such that the distance between them is the shortest,

will happen if this line is perpendicular to both the line segments. Using two equations from these conditions, we can find out the parameters s and t.

If the calculated values of s and t are between 0 and 1, then these points lie inside the line segments. If so, we can just output these points.

Otherwise, we will have to find the point on Q closest to P1, the point on Q closest to P2, the point on P closest to Q1 and the point on P closest to Q2. If any of these points lie inside the line segments, we can output these points. If not, we calculate the distance of the end points from each other and give the best result.

Comparative advantage of both approaches

Using the approach in Method 2, we avoid doing a large number of calculations for each timestep as in the other approach. We only need to perform a fixed number of calculations for each pair of points. This saves a lot of computation for each pair of waypoints, making the overall process a lot faster.

This approach can be used to handle query in cases where a lot of waypoints need to be processed, for example in cases where there are more drones or when each travels longer paths and for larger time durations.

On the other hand approach 1 handles cases in which more regions of conflict are more important to identify. It can also handle cases of parallel flight paths better. In short, it provides a complete solution at the cost of time.

It can be used in cases when accuracy is a bigger concern than time cost of queries, which are rare.

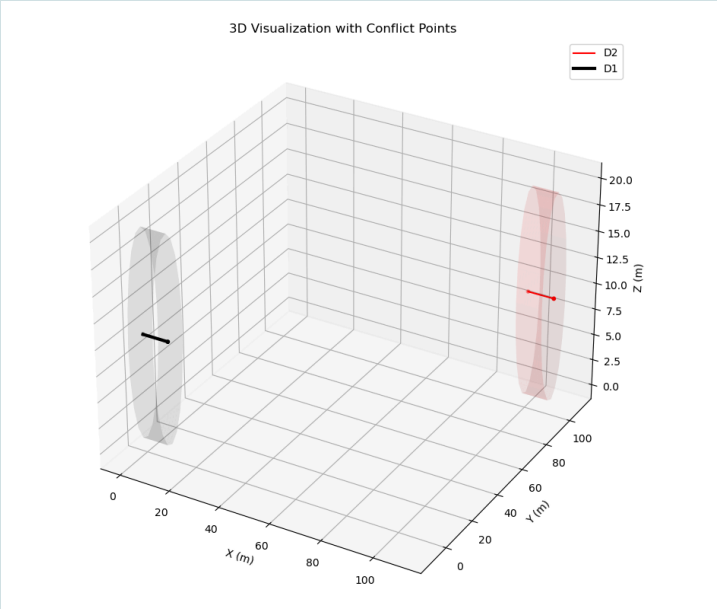
AI Integration

- To simulate multiple drones and their flight paths I used ChatGPT to generate the waypoints and timestamps.
- To debug errors in the code I used Windsurf and Claude AI
- To calculate the shortest distance between two lines and the time of flight at that point I found a thread on StackExchange. I used ChatGPT to code the mathematical calculations and the various cases mentioned.
  https://math.stackexchange.com/questions/846054/closest-points-on-two-line-segments

Test Cases and their Results

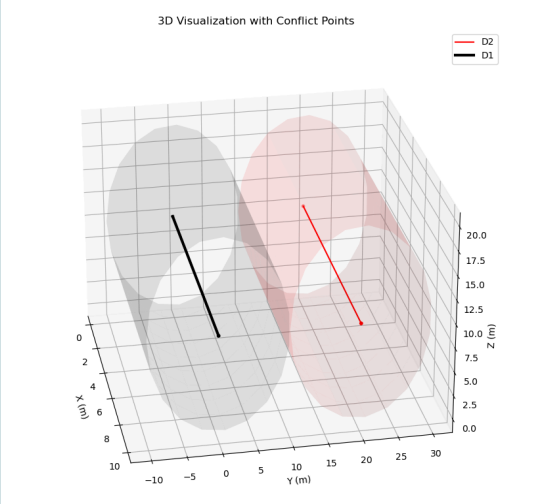
For simplicity, I have considered only two drones and simulated various scenarios that might be possible.

1. Completely different flight paths - tc1_different_paths
   Output:



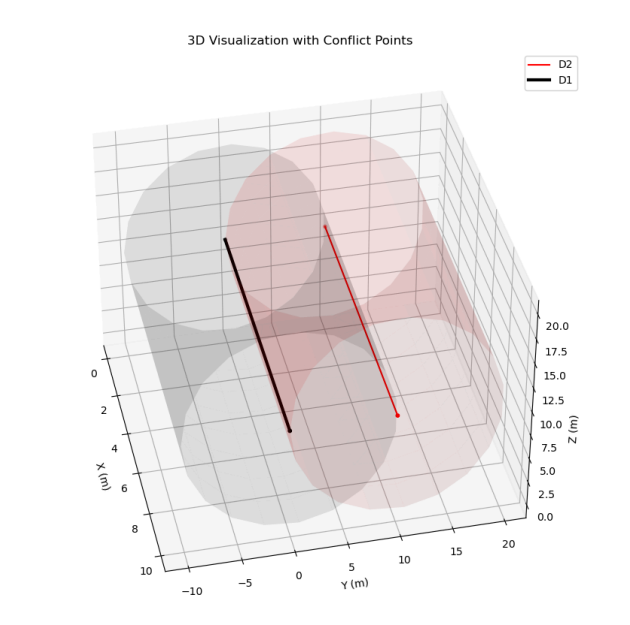
   No conflict

2. Parallel flight paths where distance between them is greater than the distance threshold - tc2_parallel_gt_threshold
   Output:



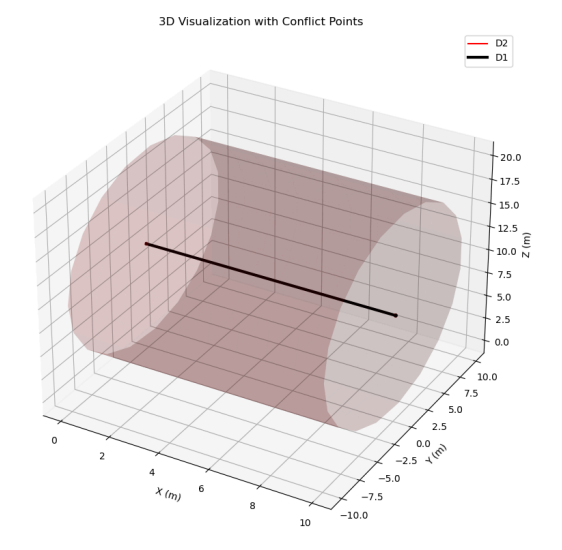
   No conflict

3. Parallel flight paths where distance between them is just equal to the distance threshold - tc3_parallel_eq_threshold
   Output:

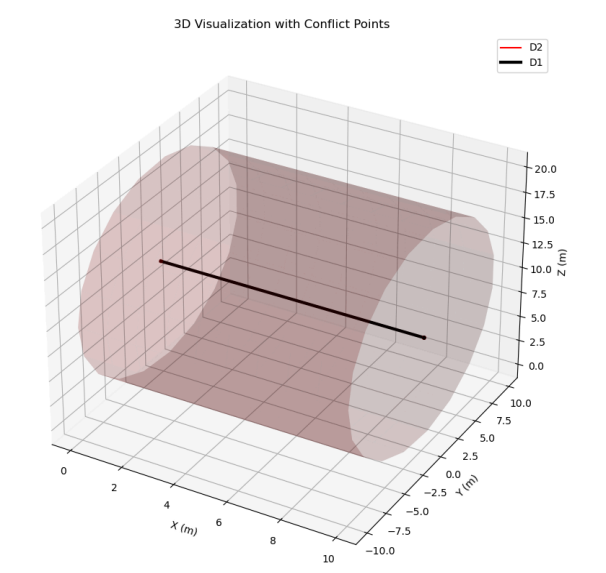


   No conflict

4. Same flight path but different timings - tc4_same_path_diff_time
   Output:



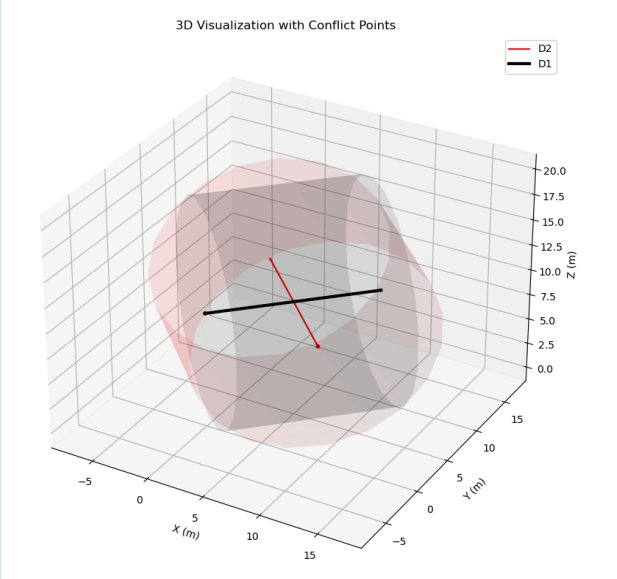
   No conflict
5. Same flight path but timings just beyond the time threshold maintaining spatial threshold - tc5_same_path_beyond_threshold
   Output:



   No conflict

6. Flight paths intersecting but cross the intersection point at different times - tc6_intersect_diff_time

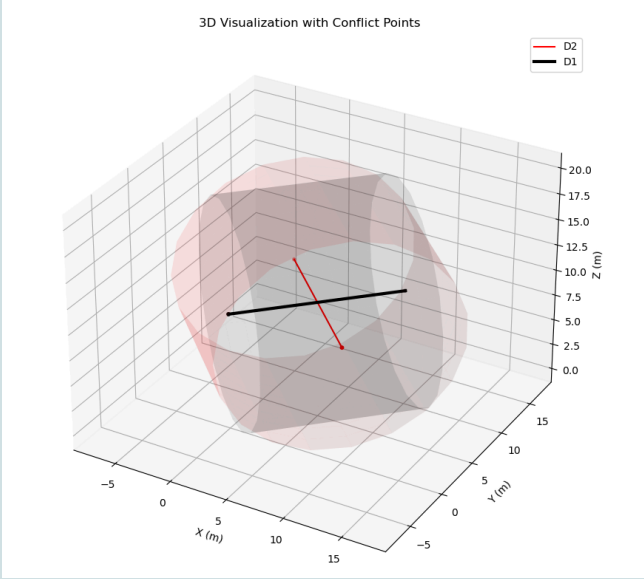
   Output:



   No conflict

7. Flight paths intersecting but cross the intersection point just beyond the time threshold maintaining spatial threshold - tc7_intersect_beyond_threshold

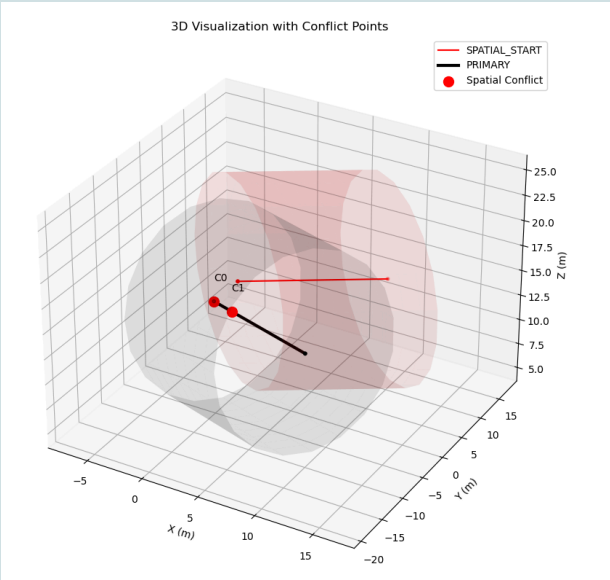
   Output:



   No conflict

8. Distance between flight paths less than threshold only at the start - tc8_spatial_start

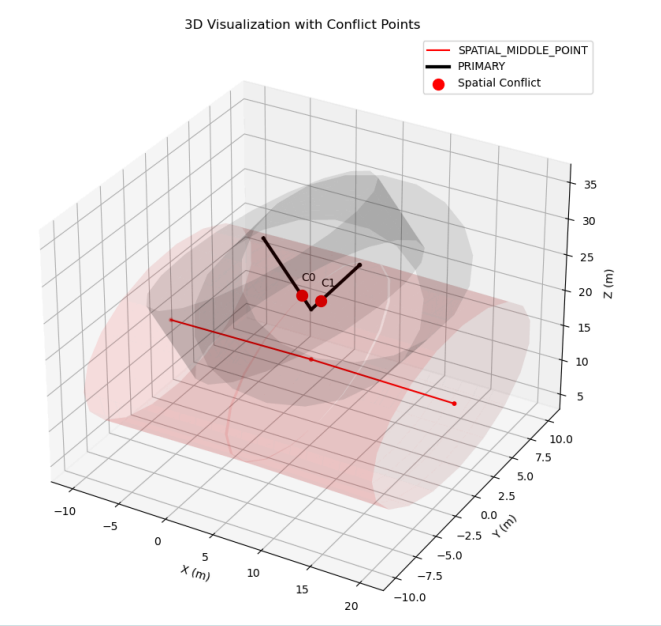
   Output:



   Spatial conflicts only

9. Distance between flight paths less than threshold for only a few points in the middle - tc9_spatial_middle_point

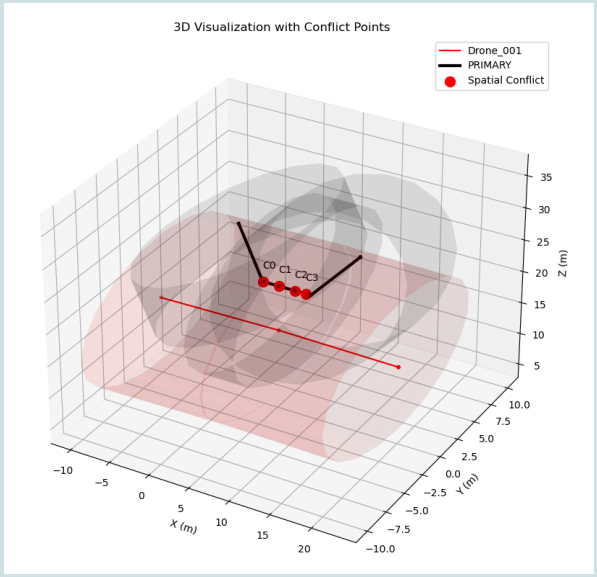
   Output:



   Spatial conflicts only

10. Distance between flight paths less than threshold for a short strip in the middle - tc10_spatial_middle_strip

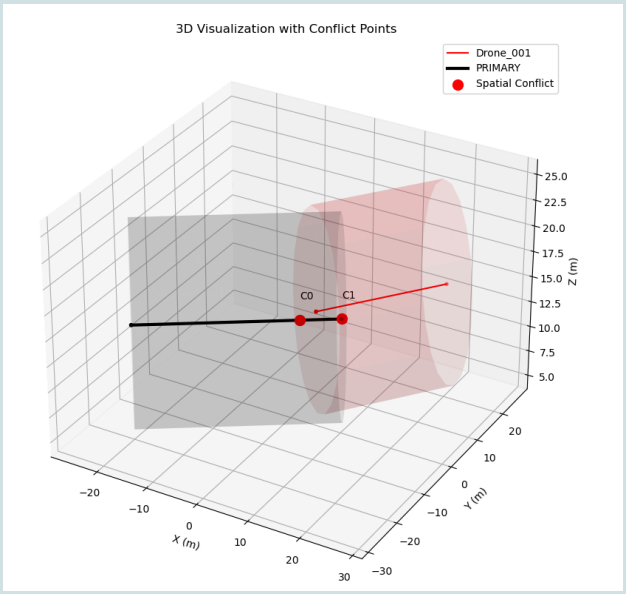
    Output:



    Spatial conflicts only

11. Distance between flight paths less than threshold only at the end - tc11_spatial_end

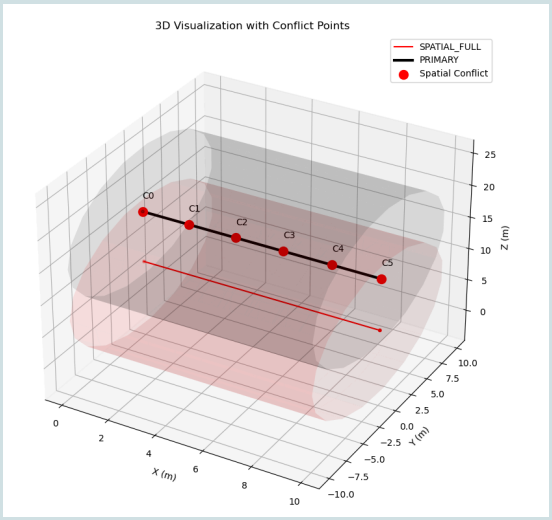
    Output:



    Spatial conflicts only

12. Distance between flight paths less than threshold throughout the path - tc12_spatial_full

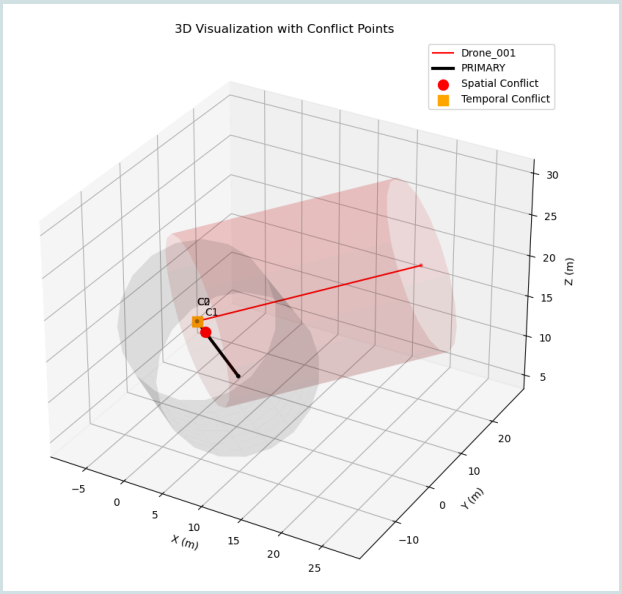
    Output:



    Spatial conflicts only

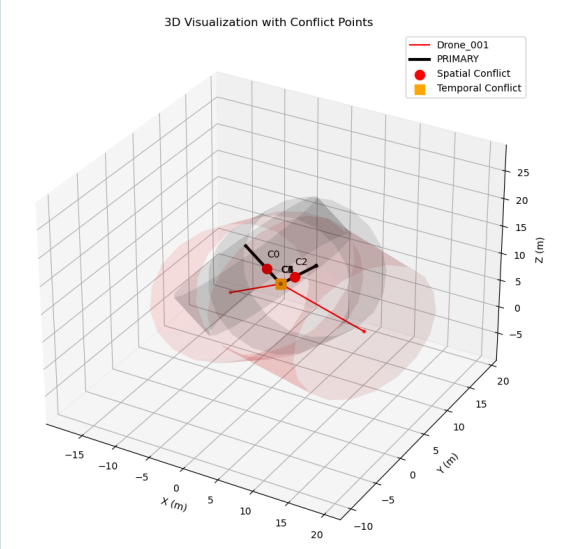
13. Position during flight is the same for a window less than threshold only at the start - tc13_st_start

    Output:

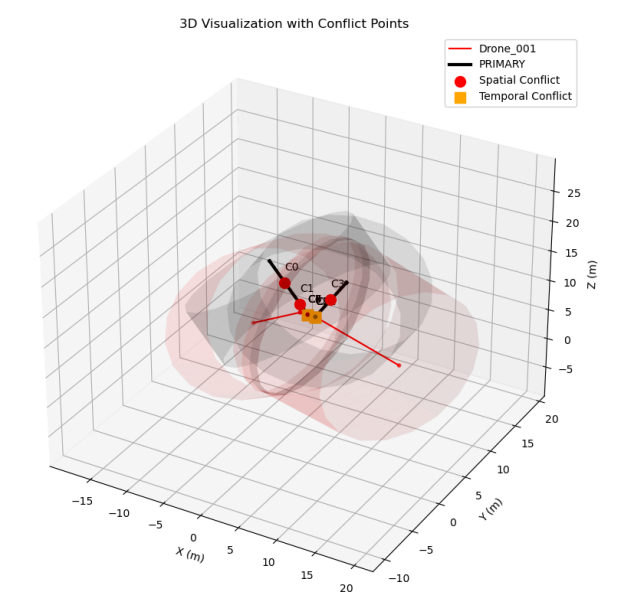


    Spatial and temporal conflicts

14. Position during flight is the same and distance less than threshold only for few points in the middle - tc14_st_middle_point
    Output:



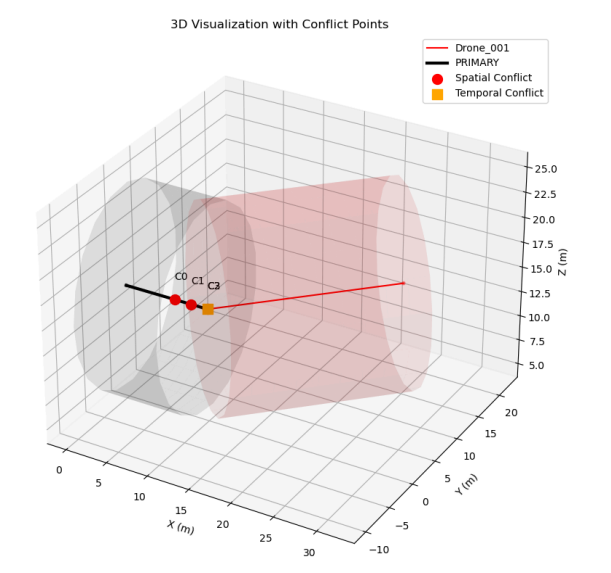
    Spatial and temporal conflicts
15. Position during flight is the same and distance less than threshold only for few points in the middle - tc15_st_middle_strip
    Output:



    Spatial and temporal conflicts

16. Position during flight is the same and distance less than threshold only for few points in the middle - tc16_st_end

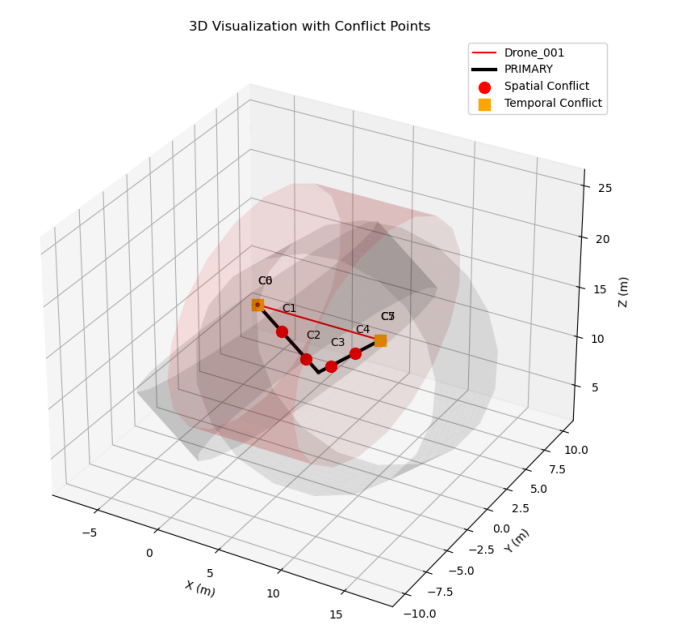
    Output:



    Spatial and temporal conflicts

17. Position during flight is the same and distance less than threshold throughout the path - tc17_st_full

    Output:



    Spatial and temporal conflicts

Scalability

I have in my solution considered the data of the drones to be present in a formatted JSON file. To perform deconfliction for 1000s of drones the data could be in CSV format for faster computation and evaluation.

In the current solution, I have only considered the timestamps and waypoints as data to detect conflicts. However, in real-life to ensure that small changes in position are also accounted for we can use the data from IMUs, air speed sensors, etc to make more accurate fault detection.

Further, from the conflict points we can use an algorithm to suggest modified waypoints in the Primary drone's flight path such that the conflicts are avoided with minimal changes in its flight.