236874 - Project in Computer Vision

System for Smooth Path Planning and Flight Control

Yoav Sahar
Guy Sudai
Maxim Wainer

Source code can be found at: https://github.com/GeckoWarrior/DroneProject-02360874.git

Introduction

UAVs are becoming mainstream and destained to acheive a wide varity ot tasks - from aerial photography to package delivery, and even search-and-rescue missions. With the increasing demand for drone utilization, new challenges arises and a need for better algorithmics and control over them.
In this project, we showcase an **end-to-end system for smooth path planning and flight control** for the DJI Tello drone in a challenging environment, with an emphasis on path efficiency and compatability to the drone's capabilities.

Keywords: A*, Spline, Motion capture, flight control.

Problem Definition:

Given a start point S, an end point E and a set of 2D obstacles O, navigate a (physical) drone from S to E in an efficient (= short and with native turns) smooth path, while ensuring obstacles avoidace.

We part the problem into two main tasks:

1. <u>Path planning:</u> Given S, E and O, generate a smooth and short path from S to E which ensure a safe distance form all obstacles in O.
2. <u>Flight control:</u> Given a target path, control a drone to follow it with a minimal error and in relatively high speeds.

This seperation of indenpedant parts also allows a wider use of our project. One may utilize our path planning algorithm for different uses, other might take our flight control system for different drone missions.

Problem Overview

Traditional path planning algorithms for drones, like A^*, usually break down the flight path into discrete steps. This means the drone flies from one point to another in straight lines, turning sharply at each waypoint. While this works, it comes with several drawbacks. First, these sudden changes in direction or speed are not energy-efficient. The drone has to slow down, adjust its trajectory, and then accelerate again, which wastes more power and shortens its battery life. Second, sharp turns put additional stress on the drone's components and payload, leading to faster wear and tear of the drone and generally making it less reliable.

Smooth motion, on the other hand, offers a solution to these problems. By creating a continuous, curved path for the drone to follow, we can reduce sudden changes in speed and direction. This makes the drone's flight more stable, energy-efficient, and gentle on its hardware. The idea is similar to how a car takes a bend on a road. Instead of making a sharp turn, it gradually follows the curve, making the ride smoother and safer.

Challenges

A generation of a smooth path and the control of a drone over it raises new challenges.
For instance, there are many ways to represent a curve, each represtation with it's pros and cons.
Lagrange interpolating polynomials have nice properies as they have a closed form formula (which enables easy intersection calculations) and all the nice mathematical propetries of a polynomial. However, it is hard to enforce their total position and they tend to be unstable (strong oscillations) with adding more control points - limiting its use for more comlex path demands. B-Splines or Bezier curves are much more flexible and controllable, but lacking a close form which makes the intersection calculation which are necessary for obstacle avoidance harder to plan and implement.

With a smooth path, comes smooth path following. This demand prevents the use of built-in functions of the drone s.a. flight in straight line functions with height control, rotation in place, and other useful function for a "classic" programmed motions. Let alone the fact that in the lab it was believed that an "operator like" control was not possible with Tello via code, a lot of parameters and inner workings of the drone are not avalible online and required a lot of testing and research to get answers.

More challenges arised along the project, and the main ones are addressed in the following sections.

Sidenote: At the begining, we base our work on an atricle that later turned out to be misleading, which posed us with challenges of adaptation and the need for new solutions.

The Solution

As mentioned above, our solution compromises of two main parts: The path planning algorithm and the flight control system. Combined togther, we get the ability to navigate the Tello drone through a complex virtual environment.

The Path Planning Algorithm:

input: list of obstacles, starting point and destination point.
output: B-spline describing the path from the start point to the destination point. the return spline is ensured to not encounter any obstacle.

way of work:

1. Initialize a set of control points using A* star algorithm on the visibility graph.

2. Add control points on the path of the A* uniformly s.t. the maximum distance of two sequencial control points is not greater than CONTROL_POINTS_MAX_DIST.

3. Goes from the 4th point to the last* point, for each point do:

3.1. If the convex hull created by the last 4 points (indexed i-3 : i) intersect with any obstacles (meaning the rule* is broken):

3.1.1. For the 3 points prior to the current point, only if it doesn't break the rule, move each point away from the intersecting obstacle by POINT_DISTANCE_FROM_OBS_FACTOR. This allows us to satisfy the *rule* by rotating the current point less in 3.1.2. This part will not move the first point even if moving it doesn't break the *rule*.

3.1.2. Align the current point to be collinear with the previous two points, then rotate the current point around its previous point left and right until the rule is satisfied. If the rule is not satisfied after rotating 180 degrees to each side, move the point closer to its previous point and try again do that over and over until the rule is satisfied (or until the current point gets to distance of CONTROL_POINTS_MIN_DIST or less from the previous point, in that case we put the current point on top of the previous point).

Check if the last control point the really the destinaion points (it is possible that it moved in 3.1.2), if so, adds a new control point on the destination point.

3.2. Look for the next point: the next point the preferred to be the latest point (with the greatest index, closest to the destination) that is visible to the current point w.r.t the obstacles. If there is no such point, the next point is set to be the destination point. remove all the control points between the current point and the chosen next point.

3.3. If the distance between the current point to the next point found in 3.2 is greater than CONTROL_POINTS_MAX_DIST, then add points between the current point and the next point uniformly so the maximal distance between 2 points is CONTROL_POINTS_MAX_DIST.

4. Repeat until this has no effect:

> **4.1.** For every point, from start to end (second : to second to last, to be specific), check if the current point can be moved to the middle of it with its neighbor points, with increasing weight to the current point (weights from 0 to FLATTEN_LEVEL_MAX). If it can be moved there without breaking the rule, move it the middle with the least weight to the current point possible.
>
> **4.2.** For every point, from end to start, check if the point can be removed without breaking the rule. If it can, remove it.

5. Create and return the normalized B-spline of degree 3 according to the control points acquired.

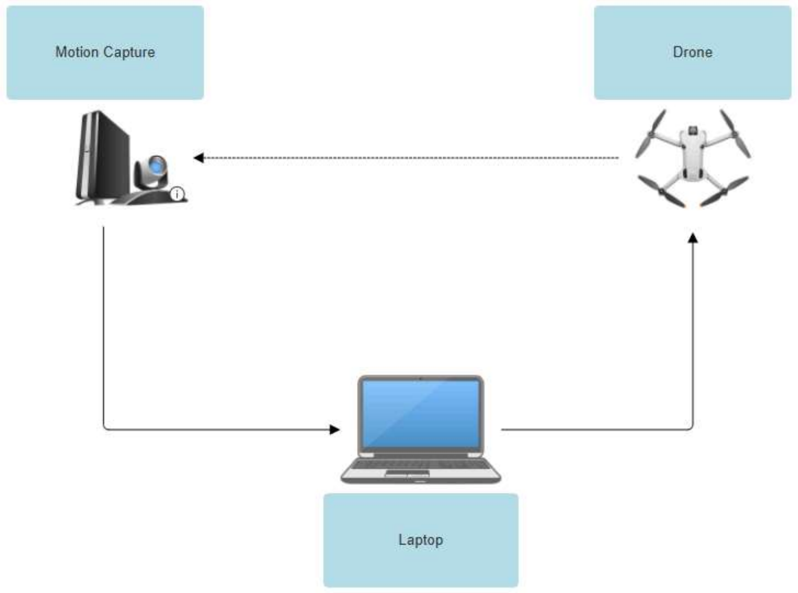
Algorithm notes:

* here we are talking about the control points in a way that the first control point is indexed 0 and is not repeated, and the last control point indexed $len - 1$ and is not repeated.
When the spline is built, the first and last control points are included 3 times in order to ensure the spline starts and the first control point and finishes at the last control point.
* When mentioning the "***rule***" we mean that for every 4 sequential points in the control points, the convex hull of these points does not intersect any obstacle. Due to the convex hull property of B-splines, this rule ensures us that the spline won't intersect any of the obstacles.
* In part 3.1.2, we can ensure this part doesnt gets stuck since in the worst case we put the current point on top of the previous point, which means the convex hull of the last 4 points is the same as the convex hull of the last 3 points. Since the convex hull of the previous last 4 points (the previous point and its 3 previouses) doesn't intersect with any obstacle, so is the convex hull of the last 3 points, and so is the convex hull of the current last 4 points.

Flight Control System:

The purpose of the flight control system is flying the drone along the planned path.
The system sits on a laptop, accesses the drone posotion via motion capture capabilities installed in the lab, and send commands to the drone over a wifi connection.

System overview:



After many research and testing, we manged to obtain the following:

- ***rc_controller*** command, which allows us to send "operator like" commands at a high rate. (This commands specifies how much force should be applied on the drone in each direction x,y,z and and rotation w - just as in a regular controller.)
- Stable **speeds**, and **height flights** for the drone.
- **Rate of commands** - one issue was finiding how fast the drone is able to process commands:
  - Send commands too fast, and its inner **command buffer** would fill, causing lag and refusal of accepting more commands.
  - Send commands too slow, and the drone will treat the space between commands as a "don't move" command, causing a buggy flight.

  We found that a good rate was 10 cmd/sec.

With this information in hand, we were able to construct a system to follow the planned path.

The idea is simple, at each frame we will simulate the ***drone's reach,*** and send the corresponding command the achieves maximum "legal" advancement in the spline (By "legal" we mean that we don't skip parts of the spline in cases of spline overlapping). Our system works on the Tello drone and with a Motion Capture system to track its positions and fix accumulation of errors, but also has has other running modes: Without Mo-Cap, then we simulate the expected positions and fly the drone accordingly. Without Tello, in that case we simulate the entaire process int the computer and show expected results.

We highlight a few important notes from the system:

- **Drone's reach** - The next furthest point the drone can reach. For that we tested 2 types of reaches:
  - Spheric reach - in this approach we checked if $||\overline{v}||_2^2 < C$.
  - Cubic reach - in this approach we checked if $||\overline{v}||_1 < C$.
    Note $\overline{v}$ is the velocity vector, and C is a constant we calculated which represents the distance the drone could reach in a single iteration.

  To calculate the Drones target in each iteration, we initialy set the target to be a point close on the spline, and while it is within the reach, choose a point a bit further on the spline. With the target point aquired, we translate the velocity on each axis to the matching command format and send it.

  Although spherical reach might be physically more accurate, in practice with cubic reach we obtain much better results. This is due to 2 main reasons:
  - The drone's command format itself is represented with cubical forces (each axis bounded at -100 to 100).
  - Cubic reach allows independant axis corrections, giving max power at one axis does not limit others, while in sphrical reach this case will force the others to be 0.
- **Simulation** - Flight time is precious, therefore pre-calculating an approximation of the drone flight helps save it. It also serves as a testing area for modifications like limits and constants, various approaches to estimate the reach and other different changes that can be tesed prior to flying the drone itself.
- **Initialization** - At take off, the drone needs a couple of seconds to stabilize before getting any command. Any command sent before the drone stablizes either will be ignored or cause unpredicted behaviour.
- **End condition and breaking** - To verify the drone gets to the end safely(for itself and for the environment) we set a condition - if the placment on the spline to end is less then a pre-defined constant, the distance between the position and the end small enough and the velocity is slow enough, then stop and land. To ensure that the drone is slow enough, we added a breaking mechanism: At each iteration check if the drone is at the last 10% of the spline, if so, make the reach smaller (meaning, take smaller steps). As the drone gets closer to the end the reach is getting smaller and velocity is getting slower.
- **Emergencies** - The equipment we worked with (the Mo-Cap system and the drone itself) is not the most accurate and reliable therefor we needed to take some precautions. In our case landing immediately. The cases we took into consideration:
  - Partially visible/Invisible: sometimes the Mo-Cap system doesn't detect the drone completle which leads to improper readings.
  - Out of bounds: when the drones position is outside the designated area the drone gets partially visible/Invisible and can hit other equipment in the lab.

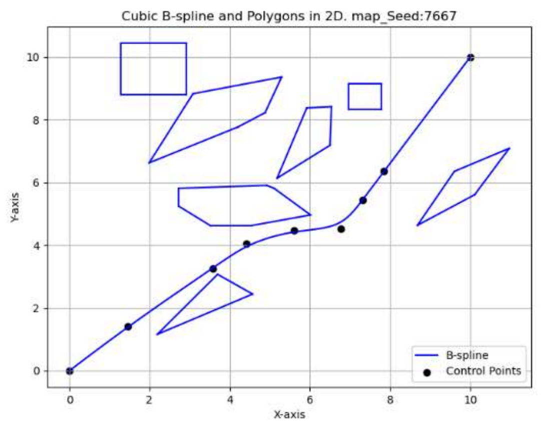
  - Lost connection: when a connection between any system is lost false data can pass or not at all.
  - Too much time passed: the flight time should be short, so when the flight time passes a pre-defined limit it indicates something is wrong.
  - Manual break: in any other case that is not predicted we left an option to stop everything immediately.

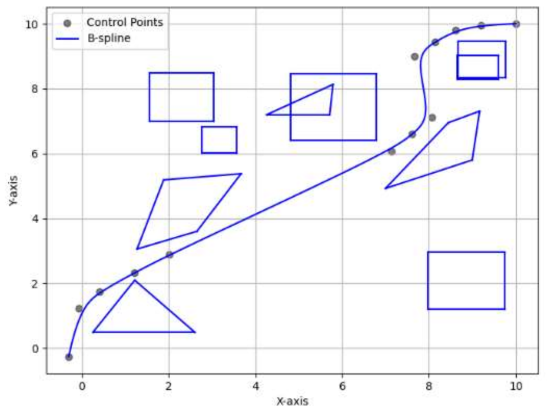
  If the flight was ended because of an emergany, there is a flag that indicates so, and an appropriate print at the console.
- **Data recording** - each flight is different from another so to get the abilty to revise paths or for further research, we support flight recording, meaning we save the actual positions, the path(spline), obstacles, the aim points and the given commands.

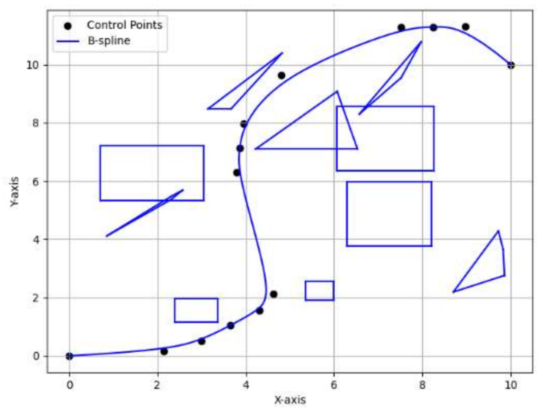
Our system achieves a low error path following of and general spline, multiple running modes, logs and flight data.

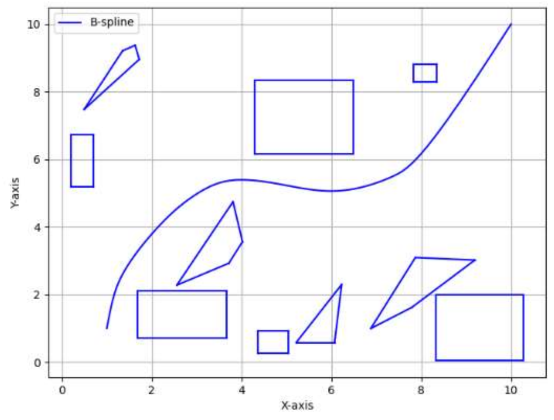
Results

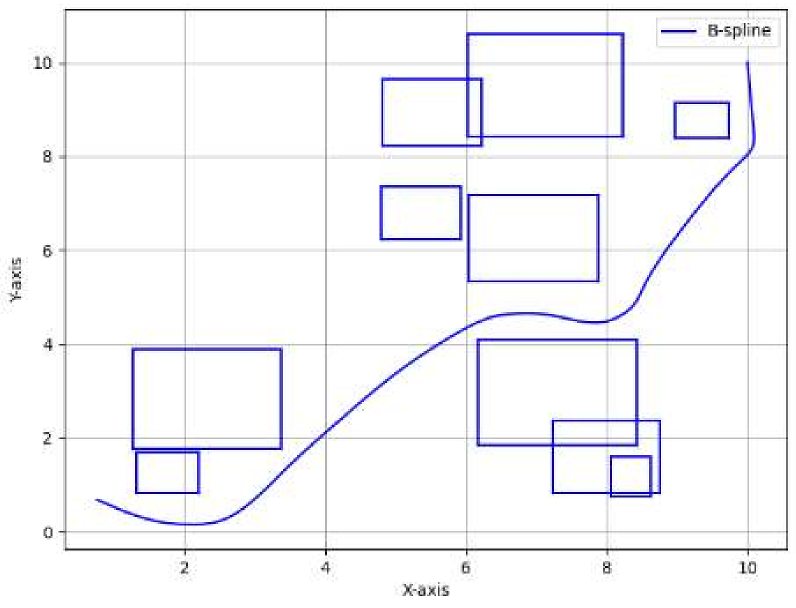
Examples of algorithm's output:



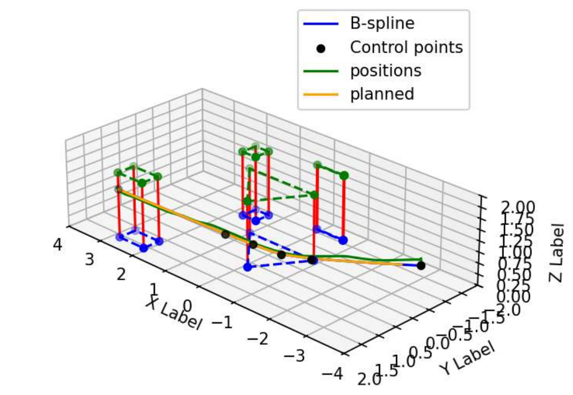
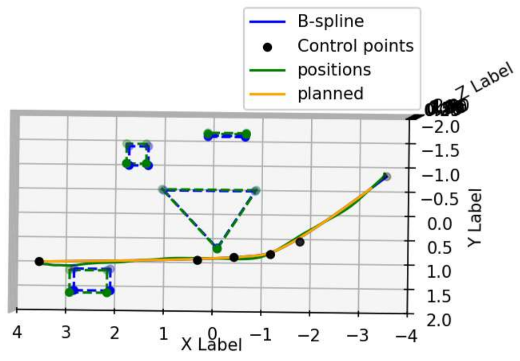
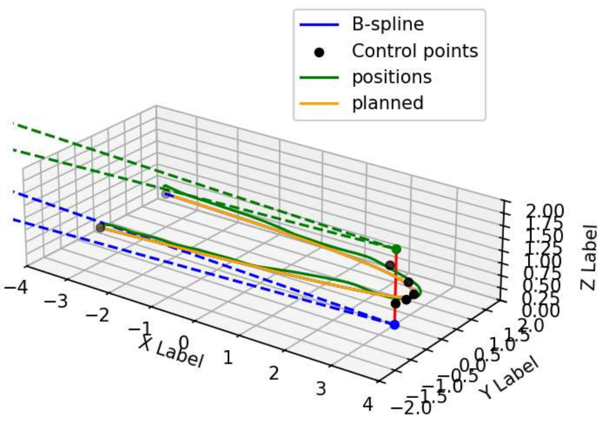
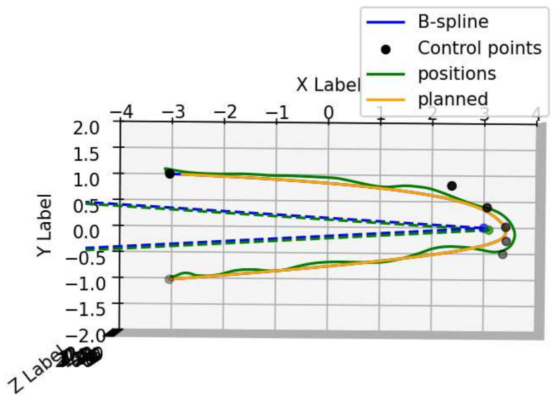



Control Points
B-spline
Y-axis
X-axis
0
2
4
6
8
10


B-spline
Y-axis
X-axis
0
2
4
6
8
10

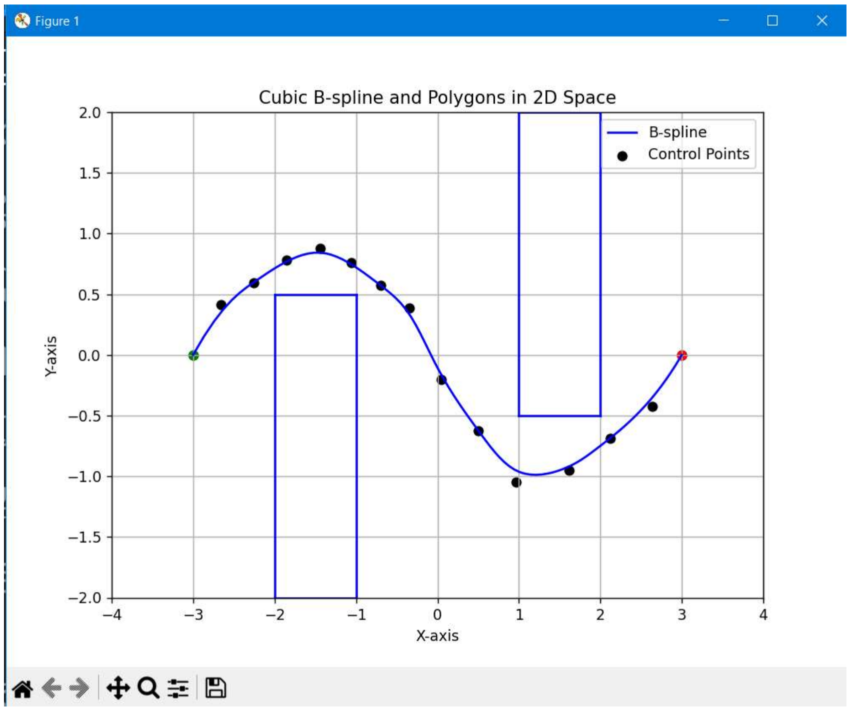

B-spline
Y-axis
X-axis
0
2
4
6
8
10

Examples of the flight control system:

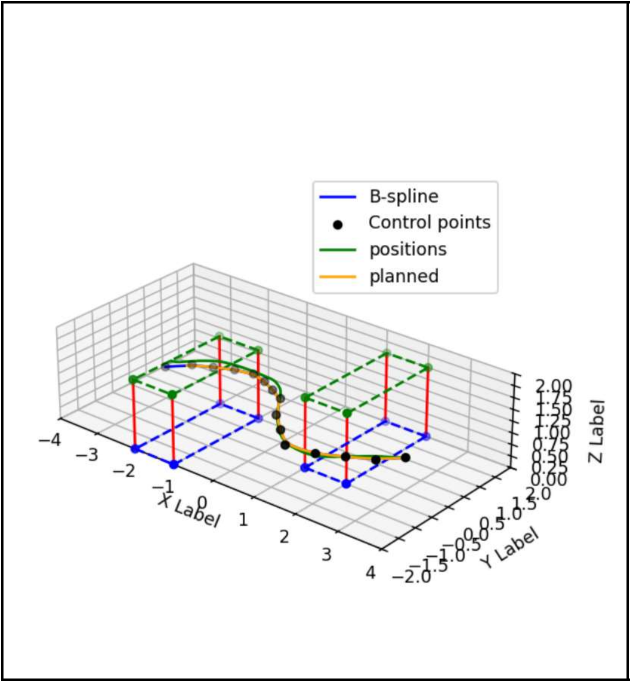
Side View:	Upper View:
	
	

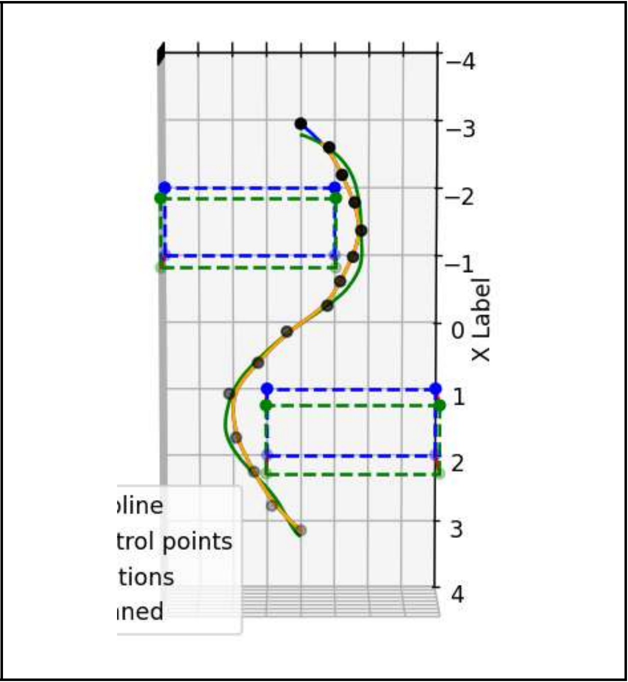
Example of an end-to-end run:

The algorithm output:



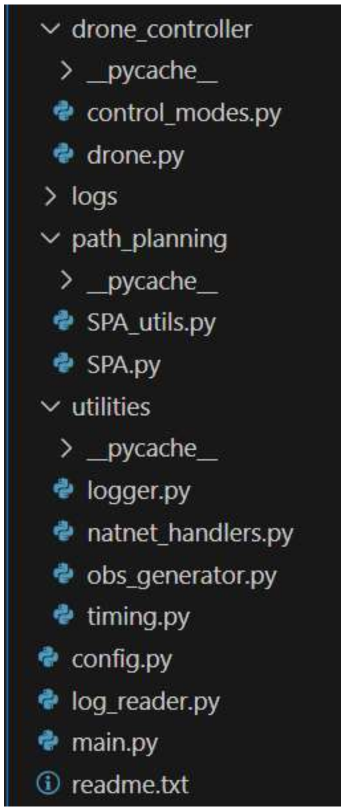
Flight results:





The video of the flight is attached to the report as a sepereate file.

Files Layout



```
drone_controller
  __pycache__
  control_modes.py
  drone.py
logs
path_planning
  __pycache__
  SPA_utils.py
  SPA.py
utilities
  __pycache__
  logger.py
  natnet_handlers.py
  obs_generator.py
  timing.py
config.py
log_reader.py
main.py
readme.txt
```

To support the entire project, we distributed it across some files.

main - Glues all toghter to a simple process: First path planning, then flight, and in the and a record of the results.
config.py - Here all project parameter can be configured: start point, end point, obstacles, flight parameters, algorithm paramaters and more. Everything the user might need from the project is handed in this file.
path_planning - This directory contains all files necessery to implement the path planning algorithm. **path_planning/SPA.py** hands the **plan_smooth_path** function to be used outside.
drone_controller - This directory contains all files necessery to implement the flight control algorithm. **drone_controller/drone.py** hands an object to easily interact with the tello, and **drone_controller/conrol_modes.py** grants support to different flight modes.
utilities - Few helpful functions, among them a logger to display and save flight and path data. natnet handler to support the Mo-Cap system in the lab, and more.

https://github.com/GeckoWarrior/DroneProject-02360874.git

Future Work

Here we offer some ideas following this project:

- **Rotation compatability:**
  In our project, for the drone movement we used the ability of the drone to move on each axis without changing its orientation. We encourage future works to explore the rotation ability of the drone (facing). A future feature that could be developed with rotation control is setting a point that the drone will be diracted at. The point could be either static or dynamic, and the drone will face drone during flight. Such feature can enable filming with drone camera, scanning the terrain or even enable real time bug-navigation with object detection. Also, better control over the drone's controls might enable better filght performance overall. We have tried to add rotation control, but derivation of the physical equations for angular velocities along with our reach algorithm was out of our scope of time. One important challenge we want to highlight is that with incorporating angle control as well, we get 4 axies for 3 dimensional world - which means there is a degree of freedom in the conrol. This introduce new challenges and research opportunities to understand what could be optimal solution. We think this feature could be with value for 2 main reasons. First, this possibly will allow the drone reach higher velocities as there is more control of the drones movment and rotation. Second, rotation of the drone enables using its camera in more diverse ways, rather than being pointed at one point, leaving more research opportunities and usage options.

- **Open Loop Control System:**
  While our project enables filght without Mo-Cap available, the preformance is poor. Currently, flight parameters and drone capabities are impirically hard coded, and still far from perfect. We have tried training various ML models on flight data we captured to predict and emulate the drone's flight better - the main issue is to precisely predict the "reach" of the drone on every iteration. With our limited time we failed to achieve better results with a ML model, but we believe that it possible with proper feature extraction, and with physics-based model. With better "reach" prediction, open loop control, a true challenge with much need, might be feasable. Also, better "reach" will enhance the closed loop control preformance, and enable higher velocities with lower error.

References

https://www.nature.com/articles/s41598-024-65463-w?fromPaywallRec=false

https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello%20SDK%202.0%20User%20Guide.pdf

https://github.com/ratcave/natnetclient

https://github.com/GeckoWarrior/DroneProject-02360874.git