POLITECNICO MILANO 1863

Image Analysis and Computer Vision Final Project

Topic F02: Reconstructing drone trajectories and cameras frommultiple images

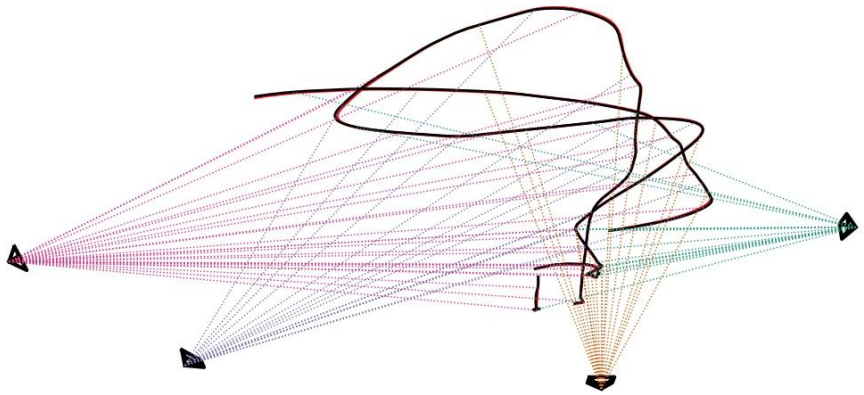
10,07,2025 I Elia Pontiggia

Contents

Problem formulation

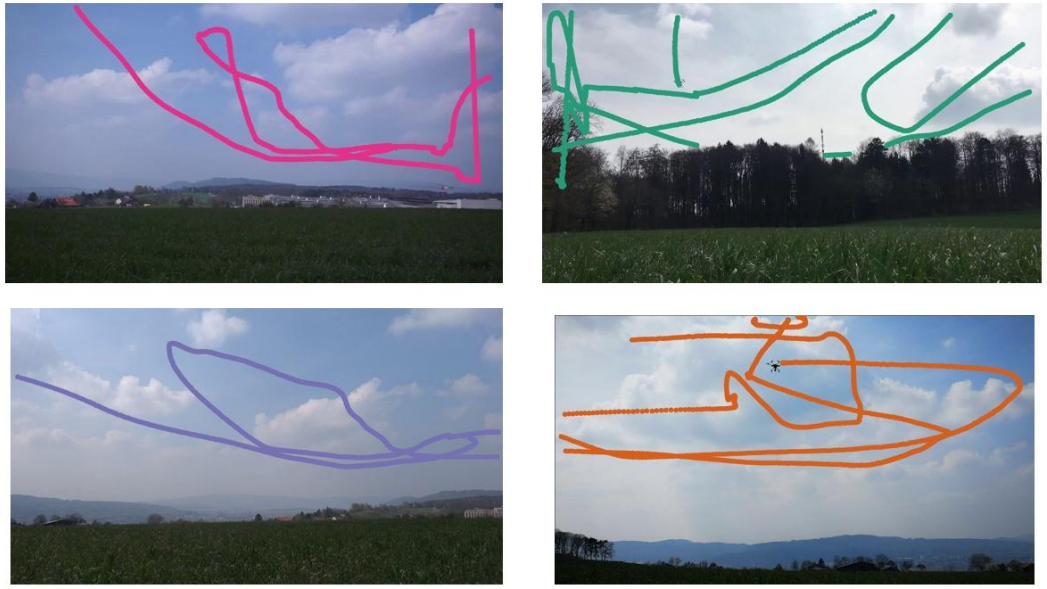
01

The problem



The main goal of the project is to develop a pipeline that tracks in the 3D space the motion of an Unmanned Aerial Vehicle. The pipeline should be:

- Accurate
- Scalable
- Cheap (no constraints on the sensors that detect the movement)
- Flexible
- Easy to deploy



To improve the accessibility, the reconstructions is based on videos taken from a network of unsynchronized, uncalibrated cameras.

State of the art

The work by Li et al., addresses this challenge by introducing a novel method for reconstructing the 3D trajectory of a UAV using only videos recorded by cameras. They have unknown poses, frame rates, and rolling shutter distortion, and are placed independently around the flight area. Thes ystem relies solely on known intrinsic parameters of each camera and automatically estimates all other variables during processing.

Reconstruction of 3D flight trajectories from ad-hoc camera networks

Jingtong Li[1*], Jesse Murray[1*], Dorina Ismaili[2], Konrad Schindler[1] and Cenek Albl[1]

https://arxiv.org/abs/2003.04784

This project is inspired by their work, and aims to implement the same architecture

Implemented features

Our project closely follows the methodology outlined in the paper, but it is not as complete as the original work

Paper aspect	Implemented	Notes
Camera intrinsic calibration	Yes	The paper assumes that the cameras are precalibrated
Drone detection	Yes	In the paper it is not explained how the drone is detected, they focus only on the trajectory reconstruction.
Time shift estimation	Yes	We implemented a simple brute-force search algorithm, while the paper uses a solver implemented by another author.
Camera registration	Yes	
Correspondences between unsynchronized cameras	Yes	We didn't perform the linear approximation of the drone motion, but it was still quite effective.
Rolling shutter correction	No	
3D trajectory reconstruction	Yes	
Bundle adjustment: spline trajectory refinement	Almost	We optimized only camera poses, not the 3D splines nor α, β parameters.
Bundle adjustment: least kinetic energy	No	
Bundle adjustment: least force	No	
Offline trajectory reconstruction	Yes	
Online trajectory reconstruction	Yes	
Dataset processed	All but 5	Absence of camera detections

Implementation

02

Notes on the implementation

- The project is entirely implemented in Python, and it heavily relies on the OpenCV library

- The dataset used is the one provided by the reference paper's authors: https://github.com/CenekAlbl/drone-tracking-datasets

- We divided the main problem into a set of subtasks:
  - Camera intrinsic calibration
  - Drone detection
  - Offline trajectory reconstruction
  - Online trajectory reconstruction

Camera intrinsic calibration

02.1

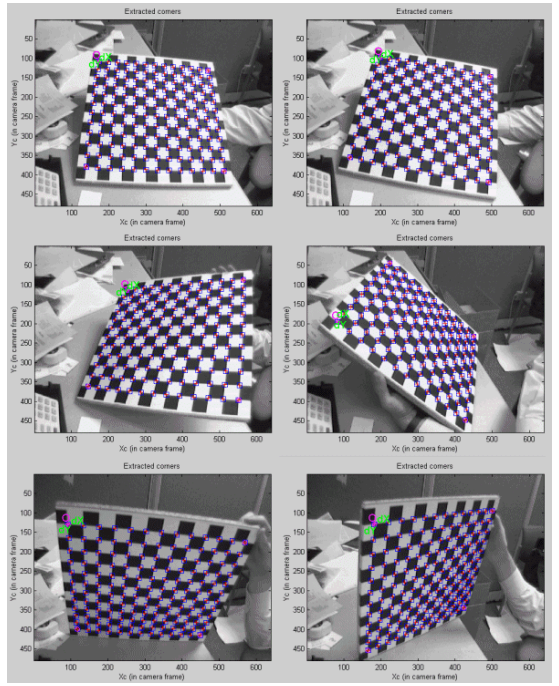
Camera intrinsic calibration

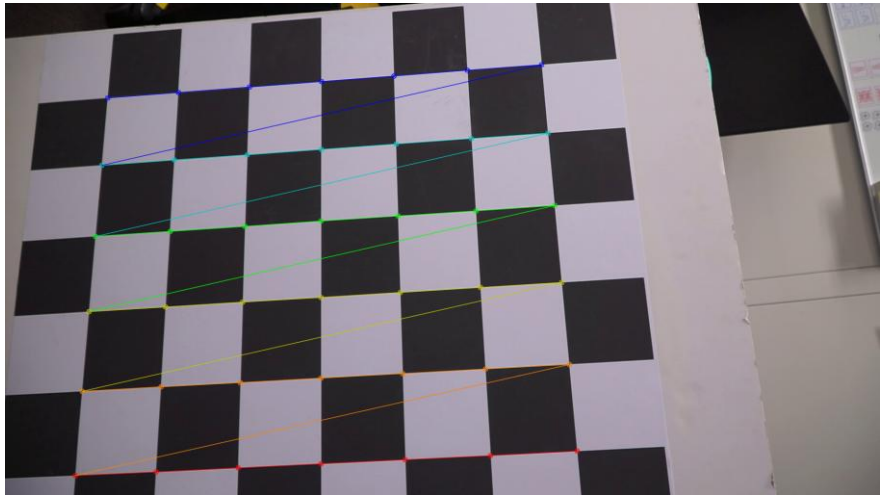
The first task is to find the intrinsic camera matrix and distortion coefficients (following OpenCV's convention) for each camera

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{dst} = (k_1, k_2, p_1, p_2, k_3)$$

OpenCV offers a set of functions that implement Zhang calibration procedure, so it is sufficient to load in memory and pass as an input some pictures of checkerboards for each camera





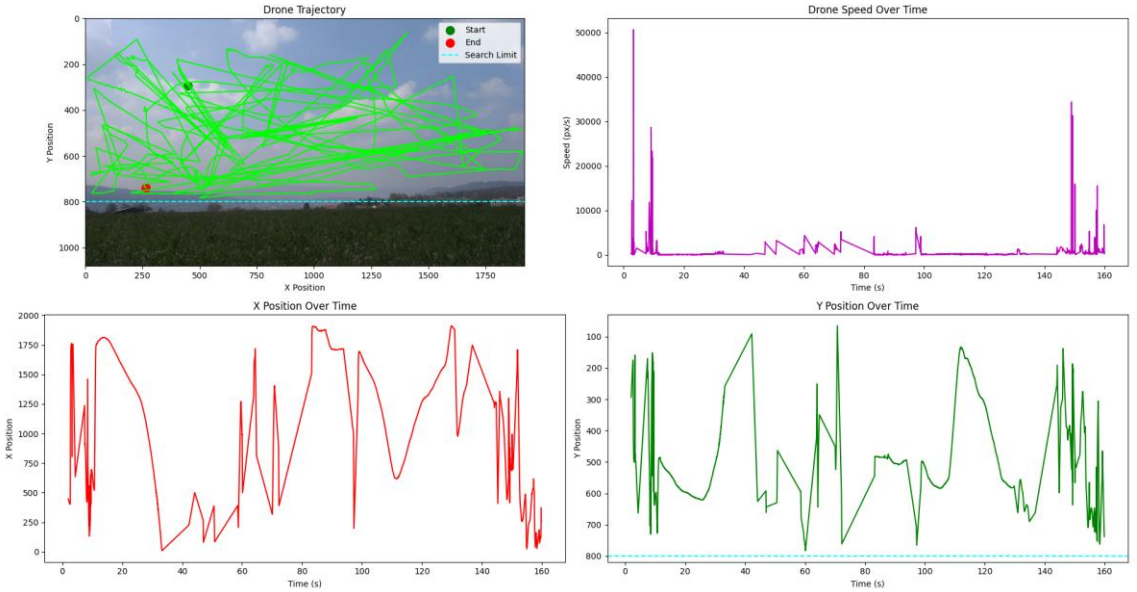
One of the images with detection overlay for camera sony5100

Drone detection

02.2

Background substraction

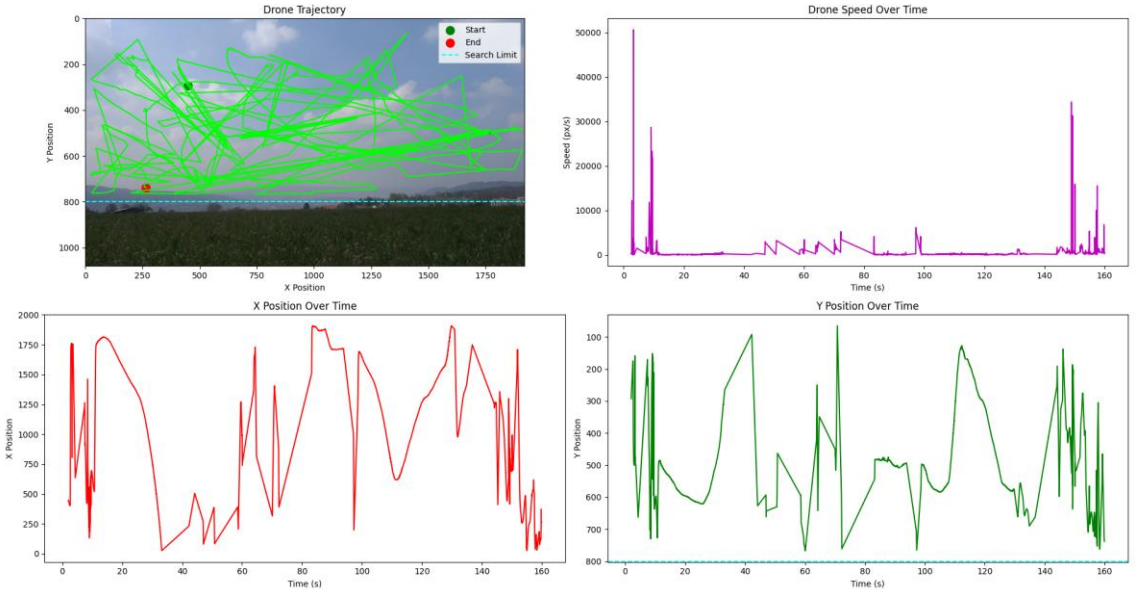
The first classical method to track the drone path onto the image planes is by applying MOG2 Background substraction and filtering by finding the biggest area or by a minimal target tracking (if available)
However, this method became unreliable in the presence of occlusions,intermittent visibility, or when the drone was stationary



Dataset 1, cam 2 algorithm analysis

Background substraction + optical flow

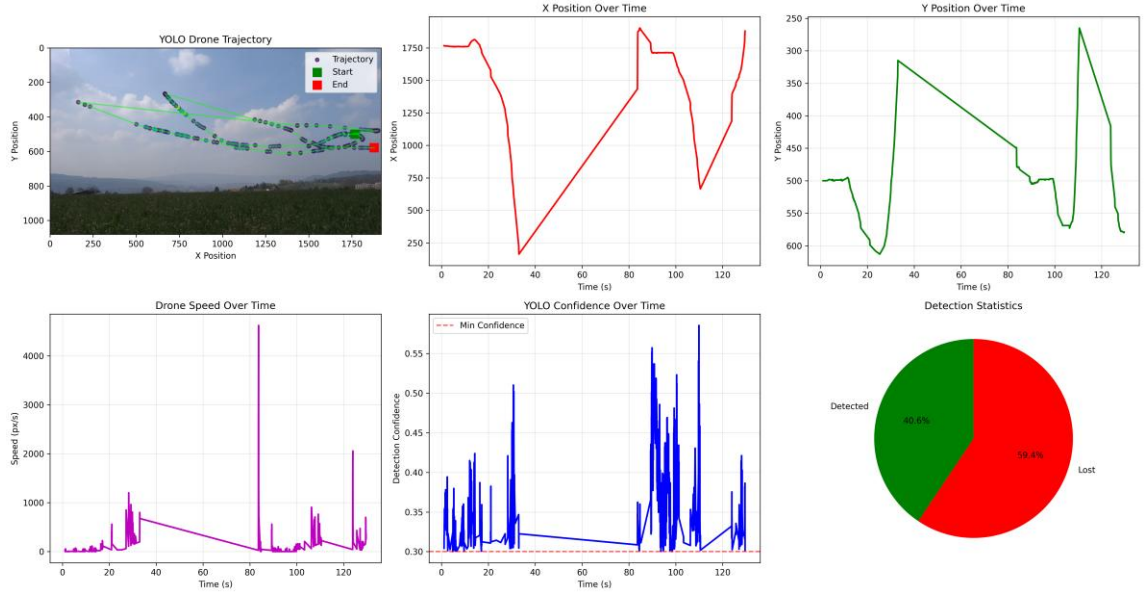
After an initial detection viabackground subtraction, we initialized tracking by identifying good features within the drone's boundingbox. Between consecutive frames, these features were tracked to estimate the drone's displacement, allowing us to update its position even in the absence of fresh detections
It didn't bring a significant improvement in the quality of the detections, as can be seen in the graphs below



Dataset 1, cam 2 algorithm analysis

YOLOv8

We aslo used deep learning-based object detection algorithm. We decided to use YOLOv8, which is a state-of-the-art object detection algorithm, with the pre-trained weights of a small and fast version of YOLOv8, without fine tuning.
It brought a significant upgrade w.r.t. the classical methods on the false positives side (brought to nearly zero), but still failed to detect the drone in certain frames



Dataset 1, cam 2 algorithm analysis

Offline trajectory reconstruction

02.3

Data loading and undistortion

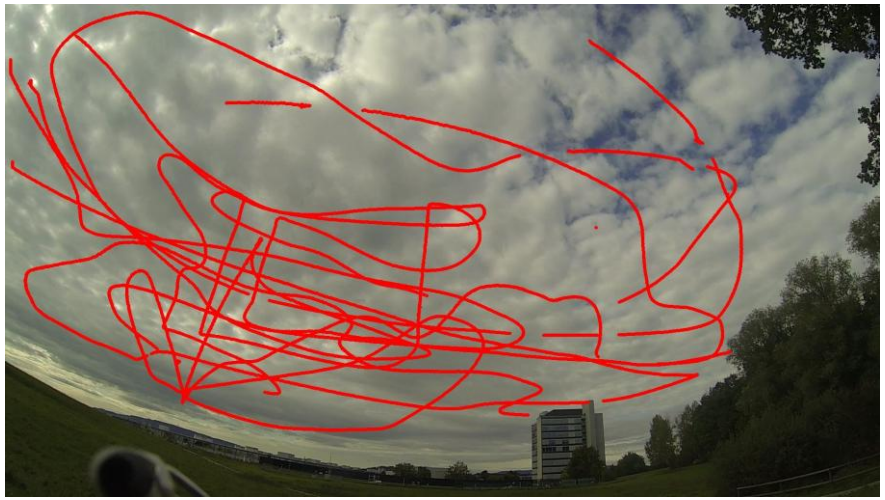
After the detection have been loaded (in our case, directly from the ground truth), the first step is to remove the tangential and radial distortion, that is present in every camera, and more accentuated in the GoPros.

To achieve this goal, we passed the previously computed K, dst to OpenCV's **undistortPoints**, point by point,

For each camera i, every frame j refers to a (global) timestamp

$$t_i^j = \alpha_i j + \beta_i$$

Where β_i is an offset and α_i is a scale factor which together map the frame index j to a global time (or, equivalently, to the frame index of a reference camera k such that $t_j^k = j$)



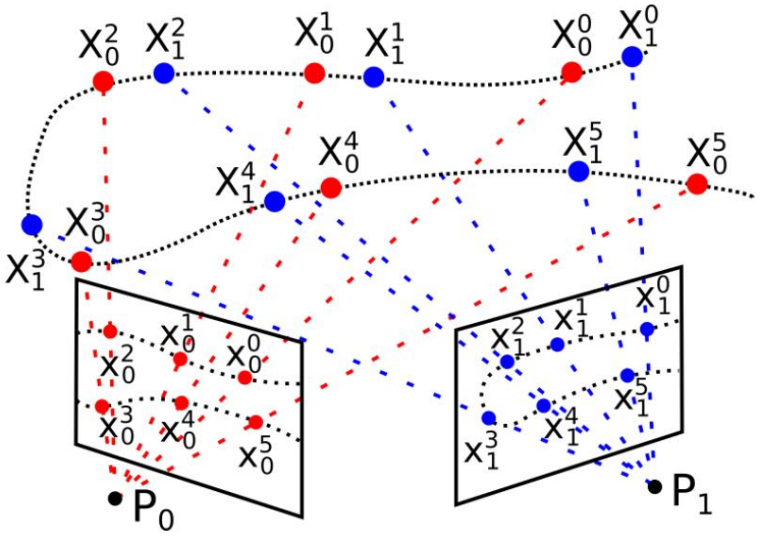
Dataset 4, camera 0 distorted image

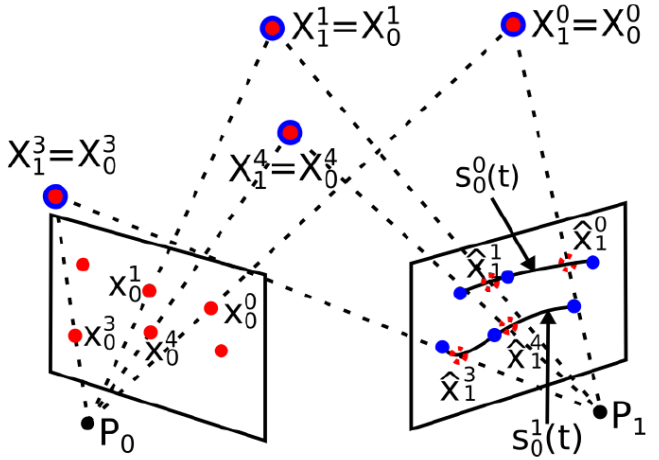
Dataset 4, camera 0 undistorted image

Computation of Fundamental matrix

Correspondences between cameras are obtained exactly like specified in the paper: for each detection in camera i, we searched for a temporally overlapping set of consecutive detections in camera j (after applying the current temporal shift). If such an overlap existed, we fit a spline to the detections in camera j and evaluated it at the timestamp of the detection in camera i.

This provided a set of matched points that can be used to compute F_{ij}





Undistortion

computation of F

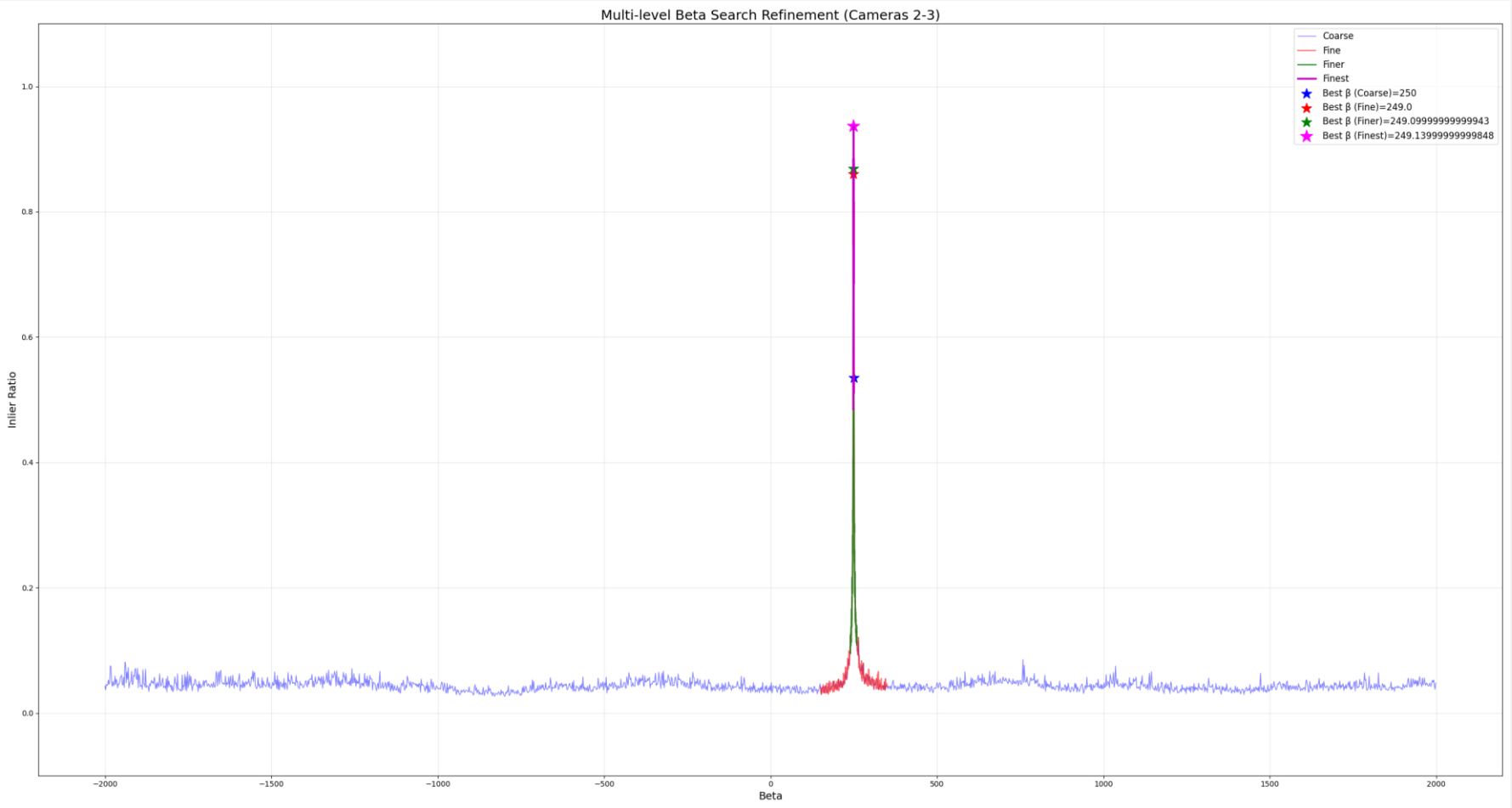
Temporal shift estimation

For each camera pair, we found α_{ij}, β_{ij} in the following way:

$$\alpha_{ij} = \frac{\text{fps}_j}{\text{fps}_i}$$

For beta, we implemented a brute-force search algorithm that shifts the timestamps of camera j and computes F between the detections obtained following the procedure above

The best β_{ij} was chosen based on the number of inliers found in the computation of F



Dataset 3: β search for the camera pair 2-3. It is visible the multi-resolution approach of the search

Undistortion | computation of F | β search

Camera positioning + first points triangulation

Selected the best camera pair (the one with the most inliers), we computed the Essential matrix with the standard formula

$$E_{ij} = K_j^T F_{ij} K_i$$

Then, with OpenCV's **RecoverPose**, we found the relative pose of the two cameras
Since the relative positions could only be determined up to an unknown scale, we set camera j's pose at the origin with an identity rotation. All subsequent positions were then expressed relative to the baseline distance between these two cameras.

subsequently, we could compute the projection matrices of the two cameras

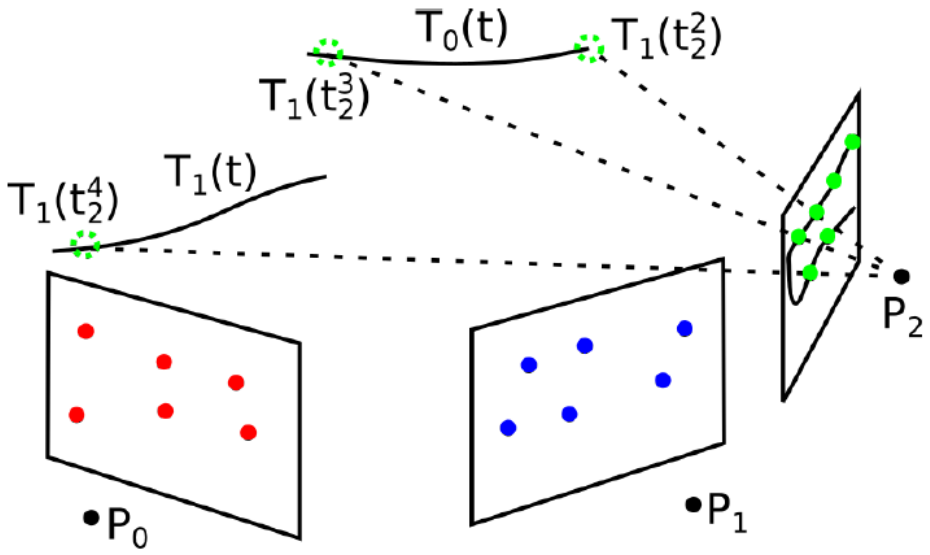
$$P = K \cdot [R|t]$$

and perform the first step of triangulation for the initial set of 2D-2D correspondences, finding an initial set of 3D points

Localization of remaining cameras

Using the same procedure for the 2D-2D correspondences, it was possible to find 2D-3D correspondences between the triangulated points and the detection of all other cameras: we fit splines to the 3D points and evaluated them at the timestamp of the detection of other cameras

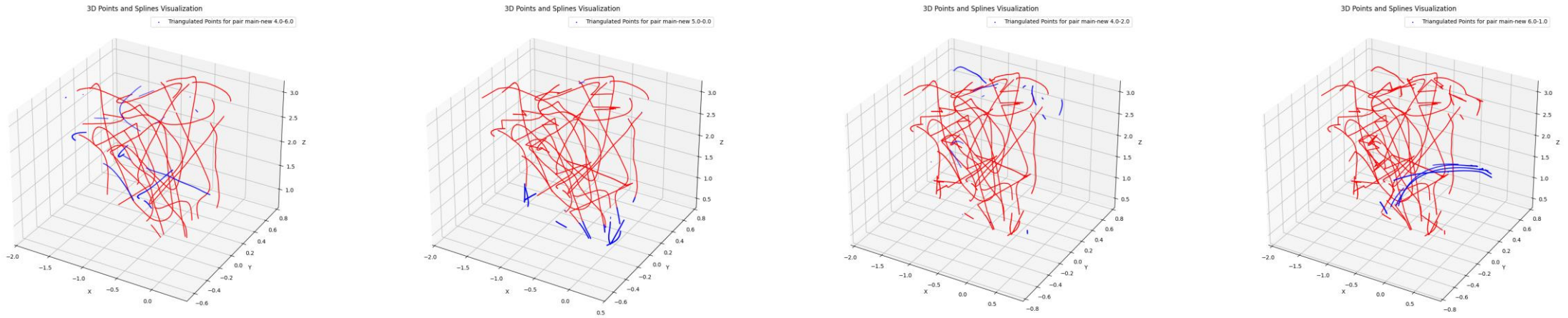
By means of those correspondences, it was possible to perform PnP + RANSAC for robustness in order to localize all other cameras



Undistortion | computation of F | β search | First camera pair | Loc. remaining cameras

3D splines extension

We could add more 3D points to the trajectory as more cameras were localized, by using newly localized cameras to triangulate points outside the temporal span of the already known 3D points
The new points could either extend the already present splines, bridge two of them, or provide more splines not yet registered



Dataset 4: example of the splines extension

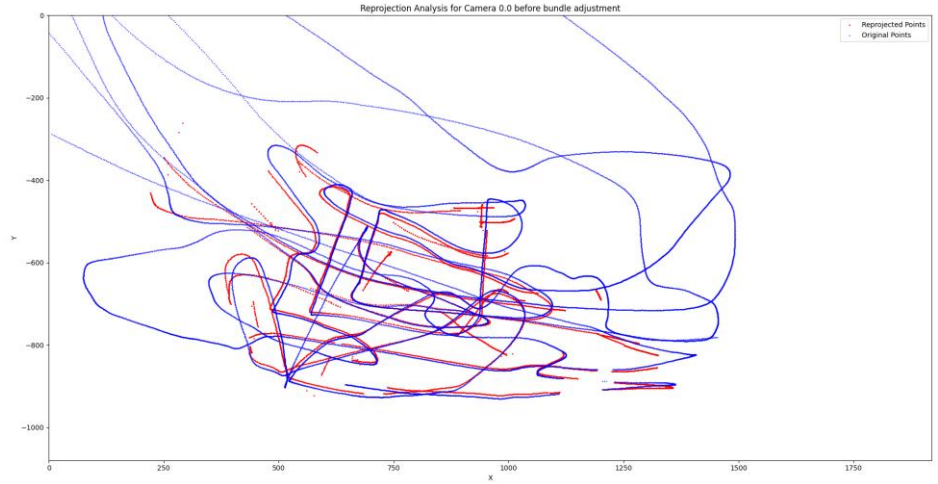
Bundle adjustment

Every time we added more 3D points, we refined the pose of every localized camera i by minimizing the reprojection error, to account for the accumulation of error

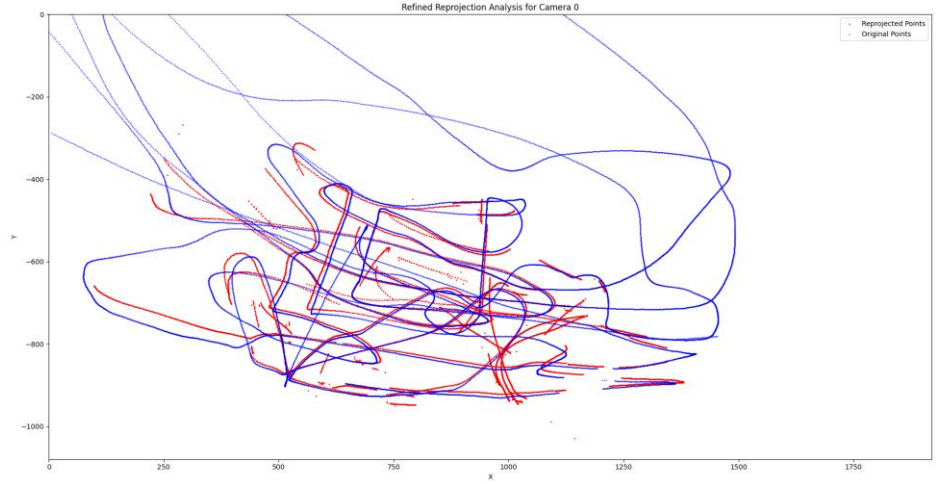
$$\underset{R_i,t_i}{\operatorname{argmin}} \sum_{k=0}^{N_i} \left\| x_i^k - P_i \cdot \tilde{X}(k) \right\|^2, \qquad \forall i \in \mathrm{Loc}$$

Where

- Loc is the set of localized camera
- x_i^k is the detection of camera i at timestamp k
- N_i is the set of timestamps with detection for camera i
- $\tilde{X}(k)$ is the 3D spline evaluated at timestamp k



Dataset 4, camera 0 before bundle adjustment



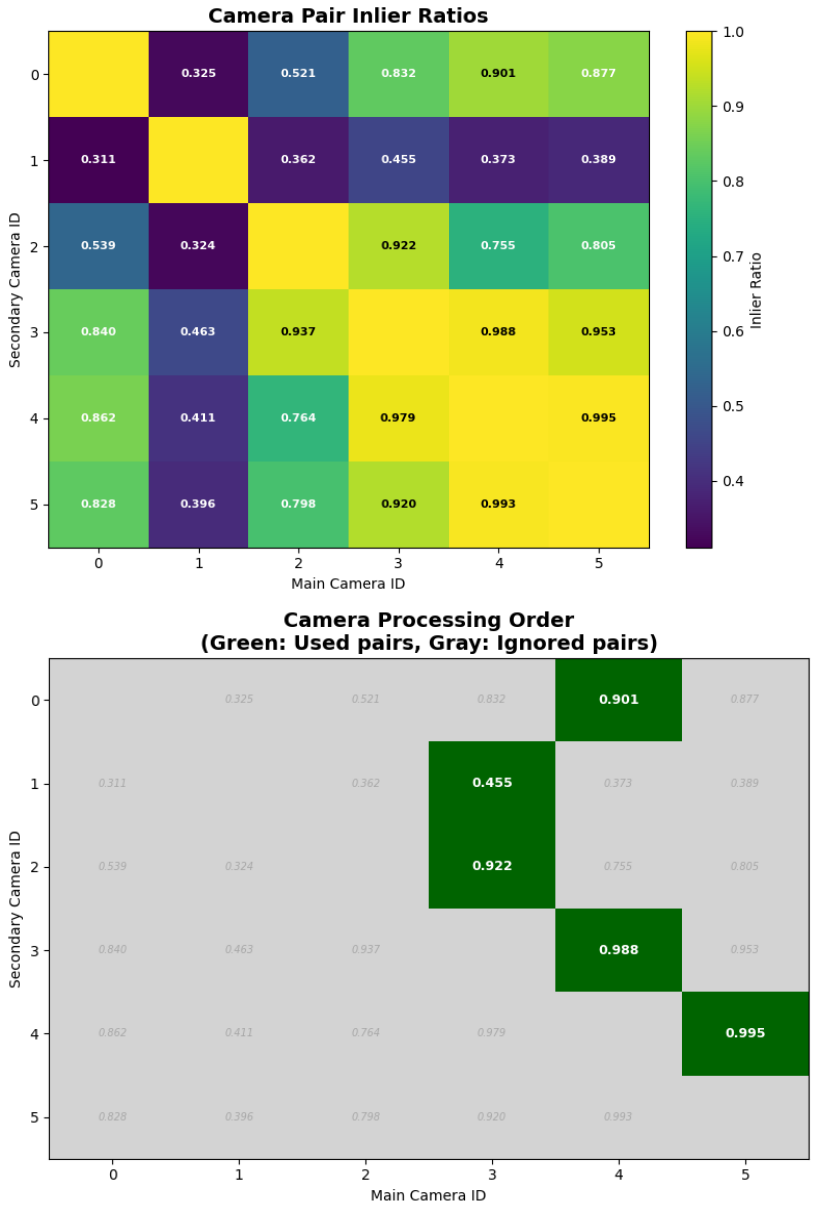
Dataset 4, camera 0 after bundle adjustment

Clarification

The order of camera being processed followed a precise schema:

The first pair of cameras, that set the scale factor, was chosen as the pair with the most inliers count

Each remaining camera was localized by selecting the one with the highest inlier ratio relative to already localized cameras



Dataset 4: selected pairs for the localization

Online trajectory reconstruction

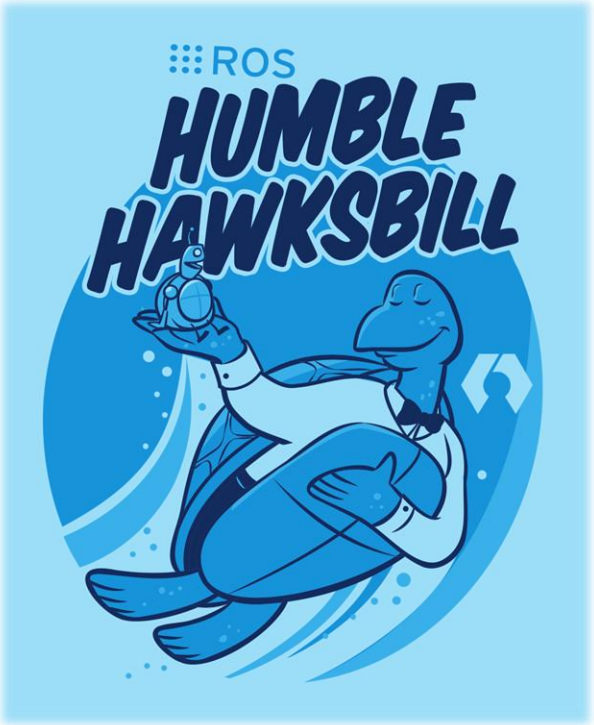
02.4

Online trajectory reconstruction

The cited paper also suggests the implementation of an online trajectory reconstruction algorithm, that is able to reconstruct the drone's trajectory in real-time as it flies.

We implemented this particular approach in a ROS2 node, that is very suitable and efficient for real-time streams of data

We still used Python and OpenCV, so the particular function used in this approach are the same mentioned above



Data preparation

To simulate the real-time stream of data, we followed these steps:

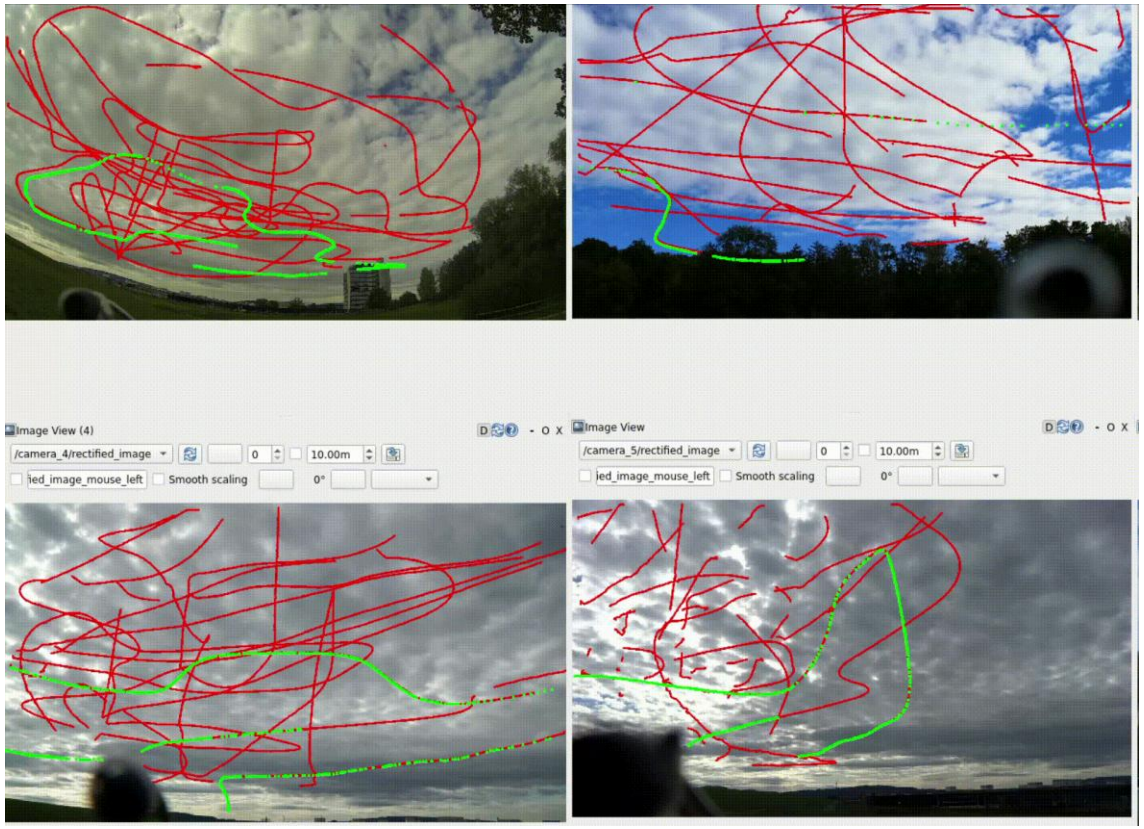
- Load detections for each camera as in the offline mode
- For each detection, compute global timestamp in the way described earlier. We used the values of α, β provided by the dataset to ensure robustness
- Sort all detections by their timestamp
- Publish it on a topic **`/detection`** (then recorded into a ros2 bag)

Detection handling

Since the data was received in real time, it was essential to handle the data stream efficiently

To achieve this, we implemented a subscriber that collected drone detections from each camera and stored them in a buffer. The buffer was organized as an array, where each element corresponded to a specific camera and contained a list of detections grouped by contiguous timestamps.

This structure enabled fast access to the detections from each camera and made it easy to identify timestamp segments that were ready for spline interpolation without the need for repeatedly searching through the entire buffer.

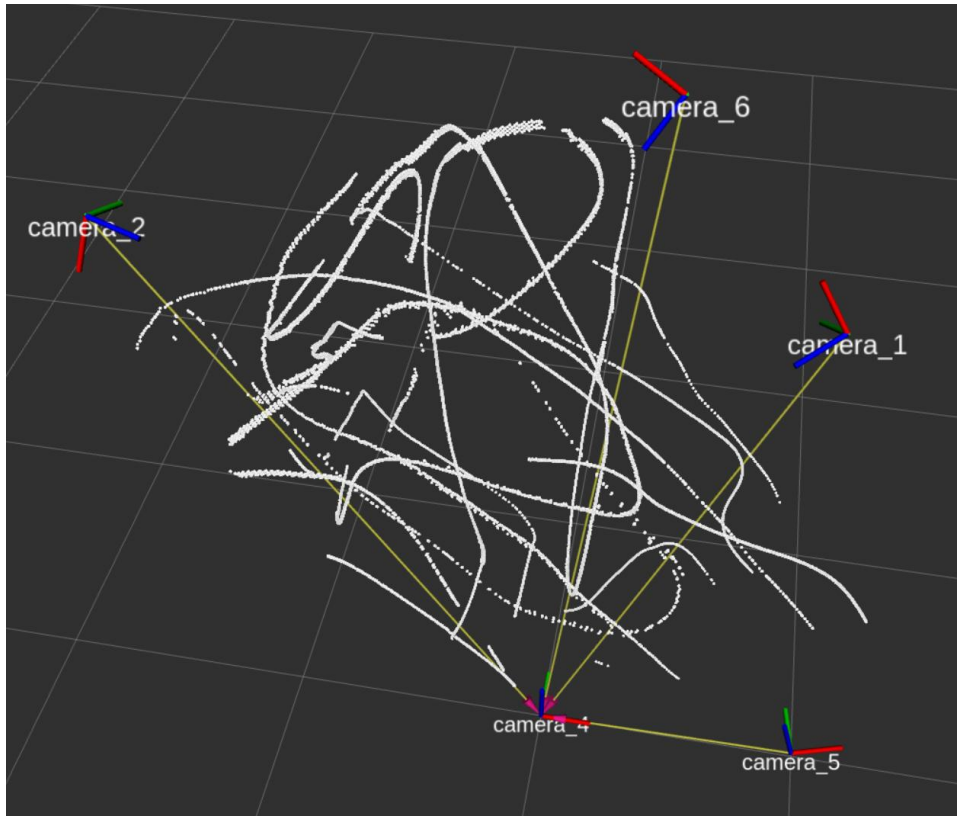


Dataset 4: visualization of live detections on some cameras
Look at the top right (camera0): is it precise? more on that later

Cameras localization

To prevent message loss during trajectory reconstruction, camera localization and reconstruction logic were implemented in a separate thread. Once enough detections were buffered, the algorithm:

- Found correspondences between cameras
- Computed F, and if enough inliers were found, computed E
- Recovered the relative pose between cameras and triangulated 2D-2D correspondences to estimate 3D drone positions
- Generated initial 3D trajectory and splines
- Periodically checked for unregistered cameras and, if sufficient correspondences with the 3D splines existed, computed their pose using the PnP algorithm



3D trajectory reconstructed by the online algorithm (dataset 4). The drone positions are shown as white dots, while the cameras are shown as ROS tf frames

3D splines extension

In parallel with the camera localization, the algorithm would also keep up to date the 3D trajectory of the drone in the following way:

- For each new detection received from a camera pc, the algorithm would interpolate with a 2D spline
- the last 5 detections of the drone in pc (if the detections were not too far apart in time)
- For every other camera i, if their last detection was within the 2D spline time range, the algorithm would evaluate the spline at the timestamp of the detection and generate a 2D-2D correspondence.
- Every correspondence pc – i would be used to triangulate a 3D point, then these points would be averaged and added to the 3D splines of the drone trajectory.

If the new 3D point was sufficiently close to last spline segment in time, the algorithm would extend the spline to include the new point.

This approach allowed the algorithm to keep the 3D trajectory of the drone up to date in real-time, while also registering new cameras as they became available.

Bundle adjustment

As the paper suggested, the algorithm would periodically perform a bundle adjustment to refine the 3D trajectory of the drone and the camera poses. This was done in the same way as in the offline trajectory reconstruction (same code), and it was performed periodically (every 1000 detections) and in a separate thread to avoid blocking the main thread that was receiving the drone detections

Unfortunately, though, this time the result of the bundle adjustment didn’t match the expectations (more on that later)

Results

03

Limitations to the implementation

The implemented pipeline, while functional and sound in theory, presented many limitations that should be mentioned

Limitations to the implementation

1. Drone detection

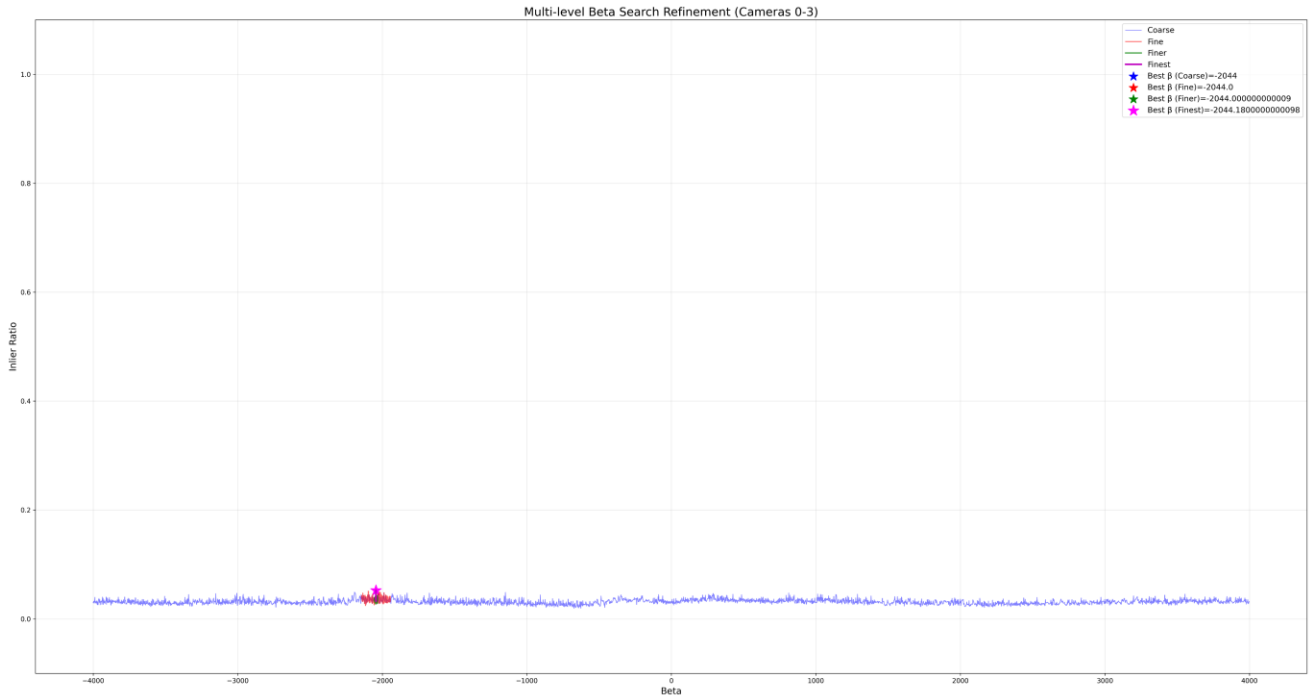
The drone detection algorithm was evaluated on some videos from the dataset. The main limitations were:

- In most of the frames, the algorithm identified the partially visible grass as the drone, because it was moved by the wind. This problem was mitigated by applying a threshold on the area of the detected object, to be set for each video
- The algorithm often mistook the drone for other moving objects in the scene, such as birds. Unfortunately, this problem could not be solved, as the drone was often too small in the frame to be distinguished from other objects. To partially mitigate this issue, we saved also the velocity of the detected object for each frame, and we think that this information could be used to filter out false positives
- The deep learning-based detection algorithm completely removed the false positives, but it was not able to detect the drone in some frames, especially when the drone was too small in the frame

Limitations to the implementation

1. **Drone detection**
2. **β search**

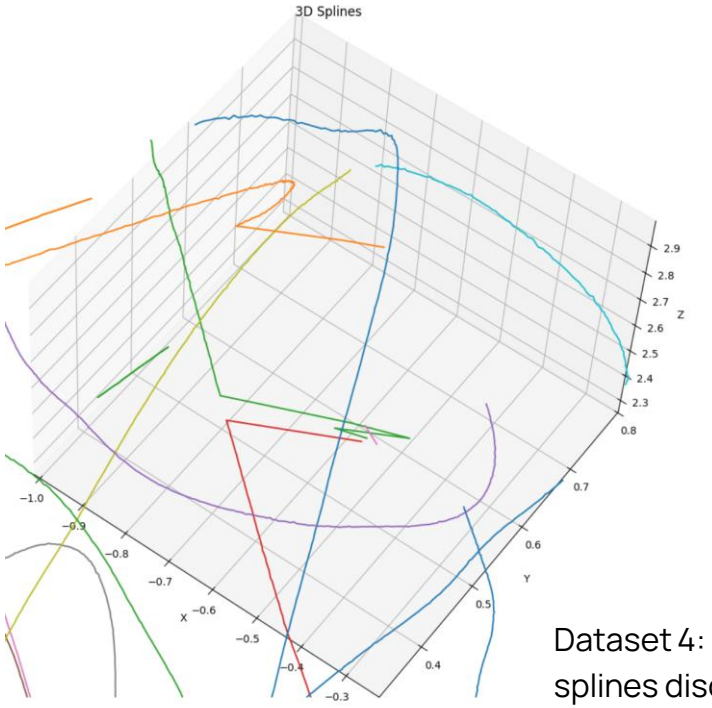
During the β search phase, some of the smartphone cameras were recording at a variable frame rate. In some cases the fluctuation in FPS was very high.
For this reason, the algorithm was not able to find a peak in the number of inliers for these cameras. This is because our implementation relies on the assumption that the frame rate is constant, and therefore, it cannot handle cameras with variable frame rates. We just ignored those cameras



Limitations to the implementation

1. **Drone detection**
2. **β search**
3. **Discontinuous 3D splines**

The process of camera localization was prone to accumulating errors. As a result, when a new camera was registered, the 3D splines were not always continuous, and the reconstructed trajectory could be inaccurate.
This led to temporal discontinuities in the splines, which in turn reduced the effectiveness of the bundle adjustment step (more on that later)



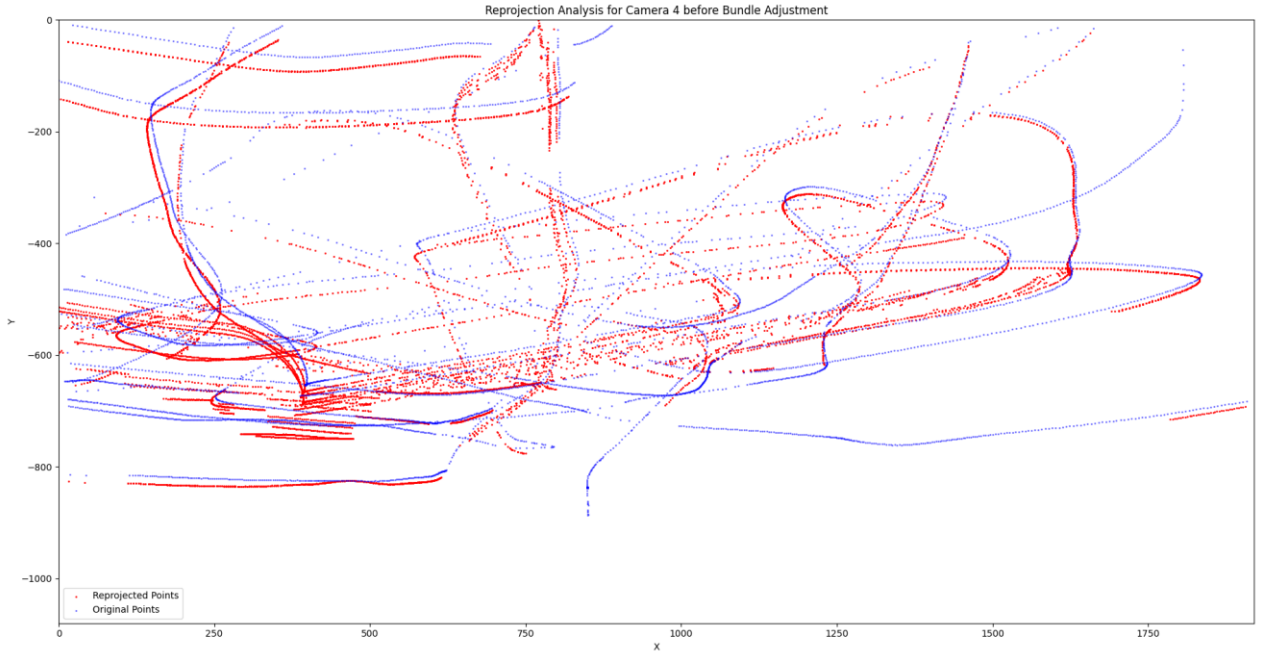
Dataset 4: example of splines discontinuity

This problem triggered a cascade effect: the inaccuracies in the 3D splines led to poor triangulation of new points, which in turn caused further inaccuracies in the camera poses.

Limitations to the implementation

1. **Drone detection**
2. **β search**
3. **Discontinuous 3D splines**

The process of camera localization was prone to accumulating errors. As a result, when a new camera was registered, the 3D splines were not always continuous, and the reconstructed trajectory could be inaccurate.
This led to temporal discontinuities in the splines, which in turn reduced the effectiveness of the bundle adjustment step (more on that later)



This problem triggered a cascade effect: the inaccuracies in the 3D splines led to poor triangulation of new points, which in turn caused further inaccuracies in the camera poses.
In the online approach this led to a very noisy trajectory, but still with reprojections close to the detections

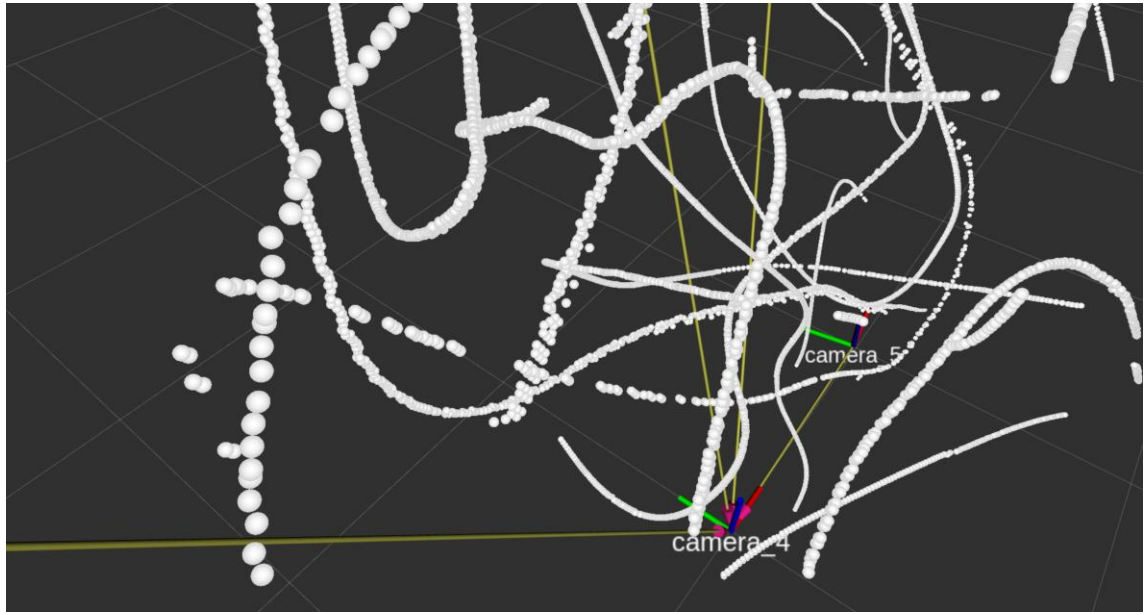
Dataset 3, online approach: it is quite visible the noise of the reprojection of the computed splines and 3D detections, leading also to double splines

Limitations to the implementation

1. **Drone detection**
2. **β search**
3. **Discontinuous 3D splines**

The process of camera localization was prone to accumulating errors. As a result, when a new camera was registered, the 3D splines were not always continuous, and the reconstructed trajectory could be inaccurate.
This led to temporal discontinuities in the splines, which in turn reduced the effectiveness of the bundle adjustment step (more on that later)

This problem triggered a cascade effect: the inaccuracies in the 3D splines led to poor triangulation of new points, which in turn caused further inaccuracies in the camera poses.
In the online approach this led to a very noisy trajectory, but still with reprojections close to the detections



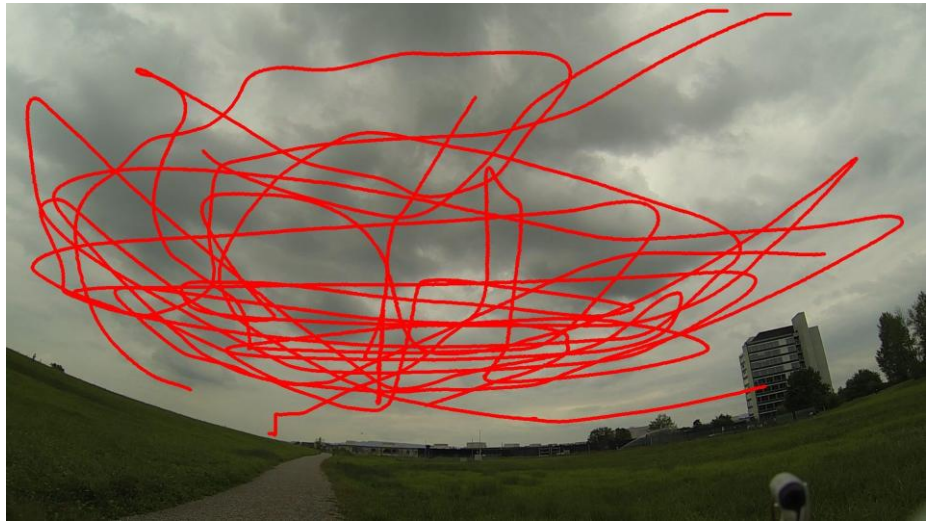
Dataset 4, online approach: noisy 3D points computation visualized

Limitations to the implementation

1. **Drone detection**
2. **β search**
3. **Discontinuous 3D splines**
4. **Fisheye cameras**

A new issue was also introduced by the real-time nature of the implementation: the algorithm was not able to correctly handle the fisheye cameras: we think that this was because the fisheye camera was calibrated as a pinhole camera, and therefore the computation of its distortion coefficients in the intrinsic calibration step was not accurate enough. Again, we ignored those cameras

In future work, we could try to distinct between pinhole and fisheye cameras, and apply a different rectification method for each type of camera.

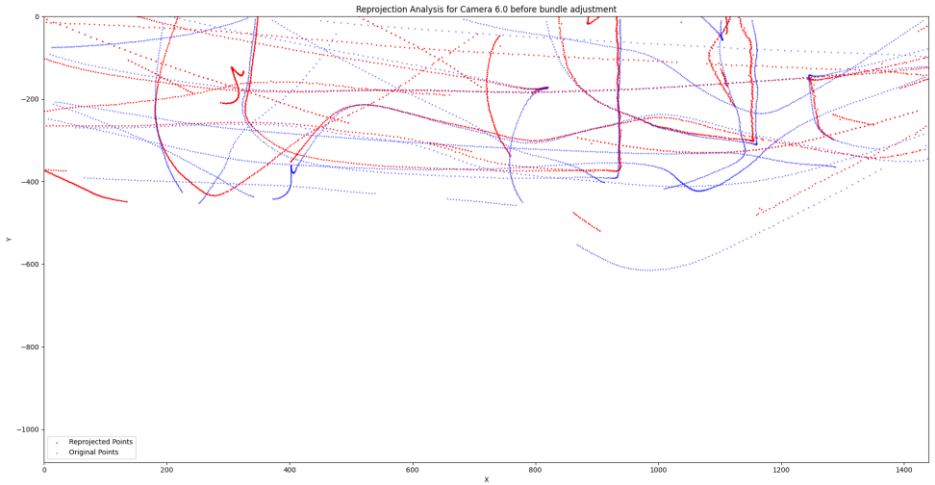


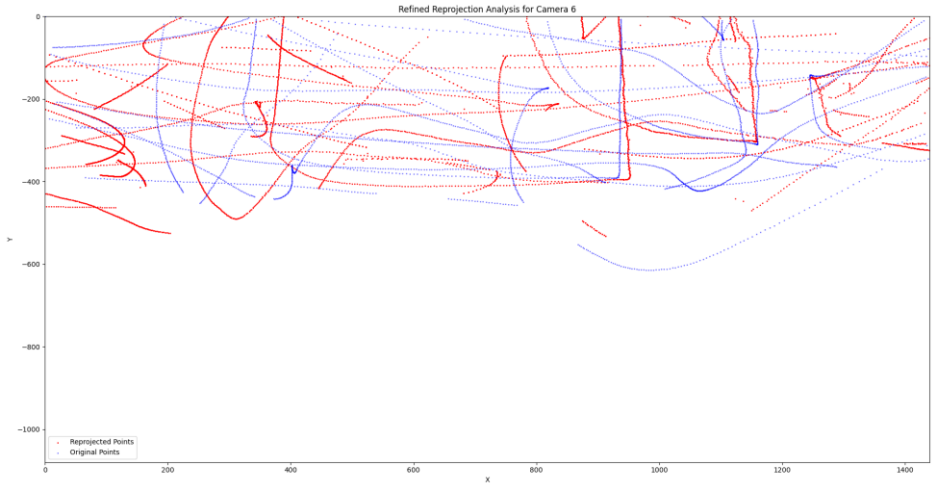
Dataset 3, camera 0 (gopro). It is clearly visible that in the rectified image (on the right), the horizon line is still quite far from a straight line, suggesting that the pinhole calibration for the fisheye cameras isn't reliable

Limitations to the implementation

1. **Drone detection**
2. **β search**
3. **Discontinuous 3D splines**
4. **Fisheye cameras**
5. **Bundle adjustment**

For all the reasons above, the bundle adjustment wasn't always able to mitigate the reprojection error, often worsening the results
In particular, in the online approach we achieved better results without applying bundle adjustment



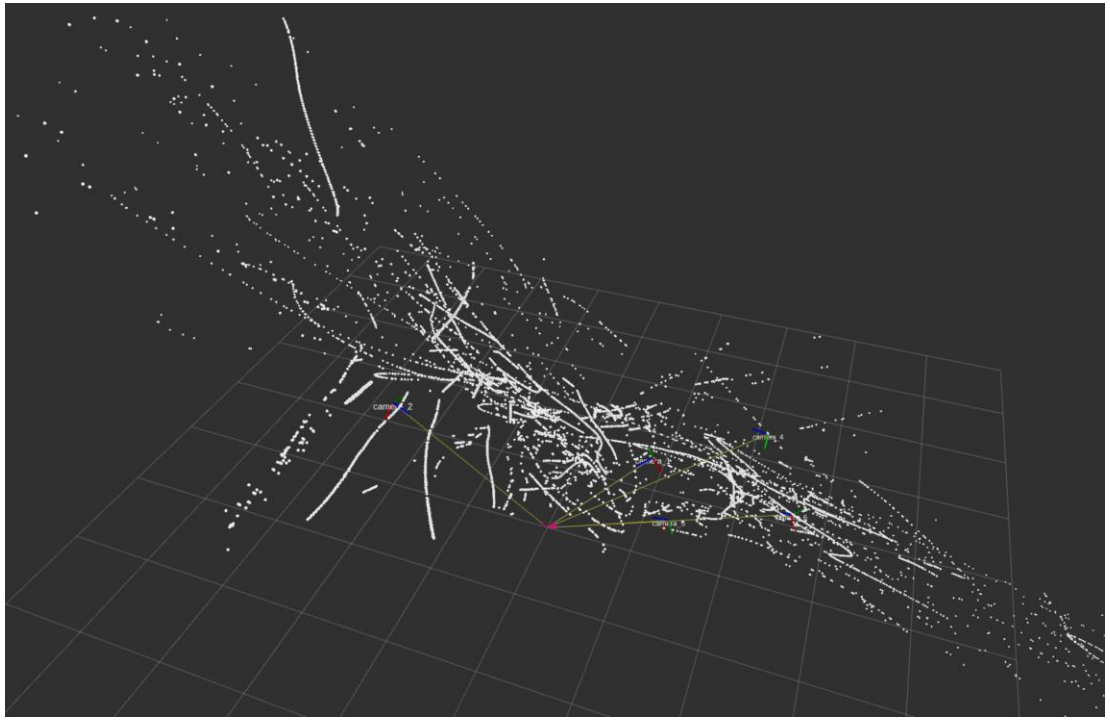


Dataset 4, camera 6: It is visible that the bundle adjustment (image on the right) worsens the results

Limitations to the implementation

1. **Drone detection**
2. **β search**
3. **Discontinuous 3D splines**
4. **Fisheye cameras**
5. **Bundle adjustment**

For all the reasons above, the bundle adjustment wasn't always able to mitigate the reprojection error, often worsening the results
In particular, in the online approach we achieved better results without applying bundle adjustment



Dataset 4, online approach: example of the algorithm failing to produce a trajectory due to bundle adjustment introducing too many errors.

Limitations to the implementation

1. **Drone detection**
2. **β search**
3. **Discontinuous 3D splines**
4. **Fisheye cameras**
5. **Bundle adjustment**
6. **Comparison with g.t.**

The dataset also provided the ground truth trajectory for the drone in some case.
Unfortunately, due to the limitations above mentioned for both methods, the reconstructed trajectory was not accurate enough to be compared with the ground truth, so we had to skip the comparison step.
However, we can still qualitatively assess the performance of the algorithm by visualizing the reconstructed trajectory in 3D space (both plotted in the offline approach and visualized in real time on rviz)

Conclusions

04

Conclusions

Project overview

This project successfully demonstrates the feasibility of reconstructing UAV trajectories using only consumer-grade, unsynchronized cameras. Despite inherent limitations, the pipeline achieves solid results both offline and in real time, highlighting the power of combining computer vision techniques with practical system integration.

Conclusions

Project overview

This project successfully demonstrates the feasibility of reconstructing UAV trajectories using only consumer-grade, unsynchronized cameras. Despite inherent limitations, the pipeline achieves solid results both offline and in real time, highlighting the power of combining computer vision techniques with practical system integration.

Key learnings

- Reliable drone detection is crucial: errors here affect the entire pipeline
- Accurate time sync and smooth splines are key for stable reconstruction
- Bundle adjustment can help or hurt depends on data quality

Conclusions

Project overview

This project successfully demonstrates the feasibility of reconstructing UAV trajectories using only consumer-grade, unsynchronized cameras. Despite inherent limitations, the pipeline achieves solid results both offline and in real time, highlighting the power of combining computer vision techniques with practical system integration.

Key learnings

- Reliable drone detection is crucial: errors here affect the entire pipeline
- Accurate time sync and smooth splines are key for stable reconstruction
- Bundle adjustment can help or hurt depends on data quality

Future directions

- Improve modeling for rolling shutter and fisheye distortion, particularly in real-time pipelines
- Integrate adaptive synchronization methods to better handle variable frame rate cameras
- Enhance the detection stage by incorporating confidence-based filtering and possibly lightweight online learning

Thank you for the attention