Master of Science in Informatics at Grenoble
Master Informatique
Specialization HDWI

Indoor navigation for a drone sharing space with people

Tommy Burnoud

August 29, 2022

Research project performed at GIPSA-LAB

Under the supervision of:
José Ernesto-Gomez
Thierry Fraichard

Defended before a jury composed of:
Dominique Vaufreydaz
Céline Coutrix

August 29 2022

Abstract

A comparison method is proposed for different clustering algorithms working with a stream of events as input. These methods are intended to be used later in a context of social navigation in aerial robotics and to take advantage of the characteristics of these new types of sensors, event cameras. It was shown during this internship that these sensors can indeed make it possible to detect and follow a moving object but lack the maturity to be used alone for more complex tasks such as depth estimation or complex object recognition.

Acknowledgement

I would like to express my sincere gratitude to Jose Ernesto-Gomez for letting me use his event-based camera and his regular feedback on my progress and Thierry Fraichard for his invaluable assistance and comments in reviewing this report as well as for his external point of view on my internship.

Résumé

Une méthode de comparaison est proposée pour differents algorithmes de ségmentations fonctionnant avec comme entrée un flux d'événements. Ces méthodes ont pour but d'être utilisés plus tard dans un contexte de navigation sociale en robotique aérienne et de profiter des caractéristiques de ces nouveaux type de capteurs, les cameras événementielles. Il a était montré durant ce stage que ces capteurs peuvent en effet permettre de détecter et suivre un objet en mouvement mais manquent de maturité pour être utilisés seuls pour des taches plus complexes comme l'estimation de profondeur ou de la reconnaissance d'objets complexe.

Contents

— 1 —
Introduction

1.1 Robotic navigation

In robotics, but also more specifically when dealing with drones, having a good representation of the robot's surrounding environment is crucial for navigation and path planning. Currently, vision-based navigation relies on optical sensors (such as cameras and lidar) to extract the visual features using computer vision algorithms into a usable representation of the environment. This representation is then used to plan a path (if feasible) to the desired destination with constraints specific for each application (ie : no collision, shortest path, ...).

A sub problem in the field of robotic navigation is called "social navigation". This is a relatively new field of research compared to robotic in general that focus on robot navigation in the presence of at least one human. The goal is to navigate among humans without collision and with respect towards the human attention. Indeed, we do not want to surprise or scare humans when it is not necessary and on the contrary, attract their attention when needed. To do so, algorithms trying to predict human motion and attention model are developed.

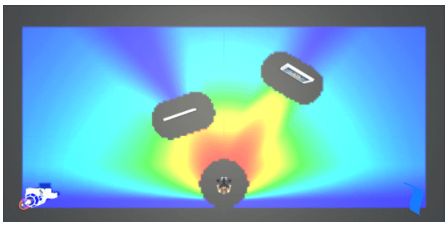

Figure 1.1: Example (from [1]) of attention model used in social navigation

1.2 Event-based camera

An event-based camera is a type of imaging sensor that respond to local changes in brightness. Instead of taking a full picture at a given frequency as standard cameras do, an event camera operates each pixel independently and asynchronously. When a pixel detects a big enough change in the measured intensity, it sends an event composed of the time at which the change has been detected, its coordinates in the image plane and the polarity of the event (+1 for a positive change and -1 for a negative one). Each pixel works by comparing a stored reference

brightness against the current brightness it measures. If the difference is bigger than a threshold, the reference is updated and an event is pushed with its polarity given by the sign of the difference. In the case where the difference is not big enough to reach the threshold value, the pixel does not output anything and keeps it's current reference value.

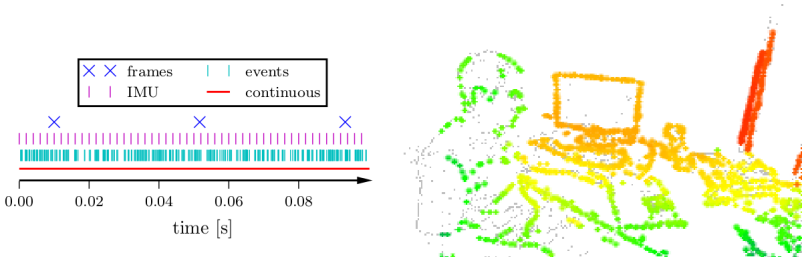

Figure 1.2: Comparison between a regular camera and an event one. The left figure represents the acquisition of a standard camera, an event-based one and an Inertial Measurement Unit (IMU). On the right is an example of an accumulation of events on 15ms with the colour representing the depth of each event.

This way of working is partly inspired by the way the eyes work in most animals. In a biological eye, only some areas send a signal to the brain when a movement is detected. This is a useful trait to evolve since it means that only a fraction of the information seen (and most likely the areas of interest) is sent to the brain for further processing.

Similarly, this type of camera does not push data for areas where no movement has been detected and outputs a lot for areas with changes. This implies that a pre-processing step is already done by the camera and that areas of interest take most of the data being pushed by the camera.

Typical sensor characteristics				
Sensor	equivalent framerate	dynamic range (dB)	spatial resolution	power consumption
Human eye	200-300	30-40	?	10 mW
High-end Digital Single-Lens Reflex (DSLR) camera	44.6	120	2–8	40W
Ultrahigh-speed camera	64	12500	0.3–4	20W
Event-based camera	120	1000000	0.1–0.2	30mW

Table 1.1: Comparison between different kinds of visual sensors (from Wikipedia [2])

As shown in the table 1.1, an event-based camera has many benefits, even compared to a specialised camera, it has a higher framerate than a high-speed camera and a better dynamic range (the ability for cameras to see areas of low luminosity and areas brightly lit at the same time) than a camera designed for this purpose (DSLR cameras). On the other hand, an event-based camera does not output a lot of information (no pixel colour, only the polarity

of the brightness change) which means that already existing computer-vision algorithms must be adapted to work with this kind of information. The low power consumption coupled with a high temporal resolution and high dynamic range makes this type of sensor well-suited for robotic applications.

1.3 Problem statement

The end goal is to study the possibility to control a drone in an indoor environment and in the presence of humans only using a single event-based camera.

Our goal will be first to detect and track any moving object in front of an event-based camera. Once this is done, we can extend the tested methods to detect humans only or to work, mounted on a moving drone.

In order to detect an object moving in front of a fixed event-based camera, we need to implement a vision algorithm that can separate the events generated by a moving object from the other events. Then, with the events coming from a moving object we will produce an estimation of the position of this object in relation with the event-based camera.

1.3.1 Simulation

First, we studied the feasibility of an event-based camera only navigation using a simulation. The goal is to be able to navigate in a simple environment without any collision, whether with obstacles or humans, using only the event-based camera. This will be used as a proof of concept for further development in the real world.

1.3.2 Real world

Once the pipeline is shown to work properly in the simulation, we can test our solution in the real world with an actual event-based camera. The tests will be increasing in complexity as the developed method reaches our goals.

We will start by simply acquiring the data from the event-based camera and finding the best way to interact with it. Next, we will process these events with a fixed camera and try different methods of clustering explorer with the simulation. Then, we will try to do the clustering but with a moving camera to finally mount it on a moving drone.

Finally, we can use the event-based camera and the algorithm to control the drone in a test environment.

This schedule is an estimation of what needs to be achieved to use an event-based camera for social navigation purposes and can vary depending on the results or the time taken to develop the pipeline.

For this problem, some solution are already discussed in the literature such as [3] that combine an event-based camera with a regular one to track a target or in [4] where a drone equipped with two event-based camera in a stereo setup is able to fly to is target location while avoiding

moving objects thrown at it. We will test some of the method discussed 2 to solve our problem.

— 2 —
State-of-the-Art

The goal of this internship can be decomposed into smaller sub-problems, Detection and Tracking of the Human in the field of view, having some kind of representation of the environment (Mapping/SLAM) and finally, the navigation part where the drone, with all the information gather in previous steps can decide its obstacle free trajectory.

2.1 Detection and tracking

In order to take into account human in the motion planning part, the drone must first identify and track him. This is where the detection/tracking part is needed.

A simple solution to this problem is to create and update a list of clusters at each new events as done in [5]. This method is well suited to extract cluster and their speed from a static event-based camera as shown by a paper relying on a similar method, [6]. However, this Mean-shift method might not works for a moving drone since the motion of the camera will produce events linked to static object in the scene as well as dynamic ones. One way to counter that is to perform what is called an ego-motion compensation step by using the Inertial Measurement Unit of the drone/camera.

Another one of the first method proposed to perform tracking using event-based camera is described in [7]. This method relies on kernels (a convolution matrix used to extract certain features) to detect and track object. The simplicity of the method make it really fast at detection but need to have a fixed camera as well as an idea of the shape of the object we're looking for.

Other techniques have been proposed to solve the motion-segmentation problem from a stream of events. [8] In this paper, a new method to segment object by their apparent motion is proposed and only require a static EBC to work. This method require a lot of events (15000 or 4s) in order to finish the initialisation step but perform in real time after that. It also only perform motion segmentation so it is unable to distinguish between a moving object and a human.

Another similar method proposed in [9] use a Gaussian Mixture Model combined with the Maximum a Posteriori method to extract blob-shaped clusters (in the image frame) of the observed moving objects (in the paper, all moving objects are consider as human).

Finally, some methods have been proposed for tracking/motion segmentation from a moving point of view. [4] uses a mix of DBSCAN (a clustering algorithm) and optical flow extraction to detect moving obstacle(s) in the field of view and avoid collision with them. The proposed pipeline only produce approximations of the position and speed of the moving obstacle but is able to detect and avoid them even for high relative speed (up to $10m.s^{-1}$).

A more complex and more time-consistent algorithm is proposed in [3] for tracking of a prey robot by a predator one. First, the DVS camera generate up to 3 Region of Interest (ROI) from the events in a given time-windows, then a Convolutional Neural Network (CNN) generate a confidence map using these ROI and then, a particle filtering step is done to estimate the localisation (in the camera frame so 2D) of the target. Compared to a regular CNN algorithm performed on a RGB image, the authors claim to have a speedup of 70 for a similar accuracy of the tracking. One the other hand, the CNN must be retrained each time we change the appearance of the target.

2.2 SLAM

Simultaneous Localisation And Mapping is a well know problem in the field of robotics consisting of building and updating a map of an unknown or evolving environment while keeping track of the robot position inside it. This is a necessary step for any autonomous robot evolving in a dynamic environment.
While being a quite old problem in robotics,only a few papers tackle this problem using at least an Event-based cameras.

A lot of methods rely on the presence of other sensors to have enough information about the environment before modelling it. For instance, in this paper [10], the authors use an RGB-D camera coupled with the Event-based one to produce a sparse voxel grid but without the camera localisation. Another example can be seen in [11]. This method uses both an event-based and a regular camera, coupled with an Inertial Measurement Unit (IMU). The IMU is used to compensate the events for ego-motion and makes them more reliable when extracting features. Furthermore the RGB camera is also used for feature extraction but is more reliable for long-term tracking. By combining all those inputs, the algorithm prove to be robust (to high drone speed, drastic change of lighting condition, ...). On the other hand, this method only produce an estimation of the drone position over time.

On the other hand, some are trying to achieve SLAM using only event-based camera, using a stereo setup such as in [12] which produce a semi-dense 3D map (a point-cloud map that only keep the edges of objects) or using a single camera in [13] and [14]. The first one only perform a visual odometry using the sensor while the second one also generate, and update, a semi-dense 3D map.

2.3 navigation

Even when the environment is mapped, the drone control problem is not resolved yet. Indeed, the drone still need to find or adjust it's trajectory to avoid potential collision.

We can decompose further this problem : drone navigation with or without humans in the environment.

Having to take into account the human perception adds a lot of constraints to the navigation planner and thus, this problem is mostly tackle en a human-less environment.

The first paper to engage in robot navigation using only an event-based camera is [6]. In this paper the problem is simplified a lot and serve as a proof of concept. The goal of the device is to play the keeper and to block the ball before it enters the goal. The setup is composed of an event-based camera above and looking down onto the goal, the goal-keeper is made with a wooden stick that can be move by a servo motor. Even if the problem is extremely simplified, the paper show the strength of event-based camera extracting in real time the position and velocity of all the balls in the field of view and and adjust the robot arm accordingly.

More recently, a team of researcher have developed a method [15] that can outperform state-of-the-art navigation techniques and compete with experts humans pilots. This methods works using a depth sensor coupled with a stereo RGB setup but the team propose to improve it by adding an event-based camera. The method is mostly based on machine learning and trained in simulation but it is capable of transfer learning into the real world.

On the other hand, some work in the new field of social navigation is done such as in [16] or [17].

— 3 —
Method

3.1 Simulation

3.1.1 Choice of simulator

To begin, a choice concerning which simulator to choose had to be made. This choice ultimately boiled down to two widely used open-source environments for quadcopters simulation: RotorS [18] and Airsim (Aerial Informatics and Robotics Simulation) [19]. Both simulators have an interface with ROS (Robot Operating System), which is useful when going from simulation to the real world since ROS can be run on many of the hardware mounted on quadcopters. RotorS have the advantage of working directly in the built-in simulator of ROS, gazebo but is also quite light in terms of execution. On the other hand, Airsim depends on Unreal Engine for the simulation which is not open-source but is capable of producing much more realistic environments. Also, Airsim already comes with a function to simulate an event-based camera and is based on a video game engine capable of simulating simple human behaviour with relative ease. Finally, it is worth mentioning that I've already worked with Unreal Engine for personal projects, this will decrease the time spent designing the environment and the way humans move.

With all things considered, we chose to use Airsim for our simulations.

3.1.2 Implementation

Firstly, we need to construct an environment for the drone to move into. We used a slightly modified version of the base environment given with the simulator, a point with no collision is added to serve as a goal to reach for the drone and a human actor roaming freely in the main area (see A.1). This basic setup can be used to test how our pipeline performs in a simple world with a single moving person using the collision counter built-in Airsim.

The event camera is simulated from the video stream given by the drone: at each frame, a subtraction between the previous and the last one is performed for each pixel. Then, we check if the absolute change in luminosity is above a given threshold and output an event corresponding to the pixel coordinate, the time the frame was captured and the sign of the luminosity change.

This stream of events is then passed to the main pipeline for processing (see figure 3.1).

First, we use a mean shift algorithm ([20]) to find the areas of high density which will be considered as clusters. Each cluster is then described by its position, size, weight, velocity and a unique ID. Then, using the drone position and rotation gathered from the simulation coupled

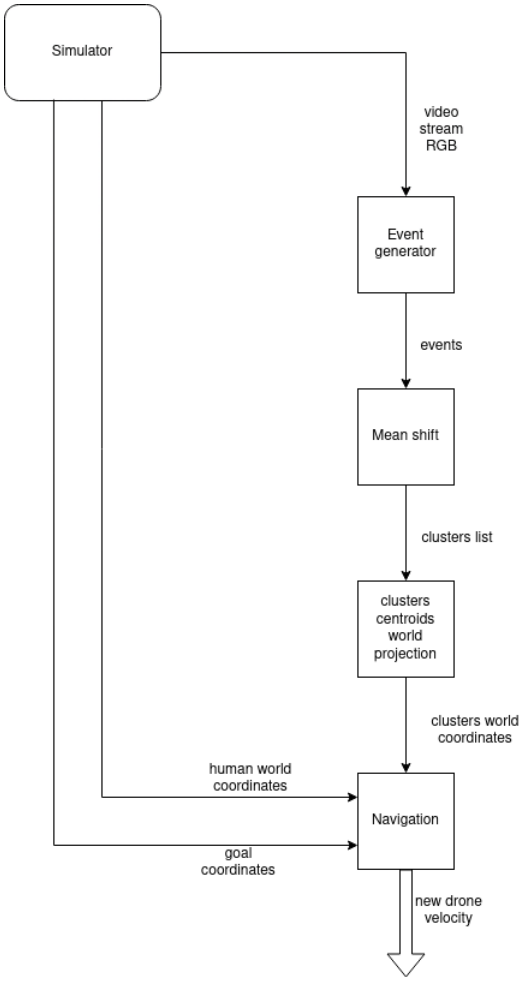

Figure 3.1: Overview of the data pipeline used to process the events and control the drone.

with the intrinsic parameters of the simulated camera we can build the camera projection matrix. The camera matrix is then used to project into camera coordinates (noted $_{ci}$) the cluster's centroid find in the previous step.

$$\begin{bmatrix} X_{ci} \\ Y_{ci} \\ Z_{ci} \end{bmatrix} = K^{-1} \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \tag{3.1}$$

Finally, to convert from camera to world coordinate the depth at location (u_i, v_i) is gathered from the simulation and noted $Depth_i$, as well as the camera transformation matrix (rotation and translation in relation to the origin of the world), noted T_c.

$$\begin{bmatrix} X_{wi} \\ Y_{wi} \\ Z_{wi} \\ 1 \end{bmatrix} = T_c * \begin{bmatrix} X_{ci} \\ Y_{ci} \\ Depth_i \\ 1 \end{bmatrix} \tag{3.2}$$

Finally, from the re-projected clusters world coordinates, human and goal world coordinates gathered from the simulator API we can compute the velocity to apply to the drone. The velocity is computed using attractive and repulsive forces similarly as in [4].

The repulsive force $F_{r,i}$ for obstacle i is computed as follows:

$$F_{r,i} = \begin{cases} \frac{k_r * \gamma}{\eta_0} * (\frac{\eta_0 - \eta_i}{\eta_0})^{\gamma - 1} * \frac{-1}{\eta_i} & if\ 0 \leq \eta_0 \leq \eta_i \\ 0 & otherwise \end{cases} \tag{3.3}$$

Where k_r, η_0, γ are parameters of the function, η_i is the vector from the drone to the obstacle i.

To compute the overall repulsive force, we simply sum repulsive forces for all obstacles in C :

$$F_r = \sum_{i \in C} F_{r,i} \tag{3.4}$$

The attractive force F_a is computed as follows:

$$F_a = \begin{cases} k_a * \frac{e}{\|e\|} * (\frac{\|e\|}{e_0})^{\gamma - 1} & if \parallel e \parallel < e_0 \\ \frac{e}{\|e\|} & otherwise \end{cases} \tag{3.5}$$

Where k_a, γ, e_0 are parameters of the function and e is the vector between the drone and the goal positions.

Finally, the overall force F applied to the drone is the sum of the attractive and repulsive forces :

$$F = F_a + F_r \tag{3.6}$$

Even though this pipeline is able to successfully navigate the drone in an environment with a human moving in it, it starts to be heavy performance-wise (my computer had to run a simulation and a graphical rendering of said simulation on top of the actual processing pipeline) and lack from a real-life experiment. This is mostly visible in the presence of artefacts in the generated events as well as some tremors and unexpected pauses in the drone movements.

3.2 Event-based Camera

During this internship, we used the DVXplorer Lite from iniVation for our experiments. This event camera has a resolution of 320x240, a temporal resolution of 200µs, a dynamic range of 120dB and can output up to 100 million events per second through a USB 3 cable.

The manufacturer provides a library (dv-processing, [21]) to interact with the camera in Python or C++ as well as a GUI used to perform simple tasks. This GUI was used to estimate the intrinsic matrix of our camera (noted K) which is used in the equation 3.1. The dv-processing library is used to interact with the camera and get the events at a fixed interval, otherwise, most of the pipeline uses openCV ([22]) or custom build functions to process the events.

The first thing done in this pipeline is to accumulate 15ms of events before sending them to the next step. this time period is an empirically found compromise between real-time computation and accuracy. If we use a smaller time value, the program becomes too heavy to be

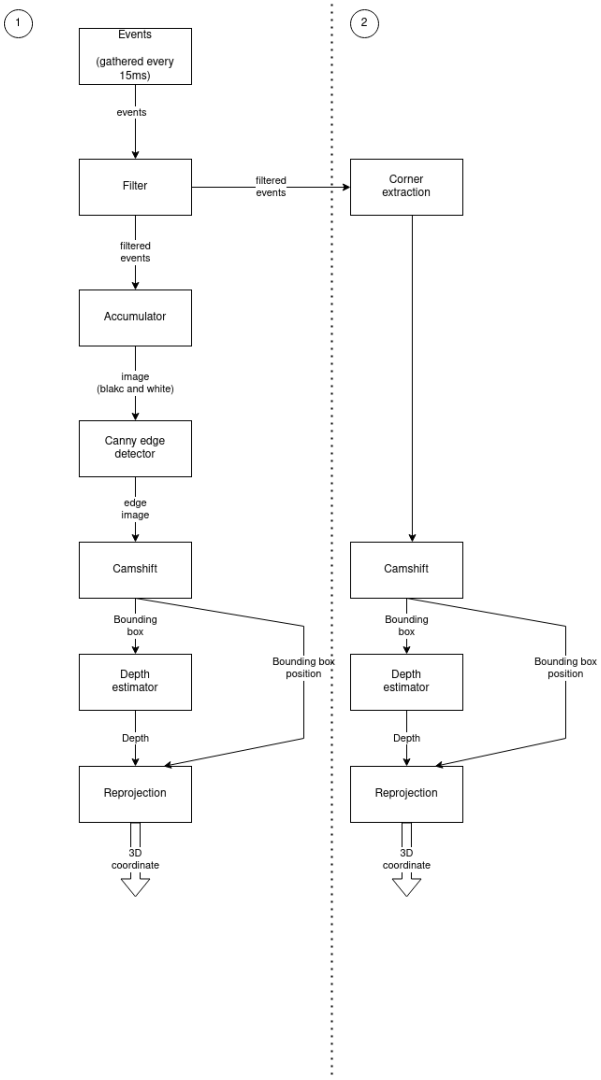

Figure 3.2: Event processing pipeline

computed in that small amount of time but on the contrary, when waiting too much time to perform the computation (when the period reaches 75ms or more), we might lose some reaction time when used on a real-time system such as a drone.

Then, these events are filtered by some function provided by the manufacturer library to remove some of the noise produced by the sensors. This is done to avoid considering noise and events from background activity as relevant and focusing our segmentation algorithm on events produced by moving objects or humans relatively close to the camera.

From the filtered events, this is where the different methods tested diverge (see 3.2).

3.2.1 Contour method

The contour method (noted 1 in the diagram) converts the events into an accumulated frame before doing any further processing. This frame is produced by adding a fixed contribution to a pixel if a corresponding event has been received and applying a decay function to the whole frame (see A.2 for example). This accumulated frame is then processed by a Canny

edge detector (implemented in openCV). This is done to further filter the incoming events and separate noise from the signal.

Next, the edges image is passed to a Camshift (Continuously Adaptive Mean Shift, implementation from openCV) to perform the segmentation and extract a bounding box of the detected cluster. The advantage of Camshift over Mean shift is that it can adapt its bounding box in size and orientation to better match the cluster's shape. This is done by finding the best size and orientation once the underlying mean shift has reached its convergence.

Then, from the bounding box, an a priori knowledge of the real size of the object and the camera's intrinsic matrix, we can produce an estimation of the depth of the detected object with the following computation :

$$Depth = \frac{F * S_r}{fx * S_p} \tag{3.7}$$

Where F is the Focal length of the camera, S_r is the real size of the observed object, fx is the size of the individual pixel sensors and S_p is the size in pixel of the cluster.

Finally, from the depth and the centre of the bounding box, we can estimate a 3D position relative to the camera as done in equations 3.1 and 3.2 where u_i and v_i are the coordinates of the centre of the bounding box.

3.2.2 Feature method

The feature method starts as well from the filtered events but processed them with a corner extraction algorithm (implemented in openCV). This step further decrease the number of observed feature by ignoring events in a sparse area and focusing on dense regions of the image (as shown in 3.3).

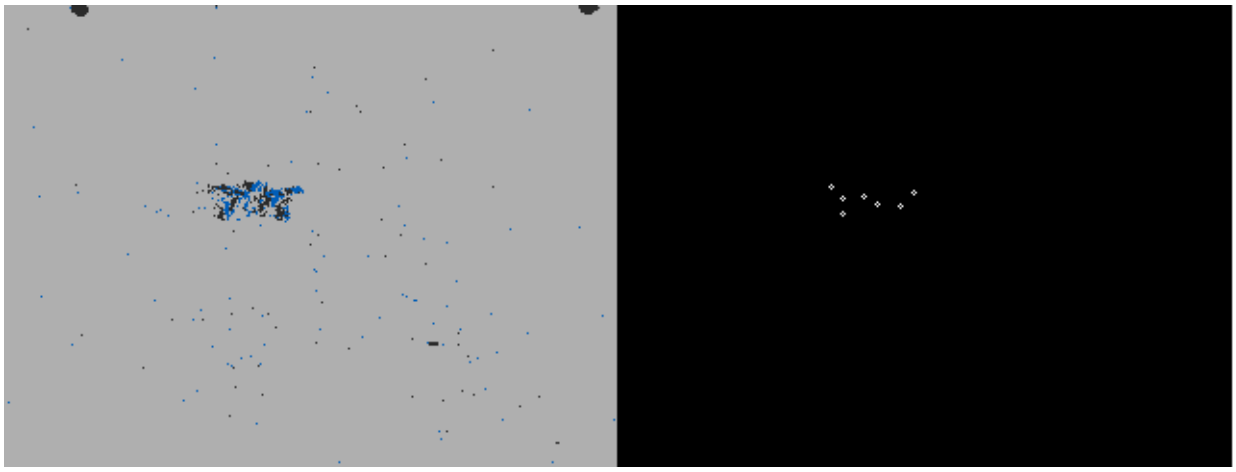

Figure 3.3: Events to corner conversion. This image also shows the clustering step with the colour of the corners.

The Camshift algorithm and the rest of the pipeline is then done as for the first method to produce an estimate of the 3D position of the object.

— 4 —
Results

4.1 Experiment

4.1.1 Description

To test our method, an experiment was designed. In this experiment, we use the "Moca" (Motion Capture) room from the Plateforme robotique aerienne [23]. This room is equipped with a VICON multi-cameras motion capture system able to track and reconstruct in real time the position and orientation of any objects equipped with infrared reflectors.

The event-based camera is placed at a fixed location, -3.58m along the x-axis and $0,875$m along the z-axis away from the origin of the room (see A.5 and A.6), situated in the middle of the space used for motion capture (see A.6).

Figure 4.1: Image of the Moca room used for the experiment.

A drone (see A.4) is then used in front of the camera, moved by manual piloting. The camera is then used to estimate the position of the drone relative to the camera and compared to the data given by the tracking system of the room.

During the experiment, it was discovered that the event-based camera used is sensitive to near infra-red wavelength and thus, detects the infra-red camera's led ring used for tracking. This led ring blinks at around 5 to 20ms and can't be turned on continuously. This caused some "bright spots" of blinking infrared light in the field of view of our camera that can only be decrease in intensity at the cost of the precision of the tracking system of the room. A balance has been found between the accuracy of tracking and the level of noise but the latter was still higher than what is seen in a standard setup without tracking. This causes the pipeline to be less

accurate than it should be but it is necessary in order to have results to evaluate our algorithms. Also, the event-based camera field of view includes some of the infra-red camera's led ring as well as the reflection of one of them(visible on the top of the image for the led ring and the middle right for the reflection on the left image in A.2). This was filtered using a mask on those specific regions for this experiment.

4.1.2 metrics

To compare our results, we will use the following metrics :

1. Position error along the three axes.
2. Computing time for the whole pipeline.
3. tracking accuracy: percentage of time the object is correctly detected

The first metric will provide a way to measure how well the pipeline is able to extrapolate the 3D position of the seen object and decompose the error along the three axes of the re-projection. The second one is used to know how much time is spent by the algorithm to produce the result.

This is particularly relevant since the pipeline is destined to be used in a robot in the end and thus, needs to be light enough for an embedded system.

And finally, the tracking accuracy is computed by dividing the number of frame where the drone is accurately detected by the method by the total number of frame of the experiment. This metric is used to measure the performance during the experiment as well as the consistency of the tested method.

4.2 results

4.2.1 Position error

We can see that both methods behave in a similar way regarding the position error even though the feature method is less accurate on the re-projection. Both methods produce their best estimate on the z-axis with an error under 25cm for the contour method and follow the same trend as the ground truth when the drone is correctly detected. On the other hand, when the tracking method loses the drone (particularly visible for both method on the z-axis around 60s) the error become an order of magnitude worse and is visible in both tested methods. This lose of tracking of the target happens when the relative speed of the drone with respect of the event-based camera is too low (for instance around 13s, 30s and 47s) or when the drone leave the field of view of the event-based camera (around 60s). When the drone is no longer detected, the computer vision algorithm will pick a group of events caused by noise (the direct or indirect light from the room tracking system, a moving shadow on the ground or noise from the sensors themselves) and produce an estimation for the drone position from the wrong cluster.

Finally, the depth estimation part produce unusable results in the context of indoor navigation and social navigation. The result barely move from the origin of this axis and when it does, it is mostly unrelated with the real movement of the drone.

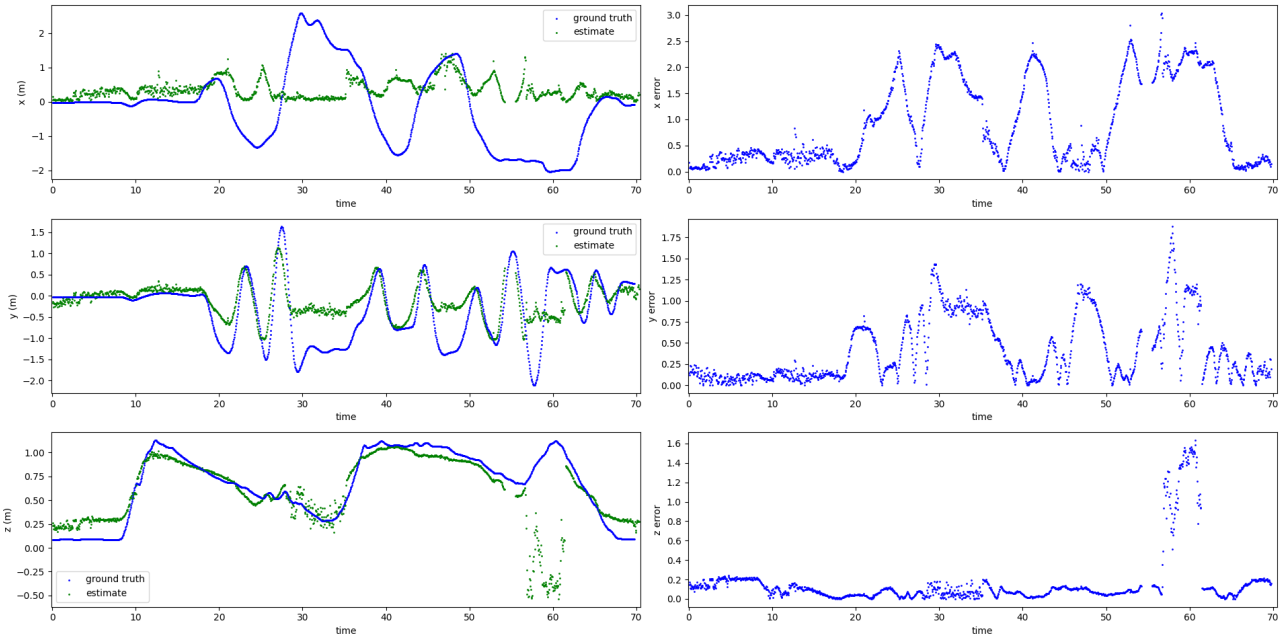

Figure 4.2: Comparison of predicted positions against ground truth for the contour method. Left is the ground truth position (in blue) and the estimation (in green) for each axis and right is the error (ie the absolute value of the difference between ground truth and estimation).

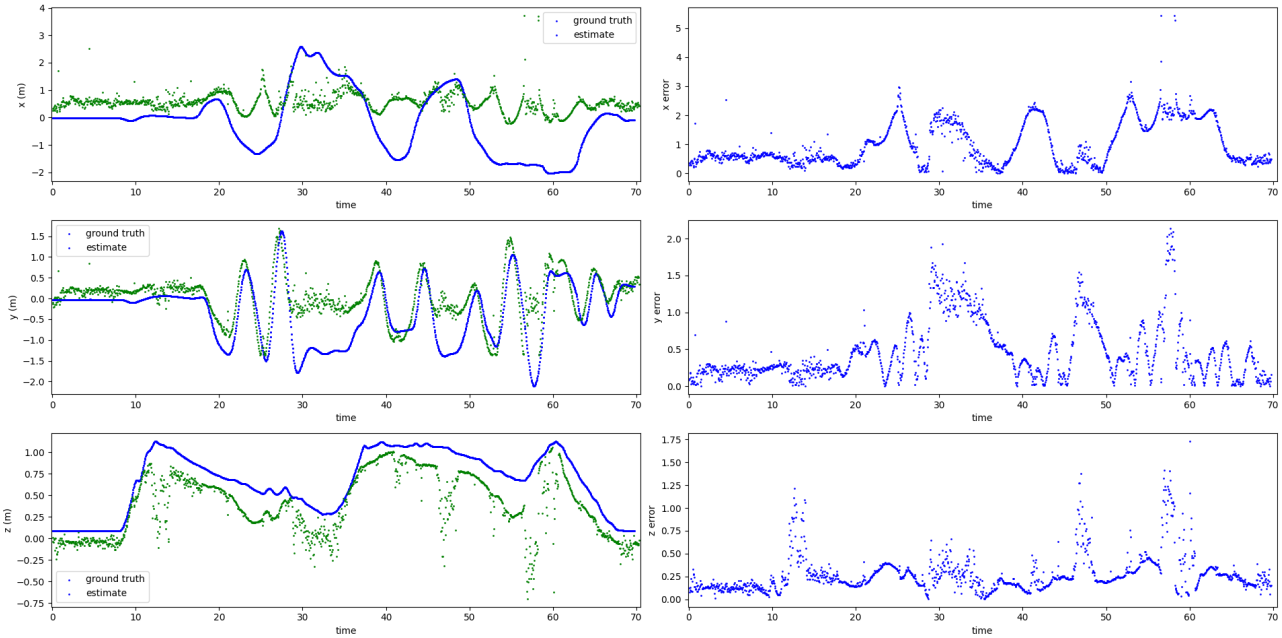

Figure 4.3: Comparison of predicted positions against ground truth for the feature method. Left is the ground truth position (in blue) and the estimation (in green) for each axis and right is the error (ie the absolute value of the difference between ground truth and estimation).

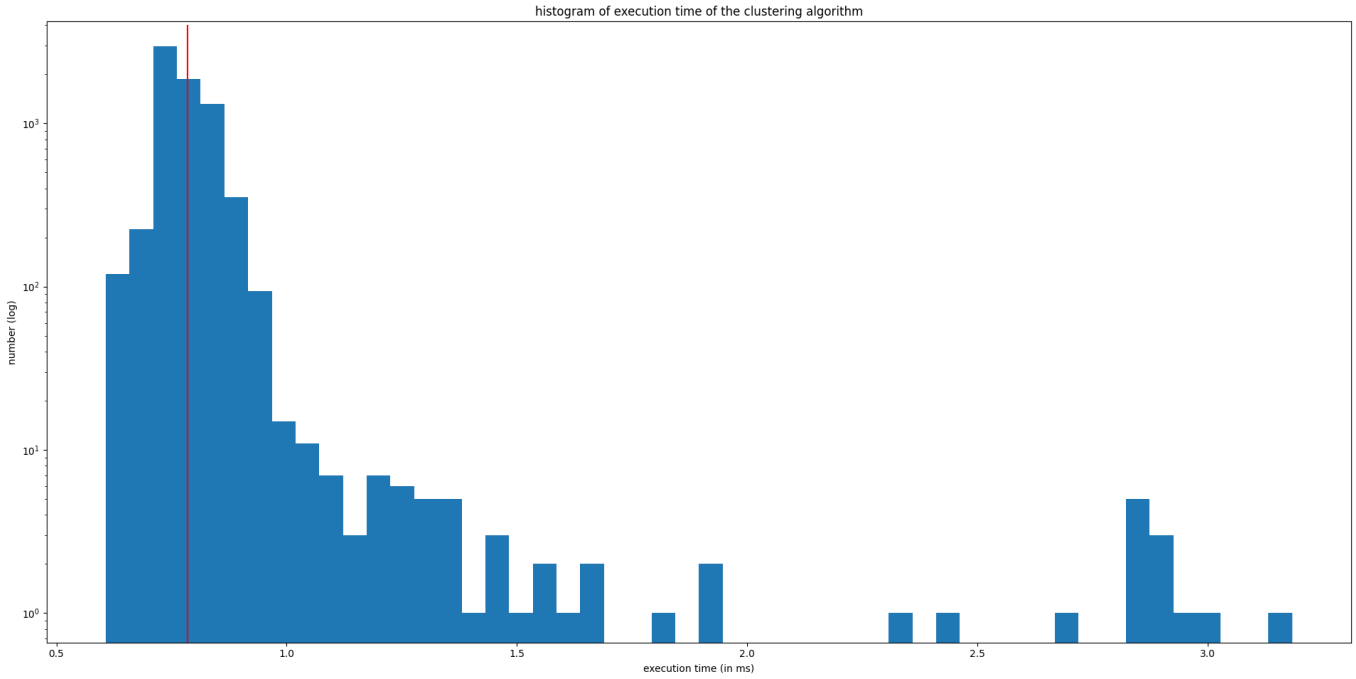

Figure 4.4: Histogram of the execution time during the experiment for the contour method. Average execution time in red.

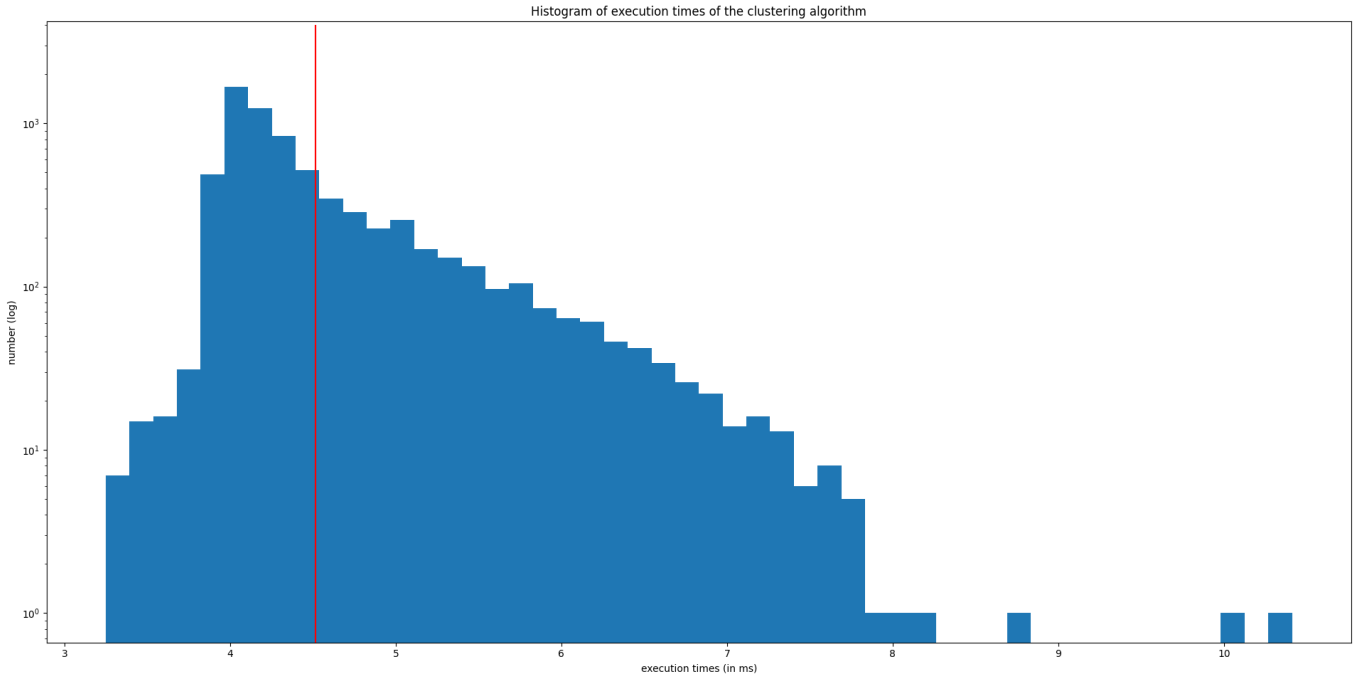

Figure 4.5: Histogram of the execution time during the experiment for the feature method. Average execution time in red.

4.2.2 Execution time

With both methods working with events accumulated over the course of 15ms, the two are on average way under the requirement for running in real time.

The contour method is by far the fastest of the two with an average execution time of 0.78ms and very few outlier after 2ms (see 4.4).

The feature method is slower with an average of 4.51ms of execution time and run in general more slowly : no execution time under 3ms, the majority of them are between 4 and 7ms with some outlier at 10ms (see 4.5).

4.2.3 tracking accuracy

tracking accuracy :

contour method	feature method
75.5546%	65.1166%

Table 4.1: Tracking accuracy for each method.

This table was made by manually counting for each method the number of frame where the produced bounding box is correctly placed over the drone or not and divided by the total number of frames in the experiment.

As already said in 4.2.1, the feature method is less consistent than the contour one and it is visible with this metric too. This is also consistent with the level of noise visible on the absolute error in 4.2 and 4.3, especially on the z-axis. the feature method having more bad position estimations is mostly due to the less accurate tracking ability of the method.

4.3 Limits

Depth estimation is very inaccurate with only an event-based camera thus an application relying only on a single one to estimate the depth for further computation cannot be relied on.

Moreover, the small resolution of the used sensor (320x240) compared with a standard camera was a problem for a reliable detection at different depths, a compromise had to be found between the field of view and the angular resolution. The larger the field of view, the lowest will be the apparent size of the object on the sensor. Because we only did one experiment, we could only test a single setup where the camera was focusing mainly on near objects and thus, perform poorly when the drone was flying more than a meter away from its starting point (see results around 30s in 4.2).

Also, the current method is not able to detect when the target is lost and instead, focuses on noise until the object re-appears. As shown by the tracking accuracy, most of the time the drone is not correctly tracked, it is because it has left the field of view of the camera or it is moving too slowly to be detected by the camera.

4.4 Pros

The designed pipelines are able to run in real-time in Python on a computer (see 4.4 and 4.5). This is a good sign for embedded applications running on slower hardware: with some optimisation and a translation to a faster language such as C, we might be able to benefit from the small temporal resolution of such cameras for fast-reacting robotic applications. As said in [4], the time window to accumulate the event is a compromise between sensitivity and real-time performance. A too small time window can render the algorithm useless because of the small amount of information stored in the events is not enough for a reliable detection. On the other hand, after a certain point in the increase of the window, the number of events to process becomes too costly for the algorithm to run in real-time while not providing much more useful information but also due to the latency between the moment an event is generated and the moment it is processed. A compromise has been experimentally found around 10 to 20ms, providing sufficient information for tracking while still being fast enough to benefit from the capacities of the camera.

The experiment also demonstrates the capability of the pipeline to track a moving object in 3D even with a rough estimation of the relative depth of the object. Accurate tracking of the object even with an unknown depth.

Finally, one of the tested methods can track multiple objects at the same time in a simple setup while still being fast enough to run in real-time as shown in 4.6.

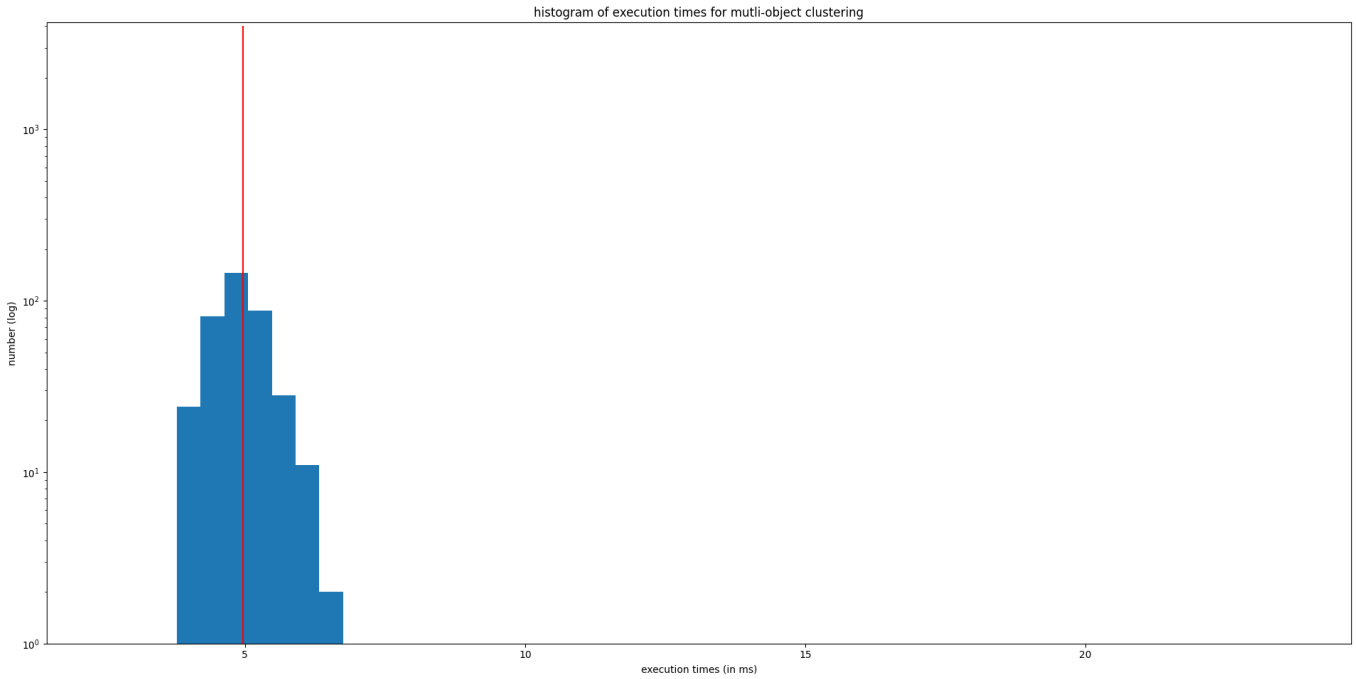

Figure 4.6: Histogram of the execution time for two objects tracking with feature method. Average execution time in red.

— 5 —
Conclusion

5.1 conclusion

Event-based cameras present some solid advantages in term of time resolution and power consumption making them particularly well suited for embedded systems and robotics application but lack the maturity that have been reach in the processing of standard images and video-stream due to their relative recent development.

During this internship we proposed a comparison of simple processing pipelines that can be use as base in future work in the filed of aerial robotic. Those pipelines proved to be quite accurate despite their simplicity and able to run fast enough to be executed in real time.

5.2 future work

Because of lack of time and the difficulties encounter during the internship, there is still work to be done to achieve indoor social navigation for drones.

Firstly, the current pipeline can't work on a moving camera. This can be corrected by implementing an ego-motion compensation step on the events prior to the segmentation part. This step is used to discriminate events produced by the camera motion from the ones produced by an object motion. Moreover, having the camera moving can be useful for obstacle detection and thus, having the possibility to navigate only using the event-based camera.

Also, for now the pipeline is not able to make the difference between a moving object and a moving human. This step is also require for a more refined navigation part such as social navigation.

Finally, the experiment clearly demonstrate that extracting the depth of an unknown object from a single event-based camera is not accurate enough for precise navigation. This can be corrected by adding another type of sensor such as a RGB-D camera (which can also be used to provide more information for a more refined segmentation), a lidar or a stereo setup with two event-based cameras.

— A —
Appendix

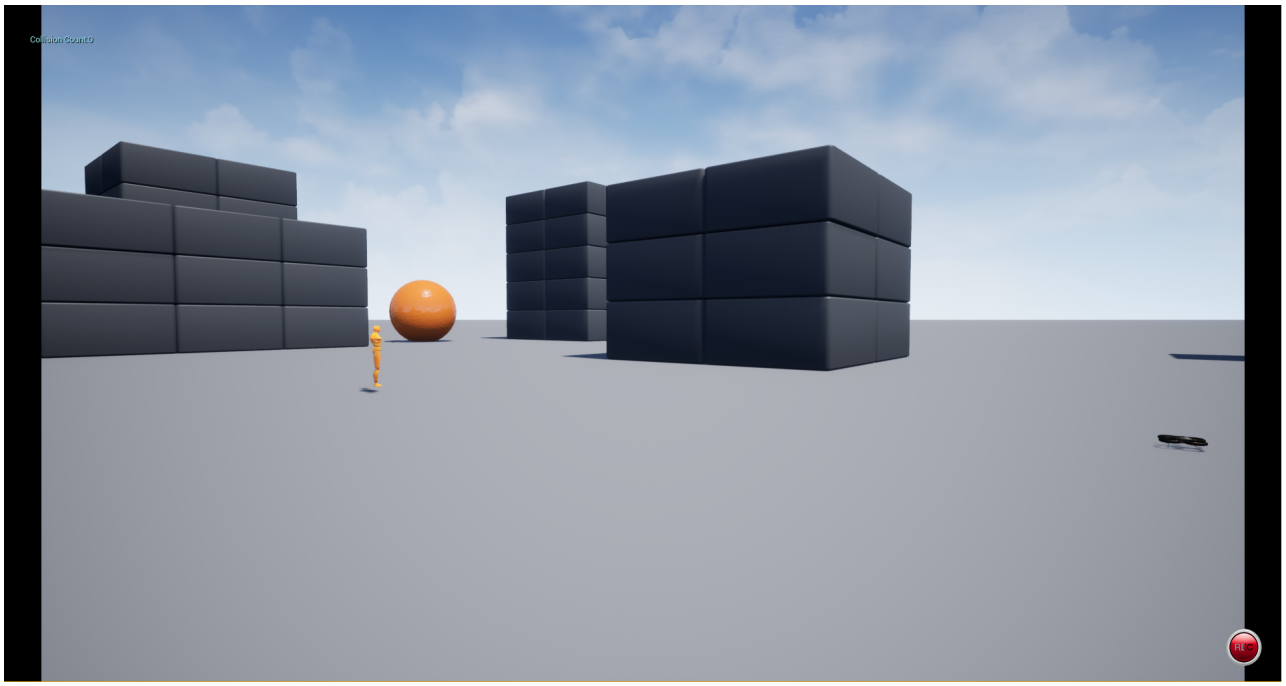

Figure A.1: Example of environment simulated in Airsim.

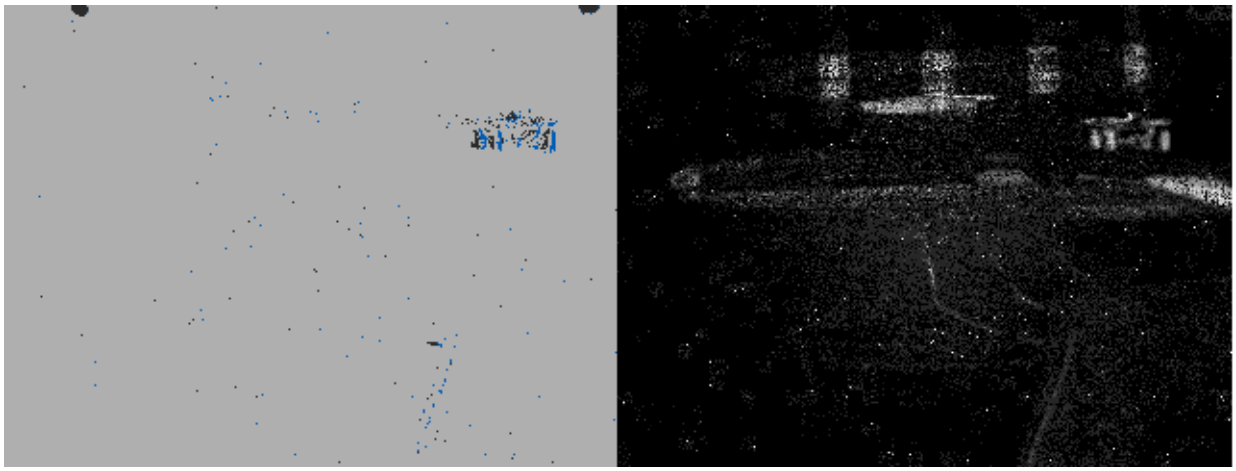

Figure A.2: example of events (left) and accumulated frame (right)

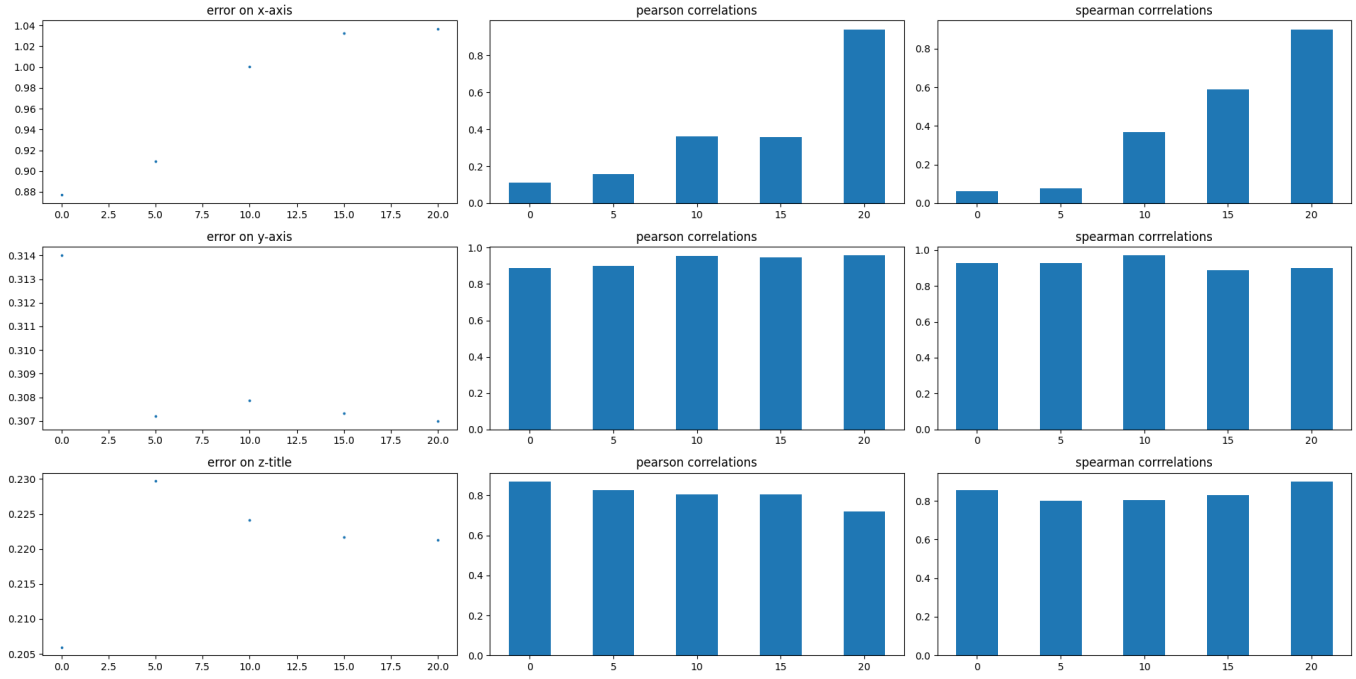

Figure A.3: Example of Mean Error and correlation coefficient used to adjust the algorithm parameters.

Figure A.4: Drone used for the experiment (36cm wide by 14cm high).

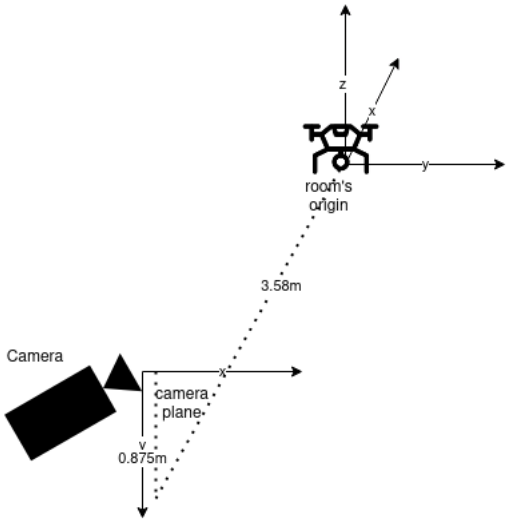

Figure A.5: Representation of the position of the camera in relation to the drone initially at the origin of the room.

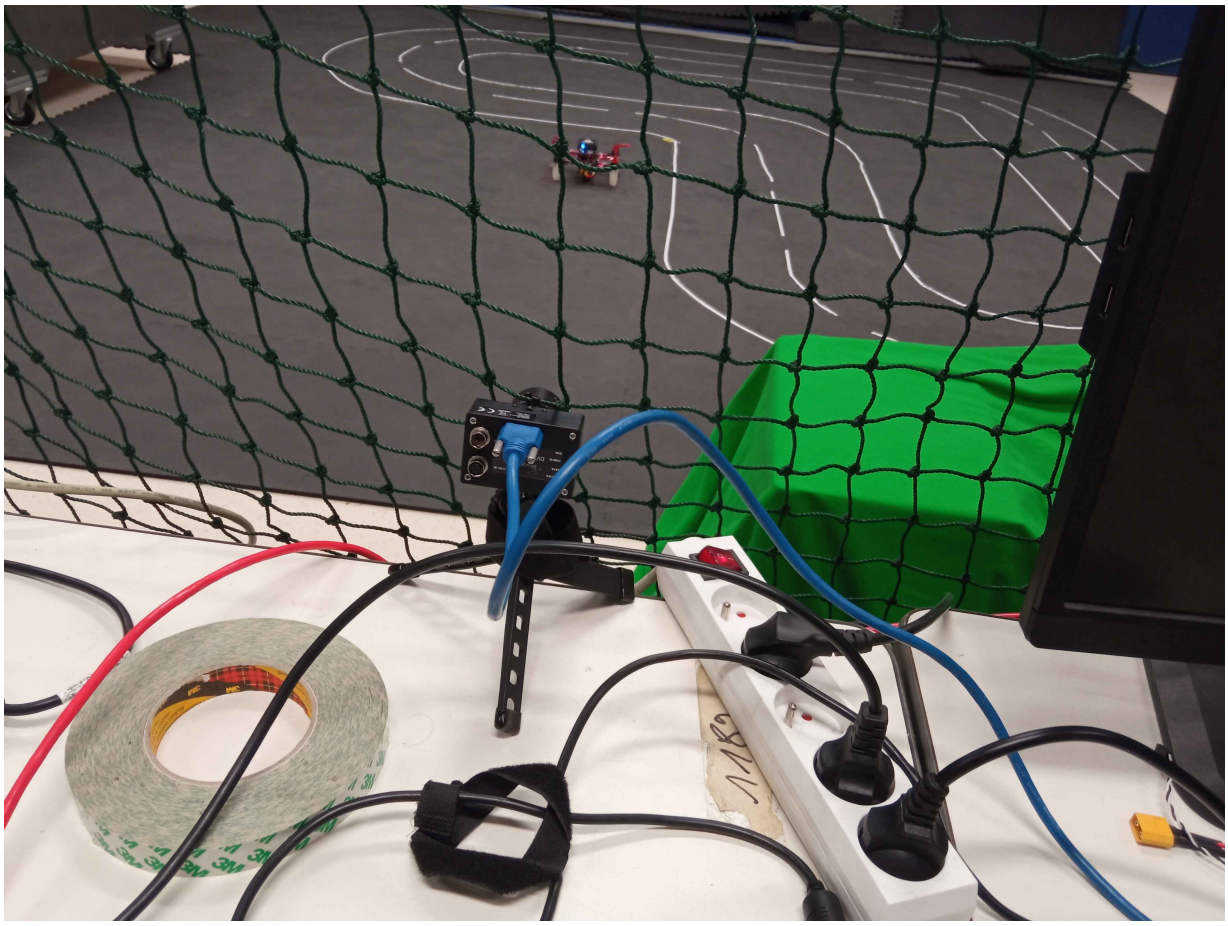

Figure A.6: Photo of the event-based camera looking at the Moca room and the drone in it. The drone starts at the origin of the Moca room.

Bibliography

[1] Remi Paulin, Thierry Fraichard, and Patrick Reignie. Using human attention to address human–robot motion. *Robotics and Automation Letters*, 2019.

[2] Event camera.

[3] Hongjie Liu, Diederik Paul Moeys, Gautham Das, Daniel Neil1, Shih-Chii Liu1, and Tobi Delbrück. Combined frame- and event-based detection and tracking. *International Symposium on Circuits and Systems (ISCAS)*, 2016.

[4] D. Falanga, K. Kleber, and D. Scaramuzza. Dynamic obstacle avoidance for quadrotors with event cameras. *Science Robotics*, 5, 2020.

[5] M. Litzenberger, C. Posch, D. Bauer, A.N. Belbachir, P. Schön, B. Kohn, , and H. Garn. Embedded vision system for real-time object tracking using an asynchronous transient vision sensor. *Digital Signal Processing Workshop, 4th IEEE Signal Processing Education Workshop*, 2006.

[6] T. Delbruck and P. Lichtsteiner. Fast sensory motor control based on event-based hybrid neuromorphic-procedural system. *International Symposium on Circuits and Systems*, 2007.

[7] Xavier Lagorce, Cédric Meyer, Sio-Hoi Ieng, David Filliat, and Ryad Benosman. Asynchronous event-based multikernel algorithm for high-speed visual features tracking. *IEEE transactions on neural networks and learning systems*, 15, 2015.

[8] Yi Zhou, Guillermo Gallego, Xiuyuan Lu, Siqi Liu, and Shaojie Shen. Event-based motion segmentation with spatio-temporal graph cuts. *transactions on neural networks and learning systems*, 2021.

[9] Ewa Piątkowska, Ahmed Nabil Belbachir, and Margrit Gelautz. Spatiotemporal multiple persons tracking using dynamic vision sensor. *Computer Society Conference on Computer Vision and Pattern Recognition Workshop*, 2012.

[10] David Weikersdorfer, David B. Adrian, Daniel Cremers, and Jorg Conradt. Event-based 3d slam with a depth-augmented dynamic vision sensor. *International Conference on Robotics & Automation*, 2014.

[11] Antoni Rosinol Vidal, Henri Rebecq, Timo Horstschaefer, and Davide Scaramuzza. Ultimate slam? combining events, images, and imu for robust visual slam in hdr and high speed scenarios. *Robotics and Automation Letters*, 2017.

[12] Yi Zhou, Guillermo Gallego, Henri Rebecq, Laurent Kneip, Hongdong Li, and Davide Scaramuzza. Semi-dense 3d reconstruction with a stereo event camera. *European Conf. on Computer Vision (ECCV)*, 2018.

[13] Timo Horstschaefer, Davide Scaramuzza, and Henri Rebecq. Real-time visual-inertial odometry for event cameras using keyframe-based nonlinear optimization. *British Machine Vision Conference*, 2017.

[14] Henri Rebecq, Timo Horstschaefer, Guillermo Gallego, and Davide Scaramuzza. Evo: A geometric approach to event-based 6-dof parallel tracking and mapping in real-time. *Robotics and Automation Letters*, 2016.

[15] Antonio Loquercioand Elia Kaufmann, René Ranftl, Matthias Müller, ladlen Koltun, and Davide Scaramuzza. Learning high-speed flight in the wild. *Science Robotics*, 2021.

[16] Boris Kluge and Erwin Prassler. Recursive agent modeling with probabilistic velocity obstacles for mobile robot navigation among humans. *Springer Tracts in Advanced Robotics*, 35, 2007.

[17] Antoine Bautin, Luis Martinez-Gomez, and Thierry Fraichard. Inevitable collision states: a probabilistic perspective. *IEEE International Conference on Robotics and Automation - ICRA 2010*, pages 4022 – 4027, 06 2010.

[18] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. *Robot Operating System (ROS): The Complete Reference (Volume 1)*, chapter RotorS—A Modular Gazebo MAV Simulator Framework, pages 595–625. Springer International Publishing, Cham, 2016.

[19] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017.

[20] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 1995.

[21] Luca Longinotti. dv-processing. https://gitlab.com/inivation/dv/dv-processing.

[22] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[23] Nicolas Marchand, Ahmad Hably, Amaury Negre, and Jonathan Dumon. Moca room. http://www.gipsa-lab.grenoble-inp.fr/recherche/poles-de-plates-formes/plates-formes.php?id_plateforme=100.