Reconstruction of trajectories-Challenge AMIES

Kyriaki Dariva, Pedro Jaramillo

Reconstruction of trajectories-Challenge AMIES

Kyriaki Dariva[1] and Pedro Jaramillo[2]

[1]Univ Lyon, Université Claude Bernard Lyon 1, CNRS UMR5208, Inria, Institut Camille Jordan, F-69603 Villeurbanne, France
[2]Université de Bordeaux, Institut de Mathématiques de Bordeaux UMR 5251, 33 405 Talence, France

October 4, 2021

Abstract

Given a set of successive images which contain the positions of people in a certain space and at a certain time, we propose a method to reconstruct their trajectories. A surveillance 3D-camera takes a photo of the place every a certain fraction of the second and provides us with the data, after processing each picture and retrieving the position of the people in its view. We connect points from successive images by considering that people's position evolves with optimal transport . Our approach is mainly implemented by using a linear programming function, already available in Python. As the camera produces imperfect measurements, we define a framework to classify data and filter out noise. After presenting the problem of optimal transport of finite points as a linear programming problem, we provide some details of our modeling procedure. Finally, we show some of the results obtained with our method while we also suggest ideas to further improve our work.

1 Introduction

The problem of trajectory reconstruction was one of the challenges in the virtual event *"Challenge mathématiques et entreprises" organized by the Agence pour les Mathématiques en Interaction avec l'Entreprise (AMIES). It was proposed by the French company EURECAM whose main activity is to design and construct sensors that count the number of people inside a certain region. They are currently interested in determining the flux of people entering and leaving an area. This could provide valuable information on their customers as it could describe their foot traffic behavior or help improve security measures without needing someone to actually review hours of footage.*

A 3D-Camera is set to oversee a certain place. It takes photos of the scene with a certain frequency and then for each photo it provides a set of positions in space while it also

identifies the position of the head. Therefore, the data we are working with are successive images, let's say $1, 2, 3, ...$ *to put them in order. In each image* i *there is a set of points and each point is described as a tuple with* 5 *coordinates:*

$$(i, x, y, z, h) \in \mathbb{N} \times \mathbb{R}^2 \times \mathbb{R}_+^2, \tag{1}$$

where (x, y) *is the position in the horizontal plane,* z *is the distance of the person's feet to the camera (in the coordinate system of the sensor) and* h *is the height measured from the top of the person's head to the floor. The* z *coordinate is only used to plot the data, it plays no role in our modeling and can be ignored. We do include it in all our definitions just to be consistent with our dataset.*

Images are taken in very short time intervals, one after the other (more than two images are produced each second). Therefore we consider linking points between two consecutive images with line segments as a first approximation. As we dispose many images, we can construct trajectories using straight lines and still be accurate enough. Thus, it suffices to determine how to link successive images together to see how trajectories evolve in time. Before giving some implementation details of how to create, delete, and track trajectories, we first describe the association process between two images.

2 Optimal transport and linear programming

Optimal transport theory can be informally described using the words of the French mathematician Gaspard Monge (1746-1818): A worker with a shovel in hand has to move a large pile of sand lying on a construction site. The goal of the worker is to erect with all that sand a target pile with a prescribed shape (for example, that of a giant sand castle). Naturally, the worker wishes to minimise her total effort, quantified for instance as the total cost, distance or time spent carrying shovelfuls of sand.

In our case the pile of sand pile represents points from one image and the sand castle represents the points in the next image. Therefore we wish to minimize the distances that would be traveled by everybody. Hence the cost function should take into account the distance between two points in different images. The cost function should take into account other quantities that should be minimized as well, but we will give more details and all the justifications in the next section. For now we only consider a general cost function which we denote by c*. Let us now translate our optimal transport problem to a linear programming problem.*

Let $(x_i)_{i \in I}$ *denote the set of points in image* k *and* $(y_j)_{j \in J}$ *the set of points in the next image* $k + 1$*.*

Firstly assume that $|I| = |J|$*. For each pair* $(x_i, y_j)_{i,j}$ *we can calculate the cost of linking the point* x_i *with* y_j*. We thus obtain a cost matrix, also denoted by* c*, of size* $I \times J$*. We now define the matrix* p *which is of the same size as* c*. For each* $i \in I$ *and* $j \in J$*, there are two possibilities for the entries of* p*:* $p_{i,j} \in \{0, 1\}$*. If* $p_{i,j} = 1$ *then the point* x_i *is linked*

to the point y_j, whereas if $p_{i,j} = 0$ then those two points are not coupled. Thus, for any coupling matrix p we obtain a linking between the points in those two images. Matrix p can have a very general form but as we are only interested in trajectory reconstruction, each point in image k can be associated to only one point in image $k+1$. Hence, we impose the restriction:

$$\sum_j p_{i,j} = 1, \quad \forall i \in I. \tag{2}$$

Similarly two points in image k cannot be link to the same point of the image $k+1$ as two different people cannot occupy the same position. Thus, we also consider the following constraint

$$\sum_i p_{i,j} \leq 1, \quad \forall j \in J. \tag{3}$$

Finally, the quantity that we want to minimize is the following function of p:

$$\sum_{i,j} p_{ij} c(x_i, y_j) \tag{4}$$

subject to the constraints (2) and (3). As (4) is a linear function of p and the constraints are also linear, our case is a linear programming problem. For more information on linear programming we refer to [1].

We reformulate this to put it in the standard form used by the linprog library in Python.

$$\begin{cases} \min_p\{c^T \cdot p\}, \text{ such that} \\ A_{eq}p = b_{eq}, \text{ and} \\ A_{ub}p \leq b_{ub}, \end{cases} \tag{LP}$$

where A_{eq} and A_{ub} are the conditions (2) and (3) in matrix form, c and p are the cost and the coupling matrix respectively, in vector form, and b_{eq} and b_{ub} are column vectors filled with ones. The values in b_{eq} and b_{ub} correspond to the right hand side of (2) and (3).

In the general case $|I| \neq |J|$: This is possible since people can enter or leave the region tracked by the camera. Let n_k denote the number of points in image k and n_{k+1} the number of points in image $k+1$. There are three possibilities:

- *$n_k = n_{k+1}$. In that case every point in image k is connected to exactly one point in image $k+1$ and every point in image $k+1$ receives exactly one point from image k. It is the case of (LP) described above with (3) becoming an equality (LP is written using only the matrix A_{eq}).*
- *$n_k > n_{k+1}$. Every point in image $k+1$ receives exactly one point from image k and every point in image k is linked to at most one point from the image $k+1$.*
- *$n_k < n_{k+1}$. Every point in image k is connected to exactly one point in image $k+1$ and every point in image $k+1$ can receive at most one point from image k.*

The solution to (LP) *is the coupling matrix* p *which gives the linking of points between two images by minimizing the total cost.*

Remark 1. *We would like to mention here that in a more general framework the vectors* b_{eq}, b_{ub} *can have other entries besides* 1 *and* 0*. For example, if we consider* b_{ub} *and* b_{eq} *to be column vectors consisted of twos, that would mean (respectively) that a point in image* k *has two possible choices in image* $k+1$ *and that each point in image* $k+1$ *could be connected with two points from the previous image. If that scenario was taken into consideration, it would improve the robustness of our solution, but remains to be treated in some future work.*

Remark 2. *We chose the revised simplex method for linprog since it is known to be computationally efficient. For* n_k *points in image* k *and* n_{k+1} *points in image* $k+1$ *the revised simplex method takes about* $n_k n_{k+1}$ *computations in practice. This means that we could easily consider hundreds of people and still manage to keep the computational cost relatively low.*

3 Modeling

In this section we will describe some of the difficulties that appear when we are working with real data. We will present our approach to overcome some of these hurdles, the assumptions that we make and some of the issues that we could not solve. Let us first introduce the notations and give some basic definitions.

In what follows the term frame will be used interchangeably to denote the image taken by the camera and the set of data points that are associated to that image.

3.1 Preliminaries

Let $\Omega \subset \mathbb{R}^2 \times \mathbb{R}_+$ *be the smallest rectangular set that contains all the data points recorded by the camera. We will call this* Ω *the domain of observation. We now detail what exactly we mean by a trajectory:*

Definition 1. *A trajectory is an ordered list of states* $[S_1, S_2, S_3....., S_{current}]$*. States are described with six coordinates:*

$$S := (f, x, y, z, h, v) \tag{5}$$

where f *is the number of the picture (frame) where the data point* (f, x, y, z, h) *was recorded and* $v = (v_x, v_y)$ *is the velocity at this point of the person that follows this trajectory. The point associated to state* S_i *goes to the point of state* S_{i+1} *with optimal transport.*

Definition 2. *We will say that a trajectory* ***dies*** *at frame* f_0 *if its last state is inside* Ω *and it is expected to be outside* Ω *before the next frame* $f_0 + 1$ *is recorded. The set of dead trajectories at frame* f_0 *will be the trajectories that have died before or at frame* f_0*. We say a trajectory is* ***active*** *at frame* f_0 *if it is not dead and if its last state is in frame* f_0*.*

Observing the given dataset helped us to come up with a solution. An algorithm was constructed based on the behavioral trends of the data that we have to describe. Apart from false measurements taken by the sensor, the fact that the camera tracks real people makes the flow very heterogeneous. Hence, wrong measurements can be attributed to different factors.

3.2 Observations

A careful study of the videos provided by the camera leads us to the following remarks:

- *The rate of frames varies in time. Additionally, it may not always be high enough to have an accurate approximation by line segments.*
- *Measurements are noisy, meaning that either there are points missing (people that are not recognized by the camera at all) or that there are points that do not correspond to human beings. Indeed, a big part of the noise is due to objects in the scene which are captured as people crossing the region.*
- *Another important source of noise is the multiple measurements of the same person in one frame. For example, it is not uncommon to see that in one frame, the head, the knees and the feet of a person are recorded by the camera as five different people that are really close to each other.*

Directly applying the approach described in section 2 will not work. That is because, even without all the above phenomena, we still need to appropriately handle the people that leave and enter the domain of observation. We will divide our solution in 3 steps.

1. *For each frame, we will first filter the currently active trajectories to remove noise and dead trajectories.*
2. *Next, we will filter noise points from the next frame.*
3. *Lastly, we will link currently active trajectories to points from the next frame. Based on the number of active trajectories in the current frame and the number of data points in the next frame, we will either neglect data points, create new trajectories or extend trajectories.*

We consider the following setup to model this problem. The state of the system at frame f_0 will be described by: the set of active trajectories at frame f_0, the set of dead trajectories at frame f_0 and the set of noise points until the frame f_0. The idea behind this, is to be able to record the past trajectories and the noise registered by the camera, and to get an idea of the type our domain (for example if it is a street, a plaza or an elevator). That way we can more accurately eliminate noise points from frames and classify trajectories as time passes and more data is collected.

3.3 Filtering trajectories

For each frame we first filter noisy trajectories and then dead trajectories. The order in which this is done is important (if done differently some noisy trajectories could be classified as real trajectories).

1. ***Noisy trajectories***

   When looking at the footage and the data provided by EURECAM we notice that for a considerable amount of the time the domain of observation had nobody inside it. In that case all the measurements were noise, and as most of the noise is linked to static objects in the scene (for example a table), the trajectories that were created just moved arbitrarily in one part of the domain. This gave us the idea to classify a trajectory as noise if it ever lived too long or if it changed directions abruptly too many times.

   Definition 3. *We say that a trajectory changes direction abruptly at a frame f_0 if its current velocity v_0 and its velocity v_1 at in the next frame verify*

   $$v_0 \cdot v_1 < 0. \tag{6}$$

   Let $n_{vc} \in \mathbb{N}$ be a fixed parameter denoting the upper bound of the number of changes in velocity direction that we allow. If a trajectory abruptly changes direction more than n_{vc} times, then we consider all of its states as noisy data points, we remove it from the set of active trajectories and we add all of the points in the noise set.

   Choosing the value of n_{vc} parameter correctly may need to take into consideration the frame rate and the general speed expected for people traversing the scene. For instance, in a hall inside a building one would generally expect people to abruptly change their direction at most once. In that case we could pick $n_{vc} = 1$ or $n_{vc} = 0$.

   Another possibility of a noisy trajectory is when points tracked by the camera generate trajectories that turn around in circles (or that do not move much) without abruptly changing directions many times. For that scenario we define the lifetime of trajectories.

   Definition 4. *We call lifetime of a trajectory the difference between the frame number of its last state and the frame number of its next state.*

   We provide a threshold for a very long lifetime. Let $n_{lt} \in \mathbb{N}$ be a fixed parameter. We consider that a trajectory has lived too long if its lifetime is greater than n_{lt}.

2. ***Dead trajectories***

   It is necessary to determine which people are leaving from Ω at frame f_0, so that we stop considering their trajectories as active in the next frame.

As we cannot actually measure data points outside Ω, we use the last state of a trajectory to predict if it will exit the scene or not. To do so, we take into account the last position of a trajectory at the current frame and the velocity with which it is moving.

Let $\delta_x, \delta_y > 0$ be two fixed parameters and let $x_{min}, x_{max}, y_{min}, y_{max} \in \mathbb{R}$ define the border of Ω. We expect a trajectory to leave Ω if its last state $(f_0, x, y, z, h, (v_x, v_y))$ verifies:

$$|x - x_{min}| < \delta_x, \ \ or \tag{7}$$
$$|x - x_{max}| < \delta_x, \ \ or \tag{8}$$
$$|y - y_{min}| < \delta_y, \ \ or \tag{9}$$
$$|y - y_{max}| < \delta_y, \tag{10}$$

and

$$(x + v_x, y + v_y) \notin \Omega. \tag{11}$$

Ideally, the last condition would suffice but we add the first set of conditions just to avoid any unforeseen behavior attributed to noise. Once a trajectory dies we remove it from the active trajectories list and add it to the list of dead trajectories.

3.4 Filtering data points

We distinguish two sorts of noise:

- *Points from static objects in the scene.*
- *Multiple measurements for the same person.*

1. ***Static objects in the scene***

   This is one of the simplest tests that we had to implement. The basic idea is that if there is an object in the scene, we do not expect to find measurements of actual people near it. This is reasonable as people tend to avoid obstacles when walking. In order to give a sense of what near means we consider the usual euclidean distance:

$$d((x_0, y_0, h_0), (x_1, y_1, h_1)) = \sqrt{|x_0 - x_1|^2 + |y_0 - y_1|^2 + |h_0 - h_1|^2}.$$

   Let $\delta_n > 0$ be a fixed parameter. For every point in a new frame, we check if its euclidean distance to any of the points in the noise set is less than δ_n. If it is the case, then we classify it as noise and we neglect it (we put it in the noise set). Otherwise, we continue to observe it. This is a crude test but it was a good enough start for testing the real data from EURECAM.

2. ***Duplicate measurements***

What we call duplicate points is when we have two points near each other (in the $xy-$plane) but with considerably different heights. For example, what tends to happen most often is that the camera registers the head of the people along with both of their feet as if they were three different people.

A first approach to deal with this imperfect measurements could be to just remove all data points with height less than a certain value. We did not choose to do it because regularly only a person's foot is registered by the camera. Nevertheless, our approach remains simplistic but quick to implement and iterate on. More precisely, for each data point in the frame we look if it is near to another point in the frame. If this is the case, we ignore the data point and keep it otherwise.

Let $\delta_d, \delta_h > 0$ be fixed parameters. To give a meaning to the notion of near, we introduce the following function

$$d_h((x_0, y_0, h_0), (x_1, y_1, h_1)) = \sqrt{|x_0 - x_1|^2 + |y_0 - y_1|^2} + \delta_h \min(|h_0 - h_1|, \min(|h_0|, |h_1|)).$$

We consider two data points (f, x_0, y_0, z_0, h_0), (f, x_1, y_1, z_1, h_1) to be near each other if they verify

$$d_h((x_0, y_0, h_0), (x_1, y_1, h_1)) < \delta_d \tag{12}$$

Points classified as multiple measurements of one person are just removed from a frame and are not considered when linking points.

Now that we have filtered the duplicates we proceed with the final step of our method, that is to link the current active trajectories to the (clean) data points from the next frame.

3.5 Linking data points to trajectories

From now on we will consider that all the data points represent real people in the scene.

As described in section 2 we will associate trajectories to data points by minimizing the cost function. In the simplest case one would consider the sum of the euclidean distances between the last position of all active trajectories at the current frame and the data points from the next frame. Even in a framework without noise, this solution would not be robust. Changes in frame rate, missing data and differences in height of data points are all factors that should be taken into consideration in the cost function. Instead of using the euclidean distance we propose the following function:

$$d_{OT}((f_0, x_0, y_0, z_0, h_0, (v_x, v_y)), (f_1, x_1, y_1, z_1, h_1)) = cost_h + cost_v + cost_d,$$

where

$$cost_h = (1 - C_{align})|h_0 - h_1| + C_{align} \min(|h_0 - h_1|, \min(|h_0|, |h_1|)), \tag{13}$$

$$C_{align} = \begin{cases} \left| \frac{v_x}{|(v_x,v_y)|} \frac{(x_1-x_0)}{|(x_1-x_0,y_1-y_0)|} + \frac{v_y}{|(v_x,v_y)|} \frac{(y_1-y_0)}{|(x_1-x_0,y_1-y_0)|} \right|, & \text{if } |v||(x_1 - x_0, y_1 - y_0)| > 0, \\ 1, \quad \text{otherwise} \end{cases}$$

$$cost_v = \sqrt{|x_1 - x_0 - (f_1 - f_0)v_x|^2 + |y_1 - y_0 - (f_1 - f_0)v_y|^2}, \quad \text{and} \tag{14}$$

$$cost_d = \sqrt{|x_0 - x_1|^2 + |y_0 - y_1|^2}. \tag{15}$$

The $cost_h$ term helps differentiate people by their height without penalizing the distance of their head to their feet. In that way, if at some frame the position of a person's foot is registered by the 3D-camera, and at the next frame what is registered is the same person's head, these two points should be relatively close according to the distance function. We do emphasize that the term $cost_h$ recognizes some height differences, for example it can differentiate adult from children fairly well.

We also considered an alignment coefficient C_{align} inside $cost_h$. As the feet are usually either in front or behind the person itself, we wanted to take that into account in our cost function. To do so, we use the velocity with which the person was moving. The less aligned a point is to a trajectory the more we penalize errors in height. This prevents us from linking one person's head with another person's feet if they happen to walk near each other.

The second term takes into consideration the inertia of a person. In short, we expect a person to continue moving with their current velocity between the current frame and the next one. We thus calculate the distance not only to the current position of the trajectory but also to its expected position. This is just a way to add some robustness to our model, as we know that people need time to change direction.

Finally, the third term is the usual 2D euclidean distance one would expect to use. We can now calculate the cost function as in section 2 and obtain a pairing between trajectory and data points. We next proceed to describe the different possible cases we could encounter at this stage.

3.5.1 Same number of trajectories and data points

Let $X \in \mathbb{N}$ be the number of active trajectories at the current frame f and $Y \in \mathbb{N}$ the number of data points in the next frame. If $X = Y$ we consider that nobody has entered the scene in view of the camera and thus we expect that each active trajectory just continues in the next frame. This means that we optimize the distance introduced in the last section and then we add each point to the end of the trajectory it is paired with. Before extending the trajectory, we transform the new point into a trajectory state by adding the velocity component.

Let a trajectory's last state be $(f_0, x_0, y_0, z_0, h_0, (v_x, v_y))$ *and the point it is paired with be* $(f_1, x_1, y_1, z_1, h_1)$*. The next state of the trajectory is then given by*

$$\left(f_1, x_1, y_1, z_1, h_1, \left(\frac{x_1 - x_0}{f_1 - f_0}, \frac{y_1 - y_0}{f_1 - f_0}\right)\right) \tag{16}$$

We now consider the case of more data points in the next frame than active trajectories in the current frame. This is not very different from what we already did but it is still worth detailing.

3.5.2 More data points than trajectories

The only possibility in that case is that people have entered the scene. We will therefore add them as new trajectories to the set of active trajectories.

First, we pair trajectories as described above. Then, the unpaired points will be transformed into a list each with only one element. If (f, x, y, z, h) *is such a data point we will consider the new trajectory that starts at this point:*

$$traj = [(f, x, y, z, h, (0, 0))], \text{ using Python code notation.} \tag{17}$$

The choice of the velocity vector $(0, 0)$ *is arbitrary. There are probably better options, but we chose it like this just to avoid adding additional cost to this state when linking it to another point later on.*

3.5.3 More trajectories than data points

Lastly we see what happens when there are more active trajectories in the current frame than data points in the next frame. As dead trajectories have already been taken into account, this can only mean that some people have not been recognized by the camera. It could also mean that we were maybe too strict when eliminating noise from the frame, but we will not consider this case.

We have a few ways to handle this scenario. A first option would be to join trajectories as usual and then extend the unpaired trajectory by just considering that their actual position is their next expected position. This is a sensible idea but in practice it makes it hard to filter noise trajectories. What happens is that static objects create new trajectories and before they can be recognized as noise they exit the scene and are thus registered as people.

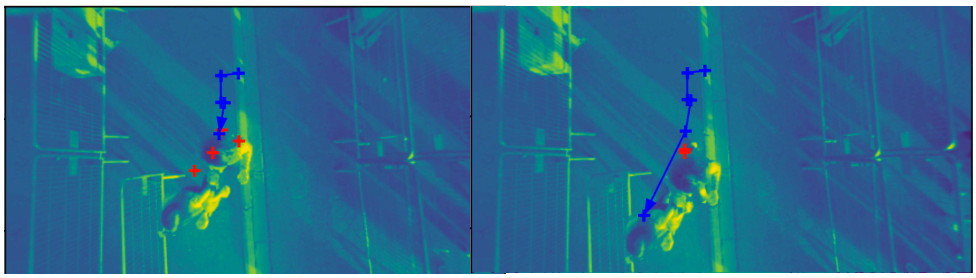
Another choice would be to keep unpaired trajectories as they are. This can also cause problems. Indeed, it is possible that such a trajectory stays inside the domain when the person corresponding to it exits the domain. Consequently this trajectory could be linked to the data point of another trajectory or eventually be considered as noise and removed .

We chose the second approach, as we prioritized to have fewer but true dead trajectories. It is however easy to imagine some kind of hybrid approach where for example we use the second option until the noise set is big enough and then change our approach to the first option.

4 Results and conclusions

We now show some of the results when we tested our code with the data provided by EURECAM. We cannot include the actual data in this work but we do provide our code and a video that shows our results [2]. For the most part in this section, we just show some images which reveal the behavior of the proposed solution.

We first show the case of two people walking together across the scene in a straight line. We can see in figure 1 two successive frames of the video recorded by the camera.

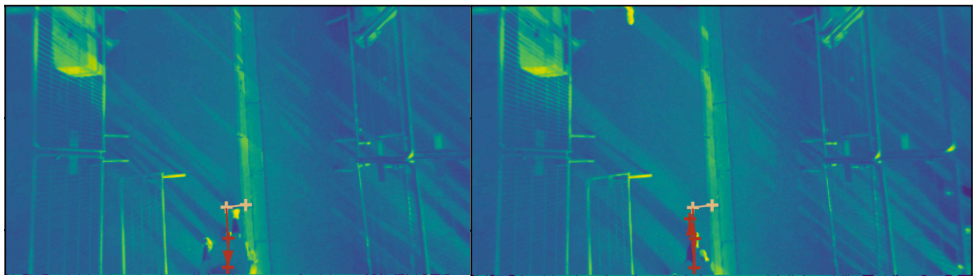


(a) First frame (b) Next frame

Figure 1: Two successive frames where only two people are traversing the scene in a relatively straight line.

We can see in figure 1a that the one person has produced multiple measurements and the other has only one, which is a foot. We count these data points as if it were only one person. This can be due to the size of the parameter δ_d that we are currently using in combination with lack of measurements in certain frames, as can been seen in figure 1b. In that case the last point of the trajectory (in blue) and the red cross are close enough according to our test. They are therefore considered to represent the same person. As a result, the red cross is not taken into account when linking trajectories.

We would like to note that we did not have many criteria to chose the parameters the way we did. Accuracy can easily be improved by optimizing the choice of parameters.



(a) First frame (b) Next frame

Figure 2: Two successive frames where one person is leaving the field of view of the camera.

Figure 2 shows a common problem that we had when people leave the domain. What tends to happen, is that when someone exits the domain, their head may leave from the current frame but the foot, which is behind the head, is tracked by the camera in the next frame. As such, the trajectory turns back as in figure 2b. Hence the person has left the domain but the trajectory representing that person still remains. It can then infect some future trajectories as in figure 3. This could be easily fixed with a small change in the part of our code which filters out duplicate measurements. We could, for example, consider removing points that are duplicate and keep the one that is the farthest away from the floor.

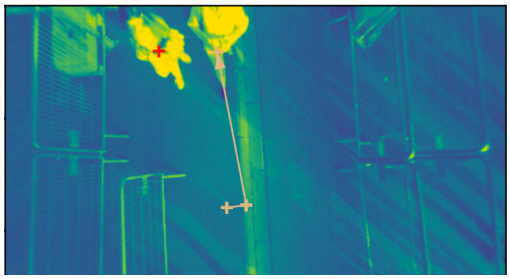
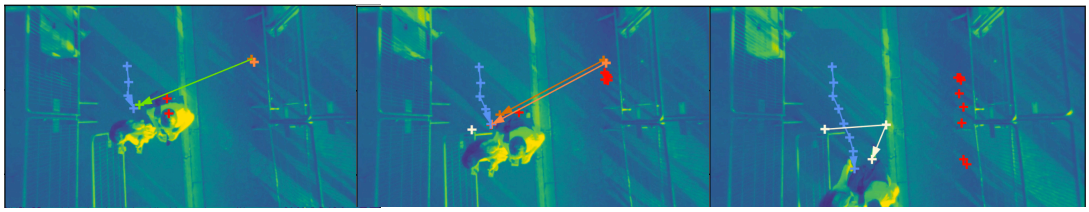


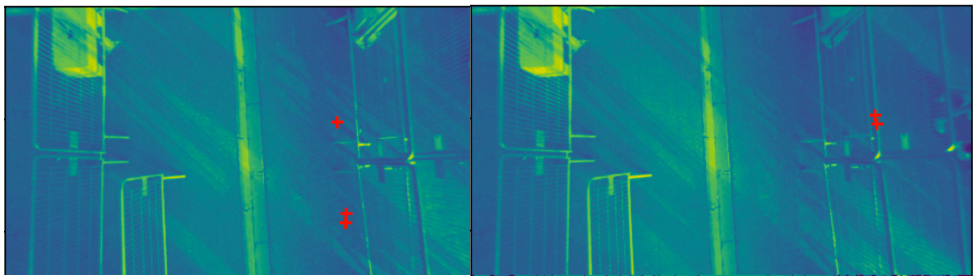
Figure 3: Someone is leaving the are while someone else is entering it. We can see that if the trajectory of a person leaving is not treated correctly it can infect future trajectories.

The same infection phenomenon can be seen in 4 below. We remark that in that case we do prevent some of the noise coming from the static objects (right part of the image) as can be seen in figure 4c. The red crosses are data points that have been picked up by the camera but we manage to correctly identify them as noise and do not infect the other trajectories.



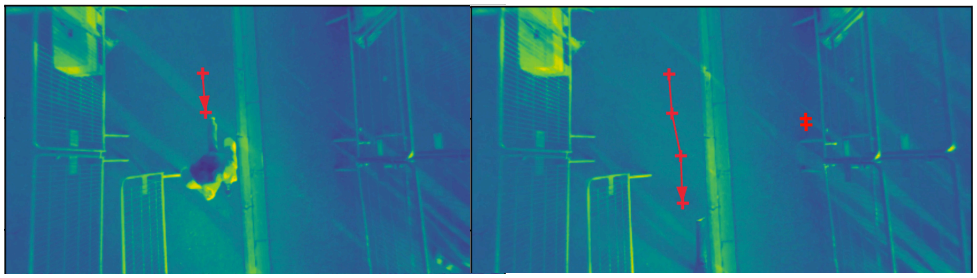
(a) First frame. (b) Next frame. (c) Last frame.

Figure 4: Noise coming from static objects can infect people that are far away if we blindly apply optimal transport without considering the actual value of the cost function.



(a) (b)

Figure 5: Two different images (not necessarily successive) show that we can manage to identify some data as noise accurately enough.



(a) (b)

Figure 6: Our method can even handle people that are moving relatively fast. These two frames belong to a person running across the area. Her trajectory is accurately registered and classified as dead when she is outside the domain of observation.

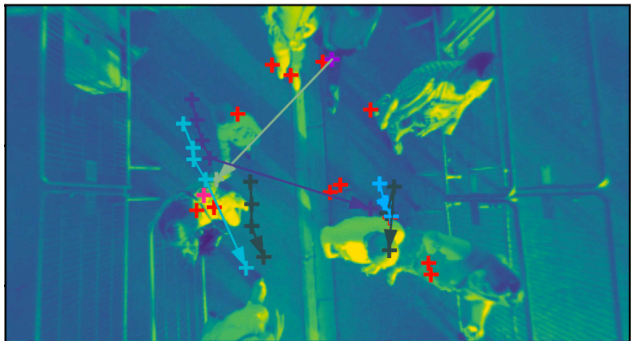


Figure 7: Image of a frame when a lot of people are passing through the domain as a group. We see most of the problems described above happening simultaneously

In figure 7 we see why our method should still be improved.

4.1 Iterating on our code

Our method can be improved with relatively simple changes. Most of these can be easily implemented in the code because most of the solution comes down to the cost function that we choose.

One small change could be to make the parameters introduced above depend on the dead trajectories and the noise set. That way, ideally, the approach would improve over time. Moreover this could help us determine the possible trends in the behavior of the people inside Ω*. If for instance the observed domain is a busy street, then we could calculate the mean velocity of people to filter out noise more accurately.*

If we wish to avoid mistakes when linking trajectories to points, we could also take into account the value of the cost function and compare it to the mean value of the total cost from previous frames. Even a simple enough test could provide some insight to help detect if we should create a new trajectory from a data point to the next frame or not. This could help in the case where a person enters Ω *and another person who is already inside is not detected by the camera.*

Other simple changes we could consider would be to better define velocities and velocities of a trajectory. As implemented in our code right now we consider the frame rate of the camera to be constant. A better definition would actually use time intervals. We could also consider improving upon our noise detection by consider more subtle tests. Right now we just consider a data point to be noise if it is near something that has already been classified as a noise. It would be better if we took into account the number of noise points that are close to it, for example.

We remark that even though all these changes can improve the accuracy of our solution, in the end the best choice is probably to deal with more than one frames at a time. That way our solution would be more robust against noise and lack of measurements.

This combination is one of the major flaws in our approach. The reason we cannot manage it more appropriately is because, as described so far, we consider one frame at a time when linking trajectories to points. If we make a mistake at some frame we cannot later go back and correct it. In order to get a significant improvement, more frames need to be taken into account, however that requires a subtle analysis.

5 Annexe

5.1 Plotting

The images are extrapolated from the videos animated with the simple transformation of a $3D$ *object* (x, y, z) *in cm to a* $2D$ *object* (ix, iy) *of the image in pixels:*

$$\begin{cases} ix = \frac{1}{2}(\frac{xfoc}{z} + cx) \\ iy = \frac{1}{2}(\frac{yfoc}{z} + cy) \end{cases}$$

where foc *is the focal in pixels and* (cx, cy) *are the coordinates of the optical center.*

6 Acknowledgements

We would like to thank Sébastien Tran Tien for his contribution to the development of the code, the video animation and his constructive discussions and suggestions.

References

[1] Jiri Matousek, Bernd Gartner Understanding and Using Linear Programming

[2] https://github.com/strantien/eurecam