Computer vision tracking project

Jacopo Gobbi
[194438]

Università degli studi di Trento
jacopo.gobbi@studenti.unitn.it

Abstract. Detection and tracking are two recurring problems in computer vision, with multiple object tracking being more complex because of having to associate identities in a frame by frame manner. In this project we develop an algorithm which makes use of a feature detector and an optical flow algorithm to establish and keep track of identities during the development of a scene, and test it on a top view case scenario.

Keywords: Tracking · Algorithm · Top view

1 Problem Statement

First, we briefly formalize the problem statement by splitting it between detection and tracking, then we better describe our specific scenario and particular problems involved.

1.1 Detection

Within an image, for one reason or another, we may be interested in detecting any number of objects of arbitrary and heterogeneous nature.
This problem could be simplified by considering sub portions of a picture as pictures themselves, with the assumption that each of these contains a single object, essentially transforming our task in a classification exercise, but probably increasing the required computational effort due to going from one image to many.
This approach is indeed expensive, and in fact, state of the art models try to not split the task so harshly, in order to save computation by sharing the effort in feature elaboration, as in [4].
We can reason about detection in terms of a single frame (picture) at a time or considering sequential frames (a video) that would allow us to distinguish objects with the help of temporal changes by introducing memory in our algorithms, by doing this we usually end up doing object detection by doing motion detection; meaning that our objects of interest might only be objects that are part of the foreground and/or are moving around.

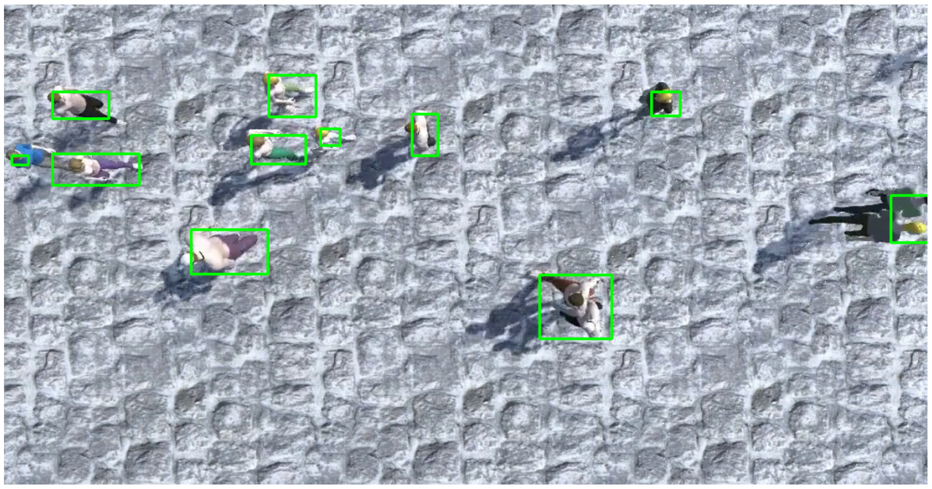

Fig. 1. Example of object detection, in this case an algorithm is trying to put a bounding box around what it considers to be people.

1.2 Tracking

Object tracking refers to keeping track of one or more arbitrary objects during a series of images depicting the temporal evolution of a scene, usually done by assuming that the change is ”slow” enough that the object will be somewhere near where it was in the previous frame.
Tracking has many applications, ranging from people monitoring, behaviour understanding, or even cell tracking, and can be essentially summed up by two aspects: understanding what is moving, and understanding what it is doing or how it is interacting with other objects in the scene.
This task is not trivial given that we might have noise, shadows, splitting, merging, occlusions or other issues essentially damaging the ability of our algorithm to build a consistent representation of what it is meant to track, this is especially problematic if our object is also changing in time due to prospective or for other reasons.
Tracking is paired with identity association in the sense that if we are tracking multiple objects, each one with its own identity, we can’t stop at finding where the objects are, but we need to also reason about mapping detections in the previous frame with detections in the new; this is usually done with some form of distance or cost, like the distance of the centers of bounding boxes, their overlap, difference in the color histograms of the regions defined by the bounding boxes, and so on.

1.3 Our specific instance of the problem

Our problem is about counting and keeping track of pedestrians observed from a top view, as it can be seen in figure 1 and 2; the camera is fixed and the

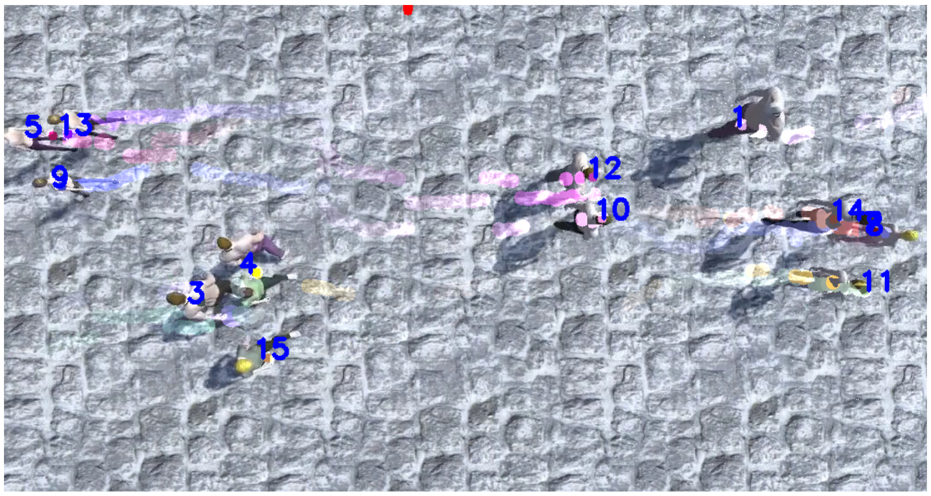

Fig. 2. Example of an algorithm associating identities to objects, and drawing their trajectories in different colors.

background is static, nevertheless, the problem is not simple due to the change in lighting and shape that people go through while moving from one side of the scene to another, made worse by the presence of shadows.
Moreover, the background is somewhat similar in color to the torso of some of these pedestrians, and occlusions are frequent; there aren't many peculiar details for each person, they are usually simply defined by the shape and color of their head, torso and legs. Luckily, people maintain their horizontal direction after bumping into others.
As an additional task, we also keep track of how many people are entering and exiting both sides of the scene.

2 Developed Method

In this section we describe the process that led to a solution, dividing it between detection and tracking. Unless stated, opencv [3] was used as an implementation of the used algorithms.

3 Detection

The main problem in this phase was due to shadows being mis-recognized as "real" moving objects, to deal with that, first we moved from the RGB color space to the HSV space, and then kept only the HUE plane, blurring it with a 4x4 filter.
HUE was chosen as an input for a background sub-tractor because the HSV space is deemed a robust representation for image analysis, and indeed this was

what let us get rid of shadows without having to play around too much with the parameters of the background subtractor; moreover, this would lighten computation by only having to deal with one plane instead of three.
As a background subtractor we used a gaussian mixture-based algorithm, named BackgroundSubtractorMOG2 in the opencv library. It was chosen due to its ability to automatically infer the number of gaussian distributions needed for each pixel, and due to the fact that it "provides better adaptability to varying scenes due illumination changes", as stated by the opencv documentation; this algorithm is based on two different contributions, [9, 10].
Later, morphological transformations were applied to the output of MOG2 to clear away noise and conglomerate blobs, more specifically, we applied 1 iteration of opening (2x2 ellipse kernel) and 5 of closing (4x4 ellipse kernel).
Done that, we use the algorithm from [8] to find the contours of people, and exclude those that are deemed improbable: either too small, too big, or in a place in the scene that is not traversed by people. What is left is the output of the detection phase: a set of contours/bounding boxes where each ones of these should contain a person in the scene.

3.1 Tracking

Tracking has been done considering that there are no strong, unique and permanent features to track on each person, instead, we use an approach that considers a swarm of many (weak, low quality) features.
Specifically, given a feature detector and an optical flow estimator, our algorithm assigns an identity to features, and tries to find those features in the next frame to establish where the identity has gone.
The feature detector and the flow estimator are interchangeable, one could use faster or slower ones depending on his computational resources, or generally try what are the ones working best for his problem; in our case, we use good features to track [7] to detect features and Lukas-Kanade [6] to find them in the subsequent frames.
Good features to track was particularly handy because it allows to easily choose the minimum quality of the points we are going to accept as features, we used a low quality threshold: we would accept any point having at least a score equal to 1% of the best feature found. This gives us robustness in the sense that even losing half or more feature points of an identity should allow us to still find it the next frame, essentially, we are trying to not have a single point of failure, distributing the risk instead.
The decision to use many weak features comes from the fact that simply using bounding boxes and a Kalman Filter [5] to keep track of their movement and identity was not enough because of occlusions and because of bounding boxes not being particularly stable, changing shape or disappearing from time to time.
By using many features we can simply try to keep track of them and establish that a single bounding box containing two groups of feature points characterized by two different identities should be split and considered two different boxes

(and identities); this is particularly useful for occlusions, partial or total.
With the same reasoning, if we have a region that is not contained in any bounding box, the presence of a group of features with the same identity is a strong signal that the region actually contains a person, but that the detection phase failed in this particular case.
More concretely, we see each point as a tuple (x, y, id, age), while each box is described as $(x, y, w, h, list\_of\_points, id)$. The algorithm only stores feature points and an history of identities (for each seen identity, a list of tuples of the form $(frame\_number, x, y)$, and consists of an update step which receives as input the output of the detection phase, a list of bounding boxes, and returns a list of boxes with an identity.
The algorithm is described by the following parameters:

- infer threshold: if a cluster of points with the same identity exists, that identity is not already taken by a box, and the number of points is equal or larger than this threshold, then we consider the bounding box of these points as a new box, associating to it the identity held by this cluster of points
- max points per box: max number of points that a box can contain, excess feature points are deleted, this is simply done for a matter of computational resources; priority is given to younger points.
- max point age: points older than this are discarded.
- max lost dist: a box in need of an identity can be given an identity that was lost only if that identity predicted position has a distance lower than this parameter.
- max lost id age: identities that have been lost more than this number of steps ago are considered definitively lost, they cannot be reused.

It can be mainly divided in 3 phases:

- updating the positions of known points (optical flow algorithm) and finding new feature points (features finding algorithm).
- establishing the identity of provided boxes (from detection), establishing new boxes through clusters of points.
- identity conflicts resolution, giving identities to boxes which still are without one, history processing.

Some details referring to the algorithm (page 7):

- line 8: feature points that are not part of the group winning the majority vote are removed from the set of points of the box they were part of.
- line 10: currently only boxes that would have a limited area are considered (currently 200 pixels).
- line 11: feature detection is not required to be run every frame.
- line 18: out of n boxes having the same identity, only 1 of them will keep it, while the rest will have a (non) identity of -1.
- line 20: an identity is considered lost if it appeared recently (the last time it was actually seen was less than max_lost_id_age steps ago) but has not appeared this frame, meaning that there was no box (either from the provided

ones or the inferred ones) to which such identity was assigned. For each box without an identity we go through the set of lost identities and look for the identity minimizing distance and respecting a minimum distance threshold; if no such identity can be found, the box identity will still be none (-1). The position of the lost identity is somewhat inferred (linearly) by using its past history.

3.2 Ad hoc heuristics

Apart from parameters obviously being tied to this particular task, some measures specific to this scenario were adopted:

- the y coordinate of a bounding box should be between 50 and 650, this is because pedestrians are not going in those specific ranges, boxes not respecting this are deleted/ignored.
- a detected bounding box should at least have an area of 100, we use this as a way to reduce noise in the input for the tracking phase, given than people are usually bigger than that.
- our feature detector was sometime having trouble finding any features on boxes of people being on the extreme right of the scene, for this reason we assign as a feature the center point of such bounding boxes, when they have no feature points themselves.
- when predicting the possible position of a lost identity, we keep the y coordinate constant while increasing x according to the direction that identity was having; this was done because pedestrians maintain their horizontal direction even after bumping into another pedestrian, but it cannot be predicted if they are going to ”steer” upwards or downwards.

4 Achieved results and issues

This algorithm has provided results without much parameter search, however, the scope of this task is limited and it is still to see if it can maintain decent tracking before and after occlusions in different and more complex scenarios.
The source code for the algorithm, the video, and the application of this algorithm to said video can be found at [1] (code and output) and [2] (only output).

4.1 Detection

The count of the number of people present in the scene has been approximated using the number of detected figures in the detection phase; compared to the ground truth, our method achieves an average error per frame of 1.93.
While this is not bad, better results could probably be achieved by playing around with the parameters of the detection phase, like what morphological transformations to apply (and which kernels), or the parameters of MOG2; given our time restrictions, we preferred to spend more time in developing a solution to the tracking problem instead of perfecting the people counter.

Algorithm 1 Feature swarm update step

```
0: function UPDATE(boxes, infer_thres, max_points_per_box, max_point_age,
   max_lost_dist, max_lost_id_age)
   # update x and y coordinates of each point using an optical flow algorithm
   # points that are not found are discarded
1: $points \leftarrow find\_points\_with\_flow\_algorithm(points)$

   # map points with an identity to their box
2: for all $point\ in\ points \mid point.id \neq -1$ do
3:    $box \leftarrow find\_nearest\_containing\_box(point)$
4:    $box.points.append(point)$
5:    $point.age \leftarrow point.age + 1$
6: end for
   # the identity of each box is decided on a majority vote between its points
   # boxes with no points simply have identity still equal to -1
7: for all $box\ in\ boxes$ do
8:    $box.id \leftarrow decide\_on\_majority(box.points)$
9: end for

   # infer more boxes with clusters of points having the same identity
10: $boxes \leftarrow boxes + infer\_boxes(points, infer\_thres)$

   # detect more feature points with a feature detector
   # this can be done every n frames instead of always
11: $new\_points \leftarrow detect\_features()$
12: $points \leftarrow points + new\_points$

   # map points with no identity to their box
13: for all $point\ in\ points \mid point.id = -1$ do
14:    $box \leftarrow find\_nearest\_containing\_box(point)$
15:    $box.points.append(point)$
16:    $point.id \leftarrow= box.id$
17: end for

   # identities contended by more boxes are won by the box
   # having more points with that identity
18: $boxes \leftarrow decide\_contended\_identities(boxes)$

   # boxes with no identity get a new one
   # either an identity that was lost near this position
   # or a newly created one
19: for all $box\ in\ boxes \mid box.id = -1$ do
20:    $box.id \leftarrow find\_lost\_identity(box, max\_lost\_dist, max\_lost\_id\_age)$
21:    if $box.id = -1$ then
22:       $box.id \leftarrow seen\_identities$
23:       $seen\_identities \leftarrow seen\_identities + 1$
24:    end if
25:    for all $point\ in\ box.points$ do
26:       $point.id = box.id$
27:    end for
28: end for

   # keep track of where identities are/are going
   # used to decide which identities are lost and where
29: $update\_history(boxes)$
30: $points \leftarrow filter\_points(points, boxes, max\_points\_per\_box, max\_point\_age)$
   # return a list of boxes with their identity
31: return boxes
```

4.2 Tracking

The performance of the tracking algorithm depends on many components: a background subtractor, a contour finder, a feature detector, and an optical flow algorithm; because of this, it is hard to understand if a change in the tracking algorithm is generally better of if we are just over-fitting either to our problem or to the output of our detection phase.
We provide a numerical comparison between the real trajectory of three selected pedestrians (not selected by us) and what our algorithm reports; this comparison is naturally not enough to establish the goodness of our results, which can be better understood by observing how identities are maintained or lost during occlusions by having a look at the output video, which can be found at [2]
The selected pedestrians were id 10, 36, and 42; the average displacement between the ground truth and the tracking algorithm was 14.40, 23.88, 19.43 pixels respectively.
The provided algorithm, as expected, generally works well when pedestrians are allowed to simply walk from one direction to another without disruptions caused by other pedestrians. Occlusions are not always correctly processed, meaning that some identities are lost one way or another, be it because the optical flow algorithm is losing all feature points belonging to an identity or because of the wrongful assignment of a lost identity to a box.
The tracking of exiting and entering pedestrians is working fine, entering is being troubled by the fact that bounding boxes of entering pedestrians are detected "late", especially on the right side. We attribute this difficulty to the particular lighting on the extreme right of the scene.

5 Conclusions and future work

We provided an algorithm for tracking which can make use of any feature detector and optical flow algorithm, which parameters can be setup to run online or offline, depending on the number of maximum number of features one would want to store, how often one would want to run the feature detection algorithm, or depending on the desired trade-off between complexity and performance of the feature finding or optical flow algorithms of choice.
Identity and occlusions are being elaborated in a decent way, but there is always room for improvement, in particular, we believe these could be future leads to improve the performance of the algorithm:

– a better way to elaborate clusters of points when inferring boxes: currently we are just getting all points with the same id (considering only ids which are still not paired to a box), computing a bounding box for these points, and if the box is reasonably sized we accept it as a new bounding box with an identity. This could be done better by efficiently finding a way to exclude points that are clustered away from the others and would make the box too large.

– predicting the position of a lost identity: when an identity is lost, we keep updating its coordinates based on a certain number of past frames; a more elegant and noise aware way would be using a Kalman filter for each identity.
– direction aware identity matching: when an identity is contended we assign the identity to the box having more points with that id, whereas when having to give an identity to a box without one we look through our set of lost identities to find one minimizing distance, these approaches are not direction aware in the sense that identities that have been consistently moving right or left should not suddenly ”teleport” in the opposite direction.
– currently, we have a threshold that fixes the maximum amount of features points that should be in each box, having a ”number of features per pixel” would allow boxes to carry a quantity of features proportionate to their area.
– we are storing the history of identities and propagating them for a while if lost, why not do the same for features?

The input video, ground truth, code and output files can be found at [1], the output video can also be found at [2].

References

1. code repository, `https://github.com/fruttasecca/unitn_cv_2018_project`
2. output video, `https://drive.google.com/file/d/16V9JVor3TN2tOA3vNJHh652rP_DWgVjP/view`
3. Bradski, G.: The OpenCV Library. Dr. Dobb's Journal of Software Tools (2000)
4. Dai, J., Li, Y., He, K., Sun, J.: R-FCN: object detection via region-based fully convolutional networks. CoRR **abs/1605.06409** (2016), `http://arxiv.org/abs/1605.06409`
5. Kalman, R.E.: A New Approach to Linear Filtering and Prediction Problems. Transactions of the ASME Journal of Basic Engineering (82 (Series D)), 35–45 (1960), `http://www.cs.unc.edu/\~{}welch/kalman/media/pdf/Kalman1960.pdf`
6. Lucas, B.D., Kanade, T.: An iterative image registration technique with an application to stereo vision. In: In IJCAI81. pp. 674–679 (1981)
7. Shi, J., Tomasi, C.: Good features to track. Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition **600** (03 2000). https://doi.org/10.1109/CVPR.1994.323794
8. Suzuki, S., be, K.: Topological structural analysis of digitized binary images by border following. Computer Vision, Graphics, and Image Processing **30**(1), 32 – 46 (1985). https://doi.org/https://doi.org/10.1016/0734-189X(85)90016-7, `http://www.sciencedirect.com/science/article/pii/0734189X85900167`
9. Zivkovic, Z.: Improved adaptive gaussian mixture model for background subtraction. In: Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004. vol. 2, pp. 28–31 Vol.2 (Aug 2004). https://doi.org/10.1109/ICPR.2004.1333992
10. Zivkovic, Z., van der Heijden, F.: Efficient adaptive density estimation per image pixel for the task of background subtraction. Pattern Recognition Letters **27**(7), 773 – 780 (2006). https://doi.org/https://doi.org/10.1016/j.patrec.2005.11.005, `http://www.sciencedirect.com/science/article/pii/S0167865505003521`