Alma Mater Studiorum
University of Bologna

Artificial Intelligence - Computer Vision

Intrusion detection

Dicosola Alessandro [Matr. 935563]

1 Introduction

The project presented is trying to deal with object intrusion detection inside the field of view of a camera. Background difference, image segmentation and binary morphology operators are used in order to build a system that is able to find an intrusion as much as possible.

I will call **pipeline** the sequence of transformations and operations done on frames and therefore the project aimed to find a general pipeline to deal with object intrusion detection.

It was implemented a detector with different kind of background subtractor methods: static background, interpolated background and adaptive background.

The latter used for avoiding false positive.

2 Project structure

The project was structured in the following way for facilitating the development and testing:

- *core.Video.Video*: Video class deals with common operation referred to VideoCapture
- *core.Pipeline.Pipeline*: Pipeline class deals with the sequence of transformations (or operations) applied to a frame.
- *core.detectors.**: contains the implementation of the proposed detector
- *core.Contour.Contour*: Contour class for handling contour operation
- *common.py*: common operations such as compute difference of two images,compute the gradient of an image, draw contours and so on.
- *distances.**: contains distances used.

3 Distances

The $\|D\|_1$ norm and $\|D\|_\infty$ norm where preferred because less destructive. [Figure 1]

4 Detector

The detector implemented is a simple background subtractor, as suggested.

The methodology used for creating the model was simply driven by trials and errors: thanks to the framework implemented I was able to add and remove operations to the pipeline and see the output in order to assert the correctness of it; in this way I built the pipeline and defined the parameters of each operation.

With a test pipeline[1] was observed that the background subtractor (using the interpolated frame) was affected by artifact near the end of the video.

This happen with lower intensity when Gaussian blur is used. [Figure 2]

[1] BackgroundNoGaussian class

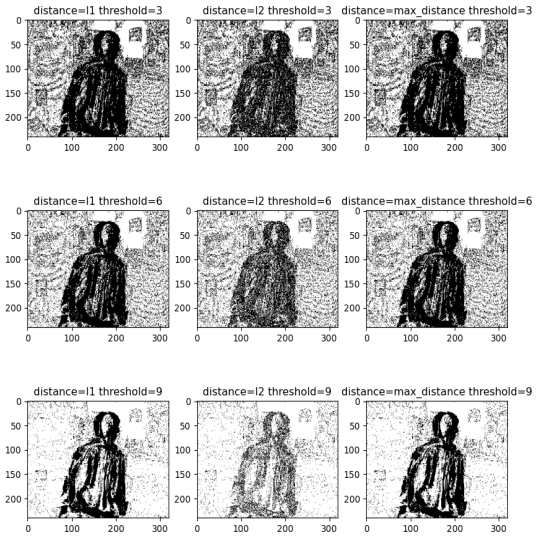


Figure 1: Threshold result on different distances

So in order to enhance the frame input a Gaussian blur is used for averaging pixels intensity without disrupting the image.

For removing noise, at the beginning, only opening with a filter of (9,9) was applied. After some inconclusive trials in which the noise was not completely deleted I've tried the median filter, the "salt and pepper filter" which is used for impulse noise (as the noisy pixels in a white-and-black image). As we can see it perform better. [Figure 3]

Moreover in order to remove additional noise the opening with a small filter is used afterwards.

Based on the kind of background used:

- **static**: it was used as background the **frame 254**. [Figure 4]
- **first**: the background was interpolated using the first 100 frames.
- **adaptive**: the background is computed each time as the weighted average summation with the previous frame. Used for creating a detector without false positive.

different parameters and pipelines are defined.

The parameters are the following:

kind	distance	threshold	alpha[2]
static	max_distance	30	-
first	max_distance	34	-
adaptive	l1	5	0.70

The pipelines are the following:

[2]alpha regulates the update speed (how fast the accumulator "forgets" about earlier images).

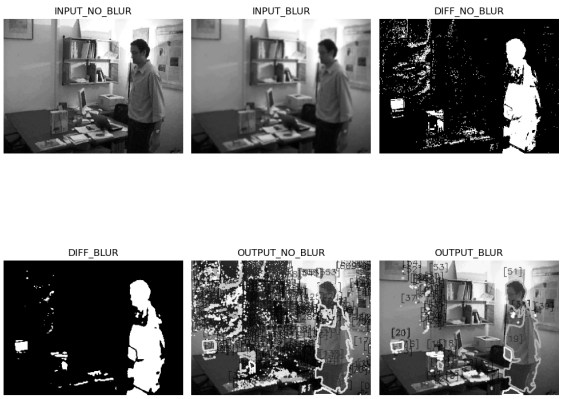


Figure 2: *Pipeline* of BackgroundNoGaussian

- With testing was found that 34 was better for kind=first. [Figure 5]
- Use alpha=0.70 in order to avoid to have detached blobs and so beeing able to reconstruct the blob with dilation. [Figure 6]

After the noise is removed, a sequence of binary morphology operators is applied.

For kind=[**static**,**first**]:

1. Close horizontally using a rectangle filter
2. Close vertically using a rectangle filter
3. Dilate for filling the void
4. Closing for filling the holes

For kind=**adaptive** a more aggressive pipeline is used:

1. Dilate for filling the void caused by median filter using a cross filter with a size = (20,20)
2. Two closing with an elliptical kernel with size (5,5) and (20,20) separated by an erode operation in order to reduce the boundaries of the blobs.

I have also tried to improve the contour detection using Canny edge detection algorithm (using the new detected edges for refining the already found edges) but was impossible to filter out noisy information from it using binary morphology operators. [Figure 7]

Figure 3: Median filter and opening (ellipse kernel) using a kernel of size (3,3) and (9,9)

5 Filtering and labelling

The information read about the contours are : area, perimeter, ratio, rectangularity, circularity, mean and standard deviation (in this case mean and standard deviation of the image inside the bounding box of the contour).

For **filtering** contours, first it is done a check on the rectangularity then on the area and perimeter: the idea was to pick big blobs and the ones with a considerable perimeter and area. This three information were chosen after reasoning on top of the logs produced by the detector.

I was thinking to use circularity, also, to filter out perfect circle but I have observed that sometimes blobs with high rectangularity have high circularity: the idea was to filter out isolated point that are enlarged by dilation.

The **labelling** is done checking area and ratio: big blobs with an height greater than three times the width are considered a person otherwise it's an object (other).

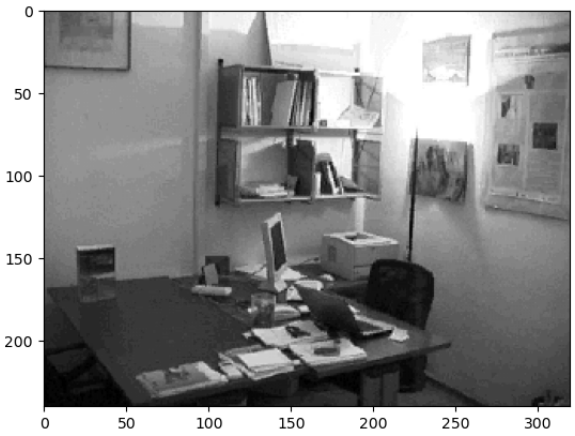
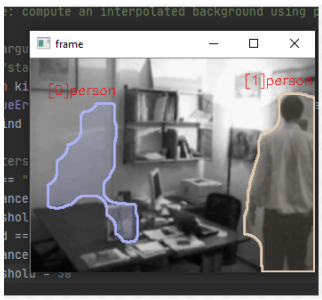
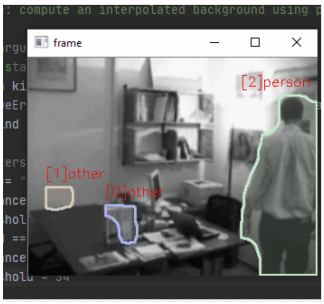


Figure 4: Background image.



(a) Using threshold=30



(b) Using threshold=34

Figure 5: Background subtractor with kind=first

(a) Using alpha=0.8 (b) Using alpha=0.7

Figure 6: Background subtractor with kind=adaptive

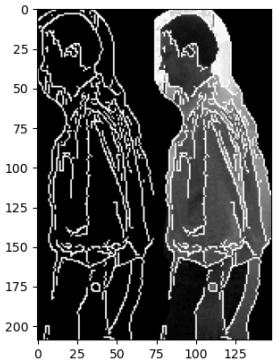


Figure 7: Canny edge detection algorithm applied to the inner area of the contour. Left: edges computed using Canny. Right: Edges overlapped on the image. Any binary morphology operator is going to remove the edges of the human figure.

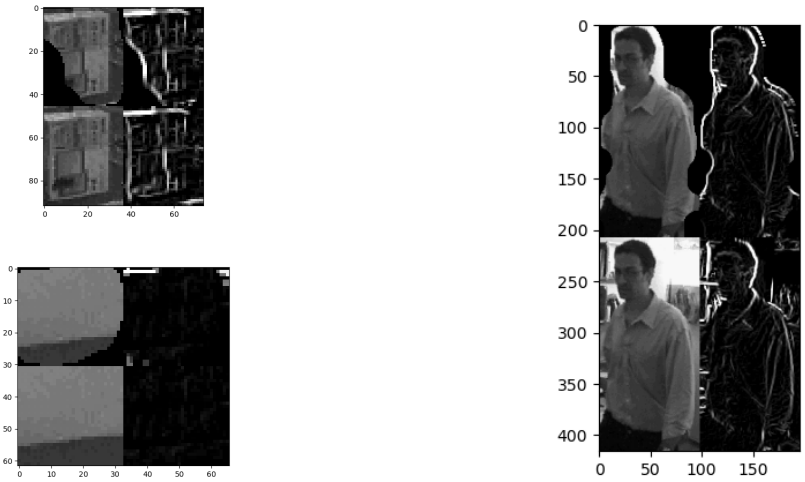


Figure 8: Gradient computed for each blob. Top: using the inner area. Bottom: using the bounding box.

6 Handling false positive

In order to detect false positives two solutions are proposed:

- Compute the standard deviation of the image, normalized over 255, after computing its gradient: I think that using the raw image could be misleading.

  The image used is the one inside the bounding box because of misleading edges found using the inner area. [Figure 8]

  In this case the contour segment is still in the image but is labeled as false positive. [Figure 9a]

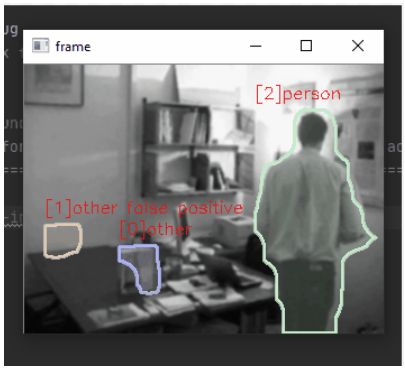
  Reasoning only on top of the contour is not possible because the segmentation doesn't return always the exact contour.

  The **threshold** used for detecting **false positive** is 0.1 chosen after reding the logs.

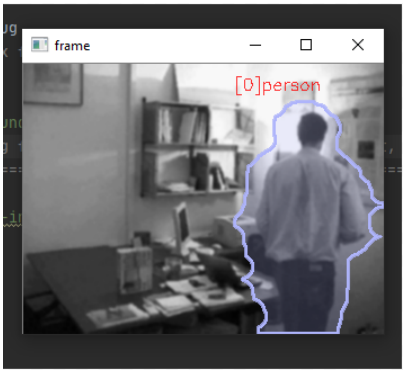
- Compute the background as an weighted sum with the previous frame at each time step. In this case the object is not detected; moreover also the other object, which is steady after beeing moved by the human, is ignored. [Figure 9b]

7 Conclusion

The performance, defined as how much the pipeline is able to capture the human silhouette, are similar between using a static background and an interpolated one since the use the same pipeline instead the adaptive one is worse but although it is able to avoid false positive. [Figure 10]



(a) Using kind=[static,first].



(b) Using kind=adaptive.

Figure 9: False positive detection

(a) Using kind=static

(b) Using kind=first

(c) Using kind=adaptive

Figure 10: Output of the frame 481.