A simple detector for crowd counting using OpenCV and Python

Martí Gelabert Gómez

November 29, 2022

Contents

1 Introduction

The objective of the assignment consists in **counting** the number of people that appear on a given set of images without an established way to approach the problem. In this case I have taken the decision of using the computer vision algorithms seen in class and not relay on deep learning. This is not because of the amount of data needed for training, because there are good datasets out there (like MS COCO), but because of the effort of building a good performing detector architecture. Using an already set framework such as YOLO or Mask-RCNN would be the way to accomplish the task, but I would think of it as "cheating", plus is not allowed. Therefore, this way, using the content seen in class, I will be more cautious about my decisions and It will be much easier to justify them.

In the following document we will be focusing on the process taken and the algorithms used, their implementation and their performance. Also, the link to the repository is here, and in the appendices sections will be the output images and some complementary ones.

2 General Procedure

To just get a rough idea of the construction of the algorithm, we will enumerate the steps taken in the following list :

1. Annotation of the data.
2. Import images as black and white.
3. Apply Adaptative Histogram equalization to the images.
4. Select our background image.
5. Subtract the background to the images using the background image selected.
6. Apply a thresholding algorithm to binarize the image.
7. Apply a dilation operation into the binarized images to expand the whites.
8. Use a contour algorithm to extract the different regions containing persons.
9. Obtain the bounding boxes from these regions.
10. Count them and compare the number of detections to the real quantity with a criterion established.

3 Algorithms and decisions took

In this section we will discuss the steps taken to generate the final program, the algorithms selected and the output we obtain from them.

3.1 Annotation

For the annotation of the dataset was used an online annotation tool and some basic criteria applied in order to decide how to label :

- The annotation would be points in the image.
- The annotations should be placed as near the head possible.
- If too many people where too close it may be annotated as just one person.
- In case of occlusion just annotate the most clear figure.
- Detections in the shore could be useful, so if the head was visibly recognizable it would be annotated.
- Some sections of the image will be masked, therefore those regions will not be annotated.

The ground truth is saved as a csv file and the program loads it when it is needed.

3.2 Background Subtraction

Background subtraction is a technique that allows to remove the background from the image, this way, the output of this technique will be an image with the foreground highlighted, it can be illustrated on the figure 1a.

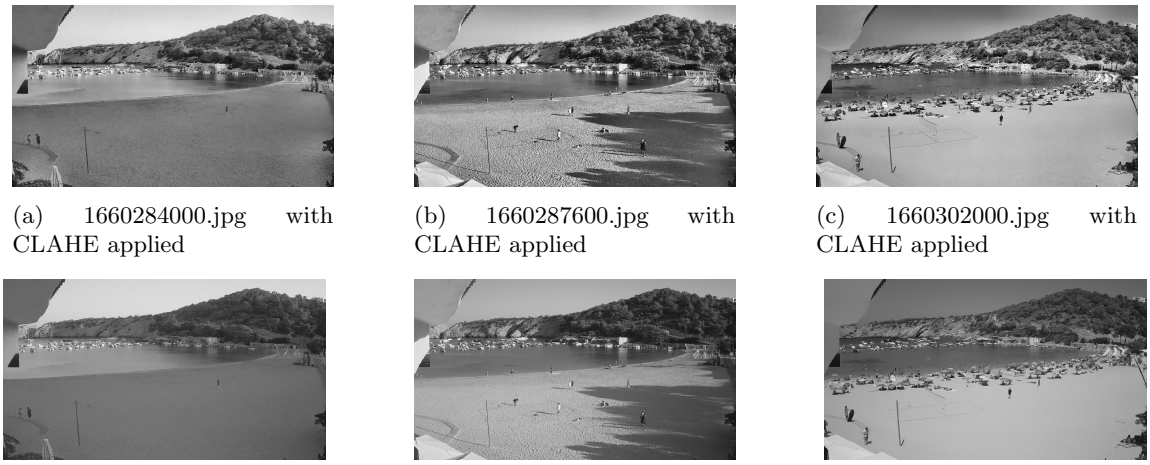
(a) Image with subtraction applied (b) Original 1660320000.jpg

Figure 1: Reference images

In the following sections will be explained how the preprocessing of this technique was prepared and for what they are used for. The sections are ordered by application.

CLAHE

First we have imported all the images in black and white integer values using `cv2.imread(img,0)`. Then for a better extraction, it has been applied an **algorithm of CLAHE** seen previously on the course to try to uniformize the illumination of the images. With a more uniform images and a higher contrast, we will be able to distinguish more precisely our foreground and artifact like **cast shadows** by objects on the images may have less impact on the **binarization process**.



(a) 1660284000.jpg with CLAHE applied (b) 1660287600.jpg with CLAHE applied (c) 1660302000.jpg with CLAHE applied

(d) 1660284000.jpg gray (e) 1660287600.jpg gray (f) 1660302000.jpg gray

Figure 2: Images from Gelabert folder where the first row presents a more uniform illumination compared with the second row images. (The images can be compared in the directory *./gen/equ/*)

With the resulting images, the higher contrast and a more similar mood between frames could facilitate us the task of find a more neutral background image. This way, the differences when applying the subtraction will be a sharper foreground in an ideal case.

Background Image

For the selection of our background image, in a starting point the **average image** was computed as the main way for accomplish foreground extraction, but the resulting image generally was really noisy and with ghosting effect very present on it, which decreased the performance of the algorithm as it can bee seen on the figure 9.

Even with a **Gaussian blur** applied to try to counter the "ghostly" figures, they are still really present on the image. Averaging is a good idea generally for noise reduction and in this case to try to obtain a neutral image where the background is **predominant**, but there is not enough images to obtain a more sharp image to really have a good background. Therefore, we **end up** using the image in the figure 8 (1660284000.jpg) with a Gaussian blur applied as our background, this way we obtain better results on the **binarization process**.

Gaussian Blur

With an image already selected for the image background, it is applied a Gaussian filtering with a kernel of size (15,15), this way we can obtain a more smooth **neutral image** that will allow the algorithm to contrast slightly better our foreground. With this technique applied, we try to obtain an image with the fewer details possible and just keep the core general ones. The ending size used for this filter was obtained by trial and error experimentation, and could be changed to get different behaviours.

Subtraction

Once we have selected our background and applied the preprocessing, we should subtract it to every image, this way, the result we end up with is the **foreground** (in this case the persons and some residuals) highlighted as we can see in the figure 4.

3.3 Binarization

Once the subtraction has been applied to all images, the resulting foreground will contain hopefully our persons. All those whitish areas should be further treated using a **binarization progress**, allowing a **solid division** between the background and our foreground. The process will return an image where the black areas and whitish areas are transformed into a pure binary one. In this assignment there has been some experimentation using **OTSU** and the basic binary thresholding function available on cv2, but for the sake of the performance, the last one has been ultimately selected.

The problem with using **OTSU** is that for each image it obtains a threshold selection automatically, and this should not be a problem, and sometimes could perform better than a fixed one. But for the samples tested, there are a lot of artifacts related to the shadow casting of some objects or the sand in some images, as it can be seen in the figure 10. The problems are located on the bottom, where there are an amount of shadows casts by the sand crests which can be a bit annoying for the next steps.

By using a set thresholding we are loosing some flexibility and a fixed margin could not be the wiser idea, but in this case using a **tight threshold** may be the best decision as may reduce the number of false detections considerably. In this case the values selected are set in the instruction `cv2.threshold(subtracted, 100, 255, cv2.THRESH_BINARY)`. The resulting image can be shown in figure 5.

Masking

For a better **filtrate** of our binary image, we will be using a binary mask to get rid of certain areas of the image where we can get non-persons in the foreground (i.e. the quay part) and those areas where there is too much information to get clear detections about it. We can see the last case in the umbrella section of the beach and the pavement area, where there are too many persons overlapping each others. The mask applied can be seen in the figure 6.

3.4 Dilation

Now, with the image binarized we will apply a **dilation** to the image. The dilation will expand the white areas on a binarized image, obtaining as results the highlighted areas in form of white chunks as it can be seen on figure 7. With these chunks the contour detection will obtain better results because they are more clear shapes. This could merge figures into one, so the parameters of the kernel size may be tinkered with to obtain the best performance possible.

With this approach we are probably obtaining areas too big that contain more than one person, this is done on purpose. The algorithm values more loose detection precision to trade it by at least getting areas where

people tend to concentrate. This way, we end up with at least an idea of where people could possibly be on our the image.

3.5 Find contours

Once a dilation image is applied, there should be a process of contour detection. Here we use the images obtained in the previous technique over the function that openCV provides, `cv2.findContours()`. From this function, we can extract the bounding boxes that contain each of the object contours. In the figure 3 appears the image with some bounding boxes applied.

4 Results and Analysis

For convenience, we will quantify our detection accuracy using the following assumptions:

- The detection will only be a **true positive** when it contains at least a label.
- If detection contains more than one person it will count as **only one detection**.
- In the case we would have a massive region, we will discard that detection.
- For checking the dimensionality of the bounding box, we would assume that width or height higher than a third of the image will not be acceptable.
- Regions minuscule will also be discarded.
- And some labels of the ground truth could be double checked.

We are not restrictive enough to lose a lot of information. Allowing more **flexibility** in order to have more valid detections let us obtain a better performance overall. Also, some cases are unlikely to happen really often. In the tables 1 and 2 is shown how the algorithm performs.

Results

As we can see, the tables 1 and 2 show a **low scoring**, having a clear unbalance in between the actual number of persons in each image and the correct detections. But of course the general problem we will still have is the quantity of false positives we end up having in some images. Even the **mean squared error** with a value of 341.666667 tells how this algorithm is having a huge error, also we are predicting with a really soft constrains and allowing a lot of error on our detections. Although this is done entirely on purpose and in the practice this has turned out to be more beneficial.

Table 1: Performance metrics using the proposed algorithm, the column of *gt* represent the number of labels on each image, *detected* represent the number detections found in the image, and the column *matched* represent the number of detections where at least contained one label of our ground truth

files	precission	recall	f1 score	gt	detected	matched
1660309200.jpg	0.336	0.444	0.383	90	119	40
1660302000.jpg	0.292	0.369	0.326	103	130	38
1660294800.jpg	0.333	0.458	0.386	72	99	33
1660320000.jpg	0.363	0.363	0.363	135	135	49
1660287600.jpg	0.200	0.471	0.281	17	40	8
1660298400.jpg	0.347	0.311	0.328	106	95	33
1660305600.jpg	0.360	0.396	0.377	101	111	40
1660316400.jpg	0.358	0.345	0.352	139	134	48
1660291200.jpg	0.353	0.346	0.350	52	51	18

Table 2: Performance metrics overall using the proposed algorithm

Metric	Scoring
MSE	341.667
Macro-average precision	0.327
Macro-average recall	0.389
Macro-average F1	0.349

Figure 3: Output without formatting. As it can be seen there are some big detections due to contours detecting multiple forms as one shape and some inner detections

Should be considered that the type of images we got and the placement of the ground truth can make a huge difference on our outcome, so this algorithm has been tuned to try to be the more general possible **only** with the images provided in the corresponding directory.

5 Is a different approach capable of improving our results?

For some of the test that were executed for this assignment, generally the effort was not useful. Ending up in situations where a precise modification to the parameters gave as output excellent results only in a subset of the images. Usually the over-tunning of some function parameters made really inconsistent outputs and provoked a bad performance. In the following subsections it will be discussed some other paths that could be considered to accomplish this task.

Color Spacing

For example, one of the approaches taken for background removal, was the use of colored images in the color space of LAB instead of gray scale. On the paper one may think that the use of color will improve our performance, because of the extra information we are just acquired from free. But in this case the process to obtain our foreground sections were in general less precise, giving as result a lot more objects as foreground, which is not ideal for our case, been a lot of those object just umbrellas and cast shadows.

The problem with shadows

In general, the sample images follow a consisting landscape without heavy cast shadows on the sand, but it is noticeable in some of them, and usually they will give you more than one headache.

We could try to counter some cases using a sharpening processes or even more filtering (i.e. median filtering), to avoid certain annoyance for shadows, but doesn't mean that it will be worth it to apply. We could have the situation again where is really useful for 1 case, but then the rest of images just got obliterated just for applying those transformations.

Annotation with bounding boxes

For a reason of time, the annotation was given only by dots on the image, but it would be a good idea to use bounding boxes and utilize more "sophisticated" methods as **intersection over union**, to allow us the execution of other metrics to improve our way of evaluating our results.

6 Conclusions

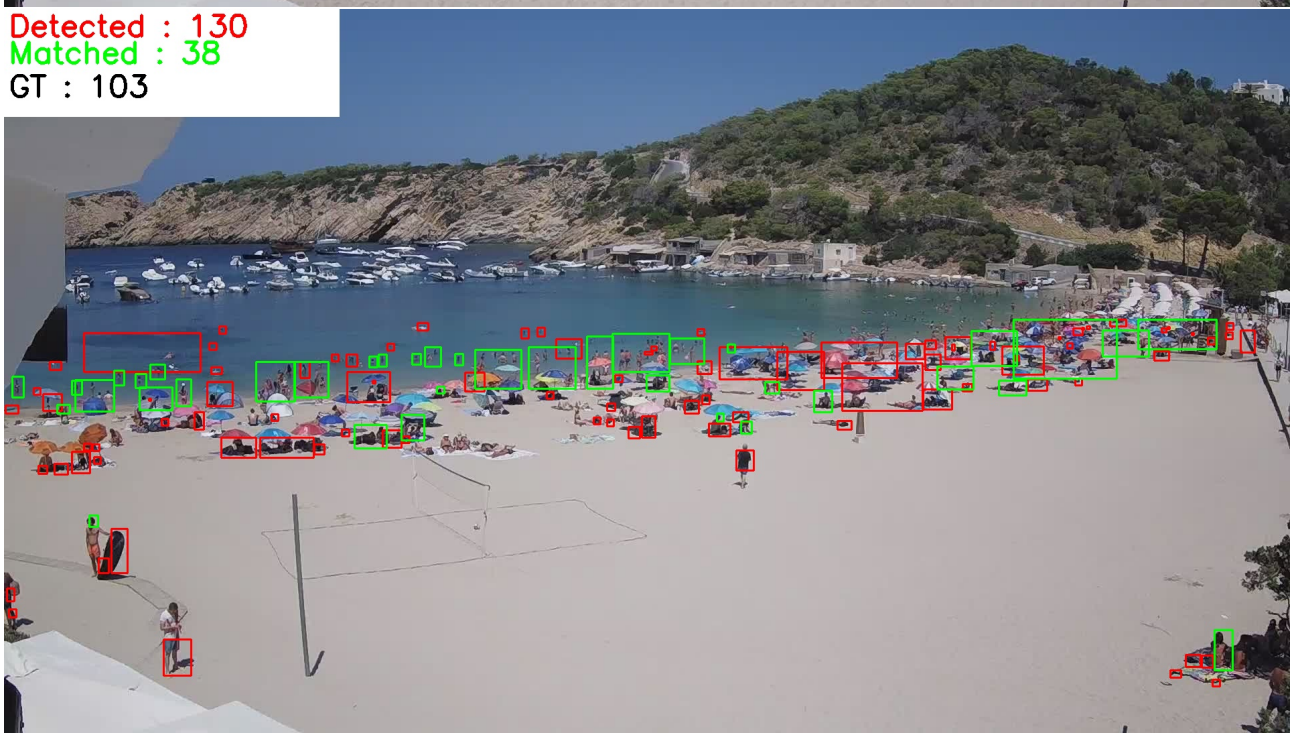
As we could be expecting, the performance is not good compared to a neural network or anything in general. There are some cases where we can't be completely sure about classifying incorrectly our detection because of how the ground truth is placed. Some detections are actually correct but slightly shifted and without a manual intervention it is not capable to compute that as good. In general, the amount of false positives is really present on our results, and overall concentrates in areas where there is a brusque lighting change like **cast shadows** or where there is a lot of **information compacted** like in the shores and further places of the landscape.

The use of **background subtraction** to try to extract completely our foreground is a tricky task, and it can be performed following different strategies, for example, computing an averaged image. In this case with just the images from the 'Gelabert' folder the output generated was not clear enough (even with some processing) to accomplish the objective, and in the end just using the empties image was the one given the best performance. We can still spice up our work if we start working with more **complex color spaces**, like LAB, or using more **complex post-processing conditions** for the sake of improving our results.

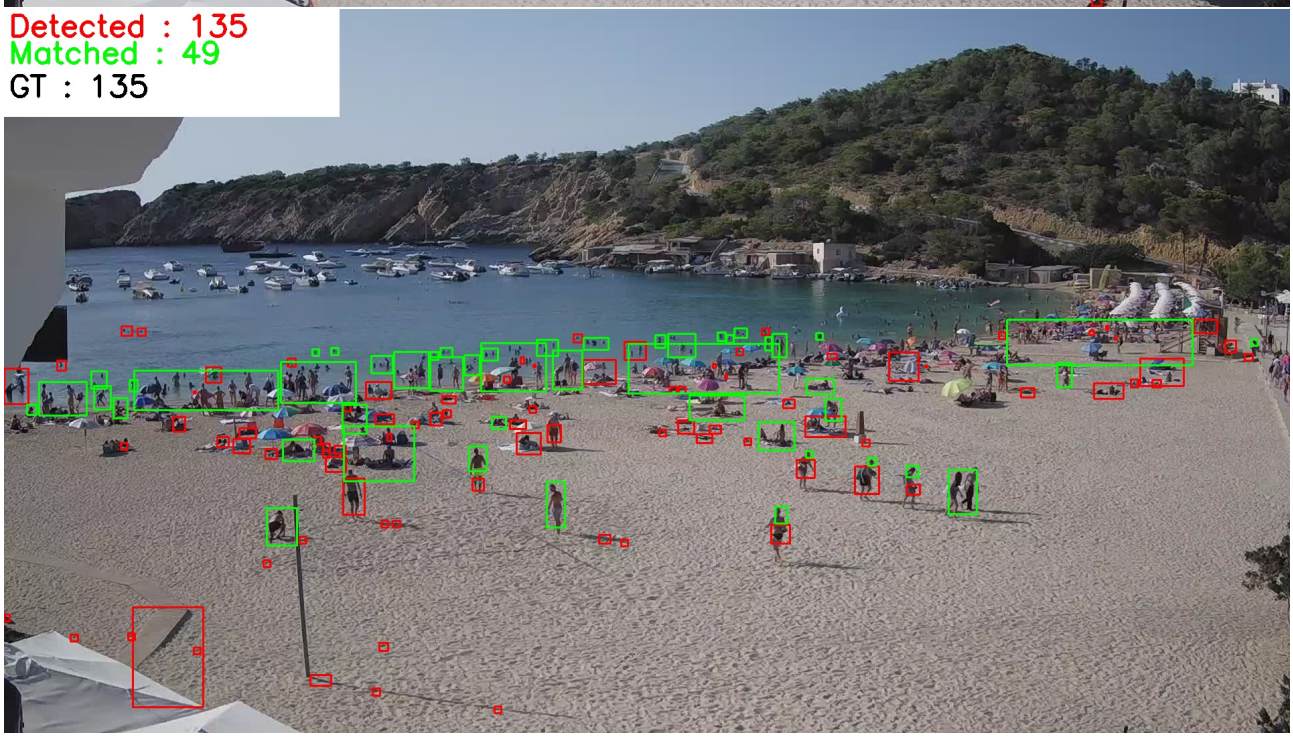
Even we could try to implement a linear regression based of pixel counting for trying to estimate the number of detections in a big region. But in some cases, working with simpler ideas may be the way to go.

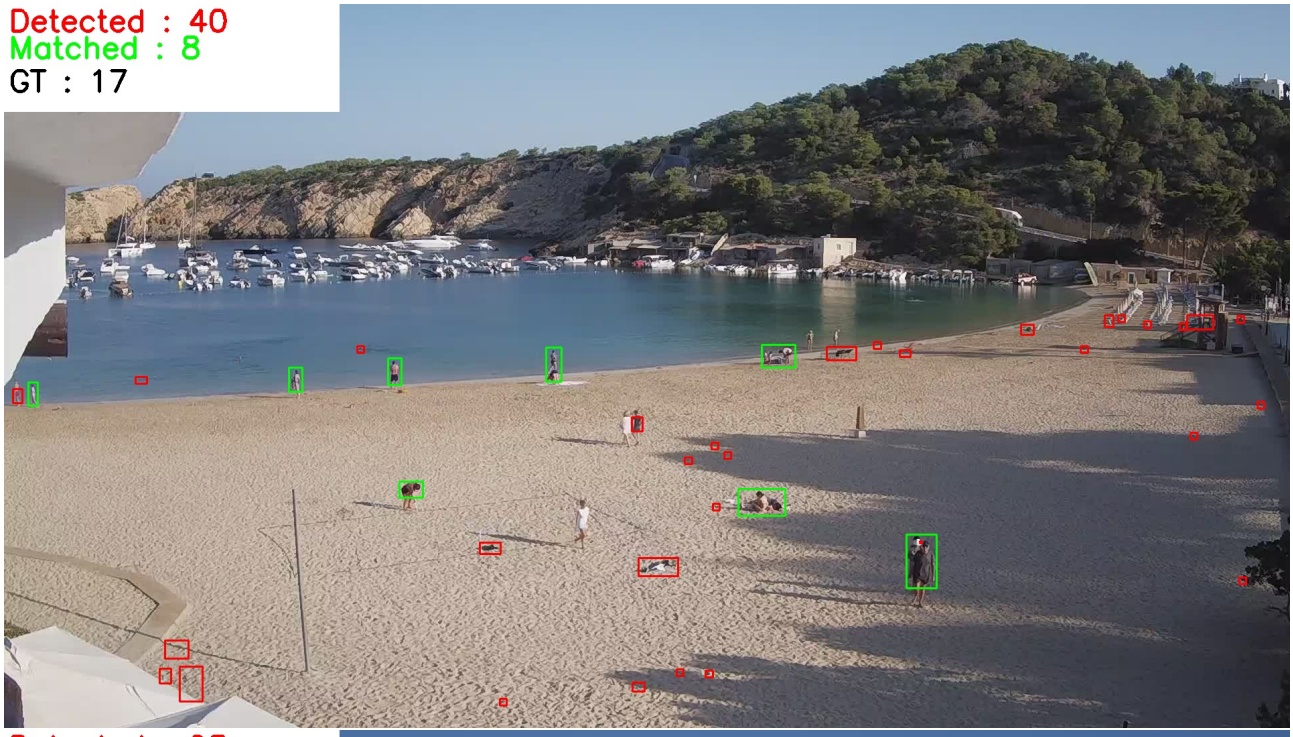
Of course, we have to take in count how flexible the annotation and the accuracy measurements where, but as it can be shown on the outputs, we are capable of at least detecting areas where people concentrates, and even single persons in some cases. Therefore, for the use of these techniques it should be enough.

A Output



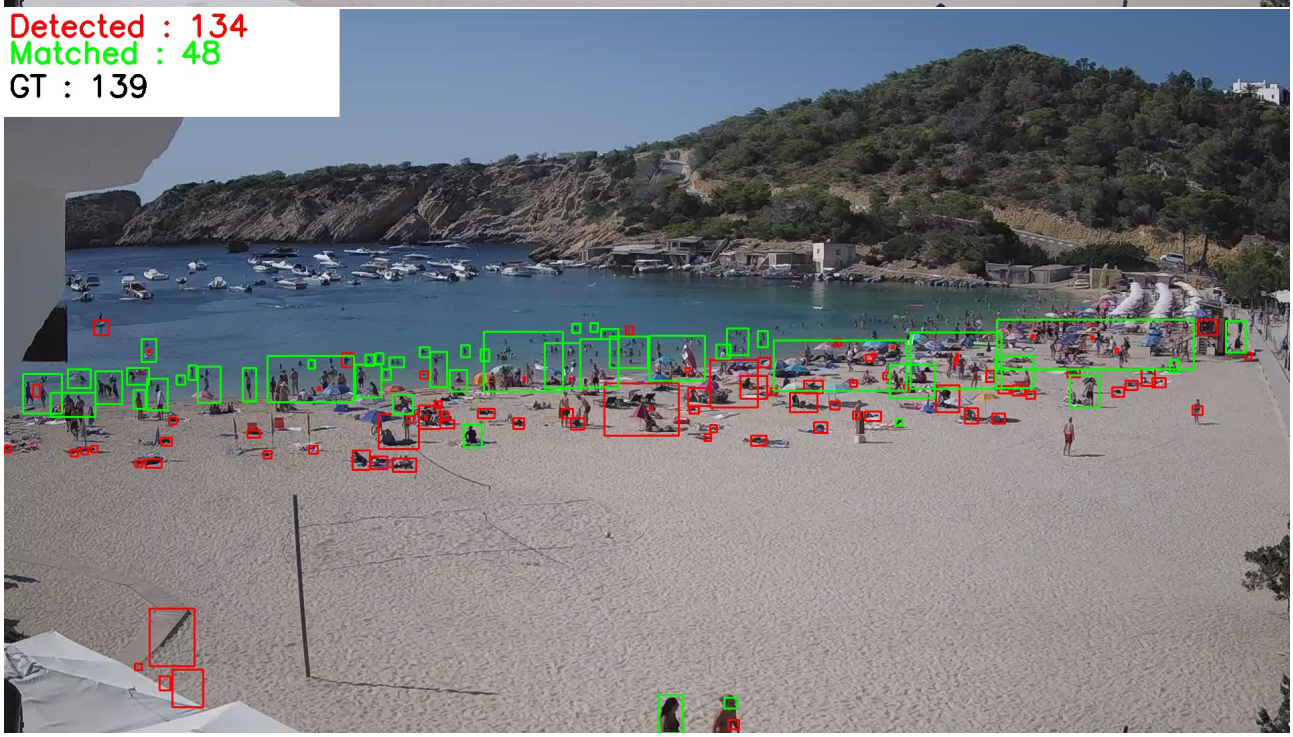

Detected : 99
Matched : 33
GT : 72

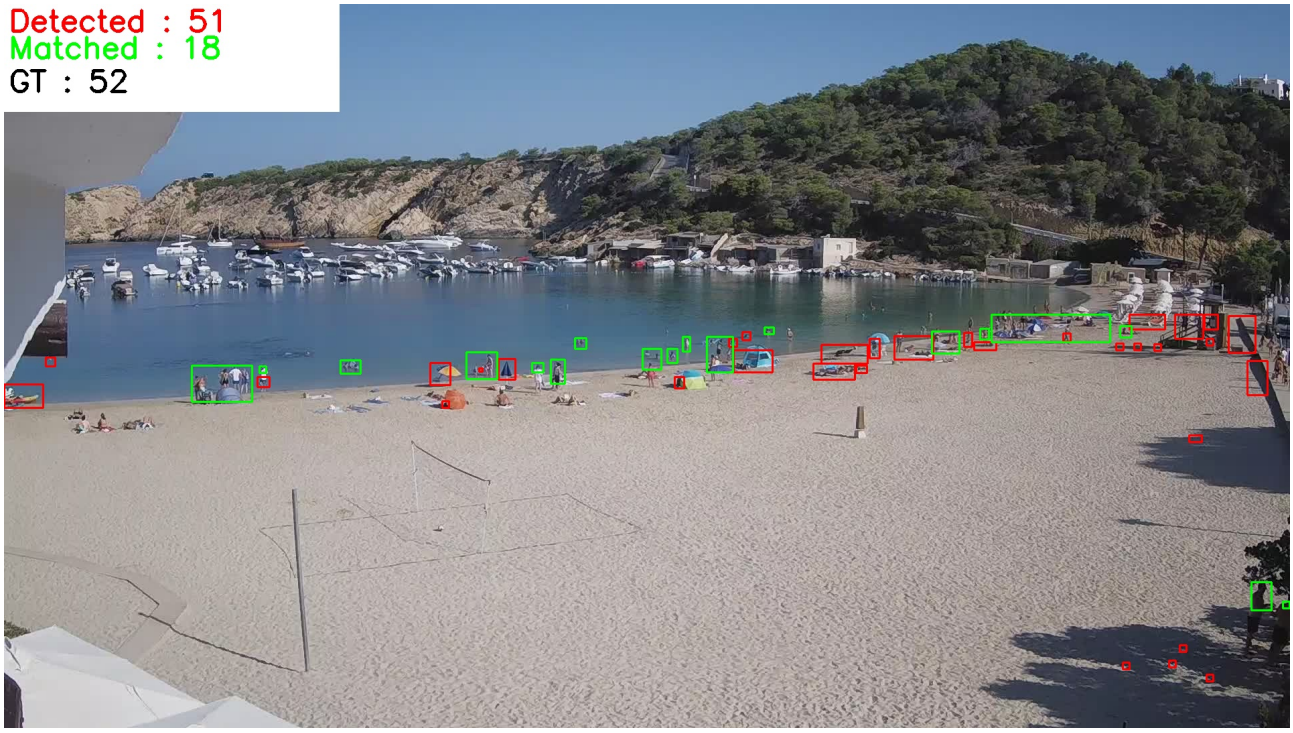

Detected : 135
Matched : 49
GT : 135


Detected : 40
Matched : 8
GT : 17


Detected : 95
Matched : 33
GT : 106


Detected : 111
Matched : 40
GT : 101


Detected : 134
Matched : 48
GT : 139



B Images related to the basic Algorithm

Figure 4: Image obtained by applying cv2.subtract(background, image)

Figure 5: Binarized image using a fix thresholding method

Figure 6: Mask image

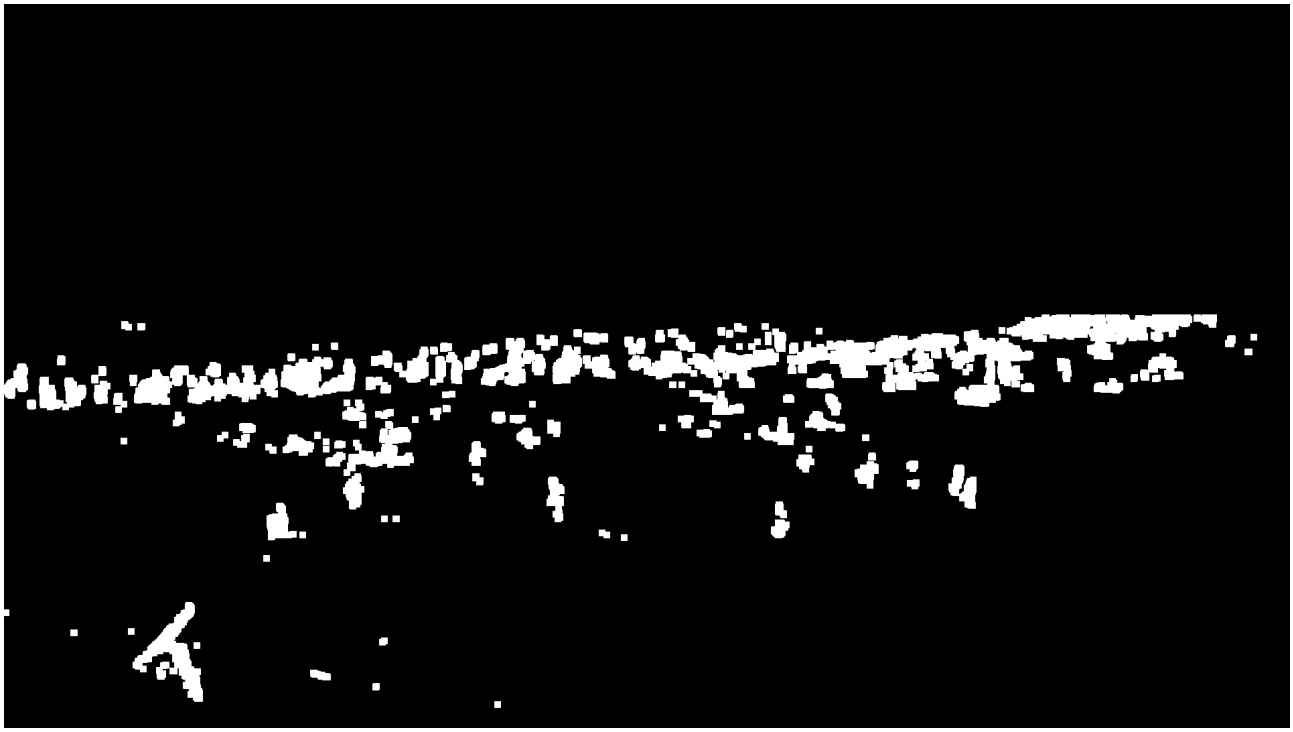


Figure 7: Dilation applied to the binarized image

Figure 8: Background Image with Gaussian blur applied

C Complementary Images

Figure 9: Average image with a gaussian blur applied with a kernel of (11,11)

Figure 10: Binarized image using OTSU