Computer Vision Assignment 2 - Report

Samuel Petit - 17333946 - petits@tcd.ie

1. Declaration of Work

This is my own work. Any material taken from other sources has been fully referenced in the text of the work. I have not worked with anyone else on this assignment and have not shown or provided my work to any other person. I have also not made my work accessible to others, both physically and electronically. All sources used in the preparation of this work have been listed in the Bibliography. I have read the statement on plagiarism in the College Calendar and understand that plagiarism is an offence that may result in penalties up to and including expulsion from the University.

Samuel Petit - 15th November 2020

a. Bibliography

The only material that was used in this assignment include code from the TIPS library, the opencv documentation and lecture slides.

2. Theory (with examples)

Before explaining in detail how I figure out if a frame is obscured or not and if a postbox has mail, I will first explain the theory underlying both solutions. My approach uses some form of selective running average method. Here are the steps my algorithm takes:

1. Reduce the brightness of the first frame and set it as our background.
    a. The brightness reduction is done as the video's exposure is quite high, reducing it helps for the actions to follow.
2. Iterating over each frame of the video, start by processing the current frame.
    a. Processing here consists of reducing the brightness by 100 (the exact same amount as above), applying gaussian noise and gaussian blur. Noise and blur introduce some randomness which yielded better results when it came to extracting a background and foreground image.
3. Finding the foreground mask, by computing the difference between the background and the currently processed frame.
4. Summing the foreground channels together, and performing otsu thresholding to the result.
    a. I decided to use otsu threshold as I have found it to yield better results than using some constant thresholding in this situation.
5. Running a Close operation on the thresholded image
6. Updating the running average foreground mask by accumulating the channels of the current processed image and the first frame background image, using a learning rate of 0.1. The image is then inverted.

7. We can then use the mask from the previous step to copy values from the original current frame into a foreground image.
8. The final step is to compute a histogram of the foreground image obtained from the previous step.
    a. The histogram uses 64 bins, a range from 0 to 255 and is normalised.

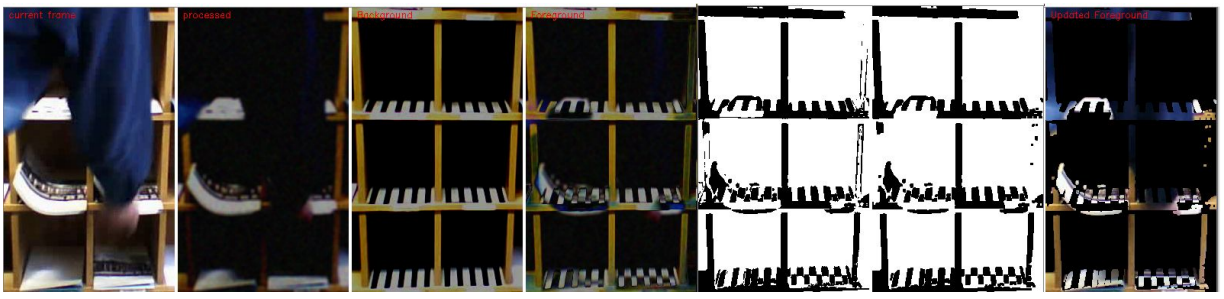
Here are some examples of the processing these steps in order for some frames:

- Frame 4:

- Frame 19:

- Frame 41

- Frame 52 (obstructed view)

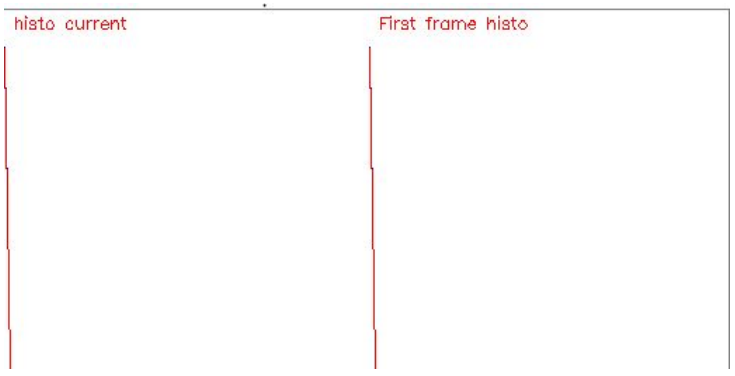


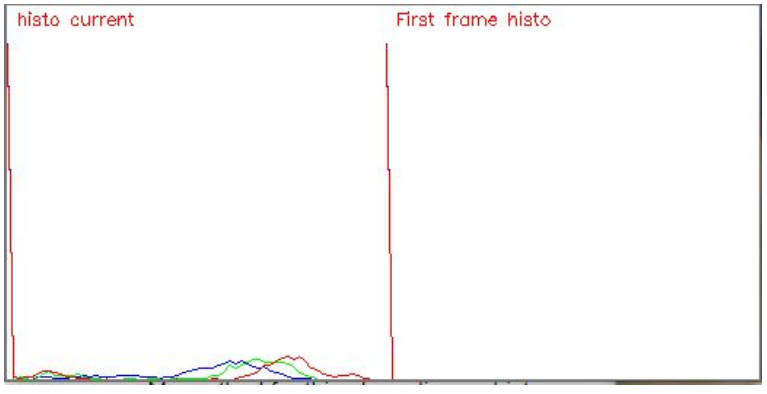
Figuring out if a frame is obscured

Once we have the components from above, that is a background, foreground image and a histogram that are relatively clean. We can use those components to figure out if the current frame is obscured. To do so:

1. Compute and store the histogram of the first frame for future uses
2. Compare the histogram for the current frame with the histogram from the first frame. Using Chi-Square's comparison method over the image's 3 channels.
3. When the comparison sum is over 3, then I determine that the view is obscured.
    a. I also set the next frame after an obstructed period to be obstructed as, while my algorithm seems to be ready and the view is no longer obstructed. The ground truth values have at least 1 or sometimes more frames set as obstructed. I've done this purely to reduce false negatives, in a world outside of this assignment I would have kept my algorithm as is.

We've previously seen examples of the processing for obscured and non-obscured views, let's now look at how our normalised histograms compare for both:

- Frame 4:
    - 

- Frame 16 (an obscured view)
    - 

We notice a clear difference in histograms during obscured and non obscured view so we can fairly reliably determine if a view is obscured. The colours are quite stable when the view is non obstructed thus we can fairly easily determine any good amount of variation in the histogram to be an obstructed view.

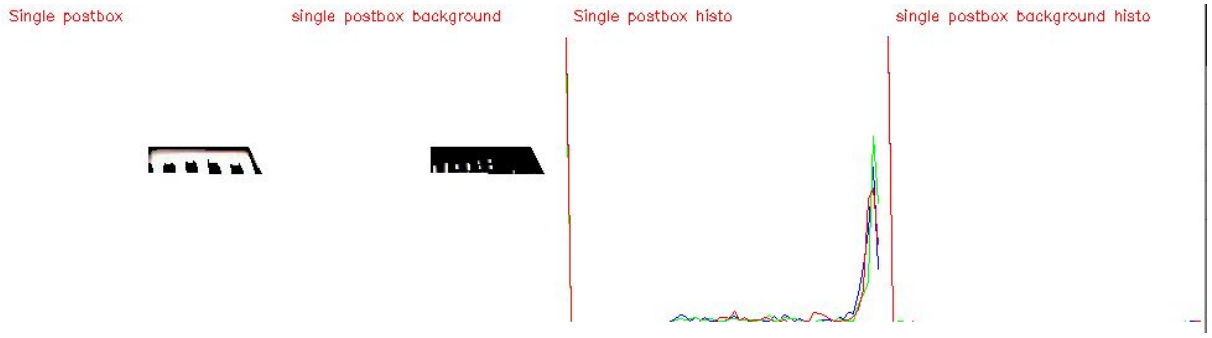
How to know if there is a post in a postbox

My method for this also relies on histogram comparisons however there are some more operations involved. The high level overview of my algorithm is:

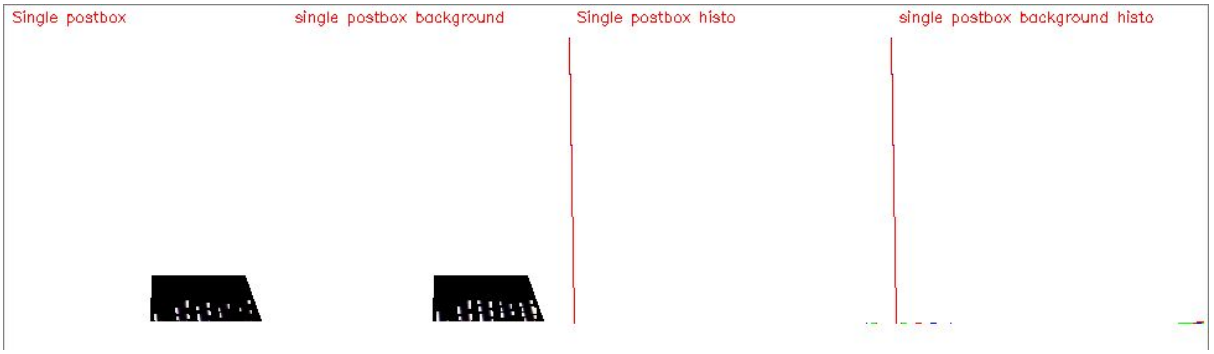
1. Iterate through the number of postboxes
2. For the current postbox, get the set of its 4 corners (or 4 points). Fill a mask using those points as a single shape.
3. Copy both the foreground and background images using the mask such as to only have non-black pixels only be within a single post box.
4. Compute histograms for both images. Then do a histogram comparison using the chi-square method.
5. If the matching sum is greater than 1.8 (seemed to work best for this context) we can set the current postbox as having mail.

Here are some examples of the processing and histogram for some frames:

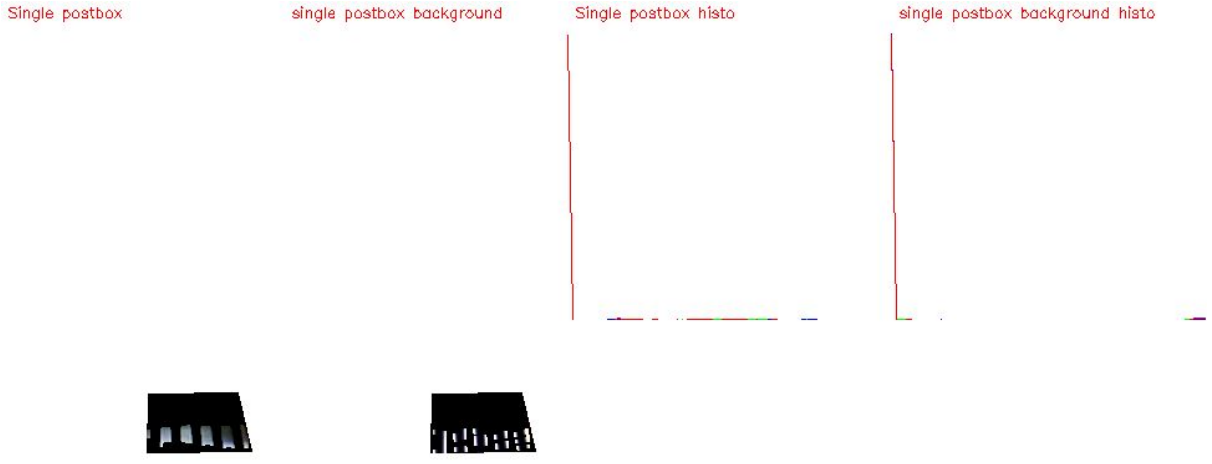
- Frame 12 (mail in box 2) → showing box 2



- Frame 14 (mail in box 2) → showing box 4



- Frame 74 (mail in boxes 3 4) → showing box 8



We notice that while it seems to work quite well like in the first case, the 3rd example shows a situation where there is no mail yet we observe some noise in the histogram which may lead to a false positive.

3. Solution criticism

I have a few concerns on the approach that I took and ideas on how to improve them.
The first that comes to mind is the fact that my approach heavily relies on histogram comparison, while the chi-square comparison method seems to work best in this context (out of the other comparison methods of opencv), the range of the comparison result values can vary quite a bit for positive outcomes (postbox has mail or view is obstructed) and thus sometimes it is quite hard to find an appropriate threshold since my solution as it is will include some amount of false positives.

- An Idea to solve this was to potentially change the histogram comparison method to some custom comparison system which would not be affected by noise as much. More generally though some of the noise could be reduced by changing the overall approach, I will discuss that idea below.

In terms of how general my solution is, the provided video is quite bright, I countered that by reducing the brightness as part of my processing however that will not fit a general use case. In fact another similar video might be shot later in the day and thus require a brightness increase! This could be solved by applying a histogram equalisation to the video frames and then making the algorithm work properly with an equalised image, which would work for an ideal solution.

Another concern is that my solution uses the first frame of the video as part of its algorithm and relies on it being representative of the background. For a more general solution I could have used a method to extract the foreground which does not rely on this assumption. In fact, even if the first frame were to be varying in brightness (as some automatic cameras calibrate) or were to be obstructed this approach would not work (or would have to use a different frame as its basis). I did attempt to use other methods which did not rely on a static background image however they proved to not be as accurate as the method I've chosen to use.

The foreground computation, while resulting in mostly usable outputs, does not seem to be ideal in our situation as sometimes adding a mail inside one of the boxes will change the shade and thus some of the colours of post boxe(s) below it. Resulting in part of the postbox being shown as the foreground and while the otsu thresholding and closing operation attempt to solve this, my algorithm still sees some false positives. This could be solved with multiple methods, one could be to run shadow detection, or perhaps attempt to remove the shadow created through histogram equalisation (once again), another idea could be to run edge or region detection on the postbox and making sure the shape fits to some degree a rectangular shape.

Finally, I've mentioned that perhaps doing foreground and background extraction for this problem is perhaps not ideal. With some thoughts I believe it would have been possible, likely easier and I believe more precise to take an edge detection or regions approach. Simply by running k means clustering such as to simplify our image. Then attempting to find

regions within a postbox, then if a region roughly matches a rectangular shape and its area is large enough, we can determine that there is mail in that box.

4. Performance metrics

Figuring out if a frame is obscured

This confusion matrix was quite straightforward to obtain, when the ground truth for the current frame is set as obscured, then if my algorithm also has the frame obscured I increase the true positive, when it has the frame set as non obscured I increase the false negative.
On the other hand, when the ground truth is not obscured and my algorithm is set as not obscured too, I increase the true negative, when it has the current frame as obscured I increase the false positive.

True Positive	52
True Negative	39
False Positive	0
False Negative	4

We notice that the detection is quite accurate for my algorithm with only 4 false negatives. Note from earlier that my approach allows for detection on some frames that the ground truth has set as obscured when it detects correctly the postbox mails, thus we see some amount of false negatives here (which has been reduced by marking the next frame after an obscured period as obscured such as to comply better with the ground truth).

Precision = TP / (TP + FP) = 52 / (52 + 0) = **1**
Recall = TP / (TP + FN) = 52 / (52 + 4) = **0.929**
F1 = 2 * (Precision * Recall) / (Precision + Recall) = 1.858 / 1.929 = **0.963**

Figuring out if postboxes have mail

The confusion matrix for this was a bit more complicated to figure out. It is only updated when my algorithm finds that the current frame is not obstructed. Then for each postbox we update the counts, that is:

- For all 6 postbox, increment the true positive when my program and the ground truth find a mail in a specific box. When my program finds a mail in a specific box that is not found in the ground truth the false positive is incremented. On the other hand, when there is no mail detected and the ground truth finds a mail, the false negative is incremented and similarly, when the ground truth does not have a mail the true negative is incremented.

The only other thing worth mentioning here is when the ground truth finds the view is obstructed and my algorithm still attempts to find postboxes, in this case I do not update this confusion matrix as it is already taken into consideration by the other confusion matrix

above. I have mentioned this case in the previous section so I won't go into more details here.

True Positive	101
True Negative	114
False Positive	19
False Negative	0

Precision = TP / (TP + FP) = 101 / (101 + 19) = **0.841**
Recall = TP / (TP + FN) = 101 / (101 + 0) = **1**
F1 = 2 * (Precision * Recall) / (Precision + Recall) = 2 * 0.841 / 1.841 = **0.914**

We notice that our values are slightly lower here than for the obstructed view, that is due to the false positives being relatively high (19). These false positives come from the issues I have outlined in the previous section.

5. Discussion of the results

There is one problem that I would like to mention regarding the computation of the postboxes having mail's confusion matrix, I have made a decision not to count the ground truth stating a single frame as obstructed and my algorithm thinking otherwise, but it could have been treated differently:

- We could have counted each postbox having or not having a mail as a false positive / false negative. This would have heavily affected my metrics. I chose not to take this approach as the counts could have been correct but given the ground truth was set as obscured there was no way to know.
- We could have also counted it as a single false positive for trying to attempt to count post boxes though that could have made our metrics misleading.

I believe that it made most sense to ignore this case such as to keep my metrics accurate as to what it tries to represent.
In an ideal world we could perhaps change the ground truth to enable some of these obstructed frames as being visible as my algorithm seems to be able to tell quite well for some frames which were counted as obstructed.

We notice from these metrics that my solution for determining an obstructed view is quite accurate, we find that an interesting difference in the recall and precision metrics which the F1 metric takes both into account: 0.963 for an F1 value is quite high especially taking into account the reservations I have made previously.

We notice a much bigger difference in precision and recall in the second table which is for determining if postboxes have mail. There is a difference of 1 - 0.841 = 0.159 which is quite considerable. This makes sense because the recall does not take the false positives into account while the precision does, giving us imbalanced results. We look at the F1 metrics which combine both and we find a value of 0.914. Taking into account the reservations here too I believe that is quite a high value where it is accurate for most postboxes on most

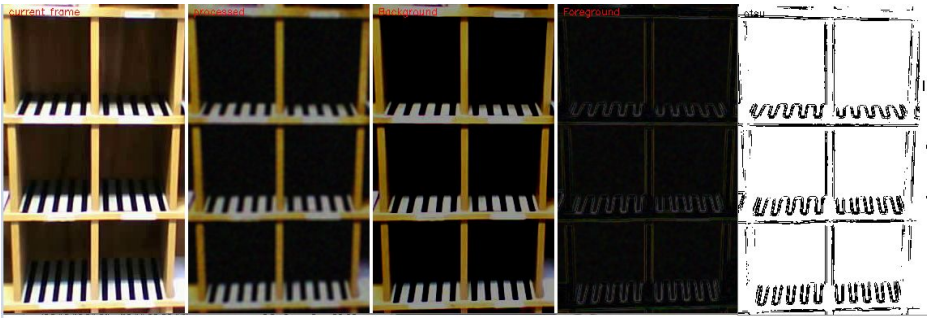
frames. A count for false positives of 19 is quite high and worth noting though, perhaps using some of the methods discussed above we could try to reduce that amount.

- There is one final detail worth noting here, once again that issue of ignoring frames that the ground truth has marked as obstructed but my algorithm has not had the potential to heavily impact this metric, once again this was addressed earlier in this section.

Perhaps by including an additional check by using shape detection on the postbox we could have gotten an F1 value closer to 1, however I am happy with my algorithm's performance nonetheless.

6. Discussion on how to find to post boxes locations

Given that the locations had not been given, I could have implemented some form of Chamfer matching as part of my algorithm, given that I already do some processing to compute the foreground of an image, I can do these exact steps as explained in the first part of this report and as implemented in the provided code, we obtain the following for the first frame:



Note that I have stopped the processing after executing the Otsu thresholding and have not executed a closing operation on the image, we could instead execute an open operation such as to reinforce our edges.
Given that binary image, I can do the following steps to finish the detection:

- Use the output from the Otsu thresholding (which we have ran an open operation on) as a mask such as to copy pixels from the background image into a new image, but only the pixels matching the mask. We obtain that way a color foreground image which includes the outline and structure of the postboxes.
- We can then run Canny edge detection on that image such as to obtain a set of edges.
- Invert the canny edge image and threshold it if required.
- Compute the distances of the inverted edge image using opencv's "distanceTransform".
- To obtain a model, we can draw a rectangle shape
    - Something like (of course it would need to be similar to the shape & size of a single postbox, this is just an example).

- 
- We can then convert the model to grayscale, threshold it such as to obtain a binary image.
- We can do a chamfer matching using both the chamfer (distances) image and the binary (edges) model.
    - We can do this by extracting the model points from the model. To do so, iterate over each pixel of the image and for each non-empty pixel (which has an edge), keep track of it as a point inside a vector (or list).
    - Then, try the model in all possible positions. This would return a matching image which represents how much a region matched the model.
- We can then use the matching values to extract a number of local minimas, in our case 6 such as to find the 6 locations of the postboxes.
    - Note that this assumes we know the number of post boxes in advance.
- From this point, we can extract the position of the postboxes, thanks to using the position of the local minimas as the center of the position of the model and using the model's size such as to extract the positions of each postbox.
    - Note that this assumes that the size of the postboxes are roughly the same. If the image is distorted on the point of view and makes boxes differently sized, we may apply a simple affine geometric transformation at the start such as to obtain similar sized postboxes.
    - If the postboxes do not have the same size we could use 2 or more different models to fit the different sizes.