Computer Vision Assignment 2 - Post Detection

Ernests Kuznecovs - 17332791

November 23, 2020

Solution Steps

The solution for detecting if the postboxes are obscured, and the detection of post in the postbox, both use almost the exact same computer vision pipeline. As an overview they are:

- Select the corners of the each of the sections to be processed.
- For each section:
- Transform the four points into rectangles to ensure that relevant edges are most vertical.
- Convert the frame into grayscale followed by binary threshold (threhsold value chosen to ensure the contrast between the box's side panel and box's background is captured)
- Sobel edge detection, taking only the horizontal derivative, and thresholding again (thresholding not very necessary but done so to achieve a binary image)

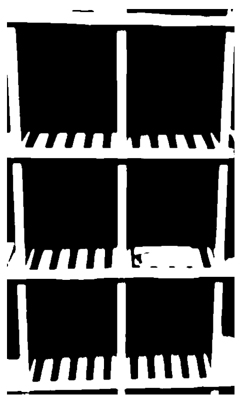
- Hough line transform is then applied to the section detecting the vertical lines.
- Non-existence of vertical lines are used to determine if something is obscuring the section.
  - Post will obscure the lines within the base.
  - An object will obscure the line on the side panels.

Obscured Postboxes

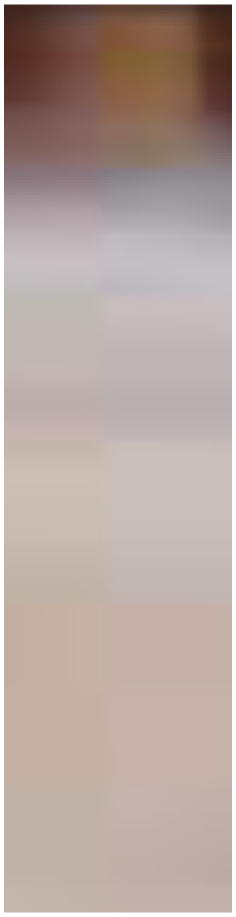
- Selection of posbox panels. For the setup of this procedure the four corners of each of the 9 postbox panels are retrieved. This was achieved

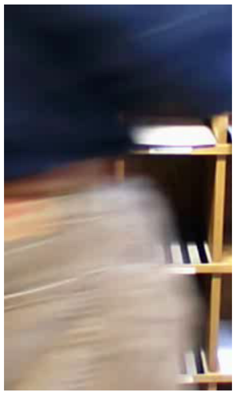
by through registering a callback funtion on a still, unobscured frame of the video. The registered callback upon mouse click printed to standard out the coordinates of where the click occured. Each corner of the 9 panels were clicked , from left to right (in the same layout as the given postbox base locations) such that when transformed into a rectangular frame, the panels will be vertical. The printing of the coordinates occured in the format of an array declaration in c++; every 4 pair of numbers were enclosed with curly braces with the appropriate commas. This was done as to allow for a human error-less copy and paste straight into the source code.

This is taking advantage of the idea that quite a defining feature for the postbox are the contrast between the side walls of the postbox and the inside of the postbox, I have highlighted the contrast in the binary thresholded image below (original frame was first converted to greyscale)


binary thresholded frame

middle panel(transformed)


middle panel obscured on frame 14

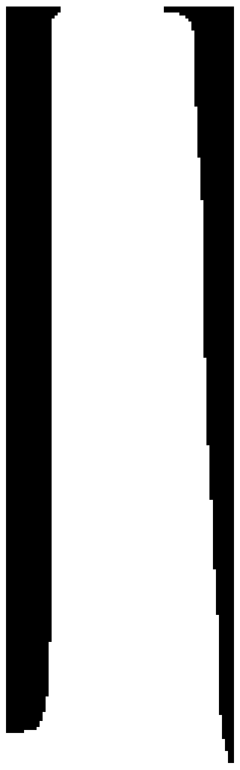

frame 14

- Transformed so that each section is vertical

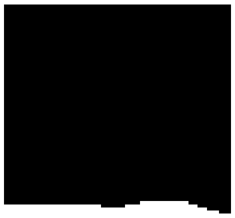
Each of the panels go through their own computer vision pipeline from here. Each of the panels are transformed into a 25, 100 dimension image, this stretches the image vertically and normalises each of the panel frames to be the same size. This also ensures that a vertical edge is clearly detectable by the horizontal derivative of sobel.

- Converted to greyscale and binary threshold

Each frame converted to greyscale and then binary thresholded. A binary threshold is done before the sobel edge detection. This way, the vertical edges of the frame create a much larger impact on the horizontal derivative.



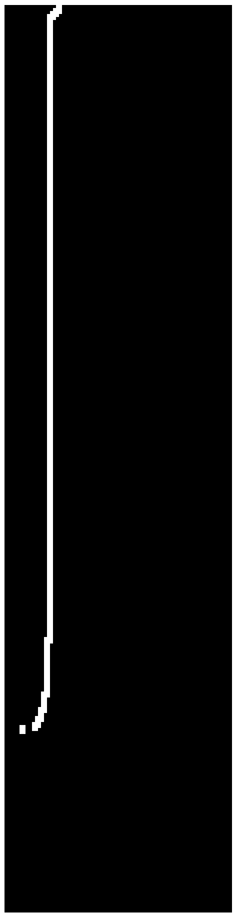
binary thresholded middle panel



obscured binary thresholded middle panel on frame 14

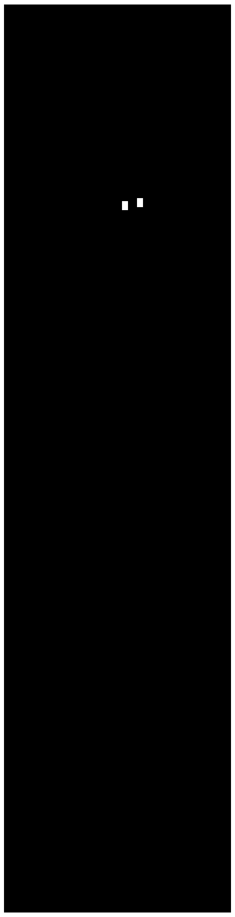
- Vertical edge detection and threshold

Sobel detects changes along the x axis and therfore returns the vertical edges. Only the horizontal derivative is considered as the only features this system are the vertical lines of the frames. Introducing a vertical derivative will only introduce information that the system doesn't care about. In a binary image, if sobels vertical line detecting mask is placed over a vertical straight line, one of the bits postive bits will have a chance of being multiplied by 0, whereas in a grayscale, that bit would have more weight to it.

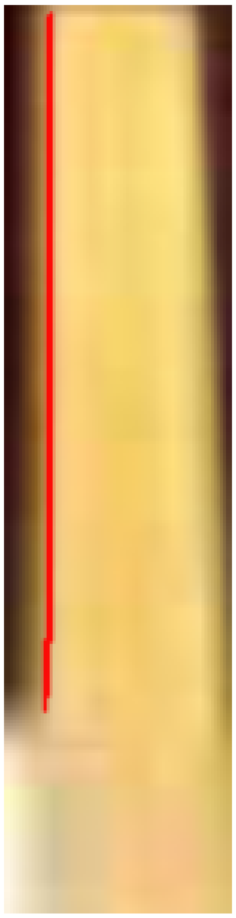

vertical edge detected and thresholded middle panel

- Straight line Hough transform on the binary edge image.

Using the binary edge image, straight lines are searched for using hough in the section, since whe have removed the possibility of horizontal edges, only


vertical edge detected and thresholded obscured middle panel on frame 14

the vertical lines will be detected. Since the resolution of the image is so low, the chances of an obscurance having a clear vertical element is extremely low.

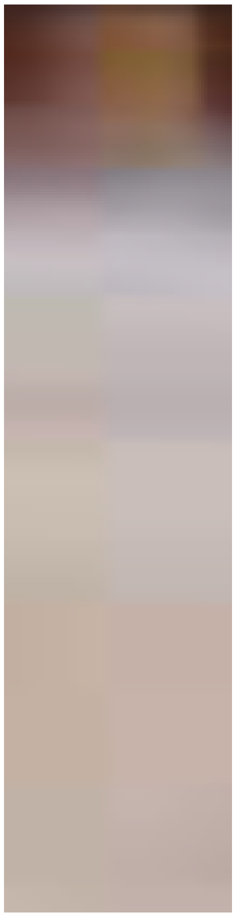
 red line showing the detected lines from Hough

- For each section, check if any line exists, if one is missing, the the postbox is obscured.

Having performed all these steps on the 9 panels, for each panel, it is checked if a line exists, if they all exist, then the postboxes are not obscured.

Post Detection

Post detection follows an identical process (different binary threshold values) for retrieving the lines in the section. The only stage which is different is the


no detected lines on obscured middle panel

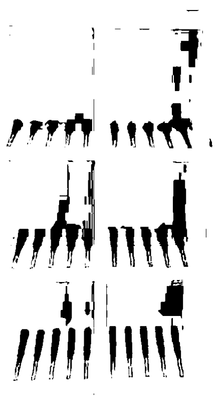
last stage which uses the lines in a slighty different way to check if the post exists.

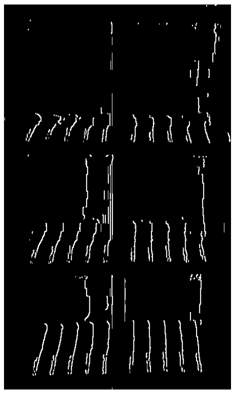
- For determining if a box contains post:

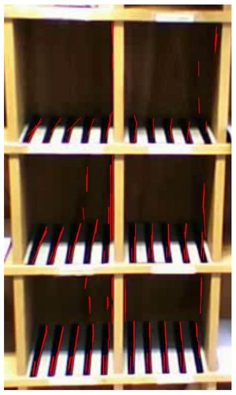
For each box, the number of lines that exist are counted. If for one post-box, number of lines are above 4, then the system classifies that as empty, otherwise there is post.

In some frames, two lines appear in the place of one, to overcome this, the distance between the line are checked, if multiple lines are close together, then they are counted as one.

frame 0 greyscale

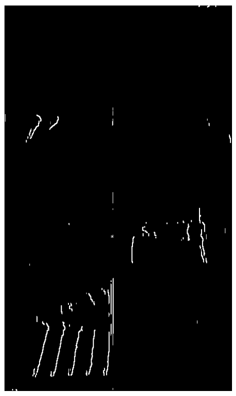

frame 0 binary thresholded

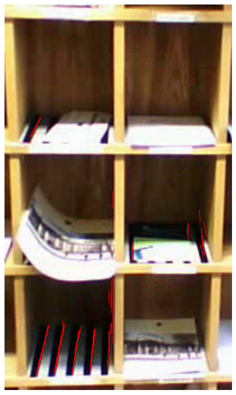

frame 0 binary vertical edge detection


frame 0 resulting hough transform

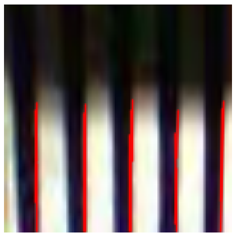
frame 42 greyscale

frame 42 binary thresholded


frame 42 binary vertical edge detection


frame 42 resulting hough transform

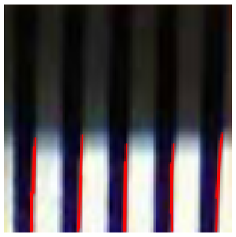
frame 4


frame 4 box 1

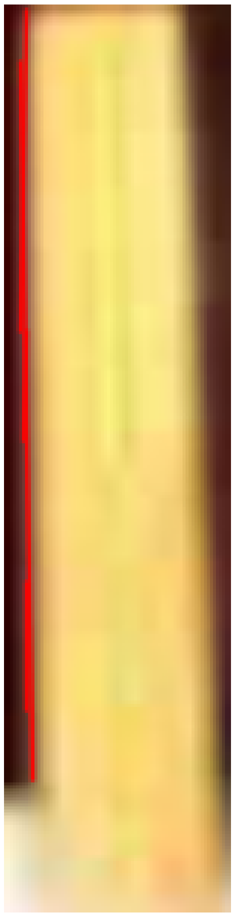
frame 4 box 2

frame 4 box 3

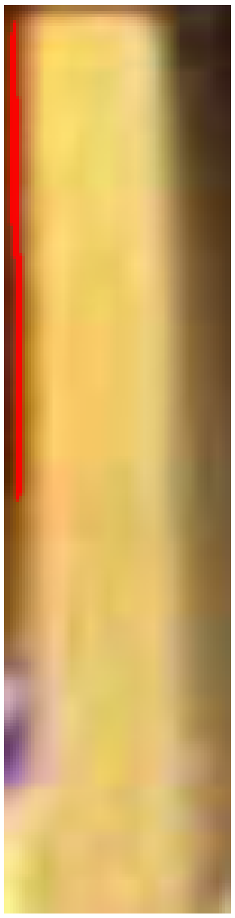
frame 4 box 4


frame 4 box 5

frame 4 box 6


frame 4 panel 1

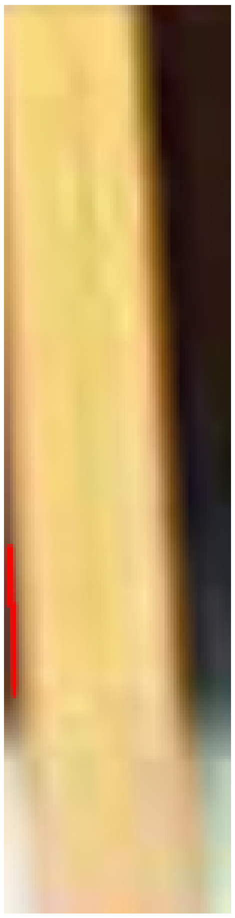
frame 4 panel 2

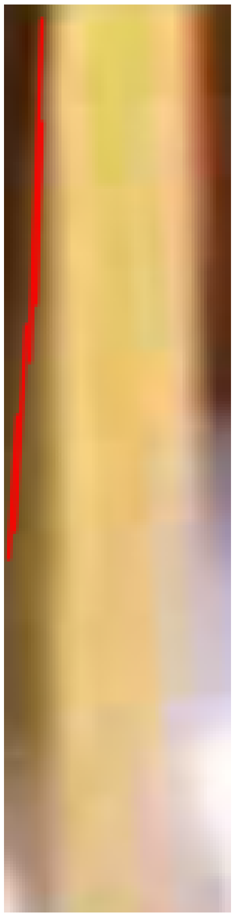

frame 4 panel 3

frame 4 panel 4

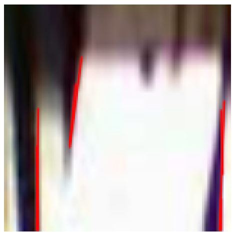
frame 4 panel 5

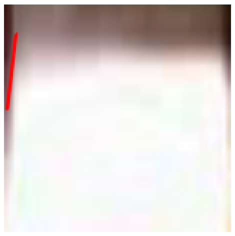
frame 4 panel 6

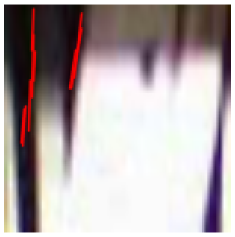

frame 4 panel 7


frame 4 panel 8

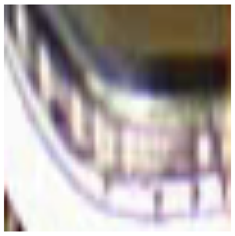
frame 19

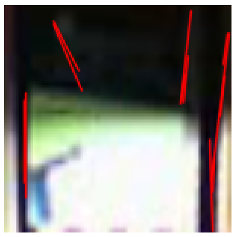

frame 19 box 1


frame 19 box 2


frame 41 box 1

frame 41 box 2


frame 41 box 3


frame 41 box 4

Frame 4, 19, 41

Discussion of the solution

Problems with the solution

Detecting whether the post boxes are obscured or not is currently a gamble hoping that the object obscuring it does not contain any vertical elements. This has a risk of creating a false positive, as the object might be obscuring the inside of the postbox while bypassing the obscuring sensor mechansism.

It is also not impossible for an object to miss obscuring the sidepanel and to just obscure the inside of the postbox, this object for example could be someones limb.

There is also the risk of a piece of having very pronounced vertical lines, triggering a false negative.

This solution might run the risk of not being able handle lighting changes, as the binary threshold values were fairly sensitive to thresholds chosen.

The slightest movement of the camera or the postboxes would heavily impact the systesm accuracy.

Generality of solution

The manual adjustment of thresholds gives a very low likelihood of working in different scenes.

The manual selection of post-box bases and panels also disalllows for this solution to work on the fly on different scenes.

Overcoming Limitations

To overcome the lighting changes, we can assume that the current lighting situation will be as bright as it gets and then set the threshold values to the upper-bound to allow a margin of different lighting to not affect the accuracy of the system.

To allow using the system on different scenes and postoxes, assuming a head-on perspective, the postboxes could have circular markings on the four corners of each box, then using hough transform for circles, the system could detect where the side panels and the inner regions of the boxes are.

Performance metrics

Obscured Postbox Detection

How well the system detected obscured frames.

- Recall: 100%
- Precision: 71%
- f1: 83%

Post in postbox Detection

Ommiting the frames when the "ground truth" identifies an obscured frame:

- For all the indiviudal postboxes 1,2,3,4,5,6
  - Recall: 100%
  - Precision: 100%
  - f1: 100%

Discussion of the results

Flexibility of 'Obscured' Definition

The ground truth seems to consider all the postboxes obscured as soon as a foreign object comes into the scene; individual postboxes may fully visible but everything is marked as obscured.

The ground truth is not totally 100% accurate for example, no object is in frame but is marked obscured e.g in frame 73.

The system that is at work only considers the postboxes obscured when the object is in front of them. Therefore the ground truth may not be very sutiable for this type of solution.

Perhaps a more accurate ground truth would mark only the truly obscured while leaving the unobscured still available for analysis.

Issues with Metrics

The metrics are too small of a sample size to determine the quality of the system. The enivronemnt is very controlled and the lighting is static.

The system could possibly be fine tuned to the current dataset, and possible give horrific performance if released as an application. The metrics in this perspective would be very misleading.

There is also an ambiguity with whether to count the ground truth obscured frames as false positives or not.

Proposal for how to locate the bases of the empty postboxes in the image

One way you could find the bases is to use simlar techniques as described in the system.

- Starting from a still coloured frame with empty postboxes.
- Convert the RGB image into greyscale.
- Binary threshold the image, picking a value that contrasts the black parallel lines in the base of the postboxes against the white background in the base; creating a binary image.
- Use sobel edge detection the obtain the horizontal derivative; the gradient edge image.
- Using the gradient image (basically greyscale), threshold again to turn in into a binary image again; the value of the threshold shouldn't matter (values 30-170 out of 250 will do) as there is going to be very sharp gradients in the image as it was edge detected against a binary image.
- Using the binary edge image, use hough line transform to detect a vector of lines (represented as a list of 2 points, 1 point for one end of the line, the other point for the other end)
- This will give a vector of vertical lines.
- Create a datastructure that holds a pair of points, and its euclidean distance between them.
- Initialise the datastructre with all the hough lines, along with the length.

- Sort the array of lines in terms of their length.
- Similar to sobel, find the rate of change across the array, generating a new array with rate of change at each index.
- Find the index of the maximum rate of change, then remove the lines in the original array from that point onwards.
- This will leave an array with only the lines from the base of the boxes, and the lines from the panels will be removed (since they are longer).
- For each line in the array, consider the point that has the lowest y value, put those points (x,y) in a separate array.
- Sort the array in terms of the y values of each point.
- Retrieve the rate of change at each index.
- Using that, find the rate of change of the rate of changes at each index.
- With this double derivative, find the two highest rates of change in the array.
- Split the array into 3 sections bases on the locations of the highest rates of change.
- Each of the 3 arrays represents one of the 3 rows of boxes in the image
- The first array will be the bottom two, the second, the middle two, and the thrid, the top two.
- The following steps can be repeated for each array:
  - Choose one array.
  - This time sort it by the x values.
  - Find the rate of change of the x values.
  - The maximum rate of change represnets the gap across the two boxes
  - Find the maximum and split the array into two.
  - Find the minimum x value of each array and its corresponding y value, reference this point with the orignial array of lines to find which line it was originally apart of.

    * Having these two points you now have the bottom left and top left points of the base and also the bottom left and top left points of the other postbox.
  - Now do the same for the maximum x value for each array.
    * This will give the top left and bottom left points.
- This repeated on the three arrays give the bases of the postboxes.