*Render Movies on Pictures inside a Movie*

**Computer Vision and Image Processing - Final project Report**

Margarita Pinhasov ID 320937741 Dana Engel ID 308158468

* **Introduction**
* **What is the problem**

We used two videos as input. the first is of a picture on the wall from a moving camera (denoted by A) and found some other video in the internet (denoted by B). We needed to return A, while instead of the picture there should be B. we used *Hough transform* and *Canny* to detect the borders of the picture inside the video (A) and wrap the frames from B on the detected quadrilateral in A.   
Finally, making the transition between the frames to put it back into a video.

* **What was our approach to solve it**

Our approach was to take two videos and spread them into frames, to find the picture's corners and borders in the first frame and then to track it in all the other frames, and then to warp the frames of B and blend them with the frames of A according to the coordinates we found.

* **Approach and Method**
* **Details about the approach we implemented and tested, how does it work and why**

We took two videos and spread them into frames: The video of the picture on the wall (A), and the video to be screened on the picture (B). This was done in a function called vid\_2\_frames.

Out of the frames of video A, we took the first one and found the corners of the picture on the wall in a function called Corners, this function uses

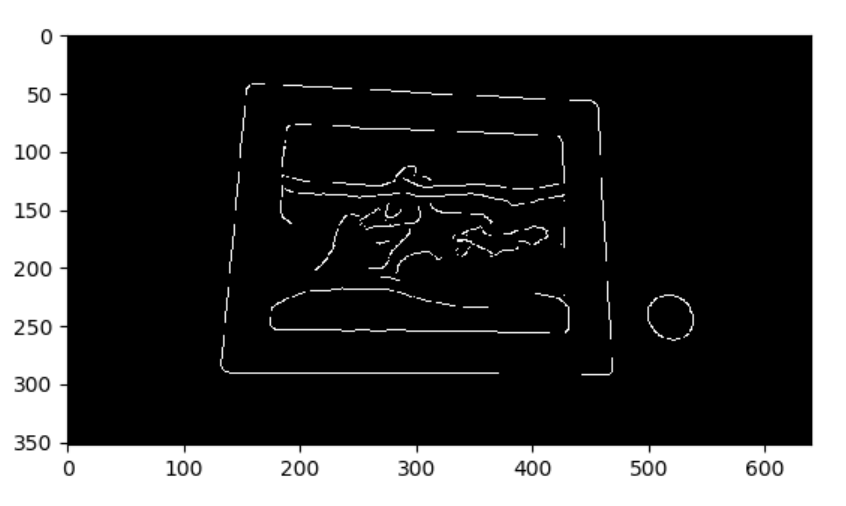


Image 1: Edge image

Hough transform to detect the lines in the frame with the help of an edge image (image 1) that is returned from the Canny algorithm. This was implemented in a function called findAllLines. With the results we got from these functions, we found the points of intersection between these lines in a function called cross\_point, but these points included the ones that are outside the picture, so we removed them.

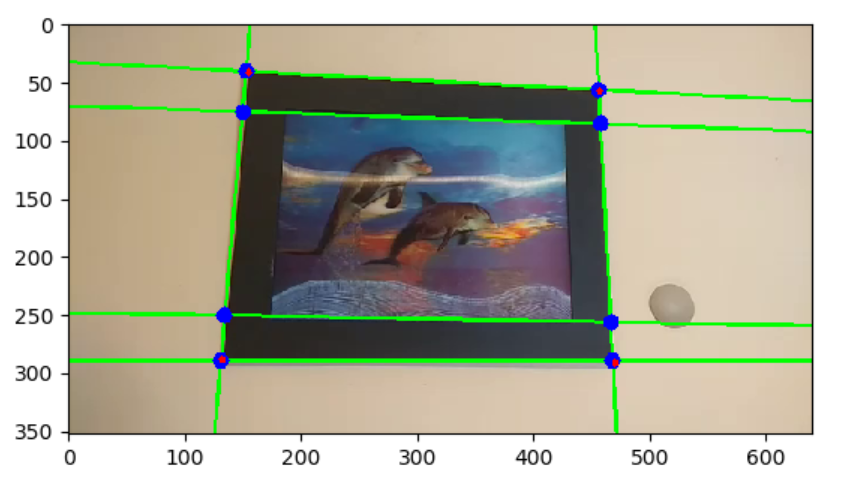


Image 2: lines in green, cross points in blue and the corners in red

Then we have made 4 sub images (a sub image is a square gray scale image that contains a small part of the hole image – image 3) each sub image is centered on one of the intersections (the blue points in image 2) and multiplied with a special kernel:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 1- | 1- |
| 1 | 1 | 0 | 1- | 1- |
| 0 | 0 | 0 | 0 | 0 |
| 1- | 1- | 0 | 1 | 1 |
| 1- | 1- | 0 | 1 | 1 |

The size of the kernel is dynamically determined by the size of the sub image.

Why it works: When this kernel is multiplied (matrix dotting) with a gray-scale sub-image (both are square matrices) its trace is farther from zero if one quadrant of the sub-image has an average value which is different from the average value of the other three quadrants. This allows us to assign a value (brightness) to a point according to the absolute value of the aforementioned trace. Performing this operation on a sub-image gives us a local heat-map where the brightest spot is the most probable location of the corner we're seeking. The intersections of the lines that we've found with the Hough method gave us a rough estimation of the corners (of the picture on the wall), which is refined by finding the heat-map (image 4) of the sub-images around the intersections.

This process takes place in a function called findCorners, corner\_xy and corner\_detector. This way, we scanned the intersection points and found only the 4 points with the maximal value – and these are the points we need.

The next step was to use the the hit map approach (without Hough) for the next frames to find the corners of the picture. this happens in warpedlist. We know that the coordinates of the corners aren’t far away from each other in between close frames so we do the same process when the middle points are the ones we found in the last frame.

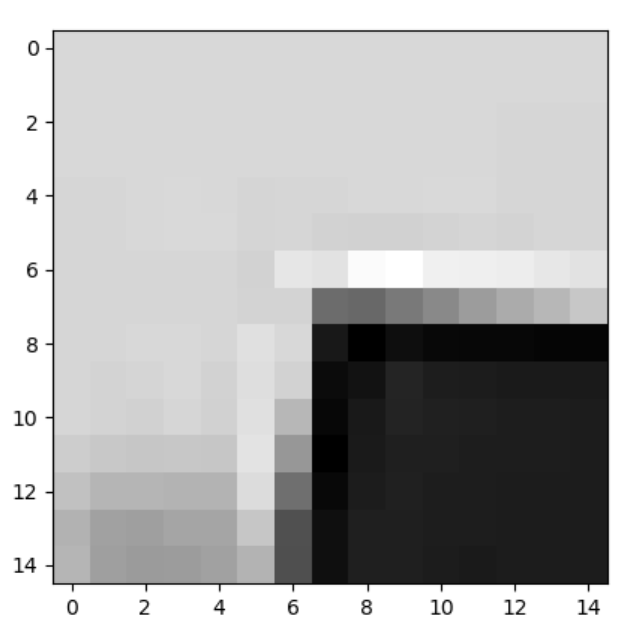


Image 3: sub image of the left upper corner in gray scale

After we found the corners of the picture we move on to the **wrapping** process, in which we need to distort the frame from video B to the shape and size of the shape we got out of the 4 points we found.

In the wrapping process we create a transformation matrix with the 4 points that we found and the 4 corners of the frame from video B (as source and destination points), then applying the transformation on the pixels location in frame B which gives us the location of the pixels in the frame from A (in a case that several pixels are sent to the same location we calculate the average of their colors). At the same time we create a mask – a black and white image in which the white pixels are the picture on the wall and everything else is black, and finally we preform the **blending.** The frame from video B after the warping is now at the location of the white shape, and the black part is the frame from video A. This process is done for every frame in video A and video B respectively in a function called warpedlist, and then we merge all the frames into a video in a function called makevideo.

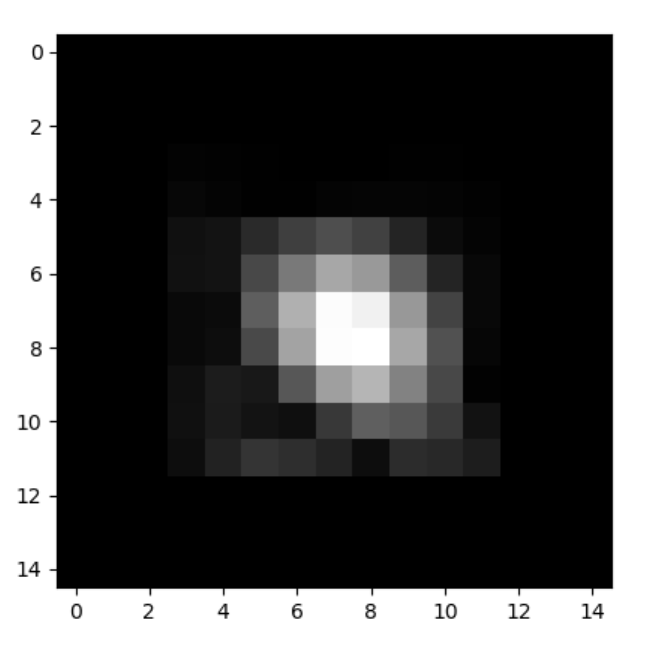


Image 4: multiplication of the sub image with the special kernel – hit map

* **What were the limitations**

The first limitation we faced was when we took a photo of a picture on the wall with a frame that had a low contrast relative to the background wall, that made it impossible to detect the picture's edges. So, our first conclusion was to use a picture that has a darker frame so that the contrast with the wall would be higher.

Then, when filming video A we realized that the movement of the camera should preferably be smooth, so that the displacement of the corners of the picture on the wall between frames will be low, so it will make it easier for the algorithm to track the picture on the wall. But, we came up with the solution to use the Hough approach when the hit map approach fails.

Another limitation we faced was in a situation where the frame of the picture isn’t entirely seen, so tracking the frame of the picture is impossible. Also, if the picture's frame is close to the border of the picture, the sub-image would be placed outside the picture's border – which will lead to an error. To solve this issue, we added padding – a large black frame around the sub image, to avoid an error.

The algorithm may act differently on different videos so adjustment of the algorithms parameters will be required for each different video.

Another limitation was when the camera is rotated (roll axis). It was hard to know where is the right/left top/down corner, and the wrapping function should get the points in a specific order, otherwise the wrapping gives the wrong output, here is one example:



The efficiency of the code was also one of our concerns, as the process for one image can take up to 40 seconds, so running it over 300 frames takes a very long time. That forced us improve the efficiency of a lot of the functions so our program will run faster (now the prosses for one frame takes on average 6 seconds).

* **Results**
  + **Describing the experiments that we have done**

At first, we tried to use the Harris and optical flow algorithms which didn’t work well on our input, and that led us to use the hit map method for tracking instead.

We tried to apply the algorithm on different videos to find the optimal parameters and adjust then accordingly. The parameters that we tried to adjust were:

The sub image size, Minimum, maximum of the line thickness of the edge image from Canny, the sigma for the blurring prosses, number of peaks, threshold for the number of votes and neighborhood size in the Hough algorithm.

All of those parameters are used in the Corner function.

* + **Which metrics did you use to measure success and why?**

We used the brightness of the pixels from the corner\_xy function and also the difference in the area of the shape defined by the four corners we found between following frames, if we found unusual results in comparison to the previous frame we used the hough algorithm like in the first frame.

The brightness was used because we found through experiments that the value of the correct corners is always above a specific threshold which is different for each video, and so if the brightness is suddenly less than the threshold, we can assume that hit map method failed to find the corners.

Similarly, the area difference of the warped picture between two following frames should be very small, so if the algorithm fined an area difference which is above a certain threshold, we conclude that something went wrong and use the Hough algorithm once more.

* + **What are the final results**

We have made a lot of different videos, in the beginning we had a lot of problems with finding the right corners in the correct order and that’s why we came up with improvements such as the hit map method and the area comparison, this helped a lot and we finally have been able to create a 10 second video without any issues. we decided to try with anther video, longer this time, the second video forced us to change a lot of the parameters so it could work better, it is not perfect , but we sure that with a little more adjustments we could get a better result.