

1. Detect line in the image

1.1. Normalise the image

If we read images in greyscale, the img21 will have min value 26 and max value 249 which are different from other images. Therefore, I normalised all images to make them has the same maximum and minimum to get consistent edge detection result among all images.

1.2. Canny edge detector

In canny edge detector, there 3 main parameters to adjust:

1. Higher hysteresis threshold (t_1)
Edge that has magnitude of gradient larger than higher than t_1 will be considered as a strong edge.
2. Lower hysteresis threshold (t_2)
Edge that has magnitude of gradient smaller than t_2 will be excluded. And edges that have a magnitude between t_1 and t_2 will only be kept if they are connected to a strong edge.
3. Aperture size
The aperture size is the size of the Sobel kernel. The larger the aperture size, the more fine-grained edges will be found, and also more noise may be introduced (see Table 1).


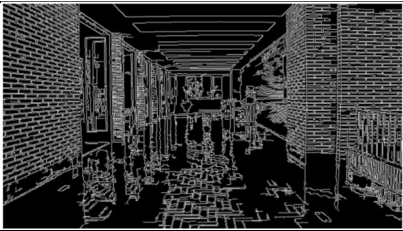
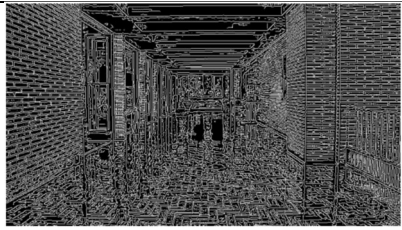
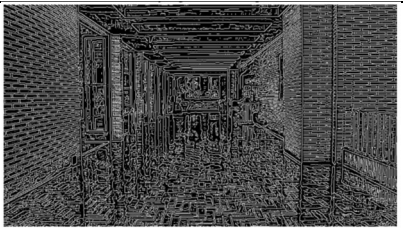
Original Image	Aperture Size = 3
	
Aperture Size = 5	Aperture Size = 7
	

Table 1. Experiment on aperture size

In our task, we want to only keep important edges, so we use small aperture size (=3) and large t_2 (=200) to only keep strong edges. The value of t_1 should be carefully adjusted to exclude trivial edges while keeping the structure edges of the scene.

After multiple experiments with different combinations of t_1 and t_2 . I found that the trickiest images are “img14, img15, & img18.” The “img14 & img15” has relatively low contrast between floor, ceiling, and walls, so the edges of the room structure are hard to detect. On the other hand, img18 contains too many small items, therefore after the edge detection, it may contain many noises.

Finally, I decided to use $t_1=40$ and $t_2=200$. As Table 2 shows, this setting can capture more lines that can contribute to vanishing point detection such as the edge of the carpet in img15 (circled in red in Table 2). Although this setting will include more trivial edges in img18, I will filter out those edges in the Hough transform.


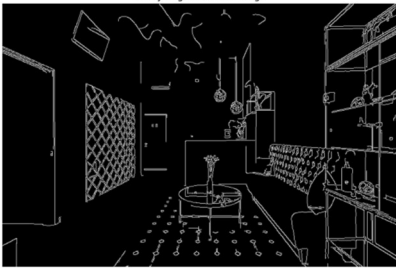
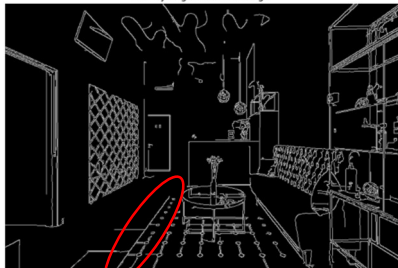

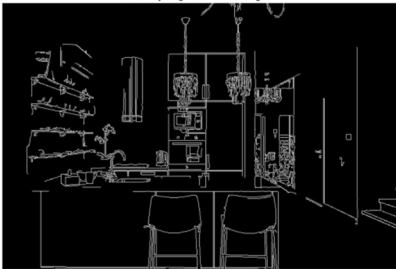
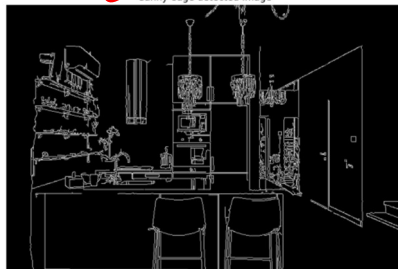

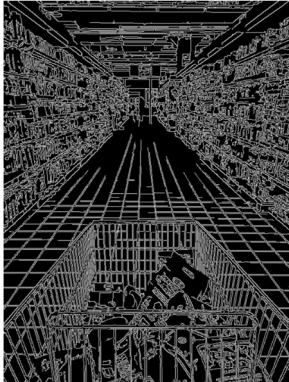
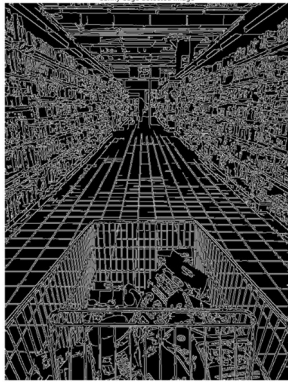
	Original Image	Canny edge with t1=80, t2=200	Canny edge with t1=40, t2=200
img15			
img14			
img18			

Table 2. Experiment on thresholds

1.3. Hough transform

In this task, I decided to use `HoughLinesP()` function to detect the lines because it has two more params which are `minLineLength` and `maxLineGap` that can be used to filter out undesirable lines (e.g. lines produced by texture region).

The parameters “rho” and “theta” are the distance and angle resolution. Here, they are set to 1 pixel and 1 degree to find most of the lines. Threshold (minVote) is set to 50 to prevent no lines from being detected in img14 and too many lines being detected in img18.

In this task, I set `minLineLength` to a large value (`=100`) to reject short lines (e.g. lines of texture) because the structure lines of the images tend to be long (see Table 3). Also, I set `maxLineGap` to a small value (`=10`) to prevent the model from connecting multiple noise lines (e.g. lines of texture) into a long line (see Table 4).

minLineLength = 100	original img21.jpg	Canny edge detected image	Hough transform detected image
	original img15.jpg	Canny edge detected image	Hough transform detected image
	original img14.jpg	Canny edge detected image	Hough transform detected image
minLineLength = 40	original img21.jpg	Canny edge detected image	Hough transform detected image
	original img15.jpg	Canny edge detected image	Hough transform detected image
	original img14.jpg	Canny edge detected image	Hough transform detected image

Table 3. Experiment on minLineLength

maxLineGap = 40	original img18.jpg	Canny edge detected image	Hough transform detected image
maxLineGap = 10	original img18.jpg	Canny edge detected image	Hough transform detected image

Table 4. Experiment on maxLineGap

1.4. Filter out lines that are too horizontal or vertical

After Hough transform, I filtered out the lines whose slope is larger than 4 or smaller than 0.25, because those lines cannot help the vanishing point detection.

2. Locate the vanishing point by RANSAC

In this task, I fed the detected lines from section 1 into my RANSAC loop. A detailed explanation of RANSAC steps, parameters setting, and stopping criteria are discussed below. The vanishing point detection result is shown in Table 5 as well as the converge iteration's number. In Table 5, the sampled lines are drawn as blue, the inliers lines are drawn as green, and lines detected by Hough transform are drawn as pink.

RANSAC Steps (for each image):

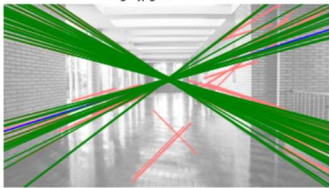
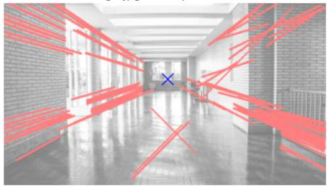
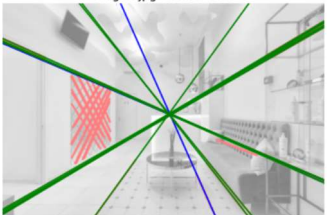
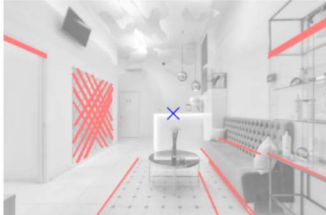
- Random sample 2 lines from the detected lines of that image.
- Compute the intersection point of the 2 sampled lines.
 - If the two sampled lines are parallel, go to the next iteration and sample again.
- Calculate the distance between the intersection point and the rest of the detected lines.
- If the distance is smaller than the threshold, then treat that line as an inlier.
- Compute the number of inliers.
- Repeat (a)~(e) for multiple iterations, until the ratio of inliers or number of iterations reach the stopping criteria.
- Use the intersection point that has the highest number of inliers as the predicted vanishing point.

Parameters setting:

- Iteration: 1000
Because the 12 images have different lighting conditions, contrast, and saturation, some of them can find the intersection point very quickly while others may need more iterations. As we can see in Table 5, img9 and img18 need around 700~900 iterations to find the best result because those two images contain more tiny details and texture causing the detected lines to contain some noise. Setting the iteration to 1000 is conservative and can guarantee the predictions on all images to converge. Although 1000 iterations seem a lot, most of the images will meet the stopping criterion in the first few epochs (see Table 5); therefore, it is not computationally expensive.
- Threshold: 5 pixels
I set the threshold to 5 pixels. This value shouldn't be too large, or it will include wrong line as the inliers. Also, this threshold should not be too small, or it will not have enough inliers to judge the performance of the prediction. After multiple experiments, 5 pixels performed the best.

Stopping criterion:

- Ratio: 0.7
I set the stopping criterion as 70% lines in an image are inliers (pass through or near the vanishing points). If this ratio is too low, then the model may stop too earlier and cannot find the best solution. Based on my observation, 70% is a reasonable ratio to allow well-predicted images to stop earlier while allowing tricky images to be properly explored.

Samples (blue) & inliers (green)	Predicted intersection point	Samples (blue) & inliers (green)	Predicted intersection point
img1.jpg Iteration 76	img1.jpg - final prediction	img15.jpg Iteration 31	img15.jpg - final prediction
			

img13.jpg Iteration 0	img13.jpg - final prediction	img14.jpg Iteration 0	img14.jpg - final prediction
img10.jpg Iteration 1	img10.jpg - final prediction	img18.jpg Iteration 829	img18.jpg - final prediction
img20.jpg Iteration 17	img20.jpg - final prediction	img21.jpg Iteration 0	img21.jpg - final prediction
img3.jpg Iteration 338	img3.jpg - final prediction	img6.jpg Iteration 88	img6.jpg - final prediction
img7.jpg Iteration 121	img7.jpg - final prediction	img9.jpg Iteration 741	img9.jpg - final prediction

Table 5. Vanishing point detection results & their converged iteration

3. Main function and evaluation

The prediction result and the error (squared Euclidean distance) are shown in Table 6. The visualisation of the predicted vanishing point of every image is shown in Table 5. The mean squared Euclidean distance between the predicted and ground truth vanishing points is 29.36 pixels.

As we can observe from Table 6, img10,14,15,20 have larger errors. I extracted the prediction results of those images and put them in Table 7.

I observed that img10 and img14 have few lines detected after Hough transform due to these two images have low contrast; this led to the larger vanishing point prediction error. After all, img10 and img14 don't have enough lines to sample, also, it is hard to choose good model parameters (x,y) because the number of inliers won't have large variances among iterations.

On the other hand, although img15 and img20 have more lines detected after Hough transform, they have more noise lines (lines generated by texture or shadow) than the lines that can be used to find the vanishing point. Therefore, they still won't have enough lines to find the best vanishing points. Moreover, they always have a low ratio of inliers due to the large number of noise lines in the images; this may cause the choice of model parameters (x,y) harder.

Overall, my model achieves 29.36 pixel mean squared Euclidean distance, which means that on average the distance between ground truth and predicted vanishing point is around 5.4 pixel. An error of 5.4 pixel in a 600x800 image is less than 1%. The performance of my model is acceptable.

Limitation of Canny edge detection and Hough transform: cannot handle images' variance by only one general set of parameters.

As we discussed above, the error of my model is dominated by some images which have few lines detected after the Hough transform. However, due to the variance of lighting conditions, saturation, and contrast among the input images, it is not likely to detect all good lines in all images by simply play around with the parameters of the Canny edge detector and Hough transformer. Some of the images would contain few detected lines or some of the images would be detected with many noise lines.

Further improvement:

To further improve the performance, we need to normalise the different contrast, saturation, and lighting conditions among all input images and even do the camera calibration to prevent the line in images from being distorted. In that case, the model should be more robust.

	<i>Ground truth vanishing point</i>	<i>Predicted vanishing point</i>	<i>Squared Euclidean distance</i>
<i>Img1</i>	(404, 183)	(404.72, 185.02)	4.59
<i>Img3</i>	(396, 276)	(394.48, 275.86)	2.32
<i>Img6</i>	(378, 258)	(375.58, 255.68)	11.24
<i>Img7</i>	(379, 423)	(377.59, 424.33)	3.74
<i>Img9</i>	(438, 367)	(435.70, 370.30)	16.14
<i>Img10</i>	(240, 466)	(239.56, 476.59)	112.39
<i>Img13</i>	(390, 260)	(388.42, 262.94)	11.12
<i>Img14</i>	(375, 270)	(372.67, 261.42)	79.01
<i>Img15</i>	(416, 270)	(415.93, 276.49)	42.14
<i>Img18</i>	(292, 213)	(293.31, 214.25)	3.28
<i>Img20</i>	(406, 288)	(398.38, 290.50)	64.22
<i>Img21</i>	(389, 208)	(387.94, 209.00)	2.13
			Avg. = 29.36

Table 6. Evaluation

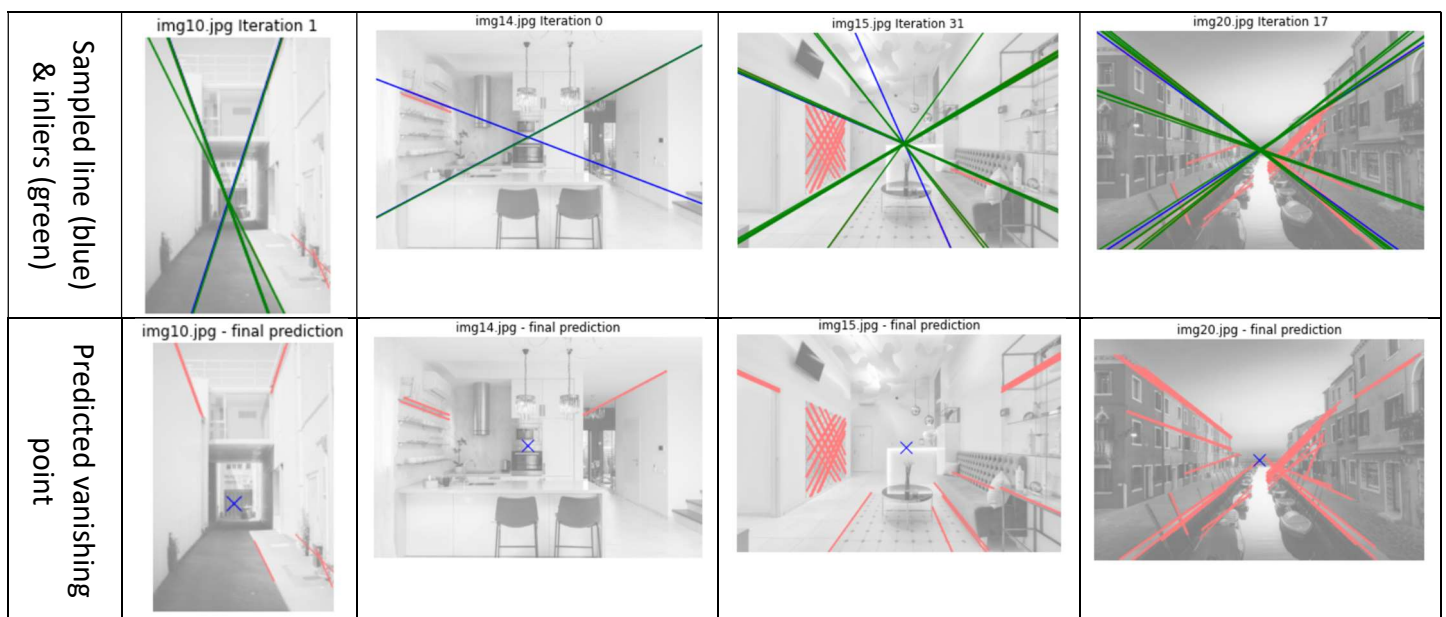


Table 7. Images with large prediction errors