Melbourne School of Computing and Information Systems

COMP90086 Machine Learning - Assignment 3:

Vanishing point detection

Tuan Khoi Nguyen - 1025294

Word Count: 498 (Q.1), 430 (Q.2), 398 (Q.3)

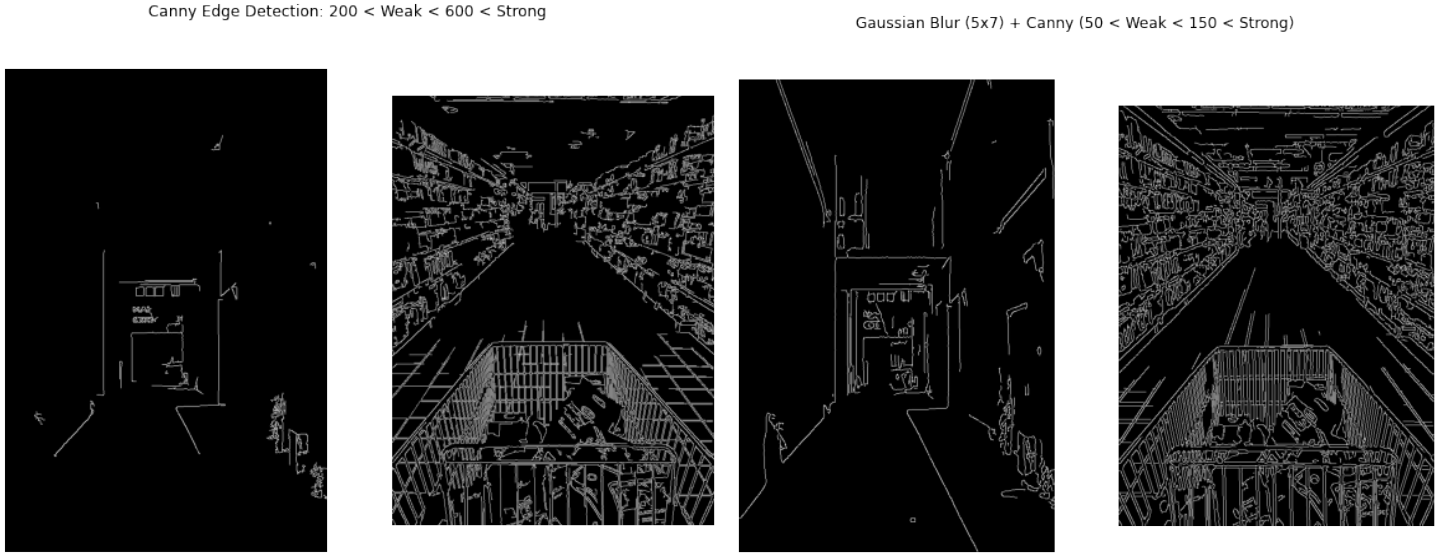
Question 1: Edge and Line detection [Word Count:498]

1. Canny Edge Detection

Canny Detection finds edges by checking for changes with gradient exceeding a threshold. Built-in function `cv2.Canny()` will be used for this task.

Gaussian Blur Filter

Observation shows that images are varied in color complexity. While some pictures has few color shifts, there also exists pictures with detailed textures. This results in a picture having most of its detail omitted, while another one still has noisy details after Canny Detection, shown in Figure 1a. Therefore, blurring is needed to remove noise and small details, making edge detection more consistent across images. Tuning shows that blurring with a 5×7 kernel works the best, which larger kernels resulting in higher risk of losing important details, and smaller kernels are less effective in removing noises.



(a) Original image

(b) Applied 5×7 Gaussian Blur Filter

Figure 1: Canny Edge Detection result on different processings

Skeletonizing

Another problem are thick or noisy lines after processing, making line detection more prone to duplication, or noise. Skeletonization helps preventing this, by thinning down thick lines or noise down to unit level.

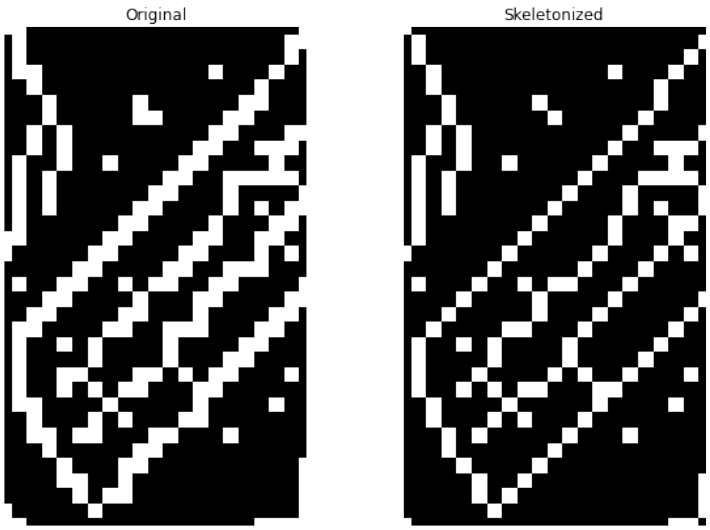


Figure 2: Skeletonization effect on overlapping pixels

Hysteresis Thresholding

Canny Edge Detection with Hysteresis Thresholding has 2 magnitude thresholds which take magnitudes above the higher one and its neighbors with magnitude higher than the lower one. As important details may share the same magnitude with noise or small textures, single threshold may miss out important details or has frequent noise. Therefore, the best result should be obtained with 2 different thresholds.

If the higher threshold is too high, the method may not be able to detect enough details, and may have frequent noise if the variable is too low. This also applies to lower threshold, where small values may lead to more noise coming in.

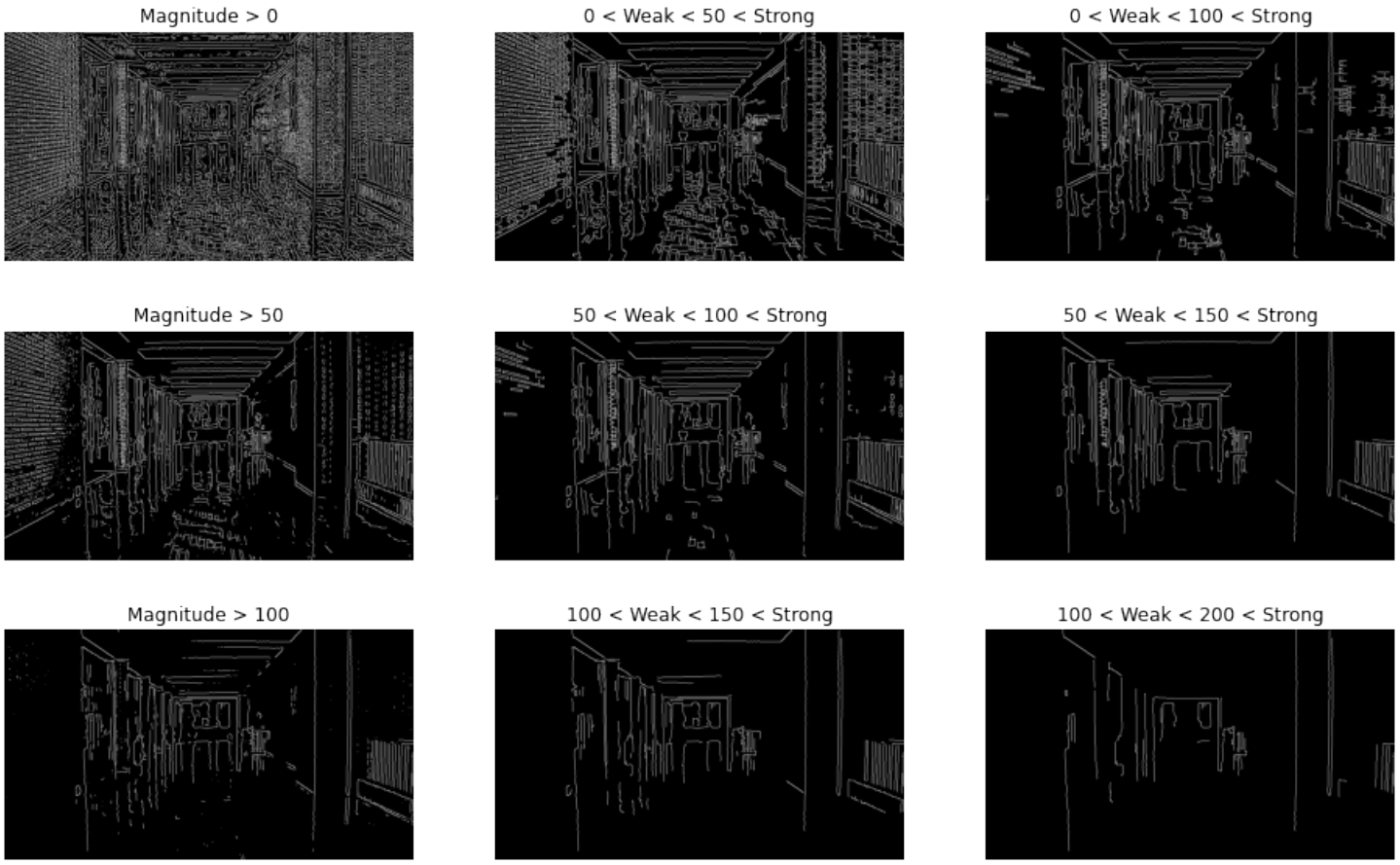


Figure 3: Tuning result for different thresholds of Canny Edge Detection

Result from Figure 3 reflects the statements above, and shows that thresholds of 50 and 100 keep the most amount of geometrical details, while effectively cancel out noise.

2. Hough Transform

Hough Transform works by retrieving lines that pass a minimum number of points. The implementation will use `HoughLinesP()`, a function that operates on a random subset of lines to reduce computation.

Range of $tan(\theta)$

To increase efficiency, lines that are close to being horizontal or vertical are left out, as observation shows that these lines are less likely to converge to the vanishing point, given that they are usually results of irrelevant textures. The tan threshold $tan(\theta)$ makes sure that the angle between the line and the horizontal surface has its absolute tan value in the range of $[\frac{1}{tan(\theta)}, tan(\theta)]$. Result shows that threshold works best at 2.3, which eliminates most irrelevant lines without losing too many important ones.

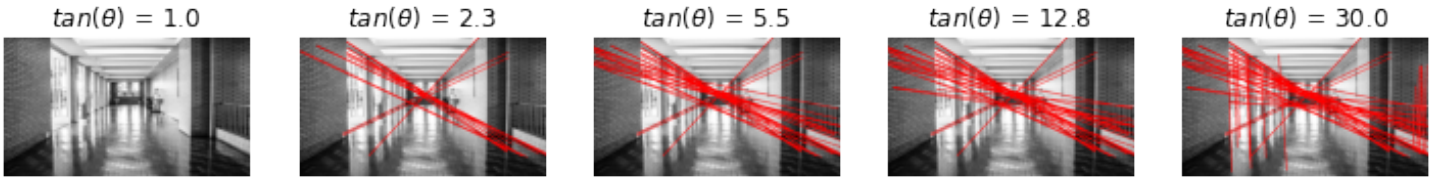


Figure 4: Hough Transformation on different angle ranges

Maximum line gap

The function also merges lines within a gap. For larger gap, more lines are detected, resulting in longer and visible lines, but may also merge noise to create more irrelevant lines. Lower gap may not able to catch all small lines, which may miss out relevant details. Results shows that 250 works best in keeping relevant lines.

Figure 5: Hough Transformation on different line gap thresholds

Voting Thresholds & Minimum line length

The threshold decides how many points the line needs to pass to be included. The function can also cross out shorter lines to avoid noise. Smaller thresholds or allowed length makes irrelevant lines appear more often, while larger values may not be able to capture important details. Figure 6 & 7 shows that voting threshold of 50 & minimum length of 200 keep most relevant lines for the image, without letting too much noise in.

Figure 6: Hough Transformation on different minimum lengths

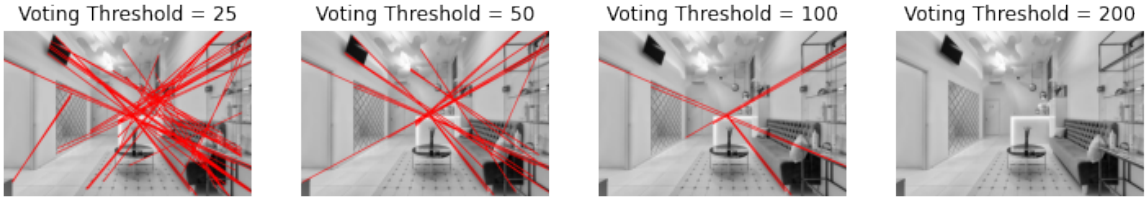


Figure 7: Hough Transformation on different voting thresholds

Question 2: Locating The Vanishing Point [Word Count:430]

1. Method

To locate the vanishing point, RANSAC is used. The implementation of RANSAC uses a number of functions that will have their working mechanism described in this section.

Helper Functions

The RANSAC loop uses 2 following helper functions:

`find_intersection_point()` Finds the intersecting point of 2 given lines. It first check if the lines are not parallel by confirming the dot product of one's orthogonal vector and the other vector is non-zero, then use dot product ratios to check for displacement vector to the intersection, from the starting point.

`find_dist_to_line()` Finds the distance of a point to a line represented by 2 coordinates that it passes through. It uses vector projection method to extract the projection vector, then find its magnitude.

RANSAC Loop

The process of each RANSAC run is implemented in the function `ransac()` as follows:

- Randomly select a number of line pairs that are not parallel and find their intersection, with `find_intersection_point()`. The number of pairs to be chosen can be set on `n_iters` variable.
- For each intersection, count the number of lines that are considered passing through the point, which has the projection distance smaller than the specified margin. The projection can be retrieved using `find_dist_to_line()` function.
- Pick the intersection that has most lines passing through as the vanishing point.

2. Tuning parameters

For RANSAC, there are 2 main parameters that affect the performance of the model: the iteration limit, and the margin. These parameters will be tuned to find the most efficient configuration, as well as evaluated through different random seeds to check for robustness.

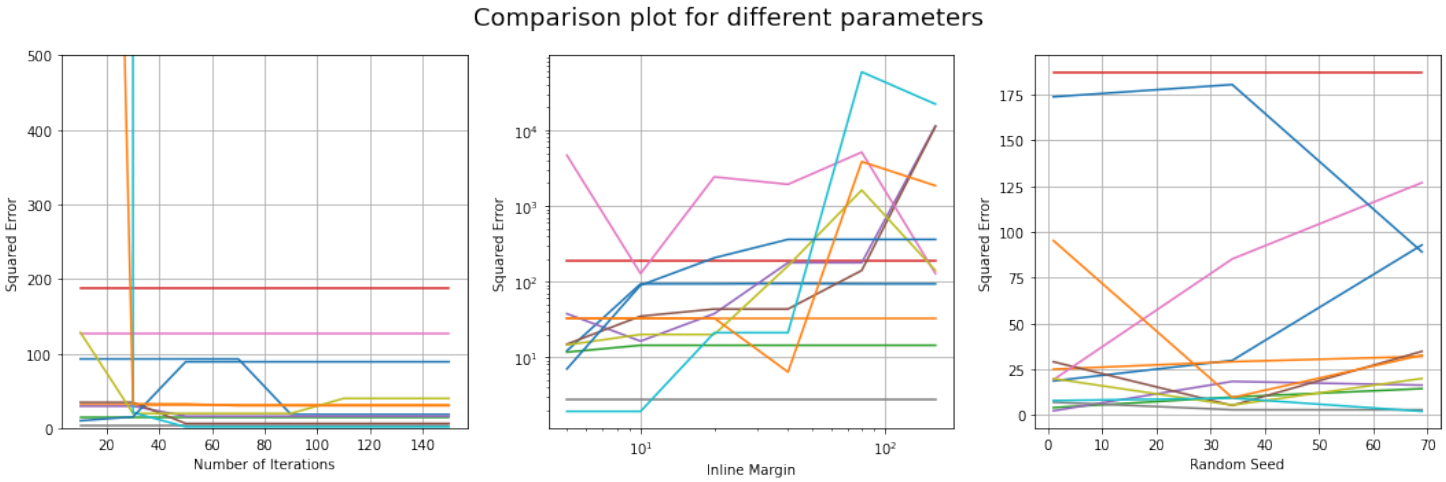


Figure 8: RANSAC run on different parameter values

Number of Iterations

The number of iterations indicate the limit on sampling the intersection points, in case the function has not run out of combinations. Large or no limit may make the process computationally costly, while smaller values may result in more unstable runs. Tuning plot on Figure 8 shows that the process starts to become stable after 120 iterations for most images, hence chosen as the limit for this task.

Inline Margin Limit

The margin indicates the maximum distance for a line to be considered passing the point. Both too small or too large values make the points having similar number of lines passing, where points will either have strictly 2 lines passing, or all lines passing through. This makes the model only have the performance of a random selection. Tuning plot on Figure 8 shows that the margin of 10 has the best stable outcome for most instances.

Random Seed

As the process involves random sampling, setting the random seed can make the results reproducible, and setting different seeds and check the stability of the outcome. Results on Figure 8 shows that different seeds has similar outcome across images, proving that the method is robust, and has reasonable squared error.

Question 3: Main Function & Evaluation [Word Count:398]

Canny Edge Detection & Hough Transform

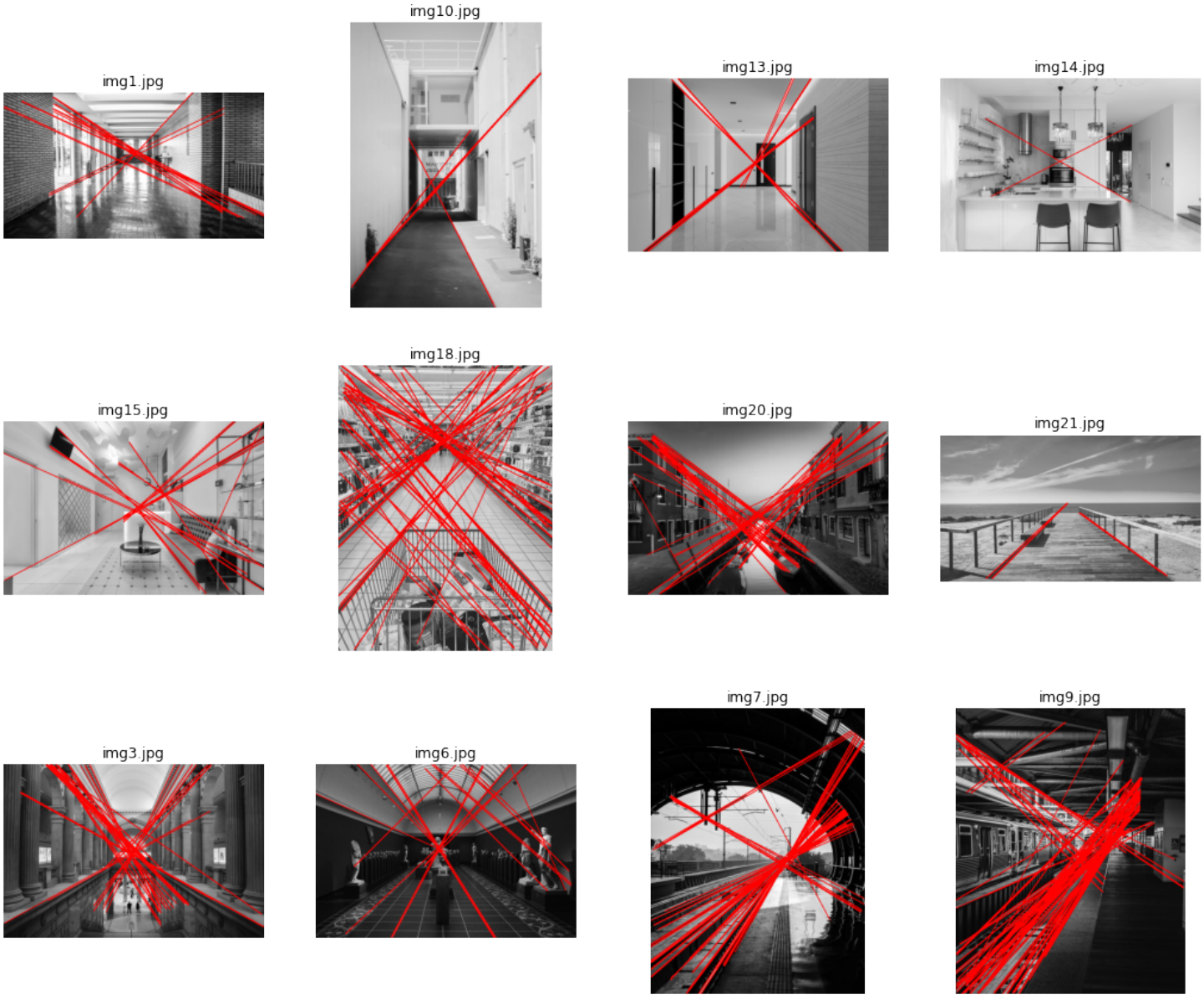


Figure 9: Edge line detection results on different input images

The resulted lines shows that the chosen parameters and additional steps worked as expected, with all images having sufficient lines that passes through the vanishing point. Most images have a reasonable level of irrelevant lines that can be handled with suitable RANSAC parameters, given that they do not outnumber the points going through the vanishing point.

However, it is noticeable that the number of the irrelevant lines in some images are still significant, such as `img18.jpg` or `img20.jpg`, which may still have a small odd of distracting the real result, given that RANSAC only takes a random subset of points. This is because the complexity difference between images are too different, making fully removing noise without losing details infeasible. In order to make the algorithm more efficient in this, further in-depth tuning needs be done in order to eliminate further noise while keeping reasonable amount of important details.

Due to the limited constraint in time and resources, the efficiency is not fully achieved. When condition allows, further improvement can be done to make edge detection more precise. These includes in-depth tuning at smaller ranges, or constructing a dynamic algorithm that automatically choose value based on image complexity.

RANSAC & Final Results

```
img1.jpg  | Point = (407.60, 185.54) | Squared Error = 19.4213
-------------------------------------------------------------
img10.jpg | Point = (238.77, 472.66) | Squared Error = 45.8613
-------------------------------------------------------------
img13.jpg | Point = (386.50, 259.50) | Squared Error = 12.4706
-------------------------------------------------------------
img14.jpg | Point = (367.34, 258.36) | Squared Error = 194.0569
-------------------------------------------------------------
img15.jpg | Point = (417.38, 265.45) | Squared Error = 22.6203
-------------------------------------------------------------
img18.jpg | Point = (298.47, 208.53) | Squared Error = 61.9201
-------------------------------------------------------------
img20.jpg | Point = (404.88, 287.78) | Squared Error = 1.3092
-------------------------------------------------------------
img21.jpg | Point = (386.50, 209.40) | Squared Error = 8.2243
-------------------------------------------------------------
img3.jpg  | Point = (391.28, 274.19) | Squared Error = 25.5121
-------------------------------------------------------------
img6.jpg  | Point = (380.13, 257.60) | Squared Error = 4.7160
-------------------------------------------------------------
img7.jpg  | Point = (376.18, 426.55) | Squared Error = 20.5457
-------------------------------------------------------------
img9.jpg  | Point = (435.81, 375.41) | Squared Error = 75.5192

MEAN SQUARED ERROR: 41.0148
```

Figure 10: Final result for each image after applying RANSAC

The resulted mean squared error of 41 shows that the chosen parameters helped the RANSAC model work well on this dataset, with most squared errors lying below 100. This proves that the method performed effectively, and have robustness against irrelevant lines.

However, there is one image that has a squared error of approximately 200 units. Looking on this image shows that the there are too few lines drawn in the image, proving that the method did not have enough information to process for this image. This is because the relevant details in this image is small, making it being omitted during edge pre-processing stages such as blurring or length thresholding. It is difficult to balance between this problem and the distracting lines mentioned in the previous section. Furthermore, as mentioned, for images with large number of distracting lines, the randomness of RANSAC may make it more prone to be distracted when running on different random seeds.

When the condition allows, tuning can also be done on RANSAC configurations after each tuned parameter set in the edge detection process. Multiple runs on different random seeds can also be done, in order to make the final outcome converge to a more robust result.