**ECE PROJECT 03- OPTION 2**

*Liwei Qing, 200496083, lqing*

*Gongfan Chen, 200330312, gchen24*

Contribution of the works.

|  |  |  |
| --- | --- | --- |
| Task | Liwei Qing | Gongfan Chen |
| Code | LSD (100%) | RANSAC (100%) |
| Parameter Tuning | LSD (100%) | RANSAC (100%) |
| Report Writing | LSD, Step 4, and Step 5 | RANSAC, Step 3 |
| Total | 50% | 50% |

Step 1: image acquisition.

* A three-point perspective image is acquired, which is named ‘*box.jpg’*.

A box on a table

Description automatically generated with medium confidence

Figure 1. Box

Step 2: computing vanishing points.

Step 2.1 Line segment detection.

* Canny edge detection and Hough Line transformation algorithms are applied to this picture. Fig. 2 shows the algorithm of canny detection.

Diagram

Description automatically generated

Figure 2. Illustration of canny edge detection algorithm

* For the canny edge detection, since edge detection is susceptible to noise in the image, step 1 is to remove the noise with a 5\*5 Gaussian filter; Then the smoothed image is filtered with a Sobel kernel in both horizontal and vertical direction to get first derivative in both directions; After getting gradient magnitude and direction, each pixel is checked if it is a local maximum in its neighborhood in the direction of gradient. If not, the pixel is suppressed; Two thresholds (minimum 60 and maximum 150) are selected to determine whether the selected edges are really edges or not. Any edges with intensity gradient more than maximum are sure to be edges and those below the minimum are sure to be non-edges. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity.
* The canny edge detector returns a binary image, which should be further processed by Hough Line transformation to obtain the lines. Here the probabilistic Hough transform is applied, which instead is a more efficient implementation of Hough Line transformation.
* Figure 3.a shows the box image after canny edge detection and figure 3.b shows the lines obtained after Hough line transformation.

Diagram

Description automatically generatedA box on a table

Description automatically generated with low confidence

Figure 3. (a) Images after canny detection and (b) Hough line transformation

* However, we can see from the above two images that there are many unrelated lines in the background, which are greatly decreasing efficiency for further processing. To minimize the influence of the background, a mask is applied to the image before canny detection. Because it is obvious that the box has different colors with those of the background, a mask is designed to filter colors. Figure 4 shows the effect of a color filtering mask, from which we can see that the background is almost removed.

A picture containing text, businesscard

Description automatically generated

Figure 4. Masked image

* Then canny edge detection and Hough line transformation are applied, the results are shown in the Fig.5.

A picture containing text

Description automatically generatedA box on a table

Description automatically generated with low confidence

Figure 5. (a) Images after canny detection and (b) Hough line transformation

* To further improve the adaptability of the color filtering mask, an interface is provided where uses could self-adjust the colors to make the mask more suitable for their purpose. For each pixel, when its HSV color is out of customized color range, the pixel value will be set to 0, otherwise it will be set to 255. By sliding the HSV color trackbars, users could customize their own mask. In Fig. 6, upper-left panel shows the mask, and upper-right panel shows the masked image, and the lower panel shows the trackbar (‘lh’ stands for lower H value; ‘ls’ stands for lower S value; ‘lv’ stands for lower V value; ‘uh’ for upper H value; ‘us’ for upper S value; ‘uv’ for upper V value).

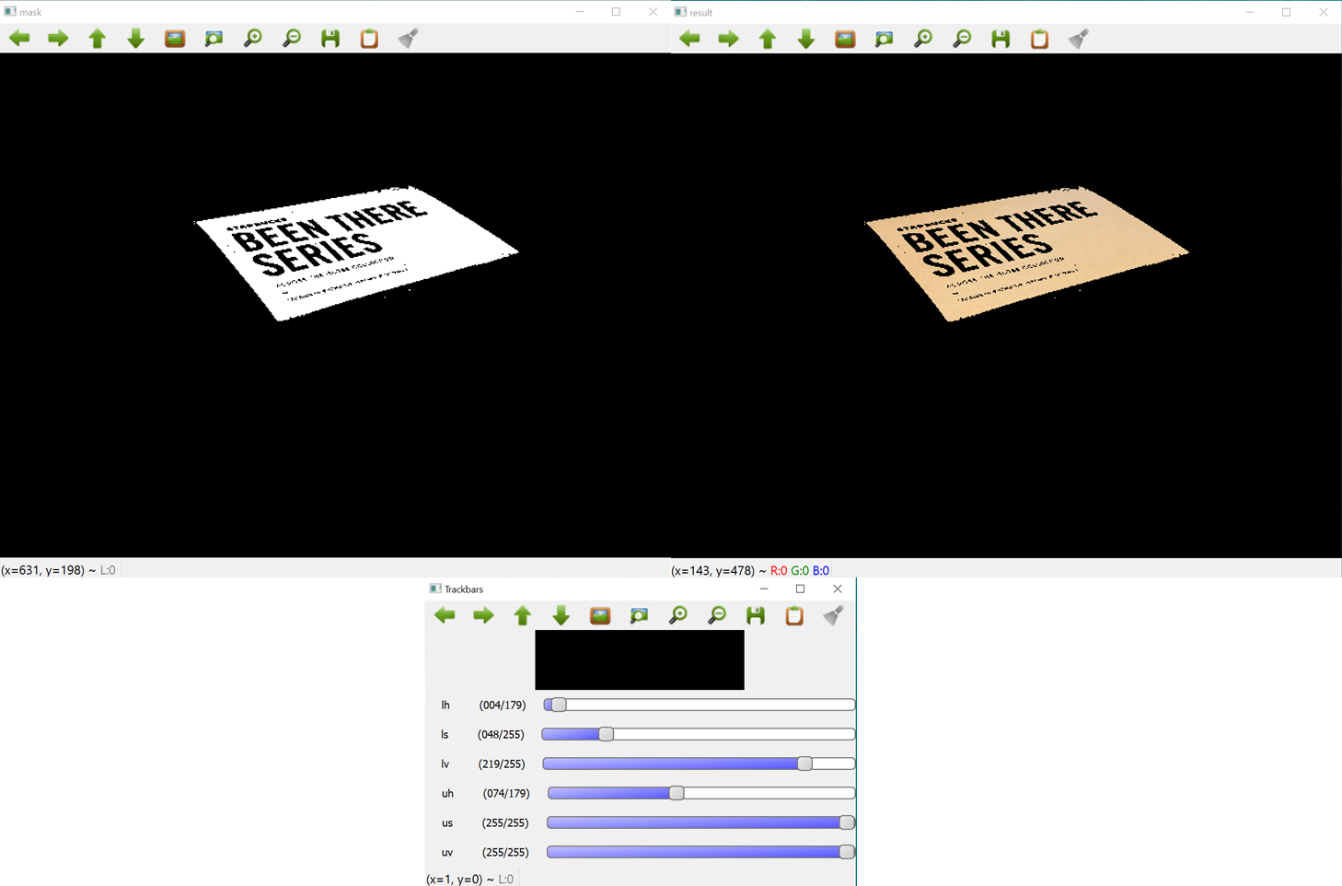


Figure 6. Interface for customized color filtering mask

* **The source code of this line segment detection part can be found at utils.py, lines 33-118.**

Step 2.2 RANSAC algorithm.

* Random sample consensus is applied to determine the optimal vanishing point given various lines detected from LSD. There are four main steps to gain the optimal vanishing point, which is shown in Fig. 7

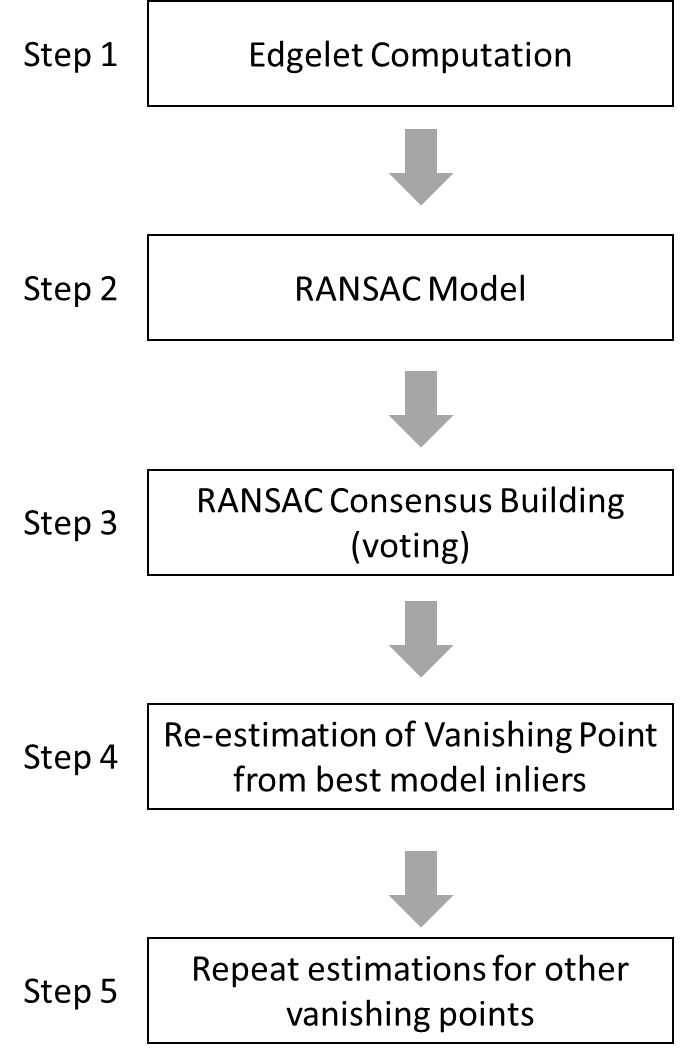


Figure 7. Illustration of generating optimal vanishing point

* An edgelet represents 3 properties of the image: (i) edge location (the image coordinates of the edge point) (ii) edge direction (unit vector along the edge) (iii) edge strength. Edgelets are extracted from only those do not belong to the corners. Edgelets can be computed from the lines detected by LSD, expressed as: , where is the homogenous coordinates for the edge pixel location, is the edge direction in homogenous coordinates (derived from the principal eigen vector of the covariance matrix) and s is the edge strength (principal eigen value of the covariance matrix). An edgelet line, , corresponding to an edgelet E is defined as the line passing through and parallel to . The computed edgelets are stored in an edgelets array, descending sorted on edge strength. A snapshot of generated edgelets is show below.

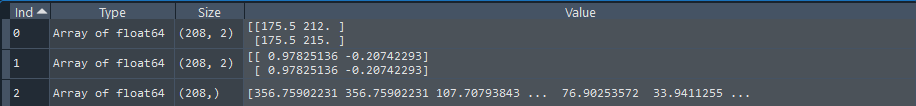


Figure 8. Computed edgelets

* In our RANSAC vanishing point detector, vanishing point can be calculated through randomly selected two edgelets and . Thehe hypothesis vanishing point . For performance reasons, we do not make the model edgelet selection completely random. Instead, we select the first model edgelet randomly from the top 20 percentile of the edgelets array (remember this array was sorted on edge strength). This effectively biases the system towards stronger edges.
* Given model , we iterate over all the other edgelets. Each casts a vote for the model

where is the smaller angle between the voting edgelet line and the line joining that edgelet’s location to the hypothesis vanishing point . is a system parameter. In other words, the vote is proportional to the direction coherence. It attains its maximum value of 1 when the voting edgelet line passes through the hypothesis vanishing point (), dropping to zero as reaches 90 degrees. In practise, we clip the vote to zero when this angle exceeds a threshold (). The hypothesis/model garnering maximal consensus yields the estimated vanishing point. After performing the above two steps, the initial best vanishing point is [-51.2357, -1.53937, -0.0519873]

* After getting the initial best vanishing point, we estimate the optimal (in a least square sense) intersection point for all the inlier edgelet lines corresponding to the best model. Let denote the set of inlier edgelets corresponding to the best model . And, let denote the (as yet undetermined) optimal vanishing point. Ideally, the edgelet line, ~lEi will pass through , yielding · = 0 (in homogenous mathematics, zero dot product indicates line-membership of point [1]). If , we get = 0. Furthermore, we weight each equation by the vote cast by the corresponding edgelet (strong voters pull the solution closer to themselves). Hence, each inlier edgelet to the best model yields an equation of the form where and = . Overall, we end up with the overdetermined homogeneous linear system

which is homogeneous linear system that we solve for = via the well-known *Singular Value Decomposition technique*. After performing this step, the new optimal vanishing point is [1028.29, 26.4387, 1]. The visualization of optimal vanishing point and its inliners are shown in Fig. 9 using the original image.

A picture containing graphical user interface

Description automatically generated

Figure 9. Visualization of optimal vanishing point

* Once the first vanishing point is estimated, we delete all its inlier edgelets from the edgelets array and repeat the process outlined above to estimate the second, third, and other vanishing points.
* In this study, we use the *threshold\_inlier=20*, *threshold\_reestimate=20*, and *threshold\_inlier=20* for RANSAC model initial estimation, model re-estimation, and remove inliners. After calculation, the three vanishing points are located at , for mask images, respectively. The visualizations of vanishing points are shown below. **Please refer to *lqing\_******gchen24\_code.py* lines 21-35 for more details about this part.**

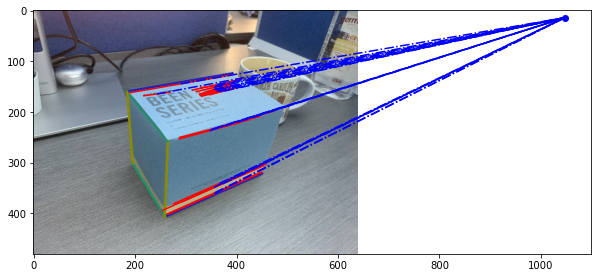


Figure 10. Visualization of first vanishing point

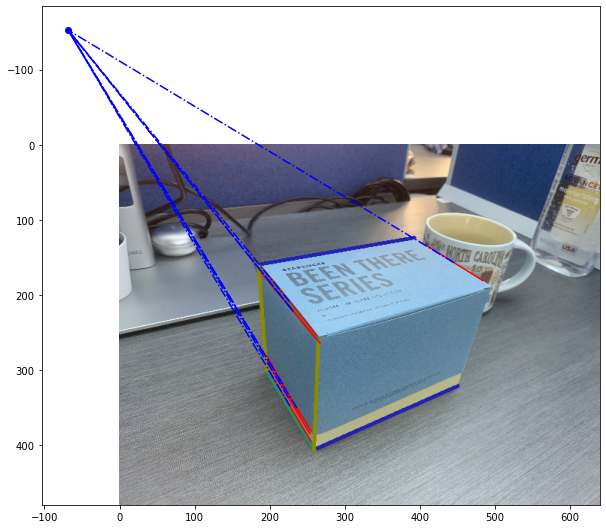


Figure 11. Visualization of second vanishing point

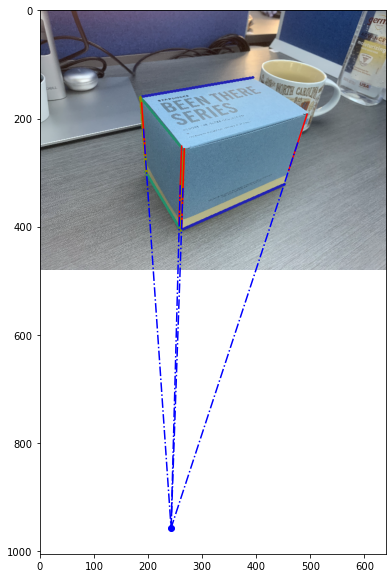


Figure 12. Visualization of third vanishing point

Step 3: computing projection and homography matrix.

* Firstly compute the intrinsic matrix K. Assume the skew coefficient between the x and y axis is 0 and the principal point is ideally in the center of the image, that is, u0 = ½\*image\_width = 320, and v0 = ½\*image\_height = 240. For the intrinsic matrix, we now still have the focal length unknown.

From the following method, find the orthocenter of the triangle ABC (the triangle is made of the three vanishing points).

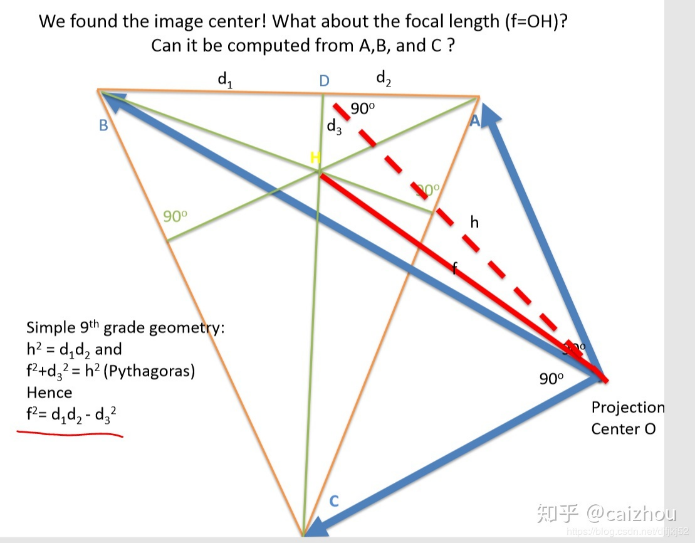
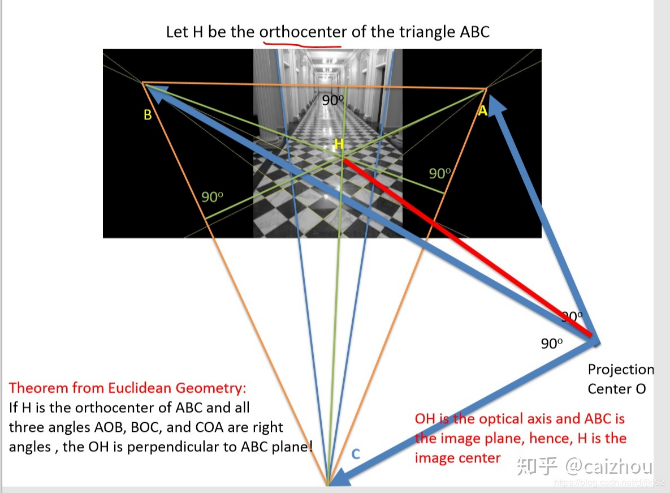


Figure 13. A method to find orthocenter point

Follow the simple geometry, the focal length can be calculated by f2 = d1\*d2-d32.

* Thus we obtain our intrinsic matrix (with f=472)

K =

**Please refer to *lqing\_ gchen24\_code.py* lines 39-45 for more details about this part.**

* Next we compute the extrinsic matrix, which includes rotation and translation. The relationship between object in the world coordinate and image coordinate is:

Where z is a scaling factor, and u v represent point in image coordinate, and X Y Z represent point in world coordinate.

Since we have found three vanishing points, let X at infinity and be the first vanishing point in the world coordinate, and the corresponding coordinate in image system is . So Normalize r1 we could eliminate the effect of scaling factor z. Therefore we have r1 = [ 0.81, -0.25, 0.53]. Similarly, we could get r2 = [-0.54,-0.54, 0.65]. and r3 could be obtain by the cross product of r1 and r2, which is [0.12,-0.81,-0.57]. **Please refer to *lqing\_ gchen24\_code.py* lines 48-55 for more details about this part.**

* To compute the translation vector, we’ll need mapping of at least two points. The first map is world (0,0,0) to image (259,407,1). The second map is world (240,0,0) to image (451,321,1). These values are fixed given the acquired image. The translation vector could be obtained through the two mappings, and the result is T=[-50.56, 138.85, 393.14]. **Please refer to *lqing\_ gchen24\_code.py* lines 57-70 for more details about this part.**
* So the projection matrix is K\*[r1 r2 r3 t], and the result is:
* The homography matrix could be obtained by the projection matrix. For the xy plane, the homography matrix is the 1st, 2nd and 4th columns of projection matrix; for the yz plane, the homography matrix is the 2nd, 3rd, and 4th columns of projection matrix; for the zx plane, the homography matrix is the 1st, 3rd, and 4th columns of projection matrix. **Please refer to *lqing\_ gchen24\_code.py* lines 73-78 for more details about this part.**

Step 4: computing the texture maps.

* Using the homography matrices obtained, we could get projection of each plane.

Text, letter

Description automatically generated

Figure 14. XY plane texture

A close-up of a book

Description automatically generated with low confidence

Figure 15. YZ plane texture

A picture containing text

Description automatically generated

Figure 16. ZX plane texture

Text

Description automatically generated A close up of a person's skin

Description automatically generated with low confidence A close-up of a document

Description automatically generated with medium confidence

Figure 17. Cropped portion of the object

* **Please refer to *lqing\_ gchen24\_code.py* lines 87-100 for more details about this part.**

Step 5: 3D reconstruction.

* Crop the textures of each plane, and use the cropped textures for 3D reconstruction. Please find the *box\_reconstructed.wrl* file in the folder, and readers’ default 3d view software (e.g. Print 3D) could let them view the file. Please note that the texture images (*xy\_texture\_crop.png, yz\_texture\_crop.png, zx\_texture\_crop.png*) are already included in the folder as well, which are need when using the 3D viewing software. The screenshot of the 3D reconstructed object is shown below.

A picture containing text

Description automatically generated

Figure 18. Reconstructed box

**Appendix:**

* Please download all additional packages using the command:

$*pip install -r requirements.txt*

* Code running instruction: one-click on file *lqing\_* *gchen24\_code.py*
* Github: https://github.com/liwei2998/ECE558\_project3\_option2