**ECE PROJECT 01- OPTION 1**

*Liwei Qing, 200496083*

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.

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

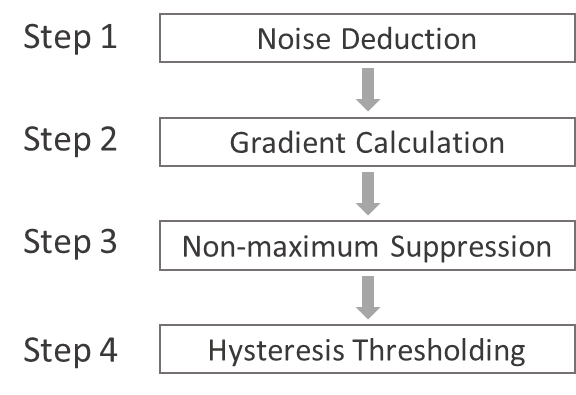


Figure 2. Illustration of canny edge detection algorithm

* 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 medium 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).

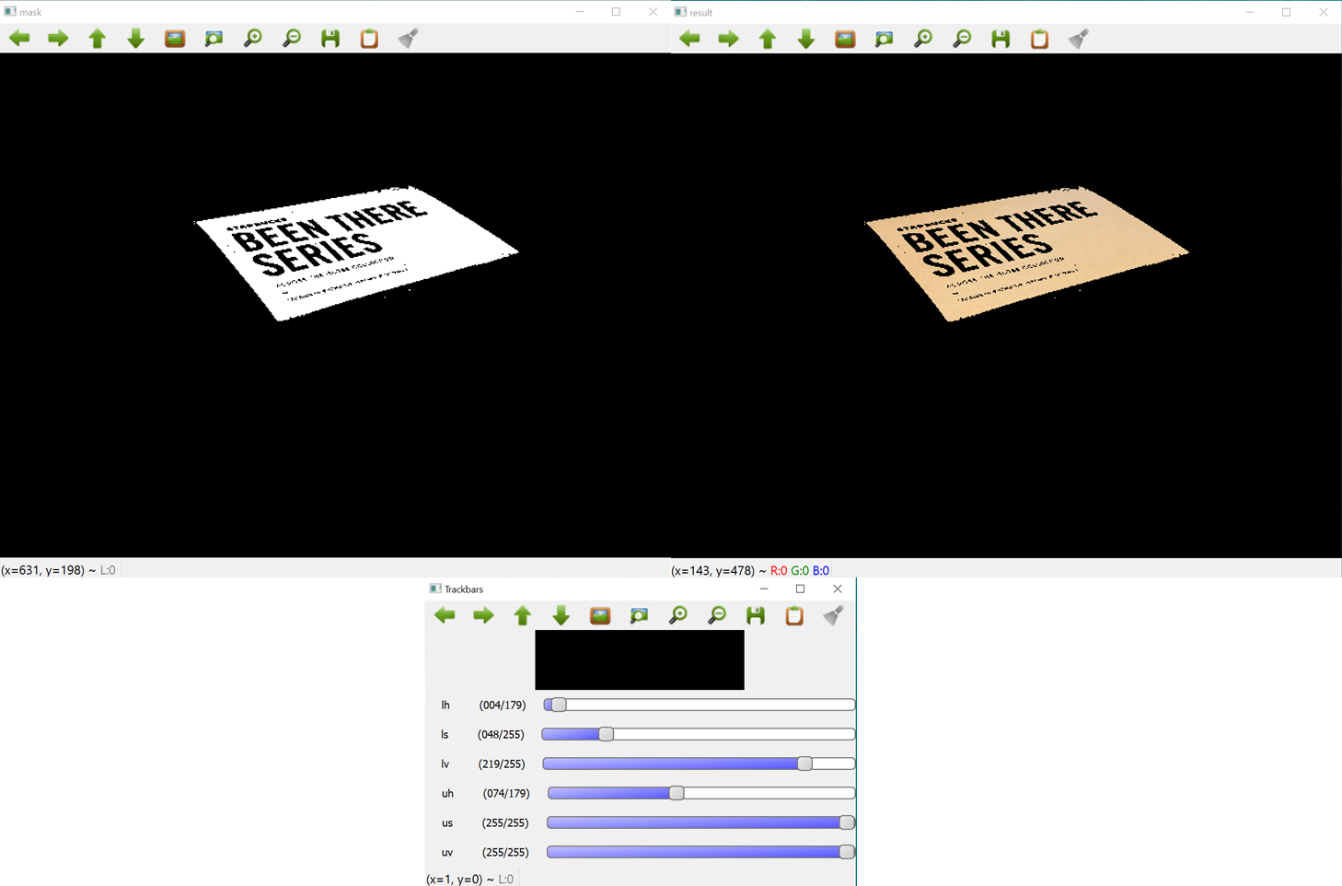


Figure 6. Interface for customized color filtering mask

* A simple annotation interface is written, **named ‘*lineAnnotation.py’***. In this python file, two lines for each of the X (red lines), Y (green lines), and Z (blue lines) directions are drawn. Readers could choose either to draw the annotation lines by themselves (style=’diy’) or use the lines already drawn by the author (style=’default’). The default annotation picture is shown below. You could also see it in the *result images* folder.

A box on a table

Description automatically generated with medium confidence

Figure.2 annotated picture

* According to the method proposed by Prof. Robert, for each line, take the two endpoints e11=(x1i, y1i,1) and e12=(x2i,y2i,1); compute a homogenous coordinate vector representing the line li as the cross product of its two endpoints (ai,bi,ci)=e11 X e12; since we only have two lines representing each direction (XYZ), we can compute a homogeneous coordinate vector V representing their point of intersection as the cross product of these two line vectors V=l1 X l2, scaling V so that the last coordinate is 1, i.e. (Vx, Vy, 1).
* For this picture, the three vanishing points are located at respectively. **Please refer to *lqing\_code*.py lines 44-56 for more details about this part.**

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).

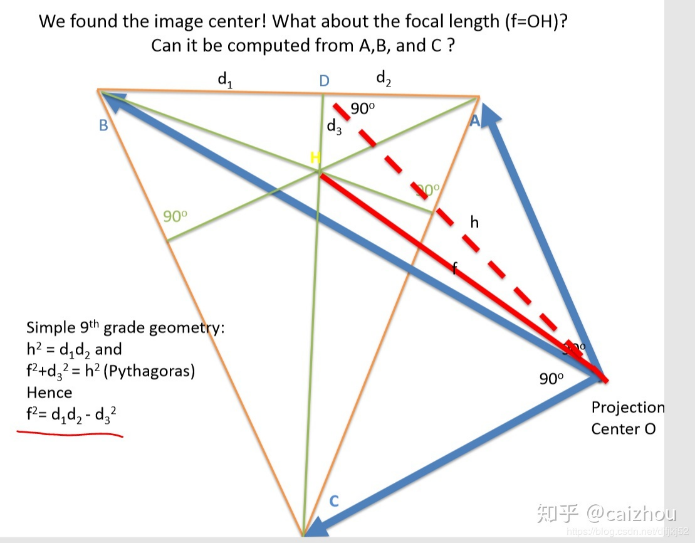
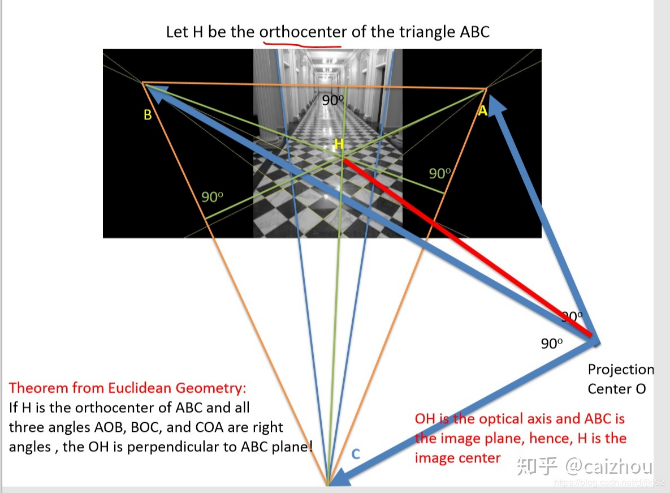


Figure. 3 a method to find orthocenter point

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

* Thus we obtain our intrinsic matrix (with f=493.2)

K =

**Please refer to *lqing\_code.py* lines 59-66 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 = [ 0.85,-0.24,0.47]. Similarly, we could get r2 = [-0.54,

-0.55,0.63]. and r3 could be obtain by the cross product of r1 and r2, which is [0.10,-0.79,

-0.60]. **Please refer to *lqing\_code.py* lines 68-76 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), which is the origin annotated before. The second map is world (240,0,0) to image (451,321,1), which is the point at the end of lower red line in the annotated image. The translation vector could be obtained through the two mappings, and the result is t=[-55.10,151.39,448.00]. **Please refer to *lqing\_code.py* lines 78-93 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\_code.py* lines 95-101 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. 4. XY plane texture

A close-up of a book

Description automatically generated with low confidence

Figure. 5. YZ plane texture

A picture containing text

Description automatically generated

Figure. 6 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. 7 cropped portion of the object

**Please refer to *lqing\_code.py* lines 103-119 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. 8. Reconstructed box