University of Tehran

Electrical and Computer Engineering Department

HW2 Machine Vision

Amin Fadaeinejad 810195442

Fall 2020

Abstract

In this computer assignment we are going to use different methods in order to detect edges and lines in particular images. And also we are going to use them in several applications in machine vision for example in face recognition and panorama picture making.

1. In this Question we are going to explain two methods about edge detection and implement one of them.

   (a) Canny Edge Detector
   The Canny edge detection algorithm is composed of 5 steps:
   Noise reduction: One way to get rid of the noise on the image, is by applying Gaussian blur to smooth it. To do so, image convolution technique is applied with a Gaussian Kernel (3x3, 5x5, 7x7 etc…). The kernel size depends on the expected blurring effect. Basically, the smallest the kernel, the less visible is the blur. In our example, we will use a 5 by 5 Gaussian kernel.

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2}\right); 1 \leq i, j \leq (2k+1)$$

Figure 1: Gaussian filter kernel equation

Gradient calculation:
The Gradient calculation step detects the edge intensity and direction by calculating the gradient of the image using edge detection operators. Edges correspond to a change of pixels' intensity. To detect it, the easiest way is to apply filters that highlight this intensity change in both directions: horizontal (x) and vertical (y). When the image is smoothed, the derivatives Ix and Iy w.r.t. x and y are calculated. It can be implemented by convolving I with Sobel kernels Kx and Ky, respectively:

$$K_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, K_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}.$$

Figure 2: Sobel filters for both direction (horizontal and vertical)

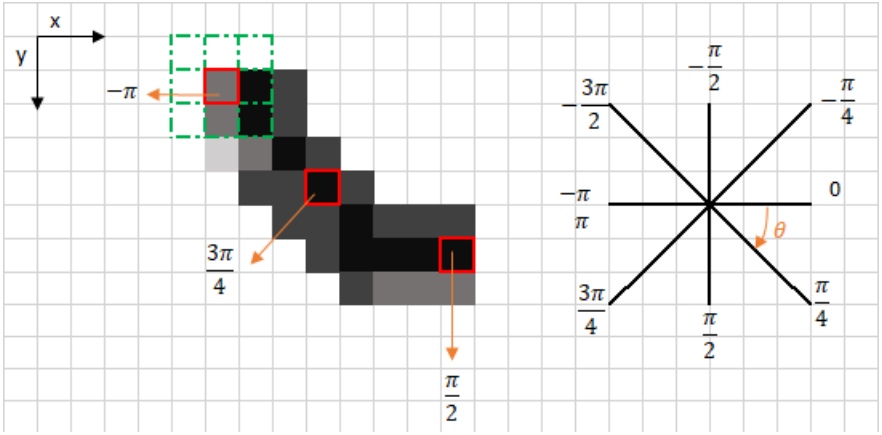
Then, the magnitude G and the slope θ of the gradient are calculated as follow:

$$|G| = \sqrt{I_x^2 + I_y^2},$$
$$\theta(x,y) = arctan\left(\frac{I_y}{I_x}\right)$$

Figure 3: Gradient intensity and Edge direction

The result is almost the expected one, but we can see that some of the edges are thick and others are thin. Non-Max Suppression step will help us mitigate the thick ones.
Moreover, the gradient intensity level is between 0 and 255 which is not uniform. The edges on the final result should have the same intensity (i-e. white pixel = 255).

Non-maximum suppression:
Ideally, the final image should have thin edges. Thus, we must perform non-maximum suppression to thin out the edges.
The principle is simple: the algorithm goes through all the points on the gradient intensity matrix and finds the pixels with the maximum value in the edge directions.



The upper left corner red box present on the above image, represents an intensity pixel of the Gradient Intensity matrix being processed. The corresponding edge direction is represented by the orange arrow with an angle of -pi radians (+/-180 degrees). Double threshold:
The double threshold step aims at identifying 3 kinds of pixels: strong, weak, and non-relevant:

i. Strong pixels are pixels that have an intensity so high that we are sure they contribute to the final edge.
ii. Weak pixels are pixels that have an intensity value that is not enough to be considered as strong ones, but yet not small enough to be considered as non-relevant for the edge detection.

iii. Other pixels are considered as non-relevant for the edge.

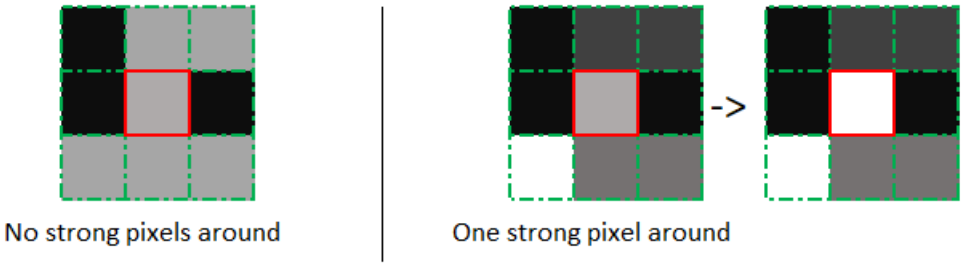
Now you can see what the double thresholds holds for:

i. High threshold is used to identify the strong pixels (intensity higher than the high threshold)

ii. Low threshold is used to identify the non-relevant pixels (intensity lower than the low threshold)

iii. All pixels having intensity between both thresholds are flagged as weak and the Hysteresis mechanism (next step) will help us identify the ones that could be considered as strong and the ones that are considered as non-relevant.

Edge Tracking by Hysteresis:
Based on the threshold results, the hysteresis consists of transforming weak pixels into strong ones, if and only if at least one of the pixels around the one being processed is a strong one, as described below:



(b) Marr-Hilderth
Computing the zero-crossings in the LoG-convolved image (to detect edges as a binary image) was proposed by Marr and Hildreth. Identification of the edge pixels can be done by viewing the sign of the LoG-smoothed image by defining it as a binary image. The algorithm to compute the zero-crossing is as follows:

i. First, convert the LoG-convolved image to a binary image by replacing the pixel values by 1 for positive values and 0 for negative values.

ii. In order to compute the zero-crossing pixels, we need to simply look at the boundaries of the non-zero regions in this binary image.

iii. Boundaries can be found by finding any non-zero pixel that has an immediate ...

Figure 4: Input image

After that we need to apply a 5x5 kernel in order to smooth the image.

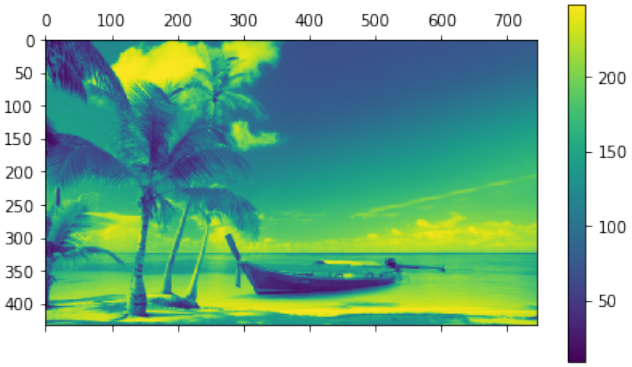


Figure 5: After Filter

After that we will apply Sobel filter to the image and measure the magnitude of the two result.

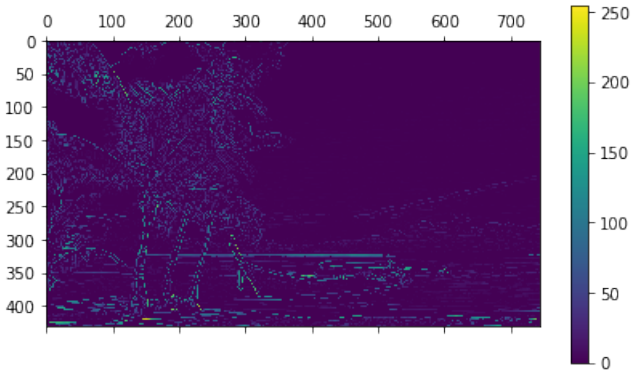


Figure 6: Non Max Suppression output

We will apply a threshold and a hysteresis to see the final result.

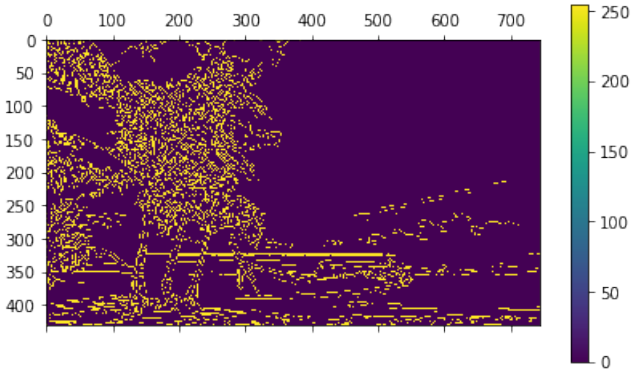


Figure 7: Final output

2. The Hough Transform is an algorithm patented by Paul V. C. Hough and was originally invented to recognize complex lines in photographs (Hough, 1962). Since its inception, the algorithm has been modified and enhanced to be able to recognize other shapes such as circles and quadrilaterals of specific types. In order to understand how the Hough Transform algorithm works, it is important to understand four concepts: edge image, the Hough Space and the mapping of edge points onto the Hough Space, an alternate way to represent a line, and how lines are detected.

   (a) Now we are going to explain how this algorithm works:
   An edge image is the output of an edge detection algorithm. An edge detection algorithm detects edges in an image by determining where the brightness/intensity of an image changes drastically ("Edge Detection — Image Processing with

Python", 2020). Examples of edge detection algorithms include: Canny, Sobel, Laplacian, etc. It is common for an edge image to be binarized meaning all of its pixel values are either a 1 or a 0. Depending on your situation, either a 1 or a 0 can signify an edge pixel. For the Hough Transform algorithm, it is crucial to perform edge detection first to produce an edge image which will then be used as input into the algorithm.

The Hough Space and the Mapping of Edge Points onto the Hough Space:

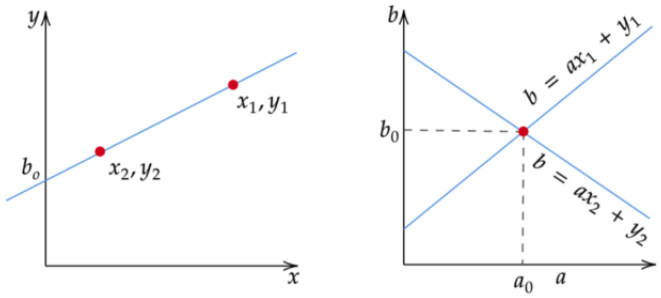


Figure 8: Mapping from edge points to the Hough Space.

The Hough Space is a 2D plane that has a horizontal axis representing the slope and the vertical axis representing the intercept of a line on the edge image. A line on an edge image is represented in the form of y = ax + b (Hough, 1962). One line on the edge image produces a point on the Hough Space since a line is characterized by its slope a and intercept b. On the other hand, an edge point (x , y) on the edge image can have an infinite number of lines pass through it. Therefore, an edge point produces a line in the Hough Space in the form of b = ax + y (Leavers, 1992). In the Hough Transform algorithm, the Hough Space is used to determine whether a line exists in the edge image.
An Alternate Way to Represent a Line

$$m = \frac{rise}{run} = \frac{y_2 - y_1}{x_2 - x_1}$$

m = slope
(x_1, y_1) = first point
(x_2, y_2) = second point

Figure 9: The equation to calculate a slope of a line.

There is one flaw with representing lines in the form of y = ax + b and the Hough Space with the slope and intercept. In this form, the algorithm won't be able to detect vertical lines because the slope a is undefined/infinity for vertical lines (Leavers, 1992). Programmatically, this means that a computer would need an infinite amount of memory to represent all possible values of a. To avoid this issue, a straight line is instead represented by a line called the normal line that passes through the origin and perpendicular to that straight line. The form of the normal line is = x cos( ) + y sin( ) where is the length of the normal line and is the angle between the normal line and the x axis.

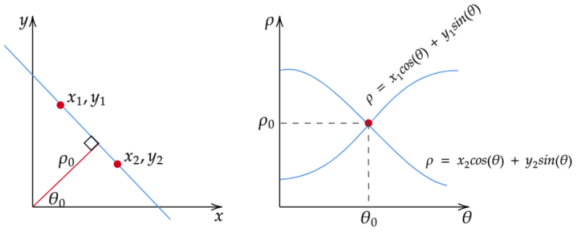


Figure 10: An alternative representation of a straight line and its corresponding Hough Space.

Using this, instead of representing the Hough Space with the slope a and intercept b, it is now represented with and where the horizontal axis are for the values and the vertical axis are for the values. The mapping of edge points onto the Hough Space works in a similar manner except that an edge point (x , y) now generates a cosine curve in the Hough Space instead of a straight line (Leavers, 1992). This normal representation of a line eliminates the issue of

unbounded value of a that arises when dealing with vertical lines.

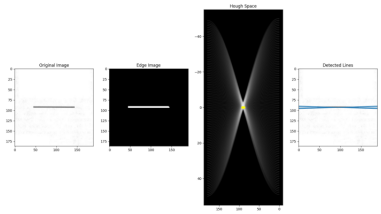


Figure 11: The process of detecting lines in an image.

The Algorithm

i. Decide on the range of ρ and θ . Often, the range of is [0, 180] degrees and is [-d, d] where d is the length of the edge image's diagonal. It is important to quantize the range of and θ meaning there should be a finite number of possible values.

ii. Create a 2D array called the accumulator representing the Hough Space with dimension (num ρ, num θ) and initialize all its values to zero.

iii. Perform edge detection on the original image. This can be done with any edge detection algorithm of your choice.

iv. For every pixel on the edge image, check whether the pixel is an edge pixel. If it is an edge pixel, loop through all possible values of θ, calculate the corresponding ρ, find the θ and index in the accumulator, and increment the accumulator base on those index pairs.

v. Loop through all the values in the accumulator. If the value is larger than a certain threshold, get the ρ and θ index, get the value of ρ and θ from the index pair which can then be converted back to the form of y = ax + b.

Conclusion

To conclude, this article showcased the Hough Transform algorithm in its simplest form. As mentioned, this algorithm can extend beyond detecting straight lines. Over the years, many improvements have been made to this algorithm that allow it to detect other shapes such as circles, triangles, and even quadrilaterals of specific shapes. This resulted in many useful real world applications ranging from document scanning to lane detection in self-driving cars. I'm confident that many more amazing technologies will be powered by this algorithm in the foreseeable future.

(b) In this part we are going to use the Hough transform.

Figure 12: The main images

We will apply the edge detector function from OpenCV in order to detect the edges of the image.

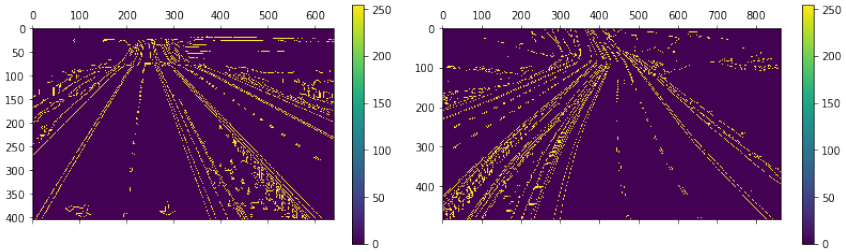


Figure 13: The edge of the image

The first result is in figure 14 where the threshold in 1000.

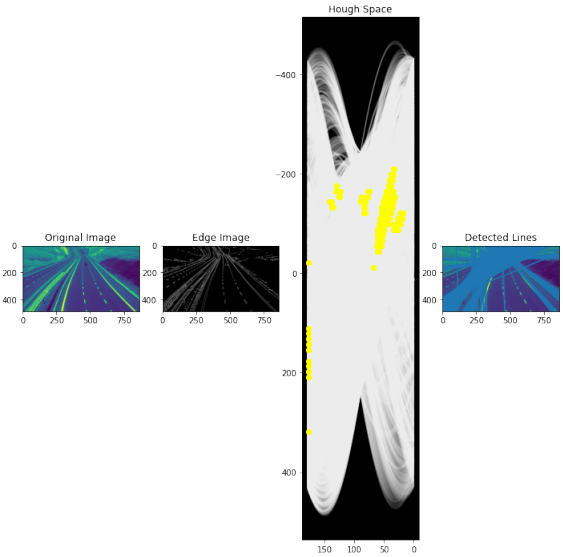


Figure 14: The output of the method

For this result we applied 700 as the threshold.

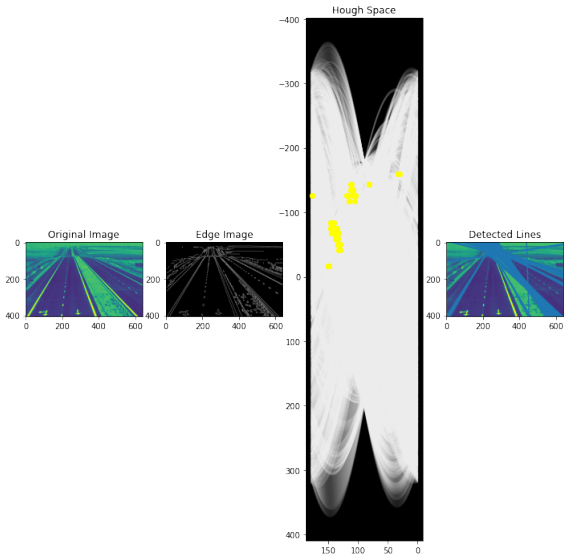


Figure 15: The output of the method

Here is a closer view for the two outputs of the network. For figure 16 changed the threshold to 1100 in order to have a cleaner result.

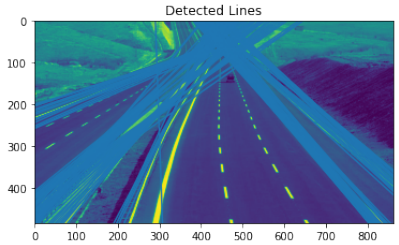


Figure 16: First

Figure 17 shows the output of the method for the second picture.

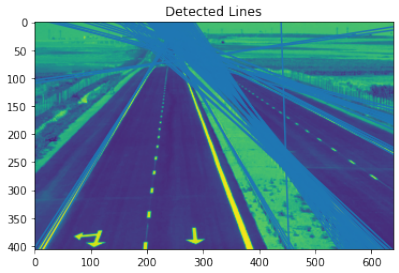


Figure 17: Second picture output

As we can see in the result we were able to get the lines for both images and by changing the threshold we can change the blue lines.

(c) Random sample consensus (RANSAC) is an iterative method to estimate parameters of a mathematical model from a set of observed data that contains outliers, when outliers are to be accorded no influence on the values of the estimates. Therefore, it also can be interpreted as an outlier detection method.[1] It is a non-deterministic algorithm in the sense that it produces a reasonable result only with a certain probability, with this probability increasing as more iterations are allowed. The algorithm was first published by Fischler and Bolles at SRI International in 1981. They used RANSAC to solve the Location Determination Problem (LDP), where the goal is to determine the points in the space that project onto an image into a set of landmarks with known locations.
A basic assumption is that the data consists of "inliers", i.e., data whose distribution can be explained by some set of model parameters, though may be subject to noise, and "outliers" which are data that do not fit the model. The outliers can come, for example, from extreme values of the noise or from erroneous measurements or incorrect hypotheses about the interpretation of data. RANSAC also assumes that, given a (usually small) set of inliers, there exists a procedure which can estimate the parameters of a model that optimally explains or fits this data.

i. The Hough transform would be a reasonable approach for this. Another reasonable approach would be to find connected regions of similar hue, and for each, test if it is approximately pool-ball-shaped.

ii. RANSAC doesn't seem like a good tool for this purpose. RANSAC is commonly used to find, e.g., a line that approximately goes through a bunch of points (but possibly with a few outliers that might not fit the line). Here, the pool balls are spheres, not lines.

iii. RANSAC is typically used for other tasks: e.g., for aligning two images. However, here image alignment probably isn't a good way to find pool balls

on a pool table.

3. In this part we are going to use the SIFT in order to extract the key points and after that we are going to use that in order to find the match for a image.

   (a) we divide the data-set to two parts one is train which we are going to use them in order to find the best threshold for the train data and after that we will apply that to the Test data.

   (b) In this part we will calculate the distance between the key points of the images and if the distance is in a particular range which we apply by assuming a threshold then we can count the number of key points that are similar and after that we will sort the number of similar key points and find the closest ones to the current image.
   We applied different thresholds in order to see the best result.

Table 1: Your first table.

Value 1	Value 2	Value 3
Index	*Threshold*	*Accuracy*
1	0.6	% 57
2	0.65	% 58
3	0.7	%62
4	0.75	%59

And we will show the result for the test data for Threshold=0.7.

True Label	Accuracy
1	0.600000
2	0.557143
3	0.914286
4	0.495238
5	0.642857
6	0.409524
7	0.614286
8	0.461905
9	0.457143
10	0.366667
11	0.504762
12	0.733333
13	0.623810
14	0.390476
15	0.742857
16	0.542857
17	0.780952
18	0.366667
19	0.409524
20	0.366667
21	0.538095
22	0.609524
23	0.819048
24	0.666667
25	0.523810

Figure 18: The accuracy for each person

Figure 18 represents the accuracy for every person in the test data base. For example for each image of a person there are a number of true predictions and wrong predictions, we will sum up all of them and get the accuracy.

4. In this part we are going to use the feature points of an image in order to make a panorama picture.
   Our panorama stitching algorithm consists of four steps:
   Step 1: Detect keypoints (DoG, Harris, etc.) and extract local invariant descriptors (SIFT, SURF, etc.) from the two input images.
   Step 2: Match the descriptors between the two images.
   Step 3: Use the RANSAC algorithm to estimate a homography matrix using our matched feature vectors.
   Step 4: Apply a warping transformation using the homography matrix obtained from Step 3.

   (a) In the first part we are going to use the Harris algorithm in order to find the feature points. And after that we got figure 19 and 20 as the result.

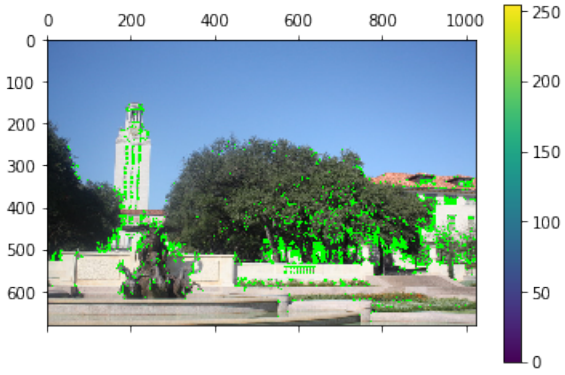


Figure 19: Feature points first picture

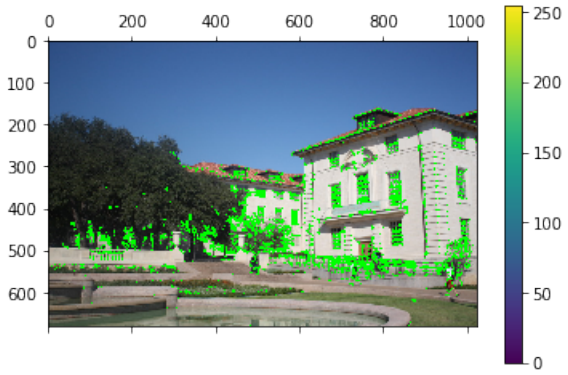


Figure 20: Feature points second picture

(b) In the second part we are going to compare the key points of the two image and declare which one is the closest to which one.

(c) In this part we will find the distance between the key points of the image the ones that are close to each other we assume that they represent the same point in the images by applying a certain threshold for the distance.

(d) In this part we are going to use the Affine transform and the Homography mapping in order to see the result for the task. Figure 21 and 22 will show the result for this thing.
RANdom SAmple Consensus or RANSAC is an iterative algorithm to fit linear models. Different from other linear regressors, RANSAC is designed to be robust to outliers.
Models like Linear Regression uses least-squares estimation to fit the best model to the data. However, ordinary least squares is very sensitive to outliers. As a result, it might fail if the number of outliers is significant.
RANSAC solves this problem by estimating parameters only using a subset of inliers in the data. The figure below shows a comparison between Linear Regression and RANSAC. First, note that the dataset contains a fairly high number of outliers.
We can see that the Linear Regression model gets easily influenced by the outliers. That is because it is trying to reduce the average error. Thus, it tends to favor models that minimize the overall distance from all data points to the model itself. And that includes outliers.
On the contrary, RANSAC only fits the model on the subset of points identified as the inliers.
This characteristic is very important to our use case. Here, we are going to use RANSAC to estimate the Homography matrix. It turns out that the Homogra-

phy is very sensitive to the quality of data we pass to it. Hence, it is important to have an algorithm (RANSAC) that can filter points that clearly belong to the data distribution from the ones which do not.

(e) In the end we will plot the result of the combination of the two images we will find the best way to concat the two images by calculating the distance of the key points, since we want to minimum the error and we are linear space for changing the images so the best way to reduce the error is to use the average of the distance, and this was we done by calculating every vector of the key points and calculated the average of all of them.

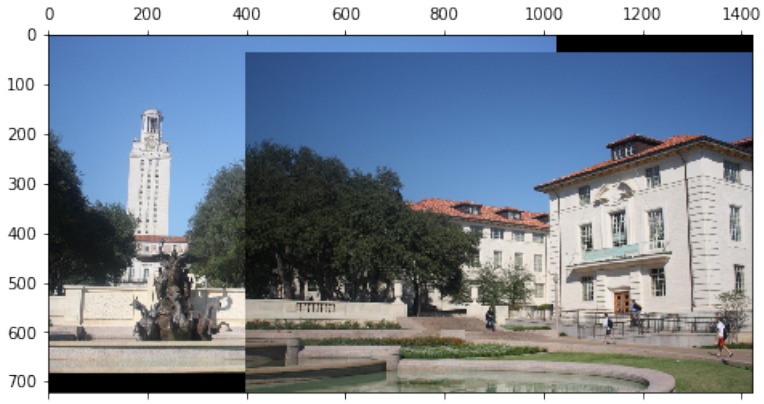


Figure 21: Affine Transform

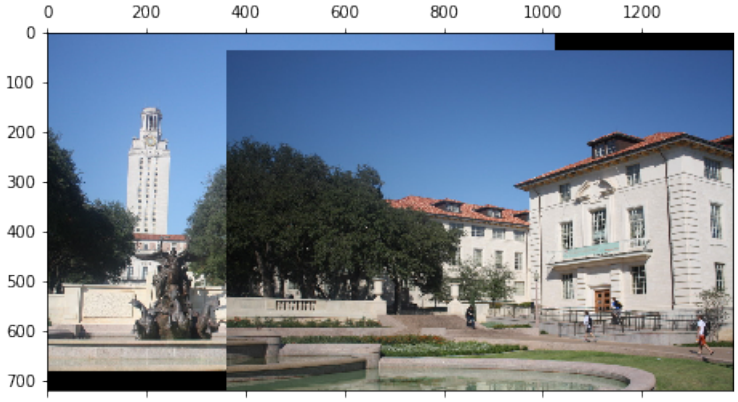


Figure 22: Homography mapping

Reference

1. Canny Edge Detection Step by Step in Python — Computer Vision
2. Edge detection with the Marr and Hildreth's algorithm using the zero-crossing computationm
3. Lines Detection with Hough Transform
4. Hough-RANSAC: A Fast and Robust Method for Rejecting Mismatches
5. Image Panorama Stitching with OpenCV
6. Panorama Formation using Image Stitching using OpenCV
