Report: Vanishing point detection

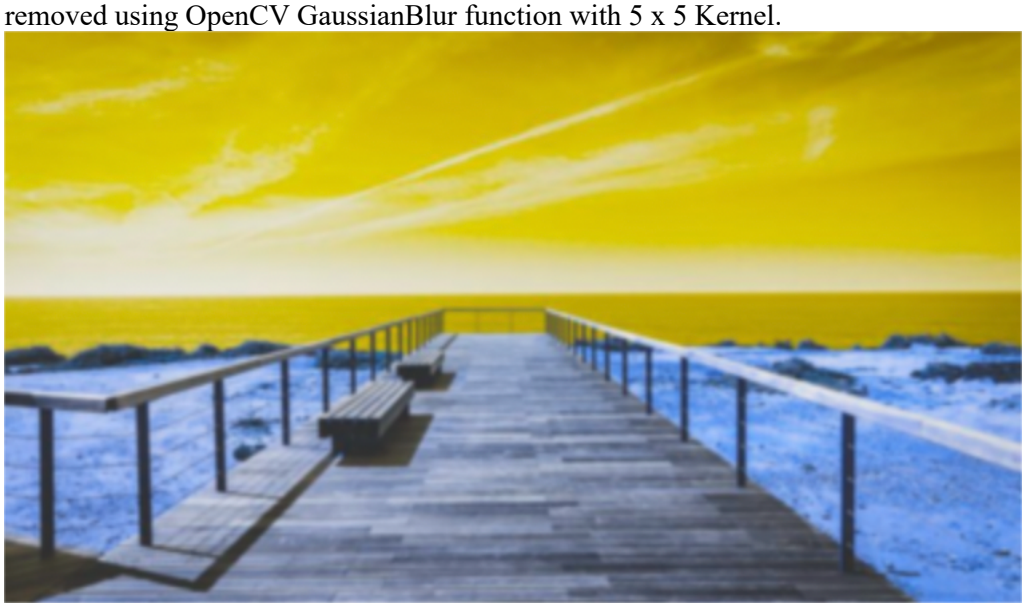
by: Ankita Dhar, Student Number: 1154197, University of Melbourne

** Let us consider image img21.jpg for this report as reference.

1. Detect lines in the image

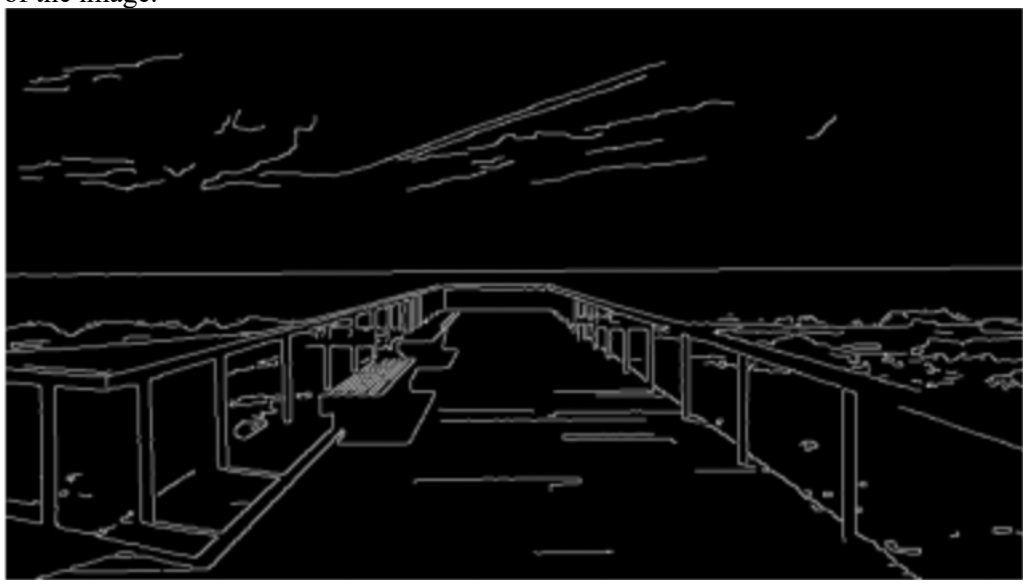
Image Processing

Image is read using OpenCV IMREAD_UNCHANGED parameter. Image noise is removed using OpenCV GaussianBlur function with 5 x 5 Kernel.



Edge Detection

Canny algorithm is run on the processed image to find edges.
It was found that threshold values when adjusted with a gap of 50 to 100 gave a good result. Also, any aperture size other than was increasing the number of texture edges of the image.



Line Formation

Hough Transform algorithm is used on the image from previous step.
With high value of threshold to vote for a line, we reduce the number of lines selected.

Report: Vanishing point detection

by: Ankita Dhar, Student Number: 1154197, University of Melbourne

Filtering Lines	In ` plotLinesToVanishingPt()` vertical and horizontal lines are filtered based on the theta and the slope values of the lines, because they donot help in finding the vanishing point.

2. Locate the vanishing point

2.1.a. find_intersection_point()	This function uses the shapely module in python to find the intersection point of two lines.
2.1.b. find_dist_to_line()	[1] This function computes the distance from a point to a line.
2.2. ransacLoop()	This function runs the main loop of RANSAC algorithm.

Selecting two lines	shuffle all the lines collected from Hough transform and the first two lines are selected to find an intersection point.
Finding inliers	Distance of remaining lines are found from the vanishing point. If the distance is lesser than RANSAC Threshold, the line is considered to be an inlier.
Finding the best Model for Vanishing Point	Previous two steps are performed repeatedly to find the best model.

Report: Vanishing point detection

by: Ankita Dhar, Student Number: 1154197, University of Melbourne

3. Main function and evaluation

Predicted vanishing points are compared against the true vanishing points provided.

Image	Actual Vanishing point	Predicted Vanishing point	Squared Euclidean distance
Img1	(404, 183)	(404.54, 184.70)	3.2
Img3	(396, 276)	(394.56, 276.78)	2.66
Img6	(378, 258)	(378.55, 259.45)	2.42
Img7	(379, 423)	(354.03, 433.36)	730.45
Img9	(438, 367)	(416.74, 343.31)	1012.80
Img10	(240, 466)	(235.76, 469.36)	29.25
Img13	(390, 260)	(391.91, 262.17)	8.41
Img14	(375, 270)	(85.90, 149.79)	98026.79
Img15	(416, 270)	(426.37, 269.19)	108.31
Img18	(292, 213)	(330.08, 622.60)	169228.39
Img20	(406, 288)	(395.86, 285.17)	110.65
Img21	(389, 208)	(380.84, 208.81)	67.16
Mean Squared Euclidean Distance			22444.21

The RANSAC algorithm seems quite effective in detecting the vanishing point. The algorithm works better after removing near vertical and horizontal lines from Hough transform.

The Mean Squared Euclidean Distance is high because the parameters are generic and not tuned for each of the images. Also, there are too many edges detected for some images like img18.jpg which causes formation of lines from Hough transform which incorrectly guides the RANSAC loop to detect vanishing point. Moreover, images like img14.jpg don't have enough edges to guide the RANSAC loop well.

After tuning the parameters for Canny, Hough Transformation and RANSAC algorithms as stated above, we could achieve a MSE of 22444.21.

Report: Vanishing point detection

by: Ankita Dhar, Student Number: 1154197, University of Melbourne

In future, we can explore more of linear perspective of the images and factors like balance, contrast and illuminations to detect vanishing point.

Reference:
[1] week7 workshop solution.