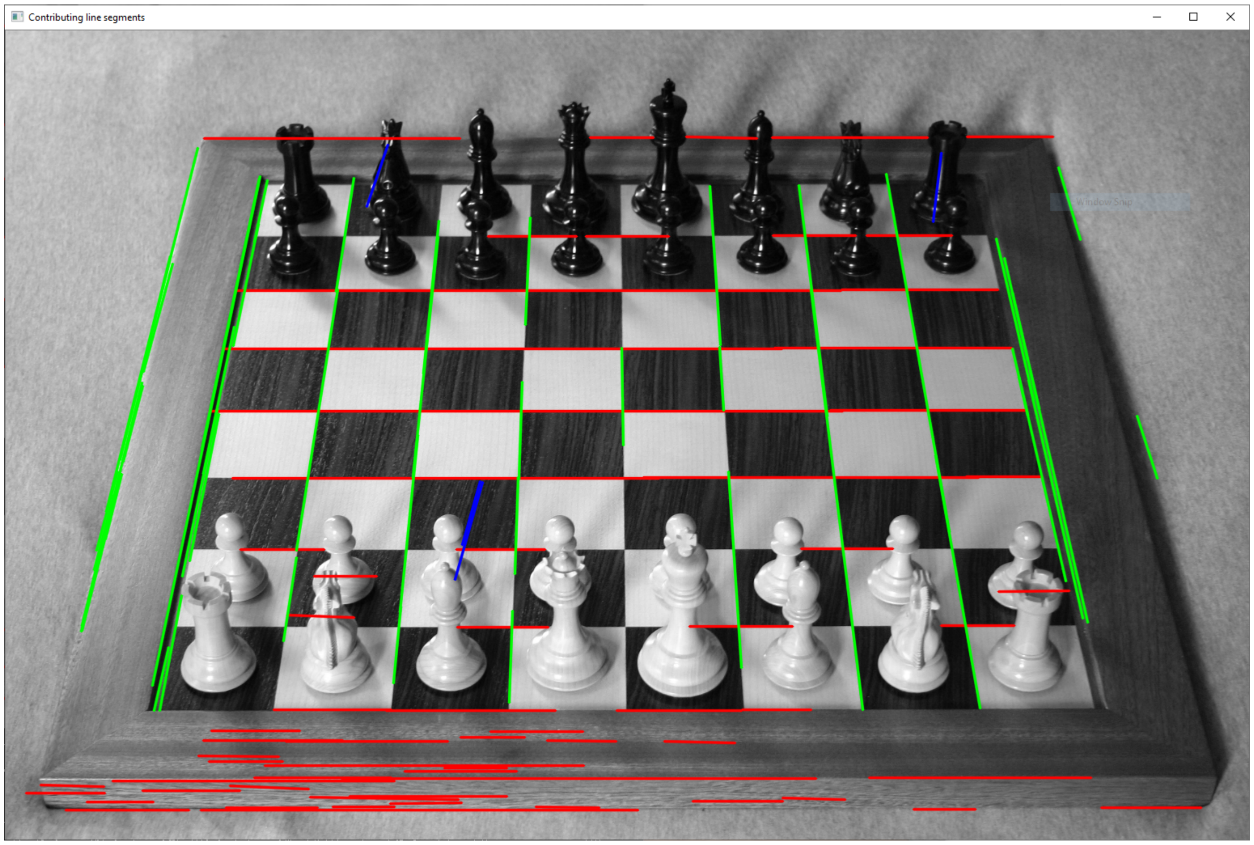
Lab 10: Hough Transforms

Due Nov 9, 2020 by 11:59pm **Points** 10 **Submitting** a file upload

Problem Overview

The goal of this exercise is to find line segments using the Hough transform, and then identify sets of parallel lines in the scene. For example, in the image **chess.jpg** (https://elearning.mines.edu/courses/25410/files/2395090/download?download_frd=1) , two sets of parallel line segments are identified; one set is red and the other is green (there is also a very small set of blue lines).



Hough Transform

As described in the Hough tutorial, OpenCV has several functions to compute the Hough transform. For this lab, use the function “cv2.HoughLinesP()”, which implements the probabilistic Hough Transform. This is a more efficient implementation of the Hough transform, and returns a set of line segments, where each line segment is represented by its endpoints (x1, y1, x2, y2). You can call this function using:

```
# Run Hough transform.  The output houghLines has size (N,1,4), where N is #lines.
# The 3rd dimension has the line segment endpoints: x0,y0,x1,y1.
houghLines = cv2.HoughLinesP(
    image=Iedges,
    rho=1,
    theta=math.pi/180,
    threshold=int(image_width * MIN_HOUGH_VOTES_FRACTION),
    lines=None,
    minLineLength=int(image_width * MIN_LINE_LENGTH_FRACTION),
    maxLineGap=10)
print("Found %d line segments" % len(houghLines))
```

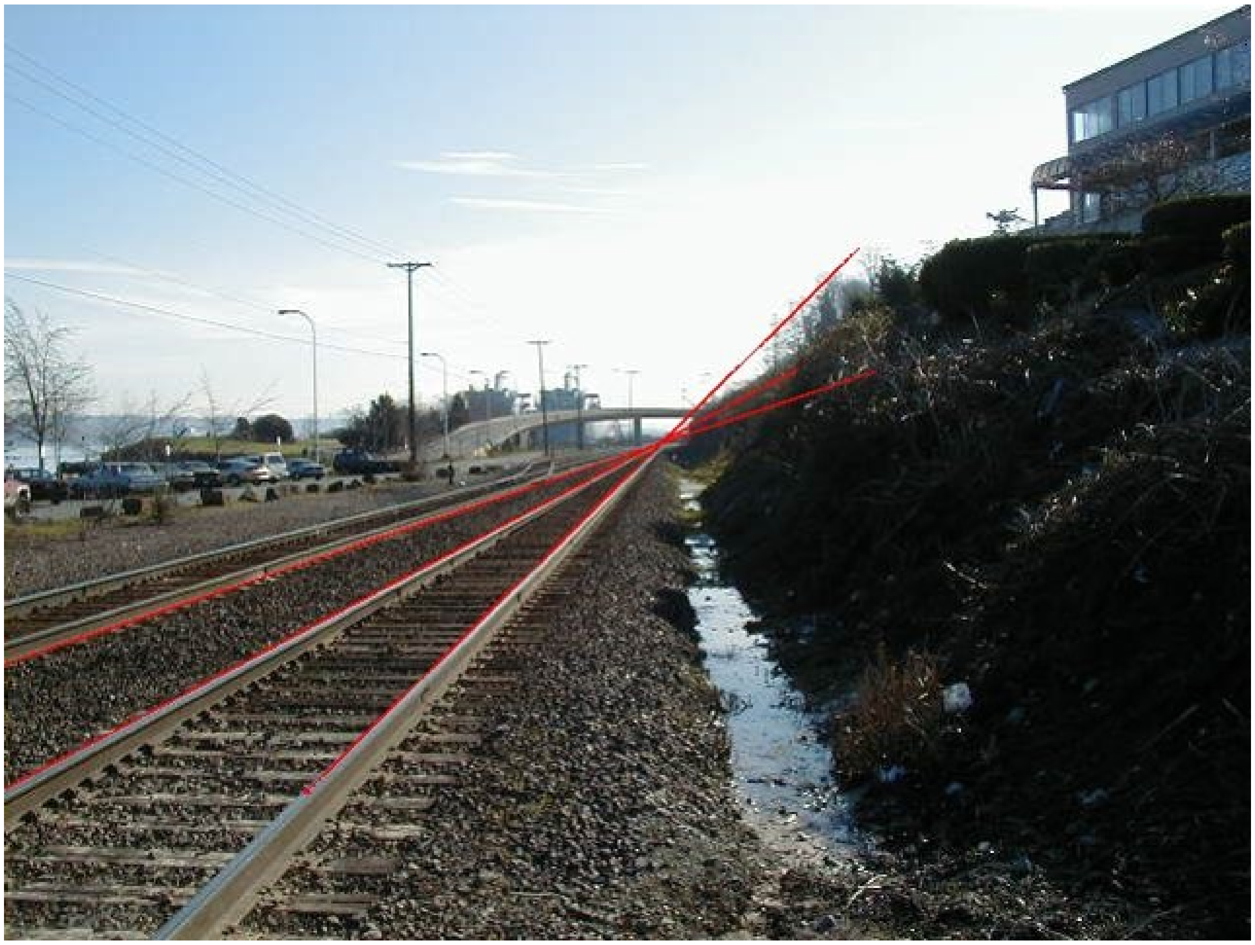
The function takes as input a binary edge produced by the Canny edge detector. The constant MIN_HOUGH_VOTES_FRACTION is multiplied by the image width to determine the Hough accumulator threshold. Only those lines are returned that get enough votes (i.e., greater than the threshold). The constant MIN_LINE_LENGTH_FRACTION is multiplied by the image width to determine the minimum line length that is detected.

To display the line segments, use the following code:

```
# For visualizing the lines, draw on a grayscale version of the image.
bgr_display = cv2.cvtColor(cv2.cvtColor(bgr_img, cv2.COLOR_BGR2GRAY), cv2.COLOR_GRAY2BGR)
for i in range(0, len(houghLines)):
    l = houghLines[i][0]
    cv2.line(bgr_display, (l[0], l[1]), (l[2], l[3]), (0, 0, 255),
             thickness=2, lineType=cv2.LINE_AA)
    cv2.imshow("Hough linesP", bgr_display)
```

Vanishing Points

As described in the Szeliski textbook (Section 4.3.3), a "vanishing point" is a point in the image where lines that are parallel in 3D, appear to meet at infinity. See the railroad tracks image below.



The vanishing point is actually the image projection of a point on one of these parallel lines, infinitely far away. The vanishing point doesn't have to fall within the image. Instead of specifying the vanishing point as an image point, we can specify the 3D direction to the point.

The function "find_vanishing_point_directions()" in the Python file **vanishing.py** (https://elearning.mines.edu/courses/25410/files/2394968/download?download_frd=1) finds vanishing points in an image, and returns their direction vectors. It takes as input the line segments from the Hough transform function. The algorithm is described in the post **https://yichaozhou.com/post/20190402vanishingpoint/**

(https://yichaozhou.com/post/20190402vanishingpoint/) . You can use these directions to reconstruct the orientation of the camera, as described in the Szeliski textbook (Section 6.3.2).

To Do

A set of test images is on the course website: **corridor1.jpg** (https://elearning.mines.edu/courses/25410/files/2394962/download?download_frd=1) , **corridor2.jpg** (https://elearning.mines.edu/courses/25410/files/2394965/download?download_frd=1) , **corridor3.png** (https://elearning.mines.edu/courses/25410/files/2394966/download?download_frd=1) . Process these images and find three sets of parallel line segments.

- Read the image and convert it to gray scale. As suggested in the Hough tutorial, reduce its size if it is large, and blur it slightly to reduce noise.
- Call the Canny edge detector to create an edge image. As suggested in the Hough tutorial, adjust the threshold to obtain a desired number of edges. For this lab, I recommend between 5% and 8% edge density (i.e., MIN_FRACT_EDGES = 0.05, MAX_FRACT_EDGES = 0.08).
- Call "HoughLinesP" to find line segments from the edge image. You will have to experiment with the constants MIN_HOUGH_VOTES_FRACTION and MIN_LINE_LENGTH_FRACTION. Adjust these to get 150 or so line segments.
- Call "find_vanishing_point_directions()", to produce an image with red, green and blue overlay segments, such as shown in the chess image above. The program should find what appears to be three orthogonal sets of line segments.

Upload

- Your Python program.
- For each of the three test images: the edge image, the complete set of line segments as found by HoughLinesP, and the final output overlay image with the red, green, and blue segments.