Project for undergraduate students registered in ELEC 444

Implement a line-finder using RANSAC. Submit **one file** called `RansacLine.m` (**do not submit or modify Demo2.m**). Call this function:

```
lines = RansacLine(edgeImageIn, noIter, fitDistance, noPts, minD)
```

`lines` is an n by 3 matrix parameterizing lines in the plane

`edgeImageIn` is a binary edge image

`noIter` is the number of iterations that you have to pick 2 points at random

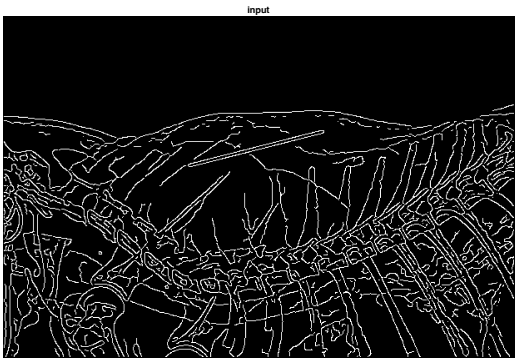
`fitDistance` is the maximum distance a pixel may lie from a line

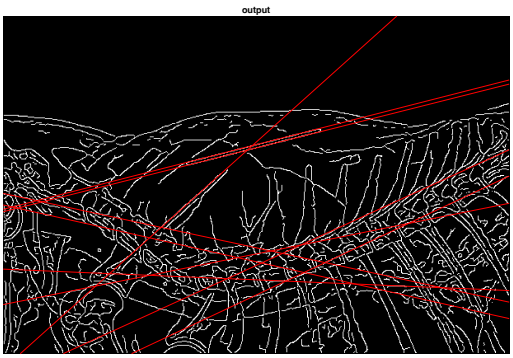
`noPts` is the minimum number of points that should vote for a line. Note that this is different from the implementation discussed in class where we pick the line with max votes. Here, we pick lines that have votes greater than `noPts`

`minD` is the minimum distance between the 2 randomly selected points. This improves RANSAC's performance because if the 2 original points are close, the line fitted can have inaccurate slope.

Hint hints for your `RansacLine` function**:** Use the matlab function "find" to get the coordinates of all of the pixel locations corresponding to an edge.

The input and output to your file are below.





And the following output is printed after running the code:
line #1: 0.0043569X + -0.00098433Y = 1
line #2: 0.0032443X + -0.00013411Y = 1
line #3: 0.0023285X + 0.0020893Y = 1
line #4: 0.0022508X + 0.0010309Y = 1
line #5: 0.0028469X + 0.0005698Y = 1
line #6: 0.0046447X + -0.001006Y = 1
line #7: 0.0020629X + 0.00097296Y = 1
line #8: 0.0041957X + 0.001061Y = 1
line #9: 0.0042487X + 0.001086Y = 1
Note that the number of lines can be different if you run the code again. For example, if we run the same code without changing anything, we might get 6 lines.
To make sure your code is correct, you can test your code with other images, such as:

```
inIM = imread('circuit.tif');
inIM = imread('gantrycrane.png');
```