Homework 2

Due Date: March 12, 2024 at 3pm Montreal time

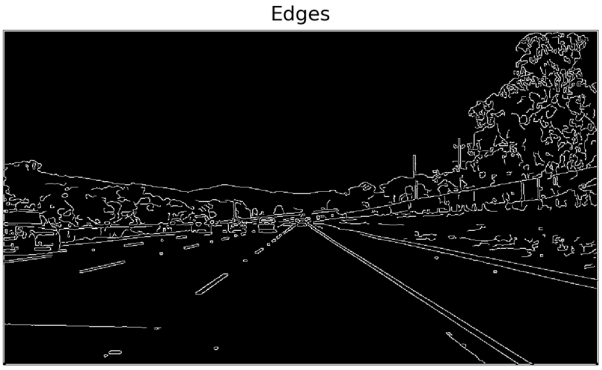
Note

- You are encouraged to meet with other students to discuss the homework, but all write-ups and codes must be done on your own. Do not take notes from those meetings. You should know how to work out the solutions by yourself.
- Please acknowledge other students with whom you discussed the problems and what resources (other than the instructor/TAs, lecture notes, and textbook) you used to help you solve the problem. This won't affect your grade.
- For programming questions, please make sure your code is clearly commented on and easy to read. If the marker cannot understand your code and it does not run correctly, you might not be able to get any partial marks.
- Follow the instructions exactly. We reserve the right to refuse to grade the homework or deduct marks if the instructions are not followed.
- Use your grace days wisely.
- Make sure your code is sufficiently optimized. In particular, do not use unnecessary loops.
- Some of the questions require setting certain parameters. You should play with these parameters to find suitable values.
- **DO NOT use advanced functions (e.g. from skimage, OpenCV) unless they are explicitly allowed. The goal of the homework is for you to implement certain algorithms from scratch.**

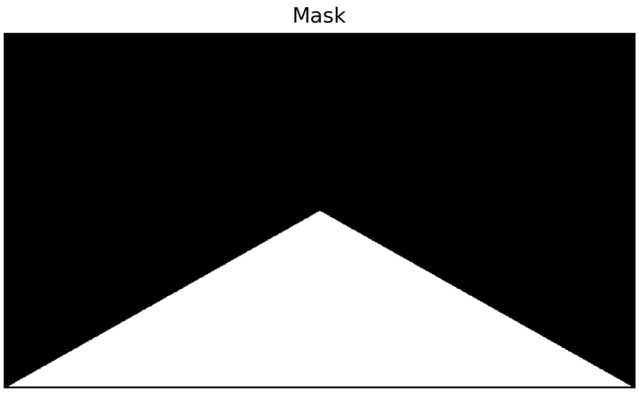
Part 1 (10 pt): Hough transform

In this problem, you will implement Hough transform to find straight lanes in an image. You need the code/data under the "hough" folder. For this problem, you can use existing Canny edge detector in skimage or OpenCV. Write a python script called "hough.py" that does the following:

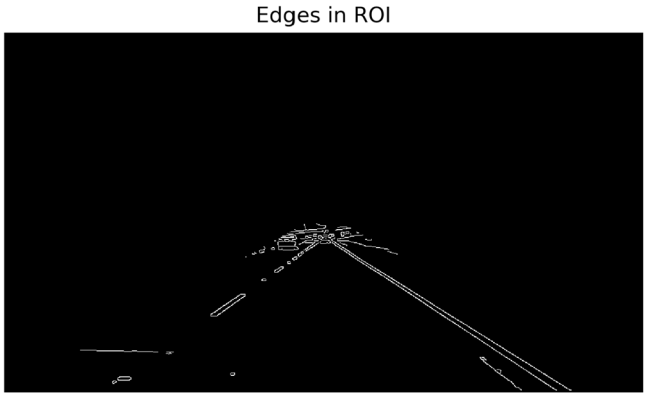
Step 1: load the image "road.jpg", convert it to grayscale, then run Canny edge detector to find edges. Your result should probably look like this:



Step 2: For this homework, we will only work with a subset of these edge points within an ROI (region of interest). The provided function "create_mask" in "utils.py" will create a binary mask like this:



By multiplying this mask with the edge map, you can get an edge map containing only those edges in the ROI as below:



Step 3: Now implement the Hough transform to find two major lanes. Note that you should use the polar representation (ρ,θ) as the Hough parameter space. Find the cell with the highest value in the Hough space, which will give you the lane in blue. You might find the function "create_line" in "utils.py" useful for converting a point (ρ,θ) in Hough space into a line in (x,y) space which can be used for plotting.

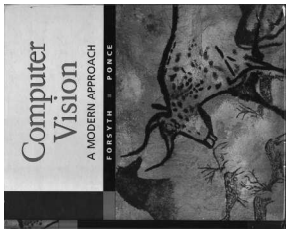
After finding the blue lane, you should apply non-max-suppression to suppress the values of its neighboring cells. Then find the next cell with the highest value in the Hough space, which will give you the orange lane. Note that without NMS, the orange lane you get will be very close to the blue lane. Your result should look like this:

You can choose free parameters in your implementation. Your code should plot the above four figures. **Put these 4 plots in your report.**

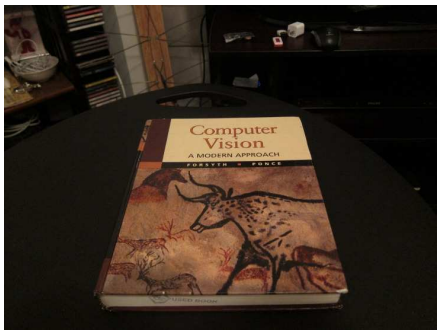
Part 2 (20pt) RANSAC, Homography

In this problem, you will implement the code for estimating the homography, and use the estimated homography in a simple AR (augmented reality) application.

You can find code/data for this problem in "homography" folder. You are given 3 images as follows:



cv_cover



cv_desk

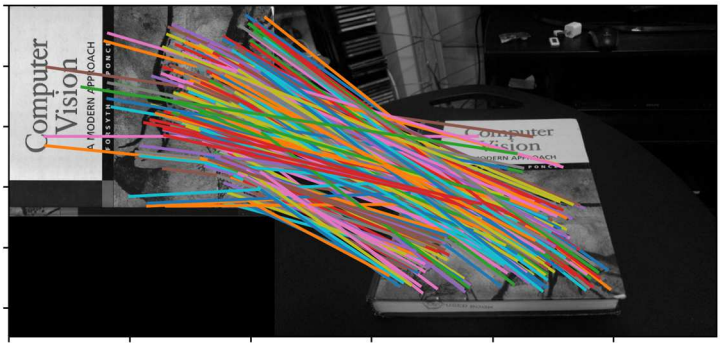
hp_cover

Your task is to estimate the homography between "cv_cover" and "cv_desk". Then use the estimated homography to warp "hp_cover" appropriately and composite with "cv_desk", so that the Harry Potter bookcover completely covers the Computer Vision bookcover in "cv_desk" (you can check out the "final result" figure at the end of this section). This involves completing the code for the following functions in "homography.py"

matchPics(I1, I2): This function takes two images "I1" and "I2" input. It runs SIFT on both images and find candidate matching points between two images. You can use existing SIFT feature detector, descriptor, and matching functions in skimage or OpenCV in your code. This function has 3 outputs: matches, locs1, locs2. The details of these variables are as follows:

- locs1 stores the locations of keypoints in I1. (locs1[i,0], locs2[i,1]) give the (row, column) number of the i-th keypoint.
- locs2 similarly stores the locations of keypoints in I2
- matches store the candidate matching keypoint pairs between these two images. Each row of matches stores the matching information of which keypoint in I1 is matched to which keypoint in I2. For example, if there are 100 matches in total, matches will be 100x2 array. For the i-th matching keypoint pair, matches[i,0] gives the index of the keypoint in I1 and matches[i,1] gives the index of the keypoint in I2.

The following figure ("raw matching result") shows a visualization of the matching result:

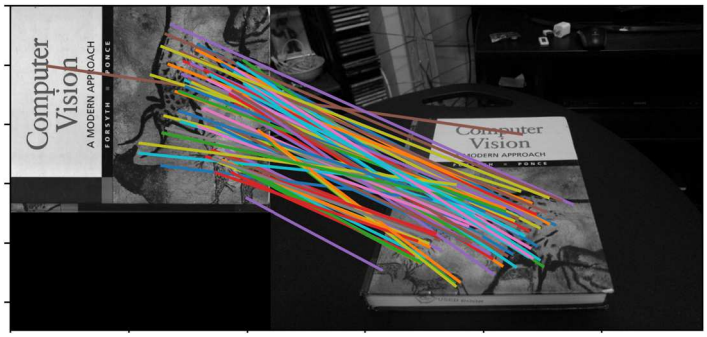


raw matching result

computeH_ransac(matches, locs1, locs2): The raw matching result contains outliers. This function implements the RANSAC algorithm for estimating the homography. You are required to implement this function from scratch using only Numpy (e.g. do not use any additional function from skimage, OpenCV, or other libraries). You might find the numpy.linalg to be useful, e.g. for doing SVD decomposition. This function has 2 outputs: bestH, inliers as follows:

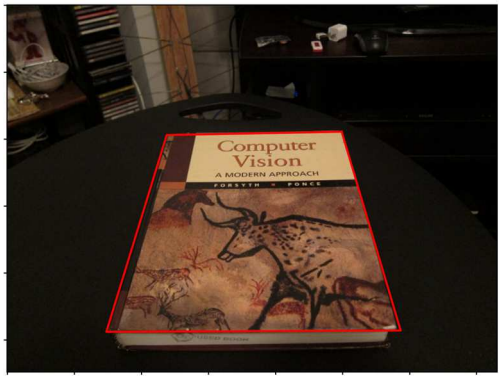
- bestH: this is a 3*3 matrix representing the estimated homography matrix
- inliers: this is an array indicating which candidate match is an inlier. For example, if there are 20 inliers, inliers will be an array of length 20. If inliers[0]=5, it means the first inlier is given by matches[5,:]

The following figure ("matching result after RANSAC") visualizes only those inlier matches:



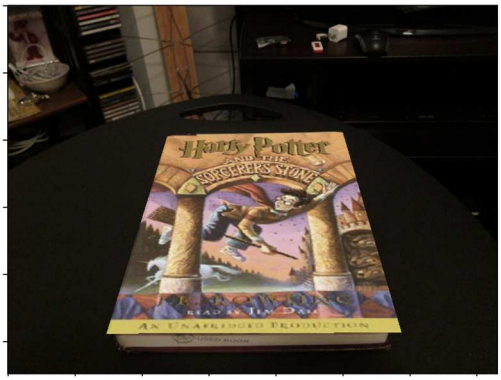
matching result after RANSAC

Once you have the homography matrix, you can create a bounding box of cv_cover and apply the homography on the coordinates of the bounding box. This will give you the corresponding bounding box in cv_desk visualized as follows ("visualization of bounding box"):



visualization of bounding box

compositeH(H, template, img): This function uses the estimated homography H to warp a template image ("template") with a target image ("img"), then return the composite image. You can either write the warping function yourself, or use existing warping function in skimage/OpenCV for this (but you must use the homography matrix estimated from your code for the warping). For example, if the template is "hp_cover" (note that you probably want to first resize "hp_cover" to have the same size of "cv_cover") and the target image is "cv_desk", the following figure ("final result") shows what the result looks like:



final result

If all your implementations are correct, executing "run.py" will generate the above figures. **Put these plots in your report**.

How to Submit

Please create a document in PDF format (name it as hw2.pdf) containing the following:

- Your name, student ID, email address
- Write down the estimated homography (3*3 matrix) in the report
- Relevant plots/figures requested in this homework. Make sure you organize and label the plots/figures appropriately, so that it is easy for the makers to find the relevant plots/figures for a given question.

Put your codes (including data files and others that are provided as part of the assignment) in a folder named lastname_firstname (replace with your last and first names). Create a ZIP file (name it lastname_firstname.zip, again, replace with your last and first names accordingly) containing the top-level directory.

For example, if I were to submit the homework, I would create a ZIP file named "wang_yang.zip". After running "unzip wang_yang.zip", I should get the following:

wang_yang/

wang_yang/hough/

wang_yang/hough/hough.py

wang_yang/hough/……(other files)

wang_yang/homography/

wang_yang/homography/run.py

wang_yang/homography/homography.py

wang_yang/homography/……(other files)

If I run "python hough" in the "wang_yang/hough" folder, I should get the plots in the report from the relevant question.

Go to this course in Moodle, then click "HW2 submission". Submit the following two files: 1) the PDF document; 2) the zip file containing your codes (see above). The submission server is configured to only accept PDF and ZIP files. If you try to submit other file format, the submission server will not accept it. So make sure you know how to create PDF and ZIP files ahead of time.

Note that you can make multiple submissions. But we will only consider the last submission. We will use the time stamp recorded on Moodle to calculate the late days you have used for this assignment.