Assignment #3

Prof. Hanbyul Joo
M1522.001000, Computer Vision

Assigned: May 8, 2025
Due: May 23, 2025, 11:59 PM

0 Instruction

In this assignment, you will implement functions for the Hough transform and homography.

- **Submission Platform**: All homework must be submitted electronically on eTL.
- **Collaboration Policy**: Discussions with peers are encouraged, but you must solve the problems and write up the solutions independently.
- **Individual Assignment**: Each student is required to submit their own individual report and code.
- **Plagiarism Policy**: Do **not** copy code or reports from others. Any form of plagiarism may result in a score of zero.
- **Coding Requirements**:
  - Use **Python** for all programming tasks.
  - You are **not allowed** to use high-level image processing or computer vision functions such as `cv2.filter2D`, unless explicitly permitted for a specific problem.
  - If you're not sure whether a function or package is allowed, please post your question on the eTL Q&A board.
- **Reporting**:
  - Answers must be **clear, unambiguous**, and **supported by experimental results** (e.g., images, plots, brief quantitative analysis).
  - Only PDF submissions compiled using LaTeX (e.g., via Overleaf or other LaTeX tools) will be accepted. No specific template will be provided.
- **Notebook Submission**:
  - You must also submit the **Jupyter Notebook (.ipynb)** file.
  - Adding new cells is allowed.
  - Make sure that **all outputs are preserved**.
  - All images in the report should be **reproducible** from the notebook.
  - TAs will primarily check if your code is **fully reproducible**.
- **Questions**: We will only respond to questions posted on the eTL Q&A board.

- **Structure**: Submit the zip file as the following folder and file structure. There may be disadvantages in final scores if the submission does not follow the specified folder structure.

```
{YOUR_NAME}_{YOUR_STUDENT_ID_NUMBER}.zip
├── {YOUR_NAME}_{YOUR_STUDENT_ID_NUMBER}.pdf (your report)
├── hough
│   ├── hough.ipynb
│   └── outputs
│       ├── 00000_output.png
│       └── 00001_output.png
└── homography
    ├── homography.ipynb
    ├── output_2-1.png
    └── output_2-2.png
```

1 Hough Transform (50 Points)

The goal of this section is to detect lines and circles in images using the classical Hough Transform. In this assignment, we first quantize the Hough parameter space and construct an accumulator array through Hough voting. We then apply Non-Maximum Suppression to select the highest voted points, which correspond to lines or circles in the original image. The Python skeleton code (`hough.ipynb`) and sample data (`00000.png`, `00001.png`) are provided.

1-1. Hough Transform for Line (20 Points)

(a) (10 Points) Implement function `hough_lines(image, threshold, rho_res, theta_res)` :

- A function to construct the accumulator array `H` for detecting lines.
- **Input:** `image` is a greyscale PIL edge image, `threshold` is a number to classify edge, `rho_res`, and `theta_res` are unit of quantized Hough parameter space.
- **Output:** `H` is a accumulator array constructed by Hough voting.
- **Hint:** Use Normal (Polar) Form parameterization to construct the accumulator array.

(b) (10 Points) Implement function `hough_lines_peak(H, rho_res, theta_res, num_lines)` :

- A function to select highest-voted `num_lines` parameters in accumulator array.
- You need to implement Non-Maximum Suppression to avoid multiple detection for same line.
- **Input:** `H` is a accumulator array, `rho_res`, `theta_res` are unit of quantized Hough parameter space, `num_lines` is a number of lines we want to detect.
- **Output:** `line_rhos`, `line_thetas` are list of highest-voted Hough parameters.
- **Hint:** Focus on locally maximum values in `H` to implement Non-Maximum Suppression.

1-2. Hough Transform for Circle (20 Points)

(a) (10 Points) Implement function `hough_circles(image, threshold, radius_range, a_res, b_res, r_res)` :

- A function to construct the accumulator array `H` for detecting circles.
- **Input:** `image` is a greyscale PIL edge image, `threshold` is a number to classify edge, `radius_range` is a tuple of (`radius_min, radius_max`), and `a_res`, `b_res`, `theta_res` are unit of quantized Hough parameter space.
- **Output:** `H` is a accumulator array constructed by Hough voting.
- **Hint:** Note that `H` is three dimension matrix in this case.

(b) (10 Points) Implement function `hough_circles_peak(H, radius_range, a_res, b_res, r_res, num_circles)` :

- A function to select highest-voted `num_circles` parameters in accumulator array.
- You need to implement Non-Maximum Suppression to avoid multiple detection for same circle.
- **Input:** `H` is a accumulator array, `radius_range` is a tuple of (`radius_min, radius_max`), and `a_res`, `b_res`, `r_res` are unit of quantized Hough parameter space, `num_circles` is a number of circles we want to detect.
- **Output:** `circles_as`, `circles_bs`, `circles_rs` are list of highest-voted Hough parameters.
- **Hint:** Focus on locally maximum values in `H` to implement Non-Maximum Suppression.

1-3. Test on Your Own Images (10 Points)

(a) (10 Points) Take your own photo (recommended) or use images from websites, and perform both line and circle detection. First, run the line and circle detection using the same parameters as in Sections 1-1 and 1-2. Then, adjust the parameters to obtain both improved and degraded results. Include the results in `{YOUR_NAME}_{YOUR_STUDENT_ID_NUMBER}.pdf` and analyze how the parameter changes affect the detection outcomes.

- Attach four images – (1) Line detection using the same parameter settings as in Section 1-1, (2) Line detection with different parameter settings, (3) Circle detection using the same parameter settings as in Section 1-2, (4) Circle detection with different parameter settings.
- Write an analysis of the results.

2 Homography Estimation and Image Stitching (50 Points)

The goal of this section is to create a stiched/panoramic image from source images using image transformation. The Python skeleton code (`homography.ipynb`) and sample data (`00002.png`, `00004.jpg`, `00005.jpg`) are provided. More details about the inputs and outputs of the functions are specified in the skeleton. Include an explanation of your implementation and corresponding results.

2-1. Inserting an Input Image to the Base Image (25 Points)

Insert any image you'd like into the digital screen of the base image, `00002.png`. You may either insert your image into the whole screen or into the right part of it, where the referee in red is shown.

(a) (5 Points) Implement function `set_corr_manual()` :

- A function to manually set correspondences between images.
- **Output:** Matched keypoint locations in images 1 and 2, `p1` and `p2`.

(b) (10 Points) Implement function `compute_H(p1, p2)` :

- A function to estimate the homography between images (either direction is fine).
- **Input:** Matched keypoint locations in images 1 and 2, `p1` and `p2`
- **Output:** The estimated homography, `H`
- **Hint:** The solution would require the usage of SVD. You are allowed to use the existing SVD functions from numpy. You may have to normalize the correspondences for numerical stability.

(c) (10 Points) Implement function `insert_image(base_image, input_img, H)` :

- A function to insert the projected input image into the base image using estimated homography.
- **Input:** Base image, input image, and the estimated homography.
- **Output:** Final output image.
- **Hint:** To avoid aliasing, try producing the merged image by solving for each pixel of the final image rather than mapping each pixel in the input image to a point in the final image.

2-2. Generating a Panoramic Image (25 Points)

You will merge the given two images into a single panoramic image using homography. Instead of manually specifying correspondences between images as you have done above, you will use SIFT descriptor to enable matching. SIFT is composed of scale, orientation, and a 128-dimensional local feature descriptor. Use two sets of descriptors from the template and target, and find the matches using nearest neighbor with the ratio test. Given the matches based on SIFT descriptors, you will compute a homography that fits the image transformation.

(a) (10 Points) Implement function `match_sift(loc1, des1, loc2, des2, distance_ratio)` :

- A function to find the matches of SIFT features between two images.
- **Input:** loc and des are the keypoint locations and SIFT descriptors of each image.
- **Output:** Matched keypoint locations in images 1 and 2, `x1` and `x2`
- **Hint:** The function is supposed to filter the correspondences based on bi-directional consistency and the ratio test. You may use the `NearestNeighbors` function imported from `sklearn.neighbors`.

(b) (10 Points) Implement function `compute_H_ransac(x1, x2, ransac_n_iter, ransac_thr)` :

- A function to estimate the homography between images using RANSAC.
- **Input:** Matched keypoint locations in images 1 and 2, `x1` and `x2`. `ransac_n_iter` is the number of RANSAC iterations and `ransac_thr` is the error threshold for RANSAC.

- **Output:** The estimated homography, `H`

(c) (5 Points) Implement function `merge_image(img_1, img_2, H)` :

- A function to merge img_2 with img_1.
- **Input:** Two images and the estimated homography.
- **Output:** Final output image.
- **Hint:** The final output image will be in a different dimension from the input images.