Assignment- 2 Documentation

Group- 7
Soumava Paul (16EE10056)
Swagatam Haldar (16EE10063)

- In this assignment, we have implemented some basic transformations related to projective space and removing projective distortion.
- The auxiliary helper functions are written in the file **utils.py** and the main tasks are implemented in the **modules.py** file.
- We call the functions in an interactive fashion in **main.py** file. The details of the functions follow.
- **Note-** In our implementation of the following functions, we assume the **i-axis (rows)** of the image to be **X- axis *(vertically down)*** and **j-axis (columns)** to be the **Y-axis *(horizontally right)***. That is, the pixel located at i th row and j th column, will have (x,y) coordinate = (i,j).

In `utils.py` file-

1. **`def isvalid(i, j, r, c)`**- Determines whether a pixel (i,j) is outside, or illegal for the image of dimensions r x c. Outputs 0 or 1 accordingly.
2. **`def extract_lines(line_pair, pair_num):`**- Returns two points (four numbers in a tuple) on one of the parallel line pairs.
3. **`def find_intersection(line1, line2):`**- Finds intersection point of two lines in y = m*x + c form.
4. **`def homography(img, matrix):`**- Returns a transformed image according to the matrix provided. Matrix is 3x3 *projectivity* which is then applied on each coordinate on the image plane i.e., (x,y,1) to find the corresponding transformed (homogeneous) coordinate.
5. **`def rectification_matrix(pl1, pl2, pl3, pl4):`**- Takes two pairs of parallel lines (parallel in real world) from the image and computes *two vanishing points* accordingly and finds the vanishing line's slope(m) and intercept(c). Then it computes a 3x3 homography that sends the vanishing line (l1,l2,l3) to (0,0,1) and returns the matrix.

In `modules.py` file-

1. **`def task1(img):`**- Provides the image and expects the user to select two points on it and it draws the line segment. Once that line is drawn, the code exits.
2. **`def task2_3(img):`**- Provides the image and expects the user to click on **two points to generate each line** that is a total of *8 points* (4 on each pair of parallel lines i.e., 2 points per line). It computes the slope of the vanishing line from the intersection

point of those two pairs and draws a line with that slope passing through the center of the image.

3. **`def task4(img):`** – Shows to the user the image where the vanishing line (through center) is already drawn and asks to select a point on the line L in the image**.** We then extend a vertical line through the point and find its **intersection with the actual vanishing line V**, say the point is (h,k). Randomly choose a slope = (-b/a) of the parallel lines (in real-world) which happens to intersect at the point (h,k). It generates six lines of the form **ax + by + ci = 0** for **1 <= i <= 6**. They are parallel in the real world and intersects the line at infinity at (-b, a, 0). Now we compute the inverse of the transform i.e., to transform the lines back to image space. Expectation is that they will intersect at (h, k) and we verified it by printing their equations. The lines are drawn with different colors in image plane but they were **almost coincident**. ***Please note that if the point of intersection with V i.e, (h,k) happens to be a point much far away from the image, the lines may not be visible in the image plane.***
4. **`def task5(img):`** – Firstly, the image of the garden is cropped as stated in the assignment (masking). Then we show the *cropped garden image* (with flower pots at the boundary) to the user and again expect him to click on 8 points (4 on each pair of parallel lines). From that we compute the homography that sends the vanishing line to infinity. Then we apply this homography on every point of the image. The resultant image has **no projective distortion** i.e., the parallel lines appear as parallel as can be readily seen from the saved output image. **(Affine Rectification)**

Running Instructions

From terminal, run the `main.py` file as **`python3 main.py -task task_index.`** The task_index to task mapping is as follows: [0:task 1, 1: tasks 2 and 3 combined, 2: task 4, 3: task 5]. Running instructions will be displayed in the terminal for each task to guide the user through the operation.
Also note that you need to run tasks 2 and 3 (for computing the vanishing line and saving the image) before you jump to task 4. We have saved some images from our runs already, for your convenience.

Packages Required

Python3, numpy, cv2 (OpenCV 4.1.0 - **important!** Some functions (eg. mouse callback) may not work without this version)