CIS 580, Machine Perception, Fall 2022
Homework 4
Due: November 21 2022, 8pm ET

1 Written part (10 pts)

1. (5 pts) Recall the Lucas-Kanade Optical Flow method. Explain in a couple sentences or less, why optical flow is only valid where the quantity $A^T A$ has rank 2, where A is the spatial gradients matrix.

2. (5 pts) Consider the RANSAC algorithm for fitting a plane (minimal set of 3 points). Your dataset is of size 1000, of which 100 are outliers. What is the minimum number of iterations needed to achieve 99% chance of success, assuming you randomly choose points at each iteration?

2 Coding part

You can run the code via the command line/terminal (if you have installed conda/anaconda you may want to use Anaconda Prompt instead). More instructions are provided in the README. **You should test your code locally before submitting to gradescope**.

1. (0 pts) We provide python functions in file `compute_grad.py` that computes the spatiotemporal derivatives I_x, I_y, I_t of the above sequence for every pixel of the image insight23.png given the image sequence. **You don't need to do anything for this part**

   We will study convolutions in a later module in class, time permitting, but for the purpose of this homework, you just need to know that you can use convolutions to compute spatial and temporal derivatives of images.

   Following 0th and 1st gaussian derivative approximations can be used:

   ```
   g = [0.015625 0.093750 0.234375 0.312500 0.234375 0.093750 0.015625];
   and
   h = [ 0.03125 0.12500 0.15625 0  -0.15625 -0.1250  -0.03125];
   ```

   The derivative wrt x is computed by convolving with h[x], then with g[y], and then with g[t]. Similarly for the derivatives wrt y and t. That means that each time we smooth in the directions orthogonal to the differentiation direction. If only 7 images are given, the result of temporal convolution with an 1x7 mask is meaningful only in the central image (insight23.png).

2. (20pts) Given the derivatives I_x, I_y, I_t fill in the function `flow_lk` in `compute_flow.py` that computes an optical flow field. The result should be the two flow components (u,v) for each pixel as well as a confidence value `smin`. Assuming that the optical flow is constant in a local neighborhood of 5x5 pixels, the flow can be computed using the 25x2 linear system consisting of 25 equations $I_x u + I_y v + I_t = 0$ where (I_x, I_y, I_t) are the spatiotemporal derivatives at every pixel. The smallest singular value `smin` of the system will be used as a confidence measure. Please use `np.linalg.lstsq` for solving the system of linear equations. In the compute_flow.py file, size of x and y in the flow_lk_patch function should both be 1 rather than (h, w). These two values are the center of the patch that you compute flow on. The return values should be the flow vector of size (2,) and confidence (1,) for a single patch. The input arguments x, y of the function should be two integers.

   Hint: (for this and future parts) A common source of bugs in computer vision assignments is usage of x,y coordinate system to row, col convention for a matrix. Remember that x = column index and y=row index when we traverse a matrix.

3. (2pts) Fill out the function `plot_flow` in `vis_flow.py` that plots the vector field given (u,v,smin) at each pixel where `smin` > `thresmin` where `thresmin` is a threshold given as argument to `flow_lk`. There is already code to do the plotting, verify that the vectors point right and a little down. Plot the vector field for thresmin = 1, 10, 30, and **submit the plots with the printed version**.

4. (23pts)

   Fill in `epipole.py` so that it computes the pixel position of the epipole satisfying $\boldsymbol{e}^T(\boldsymbol{x}_p \times \boldsymbol{u}) = 0$ where $\boldsymbol{x}_p = (x_p, y_p, 1)$ is the position of a point in pixels and $\boldsymbol{u} = (u, v, 0)$ is the computed optical flow in pixels. Assume that the motion is pure translational. x_p should be uncalibrated but be centered at

0 (i.e. values range from -256 to 256), as usual for images the x-axis is rightwards and the y-axis is downwards.

Use RANSAC following the template code, but only use the flow vectors with `smin` > `thresmin` for both the sample and test points. Use the ABSOLUTE value of $\boldsymbol{e}^T(\boldsymbol{x}_p \times \boldsymbol{u})$ as the distance function. You should minimize the least-squares error using SVD (similar to what you do in the 8-point algorithm for E). For handling the flattening of matrices use `.flatten()` function of numpy with the default argument ('C').
Include plots showing the epipole and inliners for thresmin = 1, 10, 30, and **submit the plots with the printed version. Comment on if these results seem reasonable, what pattern(s) you see in the inliers and why this might be the case**.

Note that there are two thresholds for this question. `thresh` is a threshold on `u` and `v` which determines which points are considered confident enough to be used. Points where `smin` < `thresmin` should not be used as sample points or ever be considered inliers for RANSAC. `eps` is the distance threshold for a point to be considered an inlier for RANSAC. Also remember to give the inlier indices in terms of the unthresholded (and flattened) version of $\boldsymbol{x}_p$ and $\boldsymbol{u}$. **For the autograder:** inliers should have the sample points first then the test points that are inliers (both converted back to the unthresholded indices as discussed previously).

5. (25pts) (**Submit a description of the algorithm you will use**) Unfortunately, the data we have are in pixel coordinates which are related to the normalized image coordinates via the intrinsic camera parameters:

$$\begin{pmatrix} x_p \\ y_p \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & x_0 \\ 0 & s_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = S\boldsymbol{x} \tag{1}$$

where s_x and s_y are scaling factors and (x_0, y_0) is the image center. For the given image sequence $s_x = 1118, s_y = 1121, x_0 = 357, y_0 = 268$. Complete the python function in `depth.py` that given pixel flow, confidence, epipole, and intrinsic parameters computes a depth at every pixel for which flow exists. Visualize the depths as an image by executing `main.py` with `--depth` flag and **submit the plotted depth map, too**. Plot the depth map for thresmin = 1, 10, 30.

Hint: For two equations in the following form:

$$\begin{cases} cx = a \\ dx = b \end{cases} \tag{2}$$

where a, b, c, d are known. You can solve for x by:

$$x = \sqrt{\frac{a^2 + b^2}{c^2 + d^2}} \tag{3}$$

When threshmin=10, do you see any obvious incorrect depth values in the depth map? **Please include a few sentences explaining what could have caused these errors.**

The epipole given as input to the function is uncalibrated. Please make sure to calibrate in your code.

6. The following question deals with the motion of the planar parts of the scene.

(a) (5pts) Repeat the equations for normalized image coordinates $(x, y) = (X/Z, Y/Z)$ that give the motion field in the case of a planar scene $\boldsymbol{N}^T(X, Y, Z) = 1$. Show that you can write $(\dot{x}, \dot{y})$ as a function of eight parameters which depend on $\boldsymbol{v}$, $\boldsymbol{\omega}$, and $\boldsymbol{N}$ and that it is quadratic wrt (x, y). Hint: Write out depth from the plane equation.

(b) (15pts) Choose only the spatially lower left part of the image sequence (and hence of (u, v) , too) that corresponds to the textured plain in the image. In the `main.py` file, we take the bottom left patch of the scene. Complete the python function `planar_flow.py` that computes the eight parameters of the flow field of the lower left part of the sequence. Please remember to normalise the optical flow in x direction and y direction by their respective focal lengths before using them. The planar flow equation should be in the following form:

$$\dot{x} = a_1 x^2 + a_2 xy + a_3 x + a_4 y + a_5$$
$$\dot{y} = a_2 y^2 + a_1 xy + a_6 y + a_7 x + a_8$$

Your code should return a_1 to a_8 as an array.