1 Epipolar Geometry

In class, we learned that the geometry of two camera views are related by a transform called the "fundamental matrix." We can solve for this matrix using an algorithm called the "eight-point algorithm" given at least eight correspondences between the two views. In this assignment, the eight-point algorithm has already been implemented for you in fund.m. The goal in this section is to gain some familiarity with how the fundamental matrix relates the two views.

(a) Find the fundamental matrix relating the stereo pair of desk1.gif and desk2.gif with \( n \geq 8 \) hand-clicked correspondences. Plot the epipolar lines \( l_1 \) and \( l_2 \) in both images for at least three points in the first view, and verify that they pass through the corresponding points in the second view.

(b) Solve for the coordinates of the epipoles \( e_1 \) and \( e_2 \) in both views.
(c) Repeat the above two steps for the stereo pair of house1.pgm and house2.pgm.

What to turn in

1. A printout of your code.
2. The fundamental matrix and the coordinates of the two epipoles for each stereo pair.
3. For each stereo pair, a plot of the epipolar lines in both images for at least three points. Refer to Figure 1 for an example.

2 Corner Detection

In this section, we will implement a detector that will find corner features in our images. To detect corners, we rely on the structure of a matrix $C$ defined as

$$
C = \begin{bmatrix}
\sum I_x^2 & \sum I_x I_y \\
\sum I_x I_y & \sum I_y^2
\end{bmatrix},
$$

where the sums are taken over a neighborhood $W$ of a coordinate $p$ in the image. The point $p$ is a corner if the both eigenvalues of $C$ are large. Refer to Trucco and Verri, section 4.3 for a detailed explanation.

(a) Implement a procedure that filters an image using a 2D Gaussian kernel and computes the horizontal and vertical gradients. You can use the Matlab function `conv2.m` to perform 2D convolution. You cannot use built in routines for smoothing. Your code must create your own Gaussian kernel by sampling a Gaussian function. The width of the kernel should be $\pm 3\sigma$. That is, if $\sigma = 2$, then the kernel should be 13 pixels wide.
(b) Implement Algorithm CORNERS in p. 84 of Trucco and Verri. Typically you will find multiple corner locations for the same corner in the image. To filter out the weaker corners, you can either follow the suggestion in the book, or ensure that your minimum eigenvalue is a local maxima in its neighborhood.

(c) Test your algorithm on desk1.gif, house1.pgm, and another image of your choice. Select two reasonable values for the threshold and use the same threshold for all images.

What to turn in

1. A printout of your code.

2. In your writeup, for each input image:
   a. Show the input image.
   b. Show the two components of the gradient after filtering the image with $\sigma = 1$, $\sigma = 2$, and $\sigma = 4$. You can scale or invert the image to highlight the computed gradients.
   c. For each $\sigma$, show the detected corners for the two values of the threshold. Use the same threshold for each input image.

3 Sparse Stereo Correspondence

For the last part of this assignment, given the detected corner features in the first image, we will search for the corresponding feature in the second image. To improve our correspondence algorithm, we will use the epipolar geometry of the two views to restrict our search within an epipolar line.

(a) We will test our correspondence algorithm on the desk stereo pair and the house stereo pair. For each stereo pair, select $n \geq 20$ corners in the first image and find the corresponding epipolar lines in the second image.
(b) Implement a procedure that selects pixel coordinates that lie on a line. These coordinates will be the candidates for our correspondence search in the next step. The provided function linePts.m returns the endpoints of the line, which you can use as input to your procedure.

(c) Use window-based correlation to find the corresponding corner in the second view. Choose a reasonable window size and compare the windows using the SSD and the NSSD metrics. For each metric, count the number of correct correspondences and summarize the result in a table as a success rate (percentage correct correspondences / total features).

**What to turn in**

1. A printout of your code.
2. In your writeup, for each stereo pair
   a. Show the input image and the selected corners.
   b. Show a plot of all the epipolar lines in the second view that correspond to the selected corners in the first view.
   c. Show a plot of the corresponding corners found in the second view using each correlation metric.
   d. Calculate the success rate and show your results in a table for each correlation metric.

**4 Tips & Suggestions**

- For finding correspondences, use the Matlab utility called cpselect.m to select and save correspondences between the two views. Also, use the Matlab function getpts.m to manually select points on the image.

- To draw lines and points on the images, the utility functions drawLine.m and drawPoints.m have been provided for you. Keep in mind that Matlab stores horizontal image scanlines as columns in a matrix, so you may have to swap the x and y coordinates to access the correct image location.

- For the corner detection, you can actually combine the filtering and the derivative operations into one kernel.

- Selecting pixel coordinates that lie on a line is called “line rasterization” and there is a famous algorithm called “Bresenham’s algorithm” that does this. But the simplest way to do this is to calculate the slope and y-intercept of the line and iterate over the range of x-coordinates (or y-coordinates) to compute \((x, y)\) coordinates on the line.

- When comparing windows that are close to the boundary of an image, you can assume that your window is positioned so that it doesn’t extend beyond the image boundaries. That is, throw out any corners and correspondences that are too close to the boundary.