ENGN 1610 & 2605 Image Understanding

Lab04: Feature Detection

The goal of this lab include:

- Learn how to detect corners.
- Learn how to use corners to do feature tracking across multiple images.
- Explore SIFT features using VLFeat.
- Learn how to measure feature stability by feature repeatability.

Part I: Corner Features

Problem 1. Implement a Corner Detection Algorithm Given color images shown in 1, create the following function to detect corner features:

```
function corners = corner_detector(image)
```

The necessary steps for a corner detection algorithm are:

1. Convert the image from color to grayscale. Make sure that the class of the image is converted from `uint8` to `double`.
2. Convolving the image $f(x, y)$ with spatial derivatives of a Gaussian $G(x, y, \sigma_1)$ with respect to x and y, *i.e.*, G_x and G_y, where σ_1 is called the *differentiation scale*. Typically, $\sigma_1 = 0.7$ or 1. You are allowed to use MATLAB's function `conv2`. The outputs are the Gaussian convolved gradients of f in x and y directions, *i.e.*, f_x and f_y:

$$\begin{aligned} f_x &= G_x * f \\ f_y &= G_y * f \end{aligned}$$

3. Form three spatial maps: f_x^2, f_y^2, and $f_x f_y$.
4. Blur three spatial maps with a Gaussian $G(x, y, \sigma_2)$, where σ_2 is called the *integration scale* which can vary over a range of scales, *e.g.* $\sigma_2 = 2.0$ or 3.0, *etc.*:

$$\begin{aligned} S_x &= G(x, y, \sigma_2) * f_x^2 \\ S_y &= G(x, y, \sigma_2) * f_y^2 \\ S_{xy} &= G(x, y, \sigma_2) * f_x f_y \end{aligned}$$

5. Compute the determinant and the trace of the matrix M,

$$M = \begin{bmatrix} S_x & S_{xy} \\ S_{xy} & S_y \end{bmatrix}$$

6. Compute the corner response $R(x, y)$,

$$R(x, y) = det(M) - \alpha\,(tr(M))^2, \tag{1}$$

where α is a parameter which is typically chosen as $\alpha = 0.04$.

7. Compute the local max of $R(x, y)$ by applying the non-maximum suppression over a 3×3 neighborhood of each point. You are NOT allowed to use `imregionalmax` to do non-maximum suppression.

8. Use R_0 as a threshold to prune points with $R(x, y) < R_0$, where $R_0 = 0.01 \cdot \max_{x,y} R(x, y)$

9. Return all points (x, y) with strengths $R(x, y)$ as your function output `corners`.

10. Outside your function `corner_detector`, show the image with corners. Use `imshow` followed by `hold on`, then use the `plot` command to plot corners on top of the image. Show all the images in Figure 1 with corners in your report. It is recommended to show grayscale images with visible size of overlaid corners so that features are clear and distinctive on the images.

For the window size of the Gaussians, use $\lceil 6\sigma \rceil$. Make it an odd number by adding 1 to it if necessary.

To check the correctness of your corner detection algorithm, you are recommended to use a checker board image `I = checkerboard(50,10,10);` loaded directly from MATLAB in your code. A successful corner detector should only give features at the corner of each square.

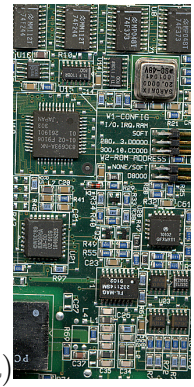
(a)  (b)  (c) 

Figure 1: (a) Circuit Board (b) Coliseum (c) Crane

Problem 2. Application of Corners: KLT Tracker In this problem, we will use the corners you developed in problem 1 to track them across a video sequence. For each corner point (x, y) we assume that it undergoes a translation from one frame to the next. We estimate this displacement, $\overline{v}$, across multiple frames. A corner in the first frame will be at $(x, y) + \overline{v}$ in the second frame, and we can again estimate $\overline{v}$ to determine its location in the subsequent frames. This iterative procedure continues until we have tracked the corners across all frames of the video sequence.

To determine the displacement $\overline{v}$ between two frames, we use a very simplified version of the KLT tracker. Pseudo-code of the algorithm to be implemented is Algorithm 1.

Algorithm 1 Simple KLT tracker

Input: a video sequence of N frames $f_{1:N}$, *i.e.*, f_1, f_2, ..., f_N

Output: `tracked_corners` storing tracked corners across all input images

Obtain `corners` from your function `corner_detector` on the first frame f_1.
`strong_corners` $\leftarrow$ top M string corners from `corners`
for i=1 to N-1 (loop over all frames except the last one) **do**
  `tracked_corners` $[i] \leftarrow \emptyset$
  Compute the gradients `[dy dx]` of f_i.
  for j=1 to M (loop over all strong corners) **do**
    $(p_x, p_y) \leftarrow$ `strong_corners`$[j]$
    $w^0(x, y) \leftarrow \forall (x, y) \in [p_x - m, p_x + m]x[p_y - m, p_y + m]$
    $W^0(x, y) \leftarrow$ interpolation$(f_i, w^0(x, y))$
    $I_x(x, y) \leftarrow$ interpolation$(dx, w^0(x, y))$, $I_y(x, y) \leftarrow$ interpolation$(dy, w^0(x, y))$
    Compute $M(p_x, p_y)$ on f_i using Equation 2
    $\overline{v}^0 \leftarrow [0\ 0]$
    for k=1 to K (or until $\|\overline{\eta}^k\| <$ accuracy threshold) **do**
      $w^k(x, y) \leftarrow w^0(x + \overline{v}_x^{k-1}, y + \overline{v}_y^{k-1})$
      $W^k(x, y) \leftarrow$ interpolation$(f_{i+1}, w^k(x, y))$
      $\delta I_t^k(x, y) \leftarrow W^0(x, y) - W^k(x, y)$
      Compute $\overline{b}^k(p_x, p_y)$ using Equation 3
      $\overline{\eta}^k = M^{-1}\overline{b}^k$
      $\overline{v}^k = \overline{v}^{k-1} + \overline{\eta}^k$
    end for
    if $\overline{v}$ converged **then**
      `strong_corners`$[j] \leftarrow (p_x, p_y) + \overline{v}^k$
    else
      `strong_corners`$[j] \leftarrow \emptyset$
    end if
  end for
  `tracked_corners` $[i] \leftarrow$ `strong_corners`
end for

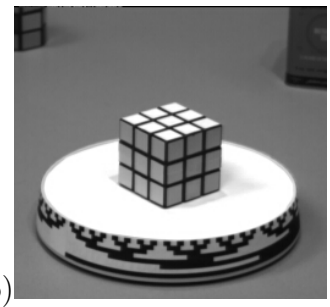
$$M(p_x, p_y) = \sum_{x=p_x-w_x}^{p_x+w_x} \sum_{y=p_y-w_y}^{p_y+w_y} \begin{bmatrix} I_x^2(x,y) & I_x(x,y)I_y(x,y) \\ I_x(x,y)I_y(x,y) & I_y^2(x,y) \end{bmatrix} \tag{2}$$

$$\overline{b}(p_x, p_y) = \sum_{x=p_x-w_x}^{p_x+w_x} \sum_{y=p_y-w_y}^{p_y+w_y} \begin{bmatrix} \delta I_t(x,y) I_x(x,y) \\ \delta I_t(x,y) I_y(x,y) \end{bmatrix} \tag{3}$$

Details of the algorithm is described below:

- **Line 1:** perform your `corner_detector` you implemented in problem 1.
- **Line 2:** Thresholding the corner response $R(x, y)$ and pick the top M corners. Make $M = 20$ or 30. These strong corners are features we track across the video sequence in the KLT tracker.
- **Line 5:** You are allowed to use `[dx dy] = gradient(image);` to generate the gradients of the image.
- **Line 8:** Make a `meshgrid` of size $2m + 1 \times 2m + 1$ centered around the corner under consideration. The most common window size is 5, but you have the freedom of playing with this parameter to get good results.
- **Line 9 and 10:** When tracking corners across multiple images, the position of the tracked corners will not always be at pixel coordinates, but rather at *subpixel* locations. Thus, we need to do interpolation to find the image intensity values as well as gradients of these subpixel corners. Use MATLAB's function `interp2` with linear interpolation as the input argument, and also set `extrapval` to 0 for values outside the image boundaries.
- **Line 13:** Use at most K=15 iterations, and pick the accuracy threshold as 0.01. The accuracy threshold represents the pixel distance. Again, you might have to play around with this parameter to get good results.
- **Line 14:** Update the coordinates of the window by the displacement vector.
- **Line 15:** Do linear interpolation on the updated window with the next frame f_{i+1}.
- **Line 21 to 25:** There are two cases after the iteration for each corner: either the corner is successfully tracked, or the corner is lost, depending on whether $\overline{v}$ converges. Also delete corners that fall outside the image boundaries during the KLT tracking.

There are three different sequences for you to try, Figure 2. Start with the rubix cube to test your algorithm as you could easily look at the tracks of the corners on the cube to see whether your KLT tracker is successful. The performance of this simple KLT tracker might be sensitive to the parameters mentioned above, so try to find the optimal parameters for each sequence.

(a)  (b) 

(c) 

Figure 2: (a) Car sequence (b) Rubix Cube (c) Pedestrian Walking

The following three challenges are optional for all students. Finishing each challenge correctly would gain 15 extra points.

Challenge 1: Improving the Tracker If you look at the simple KLT tracker, you might notice that during the tracking procedure, no new corner features are detected. Implement a new scheme: if a certain number of corner features are not presented, detect new corners from that frame and track those corners from the subsequent frames.

Challenge 2: Dealing with large translation movement Another improvement of the simple KLT tracker is to use a Gaussian pyramid. Do the simple KLT tracking at each level of the pyramid, from top to bottom, so that the difficulty of tracking corners with large translation movement can be mitigated.

Challenge 3: Structure from Motion: Factorization This is a very challenging problem, but one can recover the 3D scene from the tracks you have. The technique is called factorization, and the most famous method to be used is the **Tomasi-Kanade factorization**, `http://en.wikipedia.org/wiki/Tomasi-Kanade_factorization`. If you do attempt this, try it on the rubix cube sequence.

Part II: SIFT Features

Problem 3. Use VLFeat to Extract SIFT Features Download and install the VLFeat library from `http://www.vlfeat.org/`. Go through the tutorial of detecting SIFT features available at `http://www.vlfeat.org/overview/sift.html`. Play with the input parameters, *e.g.*, `PeakThresh` and `edgethresh`, and plot SIFT keypoints (features) on four pairs of images in Figure 3.

Figure 3

Part III: Feature Repeatability

Problem 4. How good are the features? A feature is said to be *repeatable* when it is detected again at the same place across images. A highly repeatable feature is often a stable feature, and thus repeatability can be used to measure how good a feature is.

To determine feature repeatability, we need a relationship of how points in one image are transformed into another. This relationship is called a **Homography**, and is encoded by a matrix. In the case of Figure 3(b), you can visually inspect that the features are transformed from the left image to the right image via a rotation of the image plane.

More formally, a homography is a defined by a 3×3 matrix H which relates the points p in one image to the corresponding points $\bar{p}$ in the second image, which is captured by

$$\bar{p} = Hp. \tag{4}$$

This transformation is defined in terms of *homogenous coordinates*, which means that the points $p = [x, y]^T$ and $\bar{p} = [\bar{x}, \bar{y}]^T$ have to be padded with 1 to match the dimension of the homography matrix, *i.e.*, $p = [x, y, 1]^T$ and $\bar{p} = [\bar{x}, \bar{y}, 1]^T$. Note that when applying the homography to the point p, the results of the transformation is scaled by a factor w, *i.e.*, $Hp = [wx, wy, w]^T$. Thus, to obtain $\bar{p}$, simply divide each element by the scaling factor w and use the first two elements as the position of $\bar{p}$ on the image.

Follow the steps below:

1. The ground truth homography matrix is given for each pair of images shown in figure 3 which transforms points from the left image to the right image. Detect SIFT features as you did in problem 3.

2. Apply the homography matrix to all SIFT features on the left image and overlay them on the right image. Delete transformed features that fall outside the image boundaries.

3. For each overlaid feature, find the nearest features detected on the right image. Use Euclidean distance between two features as the metric: if the distance is below a threshold, make the two features as a pair of correspondence. The threshold could be 1 or 2 (pixels).

4. Measure the repeatability rate: divide the number of left image features by the number of features that found their correspondences from the right image.

5. Now use you corner detector in problem 1 and repeat steps 1 to 4.

6. In your report, show your feature correspondences on every right image of Figure 3. A sample result is shown in Figure 4(c). Also report the repeatability rate of SIFT and corners. Which one is better than the other?

(a)  (b)  (c) 

Figure 4: (a) SIFT features on the left image. (b) SIFT features on the right image. (c) The correspondences of SIFT features.

Problem 5. Track features across a video sequence (This problem is mandatory for ENGN2605 students but is optional for ENGN1610 students who will get 15 extra points if this problem is finished correctly.)

This problem investigates the survival of features across a sequence of video based on homography transformations between pairs of images. Features that survive across all images could be stacked by connecting every features together which forms a *feature track* shown on the first image, Figure 5. Feature tracks are informative: they imply the stability of features and the motion of the camera which captures the images.

Use five images provided by this lab, and repeat your experiment in problem 4 for the first two images. Then, only the features that have correspondences from the second image

are used to transformed to the third image. The process continues until the last image is reached.

Show feature tracks which are plotted on the first image. You can show only 10 to 20 feature tracks if there are many. If you find too few features survived across the five images, try to increase the Euclidean distance threshold you made in problem 4. Use both corners and SIFT features, and answer the questions: Which feature is easier to survive from all five images? Which feature generates more feature tracks? Can you tell how the camera is moving based on the feature tracks?

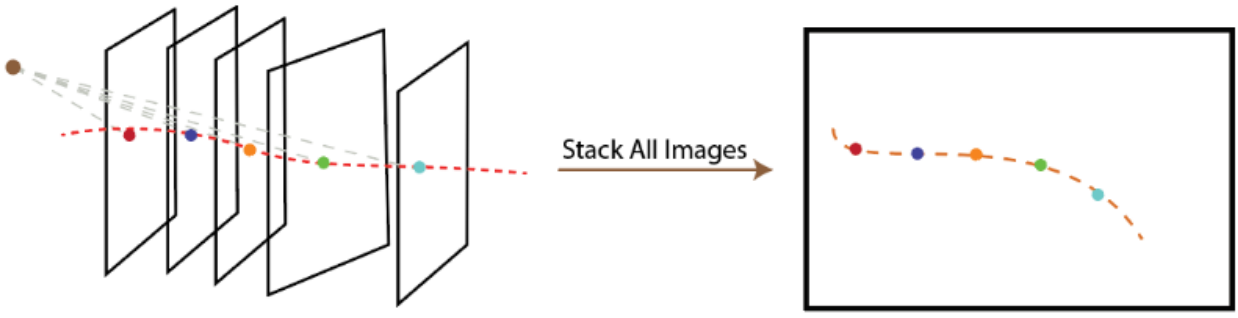


Figure 5: Illustration of how a feature track is formed.