CIS581: Computer Vision and Computational Photography
Project 4, Part A: Optical Flow
Due: Nov.22, 2017 at 11:59 pm

Instructions

- Optical Flow is a **team** project. The maximum size of a team is **three** students. The same team will carry forward for Part B and Part C. You are not permitted to do this individually.
  A team is not allowed to collaborate with another team. Only one individual from each team must submit the code for this part.
  Maximum late days allowed for this part of the project is the average of the late days of the two or three students in the team. The late days used will be subtracted from the individual tally of late days for each student.
- You **must make one submission** on Canvas. We recommend that you include a README.txt file in your submissions to help us execute your code correctly.
  - Place your **code** and **resulting videos** for part A into a folder named "OpticalFlow". Submit this as a zip file named `<Group_Number>_Project4A.zip`
- Your submission folder should include the following:
  - your .py scripts for the required functions
  - .py demo scripts for generating the tracked videos
  - any .py files with helper functions for your code, e.g. Harris corners or Shi-Tomasi corners
  - the input video/s you use
  - the resulting output tracked video/s
  - a .pdf document containing an image of the selected features overlaid over the first video frame; on top of the first frame, plot the points which have moved out of frame at some point along the sequence; additional features of implementation and references to third-party code
- This handout provides instructions for the code in Python. You are not permitted to use MATLAB for this project.
- Feel free to create your own functions as and when needed to modularize the code. For Python, add all functions in a helper.py file and import the file in all the required scripts.
- **Start early!** If you get stuck, please post your questions on Piazza or come to office hours!
- **Follow the submission guidelines and the conventions strictly! The grading scripts will break if the guidelines aren't followed.**

1 Optical Flow

In this part of the project, you will be tracking an object (face/s) in a video. There are two essential components to this: feature detection and feature tracking. Please try to read the following papers before progressing ahead:

- Detection and Tracking of Point Features by Carlo Tomasi and Takeo Kanade
- Derivation of Kanade-Lucas-Tomasi Tracking Equation by Stan Birchfield
- Good Features to Track by Jianbo Shi and Carlos Tomasi

1.1 Detect a Face

In this section, you must detect a face/s in the video by drawing a rectangular bounding box around it. You can either manually draw a bounding box around the face/s or use Face Recognition and Detection function with OpenCV to automate the process. Please remember that you should detect the face only once i.e. in the first frame of the video. If you are unable to find a 'good' face in the first frame, move on to the next frame.
Complete the following function:

```
[bbox]=detectFace(img)
```

- (INPUT) `img`: $H \times W \times 3$ matrix representing the first frame of the video
- (OUTPUT) `bbox`: $F \times 4 \times 2$ matrix representing the four corners of the bounding box where F is the number of detected faces

1.2 Feature Detection

In this section, you will identify features within the bounding box for each face using Harris corners or Shi-Tomasi features. We recommend you to use corner_harris or corner_shi_tomasi in Python. Good features to track are the ones whose motion can be estimated reliably. You can perform some kind of thresholding or local maxima suppression if the number of features obtained are too large.
Complete the following function:

```
[x,y]=getFeatures(img,bbox)
```

- (INPUT) `img`: $H \times W$ matrix representing the grayscale input image
- (INPUT) `bbox`: $F \times 4 \times 2$ matrix representing the four corners of the bounding box where F is the number of detected faces
- (OUTPUT) `x`: $N \times F$ matrix representing the N row coordinates of the features across F faces
- (OUTPUT) `y`: $N \times F$ matrix representing the N column coordinates of the features across F faces

Here, N is the maximum number of features across all the bounding boxes. You can fill in the missing rows with either 0 or -1 or any other number that you prefer.

1.3 Feature Tracking

In this section, you will apply the Kanade-Lucas-Tomasi tracking procedure to track the features you found. This involves computing the optical flow between successive video frames and moving the selected features as well as the bounding box from the first frame along the flow field.

The first assumption that the KLT tracker makes is the brightness constancy. A point should have the same intensity after translation in the next frame (where I is the image function):

$$I(x,y,t) = I(x+u, y+v, t+1) \tag{1}$$

Take the Taylor expansion of $I(x+u, y+v, t+1)$, where I_x, I_y are the x, y gradients of the image $I(x,y,t)$, computed at each element of W (for example, a 10×10 pixel window) at time t and I_t is the temporal gradient:

$$I(x+u, y+v, t) \approx I(x,y,t) + I_x \cdot u + I_y \cdot v + I_t \cdot 1 \tag{2}$$

Hence,

$$I(x+u, y+v, t) - I(x,y,t) \approx I(x,y,t) + I_x \cdot u + I_y \cdot v + I_t \cdot 1 \tag{3}$$

By equation (1), we have:

$$0 \approx \nabla I \cdot \begin{bmatrix} u \\ v \end{bmatrix} + I_t \tag{4}$$

This is a constraint where we have two unknowns (u, v). We get more by assuming that the nearby pixels at points $p_i, i \in [1, 100]$ (W as mentioned above) move with the same u and v:

$$0 = I_t(p_i) + \nabla I(p_i) \cdot \begin{bmatrix} u \\ v \end{bmatrix} \tag{5}$$

leading to,

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{100}) & I_y(p_{100}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{100}) \end{bmatrix} \tag{6}$$

You can solve this overconstrained linear system via least-squares ($Ax = b$): $(A^T A)x = A^T b$

Giving rise to:

$$\begin{bmatrix} I_x(p_1)I_x(p_1) + \ldots + I_x(p_{100})I_x(p_{100}) & I_x(p_1)I_y(p_1) + \ldots + I_x(p_{100})I_y(p_{100}) \\ I_x(p_1)I_y(p_1) + \ldots + I_x(p_{100})I_y(p_{100}) & I_y(p_1)I_y(p_1) + \ldots + I_y(p_{100})I_y(p_{100}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_x(p_1)I_t(p_1) + \ldots + I_x(p_{100})I_t(p_{100}) \\ I_y(p_1)I_t(p_1) + \ldots + I_y(p_{100})I_t(p_{100}) \end{bmatrix} \tag{7}$$

and culminating in:

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix} \tag{8}$$

This gives rise to two equations with two unknowns for each pixel. You will solve for u, v by inverting the 2×2 matrix on the left-hand side and multiplying it by the vector on the right-hand side. Once you have the per-pixel optical flow, you can track a feature through the flow by looking up its displacement in the frame and adding that to its position to get the position for the next frame.

Use scipy.interpolate.interp2d and numpy.meshgrid for computing $I_x, I_y, I(x+u, y+v, t+1)$ when x, y, u, v are not integers and for computing indices respectively.

Features will move out of image frame over the course of the sequence. Discard any features if their predicted translation lies outside the image frame.

In the function below, compute the new X, Y locations for all the detected starting feature locations within a certain bounding box.
Complete the following function:

```
[newXs, newYs] = estimateAllTranslation(startXs,startYs,img1,img2)
```

- (INPUT) `startXs`: $N \times F$ matrix representing the starting X coordinates of all the features in the first frame for all the bounding boxes
- (INPUT) `startYs`: $N \times F$ matrix representing the starting Y coordinates of all the features in the first frame for all the bounding boxes
- (INPUT) `img1`: $H \times W \times 3$ matrix representing the first image frame
- (INPUT) `img2`: $H \times W \times 3$ matrix representing the second image frame
- (OUTPUT) `newXs`: $N \times F$ matrix representing the new X coordinates of all the features in all the bounding boxes
- (OUTPUT) `newYs`: $N \times F$ matrix representing the new Y coordinates of all the features in all the bounding boxes

Precompute the gradients I_x, I_y and then estimate the translation for each feature independently in the following function:

```
[newX, newY] = estimateFeatureTranslation(startX, startY, Ix, Iy, img1, img2)
```

- (INPUT) `startX`: Represents the starting X coordinate for a single feature in the first frame
- (INPUT) `startY`: Represents the starting Y coordinate for a single feature in the first frame

- (INPUT) `Ix`: $H \times W$ matrix representing the gradient along the X-direction
- (INPUT) `Iy`: $H \times W$ matrix representing the gradient along the Y-direction
- (INPUT) `img1`: $H \times W \times 3$ matrix representing the first image frame
- (INPUT) `img2`: $H \times W \times 3$ matrix representing the second image frame
- (OUTPUT) `newX`: Represents the new X coordinate for a single feature in the second frame
- (OUTPUT) `newY`: Represents the new Y coordinate for a single feature in the second frame

In the above function, for a single X,Y location, use the gradients I_x, I_y, and images `img1`, `img2` to compute the new location. It may be necessary to interpolate Ix, Iy, `img1`, `img2` if the corresponding locations are not integers.

For transforming the four corners of the bounding box from one frame to another, please feel free to use skimage.transform.SimilarityTran
Complete the following function:

```
[Xs, Ys, newbbox] = applyGeometricTransformation(startXs, startYs, newXs, newYs, bbox)
```

- (INPUT) `startXs`: $N \times F$ matrix representing the starting X coordinates of all the features in the first frame for all the bounding boxes
- (INPUT) `startYs`: $N \times F$ matrix representing the starting Y coordinates of all the features in the first frame for all the bounding boxes
- (INPUT) `newXs`: $N \times F$ matrix representing the second X coordinates of all the features in the first frame for all the bounding boxes
- (INPUT) `newYs`: $N \times F$ matrix representing the second Y coordinates of all the features in the first frame for all the bounding boxes
- (INPUT) `bbox`: $F \times 4 \times 2$ matrix representing the four new corners of the bounding box where F is the number of detected faces
- (OUTPUT) `Xs`: $N1 \times F$ matrix representing the X coordinates of the remaining features in all the bounding boxes after eliminating outliers
- (OUTPUT) `Ys`: $N1 \times F$ matrix representing the Y coordinates of the remaining features in all the bounding boxes after eliminating outliers
- (OUTPUT) `newbbox`: $F \times 4 \times 2$ the bounding box position after geometric transformation

In the above function, you should eliminate feature points if the distance from a point to the projection of its corresponding point is greater than 4. You can play around with this value.

Please do note that your code should generalize to detecting and tracking multiple faces in a video.

1.4 Face Tracking

Once, you have completed all the above functions, combine them in the follow function:

```
[trackedVideo]=faceTracking(rawVideo)
```

- (INPUT) `rawVideo`: The input video containing one or more faces
- (OUTPUT) `trackedVideo`: The generated output video showing all the tracked features (please do try to show the trajectories for all the features) in the face as well as the bounding boxes

2 Extra Credits:

The following tasks are for extra credit. Implementing any or all of them are optional.

- Add iterative refinement to your KLT tracker
- Integrate a pyramid into your KLT tracker and demonstrate improvement on sequences with large frame-to-frame displacements.
  Refer to Pyramidal Implementation of the Lucas Kanade Feature Tracker Description of the algorithm by Jean-Yves Bouguet
- Correct for drift using the initial local image patches

3 Additional Tips and Information:

We will be releasing multiple videos with varying difficult levels (e.g. easy, medium and difficult) to test the robustness of your tracking. Hence, please do try to make your codes general.
You might need to have to incorporate some of the extra credit features in order to decently track the faces in the medium and difficult datasets.

- For videos containing fast motion or large displacements, you might need to create a pyramidal KLT tracker to estimate displacements at different resolutions
- If you lose are losing features considerably or rapidly, you might need to perform feature detection every 10-15 frames.
- If you lose tracking at some point in the video, you could reinitialize the tracking by performing face detection again.
- If the videos have 'crowd faces' or faces in the background which are smaller in size compared to the faces in the foreground, please feel free to ignore them.