Min Sun
Srdjan Krstic

Optical Flow

COS429
Computer Vision

Today's outline

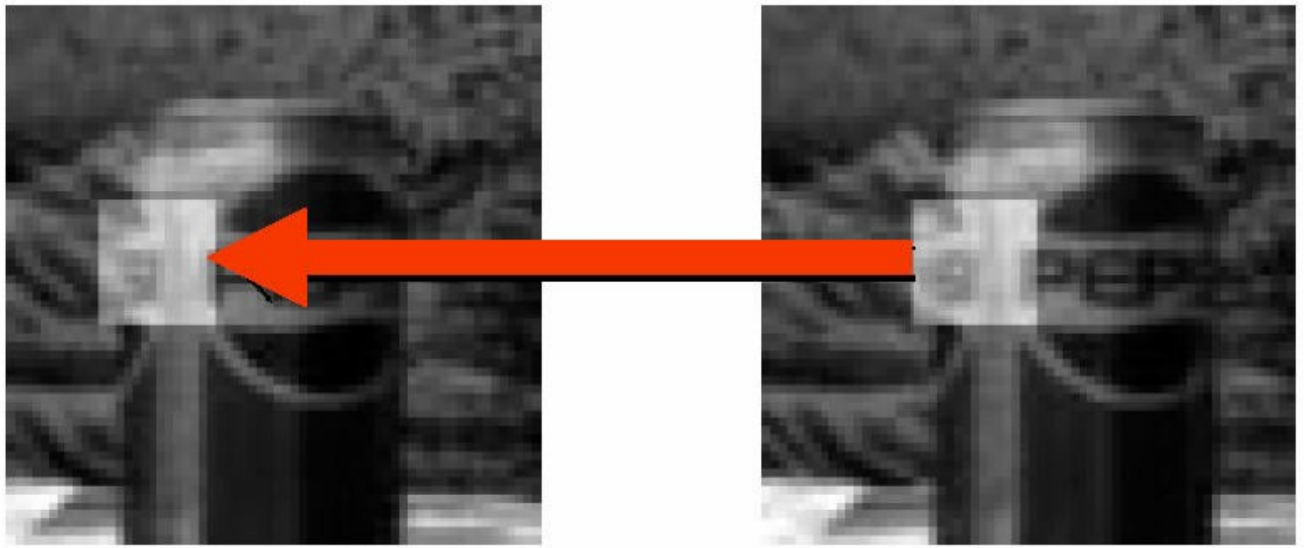
- Optical Flow theory
  - Introduction
  - Shi-Tomasi
  - Lucas-Kanade
- OpenCV implementation
  - Things to look out for
  - Example code, step-by-step

Optical flow theory - introduction

- Optical flow means tracking specific features (points) in an image across multiple frames
- Human vision does optical flow analysis all the time – being aware of movement around them
- Use cases:
  - Find objects from one frame in other frames
  - Determine the speed and direction of movement of objects
  - Determine the structure of the environment
- Questions:
  - How to determine one frame's features in another frame? (Lucas-Kanade)
  - How to choose which features are “good” to track? (Shi-Tomasi)

Optical flow theory - Lucas-Kanade

Brightness Constancy

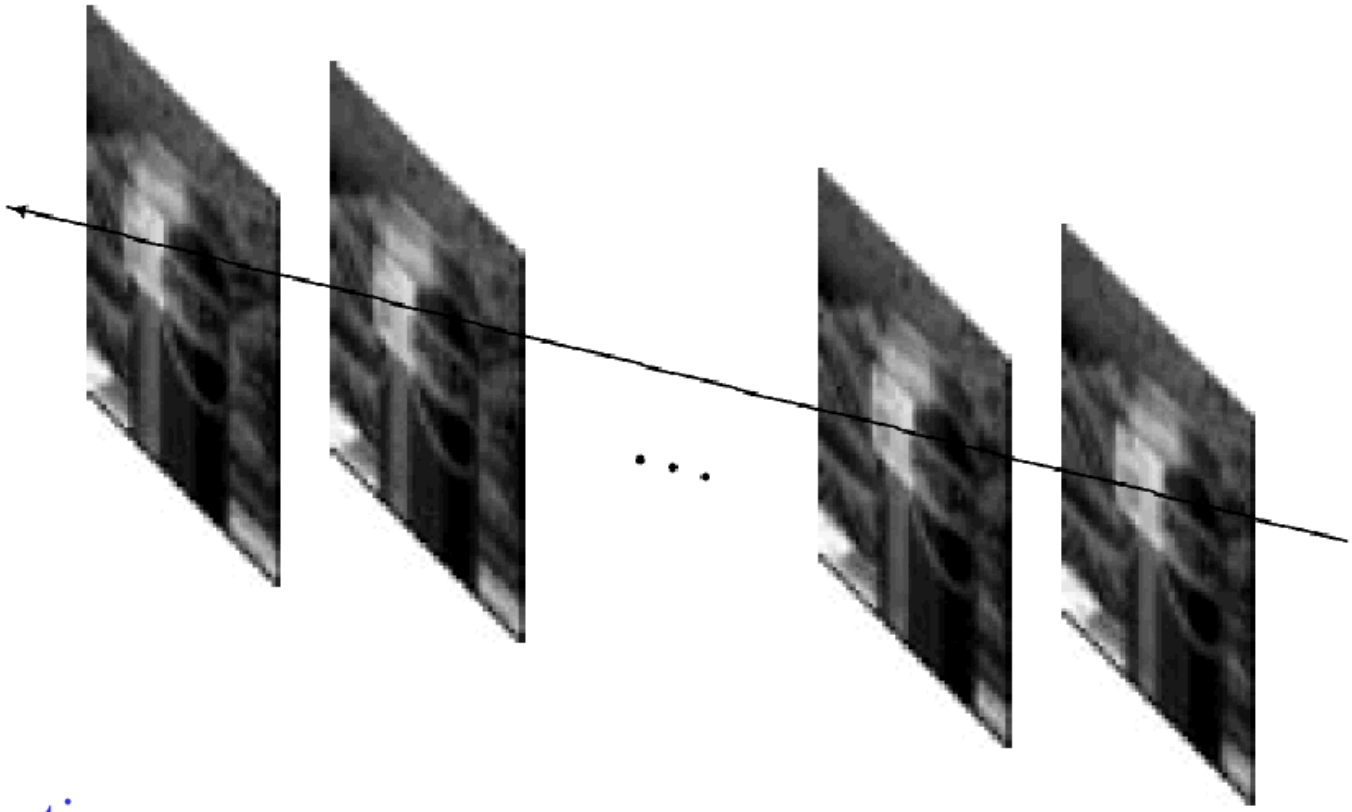


Assumption

Image measurements (e.g. brightness) in a small region remain the same although their location may change.

Optical flow theory - Lucas-Kanade

Temporal Persistence

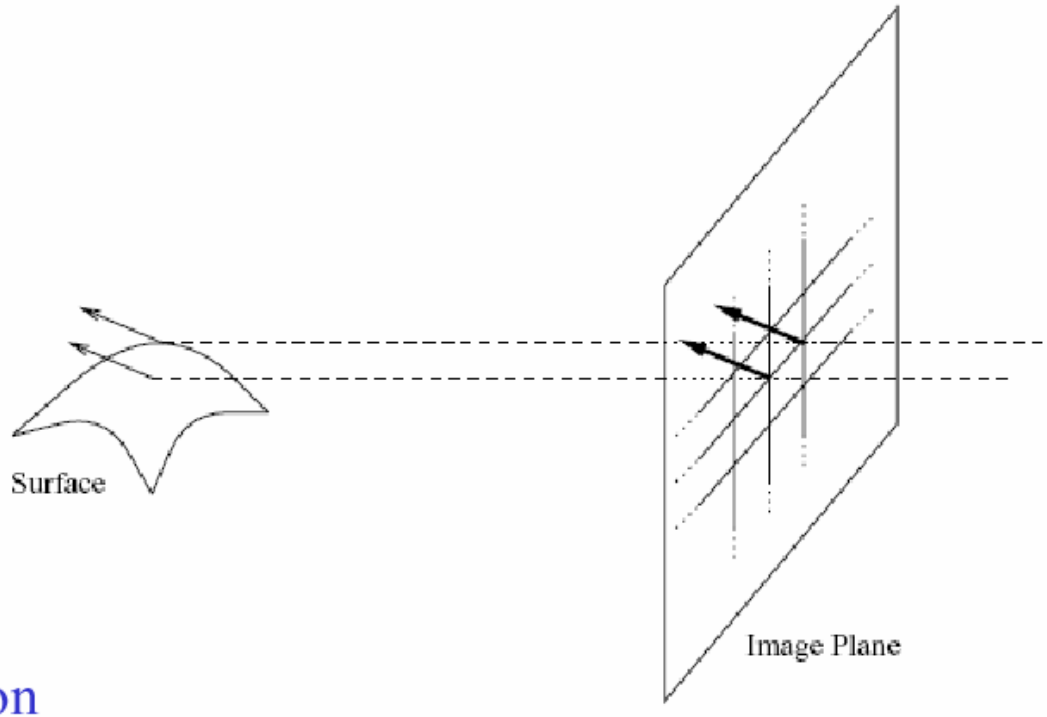


Assumption:

The image motion of a surface patch changes gradually over time.

Optical flow theory - Lucas-Kanade

Spatial Coherence



Assumption

* Neighboring points in the scene typically belong to the same surface and hence typically have similar motions.
* Since they also project to nearby points in the image, we expect spatial coherence in image flow.

Optical flow theory - Lucas-Kanade

- Prob: we have more equations than unknowns

$$\underset{25\text{x}2}{A}\ \underset{2\text{x}1}{d} = \underset{25\text{x}1}{b} \longrightarrow \text{minimize } \|Ad - b\|^2$$

  - Solution: solve least squares problem
  - minimum least squares solution given by solution (in d) of:

$$\underset{2\text{x}2}{(A^T A)}\ \underset{2\text{x}1}{d} = \underset{2\text{x}1}{A^T b}$$

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

$A^T A$ $A^T b$

  - The summations are over all pixels in the K x K window
  - This technique was first proposed by Lukas & Kanade (1981)
    - described in Trucco & Verri reading

Optical flow theory – Shi-Tomasi (cont'd)

- Shi-Tomasi calculates the following matrix

$$\begin{bmatrix} \sum_{\text{neighborhood}} \left(\frac{\partial I}{\partial x}\right)^2 & \sum_{\text{neighborhood}} \left(\frac{\partial^2 I}{\partial x \delta y}\right) \\ \sum_{\text{neighborhood}} \left(\frac{\partial^2 I}{\partial x \delta y}\right) & \sum_{\text{neighborhood}} \left(\frac{\partial I}{\partial y}\right)^2 \end{bmatrix}$$

where I is the intensity of the pixel, and ∂x and ∂y are the horizontal and vertical displacements of the center of the window containing the neighborhood

Optical flow theory – Shi-Tomasi

- A “good” feature will intuitively have two distinctive qualities: texturedness and corner
  - Lack of texture = ambiguity in tracking
  - No corner = aperture problem
  - Many algorithms – Harris, SUSAN, FAST, Shi-Tomasi...
- OpenCV implements the Shi-Tomasi algorithm

OpenCV implementation – HW5

- optical flow analysis

OpenCV implementation – Tips

- We will go through the crucial steps.
- For reference of sample code using optical flow analysis in OpenCV, please see YOUR_OPENCV_INSTALL_FOLDER/share/opencv/samples/c/lkdemo.c

OpenCV implementation – Tips (cont'd)

Opening the input file

- `CvCapture *input_video = cvCaptureFromFile(“filename.avi”);`

- Clearly, this fails when the file doesn't exist
- But also, OpenCV uses a limited set of codecs, and it fails if it cannot read the codec
  - Obviously raw AVI is good
  - MJPEG and Cinepak are good for instance
  - DV is bad

OpenCV implementation – Tips (cont'd)

Dummy get a frame from the video

- `cvQueryFrame( input_video );`

- Ugly hack.
- This is an OpenCV “gotcha”. Unless we get a frame from the video first, we can't look inside of the AVI to determine the properties of the video. In fact we can, but the results are incorrect.

OpenCV implementation – Tips (cont'd)

Read AVI properties

- `CvSize frame_size;`
- `frame_size.height = cvGetCaptureProperty( input_video, CV_CAP_PROP_FRAME_HEIGHT);`

- This is the standard format of getting the properties of the video. Similar constructions are for other properties, differing in the second parameter. For instance, if set to CV_CAP_PROP_FRAME_WIDTH, we would get the width of the frame

OpenCV implementation – Tips (cont'd)

Create a window

- `CvNamedWindow( “Optical Flow”, CV_WINDOW_AUTOSIZE);`

- This is using HIGHGUI. It creates a window which we can use for output, for the purposes of visualization and debugging.

OpenCV implementation – Tips (cont'd)

Loop through frames

- `cvSetCaptureProperty( input_video, CV_CAP_PROP_POS_FRAMES, N );`
- `IplImage *frame = cvQueryFrame( input_video );`

- To get the Nth frame, we set the “current” frame to N and then execute cvQueryFrame.
- GOTCHA: cvQueryFrame always returns a pointer to the same location in memory. So the latter calls always overwrite the former ones. The only way to store multiple frames is to manually copy them to another place in memory.

OpenCV implementation – Tips (cont'd)

Allocate and convert

- `IplImge *frame1 = cvCreateImage( cvSize(width, height), IPL_DEPTH_8U, 1 );`
- `cvConvertImage( frame, frame1, CV_CVTIMG_FLIP );`

- First line creates a new image of the appropriate size, 8-bit depth mono (single channel – grayscale)
- Second line converts the captured frame to this new format.
- GOTCHA: on top of converting, we need to flip the frame, since OpenCV by default reads AVI frames upside-down?!?!?!?

OpenCV implementation – Tips (cont'd)

- `CvPoint2D32f frame1_features[N];`
- `cvGoodFeaturesToTrack( frame1, eig_image, temp_image, frame1_features, &N, .01, .01, NULL );`
- First line allocates the memory to store the features
- Second line actually runs the algorithm
  - eig_image and temp_image are just workspace for the algorithm
  - &N is the place to store the number of features found
  - first .01 is the minimum eigenvalue of the feature to accept
  - second .01 is the minimum euclidean distance between two feature points
  - NULL is the masking frame poining to the part of the image which should be search, whole image if NULL

OpenCV implementation – Tips (cont'd)

Run Lucas-Kanade

- `char optical_flow_found_feature[];`
- `float optical_flow_feature_error[];`
- `CvTermCriteria term = cvTermCriteria( CV_TERMCRIT_ITER | CV_TERMCRIT_EPS, 20, .3 );`
- `cvCalcOpticalFlowPyrLK( ... );`

- The first three lines are set-up steps needed for the algorithm. They are arguments to cvCalcOpticalFlowPyrLK, which has 13 arguments in total, explained in detail in the implementation below.

OpenCV implementation – Tips (cont'd)

Visualizing the output

- `CvPoint p, q;`
- `p.x = 1; p.y = 1; q.x = 2; q.y = 2;`
- `CvScalar line_color;`
- `line_color = CV_RGB(255, 0, 0);`
- `int line_thickness = 1;`
- `cvLine(frame1, p,q, line_color, line_thickness, CV_AA, 0);`
- `cvShowImage("Optical Flow", frame1);`

- This shows how to draw a red line from (1,1) to (2,2)
  - CV_AA means draw the line antialiased
  - 0 means there are no fractional bits

Demo

Final Project

Skeleton code

OpenCV implementation – Tips (cont'd)

Make the output video

- `CvVideoWriter *video_writer = cvCreateVideoWriter( “output.avi”, -1, frames_per_second, cvSize(w,h) );`
- `cvWriteFrame(video_writer, frame);`
- `cvReleaseVideoWriter(&video_writer);`

- We first create a video_writer object. The “-1” argument pops up a nice GUI
- Then we write frames with cvWriteFrame
- Finally the third line deallocates the video_writer
- This part is not implemented in the below code; instead, we show each frame as it is created until user presses a key