WASP Autonomous Systems 1, 2018
Assignment: The KLT Tracker

Author: Michael Felsberg

Chapter 1

Derivation and basic implementation

1.1 Introduction

During this assignment, you will derive and implement the original version and the OpenCV version of the widely used Lucas-Kanade feature tracker (LK tracker) [3]. The LK tracking algorithm is likely to be slow if we try to track a region around every pixel in the image so we will make the restriction to track only one or a few pixels, i.e. only the regions centered around one or a few pixels. Such pixels are typical chosen by sampling regularly or by choosing them based on the eigenvalues of the data matrix [4]. The most common name for this tracker is therefore Kanade-Lucas-Tomasi tracker (KLT tracker).

1.2 Derivations

The task is to derive the forward-additive and inverse-additive versions of the KLT for shift operations. The more general cases are treated in the literature [1].

Define the dissimilarity between two local regions, one in image I and one in image J:[1]

$$\epsilon = \iint_W [J(\mathbf{x}+\mathbf{d}) - I(\mathbf{x})]^2 w(\mathbf{x})\, d\mathbf{x} \tag{1.1}$$

where $\mathbf{x} = [x, y]^T$, the displacement $\mathbf{d} = [d_x, d_y]^T$. The integration region W is a local region around a pixel. The weighting function $w(\mathbf{x})$ is usually set to the constant 1, and we will for simplicity ignore the weight in the derivation from now on.

1.2.1 Forward-additive scheme

Equation 1.1 is identical to the equation given in the original work [3]. In that work, the Taylor series expansion of $J(\mathbf{x}+\mathbf{d})$ about the point $\mathbf{x}$, truncated to the linear term, is considered.

Task 1: Write down this expansion, the resulting approximation of (1.1), and its gradient with respect to $\mathbf{d}$. Use $\nabla J = [\frac{\partial J}{\partial x}, \frac{\partial J}{\partial y}]^T$ and ignore w.

Task 2: Set the derivative to zero and reorder terms to obtain the equation $\mathbf{Zd} = \mathbf{e}$. Give $\mathbf{Z}$ and $\mathbf{e}$. What is required of $\mathbf{Z}$?

The idea of the KLT algorithm is then to solve the original non-linear problem (1.1) by steepest descent iterations using the Taylor expansion from above:

1. Set $\mathbf{d}_{\text{tot}} = 0$.
2. Compute $\mathbf{Z}$ and $\mathbf{e}$ to get $\mathbf{d}$.

[1] In optical flow you may think of images I and J as frames I_t and I_{t+1}.

3. Add $\mathbf{d}$ to $\mathbf{d}_{\text{tot}}$ and compute the shifted image $J(\mathbf{x} + \mathbf{d}_{\text{tot}})$ and its gradient $\nabla J(\mathbf{x} + \mathbf{d}_{\text{tot}})$ by interpolating the original data.
4. If method has not yet converged, go to step 2.

Due to the warping of J in step 3 in the iteration above, the expansion in the subsequent steps will be about the point $\mathbf{x} + \mathbf{d}_{\text{tot}}$. This is called the forward-additive scheme.

1.2.2 Inverse-additive scheme

In the OpenCV version of the KLT, the expansion is done about the point $\mathbf{x}$ in all steps. To avoid moving the expansion point with J, the expansion is computed for $I(\mathbf{x} - \mathbf{d})$ instead. The image J is still warped by $\mathbf{d}_{\text{tot}}$ in each iteration. This is called the inverse-additive scheme.

Task 3: Repeat the expansion above and write down the modified algorithm.

Task 4: Discuss the advantages and drawbacks of the forward and inverse schemes in terms of computational complexity and noise in the images I and J.

1.3 The KLT Tracker

Here you will implement the blocks needed to create your own KLT tracker. We recommend to perform these tasks in Python/NumPy/SciPy, as this simplifies the use of OpenCV and ROS later.

1.3.1 Gradient Function

To be able to estimate the matrix $\mathbf{Z}$ needed by the KLT tracker we need a function that estimates the horizontal and vertical derivatives for all pixels.

Task 5: Implement a function that returns the regularized (lowpass-filtered) derivatives for an image.

As filter mask, use the Scharr filter with the following coefficient matrix (x-derivative, scaled by 32):

3	0	-3
10	0	-10
3	0	-3

1.3.2 Estimating Z

Task 6: Implement a function that estimates the matrix $\mathbf{Z}$ for a specified region.

Make this function general, i.e. make it possible to use different window sizes and also non-square windows. Use a default of 21×21 pixels.

1.3.3 Difference Function

To update the displacement with the KLT equations we need to estimate $\mathbf{e}$ for a specified region.

Task 7: Implement a function that estimates $\mathbf{e}$.

Make this function general, i.e. make it possible to use different window sizes and also non-square windows. Use a default of 21×21 pixels.

1.3.4 Interpolation Function

During the gradient descent iterations, it is apparent that we need to obtain intensity values for non-integer pixel values. You need to implement a function to access these values.

Task 8: Create a function that returns the interpolated intensity values for all sub-pixels specified by a region of interest.

You might want to implement more than one interpolating function (default: bilinear interpolation) but that is not required to pass the exercise. Depending on your implementation, your region of interest might be the whole image. **Hint:** See the Python/SciPy function `interp2`.

1.3.5 Finalizing the KLT Tracker

Now when you have all the building blocks (the gradient function, the coefficient matrix function, the difference function, and the interpolating function) you should be able to finalize the inverse-additive KLT tracker for single scale tracking:

```
outputPoints = calcKLT(oldImage, newImage, inputPoints, winSize, maxIter, minDisp)
```

where inputPoints is a list of 2D points (x,y).

Task 9: Implement an iterative method to solve the displacement equation and to update the displacement vector until some stopping criterion is met, e.g. small enough displacement (default 0.01) or maximum number of iterations (default 30).

1.3.6 Test Implementation

You may use the provided two images `view0.png` and `view1.png` and consider the point $(320, 336)$. Note that you will need to turn them into grey scale first (or make your algorithm handle RGB images).

Task 10: Check that your implementation is working correctly by comparing it to single scale tracking using OpenCV `calcOpticalFlowPyrLK()`.

Note that the method in OpenCV might make smaller steps in each iteration. Compare the result after several iterations. Use some image that you displace by some few pixels.

Chapter 2

Streaming data

2.1 Introduction

In the second part of the assignment you will work with a streaming video, a selected region-of-interest (ROI) is to be tracked, using a number of feature points and the KLT tracker. As a start the ROI can be selected manually, but to challenge yourself we encourage you to try other ways to initialise it, for example, using the object/face detector that was presented in the deep learning material or the HOG detector as presented below.

2.1.1 Useful packages and functions

You already worked with video streams in OpenCV in the computer vision tutorial. If you want to use ROS, the ROS package video_stream_opencv `http://wiki.ros.org/video_stream_opencv` provides one way to produce a topic with a video stream. If you want to use python directly you can looks at `http://imageio.readthedocs.io/en/latest/examples.html`.

From OpenCV, the following functions will be useful:

- `goodFeaturesToTrack` for adaptively sampling points in the ROI [4]
- `calcOpticalFlowPyrLK` for KLT tracking [3]
- `HOGDescriptor` for detecting people [2] (optional for the lab)

Read the documentation of the packages and functions above.

2.2 Implementation, test, and system integration

Once you have designed an algorithm to address the tracking problem, implement, tested, and integrate it into a minimal system to solve the task of tracking.

Your method may select feature points by regular sampling on a grid or by `goodFeaturesToTrack`.

Task 11: What are the advantages and drawbacks of these methods?

2.2.1 Testing

Using your method directly with a web-cam might lead to failures that are difficult to reproduce. For testing purposes, it might be better to generate a synthetic test sequence that your node subscribes to. Note that noise in the image data might lead spurious estimated motions even if the motion is zero.

Task 12: What is a suitable test sequence? What should be varied in the test sequence? How can the spurious motion caused by noise be avoided?

2.2.2 System integration

Build a minimal system around your method to produce a small demo: In the first frame from the image stream, a ROI is marked manually. As soon as it is marked, the algorithm will continue to track that ROI. You may assume that the ROI has not moved significantly between the initial frame and the first frame to be tracked.

Task 13: What is avoided by the assumption? What do you observe if the object rotates? What do you observe if the lighting conditions change?

Optional: Replace the manual detector with a person detector, object detector or similar and repeat the experiments with a person/object in the camera stream.

Bibliography

[1] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 56(3):221–255, 2004.

[2] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA*, pages 886–893, 2005.

[3] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision (darpa). In *Proceedings of the 1981 DARPA Image Understanding Workshop*, pages 121–130, April 1981.

[4] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. Technical report, International Journal of Computer Vision, 1991.