Version: January 26, 2022

Linköping University

Name of student _____

Personal number _____

Date _____

Teaching assistant _____

The Harris Detector and the LK Tracker

Lab Exercise 1

1 Introduction

During this exercise, you will implement the original version of the widely used Lucas-Kanade Feature Tracker (LK Tracker). Unless implemented in low-level operations, the LK tracking algorithm is computationally heavy, so in this computer exercise we will make the restriction to track just one region (centered around one pixel) at a time. This means that we need to choose which pixel to track. Due to the close relationship between the LK tracker and the Harris detector (aka. Harris-Stephens detector) we will use the Harris detector to find good features to track.

1.1 Preparations

Before the exercise you should have read through this exercise guide and completed the home exercises. They are all clearly marked with a pointing finger. To be able to do this you should have read, and understood, the simplified paper on LK tracking that can be found at the course homepage. When we are asking questions on a specific expression or variable, e.g. $\mathbf{T}$, we are using the same notation as in the simplified paper supplied together with this document [1]. It is important to have an understanding of the different steps in the algorithm to be able to complete the computer exercises in time.

Expected preparation time: 4-8h

This exercise should be completed using Python. To use Python in the ISY computer rooms you need to issue `module add courses/TSBB15` at the shell prompt. After that you can use `python3 script.py` to run a Python script, or preferably work in an IDE such as `pycharm` (use `module avail` to find it). If you use multiple scripts (recommended) you can save and load data using either the `pickle` module, or `np.save/load`. For interactive work you can optionally use `ipython3`, which allows e.g. tab-completion and syntax highlighting (you should still use `python3` to run any scripts!).

To access the help code add `import lab1` to your scripts. Now you can use the packaged functions in your code such as `lab1.load_lab_image` and `lab1.get_cameraman`. You can also use `help(lab1)` to see a description of the package.

Warning: Bugs in scientific code are easy to introduce and hard to find. It is recommended practise to always **read the documentation** for functions that you use, and to write **test code** that verifies that they behave as you expect. In particular, be careful not to confuse (x, y) and (row, col) indexing as you will likely use both in this lab. In later labs we will use nD-indexing, with exponentially more ways to make such mistakes.

2 Algorithm Outline

It is important to have an understanding of the different steps in the LK tracking algorithm to be able to complete the computer exercises in time. Before coming to this computer exercise you should have a clear strategy on how to approach the problem.

Preparation Question: Draw a block diagram that outlines the main steps in the algorithm:

Preparation Question: Write the final equation that is needed to update the displacement:

Preparation Question: Using the notation in [1], explain what $\mathbf{T}$ is:

Preparation Question: What is required of $\mathbf{T}$, for the final equation to be solved?

The LK tracking algorithm is computationally heavy if we try to track every pixel in the image. In this computer exercise we will make the restriction to only a single pixel. This means that we need to choose one pixel to track.

Preparation Question: Given your knowledge about the aperture problem and the answer to the question above, what should characterize a pixel (or region) we choose to track?

3 The LK Tracker

Here you will implement the blocks needed to create your LK tracker.

3.1 Regularized derivatives

We start by estimating all the knowns in the LK equation. It is useful to have a function that estimates all of them in order to ensure that they all have the same regularization. Implement a function that takes a filter size and a standard deviation as inputs, and returns the *regularized derivatives*, and compatible images.

A possible function signature is: `regularized_values(I, J, ksize, sigma)` → `Ig, Jg, Jgdx, Jgdy`.
Hint: This was discussed in the computer lesson, and in [1].

3.2 Estimating T

You will need to implement a function that estimates an orientation tensor for a specified region. Make this function as general as possible, i.e. make it possible to use different window sizes and also non-square

windows.

One possible function signature is `estimate_T(Jgdx, Jgdy, x, y, window_size)` → `T`

3.3 Difference Function

To update the displacement with the LK equations we need to estimate **e** for a specified region. Implement a function that does this. Make this function as general as possible, i.e. make it possible to use different window sizes and also non-square windows.

One possible function signature is `estimate_e(Ig, Jg, Jgdx, Jgdy, x, y, window_size)` → `e`

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. Create a function that returns the interpolated intensity values for all sub-pixel positions specified by a region of interest. You might want to implement more than one interpolating function but that is not required to pass the computer exercise.

Hint: You can use `scipy.interpolate.RectBivariateSpline` to get an interpolation function. E.g.

```
>> imgc = RectBivariateSpline(np.arange(img.shape[0]),np.arange(img.shape[1]), img)
```

Preparation Question: Use `RectBivariateSpline` define an interpolation object for an image, and use your new object to cut out regions of interest in a test image. Make sure you understand the calling structure fully.

3.5 Finalizing the LK Tracker

Now when you have all the building blocks (the regularized derivative function, the orientation tensor function, the difference function and the interpolating function) you should be able to finalize the LK tracker. What you basically need is to solve the displacement equation and a control loop that updates the displacement vector until some stopping criterion is met, e.g. small enough error or maximum number of iterations.

Hint: To solve a linear equation system use `np.linalg.solve`.

Check that your implementation is working correctly. To do this, use the function `get_cameraman()` to obtain **I**,**J** and **dTrue**. **I** is the original image, **J** is a shifted version of the same image, and **dTrue** is a vector describing the displacement between the images. Track a region with [height,width]=[70,40] centered around [row,col]=[85,120]. When testing your implementation, your estimated displacement **d** should come close to **dTrue** even after the first iteration. Verify that the estimated displacement is improved when using several iterations.

Question: What are the estimated displacements after the first and second iteration? What is the true displacement?

4 The Harris Detector

For the previous exercise we manually selected a region to track, but we would like to be able to automatically detect good features to track. We will use the Harris detector for this due to the close relationship between the Harris Detector and the LK tracker. Look at the answer to the previous preparation question. You should have come to the conclusion that good features to track are basically the same features that we can detect by using the Harris detector.
Preparation Question: Write down the expression used to calculate the Harris response.

You will implement a function to estimate the Harris response for each pixel in the image. We will later use this function to extract a good feature to track with our LK tracker.

4.1 Orientation Tensor

Implement a function that returns the orientation tensor for all pixels in an image (i.e. a tensor field).

One possible function signature is:
`orientation_tensor(img, grad_ksize, grad_sigma, ksize, sigma)` $\rightarrow$ `T_field`.
Hint: This was discussed in the computer lesson.

4.2 Implementation of the Harris Detector

Now when we have the orientation tensor you should be able to estimate the Harris response for all pixels. Implement a function that estimates the Harris response for all pixels in an image.

One possible function signature is: `harris(T_field, kappa)` $\rightarrow$ `H_field`.

Test this function on the image `cornertest.png`. By choosing a threshold for the Harris response you should be able to detect good features to track.

Question: When raising the threshold for the Harris response, you will end up with some distinct features. Are these the expected features?

You will typically end up with clusters of several neigbouring points above the threshold. Such clusters can be removed by 2D non-max suppression.

Hint: The function `scipy.ndimage.filters.maximum_filter` can do this for you, e.g.

```
>> img_max = maximum_filter(img, size=3)
>> [row, col] = np.nonzero(img == img_max)
```

5 Combining LK and the Harris Detector

Now when you are sure that the LK tracker and the Harris detector are working, it is time to combine them. Use the Harris detector to find the best K features in `chessboard/img1.png` that should be tracked in the subsequent frames `chessboard/img2.png,chessboard/img3.png,..,chesssboard/img10.png`. Avoid tracking features near the border of the image, since these might cause trouble.

Question: Show your lab assistant when you got a working implementation of this. Track the best $K = 5$ features and make sure that you are displaying the result obtained between each frame.

6 Regularised LK

This exercise should be done if there is time left.

Extra

You may have noticed that the LK tracker can get stuck in local minima. To get a more robust tracker we can perform the tracking using gradually reduced regularisation. That is, we start by using a large standard deviation while extracting the gradients from the image. By reducing the standard deviation each time our LK tracker has converged, we might be able to avoid local minima. Try this with $\sigma = 4, 3, 2, 1$.

Question: Did this improve the performance?

References

[1] Björn Johansson. Derivation of the Lucas-Kanade tracker. 2007.