Kanade-Lucas-Tomasi(KLT)Tracker

Maria Barrett, dtq912
Guangliang Liu, bxd337
Alexander Wahl-Rasmussen, lgv740
Keith Lia, dhl107

January 25, 2014

1 INTRODUCTION

The report describes the process to implement a KLT tracker to track features in a continuous video.

2 ABOUT THE KLT TRACKER

The Kanade-Lucas-Tomasi (KLT) Tracker is a method for sparse optical flow estimation and it can be applied to a lot of different cases. In this assignment, it is used to find matches between fixed-sized windows in the past and current frame. There is an assumption that the displacement between two continuous frames is very small and almost the same for neighborhoods around the detected interest points. Harris corners are good features to be tracked in the algorithm.

Harris corners are the local interest points where the neighborhood shows edges in more than one direction. In our implementation, we wrote a function named harris to detect the harris corners. First we constructed a 2×2 symmetric matrix with the derivatives of image at each pixel, this could be done with convolution then we gave every pixel a matrix. The matrix is called Harris matrix, and the eigenvalues of the Harris matrix is a tool to determine if the corners are good features if they are above a certain threshold. This is due to the variation in

the image intensity is reflected in the property of the eigenvalues, i.e. their variation is correlated with the image intensity at the specific path. However, we utilize the Cayley-Hamilton method where we calculate the quotient of the matrix.

As the camera moves, the pattern of image intensities changes and this change can be described as the image motion or even better, as the optical flow. For continuous image sequences, the image at time t1+t2 (t2> 0) could be acquired through the movement of all points in the image at time t1. The aim of the algorithm is to find the displacement that has the minimum matching error on a given patch window. The matching error equation could be minimized on condition that the derivative of the equation is zero. During the period of tracking, bad features have to be dropped and new features would be imported. The bad features could, like above, be identified by finding the eigenvalue of the harris matrix, but this is not something we have implemented. Instead we calculate the optical flow on the initial points and comment on the result.

The code reads the video and plots the corresponding features in a certain frame and the later frame until the end of the video.

3 IMPLEMENTATION

Our implementation makes use of the images in the "Dudek" data set from the University of Toronto. For the first frame, we use our harris function to find good features that we can track in the subsequent frames. We do this by obtaining the gaussian derivation of the image in both axes. We obtain also the Harris matrix components "Im x^2", "Im y^2" and "Imxy", as required for the structure tensor, G.

$$\text{G}=\begin{bmatrix} Im\,\text{x}^2 & Imx*Imy \\ Imx*Imy & Im\,\text{y}^2 \end{bmatrix}$$

We obtain the matrix' determinant and trace, and divide them from each other to obtain the Harris value. Once we have this, we filter out the features that fall below a certain threshold based on the maximum Harris value. For the rest of the frames, we used to different implementations - our own LK Tracker implementation, and OpenCV's calcOpticalFlowPyrLK for comparison. For our tracker, we apply again the gaussian filter on the current frame to get the derivations in both axes, however we also calculate the "time" value. This is obtained by checking the difference in pixels, in this frame and the current one. We build the Harris matrix ,G, again, as well as the residue, e, and do the following calculation: $(1/G,\text{e})$, where

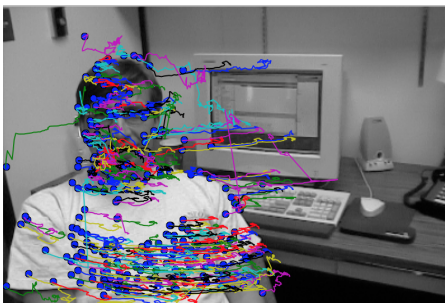
$$\text{e}=\begin{bmatrix} Imx*Imt \\ Imy*Imt \end{bmatrix}$$

This gives us a 2×1 matrix, containing the approximate displacement h at each location in the image. This is our optical flow. We then keep track of each feature and its displacement over time, and plot both on the frame.

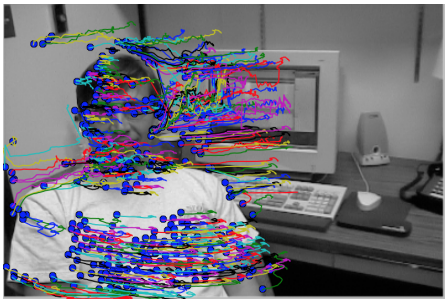
4 Experiments

In the experiments we try out different window sizes. If the window size is too small then the displacement is harder to estimate because the variation of motion is smaller and thus less reliable. A lot of things can happen in a large window - surface discontinuity, distortions due to viewpoint change - which makes it difficult to find in the next frame. We set the weighting function to 1 like in (Tomasi, Kanade) and try the different window sizes in the tracker between 5 × 5 and 100 × 100. All experiments run over the first 100 frames and use the same parameter settings for the harris corner detector: threshold = 0.01 and $\min_d ist = 10$.

5 Results

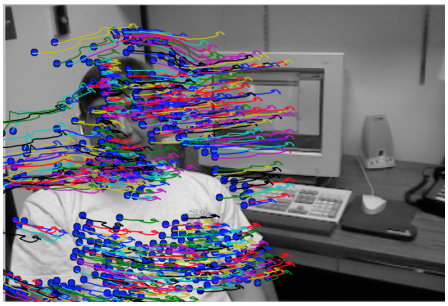


(a) Window size 5x5 with Opencv



(b) Window size 15x15 with Opencv

(a) Window size 50x50 with Opencv



(b) Window size 100x100 with Opencv

6 Discussion

As should be evident from the runtime, our implementation is not very efficient. Neither is it particularly good given that it doesn't correctly check whether a new point is actually a corner.

As for the parameters stated in the Tomasi et al. article, the weight parameters carry over quite nicely given the images seen above - in the CV-library Lucas-Kanade tracker that is. As is also reported above, we experimented on multiple window sizes and we found that a smaller window size would make the motion more vivid, but less robust, meaning that the "guess" would overestimate the displacement. On the contrary, increasing the windows size to 50+ did not seem to affect the motion in such a rapid fashion. However, we would argue that the optimal setting of window-size would be somewhere around 15×15 given that the Lucas-Kanade algorithm deals with smaller displacements in motion. Increasing the window size would thereby increase the probability of calculating noise.

Additionally, the OpenCV implementation utilizes a gaussian pyramid structure, which means that it does multiple optical flow calculations on multiple derived copies of the image, where the sigma differentiates from low to high, i.e. fine to coarse. This allows the pyramidal Lucas-Kanade to have a low optical residue vector and still handle a high displacement. So not only is the OpenCV algorithm more accurate and efficient, it is also more robust.

REFERENCES

[1] J. Shi and C.Tomasi. Good features to track, In *Proceedings of CVPR'94*. pages 593-600, 1994.

[2] C. Tomasi and T. Kanade. Detection and tracking of point features. Technical Report CMU-CS-91-132, Carnegie Mellon University, April 1991.