Computer Vision 16-720
Homework 5

Daniel Ribeiro Silva
Andrew ID: drsilva

November 13, 2014

Question 1

Q1.1

By linearizing the objective function by first-order Taylor expansion, we get:

$$\sum_X \left[I(W(X,p)) + \nabla I(W(X,p)) \frac{\partial W}{\partial p} \Delta p - T(X) \right]$$

where $X = (x, y)$ and W is the warp function, which in this case is a simple translation with two parameters (u, v).

By taking the partial derivative w.r.t. Δp and setting it to 0, we get:

$$\sum_X \left[\nabla I(W(X,p)) \frac{\partial W}{\partial p} \Delta p \right]^T \left[\nabla I(W(X,p)) \frac{\partial W}{\partial p} \Delta p \right] \Delta p = \sum_X \left[\nabla I(W(X,p)) \frac{\partial W}{\partial p} \Delta p \right]^T [T(X) - I(W(X,p))]$$

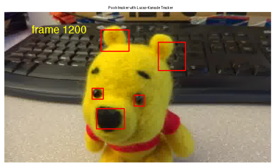
which can be rewritten as

$$A^T A \Delta p = A^T b$$

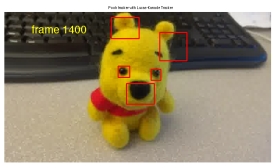
We see that $A^T A$ in this case represents the Gauss-Newton approximation to the Hessian matrix, and side we want to solve it for Δp we need that matrix to be invertible. Also, as described in [3] it must be positive definite. Finally, it's fairly important that this matrix is well-conditioned if it is large (since we would be using numeric approximation methods for this inversion).

Q1.2

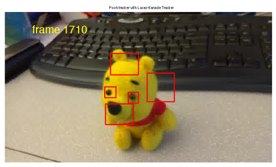
For the 101 frames displayed, the algorithm does pretty well and is able to perfectly track the car. A possible fail scenario would be if the car passes behind some object and is partially of fully occluded. A solution to that is doing robust tracking. Another scenario is if the video has large motion, which can be solved by doing multi-scale tracking. Also, if we only consider pure translation, this model could fail for more complex (but yet very common) motions. This can be solved by applying more complex motion models to the tracking algorithm.



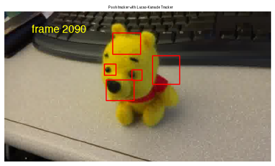
(a) Left ear starts failing



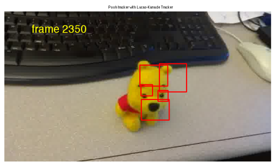
(b) Right ear starts failing



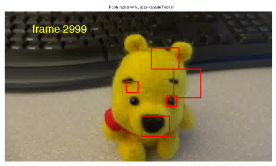
(c) Nose starts failing



(d) Left eye starts failing



(e) Right eye starts failing



(f) Last frame

Figure 1: Tracking failure moment for each component

Question 2

Q2.1

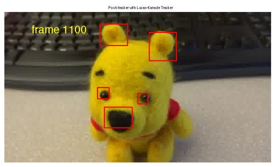
Code submitted.

Q2.2

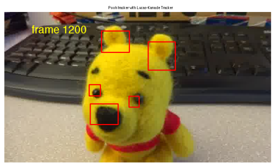
The tracking result for different frames is shown in Figure 1. The Figure shows when each body part starts failing (the corresponding frame can be read in yellow in the image). We can notice that there are 2 main causes for failure (losing track), the first one is a more complex change in the image, which can be exemplified by changes other than pure translation such as zooming or rotation. The second cause is the accumulation of errors. Basically, this means that side the algorithm is purely based on the previous frame/rectangle, errors will necessarily accumulate. The algorithm at step t will consider $t-1$ to be the ground truth. So at each step you are accumulating errors and carrying it to the next step. This was very clear for the tracking of Pooh's ears.

QX2.3

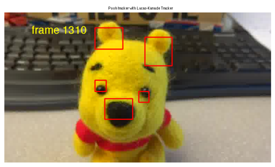
My idea was to use not only the past frame as the reference for the LK algorithm, but a set of past frames. I basically implemented it for a set of size 2 (the past 2 frames). I basically store the history o frames and at every step I compute (u, v) for each of the past frames. I then adjust the reference for the past frames (since the (u, v) they compute is time-shifted) and then compute the final (u, v) as a weighted average of the individual results. More recent frames have higher weight. The result was actually good. The tracker resisted for more frames. The initial LK algorithm would resist for about 200 frames. With this change in algorithm, it resisted for about $270 \sim 300$ frames. What's interesting is that it was less stable for the components. It avoided losing track early, but it was never perfectly centered on the components. The code can be seen at $QX2\_3.m$ and the result is shown in Figure 2 and in the submitted video $QX2\_3.avi$.



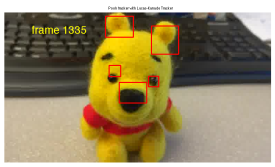
(a) Frame 1100



(b) Frame 1200



(c) Frame 1310



(d) Frame 1335 - track lost

Figure 2: Tracking failure moment for each component using the modified version of LK

Q2.4

Code submitted.

Q2.5

Code submitted. The function signatures are:

```
D = genDisplacementMatrix(ann, perturbedConfigurations)

F = genFeatureMatrix(dataPath, ann, perturbedConfigurations)
```

where *ann* is the annotation matrix as initially provided in this homework, $perturbedConfigurations$ is a cell array with each cell containing the matrix of perturbed configurations for a given frame, and *dataPath* is the path where the data is located.

Q2.6

Code submitted.

The function signature is:

```
[updatedConfigurations, W] = learnMappingAndUpdateConfigurations(D,F,perturbedConfigurations)
```

where $perturbedConfigurations$ is a cell array with each cell containing the matrix of perturbed configurations for a given frame, D is the displacement matrix, F is the Feature matrix, W is the learned weight matrix, and $updatedConfigurations$ is a cell array with each cell containing the matrix of the perturbed configurations displaced by the learned W for a given frame.

Q2.7

Code submitted. I have used $n = 200$ perturbed configurations and scales [0.8, 1, 1.2]. The sequence of loss functions (norm of the current displacement matrix) is shown below. Note that is it constantly decreasing, as expected.

```
iteration 1: 3.407009e+03
iteration 2: 1.109390e+01
iteration 3: 1.796099e+00
iteration 4: 8.809909e-01
iteration 5: 6.155334e-01
```

Q2.8

Code submitted.

Q2.9

Code submitted. The SDM algorithm worked pretty well. The tracking only started failing around frame 1500.

Q2.10

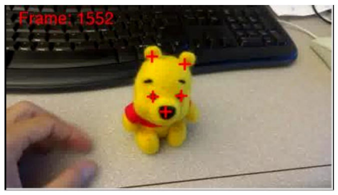
The SDM algorithm over performed the LK algorithm. The main reason is the main assumption of the LK algorithm that the displacement of the components between two consecutive frames is small and approximately constant, which is not always true. Besides, because of the usage of pure translation as the warping function, it was unable to deal with rotation and/or zooming.

Lucas-Kanade:

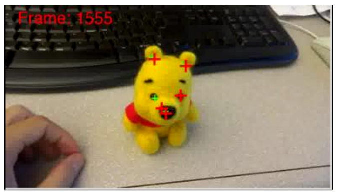
- Advantages
  - Can be generalized (by changing the warp function) to handle rotation, scaling, and shearing.
  - No need for training and/or labeled/past data. It's completely unsupervised.
- Disadvantages
  - Assumes restrictive hypothesis of constant motion.
  - As mentioned by [3], the functions for the gradient descent might not be analytically differentiable and/or the Hessian might not be positive definite.

SDM:

- Advantages
  - Is not restricted to the hypothesis of constant motion. It is able to adapt to it by learning the displacements.
  - No need to compute the Jacobian or the Hessian matrix.
  - Since it uses SIFT, it is robust to scale change and rotation (if we properly adjust the SIFT descriptors).
- Disadvantages
  - Need for training and/or labeled/past data
  - Can be very unstable if point falls slightly outside a region of previously observed SIFT features.



(a) 1552 - before losing track



(b) 1555 - losing track

Figure 3: Before and during tracking failure moment for my modified version of SDM

Q2.11

By increasing the range of the perturbation scales to [0.6, 0.8, 1, 1.2, 1.4] and increasing the number of perturbed configurations from 200 to 300 the algorithm was able to overcome the problem of scale change at frame 1410 and was able to keep tracking pooh until frame 1550. Results are shown in submitted video $QX2\_11.avi.$ and in Figure 3.