Ex2 203200480 320521461

Part 1

1. The constant brightness assumption assumes that the intensity values of the pixel (which corresponds to a certain object in the image) does not change due to time and place changes.
   We use this assumption in the LK objective function that we want to minimize when trying to find the movement vectors in order to fit the image sliding windows between the 2 images. Meaning that if we could compute the Translation\2D Affine\ Projective vectors we expect that the outcome of the subtraction between the 2 corresponding windows should be 0 under this assumption.
   This assumption does not fit the real world since in reality the brightness might change. For example, while following a road at night that is illuminated by street lights the illumination changes whenever we move out of one street light and get closer to the other.

2. The aperture problem is an important problem in image flow computation. It refers to the ambiguity in determining the true velocity using a local motion detector. This ambiguity can be well observed from the original formulation, where $\frac{dI(x,y,t)}{dt} = I_x \cdot u + I_y \cdot v + I_t$, which can be rewritten in vector dot product form as (I_x, I_y) = (u, v) = –It for dI(x, y, t) ≃ 0, under the smoothness constraint. This indicates that (u, v) cannot be uniquely determined when (I_x, I_y) is perpendicular to (u, v) . It is generally accepted that any vision system, whether a biological or an artificial system, exhibits the aperture problem.In order to solve this problem we need to assume that over a small window all the pixel neighbors have the same (u,v) so for a 5x5 window we get 25 equation of the sort $I_x \cdot u + I_y \cdot v + I_t = 0$ and now we can solve the corresponding least square problem as we do in the LK algo .

3. The optical flow problem is that we need to solve with only 1 equation for 2 or more unknown (depending on the transformation we apply). clearly this problem does not have a unique solution.
   There for in order the solve this problem we use the assumption from the previous question and that is that near by pixels move in the same way.
   Under this assumption we get n^2 for a nxn window but still remain with just 2 unknown.
   The LK is solved in the least squares as described earlier, or more accurately we can solve iteratively by $\Delta p = argmin_p \sum [I2\big(W(x;p)\big) - I1(x)$.
   LK use this approximation for I2:
   $$I2\big(W(x;p+\Delta p)\big) \approx I2\big(W(x;p)\big) + \nabla I2 \frac{\partial w}{\partial p} \Delta p$$
   When for each step we basically preform a gradient descent:
   The LK algo assumes 2 main things:
   a. All the pixels in the window move in the same direction and order
   b. The difference between pixels in 2 consecutive frames is quite small, usually we assume just 1 pixel shift (this comes from the Tylor approximation we used).

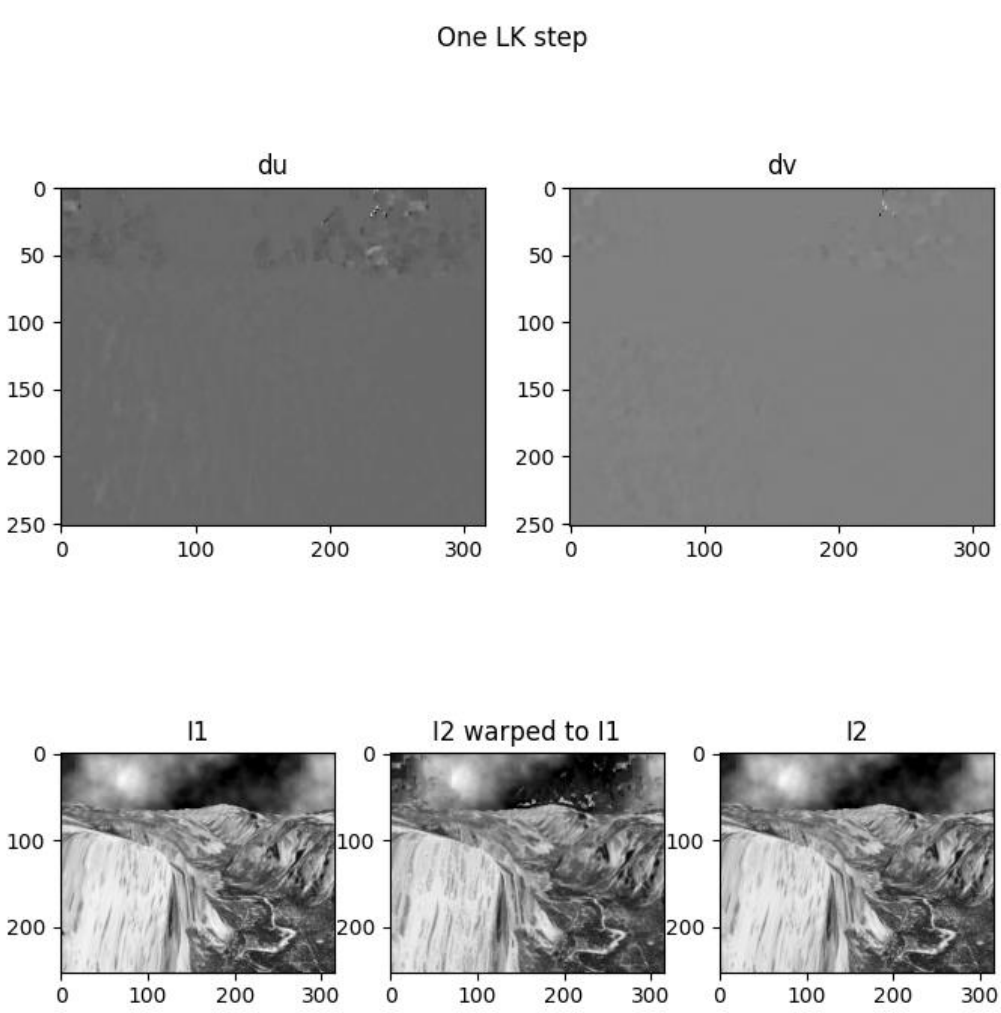
4. The LK assumption does not hold in quit a few cases.

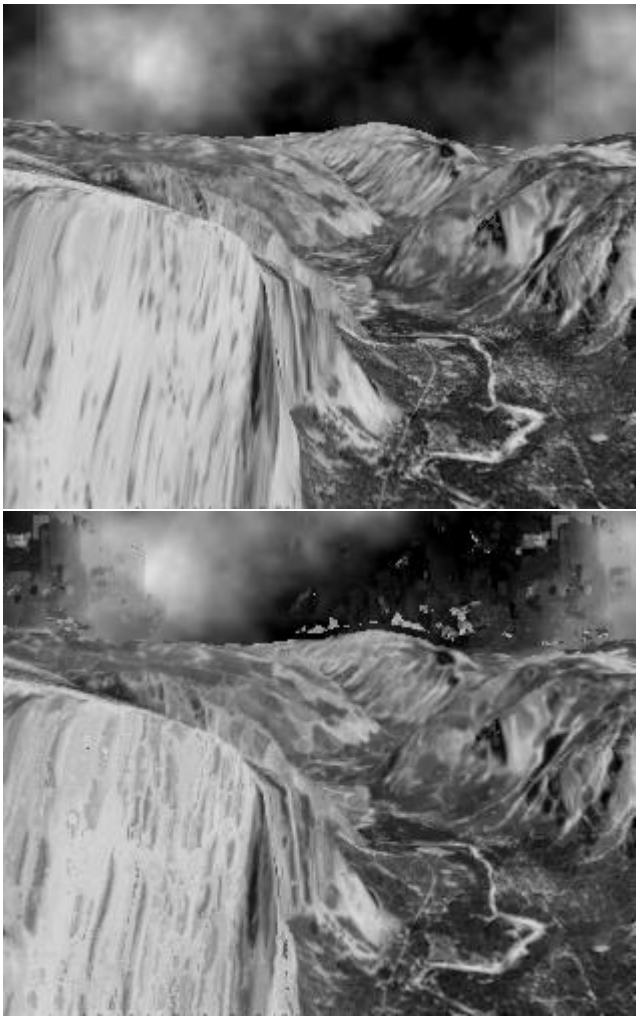
The most common one is for a static camera in a dynamic world. When there is a moving object and a background (which all of the background pixels does not move from frame to frame), In this case the LK assumption does not hold for all of the object boarders, this is due to the fact that for a window around edges will include pixels from the moving object and pixels from the static background (which are static and hence they do not move like the objects pixels do).

5. If we were given the ability to divide the image into a movement layers, meaning assume we could assonated each pixel to the movement layer it belongs to. This way for each layer the LK assumption will be correct and all the pixels will have the same movement parameters (u,v).
   For example imagine a street view camera looking at two people walking in different direction down the street and crossing each other, we would like to have one persons movement be associated with one layer and the other persons movement be assosaited with another layer. If it would be possible the we would run the LK algo per layer (as each layer corresponds to certain person's movement so all the pixels do move in the same direction).
   For each layer we can now find the correct movement parameters (u,v) after finding that we reassemble all the layers back to the original image layer (to form back just 1 layer). Notice that for most of the videos we can use this method to get a much better performance than the original LK algo, after all the video is not aware of the assumption made by the LK algo.

Part2

4.1 we shall now add the created images to our report:


One LK step
du
dv
I1
I2 warped to I1
I2



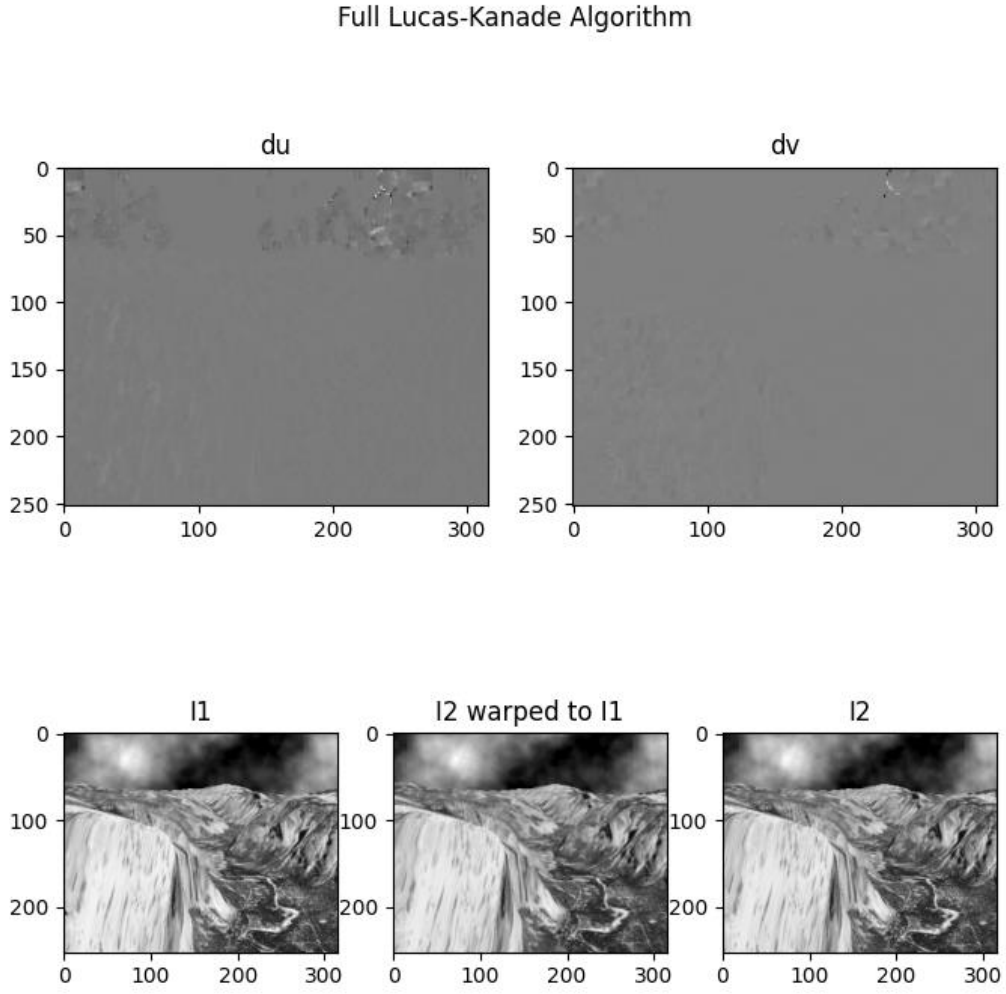
As we can we got an imperfect warped image and this is because the movement is too big between the 2 images, on top of that we did not preformed any iteration on the received warped image and the (u,v) which we know can improve the performance of the LK algorithm.

4.2 The reason that the result we got is imperfect is that the LK algorithm assumption is that the motion between the two images (I1 and I2) is small and since we did not used the pyramid this assumption does not hold, meaning the movement in the images is not small which lead to this imperfection.

6. The reason that the gif 3_after_full_lk.gif looks better now is because we used the build_pyramid method to decimate the image size and for each layer of the decimated image we used the counting (u,v) in order to calculate the I2 warped image and on top of that for each layer we iterated on the image for a user defined iteration in order to descend on the optimal optical flow. This way we managed to:

    1. Make sure our math wont break by dividing our two images to small pyramid diminutive images which the motion between pixels is indeed small
    2. By preforming iteration on each pyramid layer we make best use of the LK algorithm gradient descent property.

We shall add now the received images from the optical flow:


Full Lucas-Kanade Algorithm
du
dv
I1
I2 warped to I1
I2