Autonomous Robotics Lab Report

Master in Computer Vision

UNIVERSITE DE BOURGOGNE

Centre Universitaire Condorcet - UB, Le Creusot

24 May 2017

Submitted to: Prof. Yannick Benezeth

Submitted by: Mohit Kumar Ahuja

Aim:

- OurThe goal of this work is to familiarize yourself with optical flow problem. Horn-Schunck and Lucas-Kanade methods will be applied to image stabilization problem.

About The First Exercise:

It's very important first to know about the code before explaining anything. So, here are some points to know about the code in details:

- Exercise_1.m file is the main code file which uses all the functions and after running that file you can see the output of all the three algorithms.
- Function HS.m is made for the implementation of Horn and Shunck Method to calculate U and V.
- Function LK.m is made for the implementation of Lucas and Kanade Method to calculate U and V.
- Function hierarchical_LK.m is made for the implementation of Lucas and Kanade Method with a multiresolution scheme to improve the result of Lucas and Kanade Method.

{The code is fully commented for better understanding of the reader}

Problem Statement: Complete the implementation of Horn and Schunck (HS.m) algorithm. Your algorithm will iterate a fix number of times (given by iter variable). Comment your code.

Solution: We know that the Optical Flow Constraint Equation is,

$$I_x.U + I_y.V + I_t = 0$$

$$Ix.u+Iy.v+It.Ix+\lambda 2\nabla 2.u=0$$
$$Ix.u+Iy.v+It.Iy+\lambda 2\nabla 2.v=0$$

Where λ is the Regularization parameter. So, as explained by Professor, I first made a derivative Mask w.r.t x, y and t. And using that mask, I convoluted the Image pixels with that masks. And finally using the formula mentioned below, I computed the u and v. the figure of movement is shown in figure 1.

$$u = \bar{u} - I_x \frac{I_x\bar{u}+I_y\bar{v}+I_t}{\lambda^2+{I_x}^2+{I_y}^2} \text{ and } v = \bar{v} - I_y \frac{I_x\bar{u}+I_y\bar{v}+I_t}{\lambda^2+{I_x}^2+{I_y}^2}$$

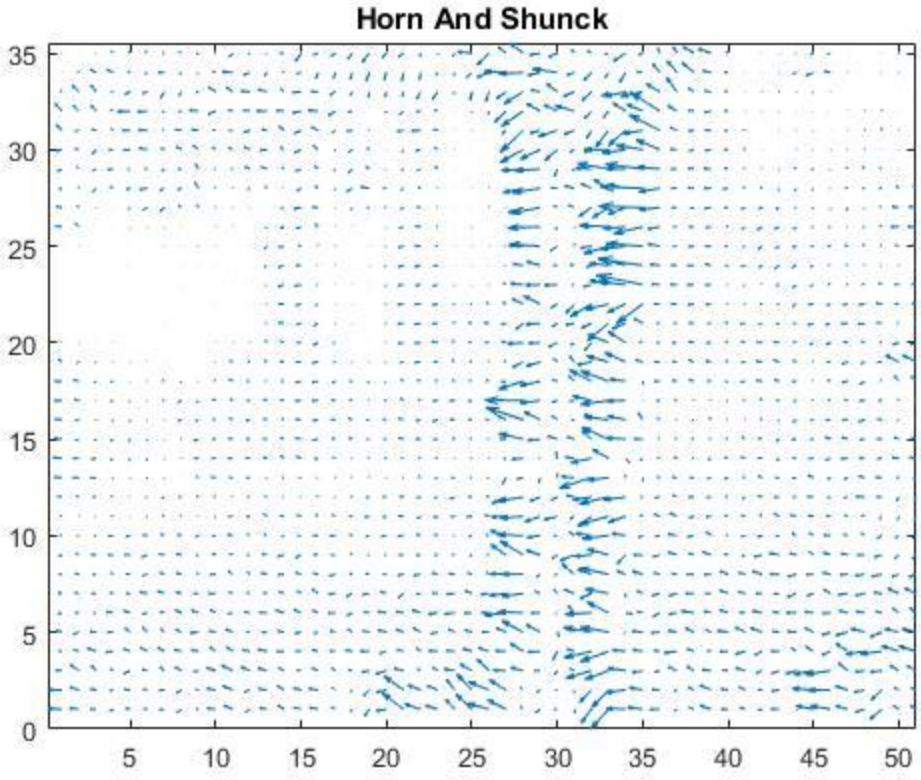


Figure 1: Output of Horn and Shunck Method

{The code is fully commented for better understanding of the reader}

Problem Statement: Complete the implementation of Lucas and Kanade (LucasKanade.m) algorithm. For this question, you will not consider the hierarchical extension. Comment your code.

Solution: We know that the Optical Flow Constraint Equation is,

$$I_x . U + I_y . V + I_t = 0$$

The optical flow is constant on the neighbourhood of the current point(x, y). Each Neighbour gives one equation. Using the equation mentioned below, where A is the concatenated matrix if Ix and Iy and B is the vectorised It,

$$A\begin{pmatrix} u \\ v \end{pmatrix} = B$$

So, I first made a derivative Mask w.r.t x, y and t. And using that mask, I convoluted the Image pixels with that masks. And finally using the formula mentioned above, I computed the u and v using the "Pinv" function for Pseudoinverse. The figure of movement is shown in figure 2.

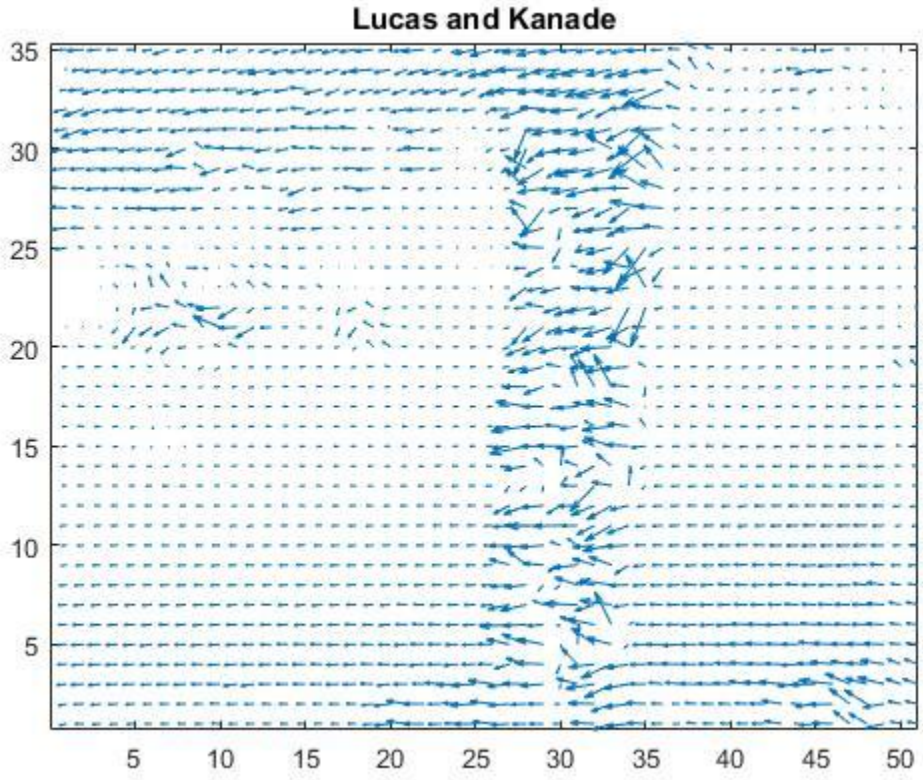


Figure 2: Output of Lucas Kanade's method

{The code is fully commented for better understanding of the reader}

Problem Statement: Improve the Lucas Kanade's method with an ultiresolution scheme. You can startwith hierarchicalLK.m. Test your implementation on the Garden sequence.

Solution: We know that the Optical Flow Constraint Equation is,

$$I_x.U + I_y.V + I_t = 0$$

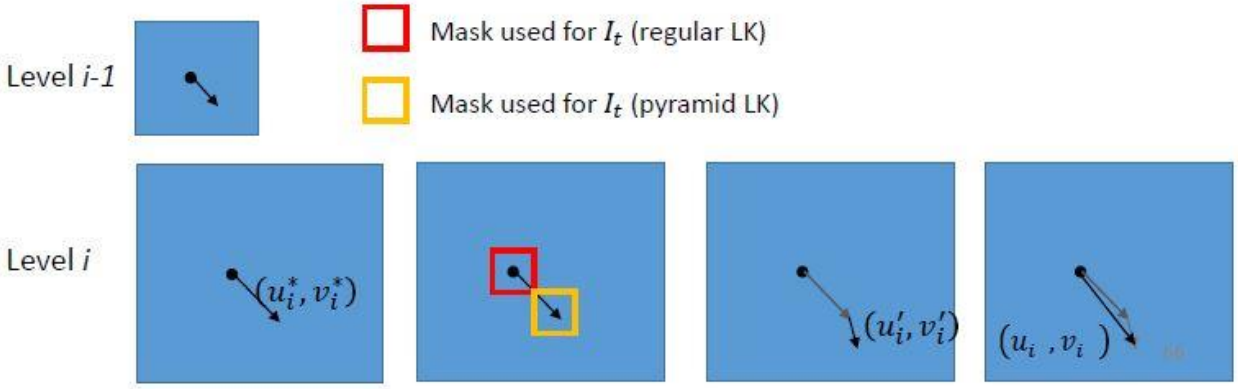
The Algorithm to improve the Lucas Kanade's method with an ultiresolution scheme is shown below,

Algorithm

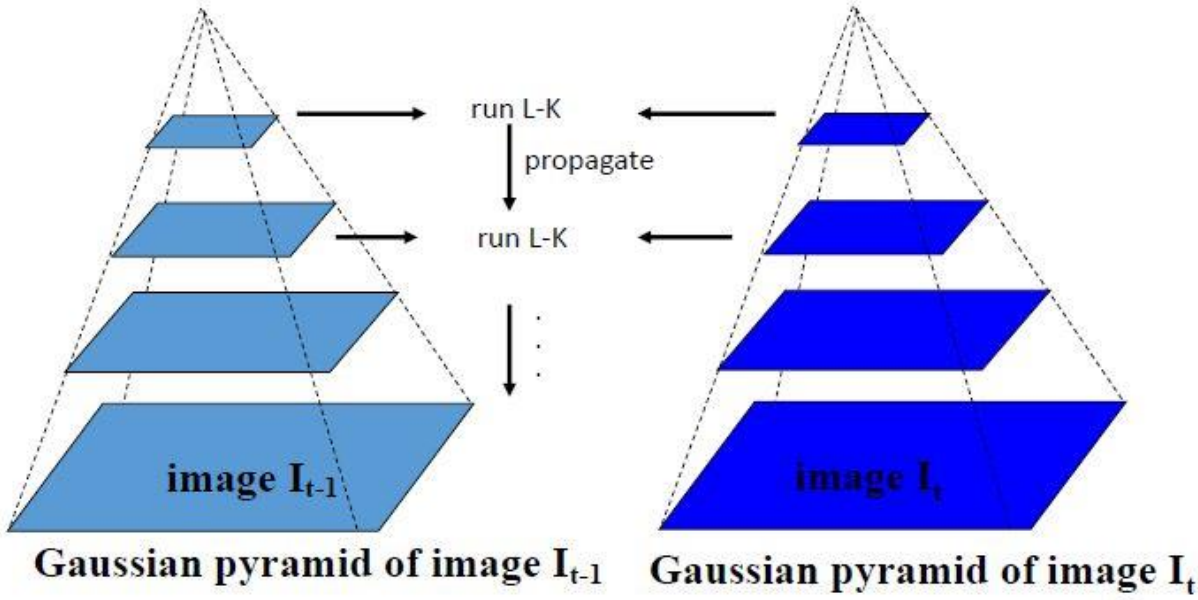
Compute simple LK at highest level

At level i

- Take flow u_{i-1} and v_{i-1} from level i-1, then bilinear interpolate it to create u_i^* and v_i^* matrices of twice resolution for level i.
- Multiply u_i^* and v_i^* by 2.
- **Compute I_t from a block displaced by $u_i^*(x, y)$ and $v_i^*(x, y)$**
- Apply LK to get u_i' and v_i' (the correction in flow)
- Add correction u_i' and v_i', i.e. $u_i = u_i' + u_i^*$ and $v_i = v_i' + v_i^*$



Coarse to fine optical flow estimation



So, I first made a derivative Mask w.r.t x, y and t. And using that mask, I convoluted the Image pixels with that masks. I computed the u and v using the "Pinv" function for Pseudoinverse. In the end we took values like the Pyramidal form by downsizing the resulting values. The figure of movement using improved Lucas Kanade's method with a multiresolution scheme is shown in figure 3.

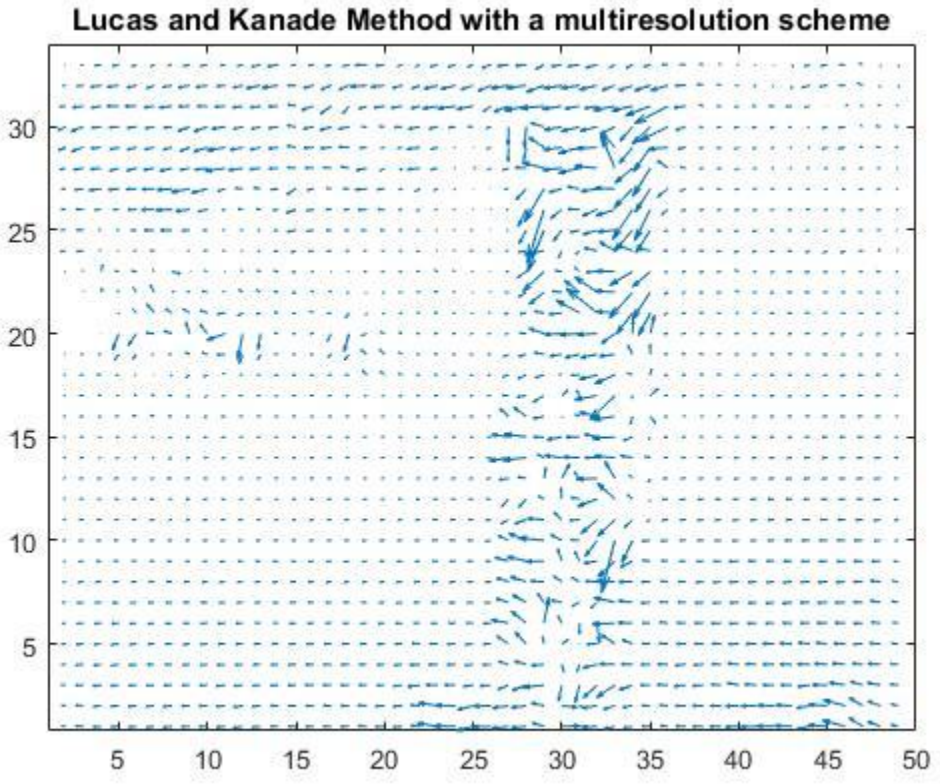


Figure 3: Output of improved Lucas Kanade's method with a multiresolution scheme

{The code is fully commented for better understanding of the reader}

Aim:

- Parametric motion estimation and application to image stabilization

About The Second Exercise:

It's very important first to know about the code before explaining anything. So, here are some points to know about the code in details:

- Exercise_2.m file is the main code file which uses all the functions and after running that file you can see the output.
- Affine_Motion.m is used to compute the Affine Parameters which will further be used to compute the displacement of points.
- Compute_u_v.m is used to compute the U and V from the equations.
- Create_New_Image.m is used to create a New Image using the displacement between first two images.

{The code is fully commented for better understanding of the reader}

Problem Statement:

- Suppose that the motion (u; v) for all pixels (x; y) in the image can be modelled by an affine model θ = (a, b, c, d, e, f)^T:

$$U = ax + by + c$$
$$V = dx + ey + f$$

Using this assumption, show that the solution (u; v) of the OFCE:

$$I_x.U + I_y.V + I_t = 0$$

Is also solution of:

$$M.\theta = P$$

Where M is an n _ 6 matrix and _ is a 6 _ 1 matrix. Where n is the number of pixels in the image.

- Propose a method (inspired by LK's OF estimation algorithm) to estimate θ.
- Application to image stabilization:
  (a) Implement a function θ =Affine_Motion (I1,I2) which computes the affine motion between I1 and I2.
  (b) Compensate the camera motion of successive images with _ to stabilize the sequence.

Solution: For the Problem statement given, I firstly computed an Affine_Motion function to calculate the Affine Parameters which will further be used to compute the displacement of points from one image to another. The equation used to compute Affine Parameters is the Optical flow constraint equation: $I_x.U + I_y.V + I_t = 0$

$$I_x.\,(ax + by + c) + I_y.\,(dx + ey + f) = -I_t$$

Which is further simplified to:

$$M * \theta = P$$

Where Theta (θ) is the Affine Parameter vector, theta = {a,b,c,d,e,f}

So, as we did before I calculated the $I_x, I_y\ and\ I_t$ Parameters by convoluting derivative masks onto the images and then using the above mentioned equation, I calculated the M and the P matrix, which lead us to compute θ by using this formula:

$$\theta = inv\,(M'*M)*M'*P$$

So, by this we got our Affine parameters which is θ. Now we will substitute the values of a,b,c,d,e and f in the u and v equation;

$$U = ax + by + c$$
$$V = dx + ey + f$$

And we got our Direction of Displacement by this. Figure 4 shows the direction of displacement. After getting the direction, we will add the displacement to the original pixel values of first image and then we round off the pixel values because there are no pixels numbered 10.5 or 222.6. And also we create an empty image of same size as our original image.

So, I applied an if statement that if the coordinates are within the values then the previous value should be replaced with the new value and afterwards we got a displaced image which is shown in Figure 7. You can see the displacement at the bottom right of the new image.

{Note: Figure 5 and Figure 6 are the Original images, im1 and im2}

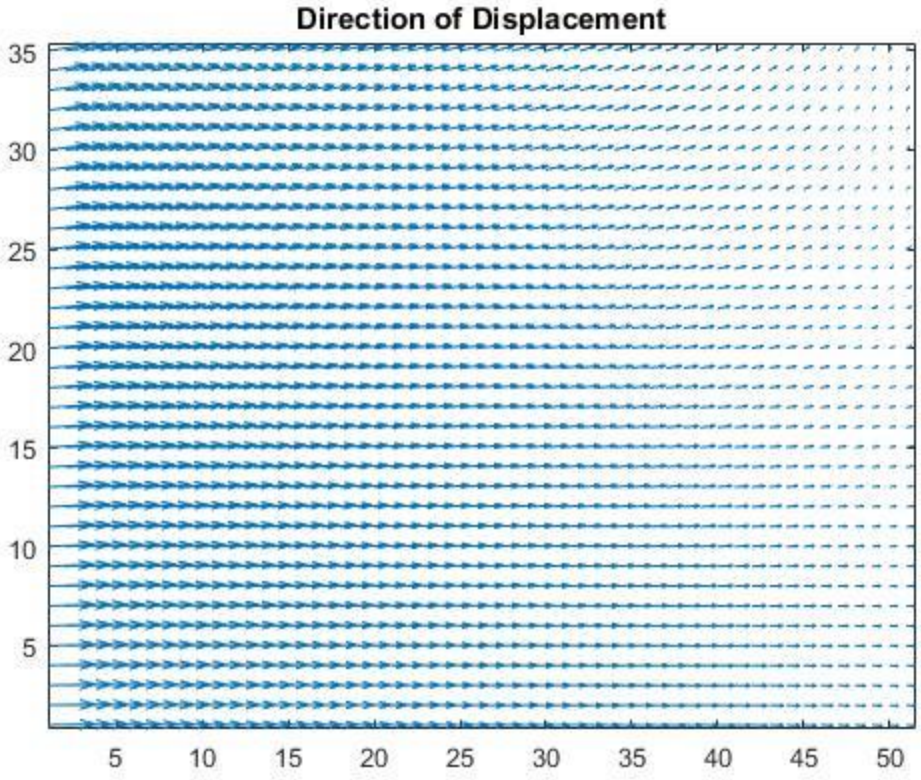


Figure 4: Output Direction of Displacement

Figure 5: Image 1 given as an Input

Figure 6: Image 2 given as an Input

Figure 7: New Displaced Image

{The code is fully commented for better understanding of the reader}