16-720 Assignment 3: Lucas-Kanade Tracking

Umut Soysal

March 22, 2019

Abstract

Contents

1 Lucas Kanade Tracking

1.1 Theory

In the scenario of two-dimensional tracking with a pure translation warp function,

$$W(x;p) = x + p \tag{1}$$

The problem can be described as follows: starting with a rectangular neighborhood of pixels $\mathbb{N} \in \{x_d\}_{d=1}^{D}$ on frame T_t, the Lucas-Kanade Tracker aims to move it by an offset $p = [p_x, p_y]^T$ to obtain another rectangle on frame T_{t+1}, so that the pixel squared difference in the two rectangles is minimized:

$$p^* = argmin_p = \sum_{x \in \mathbb{N}} \|\tau_{t+1}(x+p) - \tau_{t+1}(x)\|^2 \tag{2}$$

Starting with an inital guess p, we can compute optimal p, iteratively. In each iteration, the objective function is locally linearized by

first-order Taylor expansion:

$$\tau_{t+1}(x' + \Delta p) \approx \tau_{t+1}(x') + \frac{\partial \tau_{t+1}(x')}{\partial x'^T} \frac{\partial W(x;p)}{\partial p^T} \Delta p \tag{3}$$

where $\delta p = [\delta p_x, \delta p_y]^T$ is the template offset.

Equation 2 with Equation 3 becomes:

$$p^* = argmin_p = \sum_{x \in \mathbb{N}} \left\| \tau_{t+1}(x') + \frac{\partial \tau_{t+1}(x')}{\partial x'^T} \frac{\partial W(x;p)}{\partial p^T} \Delta p - \tau_{t+1}(x) \right\|^2 \tag{4}$$

These equations can be expressed in vectorized form as such:

$$argmin_{\Delta p} ||A\Delta p - b||_2^2 \tag{5}$$

In this form, A represents:

$$\frac{\partial \tau_{t+1}(x')}{\partial x'^T} \frac{\partial W(x;p)}{\partial p^T} \tag{6}$$

or in a compact form:

$$\nabla I \frac{\partial W}{\partial p} \tag{7}$$

and b is:

$$\tau_{t+1}(x' + \Delta p) - \tau_{t+1}(x') \tag{8}$$

or

$$T(x) - I(W(x;p)) \tag{9}$$

$\frac{\partial W}{\partial p}$ is equal to Jacobian of the function. Since the motion is pure translational:

$$W = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \end{bmatrix} \tag{10}$$

where

$$W_x = x + a \tag{11}$$

$$W_y = y + b \tag{12}$$

Jacobian of the pure translation is equal to:

$$\frac{\partial W}{\partial p^T} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{13}$$

where $\Delta p = [\Delta p_x . \Delta p_y]^T$ is the template offset.

To find minimum p we take derivative of Equation 5 with respect to p:

$$\frac{\partial}{\partial p} \sum (A\Delta p - b)^2 = \sum (2A^T(A\Delta p - b)) = 0 \tag{14}$$

$$\sum (A^T(A\Delta p - b)) = 0 \tag{15}$$

$$\sum (A^T A\Delta p - A^T b) = 0 \tag{16}$$

To have a unique solution, $det(A^T A) \neq 0$ and it should be invertible.

1.2 Lukas Kanade Tracking with Single Template

Lucas Kanade Algorithm is implemented in LukasKanade function with the following signature: LucasKanade(It, It1, rect, p0 = np.zeros(2))

1.3 Implementation

The algorithm is implemented in testCarSequence.py. The example results may be seen in Figure 1

Figure 1: Lucas-Kanade Algorithm for single template tracking

The file to show rectangle locations for each frame, carseqrects.npy file can be found in the submission files.

1.4 Template Correction

As it can be seen from Figure 4, the rectangle shifts to behind after a traffic light has been passed inside the rectange. This phenomen is called as template drifting and the result of the accumulated of the errors. The results are in Figure 4 below:

Figure 2: Lucas-Kanade with Template Correction

2 Lukas Kanade Tracking with Appearance Basis

2.1 Appearance Basis

Starting from Equation

$$\tau_{t+1}(x) = \tau_t(x) + \sum_{k=1}^{K} \omega_k B_k \tag{17}$$

$$\tau_{t+1}(x) - \tau_t(x) = \sum_{k=1}^{K} \omega_k B_k \tag{18}$$

$$\tau_{t+1}(x) - \tau_t(x) = \omega_1 B_1 + ... + \omega_i B_i + ... + \omega_K B_K \tag{19}$$

$$B_i \cdot (\tau_{t+1}(x) - \tau_t(x)) = B_i \cdot (\omega_1 B_1 + ... + \omega_i B_i + ... + \omega_K B_K) \tag{20}$$

Since B_i is orthogonal, if $i \neq j$

$$B_i \cdot B_j = 0 \tag{21}$$

So,

$$B_i \cdot (\tau_{t+1}(x) - \tau_t(x)) = \omega_i \|B_i\|^2 \tag{22}$$

$$\omega_i = \frac{B_i \cdot (\tau_{t+1}(x) - \tau_t(x))}{\|B_i\|^2} \tag{23}$$

2.2 Tracking

2.2.1 Function

Function is implemented in LucasKanadeBasis.py

2.2.2 Implementation

Function is implemented in testSylvSequence.py.

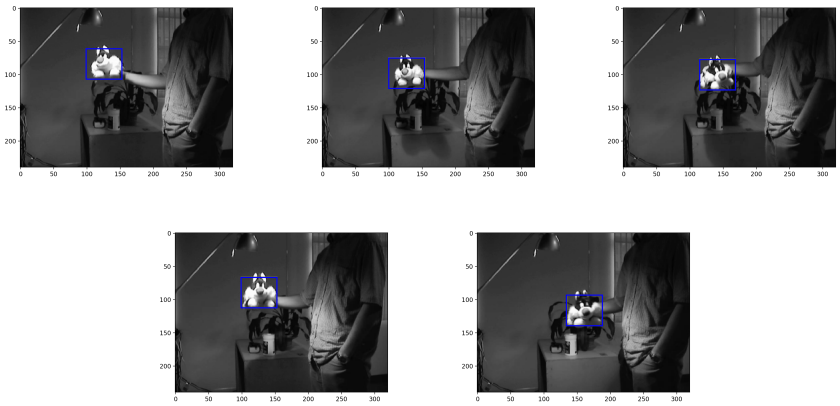

Figure 3: Lucas-Kanad Tracking with Appearance Basis

3 Affine Motion Subtraction

3.1 Dominant Motion Estimation

LucasKanadeAffine.py is implemented

3.2 Moving Object Detection

3.2.1 Function

SubstactDominantMotion.py is implemented.

3.2.2 Implementation

testAerialSequence.py is implemented.

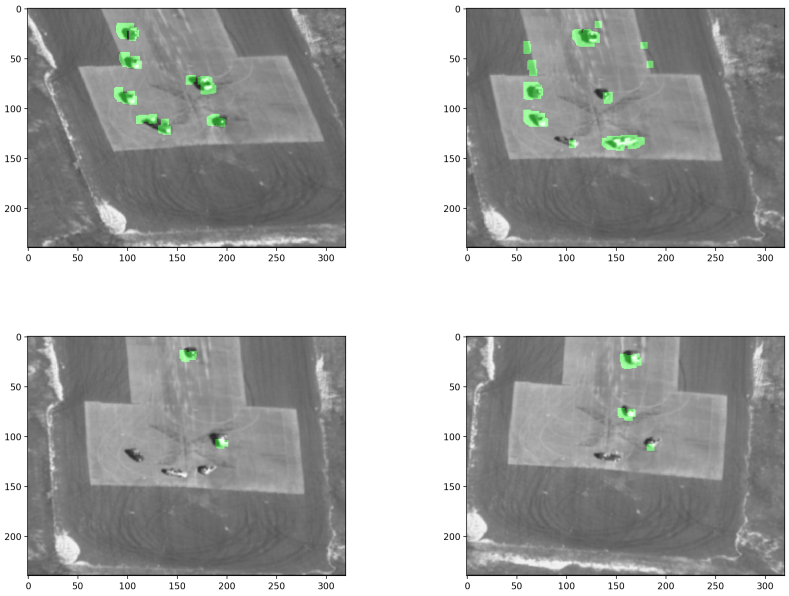

Figure 4: Lucas-Kanade Affine Tracking

4 Efficient Tracking

4.1 Inverse Composition

Lucas Kanade Algorithm is based on Hessian Calculation in each step, which is computationally heavy when we deal with the whole frame. In inverse compositional tracking, the hessian is only computed once and then kept along the iterations, which make this algorithms faster.