Q 1.1 Lucas-Kanade Derivation

We have to find u and v that minimize the following cost function:

$$J(u,v)=\sum_{(x,y)\in R_t}\left(I_{t+1}(x+u,y+v)-I_t(x,y)\right)^2$$

By linearizing $I_{t+1}(x+u,y+v)$ by first order Taylor expansion around the point (x,y) we have:

$$I_{t+1}(x+u,y+v)\simeq I_{t+1}(x,y)+u\cdot(I_{t+1})_x(x,y)+v\cdot(I_{t+1})_y(x,y)$$

In which $u(I_{t+1})_x(x_u,y_v)$ and $v(I_{t+1})_y(x_u,y_v)$ are partial derivatives of I_{t+1} at point (x_u,y_v). Now by replacing $I_{t+1}(x+u,y+v)$ in the above cost function, we have:

$$J(u,v)=\sum_{(x,y)\in R_t}\left(u\cdot(I_{t+1})_x(x,y)+v\cdot(I_{t+1})_y(x,y)+I_{t+1}(x,y)-I_t(x,y)\right)^2$$

Therefore, converting the equation to the matrix form, we need to compute:

$$\arg\min_{u,v}J(u,v)=\arg\min_{u,v}\sum_{(x,y)\in R_t}\left(\begin{bmatrix}(I_{t+1})_x(x,y) & (I_{t+1})_y(x,y)\end{bmatrix}\begin{bmatrix}u\\v\end{bmatrix}-\left[I_{t+1}(x,y)-I_t(x,y)\right]\right)^2=$$

$$\arg\min_{\Delta p}\sum_{(x,y)\in R_t}\left(A\Delta p-b\right)^2$$

To solve this system, we have:

$$\arg\min_{\Delta p}\sum_{(x,y)\in R_t}\left(A\Delta p-b\right)^2\Rightarrow 2\sum_{(x,y)\in R_t}A^T\left(A\Delta p-b\right)=0\Rightarrow\left(\sum_{(x,y)\in R_t}\left(A^TA\right)\right)\Delta p=\sum_{(x,y)\in R_t}A^Tb$$

Where:

$$A^TA=\begin{bmatrix}(I_{t+1})_x^{\,2}(x,y) & (I_{t+1})_x(x,y)\cdot I_y(x,y)\\(I_{t+1})_y(x,y)\cdot(I_{t+1})_x(x,y) & (I_{t+1})_y^{\,2}(x,y)\end{bmatrix}\Rightarrow$$

$$\sum_{(x,y)\in R_t}\left(A^TA\right)=\begin{bmatrix}\sum_{(x,y)\in R_t}(I_{t+1})_x^{\,2} & \sum_{(x,y)\in R_t}(I_{t+1})_x\cdot(I_{t+1})_y\\\sum_{(x,y)\in R_t}(I_{t+1})_x\cdot(I_{t+1})_y & \sum_{(x,y)\in R_t}(I_{t+1})_y^{\,2}\end{bmatrix}$$

and

$$A^T b = -\begin{bmatrix} (I_{t+1})_x(x,y)\cdot\big(I_{t+1}(x,y)-I_t(x,y)\big) \\ (I_{t+1})_y(x,y)\cdot\big(I_{t+1}(x,y)-I_t(x,y)\big) \end{bmatrix} \Rightarrow \sum_{(x,y)\in R_t}\left(A^T b\right) = -\begin{bmatrix} \sum_{(x,y)\in R_t}\Big((I_{t+1})_x\cdot\big(I_{t+1}-I_t\big)\Big) \\ \sum_{(x,y)\in R_t}\Big((I_{t+1})_y\cdot\big(I_{t+1}-I_t\big)\Big) \end{bmatrix}$$

The $A^T A$ is shown above with green color. It is the Gauss-Newton approximation to the Hessian matrix. It is not important if individual $A^T A$ matrices for the points have certain conditions or not; however, the $\sum_{(x,y)\in R_t}\left(A^T A\right)$ matrix should not be singular (should be invertible). Also if it is near to singular conditions (determinant is near to zero), then numerical error in our estimation of *u* and *v* will be high and the template offset cannot be calculated reliably.

Q 1.2 Lucas-Kanade Algorithm

For each iteration we want to find Δu, Δv such that the following cost function is minimized:

$$\arg\min_{\Delta u,\Delta v} J(u+\Delta u, v+\Delta v) = \sum_{(x,y)\in R_t}\Big(I_{t+1}(x+u+\Delta u, y+v+\Delta v) - I_t(x,y)\Big)^2$$

To simplify the notation, I will use (x_u, y_v) instead of $(x+u, y+v)$. By linearizing $I_{t+1}(x+u+\Delta u, y+v+\Delta v)$ by first order Taylor expansion around point $(x+u, y+v)$ we have:

$$I_{t+1}(x+u+\Delta u, y+v+\Delta v) \simeq I_{t+1}(x_u, y_v) + \Delta u\cdot(I_{t+1})_x(x_u, y_v) + \Delta v\cdot(I_{t+1})_y(x_u, y_v)$$

In which $u(I_{t+1})_x(x_u, y_v)$ and $v(I_{t+1})_y(x_u, y_v)$ are partial derivatives of I_{t+1} at point (x_u, y_v). By replacing it in the cost function above we have:

$$\arg\min_{\Delta u,\Delta v} J(u+\Delta u, v+\Delta v) = \sum_{(x,y)\in R_t}\Big(\Delta u\cdot(I_{t+1})_x(x_u, y_v) + \Delta v\cdot(I_{t+1})_y(x_u, y_v) + I_{t+1}(x_u, y_v) - I_t(x,y)\Big)^2$$

To simplify the notation, I will use I instead of I_{t+1} and $D(x,y;u,v)$ instead of $I_{t+1}(x_u, y_v) - I_t(x,y)$:

$$\arg\min_{\Delta u,\Delta v} J(u+\Delta u, v+\Delta v) = \sum_{(x,y)\in R_t}\Big(\Delta u\cdot I_x(x_u, y_v) + \Delta v\cdot I_y(x_u, y_v) + D(x,y;u,v)\Big)^2$$

To find the minimum, differentiate w.r.t. Δu and Δv:

$$\frac{\partial J(u+\Delta u, v+\Delta v)}{\partial \Delta u} = 2\sum_{(x,y)\in R_t}\Big(I_x(x_u, y_v)\Big)\Big(\Delta u\cdot I_x(x_u, y_v) + \Delta v\cdot I_y(x_u, y_v) + D(x,y;u,v)\Big) = 0$$

$$\frac{\partial J(u+\Delta u, v+\Delta v)}{\partial \Delta v} = 2\sum_{(x,y)\in R_t}\Big(I_y(x_u, y_v)\Big)\Big(\Delta u\cdot I_x(x_u, y_v) + \Delta v\cdot I_y(x_u, y_v) + D(x,y;u,v)\Big) = 0$$

Rewriting the above equations in matrix notation and eliminating the constant multiplier 2, we have:

$$\sum_{(x,y)\in R_t}\left(\begin{bmatrix} I_x(x_u,y_v)\cdot I_x(x_u,y_v) & I_x(x_u,y_v)\cdot I_y(x_u,y_v) \\ I_y(x_u,y_v)\cdot I_x(x_u,y_v) & I_y(x_u,y_v)\cdot I_y(x_u,y_v) \end{bmatrix}\begin{bmatrix}\Delta u \\ \Delta v\end{bmatrix}+\begin{bmatrix} I_x(x_u,y_v)\cdot D(x,y;u,v) \\ I_y(x_u,y_v)\cdot D(x,y;u,v)\end{bmatrix}\right)=\begin{bmatrix}0\\0\end{bmatrix}\Rightarrow$$

$$\left(\sum_{(x,y)\in R_t}\begin{bmatrix} I_x(x_u,y_v)\cdot I_x(x_u,y_v) & I_x(x_u,y_v)\cdot I_y(x_u,y_v) \\ I_y(x_u,y_v)\cdot I_x(x_u,y_v) & I_y(x_u,y_v)\cdot I_y(x_u,y_v) \end{bmatrix}\right)\begin{bmatrix}\Delta u \\ \Delta v\end{bmatrix}=-\sum_{(x,y)\in R_t}\begin{bmatrix} I_x(x_u,y_v)\cdot D(x,y;u,v) \\ I_y(x_u,y_v)\cdot D(x,y;u,v)\end{bmatrix}$$

The code that implements this formulation is provided as requested. To increase speed, I used efficient matrix operations in the algorithm. I also implemented the step-by-step algorithm Lucas-Kanade algorithm of [1] and the Inverse Compositional algorithm of [2]. All the codes work in real-time; however, I did not find much difference in performance between the three implementations. I commented the first two implementations and let the Inverse Compositional version stay uncommented.

Q 1.3 Testing Lucas-Kanade Algorithm

The results are shown in Figure 1.

Figure 1. The result of Lucas-Kanade tracking algorithm on frames 1, 100, 200, 300 and 400 of the given sequence.

Q 1.4 Lucas-Kanade Algorithm with Template Correction

The results are shown in Figure 2. It is clearly shown how the corrected template (yellow box) is able to track the car without a drift, while the normal Lucas-Kanade template (green box) drifts from the correct position after a while.

Figure 2. The result of Lucas-Kanade tracking algorithm with template correction (yellow box) vs the simple Lucas-Kanade algorithm (green box) of Q1.3 on frames 1, 100, 200, 300 and 400 of the given sequence.

Q 2.1 Lucas-Kanade Tracking with Appearance Basis

We want to find weights w_c ($c = 1,2,\ldots,k$) in the following equation:

$$I_{t+1} = I_t + \sum_{c=1}^{k} w_c B_c \Rightarrow \sum_{c=1}^{k} w_c B_c = I_{t+1} - I_t$$

Since the bases are orthogonal and of the same size, to calculate a weight w_c we can multiply both sides of the equation by B_c:

$$B_c \cdot (I_{t+1} - I_t) = B_c \cdot \left(\sum_{c=1}^{k} w_c B_c \right) = B_c \cdot (w_1 B_1 + \ldots + w_c B_c + \ldots + w_k B_k) =$$

$$B_c \cdot w_1 B_1 + \ldots + B_c \cdot w_c B_c + \ldots + B_c \cdot w_k B_k = 0 + \ldots + \|B_c\|^2 w_c + \ldots 0 \Rightarrow$$

$$w_c = \frac{B_c}{\|B_c\|^2} \cdot (I_{t+1} - I_t)$$

Therefore, for each $c = 1,2,\ldots,k$ we can compute w_c as $w_c = \frac{B_c}{\|B_c\|^2} \cdot (I_{t+1} - I_t)$.

Q 2.2-2.3 Implementation of L-K Tracking with Appearance Basis

The results are shown in Figure 3. On the given sequence, both the simple Lucas-Kanade and the Lucas-Kanade with Appearance Basis algorithms can track the toy perfectly.

Figure 3. The result of Lucas-Kanade tracking algorithm with appearance basis (yellow box) of Q 2.2 vs. the simple Lucas-Kanade algorithm (green box) of Q1.3 on frames 1, 200, 300, 350 and 400 of the given sequence.

The results on the faster FPS are shown in Figure 4. With faster FPS, the algorithm skips 4 frames to obtain the next frame (i.e. it tracks the toy in frames 1, 6, 11, 16, etc.) in order to provide a harder problem to be able to show differences between the two algorithms.

On the given sequence, with faster FPS, the simple Lucas-Kanade loses the toy in the middle of the sequence, while the Lucas-Kanade with Appearance Basis algorithm can track the toy without losing it.

Figure 4. The result of Lucas-Kanade tracking with appearance basis (yellow box) of Q 2.2 vs. the simple Lucas-Kanade algorithm (green box) of Q1.3 on frames 1, 200, 300, 350 and 400 of the given sequence with faster FPS.

Q 3.1 – 3.3 Affine Motion Subtraction

The results are shown in Figure 5. I used the warping function (warpH) from previous homeworks to warp images.

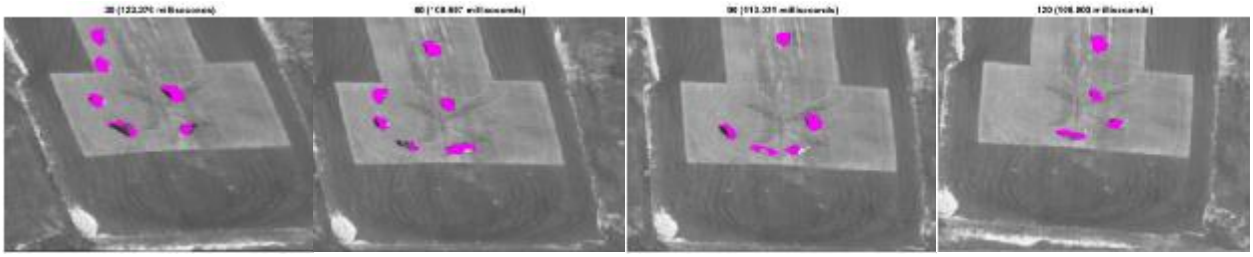

Figure 5. The result of Lucas-Kanade affine algorithm on frames 30, 60, 90 and 120 of the given sequence.