HW3 Write-Up

16-720: Computer Vision

Spring 2016

Cole Gulino

March 3, 2016

1.Lucas-Kanade Tracking

Q1.1: Starting with an initial guess of (u, v) (for instance, (0, 0)), we can compute the optimal (u^∗ , v^∗) iteratively. In each iteration, the objective function is locally linearized by first-order Taylor expansion and optimized by solving a linear system that has the form A∆p = b, where ∆p = (u, v)^T , the template offset.

- Linearize equation 1 and write it in the form $\min_{\Delta p} (A\Delta p - b)^2$. What is $A^T A$?
- What conditions must $A^T A$ meet so that the template offset can be calculated reliably?

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

Equation 1

By taking the first order approximation of Equation 1 we get the following [1]:

$$\sum_{(x,y)\in R_t} \left(uI_{t+1,x}(x,y) + vI_{t+1,y}(x,y) + [I_{t+1}(x,y) - I_t(x,y)]\right)^2$$

$$\sum_{(x,y)\in R^t} [uI_x(x,y) + vI_y(x,y) - D(x,y)]^2$$

Where:

$$I_x = \frac{\partial I_{t+1}(x,y)}{\partial x}$$

$$I_y = \frac{\partial I_{t+1}(x,y)}{\partial y}$$

$$D(x,y) = I_{t+1}(x,y) - I_t(x,y)$$

We can then put this equation into the form $\min_{\Delta p} (A\Delta p - b)^2$:

$$\min_{\Delta p} \left(\begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} - D(x,y) \right)^2$$

Where: $A = \begin{bmatrix} I_x & I_y \end{bmatrix}$; $b = D(x,y)$

If we parse out the square, we get:

$$\sum_{(x,y)\in R_t} \left(\begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} - \begin{bmatrix} I_xD \\ I_yD \end{bmatrix} \right)$$

This shows that A^TA is the Hessian Matrix:

$$A^TA = \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}$$

According to [2] updating the parameters which corresponds to the template offset, is updated according to the equation. As seen above we can formulate the equation to find (u,v) as [1]:

$$\min_{\Delta p} \left(A^TA\Delta p - A^Tb \right)$$

So we can set this to zero:

$$A^TA\Delta p = A^Tb$$

$$\Delta p = \left(A^TA\right)^{-1} A^Tb$$

Where: $$\Delta p = \begin{bmatrix} u \\ v \end{bmatrix}$$

So in order for the template offset to be calculated correctly, A^TA, the image Hessian, must be invertible.

Q1.2: Implement a function with the following signature:

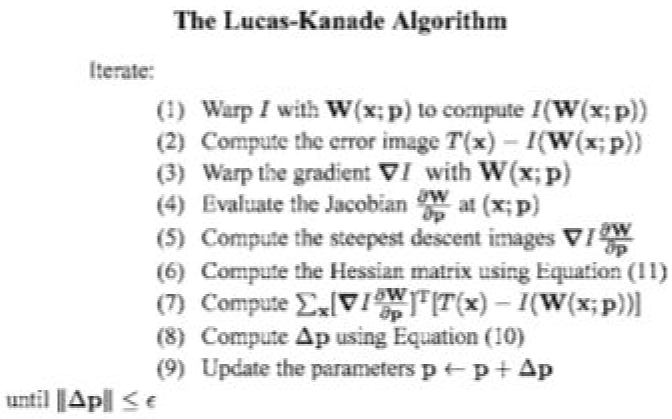
```
[u,v] = LucasKanade(It, It1, rect)
```

This computes the optimal local motion from frame I_t to frame I_{t+1} that minimizes Equation 1. `rect` is the 4-by-1 vector that represents a rectangle on the image frame I_t. The four components of the rectangle are $[x_1, y_1, x_2, y_2]$. Where (x_1, y_1) is the top-left corner and $(x_2, y_2)$is the bottom right corner.

I iterated the algorithm over and over again in order to find a valid (u,v) for each of the pixels in the image. I updated the (u,v) by the following:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \left(\begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \right)^{-1} \begin{bmatrix} I_x\left(I_{t+1}(x,y) - I_t(x,y)\right) \\ I_y\left(I_{t+1}(x,y) - I_t(x,y)\right) \end{bmatrix}$$

I implemented the algorithm similar to the paper, and I used the following algorithm from [2].



The Lucas-Kanade Algorithm

```
Iterate:
    (1) Warp $I$ with $\mathbf{W}(\mathbf{x};\mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
    (2) Compute the error image $T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
    (3) Warp the gradient $\nabla I$ with $\mathbf{W}(\mathbf{x};\mathbf{p})$
    (4) Evaluate the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ at $(\mathbf{x};\mathbf{p})$
    (5) Compute the steepest descent images $\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$
    (6) Compute the Hessian matrix using Equation (11)
    (7) Compute $\sum_{\mathbf{x}}[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^T[T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))]$
    (8) Compute $\Delta\mathbf{p}$ using Equation (10)
    (9) Update the parameters $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$
until $\|\Delta\mathbf{p}\| \leq \epsilon$
```

I set ϵ = `0.0000001`.

Q1.3: Write a test script:

`testCarSequence.m`

This script takes the images from the `carseq.mat` dataset and takes images of two consecutive frames and calculates the u and v between them using the `LucasKanade` function implemented in Q1.2. It then stores the rectangle that it calculates from the translation in a matrix called `rects`. This will be stored in a file called `carseqrects.mat`. Figure 1 shows the tracking performance at frames 1, 100, 200, 300, and 400.

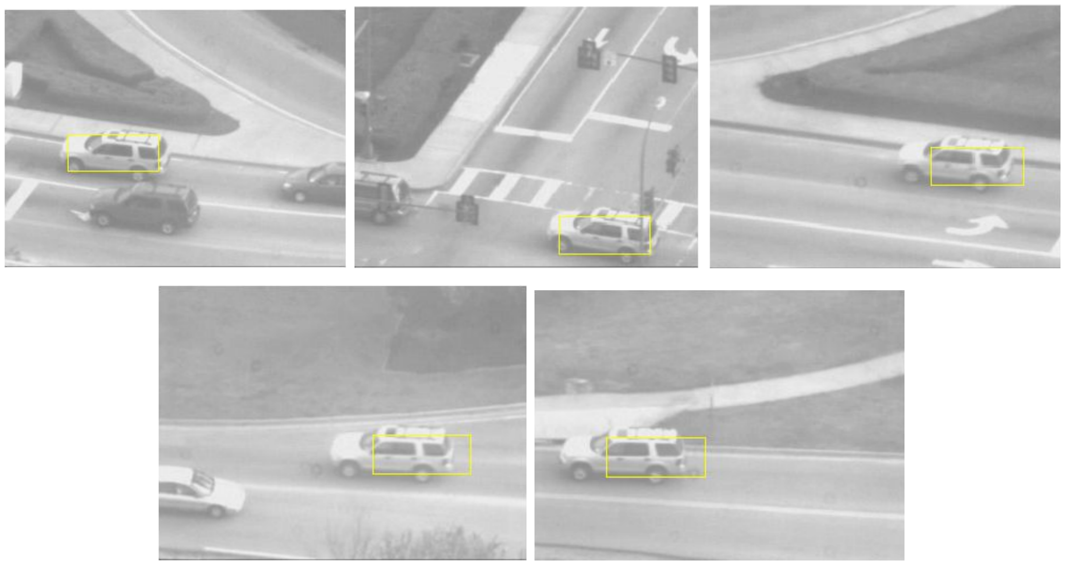


Figure 1) Tracking Performance at Frames 1, 100, 200, 300, and 400

Q1.4: Extra Credit

Implement a script that implements a method to reduce template drifting. The script that I wrote is called:

`testCarSequenceWithTemplateCorrection.m`

In order to implement this, I needed to create another variation of the original Lucas-Kanade implementation that I wrote in Q1.2. This function has the signature:

`[u,v] = LucasKanadeWTC(T, It1, rect)`

`T` is the template of the original image. It sends the portion of the original image in the original rectangle. `It1` and `rect` are the same as in Q1.2. The function sends the rectangle from the previous frame, but finds the minimization of the sum of squared difference between the rectangle in the current frame and the original frame's template. This way, the errors will not accumulate between the frames. I believe that this works because we are only considering translations. I think that this method would have poor performance tracking an object that could deform. A method that would work for a deformable object would be similar to the method implemented in Q2.2.

Figure 2 shows the tracking performance using the method in Q1.2 in green and the method in Q1.4 in yellow.

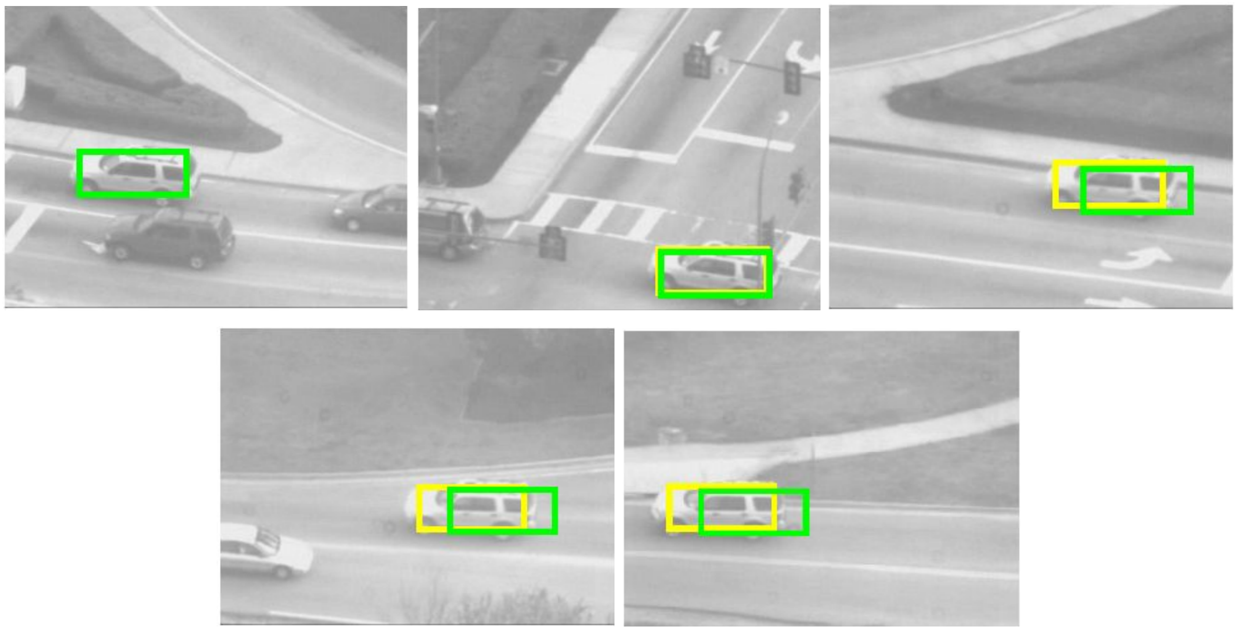


Figure 2) Tracking Performance at Frames 1, 100, 200, 300, and 400 with and without Tracking Correction

2. Lucas-Kanade Tracking with Appearance Basis

Q2.1: Express w_c for $c = 1, 2, ..., k$, as a function of I_{t+1} , I_t , and $\{B_c\}_{c=1}^{k}$, given Equation 2.

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

Equation 2

Based on intuition from [3], we can move every portion of the equation to one side.

$$I_{t+1} - I_t - \sum_{c=1}^{k} w_c B_c = 0$$

Assuming that we treat the images as vectors and that we are trying to minimize this equation, we can put this equation into the form:

$$\left\| I_{t+1} - I_t - \textstyle\sum_{c=1}^{k} w_c B_c \right\|^2_{\{B_c\}_{c=1}^{k}}$$

Assuming that $\{B_c\}_{c=1}^{k}$ are orthonormal (which if not, can be orthonormalized using Gramm-Schmidt as mentioned in [3]), the term above has a closed form solution:

$$w_c = \sum_{(x,y) \in R_t} B_c(x,y) \left[I_{t+1}(x,y) - I_t(x,y) \right] \text{ for } c = 1, 2, ..., k$$

Q2.2: Implement a function with the following signature:

```
[u,v] = LucasKanadeBasis(It, It1, rect, bases)
```

`bases` is a three-dimensional matrix that contains the bases. It has the same format as can be found in frames.

When implementing this algorithm, I went back to the paper. In [3], the authors specify how you can get the desired properties with the basis by changing how you calculate the Hessian. In my implementation, I calculated the Hessian as specified in the paper, as shown below:

$$H_Q = \sum_{\mathbf{x}} \mathbf{SD}_Q(\mathbf{x})\, \mathbf{SD}_Q^{\mathrm{T}}(\mathbf{x})$$

Where:

$$\mathbf{SD}_Q(\mathbf{x}) = \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{x};\mathbf{0}) - \sum_{i=1}^{m} \left[\sum_{\mathbf{y}} A_i(\mathbf{y}) \cdot \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{y};\mathbf{0}) \right] A_i(\mathbf{x})$$

Q2.3: Write a test script:

`testSylvSequence.m`

This loads the video frames from `sylvseq.mat` and runs the Lucas-Kanade Tracker implemented in Q2.2 to track the toy with the color yellow. The test script also runs the tracker from the Lucas-Kanade Tracker implemented in Q1.2 with the color green. This script also places the rectangles calculated in a matrix called `rects` in a file called `sylvseqrects.mat`.

Figure 3 shows the performance of the tracker at frames 1, 200, 300, 350, and 400.

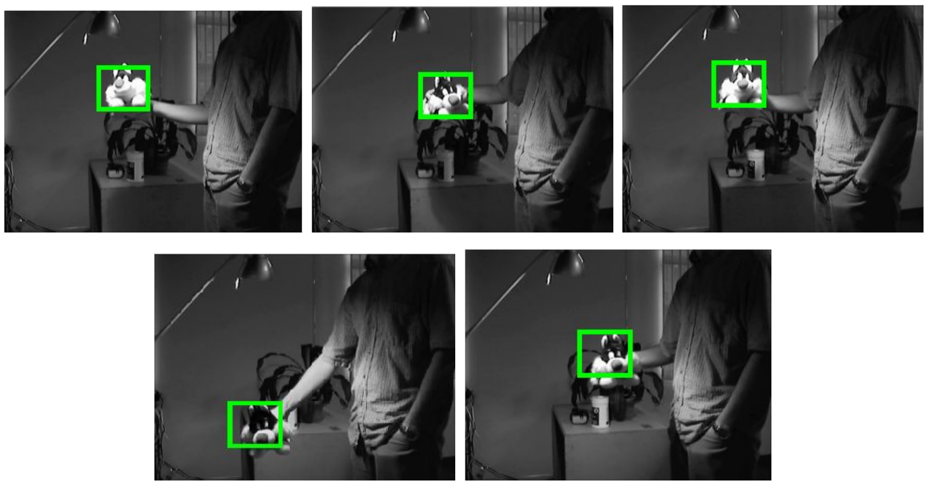


Figure 3) Lucas-Kanade Tracking with Appearance Basis at Frames 1, 200, 300, 350, and 400

3. Affine Motion Subtraction

Here we are assuming that the two images I_t and I_{t+1} are related by an affine warp. In this way, I_{t+1} is approximately an affine warped version of I_t. This assumes that the majority of the pixels correspond to the stationary objects in the scene whose depth variation is small relative to their distance to the camera.

The parameters for the affine warp are: $\Delta p = [a, b, c, d, e, f]^T$, where the affine warp can be specified as:

$$M = \begin{bmatrix} 1+a & b & c \\ d & 1+e & f \\ 0 & 0 & 1 \end{bmatrix}$$

We can then relate the images with:

$$\mathbf{x}_{t+1} = M\mathbf{x}_t$$

Where:

$$\mathbf{x} = [x, y, 1]^T$$

M will be different between each set of images. Each update of the affine parameter vector is computed via a least-squares method using pseudo-inverse as described in the class.

The matrix of image derivatives and the temporal derivatives must be computed only on the pixels lying in the region common to I_t and the warped version of I_{t+1}.

Q3.1: Write a function with the following signature:

```
M = LucasKanadeAffine(It, It1)
```

This function is very similar to the function written in Q1.2, except that it assumes an affine warp instead of a simple translation.

To solve this, I need to come up with the warp and the warp jacobian.

As seen above, I can use the equation:

$$M = \begin{bmatrix} 1+a & b & c \\ d & 1+e & f \\ 0 & 0 & 1 \end{bmatrix}$$

to specify the warp.

I can put these two equations in the form:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1+a & b & c \\ d & 1+e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

And then:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} x(1+a)+yb+c \\ xd+y(1+e)+f \\ 1 \end{bmatrix}$$

Now I can calculate the warp jacobian:

$$\mathbf{W} = \begin{bmatrix} u \\ v \end{bmatrix}$$

$$\mathbf{p} = \begin{bmatrix} a & b & c & d & e & f \end{bmatrix}$$

And then:

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & y & 1 \end{bmatrix}$$

As specified in [2], I calculated the steepest descent images:

$$\Delta I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} x & y & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & y & 1 \end{bmatrix}$$

$$\Delta I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{bmatrix} xI_x & yI_x & I_x & xI_y & yI_y & I_y \end{bmatrix}$$

I then used this to calculate the Hessian and the Δp:

$$H = \sum_{(x,y)} \left[\Delta I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^T \left[\Delta I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]$$

$$\Delta \mathbf{p} = H^{-1} \sum_{(x,y)} \left[\Delta I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^T \left[I_{t+1}\left(\mathbf{W}(x,y,\mathbf{p})\right) - I_t\right]$$

Q3.2: Write a function with the following signature:

```
mask = SubtractDominantMotion(image1, image2)
```

`image1` and `image2` form the input image pair and `mask` is the binary image of the same size that dictates which pixels are considered to be corresponding to moving objects.

Here we use the function `LucasKanadeAffine` written in Q3.1 to derive a transformation matrix M that warps I_t so that it is registered to I_{t+1} and then find the difference between the two. Where the absolute distance exceeds some threshold is where the moving objects are.

I used `imdialate` in order to enlarge large disks of my difference image, which is what I believe does a good job of estimating the large blobs that correspond to a car.

I also used `imerode` in order to erode away small disks in my difference image, which got rid of some of the outliers that looked like blobs that were expanded by the `imdialate`. I also, used `imdialate` on lines of various angles in order to remove the outliers along the edges.

Finally, I used `bwareaopen` to remove objects that are not well connected and then use `bwselect` to enhance those that are well connected.

Q3.3: Write the test script:

`testAerialSequence.m`

This script loads an image sequence from `aerialseq.mat` that runs the motion detection routine developed in Q3.2. Figure 4 shows the motion detection performance at frames 30, 60, 90, and 120.

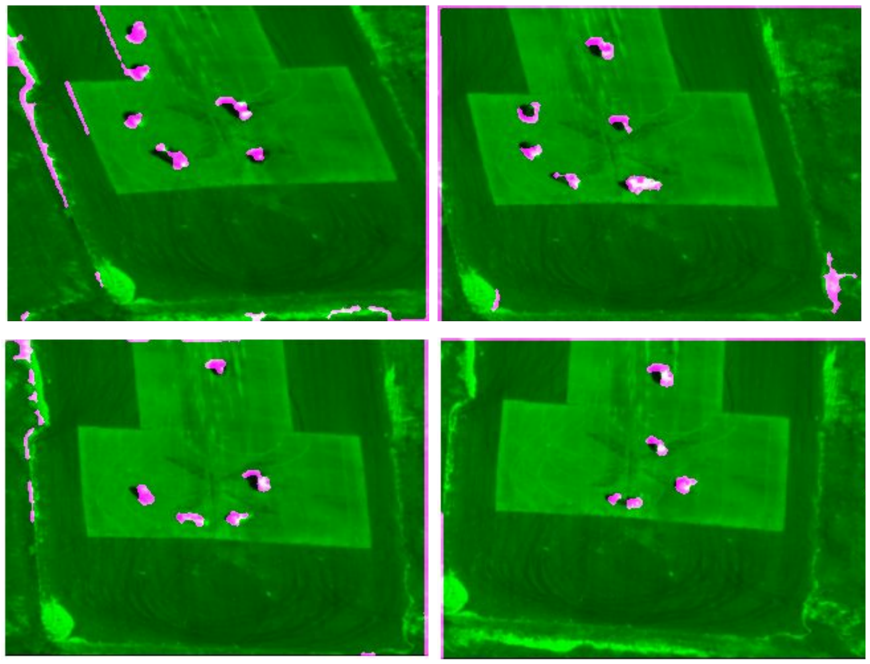


Figure 4) Motion Detection Performance at Frames 30, 60, 90, and 120

References

[1] Course Alignment Notes

[2] Lucas-Kanade 20 Years On: A Unifying Framework: Part 1

[3] Lucas-Kanade 20 Years On: A Unifying Framework: Part 2