CS 83/183: Assignment 4

March 3, 2024

Jessie Li

2 Theory Questions

Q2.1 Calculating the Jacobian

If the warp is defined as the 3×3 matrix $\mathbf{W}(\mathbf{p})$, where

$$\mathbf{W}(\mathbf{p}) = \begin{bmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \\ 0 & 0 & 1 \end{bmatrix}$$

then for each pixel,

$$\nabla \mathbf{I} = \begin{bmatrix} \mathbf{I}_u & \mathbf{I}_v \end{bmatrix} \quad (1 \times 2)$$

$$\frac{\delta \mathbf{W}}{\delta \mathbf{p}} = \begin{bmatrix} u & 0 & v & 0 & 1 & 0 \\ 0 & u & 0 & v & 0 & 1 \end{bmatrix} \quad (2 \times 6)$$

$$\begin{aligned} J &= \nabla \mathbf{I} \frac{\delta \mathbf{W}}{\delta \mathbf{p}} \\ &= \begin{bmatrix} \mathbf{I}_u & \mathbf{I}_v \end{bmatrix} \begin{bmatrix} u & 0 & v & 0 & 1 & 0 \\ 0 & u & 0 & v & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} u\mathbf{I}_u & u\mathbf{I}_v & v\mathbf{I}_u & v\mathbf{I}_v & \mathbf{I}_u & \mathbf{I}_v \end{bmatrix} \quad (1 \times 6) \end{aligned}$$

Q2.2 Computational complexity

For the Matthews-Baker method, the gradient of the template $\nabla\mathbf{T}$, Jacobian $\mathbf{J}$, and the Hessian $\mathbf{H}$ are constants that can be precomputed. If n is the number of pixels in the template $\mathbf{T}$ and p is the number of warping parameters in W,

- Computing $\nabla\mathbf{T}$ is O(n).
- Computing $\frac{\delta\mathbf{W}}{\delta p}$ is O(np).
- The Jacobian of the error function L is calculated as $\mathbf{J} = \nabla\mathbf{T}\frac{\delta\mathbf{W}}{\delta p}$. Per pixel, multiplying the 1×2 gradient $\nabla\mathbf{T}$ with the $2\times p$ Jacobian of the warp $\frac{\delta\mathbf{W}}{\delta p}$ requires O(p) operations. With n pixels, computing $\mathbf{J}$ therefore takes O(np) time (also accounting for the time to compute $\nabla\mathbf{T}$ and $\frac{\delta\mathbf{W}}{\delta p}$).
- The 6×6 Hessian $\mathbf{H}$ is computed as $\mathbf{J}^\top\mathbf{J}$, where $\mathbf{J}$ is an $n \times 6$ matrix with each row corresponding to the Jacobian evaluated at a pixel. Counting n multiplications for each of the p columns in $\mathbf{J}$, for p rows in $\mathbf{J}^\top$, computing $\mathbf{H}$ is $O(np^2)$.

The Matthews-Baker inverse compositional algorithm, is much more efficient than the Lucas-Kanade algorithm because it performs time-consuming calculations before iterating. Whereas Matthews-Baker precomputes $\nabla\mathbf{T}$, $\mathbf{J}$, and $\mathbf{H}$, Lucas-Kanade must recompute these values in each iteration.

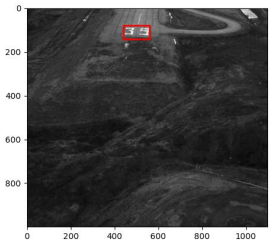
The update step in Matthews-Baker is slightly more expensive because it requires inverting $\mathbf{W}(\Delta\mathbf{p})$ and multiplying with $\mathbf{W}(\mathbf{p})$ to get the new warp function for the next iteration, but the cost of these two steps is at most O(p^3), negligible since $p << n$.

At a small cost of O(p^3) to update, Matthews-Baker Matthews-Baker skips the $O(np^2)$ computation of $\nabla\mathbf{T}$, Jacobian $\mathbf{J}$, and the Hessian $\mathbf{H}$ in each iteration of Lucas-Kanade, computing these values only once in precomputation. This significantly reduces time per iteration and time overall.

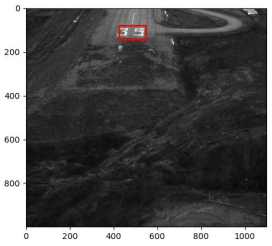
3 Implementation

Q3.1 Lucas-Kanade forward additive alignment with translation

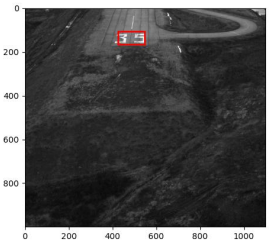
`landing`



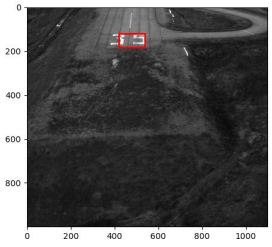
Frame 1



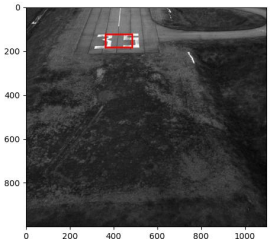
Frame 12



Frame 24



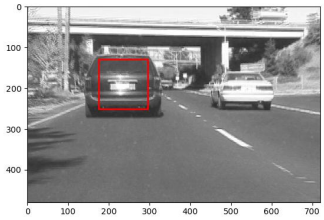
Frame 36



Frame 48

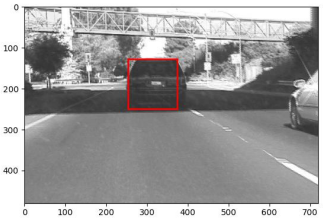
`car1`

Frame 1



Frame 50

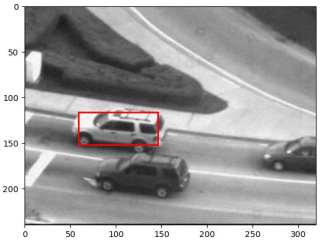
Frame 120



Frame 170

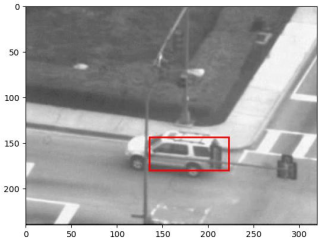
Frame 250

`car2`

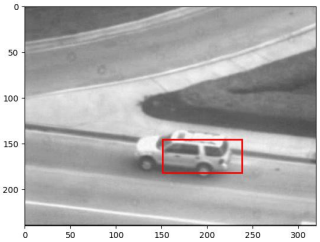


Frame 1

Frame 70

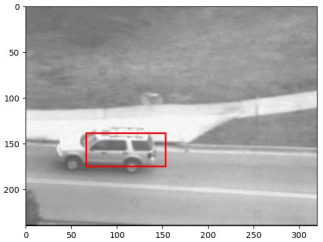


Frame 150



Frame 220

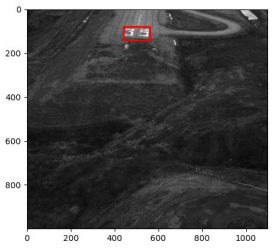
Frame 310



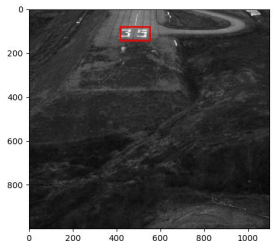
Frame 410

Q3.2 Lucas-Kanade forward additive alignment with affine transformation

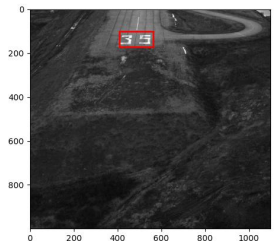
`landing`



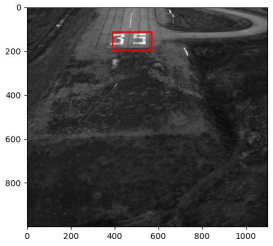
Frame 1



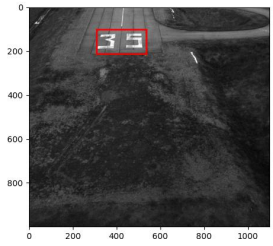
Frame 12



Frame 24



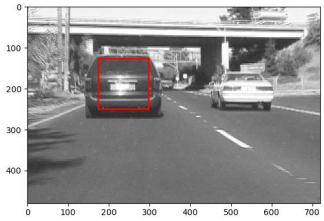
Frame 36



Frame 48

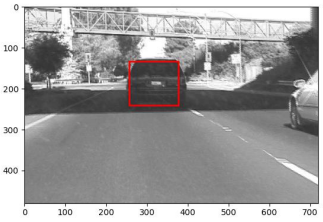
`car1`

Frame 1



Frame 50

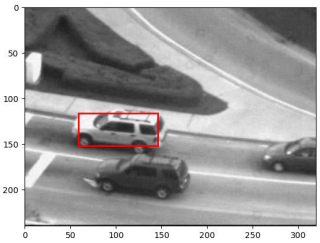
Frame 120



Frame 170

Frame 250

`car2`

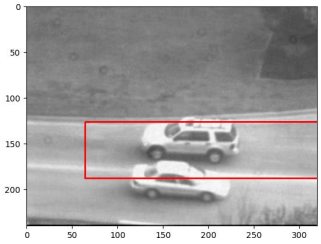


Frame 1

Frame 70

Frame 150

Frame 220

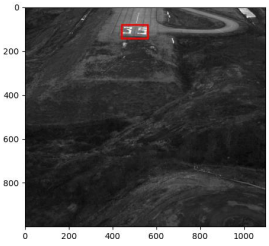


Frame 310

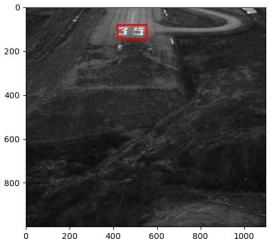
Frame 410

Q3.3 Inverse compositional alignment with affine transformation

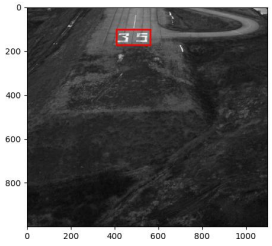
`landing`



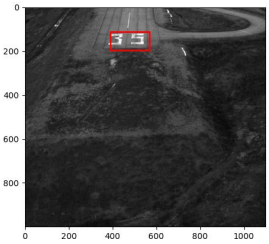
Frame 1



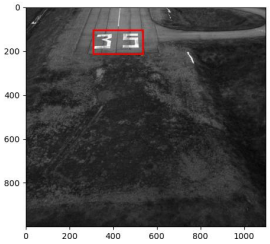
Frame 12



Frame 24



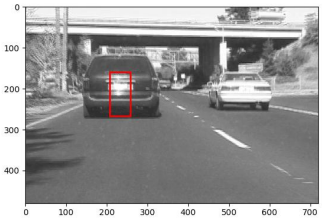
Frame 36



Frame 48

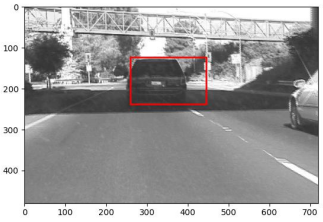
`car1`

Frame 1



Frame 50

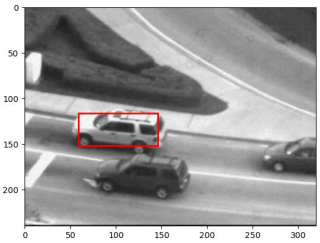
Frame 120



Frame 170

Frame 250

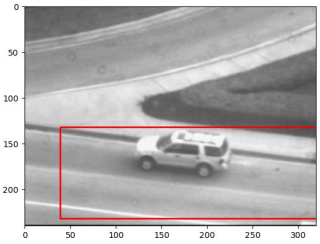
`car2`



Frame 1

Frame 70

Frame 150



Frame 220

Frame 310

Frame 410

Q3.4 Testing the algorithms

Speed

- Lucas-Kanade with pure translation is consistently faster than the affine warp algorithms. This makes sense because the warp for this method has only two parameters, rather than six, and maintains a constant window size. As error accumulates, the affine methods tend to expand the window, increasing the run time.
- The affine Lucas-Kanade forward additive algorithm generally took longer than the Matthews-Baker inverse compositional algorithm (particularly for `car2`) because it computes the gradient, Jacobian, and Hessian in each iteration, whereas Matthews-Baker precomputes these quantities once. Detailed reasoning provided in the answer to the second theory question.
- Per frame, `landing` for all three algorithms seemed to take the least amount of time, likely because the target window was smaller, and because changes in position and brightness were relatively small between consecutive frames.

Accuracy

`landing`

- Both affine algorithms captured the target fairly well.
- Lucas-Kanade with pure translation was less accurate because it assumes constant flow for all pixels in the window, but this assumption doesn't hold here. With four additional parameters to allow for scaling, the affine models were able to describe the movement of pixels in the window with greater accuracy.

`car1`

- Lucas-Kanade with only translation tracked the target with reasonable accuracy. Although it failed to adjust for the size of the car decreasing as it moves further away, the window computed by this method still remained relatively centered on the target, even through changes in brightness.
- Lucas-Kanade with an affine warp performed slightly better than both Matthews-Baker and Lucas-Kanade with only translation in terms of accuracy. The shape of the Lucas-Kanade window stayed relatively consistent and tight around the target from frame to frame. In Frame 250, the window calculated by Lucas-Kanade with pure translation is slightly offset to the bottom and left. The Matthews-Baker frame is clearly too wide, but the Lucas-Kanade affine window seems fairly on-target.

`car2`

- Pure translation Lucas-Kanade seemed to track the car in this video most accurately. Although the window started falling behind the car, particularly after Frame 150, the tracker still followed the car

with reasonable accuracy. This may be partially due to the car maintaining relatively constant size, indicating that the pixel velocities were similar within the frame.

- The affine forward additive and inverse compositional algorithms showed similar performance on this video. Both broke down when objects, like the rightmost traffic light in Frame 150 block the target, if only temporarily. In each frame after the obstacle is introduced, the window stretched in the next frame to include it. To improve the run time and performance of both algorithms, I constrained the rectangle to always be within the bounds of the template. Both algorithms still lost significant precision by Frame 220.

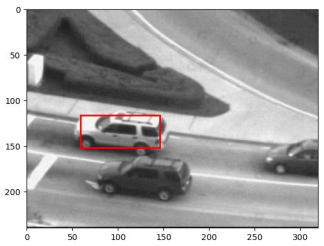
Limitations

- These tracking algorithms break down when the target moves too quickly between frames. Derived from a Taylor approximation, gradient descent assumes that $\mathbf{\Delta p}$ is small and finds a local minimum nearby, which may not be the global minimum.
- The algorithms are also sensitive to sudden changes in the brightness of the target because changes in brightness affect the sum of squared distances they try to minimize.
- Error tends to accumulate, particularly when the warp is an affine transformation, because the target window in each frame is determined by the window calculated in previous frame.
- The algorithms are not robust to occlusions because they change pixel intensities within the template.
- Flat images with too little texture or contrast may also be reduce accuracy because more patches would be likely to produce an error lower than the threshold.

4 Extra Credit

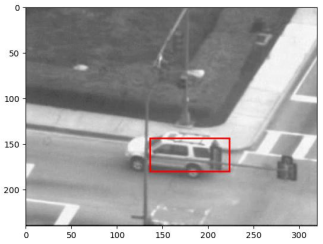
4.2 Lucas-Kanade tracking on an image pyramid

I implemented the Lucas-Kanade tracker from Q3.1 on a 4-layer image pyramid (with scales 1, 1/2, 1/4, and 1/8) instead of a single image. Each layer was generated with the `pyrDown` function provided by OpenCV. Running the algorithm on an image pyramid significantly improved run-time and slightly improved accuracy.

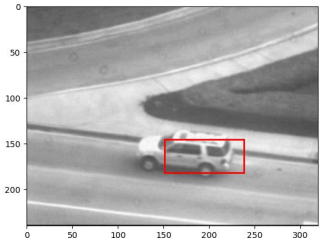


Frame 1

Frame 70

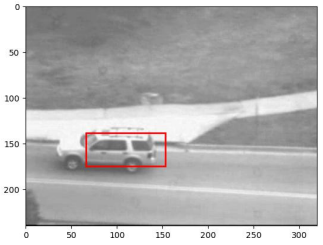


Frame 150



Frame 220

Frame 310



Frame 410

Results of running Lucas-Kanade with translation only on an image pyramid for `car2`.