16720 — PS3

Connor W. Colombo (cwcolomb)

October 22nd 2020

1 Lucas-Kanade Tracking

Q1.1

$$\text{let: } \mathcal{I}_{t+1}(\mathbf{x}'+\Delta\mathbf{p}) \approx \mathcal{I}_{t+1}(\mathbf{x}') + \frac{\partial \mathcal{I}_{t+1}(\mathbf{x}')}{\partial \mathbf{x}'^T}\frac{\partial \mathcal{W}(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}^T}\Delta\mathbf{p}$$

$$\text{where } \mathbf{p} = \begin{bmatrix} p_x \\ p_y \end{bmatrix},\ \mathbf{x}' = \mathcal{W}(\mathbf{x};\mathbf{p}) = \mathbf{x}+\mathbf{p},\ \frac{\partial \mathcal{I}_{t+1}(\mathbf{x}')}{\partial \mathbf{x}'^T} = \begin{bmatrix} \mathcal{I}_{t+1_x}(\mathbf{x}') \\ \mathcal{I}_{t+1_y}(\mathbf{x}') \end{bmatrix}$$

Let the update rule for $\mathbf{p}$ be given by:

$$\mathbf{p} \leftarrow \mathbf{p} + \arg\min_{\Delta\mathbf{p}} \|\mathbf{A}\Delta\mathbf{p} - \mathbf{b}\|_2^2$$

a. What is $\frac{\partial \mathcal{W}(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}^T}$?

$\frac{\partial \mathcal{W}(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}^T}$ is the Jacobian of $\mathcal{W}(\mathbf{x};\mathbf{p})$ with respect to $\mathbf{p}$. Thus,

$$\frac{\partial \mathcal{W}(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}^T} = \begin{bmatrix} \frac{\partial \mathcal{W}_x}{\partial \mathbf{p}_1} & \frac{\partial \mathcal{W}_x}{\partial \mathbf{p}_2} \\ \frac{\partial \mathcal{W}_y}{\partial \mathbf{p}_1} & \frac{\partial \mathcal{W}_y}{\partial \mathbf{p}_2} \end{bmatrix} = \frac{\partial}{\partial \mathbf{p}}\begin{bmatrix} x+p_1 \\ y+p_2 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I$$

$$\therefore \boldsymbol{\frac{\partial \mathcal{W}(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}^T} = I_2}$$

b. What is A and b?

The iterative extension of optimization equation (2) given in the assignment as:

$$\mathbf{p}^* = \arg\min_{\mathbf{p}} \sum_{\mathbf{x}\in\mathbb{N}} \|\mathcal{I}_{t+1}(\mathbf{x}+\mathbf{p}) - \mathcal{I}_t(\mathbf{x})\|_2^2$$

is simply

$$\mathbf{p} \leftarrow \mathbf{p} + (\Delta\mathbf{p})^* \quad | \quad (\Delta\mathbf{p})^* = \arg\min_{\Delta\mathbf{p}} \sum_{\mathbf{x}\in\mathbb{N}} \|\mathcal{I}_{t+1}(\mathbf{x}'+\Delta\mathbf{p}) - \mathcal{I}_t(\mathbf{x})\|_2^2$$

This can be linearized by employing the locally linearized first-order Taylor expansion of $\mathcal{I}_{t+1}(\mathbf{x}'+\Delta\mathbf{p})$ (given above) to be:

$$(\Delta\mathbf{p})^* = \arg\min_{\Delta\mathbf{p}} \sum_{\mathbf{x}\in\mathbb{N}} \left\| \mathcal{I}_{t+1}(\mathbf{x}') + \frac{\partial \mathcal{I}_{t+1}(\mathbf{x}')}{\partial \mathbf{x}'^T}\frac{\partial \mathcal{W}(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}^T}\Delta\mathbf{p} - \mathcal{I}_t(\mathbf{x}) \right\|_2^2$$

$$\because \frac{\partial \mathcal{W}(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}^T} = I_2$$

$$(\Delta\mathbf{p})^* = \arg\min_{\Delta\mathbf{p}} \sum_{\mathbf{x}\in\mathbb{N}} \left\| \frac{\partial \mathcal{I}_{t+1}(\mathbf{x}')}{\partial \mathbf{x}'^T}\Delta\mathbf{p} + \mathcal{I}_{t+1}(\mathbf{x}') - \mathcal{I}_t(\mathbf{x}) \right\|_2^2$$

The minimization equation can then be written in the form of:

$$\arg\min_{\Delta\mathbf{p}} \|\mathbf{A}\Delta\mathbf{p} - \mathbf{b}\|_2^2$$

where:

$$\mathbf{A} = \begin{bmatrix} \mathcal{I}_x(\mathbf{x}_1+\mathbf{p}) & \mathcal{I}_y(\mathbf{x}_1+\mathbf{p}) \\ \mathcal{I}_x(\mathbf{x}_2+\mathbf{p}) & \mathcal{I}_y(\mathbf{x}_2+\mathbf{p}) \\ \vdots & \vdots \\ \mathcal{I}_x(\mathbf{x}_D+\mathbf{p}) & \mathcal{I}_y(\mathbf{x}_D+\mathbf{p}) \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} \mathcal{I}_t(\mathbf{x}_1) - \mathcal{I}_{t+1}(\mathbf{x}_1+\mathbf{p}) \\ \mathcal{I}_t(\mathbf{x}_2) - \mathcal{I}_{t+1}(\mathbf{x}_2+\mathbf{p}) \\ \vdots \\ \mathcal{I}_t(\mathbf{x}_D) - \mathcal{I}_{t+1}(\mathbf{x}_D+\mathbf{p}) \end{bmatrix}$$

c. What conditions must $\mathbf{A}^T\mathbf{A}$ meet so that a unique solution to $\Delta\mathbf{p}$ can be found?

A solution to the minimization problem described in part b can be found using the left-pseudo inverse of A, that is:

$$\Delta\mathbf{p} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$$

which has a unique solution such that **$\mathbf{A}^T\mathbf{A}$ is invertible (and not ill-conditioned)**. That is, the eigenvalues of $\mathbf{A}^T\mathbf{A}$ are not at or near 0. Alternatively, this is equivalent to saying $\mathbf{A}$ must have full column rank.

Q1.3

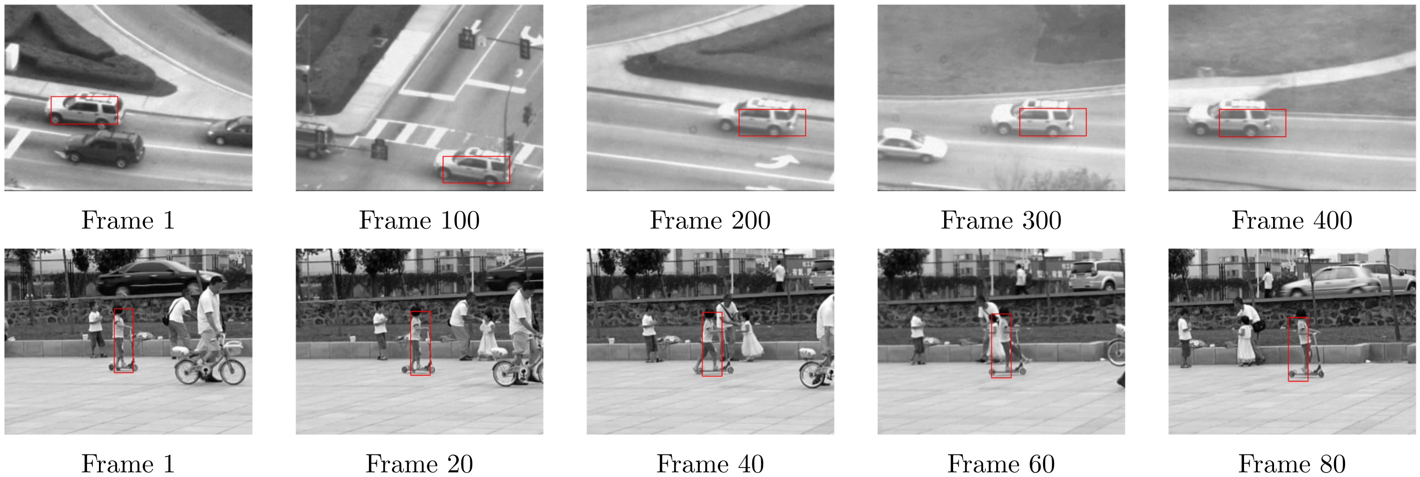

Figure 1: Tracking outputs for `carseq.npy` (top) and `girlseq.npy` (bottom)

Q1.4

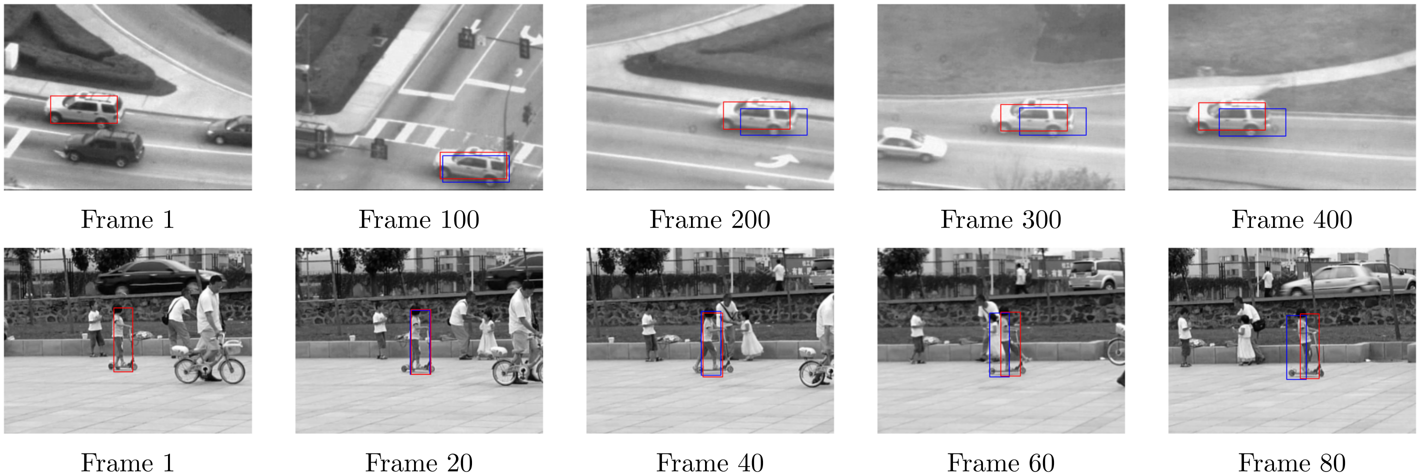

Figure 2: Comparisons of tracking outputs for `carseq.npy` (top) and `girlseq.npy` (bottom). Blue comes from baseline (Q1.3) tracker and Red comes from tracker with template correction (Q1.4).

As can be seen via visual inspection of Figure 2 above, the template drift observed in the baseline tracker (blue) is largely not present in the tracker with template correction (red) for both image sequences.

2 Affine Motion Subtraction

2.1 Dominant Motion Estimation

Q2.1

The following shows how the value which was used for $\frac{\partial \mathcal{W}(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}^T}$ in `LucasKanadeAffine.py` was determined.

$$\text{let: } \mathcal{I}_{t+1}(\mathbf{x}' + \Delta\mathbf{p}) \approx \mathcal{I}_{t+1}(\mathbf{x}') + \frac{\partial \mathcal{I}_{t+1}(\mathbf{x}')}{\partial \mathbf{x}'^T} \frac{\partial \mathcal{W}(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}^T} \Delta\mathbf{p}$$

$$\text{where } \mathbf{p} = \begin{bmatrix} p_x \\ p_y \end{bmatrix}, \; \frac{\partial \mathcal{I}_{t+1}(\mathbf{x}')}{\partial \mathbf{x}'^T} = \begin{bmatrix} \mathcal{I}_{t+1_x}(\mathbf{x}') \\ \mathcal{I}_{t+1_y}(\mathbf{x}') \end{bmatrix}, \; \mathbf{x}' = \mathcal{W}(\mathbf{x};\mathbf{p}), \; \tilde{\mathbf{x}}' = \mathbf{M}\tilde{\mathbf{x}}, \; \mathbf{M} = \begin{bmatrix} 1+p_1 & p_2 & p_3 \\ p_4 & 1+p_5 & p_6 \\ 0 & 0 & 1 \end{bmatrix}, \; \tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$$

$$\therefore \mathcal{W}(\mathbf{x};\mathbf{p}) = \begin{bmatrix} 1+p_1 & p_2 \\ p_4 & 1+p_5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} p_3 \\ p_6 \end{bmatrix} = \begin{bmatrix} x + xp_1 + yp_2 + p_3 \\ xp_4 + y + yp_5 + p_6 \end{bmatrix}$$

$$\therefore \frac{\partial \mathcal{W}(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}^T} = \begin{bmatrix} \frac{\partial \mathcal{W}_x}{\partial \mathbf{p}_1} & \frac{\partial \mathcal{W}_x}{\partial \mathbf{p}_2} & \frac{\partial \mathcal{W}_x}{\partial \mathbf{p}_3} & \frac{\partial \mathcal{W}_x}{\partial \mathbf{p}_4} & \frac{\partial \mathcal{W}_x}{\partial \mathbf{p}_5} & \frac{\partial \mathcal{W}_x}{\partial \mathbf{p}_6} \\ \frac{\partial \mathcal{W}_y}{\partial \mathbf{p}_1} & \frac{\partial \mathcal{W}_y}{\partial \mathbf{p}_2} & \frac{\partial \mathcal{W}_y}{\partial \mathbf{p}_3} & \frac{\partial \mathcal{W}_y}{\partial \mathbf{p}_4} & \frac{\partial \mathcal{W}_y}{\partial \mathbf{p}_5} & \frac{\partial \mathcal{W}_y}{\partial \mathbf{p}_6} \end{bmatrix} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & y & 1 \end{bmatrix}$$

2.2 Moving Object Detection

Q2.3

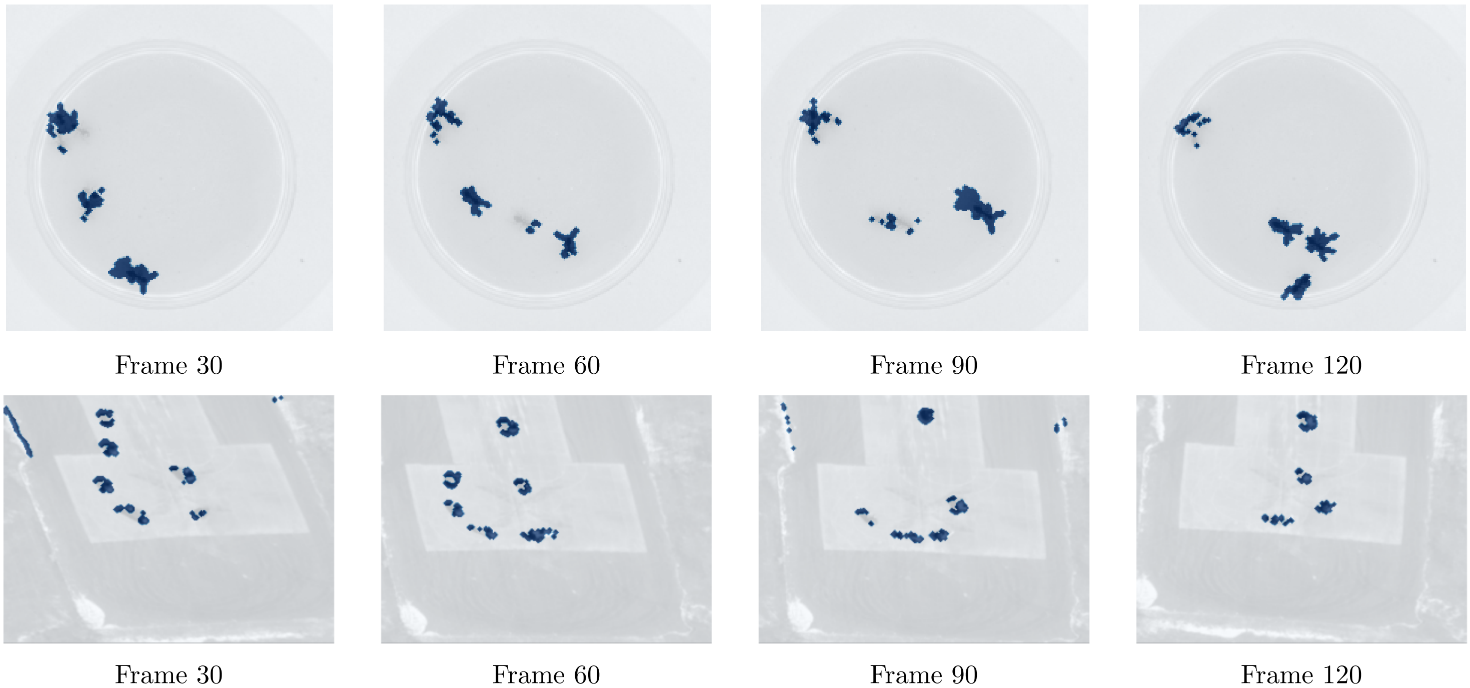

Figure 3: Comparisons of tracking outputs for `antseq.npy` (top) and `aerialseq.npy` (bottom). Masks are overlaid in blue over each of the requested frames.

As can be seen via visual inspection of Figure 3 above, the moving objects were detected correctly and, for the most part, exclusively, barring some hard edges in the terrain of frames 30 and 90 in `aerialseq.npy`.

3 Efficient Tracking

3.1 Inverse Composition

Q3.1

The vectorized Inverse Compositional Lucas-Kanade algorithm, using the provided linearization, is effectively equivalent to computing:

$$\arg\min_{\Delta\mathbf{p}} \|\mathbf{A}'\Delta\mathbf{p} - \mathbf{b}'\|_2^2$$

$$\text{where: } \mathbf{A}' = \frac{\partial\mathcal{I}_{t+1}(\mathbf{x}')}{\partial\mathbf{x}'^T}\frac{\partial\mathcal{W}(\mathbf{x};\mathbf{0})}{\partial\mathbf{p}^T},\ \mathbf{b}' = \mathcal{I}_{t+1}(\mathcal{W}(\mathbf{x};\mathbf{p}) - \mathcal{I}_t(\mathbf{x})$$

Thus, this is approach is considerably more computationally efficient since in this case $\mathbf{A}'$ is quite clearly not a function of $\mathbf{p}$ and, as such, can be removed from the iterative computation and precomputed, meaning only $\mathbf{b}'$ (effectively the error image) has to be computed on each iteration (in addition, of course, to the update rule.

Note substituting `InverseCompositionAffine.py` for `LucasKanadeAffine.py` in SubtractDominantMotion.py confirms this predication by returning equivalent results for testAerialSequence.py and testAntSequence.py in 1/16th the time. (Note, this speed increase is partly due to the fact that `LucasKanadeAffine.py` uses a 1-level deep for loop to compute the steepest descent images whereas `InverseCompositionAffine.py` uses a 3D np.einsum implementation, which if I had more time I would revise `LucasKanadeAffine.py` to use. Nonetheless, it stands to reason that a performance increase should be observed due to an algorithm change alone due to the above reasoning.)