CS 83: Computer Vision

Video Tracking

Amittai Siavava

03/03/2024

Abstract

This project implements Lucas Kanade, Lucas Kanade Affine, and Inverse Compositional algorithms for tracking optical flow, helping keep track of objects in a video.

Credit Statement

I discussed ideas with **Ivy (Aiwei) Zhang** and **Angelic McPherson**. However, the code and writeup are entirely my own, with reference to class notes.

Contents

1. Theory Questions

Problem 1.

Assuming the affine warp model $\mathbf{W} = \begin{bmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \\ 0 & 0 & 1 \end{bmatrix}$, derive the expression for the Jacobian matrix $\mathbf{J}$ in terms of the warp parameters $\mathbf{p} = [p_1, p_2, p_3, p_4, p_5, p_6]^\top$.

The Jacobian matrix $\mathbf{J}$ is the matrix of partial derivatives of the warp $\mathbf{W}$ with respect to the warp parameters $\mathbf{p}$. We have

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

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \mathbf{W}}{\partial p_1} & \frac{\partial \mathbf{W}}{\partial p_2} & \frac{\partial \mathbf{W}}{\partial p_3} & \frac{\partial \mathbf{W}}{\partial p_4} & \frac{\partial \mathbf{W}}{\partial p_5} & \frac{\partial \mathbf{W}}{\partial p_6} \end{bmatrix}$$

We can compute the partial derivatives of $\mathbf{W}$ with respect to the warp parameters $\mathbf{p}$ as follows:

$$\frac{\partial \mathbf{W}}{\partial p_1} = \begin{bmatrix} x \\ 0 \end{bmatrix}, \quad \frac{\partial \mathbf{W}}{\partial p_2} = \begin{bmatrix} 0 \\ x \end{bmatrix}, \quad \frac{\partial \mathbf{W}}{\partial p_3} = \begin{bmatrix} y \\ 0 \end{bmatrix},$$
$$\frac{\partial \mathbf{W}}{\partial p_4} = \begin{bmatrix} 0 \\ y \end{bmatrix}, \quad \frac{\partial \mathbf{W}}{\partial p_5} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \frac{\partial \mathbf{W}}{\partial p_6} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Therefore, the Jacobian matrix $\mathbf{J}$ is:

$$\mathbf{J} = \begin{bmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{bmatrix}$$

Note: Since the last row of $\mathbf{W}$ is always $[0, 0, 1]$ and therefore leaves the x-coordinate (in homogeneous coordinates) unchanged, we do not need to include the partial derivatives of the last row of $\mathbf{W}$ with respect to the warp parameters $\mathbf{p}$ since the last row would be all zeros.

Problem 2.

Find the computational complexity (Big O notation) for the initialization step (pre-computing $\mathbf{J}$ and $\mathbf{H}^{-1}$) and for each runtime iteration (Equation 13) of the Matthews-Baker method:

$$\Delta\mathbf{p}^* = \mathbf{H}^{-1}\mathbf{J}^\top \left[\mathbf{I}(\mathbf{W}(\mathbf{x};\mathbf{p})) - \mathbf{T}\right]$$

Express your answers in terms of n, m, and p, where

(i) n is the number of pixels in the template $\mathbf{T}$,

(ii) m is the number of pixels in an input image $\mathbf{I}$, and

(iii) p is the number of parameters used to describe the warp $\mathbf{W}$.

How does this compare to the runtime of the regular Lucas-Kanade method?

1. **Initialization Step**:
   (a) **Pre-computing $\mathbf{J}$**: The computational complexity of pre-computing $\mathbf{J}$ is $O(np)$ since we need to compute the partial derivatives of the warp $\mathbf{W}$ with respect to each of the warp parameters $\mathbf{p}$, for each of the n pixels in the image.
   (b) **Pre-computing $\mathbf{H}^{-1}$**: The computational complexity of pre-computing $\mathbf{H}^{-1}$ is $O(p^3)$ since we need to compute the inverse of the Hessian matrix $\mathbf{H}$.

3. Dense Reconstruction

In applications such as 3D modelling, 3D printing, and AR/VR, a sparse model is not enough. When users are viewing the reconstruction, it is much more pleasing to deal with a dense reconstruction. To do this, it is helpful to rectify the images to make matching easier.

3.1. **Image Rectification.**

Initially, I would get an awkwardly-cropped image when I tried to rectify the images. I discovered that setting the M scaling factor passed to `eight_point` to 1 fixes the issue. The sparse projections looks unaffected, but I saw other students mention on Slack that they ran into the same issue.

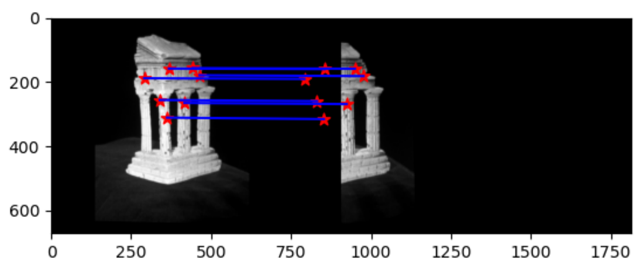


Figure 1. Dense Reconstruction, $M = \mathbf{max}(I_x, I_y)$

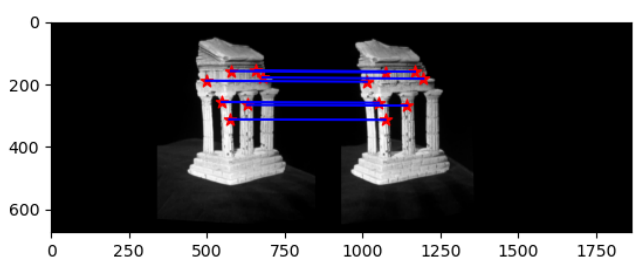


Figure 2. Dense Reconstruction, $M = 1$

3.2. **Disparity Map and Depth Map.** Computed disparity and depth maps

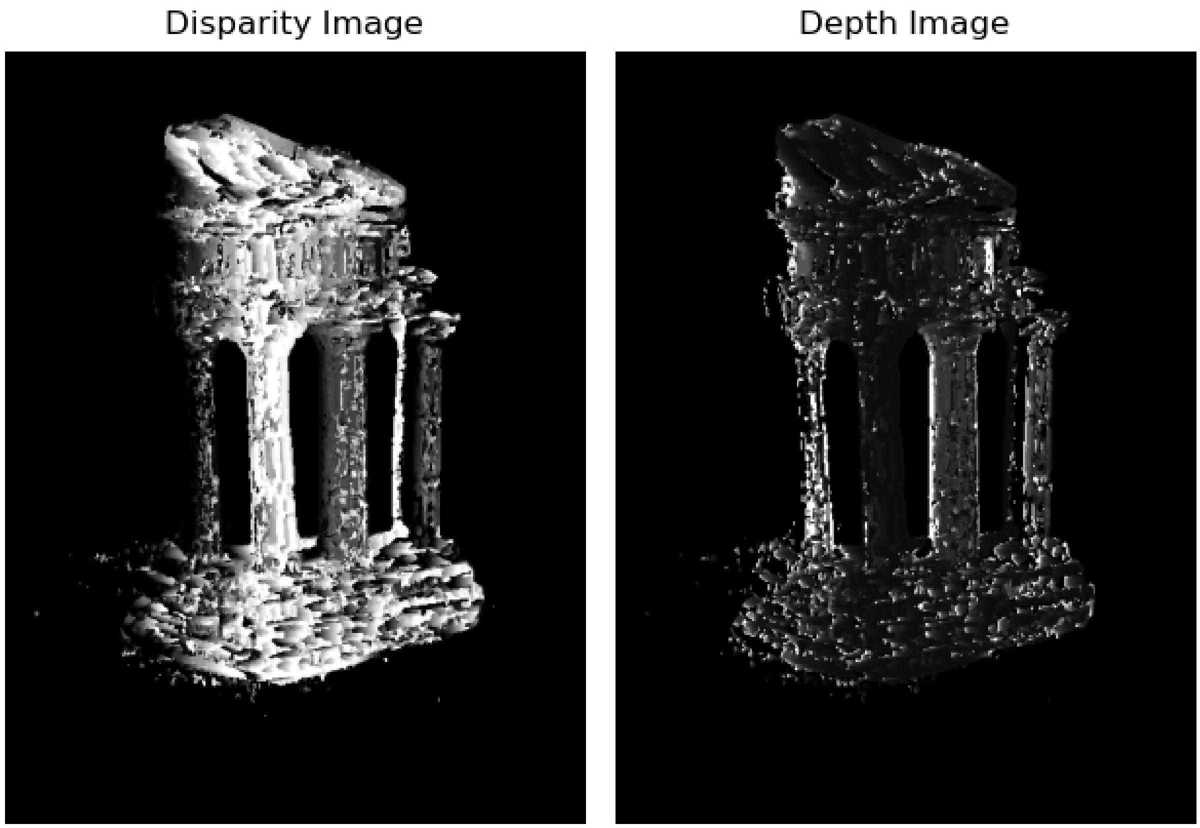


Figure 3. Disparity and Depth Maps

4. Pose Estimation

4.1. Camera Matrix Estimation. Pose estimation errors

Estimated camera matrix

$$P = \begin{bmatrix} 1.19121690 \times 10^{-1} & 5.45061053 \times 10^{-1} & 1.90356266 \times 10^{-1} & -9.32574074 \times 10^{-2} \\ -3.72738881 \times 10^{-1} & 1.96903454 \times 10^{-1} & -4.69365814 \times 10^{-1} & 4.95756458 \times 10^{-1} \\ -2.82243252 \times 10^{-4} & -1.29896978 \times 10^{-4} & 6.42031422 \times 10^{-5} & 1.39340650 \times 10^{-3} \end{bmatrix}$$

Metric	**Value**
Reprojection Error with clean 2D points	$1.510190084711698 \times 10^{-10}$
Pose Error with clean 2D points	$6.782416819715269 \times 10^{-12}$
Reprojection Error with noisy 2D points	5.056479446706304
Pose Error with noisy 2D points	1.1255732262731768

Table 1. Pose Estimation Errors (see Figure 6)

4.2. Intrinsic/Extrinsic Parameters Estimation.

$$K = \begin{bmatrix} -3.91311040 \times 10^{-1} & 2.16322603 \times 10^{-2} & -5.05036684 \times 10^{-1} \\ 0.00000000 \times 10^{0} & -5.79132614 \times 10^{-1} & -3.84387155 \times 10^{-2} \\ 0.00000000 \times 10^{0} & 0.00000000 \times 10^{0} & 4.38131343 \times 10^{-4} \end{bmatrix}$$

$$R = \begin{bmatrix} -3.04416892 \times 10^{-1} & -9.52538789 \times 10^{-1} & 4.58882152 \times 10^{-4} \\ 9.52538627 \times 10^{-1} & -3.04417134 \times 10^{-1} & -6.10562418 \times 10^{-4} \\ 7.21275976 \times 10^{-4} & 2.51237461 \times 10^{-4} & 9.99999708 \times 10^{-1} \end{bmatrix}$$

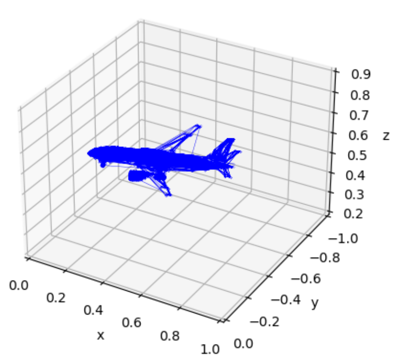
$$t = \begin{bmatrix} 1.38022671// - 4.14750766//2.38863304 \end{bmatrix}$$

Metric	**Value**
Intrinsic Error with clean 2D points	141.45256375245376
Rotation Error with clean 2D points	1.8653823929245028
Translation Error with clean 2D points	3.0626221861668257
Intrinsic Error with noisy 2D points	141.45251343470733
Rotation Error with noisy 2D points	1.8735506641560857
Translation Error with noisy 2D points	4.420673631875621

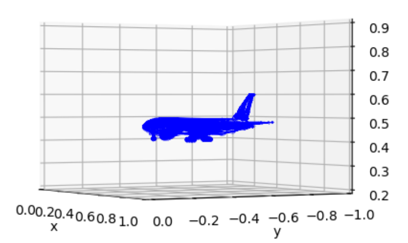
Table 2. Intrinsics/Extrinsics Estimation Errors (see Figure 7)

4.3. **CAD Alignment and Projection.** An aeroplane CAD model aligned onto an image using calculated depth information.

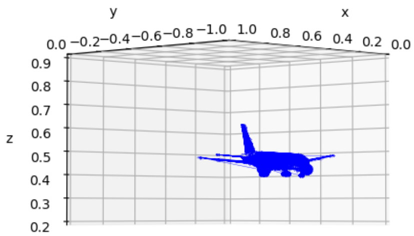
Points Projected onto Image



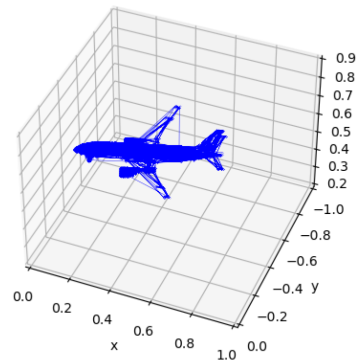
Warped CAD (no rotation)



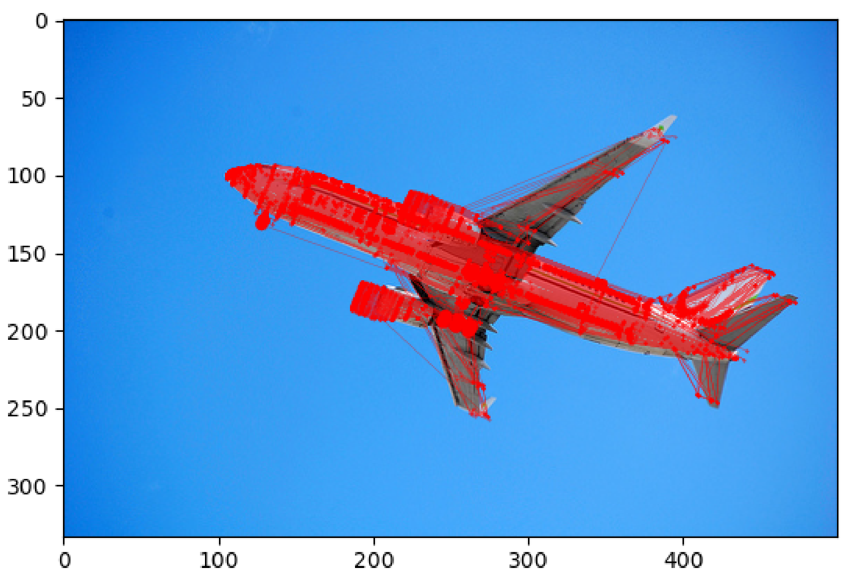
Perspective 2



Perspective 3



Perspective 4



Warped CAD

Appendix A. Extra Figures

```
⊗ λ> ./submission.py
  Optimization terminated successfully.
           Current function value: 0.000106
           Iterations: 74
           Function evaluations: 7275
  F = array([[-1.12822743e-09,  1.23273153e-07, -6.24786160e-06],
         [ 6.41080466e-08,  1.04807659e-10, -1.11138892e-03],
         [-1.31615524e-05,  1.06851400e-03,  4.47839257e-03]])
```

Figure 4. Raw Fundamental Matrix

```
● λ> ./test_temple_coords.py
  Optimization terminated successfully.
           Current function value: 44.556690
           Iterations: 37
           Function evaluations: 4050
  ESSENTIAL MATRIX: [[-4.90914507e-01  3.96179658e+00  2.19734674e+00]
   [ 2.12129669e+01 -1.48056692e-01 -9.42668155e+01]
   [ 6.10057409e-02  9.54224292e+01  1.09934587e-01]]
```

Figure 5. Raw Essential Matrix

```
● λ> ./test_pose.py
  Reprojection Error with clean 2D points: 1.510190084711698e-10
  Pose Error with clean 2D points: 6.782416819715269e-12
  Reprojection Error with noisy 2D points: 5.056479446706304
  Pose Error with noisy 2D points: 1.1255732262731768
○ λ>
```

Figure 6. Pose Estimation Errors

```
● λ> ./test_params.py
  Intrinsic Error with clean 2D points: 141.45256375245376
  Rotation Error with clean 2D points: 1.8653823929245028
  Translation Error with clean 2D points: 3.0626221861668257
  Intrinsic Error with noisy 2D points: 141.45251343470733
  Rotation Error with noisy 2D points: 1.8735506641560857
  Translation Error with noisy 2D points: 4.420673631875621
○ λ> _
```

Figure 7. Intrinsics/Extrinsics Estimation Errors