"[ENPM 673] Project-4"

Perception for Autonomous Robots

PROJECT REPORT

Nevil Patel-116897068

Shesh Mali-116831055

Akshay Kurhade-116914529

Objective-

To implement Lucas-Kanade(LK) template tracker and evaluate the same on three provided video sequences.

(a) Bolt

(b) Car

(c) DragonBaby

Figure 1: Tracking sequences

Lucas-Kanade Template Tracker-

The Lucas-Kanade optical flow algorithm is a simple technique which can provide an estimate of the movement of interesting features in successive images of a scene. One of the assumptions of Lucas Kanade is the two images are separated by a small-time increment Δt, in such a way that objects have not displaced significantly. We start by converting the image into a grayscale image to make it easier to detect edges and features of the image

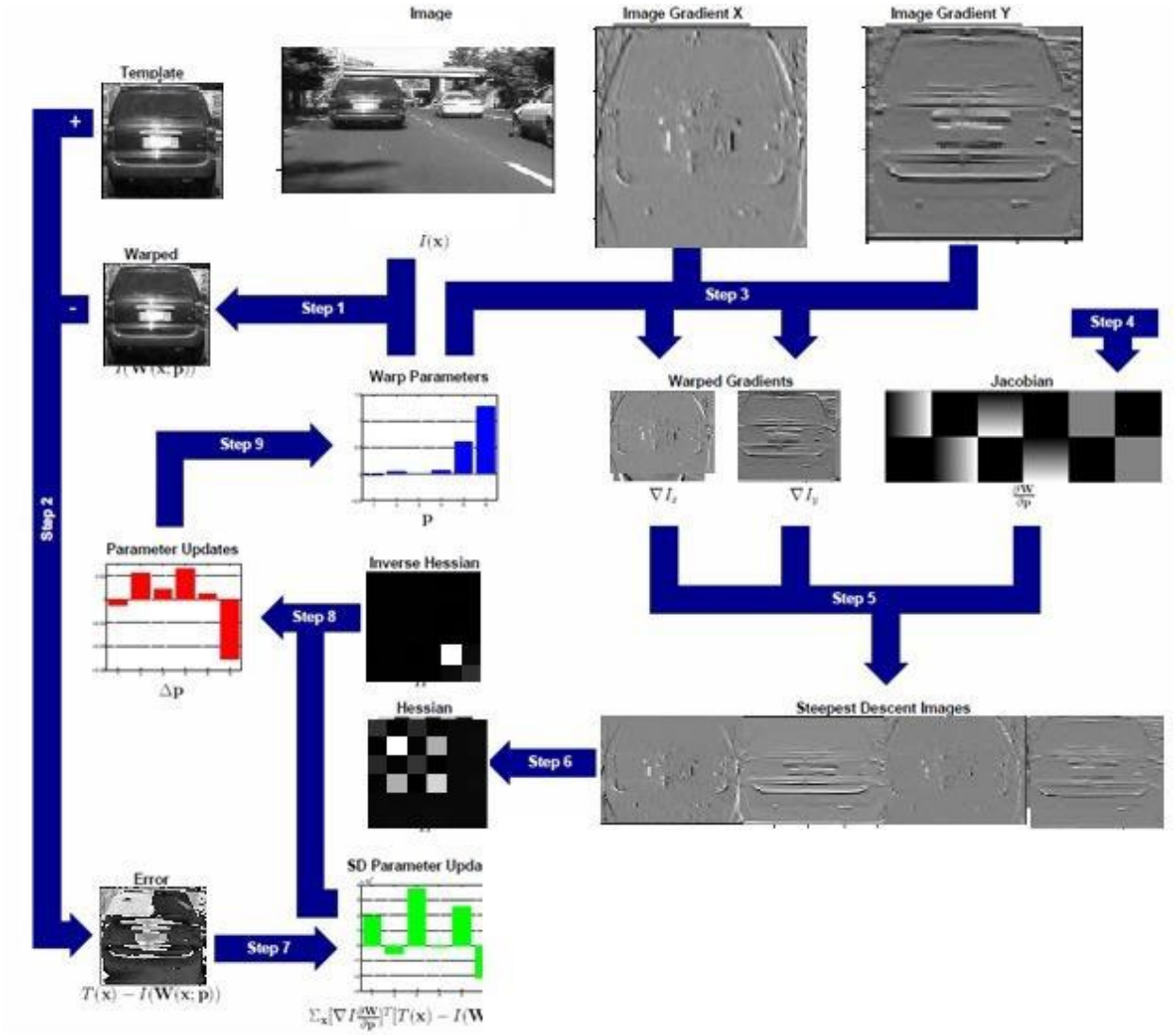


Fig. Lucas-Kanade template tracker pipeline

The goal of the Lucas-Kanade algorithm is to align a 2D template $T(u)$ to a 2D input image $I(u)$, where $u = (u, v)^T$ is a 2D column vector containing the pixel coordinates. Let $w(u; p)$ denote the parameterized set of allowed warps, where $p = (p_1, \ldots., p_n)^T$ is a vector of parameters. The warp $W(u; p)$ takes the pixel u in the template T and maps it to the sub-pixel location $W(u; p)$ in the image I.

Methodology-

Consider an image $I(x, y)$. For a very small displacement, the new image can be represented by,

$$H(x, y) = I(x + u, y + v)$$

Here,

u, v= displacement of the pixel

To obtain (u, v) we will solve,

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

The summation is of all pixels in a KxK window.

This equation is solvable only when –

1.) It is invertible.

2.) Eigen values should not be too small.

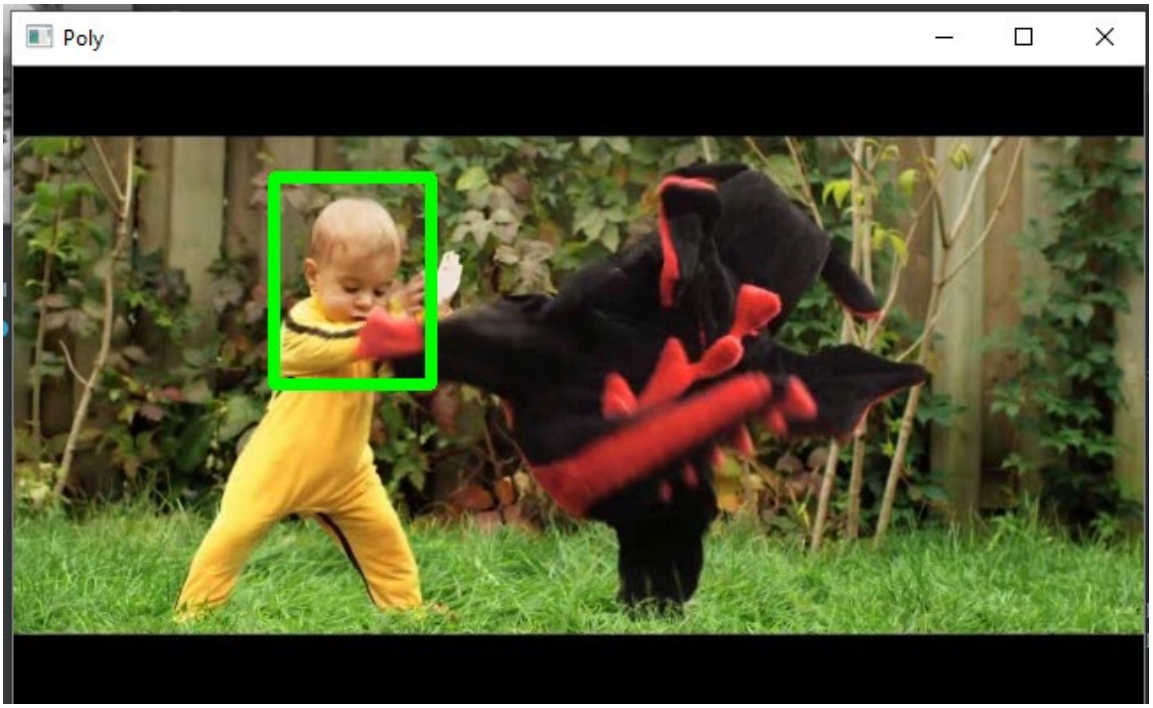
3.) It should be well conditioned i.e. eigen values should not be too large.

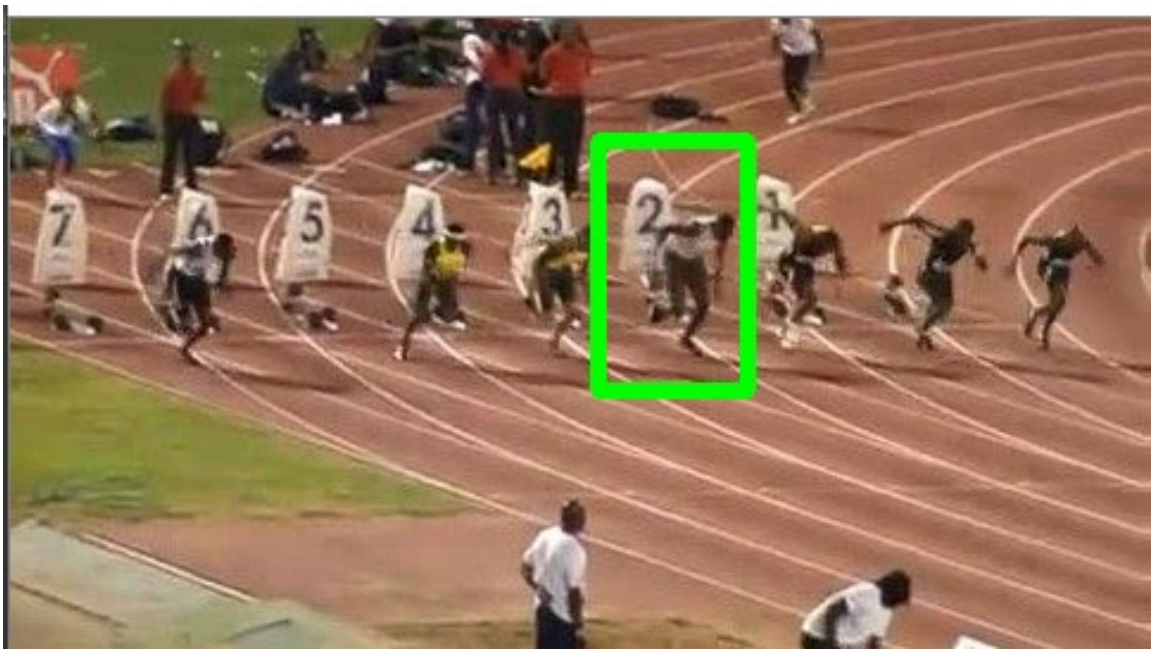
Implementation-

Preprocessing-

1) Convert each frame to grayscale
2) Manually define template ROI

```
rect = np.array([[250, 50], [310, 50], [310, 150], [250, 150]])  # bolt
rect = np.array([[123, 104], [336, 104], [336, 275], [123, 275]]) #Car
rect = np.array([[140, 60], [220, 60], [220, 200], [140, 200]])  # baby
```





Lucas-Kanade tracker implementation-

Step1: Warp I with $W(x;p)$ to compute $I(W(x;p))$

Step 2: Computing the error images $T(x) - I(W(x;p))$

Step 3: Warping the gradient ∇I with $W(x;p)$

Step 4: Evaluate the Jacobian $\partial W/\partial p \; at \; (x;p)$

Step 5: Compute the steepest descent images $\nabla I \; \partial W/\partial p$

Step 6: Compute hessian matrix

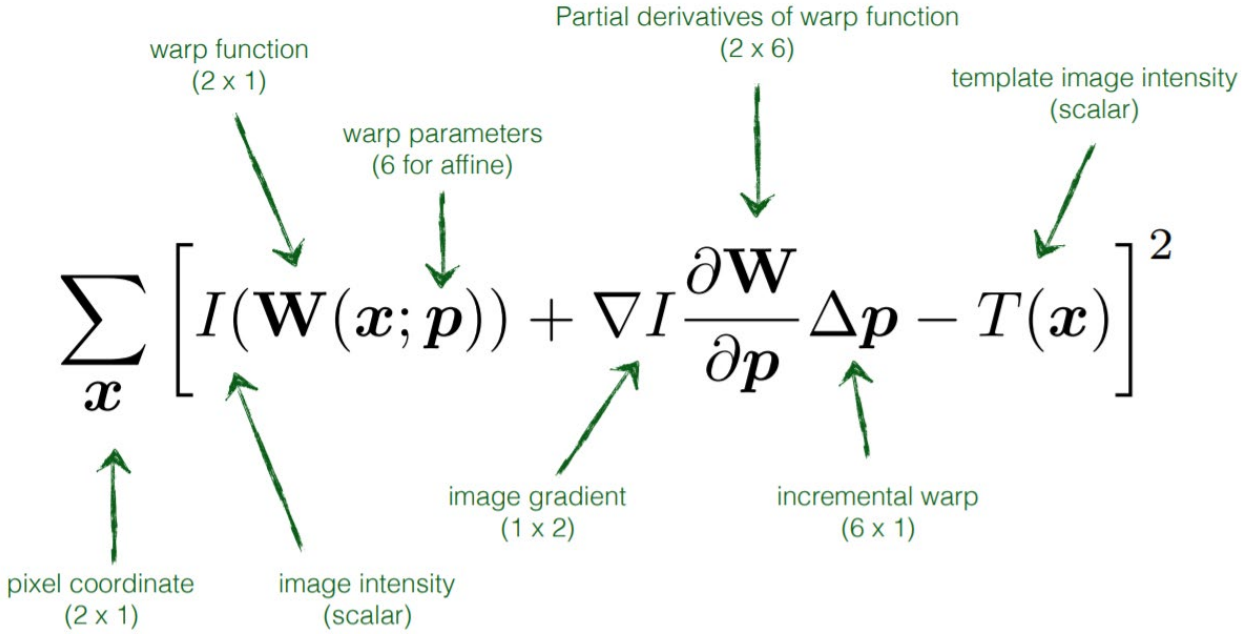
Hessian matrix = H = $\Sigma_x \left[\nabla I \;\; \frac{\partial W}{\partial p}\right]^T \left[\nabla I \frac{\partial W}{\partial p}\right]$

Step 7: Compute $\Sigma_x \left[\nabla I \;\; \frac{\partial W}{\partial p}\right]^T [T(x) - I(W(x;p))]$

Step 8: Compute Δp using the equation below: $\Delta p \; = \; H^{-1}\Sigma_x \left[\nabla I \;\; \frac{\partial W}{\partial p}\right]^T [T(x) - I(W(x;p))]$

Step 9: Update parameters $p \; \rightarrow \; p \; + \; \Delta p$

Where,



Implementation issues and approached solution-

Template:

The template should represent the entire dataset over which the tracking has to be done. Thus, resistance to illumination or brightness can be encoded in the template itself. The template should contain enough features so that the image gradients can be matched with the template. Also, the change between template and the object frames must be incremental since we do Taylor series approximation to make the problem linear and minimize the error. There is a necessity that the warp function used must be continuous since we calculate the Jacobians. Affine transformation is one such warp that satisfies the conditions of Lucas Kanade algorithm.

When there is noise, the gradients are disrupted in all the directions. Since the LK algorithm primarily works based on Jacobians and image derivatives, noise reduction in necessary component in pipeline. The basic noise reduction technique is gaussian blur and it is faster. This will smoothen the edges which is not a problem since blur gives us thicker and smoother gradients which are the features on which LK algorithm works.

Also, to make the detection more robust multiple templates are defined for template correction in an event where there is a sudden change in gradient, illumination change.

Countering Intensity Variation:

This is a major issue when we have a fixed template for the entire tracking dataset and the dataset has good variation in brightness thus affecting intensity. This is a major concern since we work on intensity space in LK algorithm. To counter this effect one solution is to do brightness correction such as histogram equalization or gamma correction. Histogram equalization works better in cases where there are no sudden changes in illumination or gradient while gamma correction was found to be better suitable in opposite cases.

Affine transform vs Inverse affine for LK:

Initially when we developed LK algorithm, we use normal affine transform. This approach was not converging, and the error was constantly increasing irrespective of learning rate. Also, the standard affine transformation that comes with open CV cannot be applied since there few cases which need to be handled explicitly. So, we used inverse affine transform from template to image and use it instead. This allowed is to use the standard affine transformation of open CV package. Thus, the LK algorithm converges to the required tolerance and the algorithm runs much faster.

Blur Correction:

The algorithm still converges, and acceptable tracking performance is achieved for Car data base even without noise reduction. But in case of vase and human data set, the performance is not as expected. In our case gaussian blur with kernel size of 3x3 is giving good enough results for the purpose of tracking. Even with car dataset, a tighter boundary is obtained over the entire data set after gaussian blur. One more advantage of choosing gaussian kernel is that it is separable, and kernel can be applied much greater speed than averaging or median filter. Median filter is not needed since we are not using any edge features for tracing.

Sobel filter:

To find the image gradients we make use of Sobel filter if size 3x3 in x and y direction separately. We take gradients on the full-size image and then warp to required rectangle. This makes sure that we do not lose the edge data rather than warping the image and taking gradients.

Output Video Link-

[Google Drive Link]

References-

1. Baker, S., & Matthews, I. (2004). Lucas-Kanade 20 Years On: A Unifying Framework. International Journal of Computer Vision, 56(3), 221-255. doi:10.1023/b:visi.0000011205.11775.fd https://www.ri.cmu.edu/pub_files/pub3/baker_simon_2002_3/baker_simon_2002_3.pdf
2. https://www.ri.cmu.edu/project/lucas-kanade-20-years-on/
3. https://github.com/ziliHarvey/Lucas-Kanade-tracking-and-Correlation-Filters