ENPM 673: Perception for Autonomous Robots **Released: April-7.**

Report

Name: Arpit Aggarwal and Shantam Bajpai

1 Optical Flow Theory

Optical Flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of object or the camera. Optical Flow works on two assumptions:

1. The pixel intensities of the object doesn't change between consecutive frames.
2. Neighbouring pixels have similar motion.

That is, if at time 't', the pixel intensity is defined as I(x, y, t) and at time 't + dt', the pixel intensity is defined as I(x + dx, y + dy, t + dt), the following equation holds:

$$I(x, y, t) = I(x + dx, y + dy, t + dt)$$

Using Taylor series, we obtain the the Optical Flow Equation defined as:

$$f_x * u + f_y * v + f_t = 0$$

where,

$f_x = \frac{\partial f}{\partial x}; f_y = \frac{\partial f}{\partial y}$
$u = \frac{dx}{dt}; v = \frac{dy}{dt}$

The Lucas Kanade method is used to find the solution to the (u, v) of the Optical Flow equation using method of Least squares fit.

2 Lucas Kanade Method

Optical flow is defined as the apparent motion of objects and surfaces. Mathematically this can be shown as $I(x, y, t) \rightarrow I(x, y, t + 1)$. The image is a function of the spatial coordinates (x,y) and time t.
Now in Optical flow the conventional method of finding the flow field from one pixel to another suffers from a major flaw called the Aperture problem. If we are looking locally one cannot tell the

change in direction of the pixel and this is referred to as the aperture problem. In Lucas-Kanade the basic idea is to impose local constraints to get more equations for a pixel. Lucas Kanade states that the optical flow is essentially constant in a local neighborhood of the pixel under consideration and solves the basic optical flow equations for all the pixels in the neighborhood by the least squares criterion.
Below is the algorithm we followed to implement our Lucas Kanade Algorithm:

1. Warp the image I to obtain $I(W[x, y]; P)$

2. Compute the error image $T(x) - I(W[x, y]; P)$

3. Warp the Gradient ∇I with $(W[x, y]; P)$

4. Evaluate $\frac{\partial W}{\partial P}$ at $([x, y]; P)$

5. Compute Steepest descent images $\nabla I \frac{\partial W}{\partial P}$

6. Compute the Hessian Matrix $\sum (\nabla I \frac{\partial W}{\partial P})^T \nabla I \frac{\partial W}{\partial P}$

7. Compute $\sum (\nabla I \frac{\partial W}{\partial P})(T(x) - I(W[x, y]; P))$

8. Compute ΔP

$P \leftarrow P + \Delta P$

9. Keep Iterating till the magnitude of ΔP is negligible

10. And also $\Delta P = (A^T A)^{-1} A^T b$

Here $A^T A$ is the Hessian Matrix that has been computed above and A is the steepest descent image and b is basically the error image that has been computed in step 2.
In addition to this we have used the affine warp function that is given as

$$\begin{bmatrix} 1 + p_1 & p_3 & p_5 \\ p_2 & 1 + p_4 & p_6 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

3 Tracker Evaluation

The tracker performs well for scenarios where there is less brightness change. In the case of car video, the tracker breaks down. We applied two updates to make this tracker perform better in such scenarios:
1. Normalized the pixel values. That is, we multiplied the pixel values in the current frame by a scale value which is the ratio of the mean of pixels in template frame and the mean of pixels in the current frame.
2. Apply weighted Least swaures method, known as Huber loss. We multiplied the error term with

a weight matrix which helped in rejecting outliers that were causing the tracker to breakdown.

4 Dragon Baby Video

For the Dragon Baby the following was the template chosen. The ROI is shown in the red bounding box which is used for the Lucas-Kanade Algorithm. The ROI top-left coordinate was (160, 83) and ROI bottom right coordinate was (216, 148).

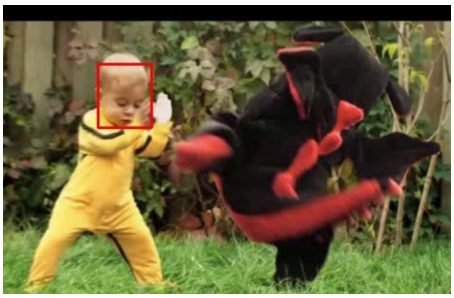

Figure 1: Template for Dragon Baby

After running the algorithm the following was the output obtained:
https://drive.google.com/file/d/16d9Itfe-IBtyOXCfF3sgfJLdObpc-xE8/view?usp=sharing

5 Bolt Video

For the Bolt the following was the template chosen. The ROI is shown in the red bounding box which is used for the Lucas-Kanade Algorithm. The ROI top-left coordinate was (269, 75) and ROI bottom right coordinate was (303, 139).

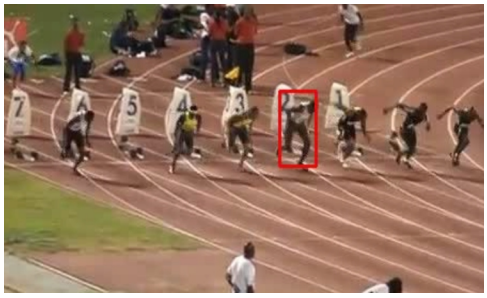

Figure 2: Template for Bolt

After running the algorithm the following was the output obtained:

https://drive.google.com/file/d/1_dlchPk_n9N20Q0mazmpFR3MVPn_w2cT/view?usp=sharing

6 Car Video

For the Car the following was the template chosen. The ROI is shown in the red bounding box which is used for the Lucas-Kanade Algorithm. The ROI top-left coordinate was (70, 51) and ROI bottom right coordinate was (177, 138).

Figure 3: Template for Car

After running the algorithm the following was the output obtained:
https://drive.google.com/file/d/1BnE9x5Ij-PFnYH0KSmrkHsKBG8GxAu6A/view?usp=sharing

7 References

1. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_video/py_lucas_kanade/py_lucas_kanade.html
2. https://www.ri.cmu.edu/pub_files/pub3/baker_simon_2004_1/baker_simon_2004_1.pdf
3. https://www.youtube.com/watch?v=tzO245uWQxA