ENPM 673
Lucas-Kanade tracker

Members- Mrinali Vyas
- Srujan Panuganti
- Akshita Pothamshetty

Instructions to run the code

Import the following python 3 libraries

1. Import cv2
2. Import numpy
3. Import glob

In order to run the code import the dataset from the drive link and run the file name.

The following codes will run the code for the car, vase, human datasets respectively.

Run the following code :

1. lucas_car
2. lucas_human
3. Lucas_vase

The output video and dataset are included in the drive folder attached below.

The Google drive link is as given below :

https://drive.google.com/drive/folders/1pa1THqGfXMiw1xWdvLoGsnV0P4AnRXjz?usp=sharing

Introduction

In this project, we implement the Lucas-Kanade algorithm to track the template. We use this to track a human walking on the road, a box on a table and car on the road.

Step 1: Generating a template

To initialize the tracker we first define a template by drawing a bounding box around the object to be tracked in the first frame. Then for each following frames, the tracker will update an affine transform that warps the present frame is aligned with the warped present frame.

We initialize the coordinates of the bounding box which is to be tracked by cropping the region of interest from the first frame. This cropped region is used as the template image.

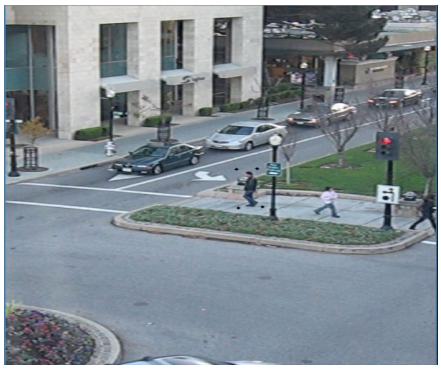

Fig: Selecting the points to crop the template

Fig : Cropped template image

Step 2: Evaluation of tracker

The goal of this algorithm is to align template image T(x) to an input image I(x). We warp our image to a set of vector parameters (p). The warp takes the pixel x in the coordinate frame of template T and maps to sub-pixel location in the coordinate frame of image I. When we do optical flow we consider set of affine parameters as the following:

$$\mathbf{W}(\mathbf{x};\mathbf{p}) = \begin{pmatrix} (1+p_1)\cdot x & + & p_3 \cdot y & + & p_5 \\ p_2 \cdot x & + & (1+p_4)\cdot y & + & p_6 \end{pmatrix} = \begin{pmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

The goal is to minimize the sum of squared error between the two images the template and image I warped back onto the coordinate frame of the template.

$$\sum_{\mathbf{x}} [\, I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x}) \,]^2 .$$

Fig : Warped Image

With respect to delta p and then the parameters are updated as (p + delta p). We iterate these two steps till 2 conditions are met either the norm is less than the threshold or the number of iterations.

The algorithm basically tries to evaluate the equation given below.

$$\sum_{\mathbf{x}} \left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) + \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta \mathbf{p} - T(\mathbf{x}) \right]^2 .$$

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

$$\sum_{\mathbf{x}} [\, I(\mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p})) - T(\mathbf{x}) \,]^2$$

Step 2 - After calculating the warp values of the template image we calculate the error image between the template and warped image.

$$T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))$$

Fig: Error Image

Step 3 - We now calculate the gradient of the warped image using a Sobel operator. The kernel size is 3.

$$\nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right)$$

Fig : Gradient x and Gradient y

Step 4 - We now calculate the Jacobian matrix in the following manner.

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{pmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{pmatrix}.$$

Step 5 - We now multiply the gradient and the jacobian matrix to give the Steepest Descent

$$\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$$

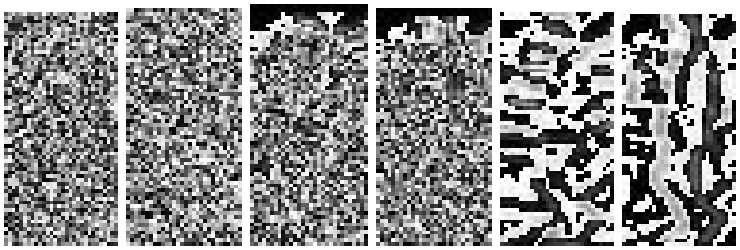

Fig: Steepest Descent images

Step 6 - Using the steepest descent we calculate the Hessian matrix.

$$H = \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]$$

Step 7 - We now compute the delta_p

$$\Delta \mathbf{p} = H^{-1} \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))]$$

Step 8 - We now update the values of P until one of the convergence conditions are met.

Step 9 - Results

Template Points for car- [120,90,340,280]

Template Points for human- [250,290,295,360]

Template Points for vase- [123,72,172,148]

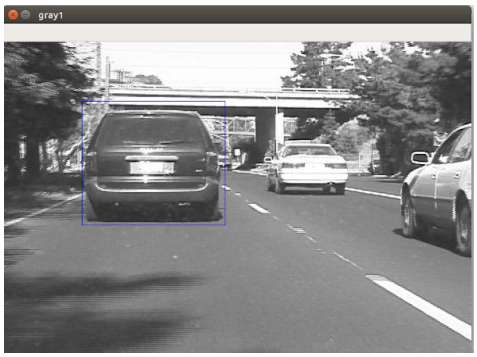

Fig : Ouput for car dataset

Fig: Output for Human dataset

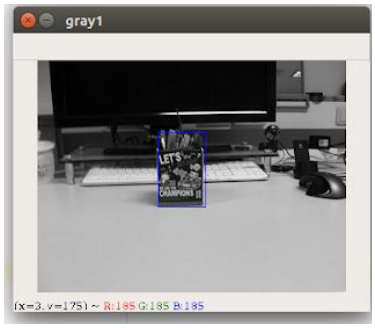

Fig : Ouput for vase dataset

Step 3: Problems

1. Eliminating the For loops : The first problem faced was because of multiple for loops in the while loop in the function for each iteration the speed of the code was very slow and heavily affected performance.
2. Lighting conditions under the bridge - In the car dataset as the lighting condition changes under the bridge are different than the ones in templalte frame the algorithm does not accurately track the car. In order to rectify this, we update the template image after a certain count.

Step 4: Reference

1. https://www.ri.cmu.edu/pub_files/pub3/baker_simon_2002_3/baker_simon_2002_3.pdf
2. https://www.ri.cmu.edu/pub_files/pub3/baker_simon_2002_3/baker_simon_2002_3.pdf