# CS5512 Assignment 5 Report

Ronit Rudra
A20379221

December 2, 2017

**Abstract**

This report describes the implementation of Optical Flow Tracking using Block Methods, in particular, the Lucas-Kanade Estimation. The desired outcomes, implementation details and performance as well as the challenges faced would be discussed in the upcoming sections.

## 1 Problem Statement

The objective was to:

1. Perform Lucas-Kanade Optical Flow Tracking on inputs either from camera or from a video file.
2. Visualize moving points and the reliability of tracking based on eigen-value ratios of the system matrix.

The Tracking done is sparse in nature, rather than computing vectors for each and every point in the image.

The features required by the program are:

- Would be run from the command prompt/terminal.
- Do processing based on what is passed as the command line arguments. If a filename is passed then the video file is used for processing, otherwise default to inputs from webcam.
- For each frame of the video, perform Lucas-Kanade Optical Flow Tracking.
- Draw the tracked vectors on the image.
- Modify intensity of drawn lines by computing reliability of tracking.
- Be able to pause or unpause the video by pressing 'p'.

## 2 Proposed Solution

The Solution relies on two processes:

1. Finding Velocity Projections
2. Determining Reliability of Tracking

The Optical Flow Constraint Equation is given by:

$$I(x(t+\delta t), y(t+\delta t), t+\delta t) = I(x(t), y(t), t) \tag{1}$$

The Lucas-Kanade Equation for estimating the velocities is given by:

$$\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum Iy^2 \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} -\sum I_x I_t \\ -\sum I_y I_t \end{bmatrix} \tag{2}$$

The above equation is obtained by finding the optimum of the following objective function:

$$E(v) = \sum_{(x,y) \in Patch} (\nabla I(x,y) \cdot v + I_t)^2 \tag{3}$$

The velocity vectors $v$ can be approximated by least squares estimation of equation(2) as it is of the form $Ax = b$ or can be directly computed by $A^{-1}b$. The $I_x$ and $I_y$ are the mean gradients of the two images which are one time step apart while $I_t$ is the intensity derivative w.r.t time in the two images.

The following section describes the implementation of the Optical Flow Algorithm.

## 3 Implementation Details

The algorithm is given below is for the sparse Optical flow where only feature points selected by the Shi-Tomasi Corner Detector are used as the center points for the block operation. This is different than dense Optical Flow where flow vectors for each and every pixel is calculated. A corner detector is useful as the window around the corner would have distinct gradients in the x and y direction which results in accurate velocity vectors in both the directions. Furthermore, the pyramidal Optical Flow was not implemented. This means that the program only tracks small movements unlike the Pyramidal Optical Flow version of the Lucas-Kanade Algorithm.

**Algorithm 1** Estimating Velocity

$I_1 \leftarrow$ Image at time $t$
$I_2 \leftarrow$ Image at time $t+1$
$P \leftarrow$ Set of good feature points in $I_1$
$D_x1, D_y1 \leftarrow$ Gradient of $I_1$
$D_x2, D_y2 \leftarrow$ Gradient of $I_2$
$I_t1, I_t2 \leftarrow I_1$ and $I_2$ convolved with smoothing filter
$I_x \leftarrow (D_x1 + D_x2)/2$
$I_y \leftarrow (D_y1 + D_y2)/2$
$I_t \leftarrow I_t2 - I_t1$
**for** $p \in P$ **do**
  $A \leftarrow$ System Matrix using $I_x$ and $I_y$
  $b \leftarrow$ Vector using $I_x$, $I_y$ and $I_t$
  $v \leftarrow$ Estimate using $A^{-1}b$
  $\tilde{p} \leftarrow p + v$ is the new location of $p$
**end for**

The velocity vectors can also be estimated using the least squares estimation as the system is over-determined. The new locations are simply the sum of the old locations and the velocity as the time step is one frame.

After the new locations of the points have been estimated, the reliability of tracking is estimated by taking the ratio of the smaller eigenvalue to the larger eigenvalue of $A$. The larger the ratio, the better is the tracking. This is done as follows for each System Matrix generated for a window:

**Algorithm 2** Estimating Reliability

$A \leftarrow$ System matrix of a window
$U, D, V^T gets$ Singular Value Decomposition of A
$R \leftarrow \frac{D_{min}}{D_{max}}$ is the Reliability factor

Since the system matrix is a $2x2$ matrix and $D$ is the diagonal of eigenvalues in decreasing order, the reliability is simply the second value divided by the first. After obtaining the reliability estimates, the points used to draw and track the feature points are modified as follows:

**Algorithm 3** Modifying Intensity

$R \leftarrow$ Set of reliability factors for tracked points
$C \leftarrow$ Set of predefined colors in RGB corresponding to each point
**for** $r \in Rc \in C$ **do**
  $c_{hsv} \leftarrow$ Conversion of $c$ to HSV color space
  $c_{hsv} \leftarrow$ Modify $v$ value as $v \cdot r$
  $c \leftarrow$ conversion of $c_{hsv}$ toRGB space
**end for**

The colors associated with every point are converted to HSV color space before modifying the intensities as HSV is an orthogonal color space with the $v$ component responsible for intensity. After modifying the intensity, it is again converted back to RGB and retains the chrominance components.

# 4 Results and Discussion

The following images show how the program performed for different images when running the main.py script on multiple video files. The video file names are provided in the figure captions.

Figure 1: Good Tracking for big.mov

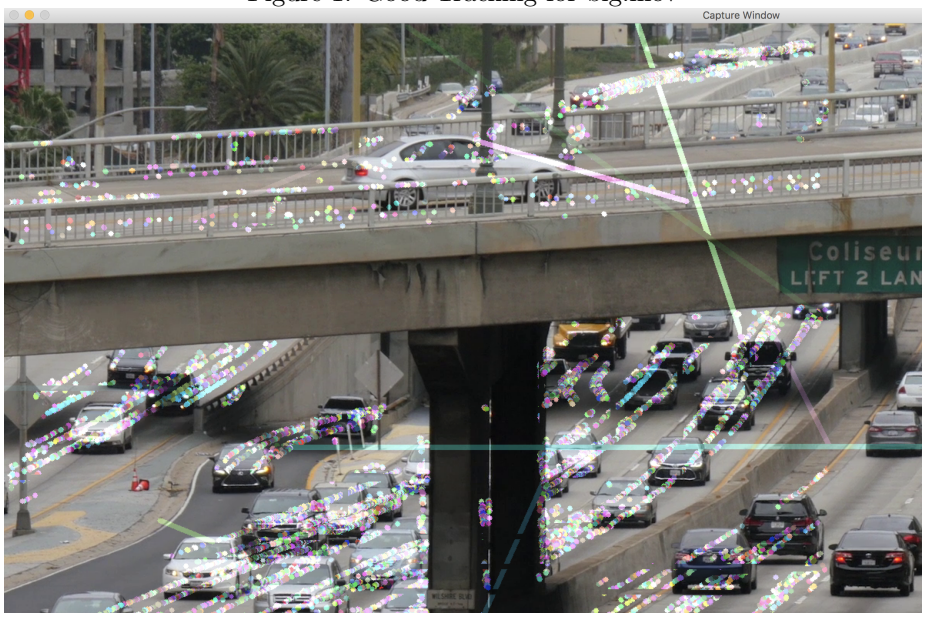

Figure 2: Static Objects Tracked for Walk1.mov

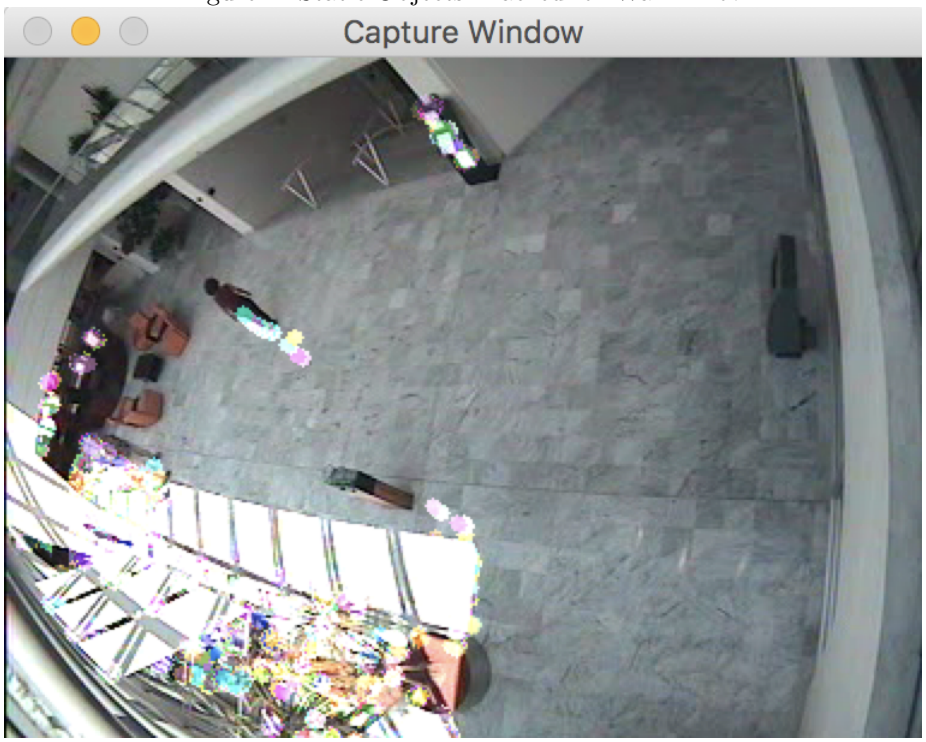

Figure 3: Good Tracking for anni008.mov

Figure 4: Spurious Tracking for 4.mov


The algorithm was able to successfully track most features, it was limited by the corner detector feeding it good features to track. This led to the tracking of static objects as a consequence (see figure 2). Furthermore, as the velocities tend to be small, a lot of the new calculated points end up at the same location as the old ones due to rounding errors. Spurious points were tracked and displaced leading to large drawn lines (see figure 1 and 4). This usually occurred in high resolution video files. Overall, the algorithm performed as expected.

# 5 References

1. `https://docs.opencv.org/3.3.1/d7/d8b/tutorial_py_lucas_kanade.html`
2. `https://www.xilinx.com/support/documentation/application_notes/xapp1300-lucas-kanade-optical-flow.pdf`
3. `http://www.cs.ucf.edu/~mikel/Research/Optical_Flow.htm`
4. `http://www.inf.fu-berlin.de/inst/ag-ki/rojas_home/documents/tutorials/Lucas-Kanade2.pdf`