Contents

1 Basics of Optical Flow **2**
1.1 Thought Experiments . . . 2
1.1.1 Optical flow and slow motion videos. . . . 2
1.1.2 Bullet time scene creation using optical flow. . . . 3
1.1.3 WDMC painterly effects using optical flow. . . . 5
1.1.4 Lambartian ball and optical flow . . . 6
1.2 Concept Review . . . 7
1.2.1 Assumptions made in optical flow estimation. . . . 7
1.2.2 Objective function of classical optical flow problem. . . . 7
1.2.3 Why do we use first order Taylor series approximation. . . . 7
1.2.4 Geometrically show optical flow constraint is ill-posed. . . . 8

2 Single scale Lucas-Kanade Optical Flow **9**
2.1 Keypoint Selection: Selecting Pixels to Track . . . 9
2.1.1 Implement Harris Corner detector and Shi-Tomasi corner detector. . . . 9
2.2 Forward-Additive sparse Optical Flow . . . 11
2.3 Analysing Lucas Kanade Method . . . 13
2.3.1 Why valid when A^TA has rank 2? . . . 13
2.3.2 Note on thresholds used in implementations . . . 13
2.3.3 Varying window sizes. . . . 14
2.3.4 Where Lucas Kanade fails . . . 15
2.3.5 Ground truth in HSV color space . . . 15

3 Multi-Scale Lucas Kanade Algorithm **16**
3.1 Iterative refinement . . . 16
3.2 Multiscale refinement . . . 17

1 Basics of Optical Flow

1.1 Thought Experiments

1.1.1 Optical flow and slow motion videos.

Description: Describe how optical flow could be used to create slow motion video.

Solution: To create slow motion version of an existing video, we can adapt different techniques. Assuming we have a 30fps video and convert it to slow motion by a factor of 2x, it can be turned into a 60fps video in following ways-

1. **By simple frame replication:** If we replicate every frame twice and render the video as a 60fps video, we can have the slow motion effect but it will not be smooth and have abrupt transitions. This is because every other frames contains redundant information due to replication.
2. **Frame blending:** Observing the pixel in two consecutive frames, this technique tries to create an intermediate frame by calculating average location from the two frames. This can create a slow motion video but the effect will not be smooth since individual pixels do not have an understanding of the motion. The sense of direction is important since there may be multiple paths for translation of a point between the two frames as represented in Fig.1
3. **Optical Flow:** This techniques provides motion-vector field around every pixel (dense) or target points (sparse). This information can be used to perform frame interpolation (insert an intermediate estimated frame between two known frames) guided by sense of direction. This way, the inserted frames provide smoother transitions during rendering and the resulting slow motion video appears more realistic and correct.

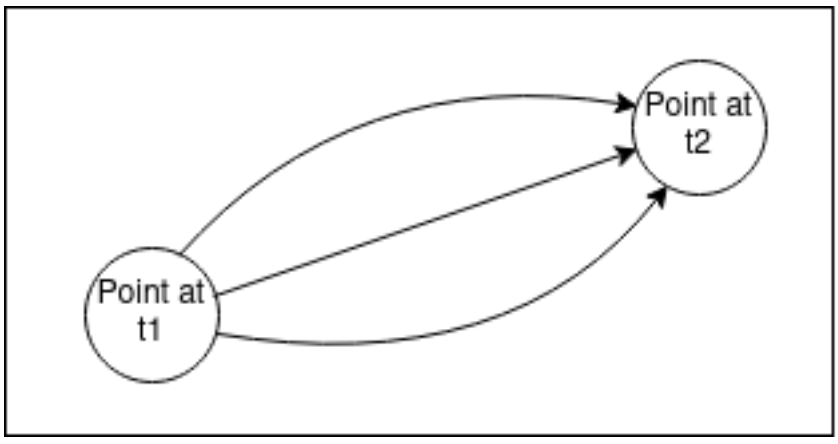

Figure 1: Different possible paths a point can take during transition from one point to another.

1.1.2 Bullet time scene creation using optical flow.

Description: Explain briefly how optical flow was used to create the "Bullet Time" scene from movie, "The Matrix".

Solution: In movie "The Matrix", the bullet time scene is a combination of slow motion rendering along with camera pan around the subject. For this, they use multiple cameras to capture subject from different angles and rendering the frames in slow motion using sense of direction provided by optical flow.
Primarily, optical flow techniques could be used to merge the frames from consecutive cameras to provide a smooth transition and provide a slow motion effect. Breaking down the creation of this scene into smaller modules-

1. **Chroma keying:** The scenes were recorded inside a film studio of a production studio and backgrounds were added using chroma key composting where a frame consists of two layers (foreground and background) where one is separated by a specific colored screen (usually green). Here, this is visible in Fig.3, 6

Figure 2: Background setup with green screens and cameras to capture subject (Neo).

Figure 3: Background substituted using chroma keying.

2. **Camera pan around:** This effect was created by a sophisticated setup where individual cameras were carefully calibrated side to side. This arrangement can be viewed in Fig.4,5. For "Bullet Time" scene, observing Fig.2, the camera arrangement was not always straight (horizontal/vertical) but curved in order to provide the frame transition from curved angles.

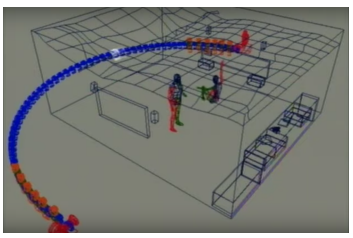

Figure 4: Camera arrangement

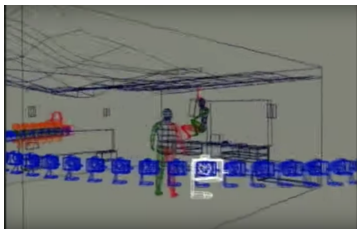

Figure 5: Camera arrangement

Figure 6: Real time arrangement

3. **Merging frames using Optical Flow:** With the mentioned camera arrangement, we have captured the subject from different angles separated by some Δd distance. If we

construct a video by taking one frame from each of the consecutive cameras, we will get a very abrupt transitions due to the Δd separation between them. Thus, to get a smooth transitions, one can use the motion vector field provided by optical flow to interpolate some intermediate frames. This guided interpolation can provide smooth transitions due to correct sense of direction and provide a smooth pan-around effect. This transition can be viewed in Fig.7

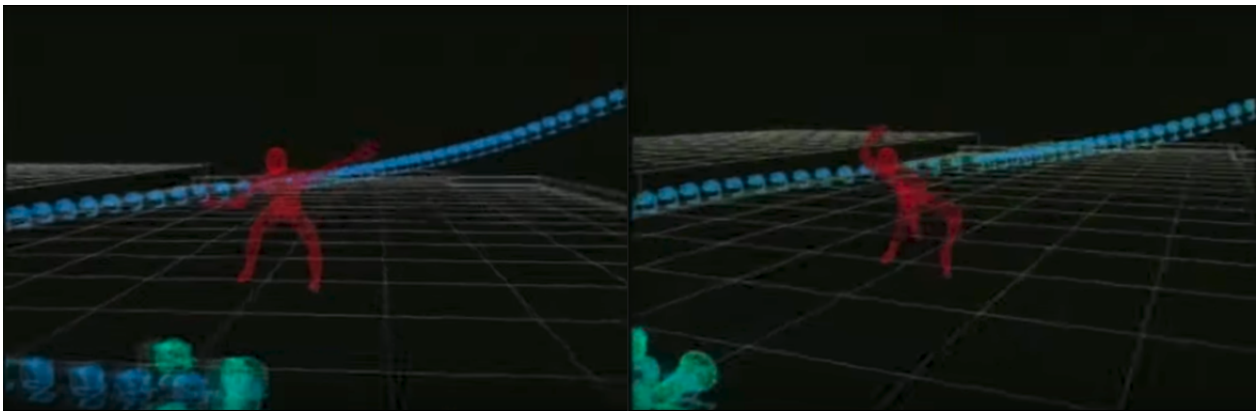

Figure 7: Different frames captured by the camera setup. The transition will be abrup due to Δd separation between them. Optical flow can be used to perform frame interpolation for smooth transitions.

4. **Creating slow motion effect using Optical Flow:** As explained in section 1.1.1, we can provide slow motion effect by implementing frame interpolation guided by the sense of direction from optical flow algorithm. The scene "Bullet Time" used a combination of pan around effect (by using multiple cameras) and slow motion effect (using optical flow) to create that iconic moment.

1.1.3 WDMC painterly effects using optical flow.

Description: Describe briefly on how optical flow is used to create "painterly effect"

Solution: In WDMC, the real world scenes were captured and motion field vectors were optained using the optical flow algorithm. Since working on every frame is not fesible, a frame was selected and a particle tool was applied to get a static painterly image. Using the motion field vectors from the original video, extrapolation was performed over a 3D model of the scene to generate the moving painterly effect which is visible in Fig.10. Fig.8, 9 reflect the use of to apply the processed background.
Some scenes required subject's presence for interactive movements of objects in the real world. One such scene is represented from Fig.11 to Fig.16. Using 3D simulation of real world, artificial elements were added (Fig.14) and there movement was controlled by the real optical flow captured (Fig.12) from the original image.

Figure 8: Subject captured without background

Figure 9: Real background layerd over foreground

Figure 10: Painterly effect added to the background

Figure 11: Actual video with subject going down a hill

Figure 12: Optical flow tracked from real video

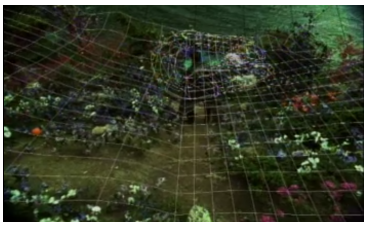

Figure 13: Real world and camera coordinates calibrated for 3D simulation

Figure 14: Elements added in simulation to represent grass, flowers, etc.

Figure 15: Elements refined and optical flow information applied over elements

Figure 16: Final rendered painterly affected video

1.1.4 Lambartian ball and optical flow

Description: Consider a Lambertian ball that is: (i) rotating about its axis in 3D under constant illumination and (ii) stationary and a moving light source. What does the 2D motion field and optical flow look like in both cases.

Solution:

1. For ball rotating about it's axis under constant illumination, **there will not be any optical flow generated but will have motion field** since every point of the Lambertian ball will appear same. There will not be any derrivative and hence the ball will appear still according the optical flow.
2. For stationary ball but moving light source, **the ball will appear to rotate according to the optical flow since the individual patches will have varying light and hence optical flow may take these as patch moving in the direction of light source BUT there will not be any motion field**. Thus, optical flow will apper to move.

1.2 Concept Review

1.2.1 Assumptions made in optical flow estimation.

List down the important assumptions made in optical flow estimation. Describe each one of them in one-two lines.: Solution:

List of primary assumptions made in optical flow estimation-

1. **Brightness constancy:** We assume that the pixels under observation (within the window) will have similar brightness/illumination after translation. If I(x,y,t) represents the image at time t and I(x+u, y+v, t+1) represents the image at time t+1, then we assume that-

$$I(x, y, t) \;=\; I(x + u, y + v, t + 1)$$

2. **Spatial coherance:** We assume that the neighbourhood of the pixels have the same translation (u,v). If this assumption failed, we may not be able to formalize our over-constrained system of equations.

3. **No abrupt changes within frames:** We assume that Δu and Δv are small. This assumption enables us to apply first-order taylor series expantion such that we can ignore the higher order terms.

1.2.2 Objective function of classical optical flow problem.

Formalize the objective function of the classical optical flow problem. Clearly mark the data term and the spatial term. What does the objective function imply about the noise distribution?: Solution:

equation

1.2.3 Why do we use first order Taylor series approximation.

In optimization, why is the first-order Taylor series approximation done?: Solution:

The Taylor series expansion is defined by-

$$\sum_{n=0}^{\infty} \frac{f^n(a)}{n!}(x-a)^n$$

$$\implies f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + ...$$

For optical flow calculation, we made an assumption that the Δu and Δv are very small. Thus the higher order terms become negligible (≈ 0). Thus, we are just left with the first order terms of the Taylor series expansion. Due to our other assumption about brightness constancy, we claim

that $I(x, y, t) = I(x + u, y + v, t + 1)$. In order to get the value of Δu and Δv, we need to expand $T(x + u, y + v, t + 1)$ which gives us-

$$T(x + u, y + v, t + 1) = T(x, y, t) + \frac{\delta I}{\delta x}.u + \frac{\delta I}{\delta y}.v + \frac{\delta I}{\delta t} + (Higher\ order\ terms\ \approx 0)$$

Simplifying which gives us-

$$T(x + u, y + v, t + 1) - T(x, y, t) \approx \frac{\delta I}{\delta x}.u + \frac{\delta I}{\delta y}.v + \frac{\delta I}{\delta t}$$

$$\implies \frac{\delta I}{\delta x}.u + \frac{\delta I}{\delta y}.v + \frac{\delta I}{\delta t} \approx 0$$

$$\implies \bigtriangledown I.[u, v]^T + \frac{\delta I}{\delta t} \approx 0$$

We use this further to formulate our optical flow equations to determine (u,v).

1.2.4 Geometrically show optical flow constraint is ill-posed.

Geometrically show how the optical flow constraint equation is ill-posed. Also, draw the normal flow clearly.: Solution:

The optical flow constrained equation is defined as-

$$\implies \bigtriangledown I.[u, v]^T + \frac{\delta I}{\delta t} \approx 0 \tag{1}$$

Observing the Fig.17, If we assume that the point (u,v) satisfies the equation (1), then so does the point (u+u', v+v'). This is because the equation reflects that the optical flow of the image will lie along the line. Since we don't know where exactly is (u,v) we cannot define a particular constraint and there exist multiple possible solution to the equation. This reflects the ambiguity in motion and marks the equation as ill-posed.

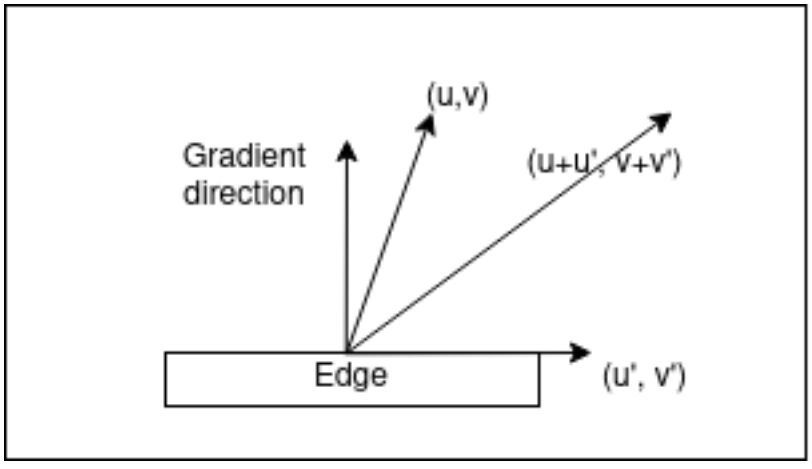

Figure 17: Geometric representation of a case where the optical flow constrained equation fails.

2 Single scale Lucas-Kanade Optical Flow

2.1 Keypoint Selection: Selecting Pixels to Track

2.1.1 Implement Harris Corner detector and Shi-Tomasi corner detector.

Description: Implement (i) Harris Corner Detector and (ii) Shi-Tomasi Corner Detector and visualize the feature set of pixels obtained by both algorithms that will be tracked by the sparse LK method. Visualize the detected pixels superimposed on the images for at least one image from each of the given sequences.

Solution: A class named *corner_detector* was implemented which contains implementation of both harris corner detector and shitomasi corner detector. This can be viewed "corner_detector.py" file in the src folder available with this repository.

1. **Harris Corner Detector:** The algorithm for the same is as follows-

Algorithm 1: Implementing harris corner detector

```
1. Calculate the gradients Ix and Iy using sobel filter
2. Calculate Ixx (Ix^2), Iyy (Iv^2) and Ixy (Ix*Iy).
3. Since the gradient matrix is defined as H = [[Ixx Ixy], [Ixy, Iyy]],
   calculate the determinant of H as (Ixx*Iyy - Ixy**2) and trace of H as (Ixx + Iyy)
4. Using these values, generate a response image as determinant/trace.
5. Apply appropriate threshold and dilate the response image to expand the corners in mask.
6. Convolve a window over the response image to find contours (used cv2.findContours) to
   get individual corners.
7. Store a list with midpoint of every coutour found.
8 Return the list. This represents the detected corners.
```

2. **ShiTomasi corner detector:** ShiTomasi corner detector simply defines a corner based on min(lambda1, lambda1) where lambda1 and lambda2 are eigen values of the gradient matrix. Howere, in this implementaton, there was a slight modification in order to avoid the calculation of eigen values and transform the min() function in terms of determinant and tract.
   The algorithm for the same is as follows-

Algorithm 2: Implementing ShiTomasi corner detector

```
1. Calculate the gradients Ix and Iy using sobel filter
2. Calculate Ixx (Ix^2), Iyy (Iv^2) and Ixy (Ix*Iy).
3. Since the gradient matrix is defined as H = [[Ixx Ixy], [Ixy, Iyy]],
   calculate the determinant of H as (Ixx*Iyy - Ixy**2) and trace of H as (Ixx + Iyy)
4. Using these values, generate a response image as 0.5*(trace - abs(np.sqrt(trace**2 - 4*determinant))).
5. Apply appropriate threshold and dilate the response image to expand the corners in mask.
6. Convolve a window over the response image to find contours (used cv2.findContours) to
   get individual corners.
7. Store a list with midpoint of every coutour found.
8 Return the list. This represents the detected corners.
```

   The runtime results are visible in Fig.18, 20 and 22 for Harris corner detector and Fig.19, 21 and 23 for ShiTomasi corner detector

Runtime:

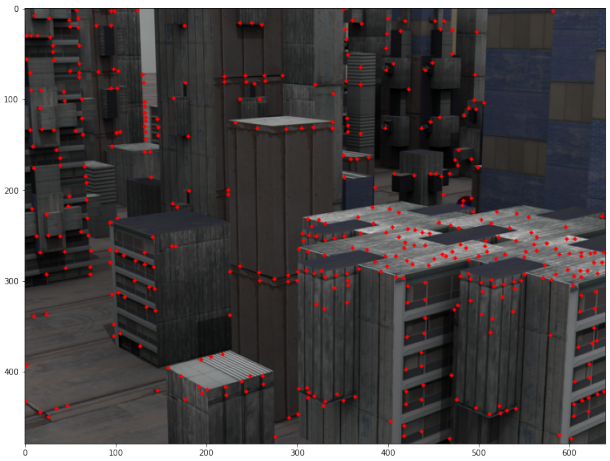

Figure 18: Harris corner detection applied over the Urban2 image

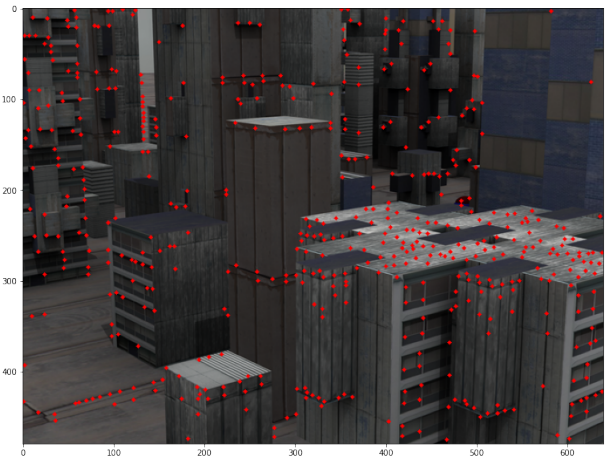

Figure 19: Background substituted using chroma keying.

Figure 20: Harris corner detection applied over the RubberWhale2 image

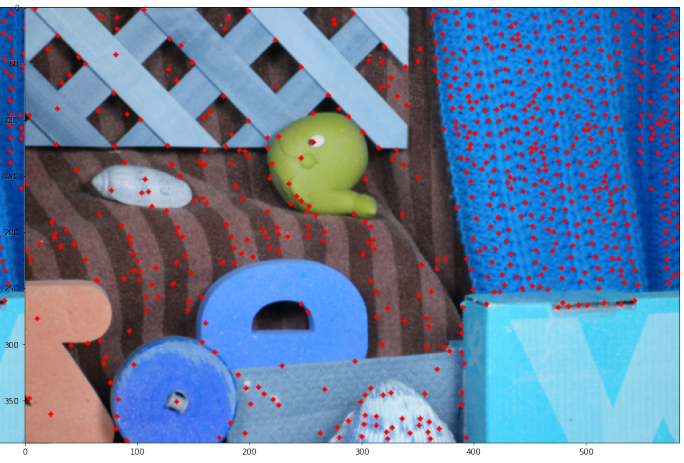

Figure 21: Background substituted using chroma keying.

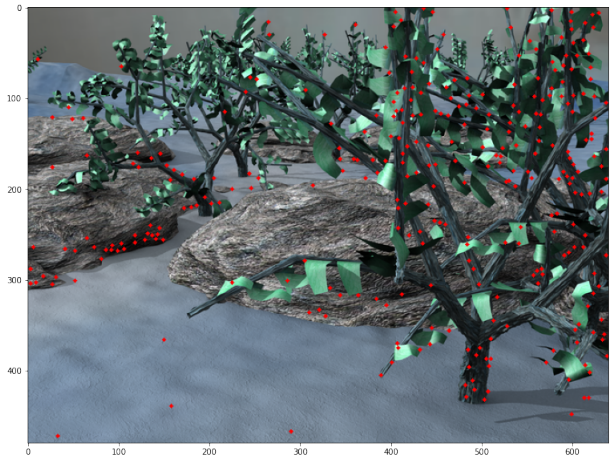

Figure 22: Harris corner detection applied over the Grove3 image

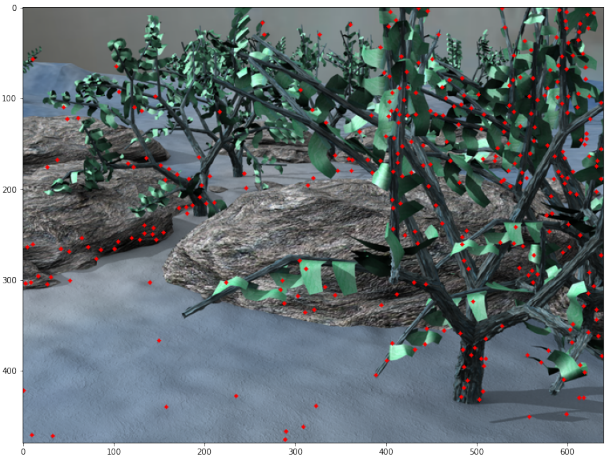

Figure 23: Background substituted using chroma keying.

2.2 Forward-Additive sparse Optical Flow

Description: Implement Sparse Optical Flow algorithm over the given data images using singe scale Lucas Kanade algorithm

Solution: A class names "Lucas_Kanade" was defined in "lucas_kanade_simple_and_iterative.py" file present in the src folder with this report. This class contains the implementation of single scalen and iterative refinement version of lucas kanade algorithm.
The algorithm followed to implement single scale Lucas Kanade algorithm-

Algorithm 3: Implementing ShiTomasi corner detector

```
1. Get the target features from the implemented corner detecor algorithm (harris or shi-tomasi).
2. Normalize the image
3. Determine the Ix and Iy gradients using Sobel filter.
4. Calculate the Ixx as Ix*Ix, Iyy as Iy*Iy and Ixy as Ix*Iy.
5. Calculate It as difference of two images (img2-img1). [Subtracting img1 from img2
   since b vector as negative sign which gets considered within]
6. Iterate over every feature: (x,y)
    6.1. Define a window around the feature (x,y). [Window size is user defined]
    6.2. Extract Ix, Iy, Ixx, Ixy, Iyy and It for the same window.
    6.3. Define matrix (A^t)A = [[sum(Ixx) sum(Ixy)], [sum(Ixy), sum(Iyy)]],
    6.4. Calculate Itx = Ix*It and Ity = Iy*It for the same window
    6.5. Calculate vector (A^t)d = [[sum(Itx) sum(Ity)]].T (and take the transpose of it)
    6.6. Find the inverse of (A^t)A and then find the dot product with (A^t)d.
    6.7. Store the resulting (u,v) direction vectors
7. Plot all the direction vectors (u,v) for each feature over the image. (plt.quiver())
8 Return the (u,v) director vectors. This represents the optical flow across the points.
```

Runtime:

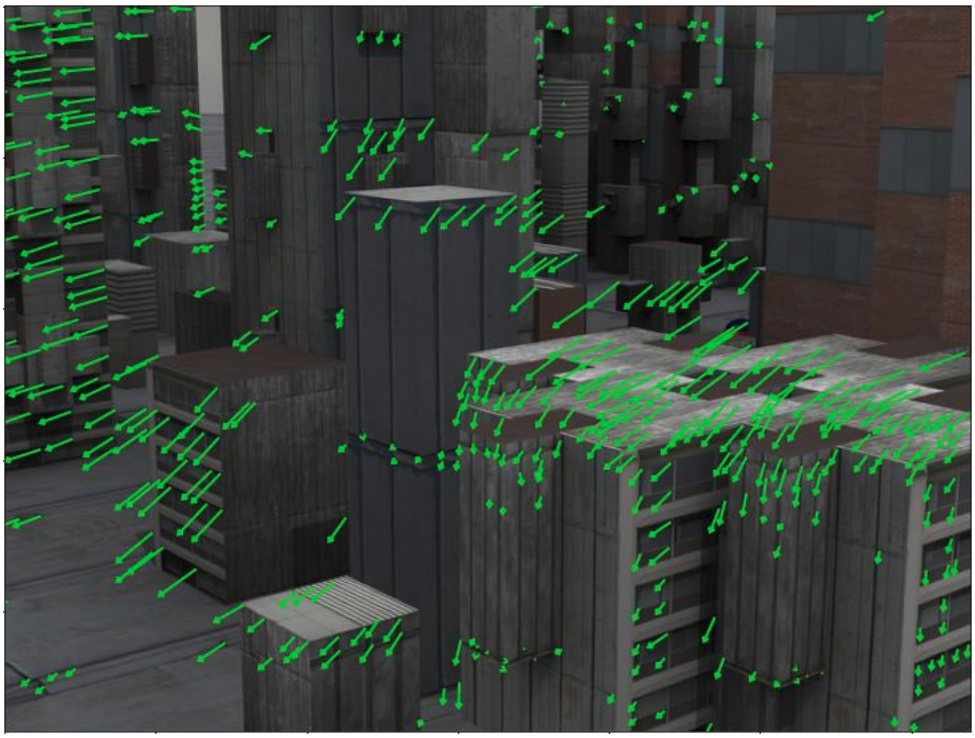

Figure 24: Implement single scale lucas kanade algorithm on Urban2 (frame 7-8)

Figure 25: Implement single scale lucas kanade algorithm on RubberWhale (frame 7-8)

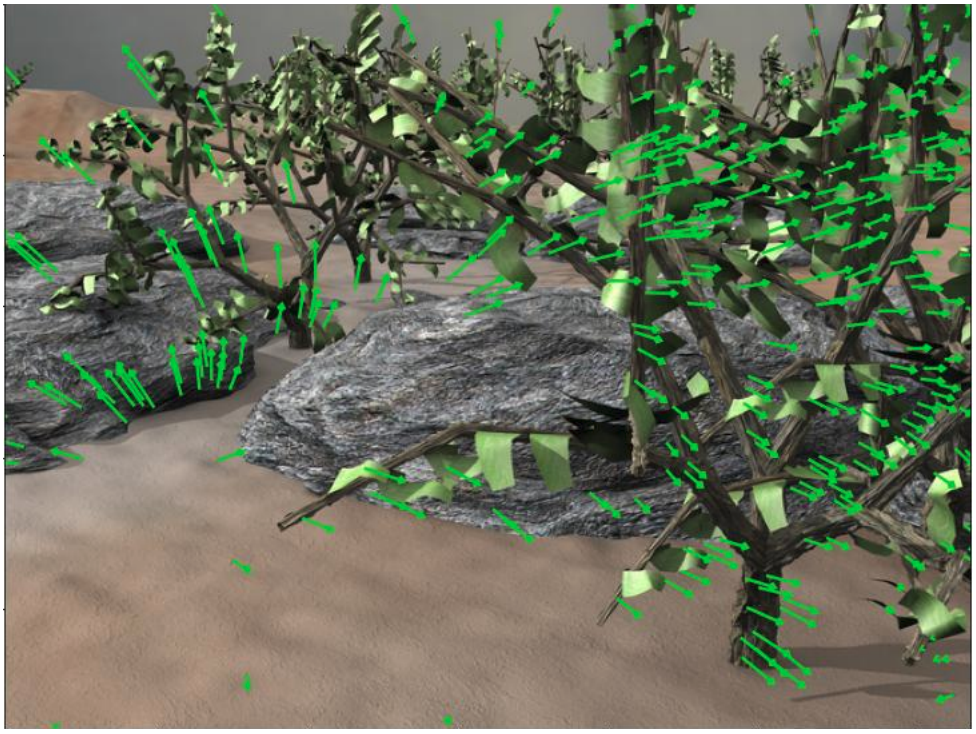

Figure 26: Implement single scale lucas kanade algorithm on grove3 (frame 7-8)

2.3 Analysing Lucas Kanade Method

2.3.1 Why valid when A^TA has rank 2?

Description: Why is optical flow only valid in the regions where local structure tensor A^TA has a rank 2? What role does the threshold τ play here?

Solution: A^TA needs to have rank 2 in order to stay invertible. Being invertible is a primal condition under the defined constraints in order to be solvable to return direction vectors (u,v). If it does not have rank 2, then it may not be invertible or become singular which is not desirable. The threshold τ ignores the bad patches. In our implementation, since we have implemented the sparse version of lucas kanade, we have already defined good selective features and hence varying τ does not have significant effect. In case of dense, varying τ would have filtered noise and we would be able to control the amount of features to be considered for optical flow.

2.3.2 Note on thresholds used in implementations

Description: In the experiments, did you use any modifications and/or thresholds? Does this algorithm work well for these test images? If not, why?

Solution: Yes, there are few thresholds used in the implementation of different algorithms in this section.

1. **A binary threshold to filter corners in Harris Corner Detector:** It was found out that using the response matrix as $\frac{determinant}{trace}$, all the corners were highlighted with gray-level values greater than 230. Lower value added more noise and higher value reduced the number of interest points.

2. **A binary threshold to filter conrers in Shitomasi Corner Detector:** The response matrix generated using the modified min() equation, all the corners were highlighted when threshold beyond 250 was applied. Lower value added more noise and higher value reduced the number of interest points.

3. τ **in Lucas Kanade algorithm:** Although ineffective in sparse optical flow due to selective features, this filters the features if they have smaller eigen values less that τ else keeps them.

2.3.3 Varying window sizes.

Description: Try experimenting with different window sizes. What are the trade-offs associated with using a small versus a large window size? Can you explain what's happening?

Solution: Different experiments were performed by varying the window sizes and it was found out that larger window size (close to 50x50) was desirable for the given images since the movement was large. It is visible in Fig.27. Here, in a window of 50x50, the feature had translated large enough which could not be captures in smaller window. **Thus, keeping a small window size in this case had poor performance and some random direction vectors** whereas **larger window size limited the number of optical flow vectors and also introduce more noise since in a large window, we may loose our primary assumptions regarding spatial coherance and brightness coherance.**

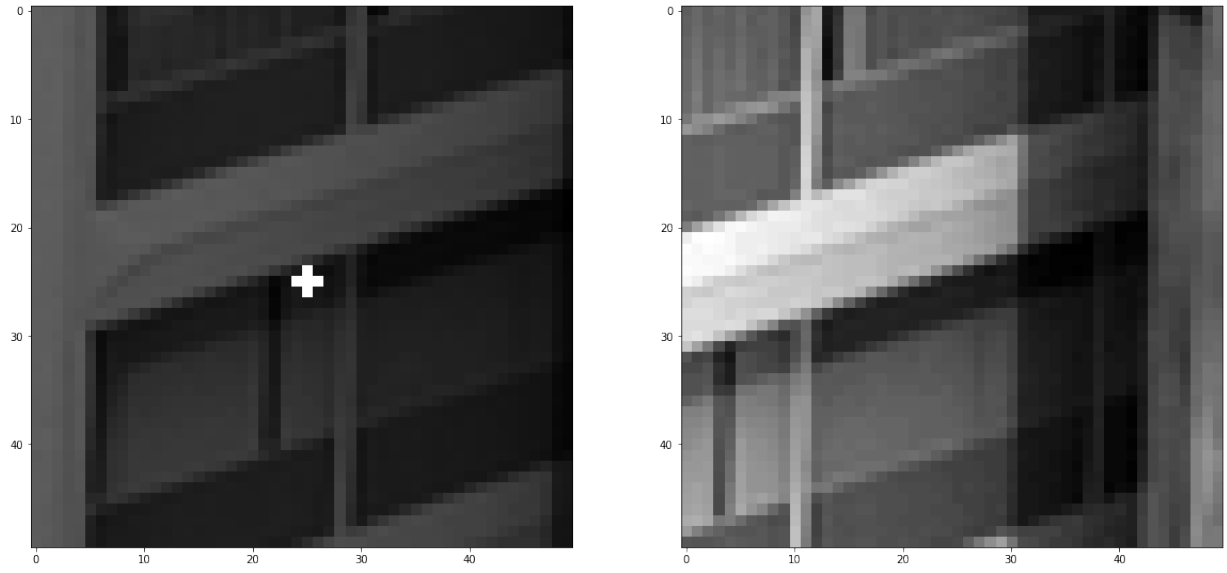

Figure 27: Displacement of a feature (corner) between two consecutive frames of given dataset.

2.3.4 Where Lucas Kanade fails

Description: There are locations in an image where Lucas–Kanade optical flow will fail, regardless of choice of window size and sigma for Gaussian smoothing. Describe two such situations.You are not required to demonstrate them.

Solution: The points such as center of a circle may always fail since all the points in its neighbour, irrespective of the window size, will not follow spatial coherence and hence a certain direction remains undefined. There may exist some other points where half of the portion is shifting in other direction and rest in some other direction. Here also, we may fail to maintaint the spatial coherence and hence fail to define the correct direction vector.

2.3.5 Ground truth in HSV color space

Description: Did you observe that the ground truth visualizations are in HSV colour space? Try to reason it.

Solution:

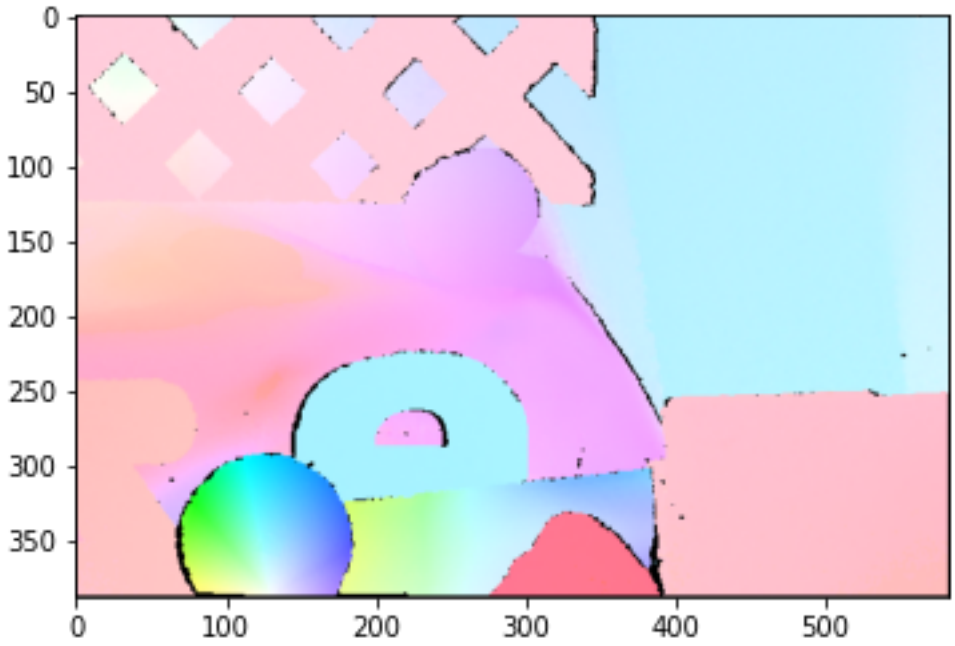

Figure 28: Visualization of ground truth as HSV

3 Multi-Scale Lucas Kanade Algorithm

3.1 Iterative refinement

This is implemented withing the "lucas_kanade_simple_and_iterative.py" under the name of "iterative_solver(window_size, max_iteration)" function.
The results obtained on given dataset images are-

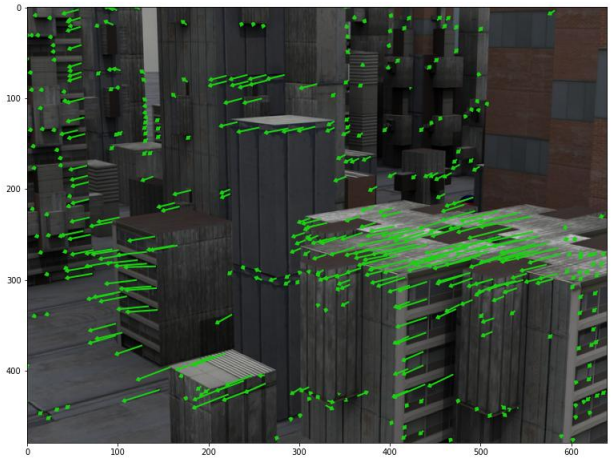

Figure 29: Implement iterative refinement on lucas kanade algorithm on Urban2 (frame 7-8)

Figure 30: Implement iterative refinement on lucas kanade algorithm on RubberWhale (frame 7-8)

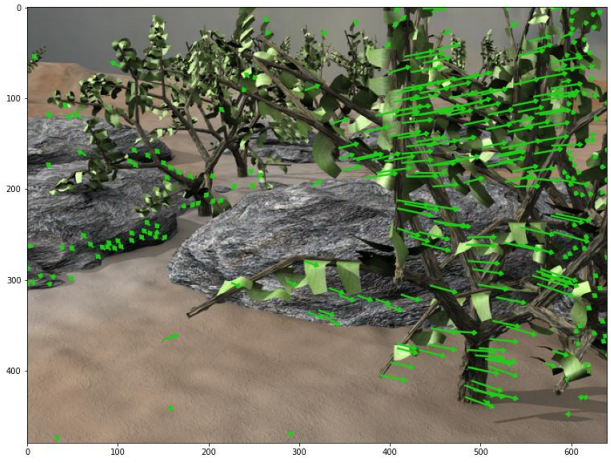

Figure 31: Implement iterative refinement on lucas kanade algorithm on grove3 (frame 7-8)

3.2 Multiscale refinement

This is implemented withing the "lucas_kanade_multiscale.py" available in the src folder with this file.
The results obtained on given dataset images are-

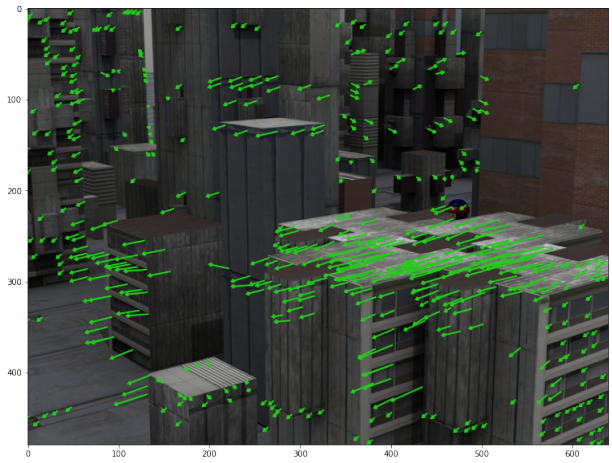

Figure 32: Implement multiscale refinement lucas kanade algorithm on Urban2 (frame 7-8)

Figure 33: Implement multiscale refinement lucas kanade algorithm on RubberWhale (frame 7-8)

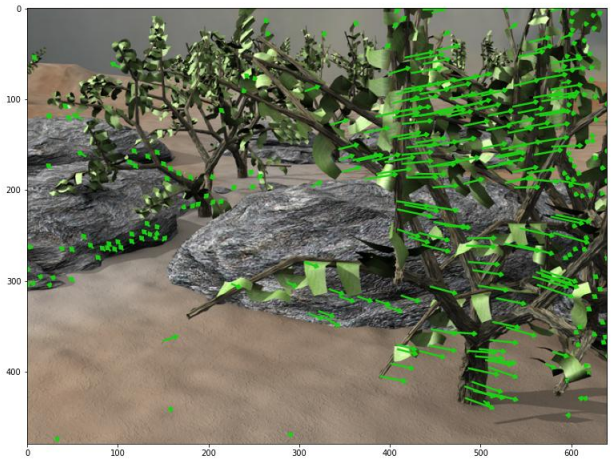

Figure 34: Implement multiscale refinement lucas kanade algorithm on grove3 (frame 7-8)