Computer Vision Assignment-5

MNS Subramanyam
20161190

Lucas-Kanade optical flow algorithm

Procedure

Optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of object or camera. It is 2D vector field where each vector is a displacement vector showing the movement of points from first frame to second.

Optical flow works on several assumptions:

1. The pixel intensities of an object do not change between consecutive frames.
2. Neighbouring pixels have similar motion.

Consider a pixel $I(x, y, t)$ in first frame (Check a new dimension, time, is added here. Earlier we were working with images only, so no need of time). It moves by distance (dx, dy) in next frame taken after dt time. So since those pixels are the same and intensity does not change, we can say,

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

Then take taylor series approximation of right-hand side, remove common terms and divide by dt to get the following equation:

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

where:

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

Above equation is called Optical Flow equation. In it, we can find f_x and f_y, they are image gradients. Similarly f_t is the gradient along time. But (u, v) is unknown. We cannot solve this one equation with two unknown variables. So several methods are provided to solve this problem and one of them is Lucas-Kanade.

The Lucas–Kanade method assumes that the displacement of the image contents between two nearby instants (frames) is small and approximately constant within a neighborhood of the point *p* under consideration. Thus the optical flow equation can be assumed to hold for all pixels within a window centered at *p*. Namely, the local image flow (velocity) vector (V_x, V_y) must satisfy

$$
\begin{aligned}
I_x(q_1)V_x + I_y(q_1)V_y &= -I_t(q_1) \\
I_x(q_2)V_x + I_y(q_2)V_y &= -I_t(q_2) \\
&\vdots \\
I_x(q_n)V_x + I_y(q_n)V_y &= -I_t(q_n)
\end{aligned}
$$

where $q_1, q_2, \ldots, q_n$ are the pixels inside the window, and $I_x(q_i), I_y(q_i), I_t(q_i)$ the partial derivatives of the image **I** with respect to position *x*, *y* and time *t*, evaluated at the point **qi** and at the current time.

These equations can be written in matrix form. $Av = b$ where,

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix} \qquad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix} \qquad b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix}$$

$$A^T A v = A^T b \text{ or}$$
$$\mathrm{v} = (A^T A)^{-1} A^T b$$

Optical Flow Algorithm

In [2]:
```
def optical_flow(I1g, I2g, window_size, tau=1e-2):
    kernel_x = np.array([[-1., 1.], [-1., 1.]])
    kernel_y = np.array([[-1., -1.], [1., 1.]])
    kernel_t = np.array([[1., 1.], [1., 1.]])
    w = int(window_size/2)
    I1g = I1g / 255.
    I2g = I2g / 255.
    mode = 'same'
    fx = signal.convolve2d(I1g, kernel_x, boundary='symm', mode=mode)
    fy = signal.convolve2d(I1g, kernel_y, boundary='symm', mode=mode)
    ft = signal.convolve2d(I2g, kernel_t, boundary='symm', mode=mode) + signal.convolve2d(I1g, -kernel_t, boundary='symm',
    u = np.zeros(I1g.shape)
    v = np.zeros(I1g.shape)
    for i in range(w, I1g.shape[0]-w):
        for j in range(w, I1g.shape[1]-w):
            Ix = fx[i-w:i+w+1, j-w:j+w+1].flatten()
            Iy = fy[i-w:i+w+1, j-w:j+w+1].flatten()
            It = ft[i-w:i+w+1, j-w:j+w+1].flatten()
            b = np.reshape(It, (It.shape[0],1))
            A = np.vstack((Ix, Iy)).T
            if np.min(abs(np.linalg.eigvals(np.matmul(A.T, A)))) >= tau:
                nu = np.matmul(np.linalg.pinv(A), b)
                u[i,j]=nu[0]
                v[i,j]=nu[1]
    return u, v
```

Detection and segmentation of moving objects in a video

Procedure

For the process of segmentation, I have found the values of u^2 + v^2 that is the magnitude of velocity and direction, next we threshold this matrix and plot as a gray scale, we can get the values of segmented image.

Code

Detection and Segmentation

```
im1 = cv2.imread('../data/eval-data-gray/Dumptruck/frame10.png')[:,:,0]
im2 = cv2.imread('../data/eval-data-gray/Dumptruck/frame11.png')[:,:,0]
u, v = optical_flow(im1, im2, 15)
img = u*u + v*v
```

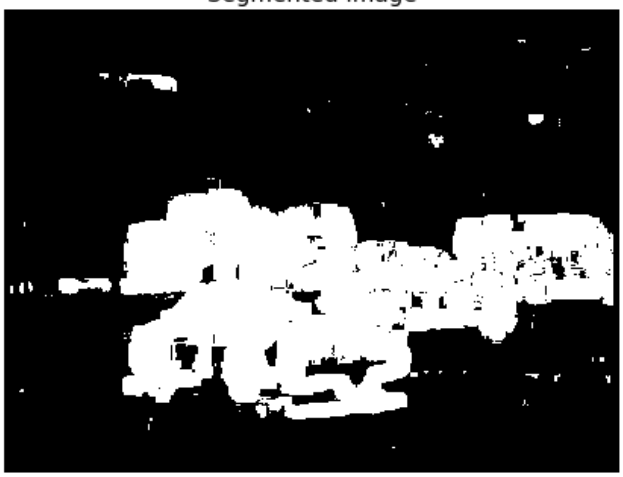
Results

Orginal Image

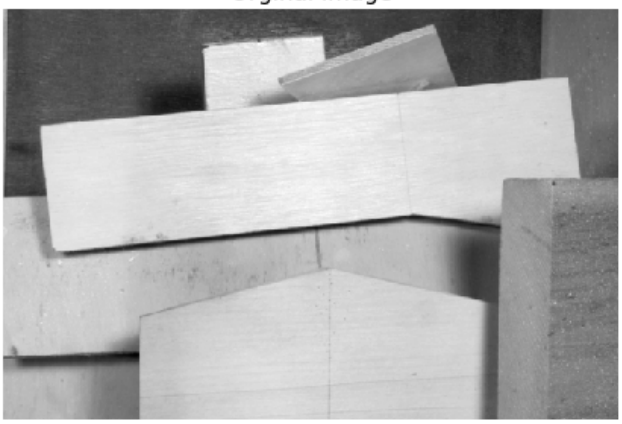
Segmented Image

Orginal Image

Segmented Image



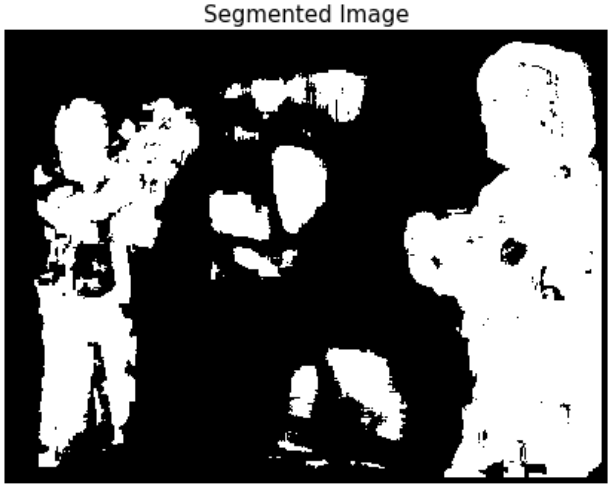
Orginal Image



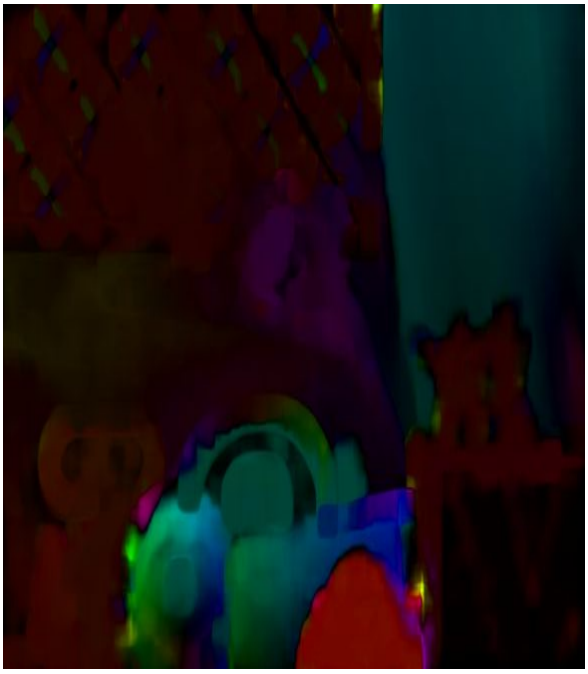
Segmented Image

Orginal Image

Segmented Image

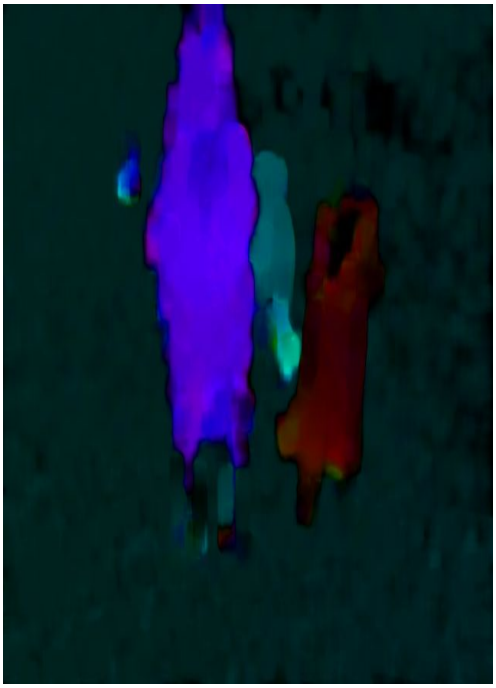


Army :-

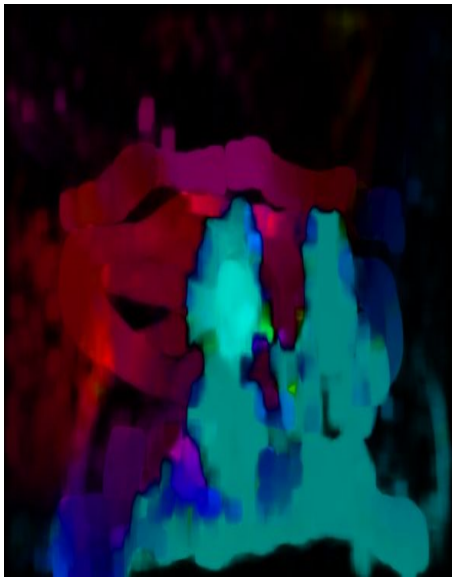


Urban :-

Backyard :-



Mequon :-



Tracking of objects in a video sequence

Procedure

For object tracking we will be showing tracking points for the image.

Code

```
t = 15
plt.figure(figsize=(16, 16))
plt.imshow(im1, cmap=plt.cm.gray)
for i in range(im1.shape[0]):
    print(i,end='\r')
    for j in range(im1.shape[1]):
        if abs(u[i,j])>t or abs(v[i,j])>t: # setting the threshold to plot the vectors
            plt.arrow(j,i,v[i,j],u[i,j],head_width = 1, head_length = 5, color = 'red')

plt.show()
```

Results

Object tracking in the video images.