Lab1

June 14, 2022

1 St121956 - Lab1 Report

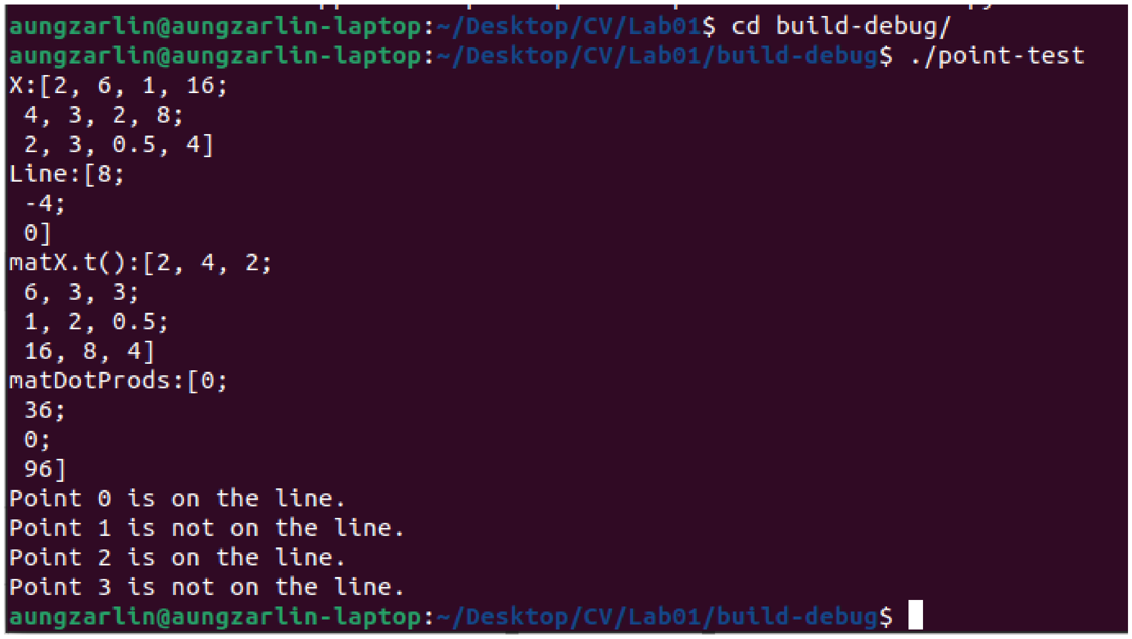
1.1 Exercise to do in lab

1. Write OpenCV C++, OpenCV Python, and Octave code to test which, if any, of the homogeneous 2D points (2, 4, 2), (6, 3, 3), (1, 2, 0.5), and (16, 8, 4) are on the homogeneous 2D line (8, -4, 0). Output the inhomogeneous representations of the points that are on the line.
2. Figure out how to plot the 2D line (8, -4, 0) and the four homogeneous 2D points from exercise 1 above in Octave (you will need to install gnuplot).

1(a). C++ Code

[ ]:
```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <iostream>

using namespace cv;
using namespace std;

int main()
{
    double adDataX[] = { 2, 6, 1, 16, 4, 3, 2, 8, 2, 3, 0.5, 4};
    cv::Mat matX( 3, 4, CV_64F, adDataX );
    cout << "X:" << matX << endl;
    double adDataL[] = { 8, -4, 0 };
    cv::Mat matL( 3, 1, CV_64F, adDataL );
    cout << "Line:" << matL << endl;
    cv::Mat matDotProds = matX.t() * matL;
    cout << "matX.t():" << matX.t() << endl;
    cout << "matDotProds:" << matDotProds << endl;

    for (int i = 0; i < 4; i++) {
        if (fabs(matDotProds.at<double>(i, 0)) < 1e-6) {
            cout << "Point " << i << " is on the line." << endl;
        } else {
            cout << "Point " << i << " is not on the line." << endl;
        }
    }
    return 0;
}
```

Result

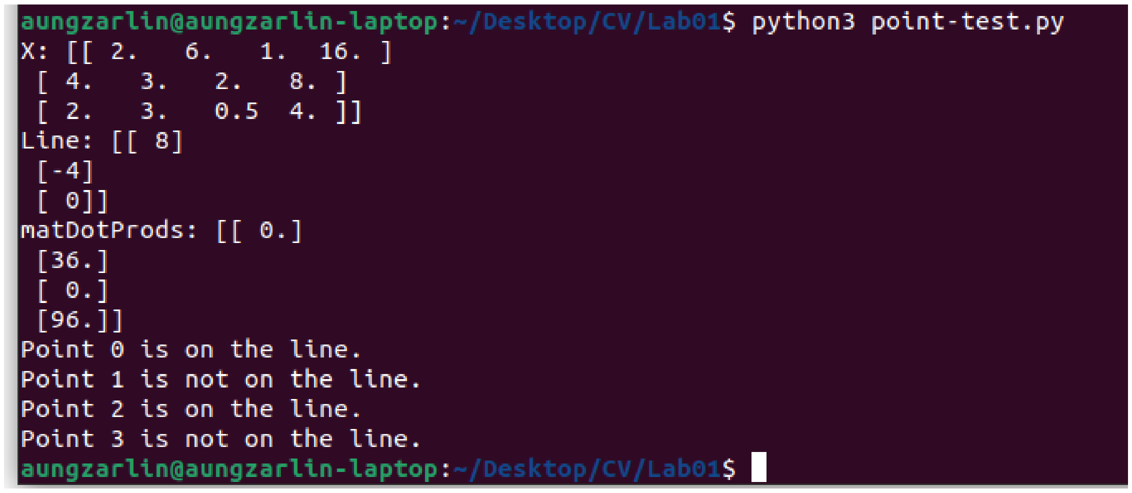


```
aungzarlin@aungzarlin-laptop:~/Desktop/CV/Lab01$ cd build-debug/
aungzarlin@aungzarlin-laptop:~/Desktop/CV/Lab01/build-debug$ ./point-test
X:[2, 6, 1, 16;
 4, 3, 2, 8;
 2, 3, 0.5, 4]
Line:[8;
 -4;
 0]
matX.t():[2, 4, 2;
 6, 3, 3;
 1, 2, 0.5;
 16, 8, 4]
matDotProds:[0;
 36;
 0;
 96]
Point 0 is on the line.
Point 1 is not on the line.
Point 2 is on the line.
Point 3 is not on the line.
aungzarlin@aungzarlin-laptop:~/Desktop/CV/Lab01/build-debug$ 
```

1(b) . Python Code

[ ]:
```
import numpy as np
import cv2

matX = np.array([[ 2, 6, 1, 16], [ 4, 3, 2, 8], [ 2, 3, 0.5, 4]])
matL = np.array([[ 8],[ -4]  , [ 0]])

matDotProds = np.matmul( matX.T, matL )

print("X:", matX)
print("Line:", matL)
print("matDotProds:", matDotProds)

for i in range(4):
    if matDotProds[i,0] < 1e-6:
        print("Point", i, "is on the line.")
    else:
        print("Point", i, "is not on the line.")
```

Result



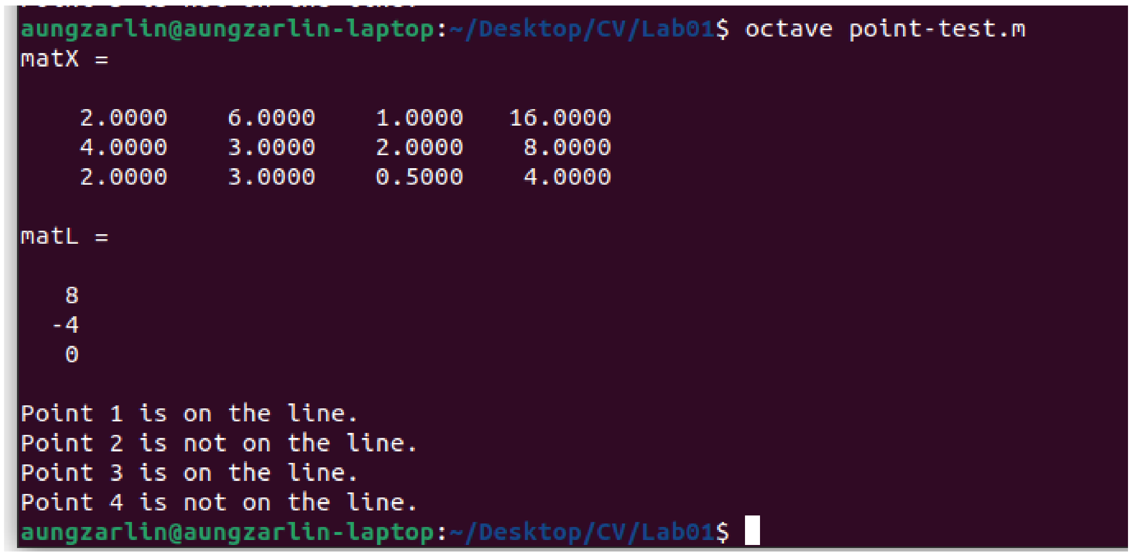
```
aungzarlin@aungzarlin-laptop:~/Desktop/CV/Lab01$ python3 point-test.py
X: [[ 2.   6.   1.  16. ]
 [ 4.   3.   2.   8. ]
 [ 2.   3.   0.5  4. ]]
Line: [[ 8]
 [-4]
 [ 0]]
matDotProds: [[ 0.]
 [36.]
 [ 0.]
 [96.]]
Point 0 is on the line.
Point 1 is not on the line.
Point 2 is on the line.
Point 3 is not on the line.
aungzarlin@aungzarlin-laptop:~/Desktop/CV/Lab01$ 
```

1(c) . Octave code

[ ]:

```
matX = [ 2, 6, 1, 16; 4, 3, 2, 8; 2, 3, 0.5, 4]
matL = [ 8; -4; 0]
matDotProds = matX' * matL;

for i = 1:4,
if matDotProds(i, 1) < 1e-6,
printf('Point %d is on the line.\n', i);
else
printf('Point %d is not on the line.\n', i);
end;
end;
```

Result

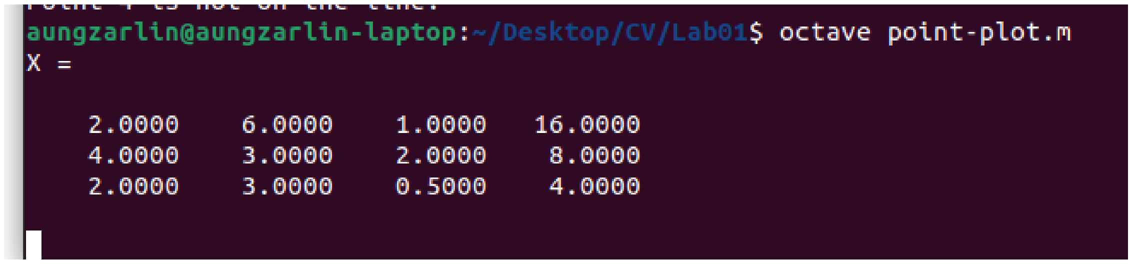


```
aungzarlin@aungzarlin-laptop:~/Desktop/CV/Lab01$ octave point-test.m
matX =

    2.0000    6.0000    1.0000   16.0000
    4.0000    3.0000    2.0000    8.0000
    2.0000    3.0000    0.5000    4.0000

matL =

   8
  -4
   0

Point 1 is on the line.
Point 2 is not on the line.
Point 3 is on the line.
Point 4 is not on the line.
aungzarlin@aungzarlin-laptop:~/Desktop/CV/Lab01$ 
```

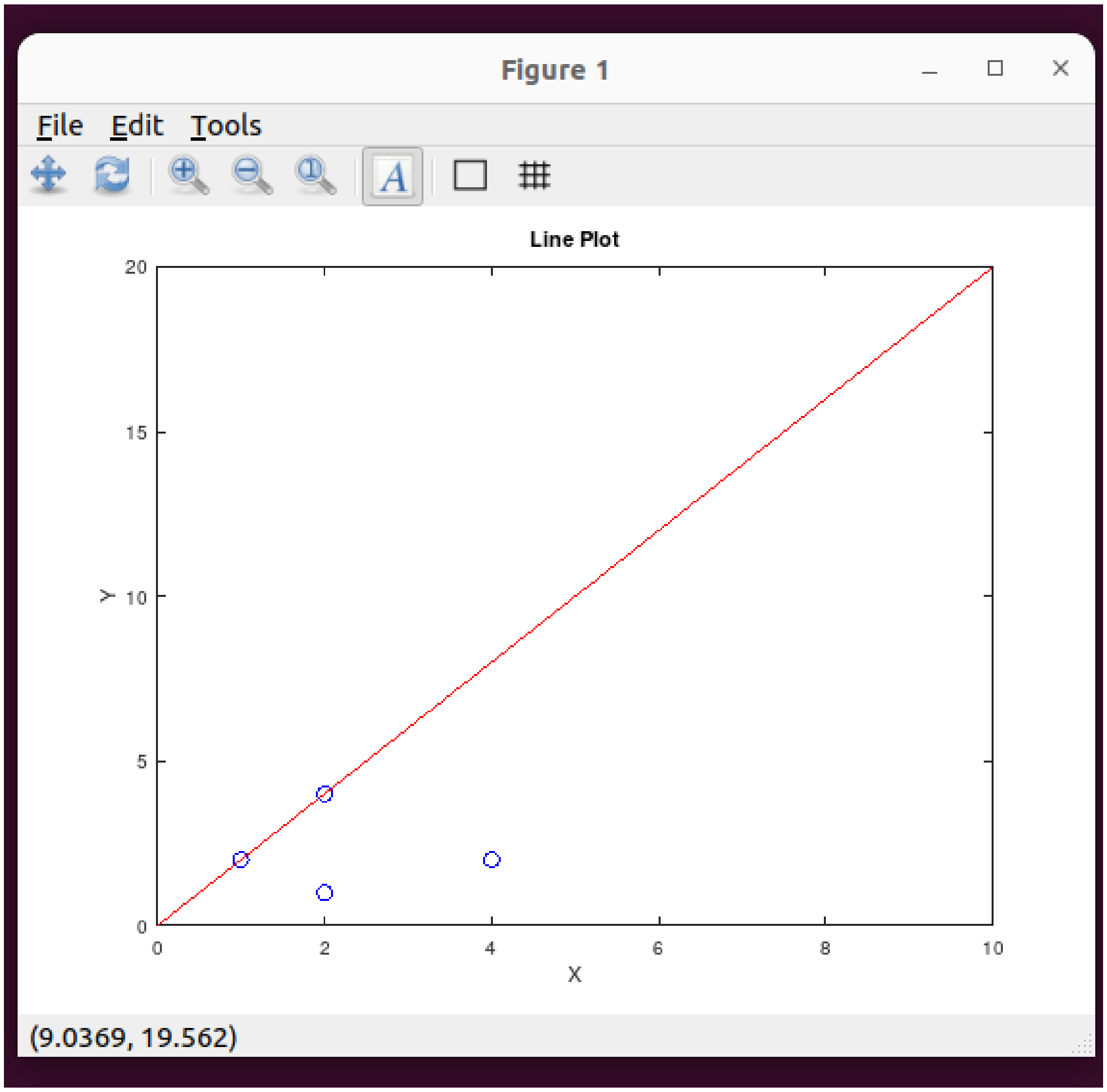
2. Octave code

```
[ ]: % Point and line data

     X = [2, 6, 1, 16; 4, 3, 2, 8; 2, 3, 0.5, 4]
     L = [8; -4; 0];

     % Plot the line


     p1x = 0;
     p1y = (-L(1) * p1x - L(3)) / L(2);

     p2x = 10;
     p2y = (-L(1) * p2x - L(3)) / L(2);

     plot([p1x, p2x],[p1y, p2y], 'r-')
     title('Line Plot')
     xlabel("X");
     ylabel("Y");

     % Plot the points

     hold on;
     for i = 1:4
         plot([X(1,i)/X(3,i)], [X(2,i)/X(3,i)], 'bo')
     end
     waitfor(gcf)
```

Result



```
aungzarlin@aungzarlin-laptop:~/Desktop/CV/Lab01$ octave point-plot.m
X =

    2.0000    6.0000    1.0000   16.0000
    4.0000    3.0000    2.0000    8.0000
    2.0000    3.0000    0.5000    4.0000

```



1.2 Take home Exercise

1. Figure out how to read and display the video in an OpenCV window (using C++!). You may find the VideoCapture tutorial useful.
2. Do the same thing using OpenCV from Python.
3. Get the sparse optical flow example from the OpenCV optical flow tutorial page working on this video. Try both C++ and Python.

1 and 2 is the same as in Lab 2 instruction.

C++ Code

[ ]:
```
#include <opencv2/opencv.hpp>
#include <iostream>

using namespace cv;
using namespace std;

// In C++, you can define constants variable using #define
#define VIDEO_FILE "robot.mp4"
#define ROTATE false

int main(int argc, char** argv)
{
    Mat matFrameCapture;
    Mat matFrameDisplay;
    int iKey = -1;

    // Open input video file
    VideoCapture videoCapture(VIDEO_FILE);
    if (!videoCapture.isOpened()) {
        cerr << "ERROR! Unable to open input video file " << VIDEO_FILE << endl;
        return -1;
    }

    // Capture loop
    while (iKey != int(' '))        // play video until user presses <space>
    {
        // Get the next frame
        videoCapture.read(matFrameCapture);
        if (matFrameCapture.empty())
        {
            // End of video file
            break;
        }

        // We can rotate the image easily if needed.
#if ROTATE
        rotate(matFrameCapture, matFrameDisplay, RotateFlags::ROTATE_180);   //
 ↪rotate 180 degree and put the image to matFrameDisplay
#else
        matFrameDisplay = matFrameCapture;
#endif

        float ratio = 480.0 / matFrameDisplay.rows;
        resize(matFrameDisplay, matFrameDisplay, cv::Size(), ratio, ratio,␣
 ↪INTER_LINEAR); // resize image to 480p for showing

        // Display
        imshow(VIDEO_FILE, matFrameDisplay); // Show the image in window named␣
 ↪"robot.mp4"
        iKey = waitKey(30); // Wait 30 ms to give a realistic playback speed
    }
    return 0;
}
```

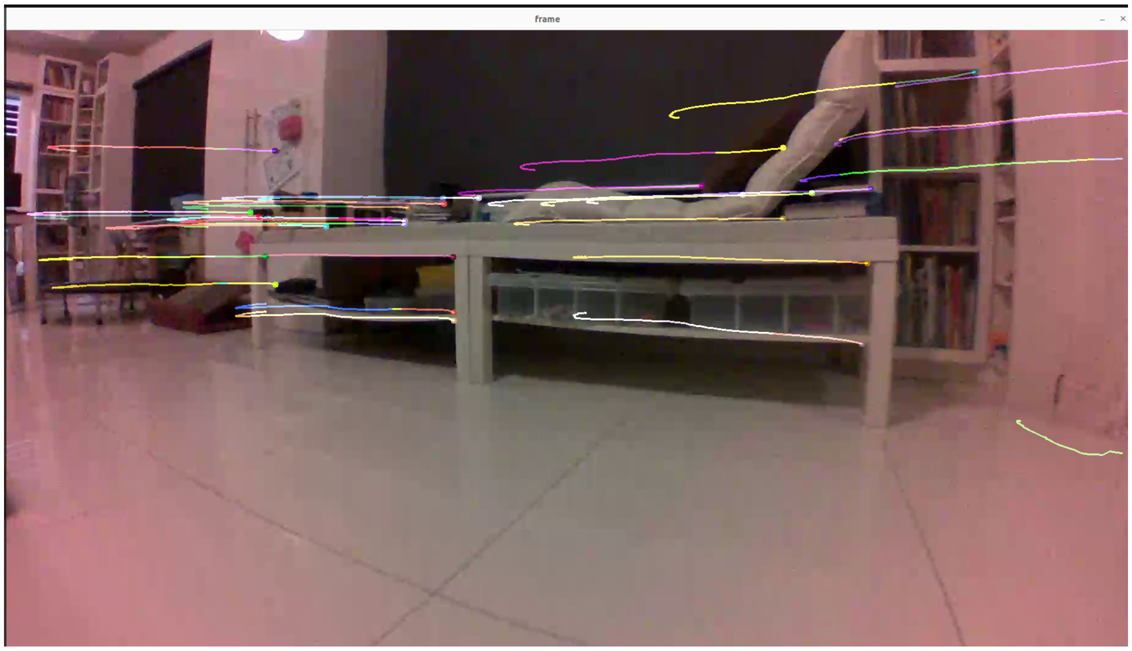
Python Code

[ ]:
```
import cv2
import numpy as np
import sys

VIDEO_FILE = 'robot.mp4'
ROTATE = False

if __name__ == '__main__':

    key = -1;

    # Open input video file
    videoCapture = cv2.VideoCapture(VIDEO_FILE);
    if not videoCapture.isOpened():
        print('Error: Unable to open input video file', VIDEO_FILE)
        sys.exit('Unable to open input video file')

    width  = videoCapture.get(cv2.CAP_PROP_FRAME_WIDTH)   # float `width`
    height = videoCapture.get(cv2.CAP_PROP_FRAME_HEIGHT)  # float `height`

    # Capture loop
    while (key != ord(' ')):         # play video until user presses <space>
        # Get the next frame
        _, matFrameCapture = videoCapture.read()
        if matFrameCapture is None:
            # End of video
            break

        # Rotate if needed
        if ROTATE:
            _, matFrameDisplay = cv2.rotate(matFrameCapture, cv2.ROTATE_180)
        else:
            matFrameDisplay = matFrameCapture;

        ratio = 480.0 / height
        dim = (int(width * ratio), int(height * ratio))
        # resize image to 480p for display
        matFrameDisplay = cv2.resize(matFrameDisplay, dim)

        # Show the image in window named "robot.mp4"
        cv2.imshow(VIDEO_FILE, matFrameDisplay)
        key = cv2.waitKey(30)
```

3. C++ code (Lucus-kanade optical flow)

[ ]:
```
#include <iostream>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/videoio.hpp>
#include <opencv2/video.hpp>
using namespace cv;
using namespace std;

#define VIDEO_FILE "Lab01-robot-video.mp4"

int main(int argc, char **argv)
{
    VideoCapture capture(VIDEO_FILE);
    if (!capture.isOpened()){
        //error in opening the video input
        cerr << "Unable to open file!" << endl;
        return 0;
    }
    // Create some random colors
    vector<Scalar> colors;
    RNG rng;
    for(int i = 0; i < 100; i++)
    {
        int r = rng.uniform(0, 256);
        int g = rng.uniform(0, 256);
        int b = rng.uniform(0, 256);
        colors.push_back(Scalar(r,g,b));
    }
    Mat old_frame, old_gray;
    vector<Point2f> p0, p1;
    // Take first frame and find corners in it
    capture >> old_frame;
    cvtColor(old_frame, old_gray, COLOR_BGR2GRAY);
    goodFeaturesToTrack(old_gray, p0, 100, 0.3, 7, Mat(), 7, false, 0.04);
    // Create a mask image for drawing purposes
    Mat mask = Mat::zeros(old_frame.size(), old_frame.type());
    while(true){
        Mat frame, frame_gray;
        capture >> frame;
        if (frame.empty())
            break;
        cvtColor(frame, frame_gray, COLOR_BGR2GRAY);
        // calculate optical flow
        vector<uchar> status;
        vector<float> err;
        TermCriteria criteria = TermCriteria((TermCriteria::COUNT) +␣
↪(TermCriteria::EPS), 10, 0.03);
        calcOpticalFlowPyrLK(old_gray, frame_gray, p0, p1, status, err,␣
↪Size(15,15), 2, criteria);
        vector<Point2f> good_new;
        for(uint i = 0; i < p0.size(); i++)
        {
            // Select good points
            if(status[i] == 1) {
                good_new.push_back(p1[i]);
                // draw the tracks
                line(mask,p1[i], p0[i], colors[i], 2);
                circle(frame, p1[i], 5, colors[i], -1);
            }
        }
        Mat img;
        add(frame, mask, img);
        imshow("Frame", img);
        int keyboard = waitKey(30);
        if (keyboard == 'q' || keyboard == 27)
            break;
        // Now update the previous frame and previous points
        old_gray = frame_gray.clone();
        p0 = good_new;
    }
}
```

Result

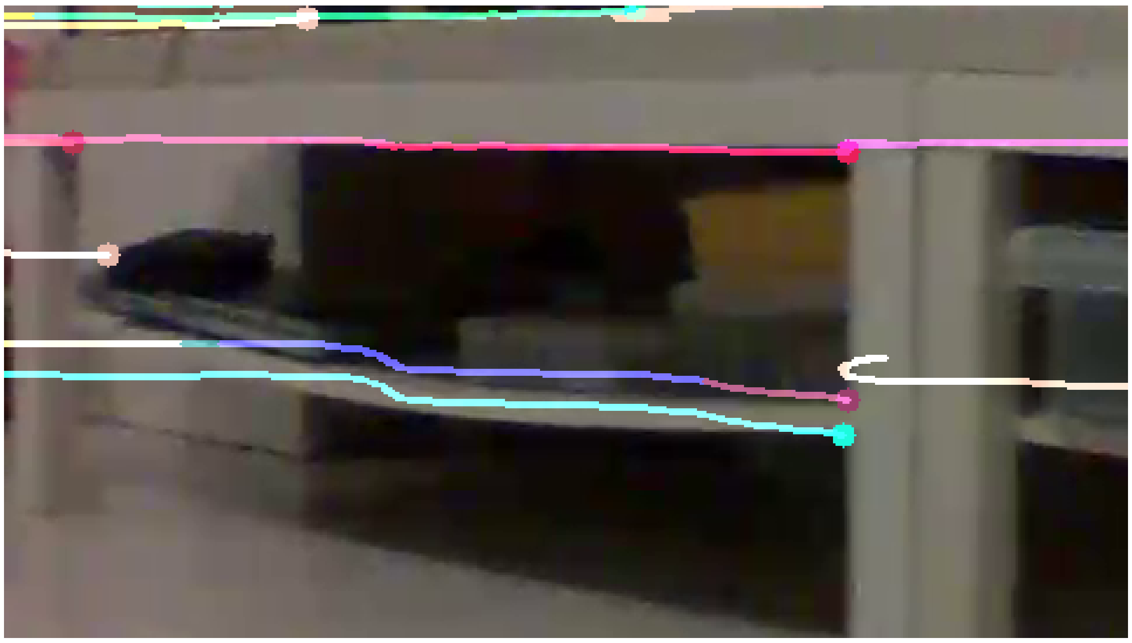


3. Python code (Lucus-kanade optical flow)

```
[ ]: import numpy as np
     import cv2 as cv

     VIDEO_FILE = "Lab01-robot-video.mp4"

     cap = cv.VideoCapture(VIDEO_FILE)
     # params for ShiTomasi corner detection
     feature_params = dict( maxCorners = 100,
                            qualityLevel = 0.3,
                            minDistance = 7,
                            blockSize = 7 )
     # Parameters for lucas kanade optical flow
     lk_params = dict( winSize  = (15, 15),
                       maxLevel = 2,
                       criteria = (cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT,␣
      ↪10, 0.03))
     # Create some random colors
     color = np.random.randint(0, 255, (100, 3))
     # Take first frame and find corners in it
     ret, old_frame = cap.read()
     old_gray = cv.cvtColor(old_frame, cv.COLOR_BGR2GRAY)
     p0 = cv.goodFeaturesToTrack(old_gray, mask = None, **feature_params)
     # Create a mask image for drawing purposes
     mask = np.zeros_like(old_frame)
     while(1):
         ret, frame = cap.read()
         if not ret:
             print('No frames grabbed!')
             break
         frame_gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
         # calculate optical flow
         p1, st, err = cv.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None,␣
      ↪**lk_params)
         # Select good points
         if p1 is not None:
             good_new = p1[st==1]
             good_old = p0[st==1]
         # draw the tracks
         for i, (new, old) in enumerate(zip(good_new, good_old)):
             a, b = new.ravel()
             c, d = old.ravel()
             mask = cv.line(mask, (int(a), int(b)), (int(c), int(d)), color[i].
      ↪tolist(), 2)
             frame = cv.circle(frame, (int(a), int(b)), 5, color[i].tolist(), -1)
         img = cv.add(frame, mask)
```

```
    cv.imshow('frame', img)
    k = cv.waitKey(30) & 0xff
    if k == 27:
        break
    # Now update the previous frame and previous points
    old_gray = frame_gray.copy()
    p0 = good_new.reshape(-1, 1, 2)
cv.destroyAllWindows()
```

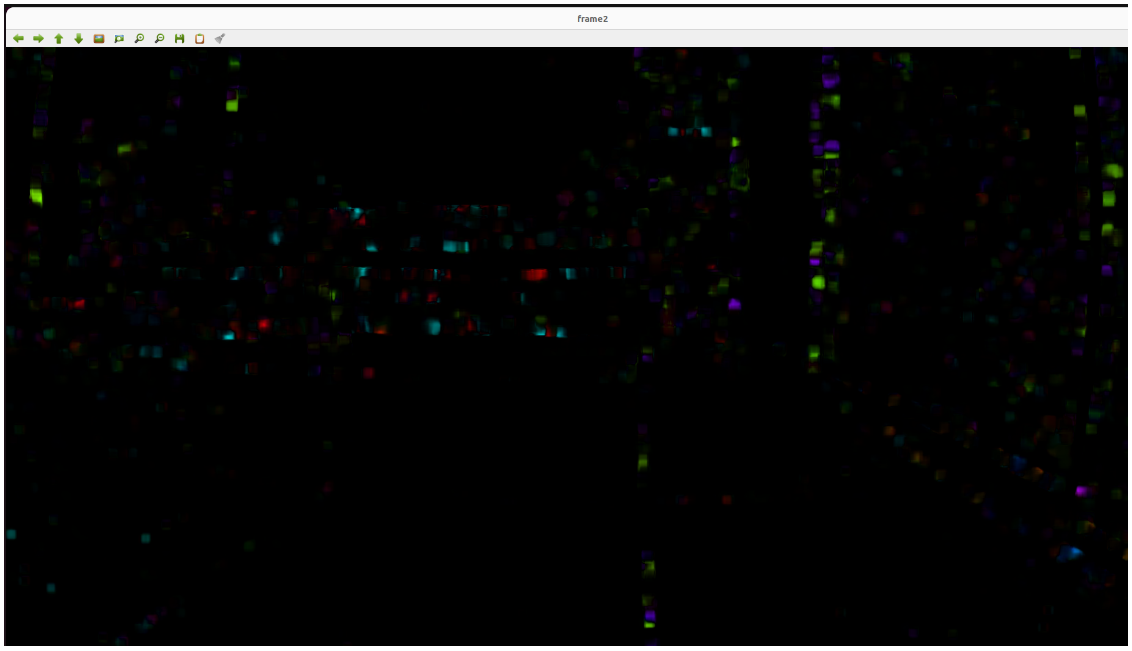
Result



3. C++ Code (Dense Optical Flow)

[ ]:
```
#include <iostream>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/videoio.hpp>
#include <opencv2/video.hpp>
using namespace cv;
using namespace std;

#define VIDEO_FILE "Lab01-robot-video.mp4"

int main()
{
    VideoCapture capture(VIDEO_FILE);
    if (!capture.isOpened()){
        //error in opening the video input
        cerr << "Unable to open file!" << endl;
        return 0;
    }
    Mat frame1, prvs;
    capture >> frame1;
    cvtColor(frame1, prvs, COLOR_BGR2GRAY);
    while(true){
        Mat frame2, next;
        capture >> frame2;
        if (frame2.empty())
            break;
        cvtColor(frame2, next, COLOR_BGR2GRAY);
        Mat flow(prvs.size(), CV_32FC2);
        calcOpticalFlowFarneback(prvs, next, flow, 0.5, 3, 15, 3, 5, 1.2, 0);
        // visualization
        Mat flow_parts[2];
        split(flow, flow_parts);
        Mat magnitude, angle, magn_norm;
        cartToPolar(flow_parts[0], flow_parts[1], magnitude, angle, true);
        normalize(magnitude, magn_norm, 0.0f, 1.0f, NORM_MINMAX);
        angle *= ((1.f / 360.f) * (180.f / 255.f));
        //build hsv image
        Mat _hsv[3], hsv, hsv8, bgr;
        _hsv[0] = angle;
        _hsv[1] = Mat::ones(angle.size(), CV_32F);
        _hsv[2] = magn_norm;
        merge(_hsv, 3, hsv);
        hsv.convertTo(hsv8, CV_8U, 255.0);
        cvtColor(hsv8, bgr, COLOR_HSV2BGR);
        imshow("frame2", bgr);
        int keyboard = waitKey(30);
        if (keyboard == 'q' || keyboard == 27)
            break;
        prvs = next;
    }
}
```

Result

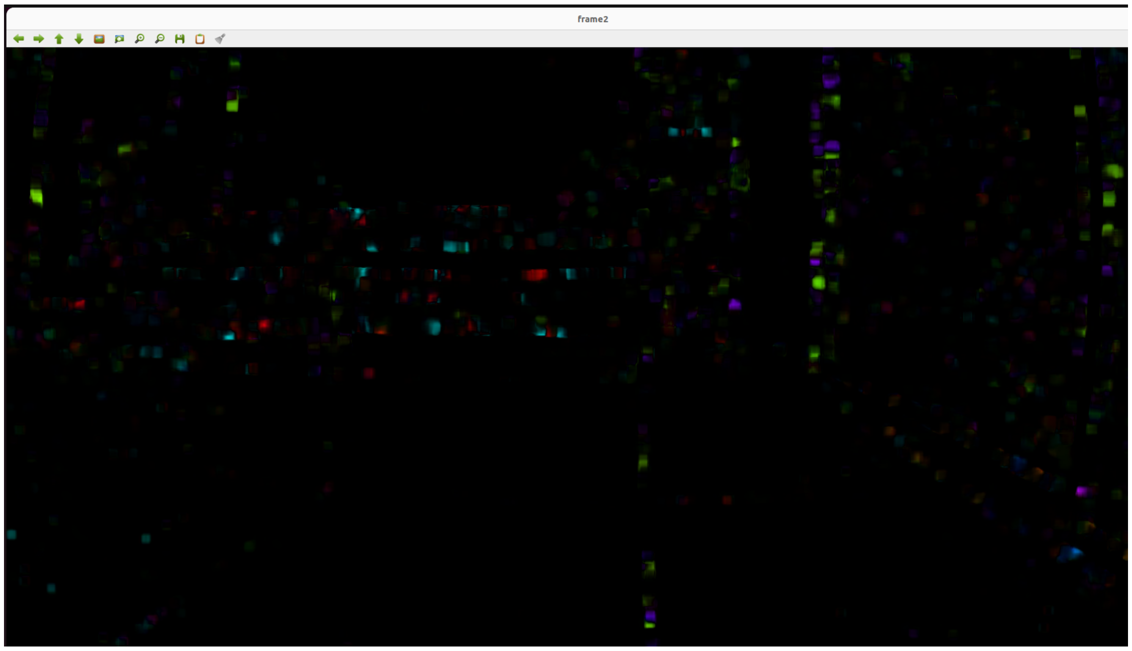


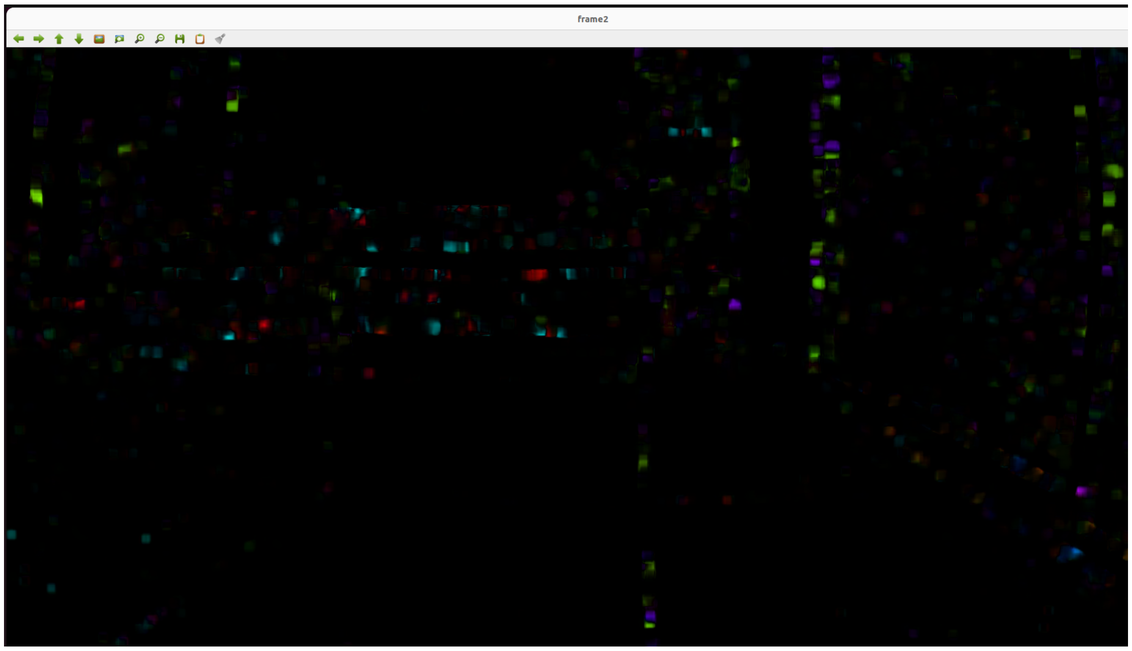
3. Python code (Dense Optical FLow)

```
[ ]: import numpy as np
     import cv2 as cv
     cap = cv.VideoCapture(cv.samples.findFile("Lab01-robot-video.mp4"))
     ret, frame1 = cap.read()
     prvs = cv.cvtColor(frame1, cv.COLOR_BGR2GRAY)
     hsv = np.zeros_like(frame1)
     hsv[..., 1] = 255
     while(1):
         ret, frame2 = cap.read()
         if not ret:
             print('No frames grabbed!')
             break
         next = cv.cvtColor(frame2, cv.COLOR_BGR2GRAY)
         flow = cv.calcOpticalFlowFarneback(prvs, next, None, 0.5, 3, 15, 3, 5, 1.2,␣
      ↪0)
         mag, ang = cv.cartToPolar(flow[..., 0], flow[..., 1])
         hsv[..., 0] = ang*180/np.pi/2
         hsv[..., 2] = cv.normalize(mag, None, 0, 255, cv.NORM_MINMAX)
         bgr = cv.cvtColor(hsv, cv.COLOR_HSV2BGR)
         cv.imshow('frame2', bgr)
         k = cv.waitKey(30) & 0xff
         if k == 27:
             break
         elif k == ord('s'):
             cv.imwrite('opticalfb.png', frame2)
             cv.imwrite('opticalhsv.png', bgr)
         prvs = next
```

```
cv.destroyAllWindows()
```

Result





[ ]:
```

```