CSc 8830: Computer Vision Homework Assignment 3

Submission in Classroom:

For Part A,

convert your problem solving by hand into a digital format (typed or scanned only. You can use camera scanner apps) and embedded/appended into the final PDF documentation. Camera images of paper worksheets will NOT be accepted

For Part B submit a MATLAB Live script (.mlx file) and also convert the .mlx file to PDF and append to PDF from Part A.

The MATLAB Live Script document must contain all the solutions, including graphs. The file must be saved as ".mlx" format. See here for live scripts: https://www.mathworks.com/help/matlab/matlab_prog/create-live-scripts.html

For Part C, manage all your code in a github repo for each assignment. Provide a link to the repo in the PDF document for Part A. Create a working demonstration of your application and record a screen-recording or a properly captured footage of the working system. Upload the video in the Google classroom submission.

Hardware: Unless otherwise specified, use the OAK-D Lite camera provided to you. **Software:** Either of the following will work: Use MATLAB R2018b or later version as installed in your machine (installation instructions already provided) **OR** Use MATLAB Online (https://www.mathworks.com/products/matlab-online.html).

For OAK-D you can implement your solutions in either Python or C/C++: https://docs.luxonis.com/en/latest/

PART A: Theory

1. Capture a 10 sec video footage using a camera of your choice. The footage should be taken with the camera in hand and you need to pan the camera slightly from left-right or right-left during the 10 sec duration. Pick any image frame from the 10 sec video footage. Pick a region of interest corresponding to an object in the image. Crop this region from the image. Then use this cropped region to compare with randomly picked 10 images in the dataset of 10 sec video frames, to see if there is a match for the object in the scenes from the 10 images. For comparison use sum of squared differences (SSD) or normalized correlation.

[24]
```python
import cv2
import glob
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import fftconvolve
import math
```

[25]
```python
from google.colab.patches import cv2_imshow
```

[26]
```python
!curl -o logo.png https://colab.research.google.com/img/colab_favicon_256px.png
import cv2
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  4534  100  4534    0     0  78172      0 --:--:-- --:--:-- --:--:-- 78172
```

[27]
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```python
import cv2
vidcap = cv2.VideoCapture('/content/cv_assign3_vdo.mp4')
success,image = vidcap.read()
count = 0
while success:
  cv2.imwrite("frame%d.jpg" % count, image)     # save frame as JPEG file
  success,image = vidcap.read()
  print('Read a new frame: ', success)
  count += 1
```

```
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
```

[29]
```python
def ssd(A,B):
    squares = (A[:,:,:3] - B[:,:,:3]) ** 2
    return math.sqrt(np.sum(squares))
```

[34]
```python
import cv2

imdir = ''
ext = ['jpg']    # Add image formats here

files = []
[files.extend(glob.glob(imdir + '*.' + e)) for e in ext]

images = [cv2.imread(file) for file in files]
```

```python
plt.imshow(cv2.cvtColor(images[0], cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7f5b3e8f7b10>
```

[39]
```python
cropped_image = images[0][100:200,430:530]
plt.imshow(cv2.cvtColor(cropped_image, cv2.COLOR_BGR2RGB))

cv2.imwrite("Cropped Image.jpg", cropped_image)
```

```
True
```

```
cropped_image = images[0][100:200,430:530]
plt.imshow(cv2.cvtColor(cropped_image, cv2.COLOR_BGR2RGB))

cv2.imwrite("Cropped Image.jpg", cropped_image)
```

True

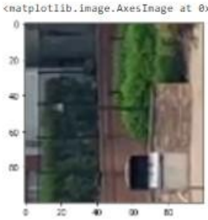
```
d=dict()
```

```
d=dict()
d_norm=dict()
for i in range(100,200,20):
    for j in range(430,530,20):
        d[str(i)+":"+str(i+100),str(j)+":"+str(j+100)]=ssd(cropped_image,images[0][i:i+100,j:j+100])
        #d_norm[str(i)+":"+str(i+100),str(j)+":"+str(j+100)]=ncc(norm_data(cropped_image),norm_data(images[0][i:i+100,j:j+100]))
```

```
[75] a=min(d.items(), key=lambda x: x[1])
     y1,y2=map(int,a[0][0].split(':'))
     x1,x2=map(int,a[0][1].split(':'))
```

```
[76] plt.imshow(cv2.cvtColor(images[0][y1:y2,x1:x2], cv2.COLOR_BGR2RGB))
```

<matplotlib.image.AxesImage at 0x7f5b3e3fb350>



```
[77] color = (255, 0, 0)
```

```
[77] color = (255, 0, 0)

     # Line thickness of 2 px
     thickness = 2

     # Using cv2.rectangle() method
     # Draw a rectangle with blue line borders of thickness of 2 px
     image = cv2.rectangle(images[0], (x1,y1), (x2,y2), color, thickness)
```

```
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
```

<matplotlib.image.AxesImage at 0x7f5b3e3dd890>

```python
cap = cv.VideoCapture("/content/cv_assign3_vdo.mp4")
ret, first_frame = cap.read()
prev_gray = cv.cvtColor(first_frame, cv.COLOR_BGR2GRAY)
mask = np.zeros_like(first_frame)
mask[..., 1] = 255

while(cap.isOpened()):

    ret, frame = cap.read()

    # Opens a new window and displays the input
    # frame
    cv2_imshow(frame)

    # Converts each frame to grayscale - we previously
    # only converted the first frame to grayscale
    gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)

    # Calculates dense optical flow by Farneback method
    flow = cv.calcOpticalFlowFarneback(prev_gray, gray,None,0.5, 3, 15, 3, 5, 1.2, 0)

    # Computes the magnitude and angle of the 2D vectors
    magnitude, angle = cv.cartToPolar(flow[..., 0], flow[..., 1])
    # Sets image hue according to the optical flow
    # direction
    mask[..., 0] = angle * 180 / np.pi / 2

    # Sets image value according to the optical flow
    # magnitude (normalized)
    mask[..., 2] = cv.normalize(magnitude, None, 0, 255, cv.NORM_MINMAX)

    # Converts HSV to RGB (BGR) color representation
    rgb = cv.cvtColor(mask, cv.COLOR_HSV2BGR)

    # Opens a new window and displays the output frame
    cv2_imshow(rgb)

    # Updates previous frame
    prev_gray = gray

    # Frames are read by intervals of 1 millisecond. The
```

another tab. Show diff

0s completed at 10:50 PM

```
import os
import time

import imutils
detectorPaths = {

    "face": "face.xml",
    "smile": "smile.xml",
}

print("[INFO] loading haar cascades...")
detectors = dict()

for (name, path) in detectorPaths.items():
    detectors[name] = cv2.CascadeClassifier(path)

print("[INFO] starting video stream...")
vs = cv2.VideoCapture(0)

while True:
    _,frame = vs.read()
    frame = imutils.resize(frame, width=500)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faceRects = detectors["face"].detectMultiScale(
        gray, scaleFactor=1.05, minNeighbors=5, minSize=(30, 30),
        flags=cv2.CASCADE_SCALE_IMAGE)

    for (fX, fY, fW, fH) in faceRects:
        faceROI = gray[fY:fY + fH, fX:fX + fW]
        smileRects = detectors["smile"].detectMultiScale(
            faceROI, scaleFactor=1.1, minNeighbors=10,
            minSize=(15, 15), flags=cv2.CASCADE_SCALE_IMAGE)
        for (sX, sY, sW, sH) in smileRects:
            ptA = (fX + sX, fY + sY)
            ptB = (fX + sX + sW, fY + sY + sH)
            cv2.rectangle(frame, ptA, ptB, (255, 0, 0), 2)
        cv2.rectangle(frame, (fX, fY), (fX + fW, fY + fH),
                      (0, 255, 0), 2)
    cv2.imshow("Frame", frame)
    if cv2.waitKey(1) == ord("q"):
        break
```


```
while True:
    _,frame = vs.read()
    frame = imutils.resize(frame, width=500)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faceRects = detectors["face"].detectMultiScale(
        gray, scaleFactor=1.05, minNeighbors=5, minSize=(30, 30),
        flags=cv2.CASCADE_SCALE_IMAGE)

    for (fX, fY, fW, fH) in faceRects:
        faceROI = gray[fY:fY + fH, fX:fX + fW]
        smileRects = detectors["smile"].detectMultiScale(
            faceROI, scaleFactor=1.1, minNeighbors=10,
            minSize=(15, 15), flags=cv2.CASCADE_SCALE_IMAGE)
        for (sX, sY, sW, sH) in smileRects:
            ptA = (fX + sX, fY + sY)
            ptB = (fX + sX + sW, fY + sY + sH)
            cv2.rectangle(frame, ptA, ptB, (255, 0, 0), 2)
        cv2.rectangle(frame, (fX, fY), (fX + fW, fY + fH),
                      (0, 255, 0), 2)
    cv2.imshow("Frame", frame)
    if cv2.waitKey(1) == ord("q"):
        break
```

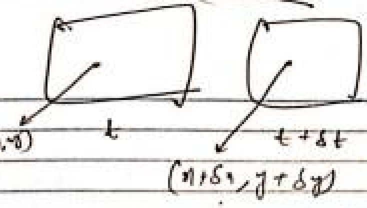

```
cv2.destroyAllWindows()
```

```
[INFO] loading haar cascades...
[INFO] starting video stream...
```

2. Derive the motion tracking equation from fundamental principles. Select any 2 consecutive frames from the set

from problem 1 and compute the motion function estimates.

optical flow Constraint function



disp = $(\delta x, \delta y)$ optical flow $(u,v) = \left(\frac{\delta x}{\delta t}, \frac{\delta y}{\delta t}\right)$

umption 1:
Brightness

$$I(x+\delta x, y+\delta y, t+\delta t) = I(x,y,t)$$

umption 2:
displacement $(\delta x, \delta y)$ and time step δt are small

$I(x,y,t)$ · $I(x+\delta x, y+\delta y, t+\delta t)$

Taylor series Expansion
Expand a function as an infinite sum of its derivatives.

$$f(x+\delta x) = f(x) + \frac{\partial f}{\partial x}\delta x + \frac{\partial^2 f}{\partial x^2}\frac{\delta x^2}{2!} + \cdots \frac{\partial^n f}{\partial x^n}\frac{\delta x^n}{n!}$$

If δx is small:

$$f(x+\delta x) = f(x) + \frac{\partial f}{\partial x}\delta x + O(\delta x^2) \rightarrow \text{almost zero}$$

For a function of 3 variables with small $\delta x, \delta y, \delta t$:

$$f(x+\delta x, y+\delta y, t+\delta t) \approx f(x,y,t) + \frac{\partial f}{\partial x}\delta x + \frac{\partial f}{\partial y}\delta y + \frac{\partial f}{\partial t}\delta t$$

from Assumption #2.

$$I(x+\delta x, y+\delta y, t+\delta t) = I(x,y,t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t$$

$$I(x+\delta x, y+\delta y, t+\delta t) = I(x,y,t) + I_x\delta x + I_y\delta y + I_t\delta t$$

$$I(x+\delta x, y+\delta y, t+\delta t) = I(x,y,t) \quad \text{— (1)}$$

$$I(x+\delta x, y+\delta y, t+\delta t) = I(x,y,t) + I_x\delta x + I_y\delta y + I_t\delta t \quad \text{— (2)}$$

Sub (1) from (2) : $I_x\delta x + I_y\delta y + I_t\delta t = 0$

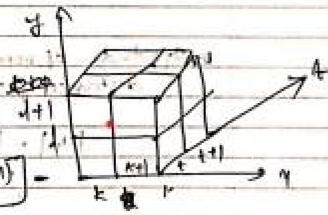
divide by δt and take limit as $\delta t \to 0$: $I_x\frac{\partial x}{\partial t} + I_y\frac{\partial y}{\partial t} + I_t = 0$

Constraint Equation: $\boxed{I_x u + I_y v + I_t = 0}$

(u,v) : optical flow

Computation partial derivatives (I_x, I_y, I_t) can be easily computed from 2 frames

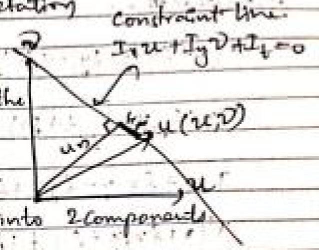
$I_x(k,l,t) =$



$$\frac{1}{4}\left[I(k+1,l,t) + I(k+1,l,t+1) + I(k+1,l+1,t) + I(k+1,l+1,t+1)\right] - \frac{1}{4}\left[I(k,l,t) + I(k,l,t+1) + I(k,l+1,t) + I(k,l+1,t+1)\right]$$

Similarly for $I_y(k,l,t)$ and $I_t(k,l,t)$

Geometric Interpretation



for any point (x,y) in the Image
its optical flow (u,v) lies on the line:

$$I_x u + I_y v + I_t = 0$$

optical flow can be split into 2 components

$$u = u_n + u_p$$

u_n = Normal flow
u_p = Parallel flow

Normal flow:

direction of normal flow
unit vector $\perp^{ar}$ to the Constraint line.

$$\hat{u}_n = \frac{(I_x, I_y)}{\sqrt{I_x^2 + I_y^2}}$$

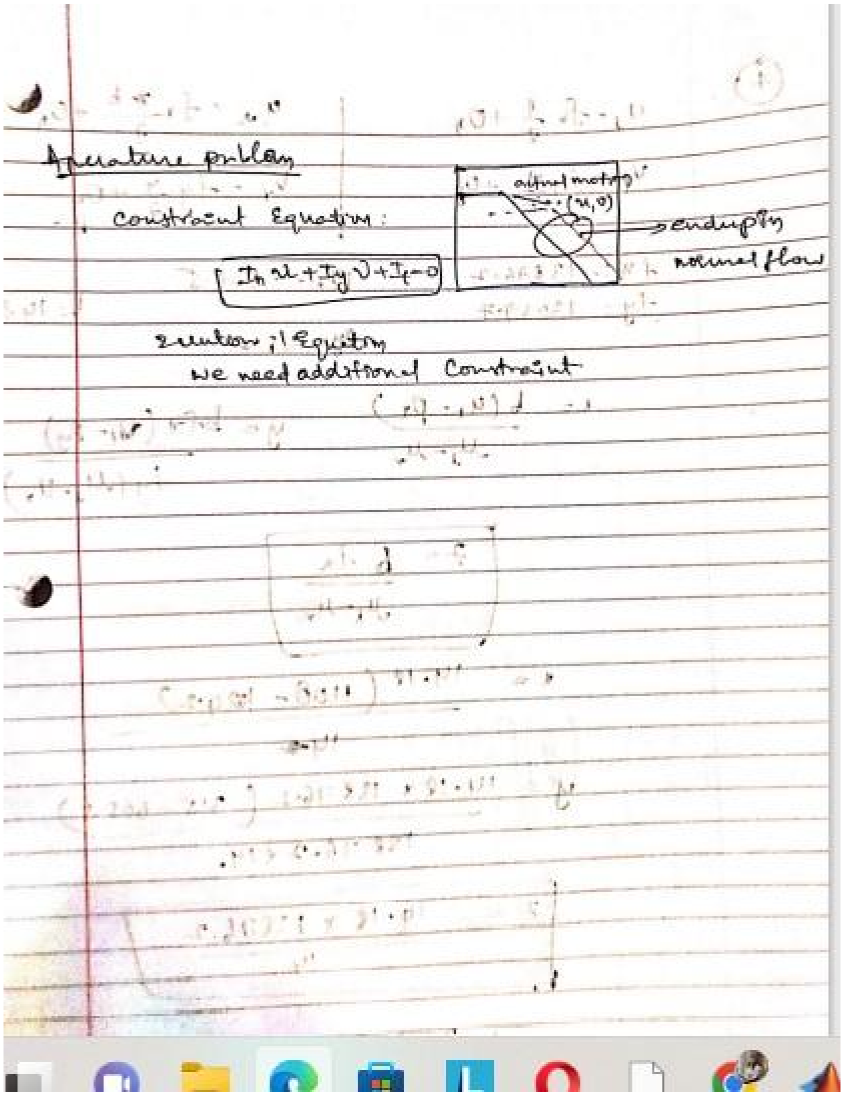
Magnitude of normal flow:
distance of origin from the constraint line:

$$|u_n| = \frac{|I_t|}{\sqrt{I_x^2 + I_y^2}}$$

$$\boxed{u_n = \frac{|I_t|}{(I_x^2 + I_y^2)} (I_x, I_y)}$$

Parallel flow

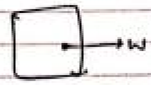
we cannot determine u_p the optical flow component parallel to the constraint line.



3. Derive the procedure for performing Lucas-Kanade algorithm for motion tracking when the motion is known to be affine: u(x,y) = a1*x + b1*y + c1; v(x,y) = a2*x + b2*y + c2 (the numbers are subscripts, not power)

3

Lucas-Kanade Solution



for all points $(k,l) \in W$:

$$I_x(k,l)u + I_y(k,l)v + I_t(k,l) = 0$$

matrix form:

$$\underbrace{\begin{bmatrix} I_x(1,1) & I_y(1,1) \\ I_x(k,l) & I_y(k,l) \\ \vdots & \\ I_x(n,n) & I_y(n,n) \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{u} = \underbrace{\begin{bmatrix} I_t(1,1) \\ I_t(k,l) \\ \vdots \\ I_t(n,n) \end{bmatrix}}_{B}$$

Solve linear system:

$$Au = B$$

$$A^TAu = A^TB$$

In matrix form:

$$\underbrace{\begin{bmatrix} \sum_w I_x I_x & \sum_w I_x I_y \\ \sum_w I_x I_y & \sum_w I_y I_y \end{bmatrix}}_{A^TA} \underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{u} = \underbrace{\begin{bmatrix} -\sum_w I_x I_t \\ -\sum_w I_y I_t \end{bmatrix}}_{A^TB}$$

$$u = (A^TA)^{-1}A^TB$$

$\boxed{Au = B}$ $\boxed{A^TAu = A^TB}$

4. Fix a marker on a wall or a flat vertical surface. From a distance D, keeping the camera stationed static (not handheld and mounted on a tripod or placed on a flat surface), capture an image such that the marker is registered. Then translate the camera by T units along the axis parallel to the ground (horizontal) and then capture another image, with the marker being registered. Compute D using disparity based depth estimation in stereo-vision theory. *(Note: you can pick any value for D and T. Keep in mind that T cannot be large as the marker may get out of view. Of course this depends on D)*

$u_l = f_x \frac{x}{z} + O_x$

$v_l = f_y \frac{y}{z} + O_y$

$u_r = f_x \frac{x-b}{z} + O_x$

$V_r = f_y \frac{y}{z} + O_y$

f_x :- 13876.2

f_y : 13029.7

P_x : 605.5

P_y 315.2

$b = 14.18\,cm$

$$x = \frac{b(u_l - P_x)}{u_l - u_r}$$

$$y = \frac{b f_x (v_l - P_y)}{f_y (u_l - u_r)}$$

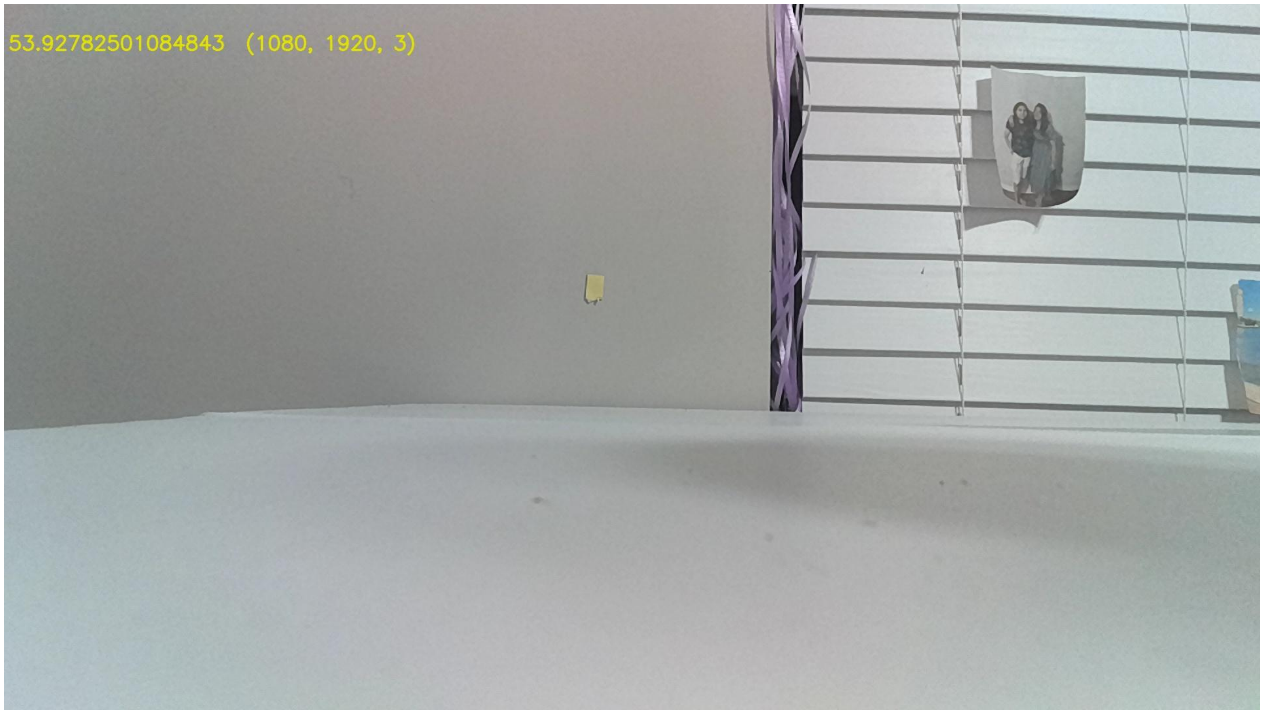
$$\boxed{z = \frac{b f_x}{u_l - u_r}}$$

$$x = \frac{14.18(1108 - 1024.2)}{14}$$

$$y = \frac{14.18 \times 13876.2(315 - 605.5)}{13876.2 \times 14.}$$

$$\boxed{z = \frac{14.18 \times 13876.2}{14}}$$

$\therefore z = 14054.60$



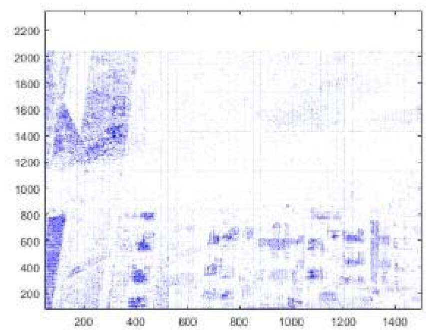
PART B: MATLAB Prototyping

5. For the video (problem 1) you have taken, plot the optical flow vectors on each frame using MATLAB's optical flow codes. (i) treating every previous frame as a reference frame (ii) treating every 11th frame as a reference frame (iii) treating every 31st frame as a reference frame

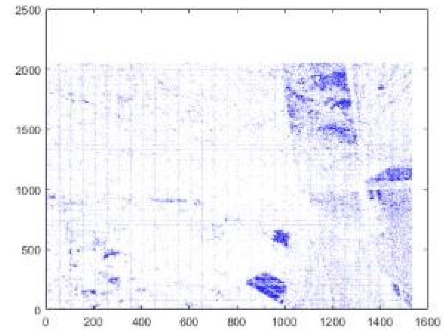
```
vidReader = VideoReader('cv_assign3_vdo.mp4');

opticFlow = opticalFlowLK('NoiseThreshold',0.009);
h = figure;
movegui(h);
hViewPanel = uipanel(h,'Position',[0 0 1 1],'Title','Plot of Op
hPlot = axes(hViewPanel);

while hasFrame(vidReader)
    frameRGB = readFrame(vidReader);
    frameGray = im2gray(frameRGB);
    flow = estimateFlow(opticFlow,frameGray);
    imshow(frameRGB)
    hold on
    plot(flow,'DecimationFactor',[5 5],'ScaleFactor',10,'Parent
    hold off
    pause(10^-3)
end
```

Plot of Optical Flow Vectors

```
clear;
close all;

tic
%% Read The Input
%v = imread("C:\Users\Sahithi Kolla\Downloads\cv2_7_data\What
%A=squeeze(v);

a1 = imread("C:\Users\Sahithi Kolla\Downloads\cv2_7_data\What
a2 = imread("C:\Users\Sahithi Kolla\Downloads\cv2_7_data\What
a1= rgb2gray(a1);
a2 = rgb2gray(a2);

%
%figure;
%imshow(imresize(uint8(A(:,:,1)),0.5));
%hold on
%for i2 = 1:5:size(A,3)-5

% i2=20;
%a1 = imresize(A(:,:,i2),0.5);
%a2 = imresize(A(:,:,i2+5),0.5);
```



```
%% Creating the u and v matrices
u = zeros(b(1)-1,b(2)-1);
v = zeros(b(1)-1,b(2)-1);

    for i = 2:b(1)-3
        for j = 2:b(2)-3
            x = fx(i-1:i+1,j-1:j+1);
            x = x(:);
            y = fy(i-1:i+1,j-1:j+1);
            y = y(:);
            t = -ft(i-1:i+1,j-1:j+1);
            t = t(:);
            Amat = [x(:),y(:)];
            bvec = t(:);
            if det((Amat')*Amat) ~=0
                U = ((Amat')*Amat)\(Amat'*bvec);
                u(i,j) = U(1);
                v(i,j) = U(2);
            end

        end
    end
```



6. Run the feature-based matching object detection on the images from problem (1). *Tutorial for feature-based matching object detection is available here:* *https://www.mathworks.com/help/vision/ug/object-detection-in-a-cluttered-scene-using-point-feature-matching.html*

```
RGB = imread('crop_frame217.jpg');
RGB2 = imread('Frame217.jpg');

boxImage = rgb2gray(RGB);
figure;
imshow(boxImage);
title('Image');


sceneImage = rgb2gray(RGB2);
figure;
imshow(sceneImage);
title('Image of a Cluttered Scene');



boxPoints = detectSURFFeatures(boxImage);
scenePoints = detectSURFFeatures(sceneImage);


figure;
imshow(boxImage);
title('100 Strongest Feature Points from Box Image');
hold on;
```

```
figure;
imshow(boxImage);
title('100 Strongest Feature Points from Box Image');
hold on;
plot(selectStrongest(boxPoints, 100));


figure;
imshow(sceneImage);
title('300 Strongest Feature Points from Scene Image');
hold on;
plot(selectStrongest(scenePoints, 300));


[boxFeatures, boxPoints] = extractFeatures(boxImage, boxPoint
[sceneFeatures, scenePoints] = extractFeatures(sceneImage, sc

boxPairs = matchFeatures(boxFeatures, sceneFeatures);

matchedBoxPoints = boxPoints(boxPairs(:, 1), :);
matchedScenePoints = scenePoints(boxPairs(:, 2), :);
figure;
showMatchedFeatures(boxImage, sceneImage, matchedBoxPoints, .
```

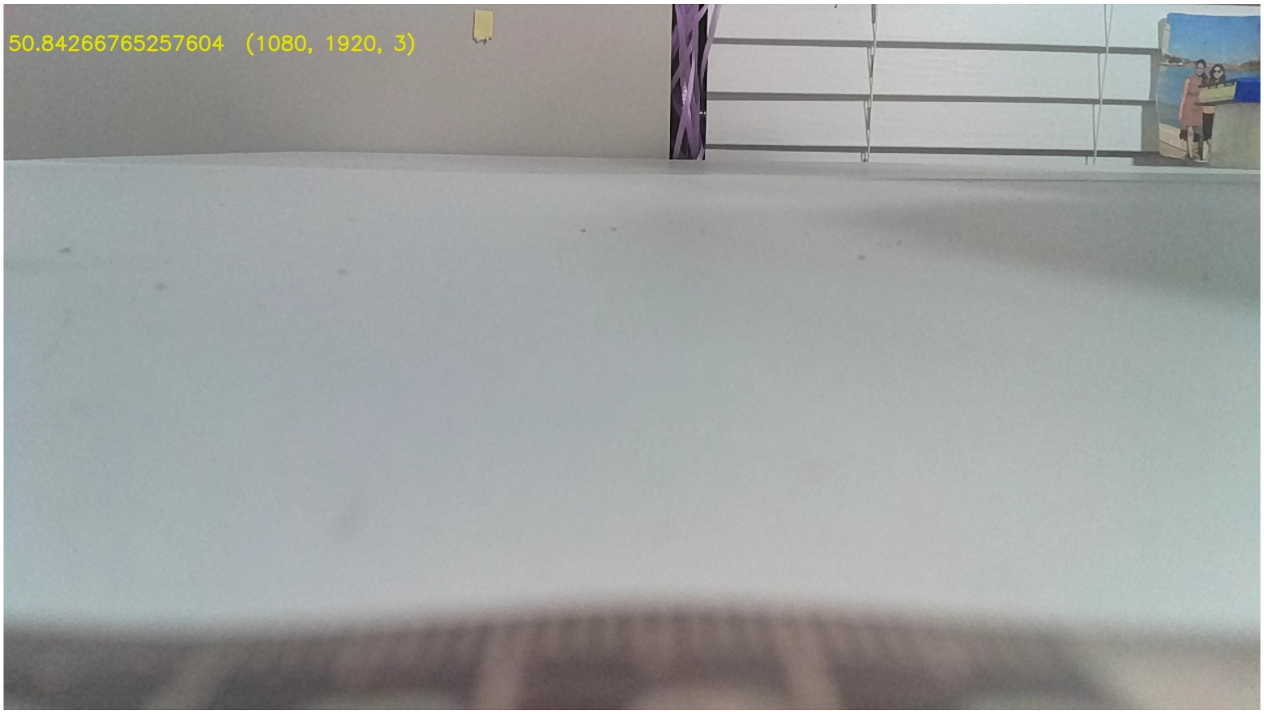
```
[tform, inlierIdx] = ...
    estimateGeometricTransform2D(matchedBoxPoints, matchedSce
inlierBoxPoints   = matchedBoxPoints(inlierIdx, :);
inlierScenePoints = matchedScenePoints(inlierIdx, :);


figure;
showMatchedFeatures(boxImage, sceneImage, inlierBoxPoints, ..
    inlierScenePoints, 'montage');
title('Matched Points (Inliers Only)');


boxPolygon = [1, 1;...                           % top-left
        size(boxImage, 2), 1;...                 % top-right
        size(boxImage, 2), size(boxImage, 1);... % bottom-rig
        1, size(boxImage, 1);...                 % bottom-lef
        1, 1];                   % top-left again to close th


newBoxPolygon = transformPointsForward(tform, boxPolygon);

figure;
imshow(sceneImage);
```

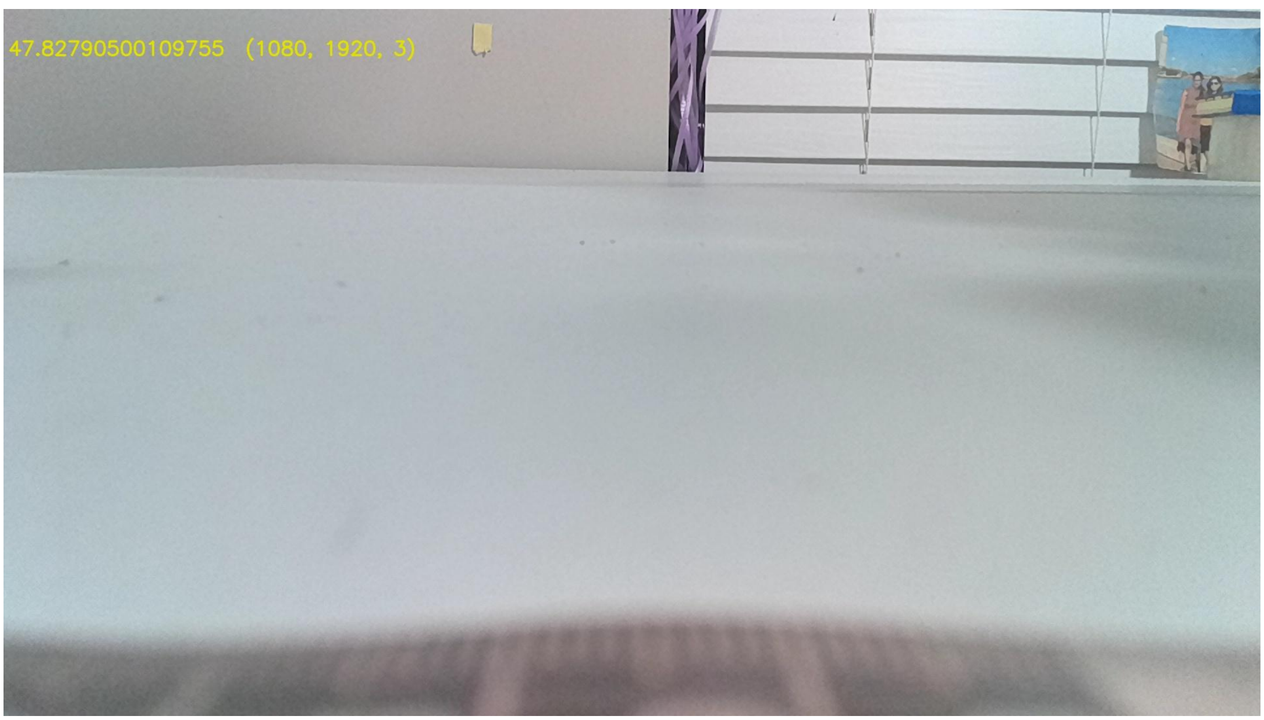
7. Refer to the Bag of Features example MATLAB source code provided in the classroom's classwork page. In your homework, pick an object category that would be commonly seen in any household (e.g. cutlery) and pick 5 object types (e.g. for cutlery pick spoon, fork, butter knife, cutting knife, ladle). Present your performance evaluation.

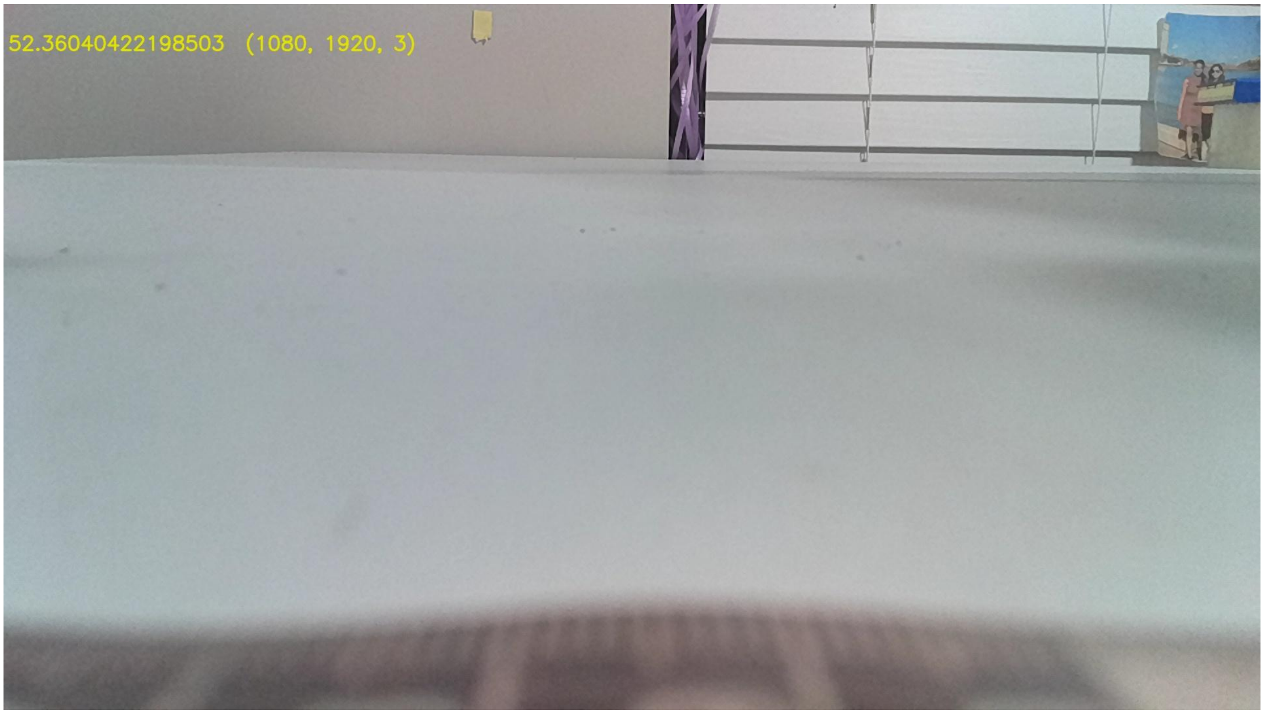
```
tbl = countEachLabel(imds)
figure
montage(imds.Files(1:16:end))

[trainingSet, validationSet] = splitEachLabel(imds, 0.6, 'ra

bag = bagOfFeatures(trainingSet);

img = readimage(imds, 1);
featureVector = encode(bag, img);

% Plot the histogram of visual word occurrences
figure
bar(featureVector)
title('Visual word occurrences')
xlabel('Visual word index')
ylabel('Frequency of occurrence')


categoryClassifier = trainImageCategoryClassifier(trainingSe


confMatrix = evaluate(categoryClassifier, trainingSet);

confMatrix = evaluate(categoryClassifier, validationSet);
```

```
title('Visual word occurrences')
xlabel('Visual word index')
ylabel('Frequency of occurrence')


categoryClassifier = trainImageCategoryClassifier(trainingSet


confMatrix = evaluate(categoryClassifier, trainingSet);

confMatrix = evaluate(categoryClassifier, validationSet);

mean(diag(confMatrix))
img = imread(fullfile('MerchData','MathWorks Cap','Hat_0.jpg'
figure
imshow(img)


[labelIdx, scores] = predict(categoryClassifier, img);

% Display the string label
categoryClassifier.Labels(labelIdx)
```

8. Repeat the image capture experiment from problem (4), however, now also rotate (along the ground plane) the camera 2 (right camera) towards camera 1 position, after translation by T. Make sure the marker is within view. Note down the rotation angle. Run the tutorial provided for uncalibrated stereo rectification in here: https://www.mathworks.com/help/vision/ug/uncalibrated-stereo-image-rectification.html Exercise this tutorial for the image pairs you have captured. You can make assumptions as necessary, however, justify them in your answers/description. *(Note: you can print out protractors from any online source and place your cameras on that when running experiments: http://www.ossmann.com/protractor/conventional-protractor.pdf).*


50.84266765257604 (1080, 1920, 3)


47.82790500109755 (1080, 1920, 3)



PART C: Application development

9. Implement a real-time face tracking application that will recognize multiple faces within a scene and track the lip movements of the person of interest. Validate with at least 3 faces in a scene.

```python
import cv2

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
# eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
smile_cascade = cv2.CascadeClassifier('haarcascade_smile.xml')

cap = cv2.VideoCapture(0)

while True:

    ret, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    for (x, y, w, h) in faces:
        cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)

        roi_gray = gray[y:y + h, x:x + w]
        roi_color = img[y:y + h, x:x + w]

        # eyes = eye_cascade.detectMultiScale(roi_gray)

        smile = smile_cascade.detectMultiScale(roi_gray,
                                               scaleFactor=1.7,
                                               minNeighbors=22,
                                               minSize=(25, 25),
                                               )

        # for (ex, ey, ew, eh) in eyes:
        #     cv2.rectangle(roi_color, (ex, ey), (ex + ew, ey + eh), (0, 255, 0), 2)

        for (ex, ey, ew, eh) in smile:
            cv2.rectangle(roi_color, (ex, ey), (ex + ew, ey + eh), (0, 255, 255), 2)

    cv2.imshow('img', img)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
cv2.destroyAllWindows()
```