# Assignment 3 - Computer Vision (CSc8830)

Instructor: Dr. Ashwin Ashok - Spring 2024

Student name: Reza Mansouri

Repository address:
https://github.com/rezmansouri/csc8830/tree/main/Assignment%203

# Question 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.

In [1]:
```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
```

## Reading the video and getting the frames

In [2]:
```
def get_video_frames(path):
    video = cv.VideoCapture(path)
    ret, frame = video.read()
    frames = []
    while ret:
        frame = cv.cvtColor(frame, cv.COLOR_RGB2GRAY)
        frames.append(frame)
        ret, frame = video.read()
    video.release()
    return np.array(frames)
```

In [3]:
```
frames_dataset = get_video_frames('input/video.mp4')
```

## Showing 6 consecutive frames

In [4]:
```
n_frames = len(frames_dataset)
fig, axes = plt.subplots(1, 6, figsize=(30, 10))
step = n_frames // 6
for i in range(0, n_frames, step):
    axes[i//step].axis('off')
    axes[i//step].imshow(frames_dataset[i], cmap='gray')
```

## Choosing a target frame and an object

In [5]:
```
target_frame_ix = 100
target_frame = frames_dataset[target_frame_ix]
plt.figure(figsize=(10, 8))
plt.axis('off')
plt.imshow(target_frame, cmap='gray')
plt.show()
```

## Cropping the object out

In [6]:
```
object = target_frame[325:610, 293:385]
plt.figure(figsize=(5, 3))
plt.axis('off')
plt.imshow(object, cmap='gray')
plt.show()
```

## Getting 10 random frames

In [11]:
```
random_frames_ix = np.random.choice(np.arange(n_frames), size=10, replace=F
print(random_frames_ix)
```

```
[260 246  62  84 241 243 155 129 267 164]
```

In [10]:
```
for i in random_frames_ix:
    image = Image.fromarray(frames_dataset[i])
    image.save(f'5_frame_{i}.jpg')
```

## Sliding a window of the same size as the object on each frame and computing $SSD$

In [8]:
```
differences = []
coordinates = []
for ix in random_frames_ix:
    frame = frames_dataset[ix]
    ssds = []
    coords = []
    for i in range(0, frame.shape[0] - object.shape[0], 30):
        for j in range(0, frame.shape[1] - object.shape[1], 30):
            candidate = frame[i:i+object.shape[0], j:j+object.shape[1]]
            ssd = np.sum(np.power(candidate-object, 2))
            ssds.append(ssd)
            coords.append((i, j))
    differences.append(ssds)
    coordinates.append(coords)
```

## Finding the best match $min(SSD)$ in each frame

In [9]:
```
for i, ix in enumerate(random_frames_ix):
    minimun_diff_ix = np.argmin(differences[i])
```

```
    coords = coordinates[i][minimun_diff_ix]
    overlay = np.zeros(frames_dataset.shape[1:])
    overlay[coords[0]:coords[0]+object.shape[0], coords[1]:coords[1]+object
    plt.title(f'SSD: {differences[i][minimun_diff_ix]}')
    plt.axis('off')
    plt.imshow(frames_dataset[ix], cmap='gray')
    plt.imshow(overlay, cmap='magma', alpha=.4)
    plt.show()
```

SSD: 1795703

SSD: 2041999

SSD: 2240904

SSD: 2160143

SSD: 2148029

SSD: 2022475

SSD: 2276699

SSD: 2157429

SSD: 1984929

SSD: 2084172

# Question 2

## Part a

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.

Assignment 3 - Question 2 (a) - Reza Mansouri - 06278647

We have 2 assumptions, which will make the problem constrained so that it will be solvable:

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

meaning for point $(x,y)$ in the image at frame $t$, the intensity (brightness / pixel value) does not change when it is moved by $(\delta x, \delta y)$ later at frame $t+\delta t$.

Taylor series expansion says:

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

if $\delta x$ is small, $f(x+\delta x) = f(x) + \frac{\partial f}{\partial x}\delta x$

thus, our next assumption is:

(II) $\delta x, \delta y$ and $\delta t$ are small

$$\Rightarrow I(x+\delta x, y+\delta y, t+\delta t) = I(x,y,t) + \underbrace{\frac{\partial I}{\partial x}}_{I_x}\delta x + \underbrace{\frac{\partial I}{\partial y}}_{I_y}\delta y + \underbrace{\frac{\partial I}{\partial t}}_{I_t}\delta t$$

subtracting (I) from (II): $I_x \delta x + I_y \delta y + I_t \delta t = 0$

dividing by $\delta t$ and taking the limit as $\delta t \to 0$:

$$I_x \frac{\partial x}{\partial t} + I_y \frac{\partial y}{\partial t} + I_t = 0 \rightarrow \boxed{I_x u + I_y v + I_t = 0} \quad (\ast)$$

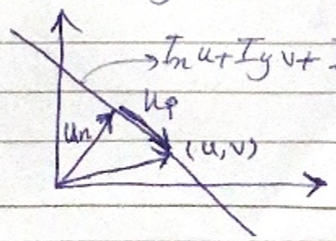
$\rightarrow (u, v)$

$\rightarrow \left(\frac{\partial x}{\partial t}, \frac{\partial y}{\partial t}\right)$: velocity of the point, i.e. the optical flow

The $(\ast)$ equation is a straight line, constraining $(u,v)$:

We only know $\vec{u_n}$.
$\vec{u_p}$ can be anything
$\vec{u_n}$ is our estimate.

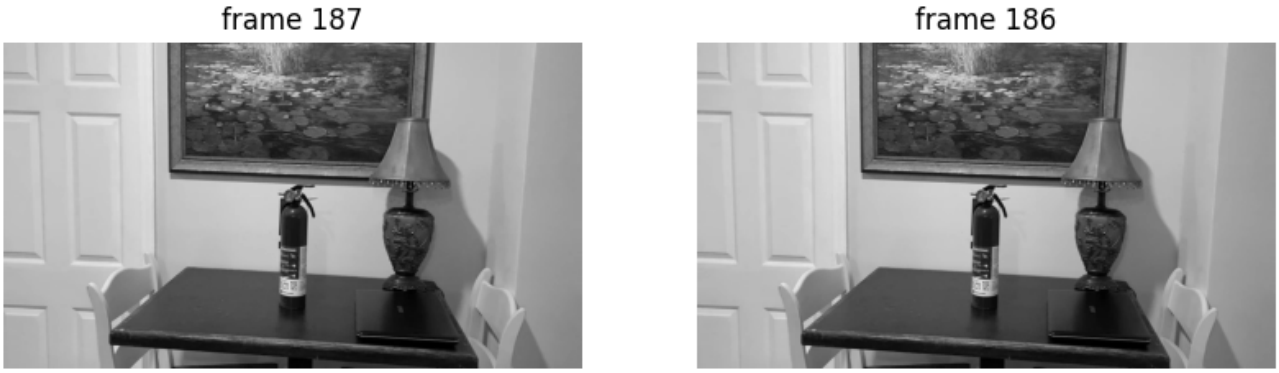
Kian




$\vec{u_n}$, the normal optical flow vector can be computed as:

magnitude: $\frac{|I_t|}{\sqrt{I_x^2 + I_y^2}}$ direction: $\frac{(I_x, I_y)}{\sqrt{I_x^2 + I_y^2}}$

In [10]:
```
random_frame_ix = np.random.randint(low=0, high=n_frames-1)
frame_1 = frames_dataset[random_frame_ix]
frame_2 = frames_dataset[random_frame_ix+1]
plt.figure(figsize=(10, 8))
plt.subplot(1, 2, 1)
plt.axis('off')
plt.title(f'frame {random_frame_ix + 1}')
plt.imshow(frame_2, cmap='gray')
plt.subplot(1, 2, 2)
plt.axis('off')
plt.title(f'frame {random_frame_ix}')
plt.imshow(frame_1, cmap='gray')
plt.show()
```




In [11]:
```
frame_block = np.concatenate([np.expand_dims(frame_1, 2), np.expand_dims(fr
```

## Step 1: finding $I_x$, $I_y$, and $I_t$ using finite differences

In [12]:
```
Iy = np.mean(np.diff(frame_block, axis=0), axis=2)
Ix = np.mean(np.diff(frame_block, axis=1), axis=2)
It = np.diff(frame_block, axis=2).squeeze()
```

In [13]:
```
Iy = Iy[:, :-1]
Ix = Ix[:-1, :]
It = It[:-1, :-1]
```

## Step 2: computing magnitued and direction of optical flow $(u, v)$

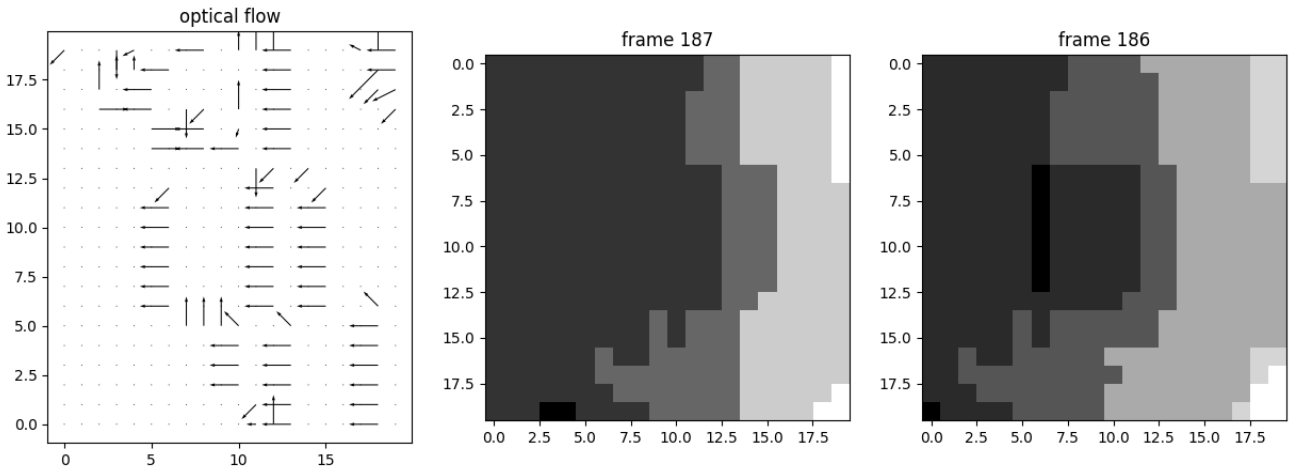
In [14]:
```
denominator = np.sqrt(Ix ** 2 + Iy ** 2)
nominator = np.abs(It)
un_magnitude = np.zeros(nominator.shape)
defined_indices = denominator!=0
un_magnitude[defined_indices] = nominator[defined_indices] / denominator[de
```

In [15]:
```
un_direction = np.arctan2(Iy, Ix)
```

In [42]:
```
u = -un_magnitude * np.cos(un_direction)
v = -un_magnitude * np.sin(un_direction)
```

## Plotting the results for a 10 by 10 patch

In [54]:
```
k, l = 130, 130
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
axs[0].set_title('optical flow')
axs[0].quiver(u[k:k+20, l:l+20], v[k:k+20, l:l+20], scale=25)
plt.gca().set_aspect('equal', adjustable='box')
axs[1].set_title(f'frame {random_frame_ix + 1}')
axs[1].imshow(frame_2[k:k+20, l:l+20], cmap='gray')
axs[2].set_title(f'frame {random_frame_ix}')
axs[2].imshow(frame_1[k:k+20, l:l+20], cmap='gray')
plt.show()
```




# Part b

Derive the procedure for performing Lucas-Kanade algorithm for motion tracking when the motion is known to be affine:

$$u(x, y) = a_1 * x + b_1 * y + c_1, v(x, y) = a_2 * x + b_2 * y + c_2$$

Assignment 3 – Question 2 (b) – Reza Mansouri – 002784647

When the motion is affine, i.e., representable by translation, rotation, etc., Lucas-Kanade algorithm works effectively for motion tracking. As the optical flow estimation is an underconstrained problem, the Lucas-Kanade algorithm assumes that the optical flow / motion field within a small window is the same for all the pixels and further uses this to solve the optical flow estimation problem.

* for all points $(i,j) \in W$:

$$I_x(i,j)u + I_y(i,j)v + I_t(i,j) = 0$$

* for a window size of m by m:

$$\begin{bmatrix} I_x(1,1) & I_y(1,1) \\ \vdots & \vdots \\ I_x(m,m) & I_y(m,m) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} I_t(1,1) \\ \vdots \\ I_t(m,m) \end{bmatrix}$$

→ general Lucas-Kanade equation

known $(m^2 \times 2)$ unknown $(2 \times 1)$ known $(m^2 \times 1)$

* As the motion is assumed to be affine we can write the general form above, for each pixel $(i,j)$ in W as:

$$\begin{bmatrix} I_x(1,1) & I_y(1,1) & 1 \\ I_x(i,j) & I_y(i,j) & 1 \\ \vdots & \vdots & \vdots \\ I_x(m,m) & I_y(m,m) & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix} = \begin{bmatrix} I_t(1,1) \\ \vdots \\ I_t(m,m) \end{bmatrix}$$

→ same for $\begin{bmatrix} a_2 \\ b_2 \\ c_2 \end{bmatrix}$

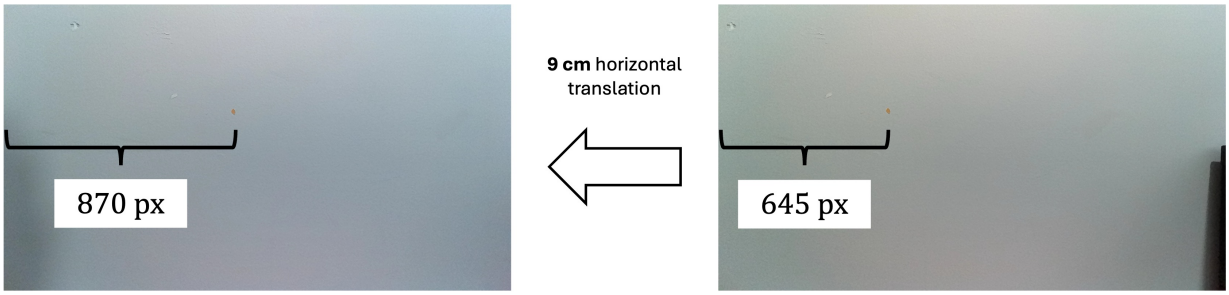
$$\begin{bmatrix} I_x(1,1) & I_y(1,1) & 1 \\ I_x(i,j) & I_y(i,j) & 1 \\ \vdots & \vdots & \vdots \\ I_x(m,m) & I_y(m,m) & 1 \end{bmatrix} \begin{bmatrix} a_2 \\ b_2 \\ c_2 \end{bmatrix} = \begin{bmatrix} I_t(1,1) \\ \vdots \\ I_t(m,m) \end{bmatrix}$$

These systems of equations are solvable via least squares. once $a_1, b_1, c_1, a_2, b_2$ and $c_2$ are obtained, $u(x,y)$ and $v(x,y)$ are computable.

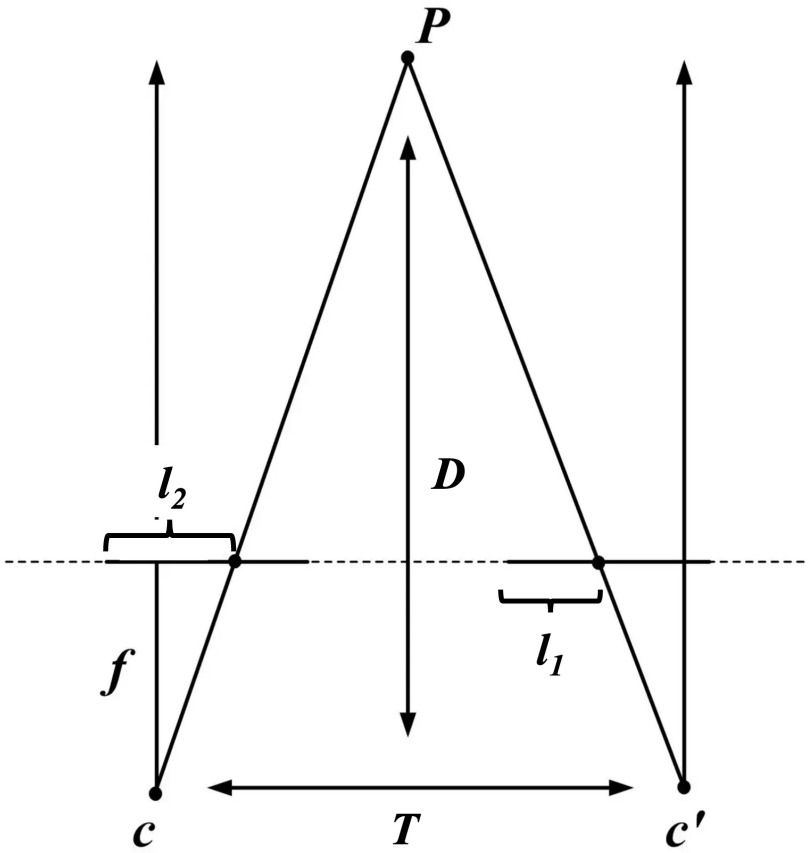
# Question 3

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)

As the solution requires the focal length, I used the oak-d camera, because I obtained its focal length in assignment 1 during calibration ($f_x \approx f_y = f$). Here is my experiment setup. The marker is the small orange label on the wall.




Depth estimation via disparity is finding $D$ by knowing $f$ (focal length), $T$ (the translation magnitude), $l_1$, and $l_2$ which are the distances of the projection of the point $P$ from the left border of the image planes.




Here are my knowns:

$T = 9$ cm
$f = 1430$ px (from assignment 1)
$l_1 = 645$ px
$l_2 = 870$ px

And I want to estimate $D$ which has the ground truth of 60 cm.

$$D = \frac{f \times T}{l_2 - l_1}$$

$$D = \frac{1430 \times 9}{870 - 645} = \frac{12870}{225} = 57.2 \approx 60\ cm$$

# Question 4

## 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

I copied the code and output from my `.mlx` file for this question for showing it here. Please find the live script in the attachments. The plots are for some single frames. The full version is available via running the `.mlx` file.

Reading the video:

```
vidReader = VideoReader('./input/video.mp4','CurrentTime',0);
```

Specifying the optical flow estimation method as `opticalFlowHS` .

```
opticFlow = opticalFlowHS
```

```
>>> opticFlow =
    opticalFlowHS with properties:

              Smoothness: 1
           MaxIteration: 10
     VelocityDifference: 0
```

## 1. Plotting the optical flow vectors when treating every previous frame as a reference frame.

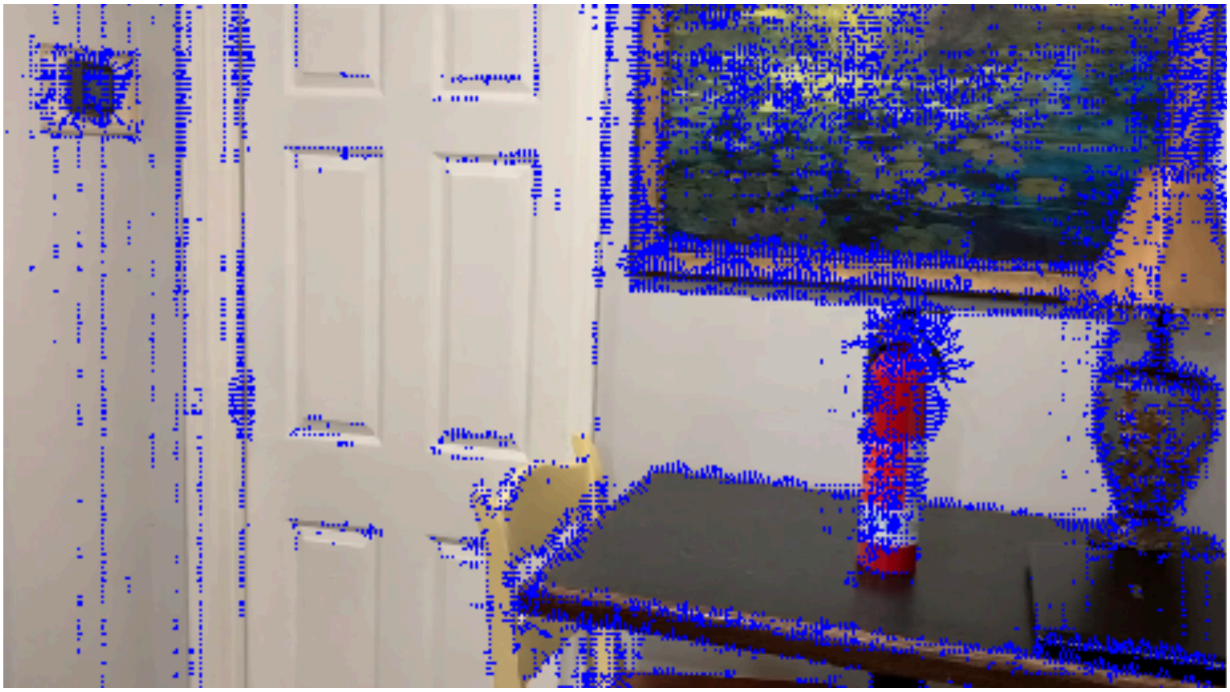
```
h = figure;
movegui(h);
hViewPanel = uipanel(h,'Position',[0 0 1 1],'Title','Plot of
Optical Flow Vectors');
hPlot = axes(hViewPanel);

while hasFrame(vidReader)
    frameRGB = readFrame(vidReader);
    frameGray = im2gray(frameRGB);
    flow = estimateFlow(opticFlow,frameGray);
    imshow(frameRGB)
```

```
    hold on
    plot(flow,'DecimationFactor',[5
5],'ScaleFactor',60,'Parent',hPlot);
    hold off
    pause(10^-3)
end
opticFlow.reset();
vidReader = VideoReader('./input/video.mp4','CurrentTime',0);
```

## 2. Plotting the optical flow vectors when treating 11th frame as a reference frame.

```
h = figure;
movegui(h);
hViewPanel = uipanel(h,'Position',[0 0 1 1],'Title','Plot of
Optical Flow Vectors');
hPlot = axes(hViewPanel);
frameCount = 0;

while hasFrame(vidReader)
    frameCount = frameCount + 1;
    if mod(frameCount, 11) == 0
        frameRGB = readFrame(vidReader);
        frameGray = im2gray(frameRGB);
        flow = estimateFlow(opticFlow,frameGray);
        imshow(frameRGB)
        hold on
        plot(flow,'DecimationFactor',[5
5],'ScaleFactor',60,'Parent',hPlot);
        hold off
```

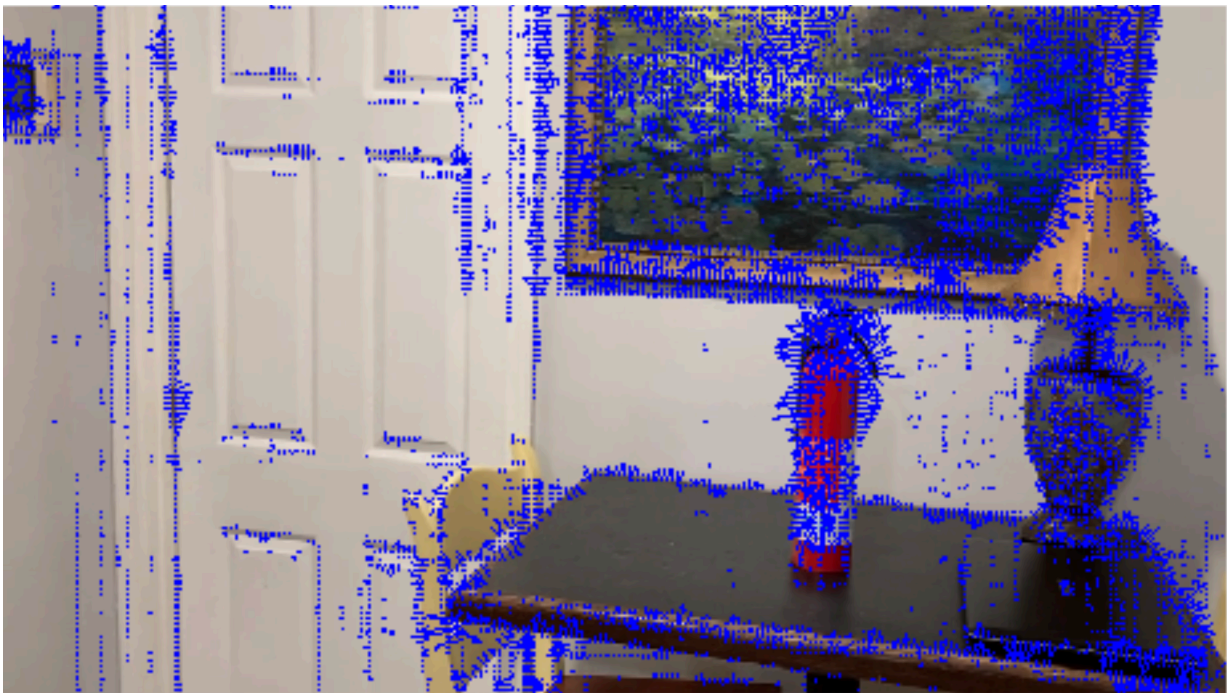
```
        pause(10^-3)
    end
end
opticFlow.reset();
vidReader = VideoReader('./input/video.mp4','CurrentTime',0);
```



### 3. Plotting the optical flow vectors when treating 31st frame as a reference frame.

```
h = figure;
movegui(h);
hViewPanel = uipanel(h,'Position',[0 0 1 1],'Title','Plot of
Optical Flow Vectors');
hPlot = axes(hViewPanel);
frameCount = 0;

while hasFrame(vidReader)
    frameCount = frameCount + 1;
    if mod(frameCount, 31) == 0
        frameRGB = readFrame(vidReader);
        frameGray = im2gray(frameRGB);
        flow = estimateFlow(opticFlow,frameGray);
        imshow(frameRGB)
        hold on
        plot(flow,'DecimationFactor',[5
5],'ScaleFactor',60,'Parent',hPlot);
        hold off
        pause(10^-3)
    end
end
```

```
opticFlow.reset();
vidReader = VideoReader('./input/video.mp4','CurrentTime',0);
```



# Question 5

Run the feature-based matching object detection on the images from problem (1). MATLAB (not mandatory for this problem).

I copied the code and output from my `.mlx` file for this question for showing it here. Please find the live script in the attachments.

## Step 1: Read Images

Read the reference image containing the object of interest.

```
boxImage = imread('./input/5_object.jpg');
sceneImage = imread('./input/5_frame_15.jpg');
```

## Step 2: Detect Point Features

Detect point features in both images.

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

## Step 3: Extract Feature Descriptors

Extract feature descriptors at the interest points in both images.

```
[boxFeatures, boxPoints] = extractFeatures(boxImage, boxPoints);
[sceneFeatures, scenePoints] = extractFeatures(sceneImage,
scenePoints);
```

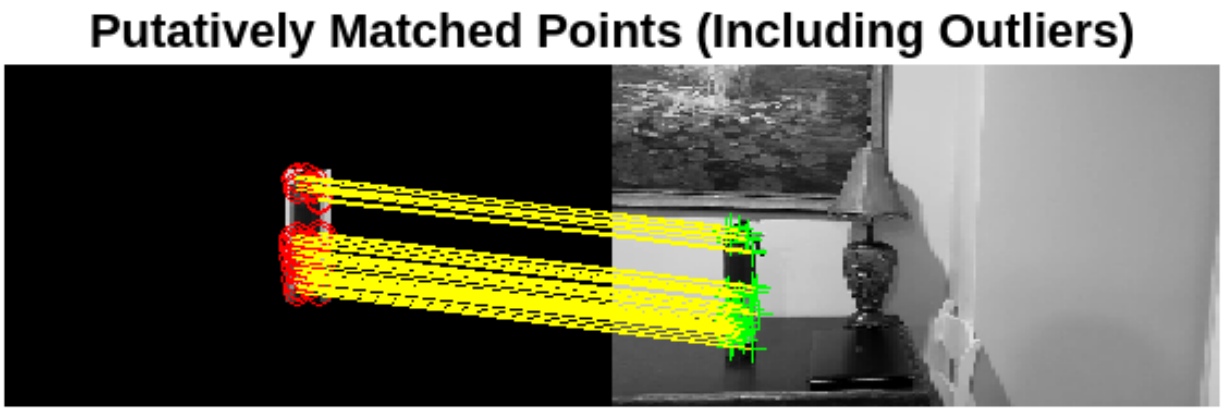
## Step 4: Find Putative Point Matches

Match the features using their descriptors.

```
boxPairs = matchFeatures(boxFeatures, sceneFeatures);
```

Display putatively matched features.

```
matchedBoxPoints = boxPoints(boxPairs(:, 1), :);
matchedScenePoints = scenePoints(boxPairs(:, 2), :);
figure;
showMatchedFeatures(boxImage, sceneImage, matchedBoxPoints, ...
    matchedScenePoints, 'montage');
title('Putatively Matched Points (Including Outliers)');
```




## Step 5: Locate the Object in the Scene Using Putative Matches

```
[tform, inlierIdx] = estgeotform2d(matchedBoxPoints,
matchedScenePoints, 'affine');
inlierBoxPoints   = matchedBoxPoints(inlierIdx, :);
inlierScenePoints = matchedScenePoints(inlierIdx, :);
```

Get the bounding polygon of the reference image.

```
boxPolygon = [1, 1;...                           % top-left
    size(boxImage, 2), 1;...                 % top-right
    size(boxImage, 2), size(boxImage, 1);... % bottom-right
    1, size(boxImage, 1);...                 % bottom-left
    1, 1];                   % top-left again to close the polygon
```

Transform the polygon into the coordinate system of the target image. The transformed polygon indicates the location of the object in the scene.

```
newBoxPolygon = transformPointsForward(tform, boxPolygon);
```

Display the detected object.

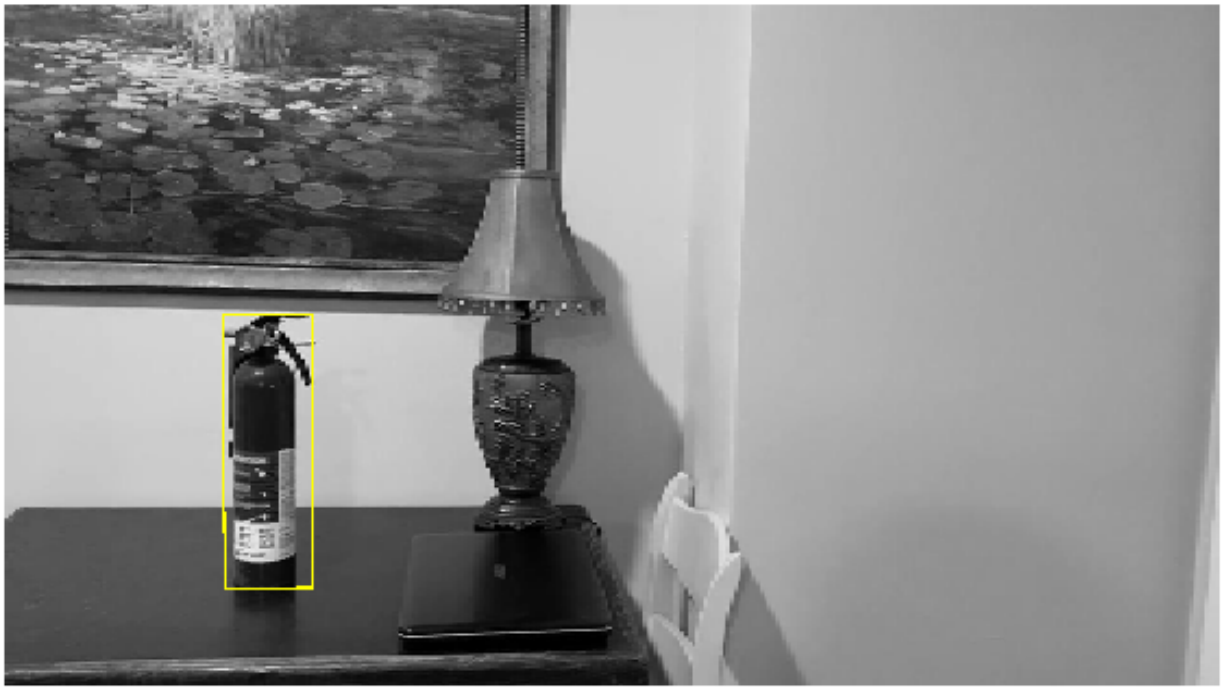
```
figure;
imshow(sceneImage);
hold on;
line(newBoxPolygon(:, 1), newBoxPolygon(:, 2), Color='y');
title('Detected Box');
```


## Step 6: Doing the same for 9 other random frames

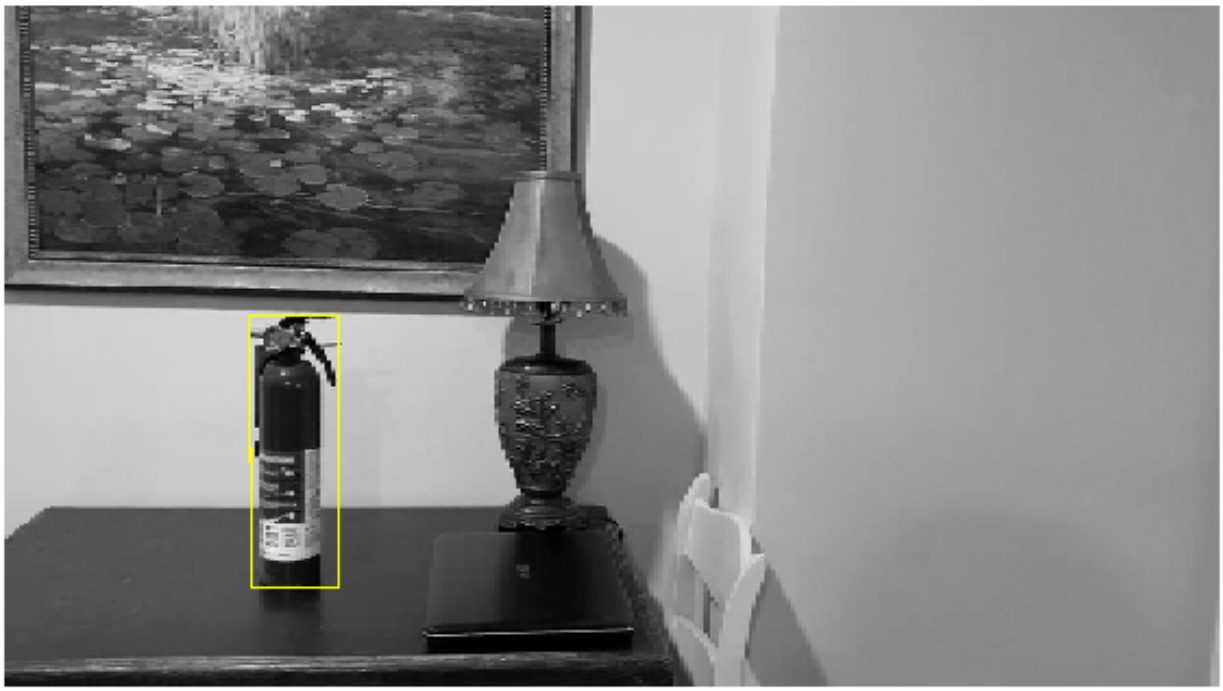
```
paths =
["./input/5_frame_84.jpg","./input/5_frame_85.jpg","./input/5_frame_92

"./input/5_frame_119.jpg","./input/5_frame_163.jpg","./input/5_frame_1

"./input/5_frame_199.jpg","./input/5_frame_267.jpg","./input/5_frame_1
for i=1:numel(paths)
    sceneImage = imread(paths(i));
    scenePoints = detectSURFFeatures(sceneImage);

    [sceneFeatures, scenePoints] = extractFeatures(sceneImage,
scenePoints);
```

```
    boxPairs = matchFeatures(boxFeatures, sceneFeatures);

    matchedBoxPoints = boxPoints(boxPairs(:, 1), :);
    matchedScenePoints = scenePoints(boxPairs(:, 2), :);

    [tform, inlierIdx] = estgeotform2d(matchedBoxPoints,
matchedScenePoints, 'affine');

    boxPolygon = [1, 1;...
    size(boxImage, 2), 1;...
    size(boxImage, 2), size(boxImage, 1);...
    1, size(boxImage, 1);...
    1, 1];
    newBoxPolygon = transformPointsForward(tform, boxPolygon);

    figure;
    imshow(sceneImage);
    hold on;
    line(newBoxPolygon(:, 1), newBoxPolygon(:, 2), Color='y');
    title('Detected Box');
end
```

## Detected Box



## Detected Box



## Detected Box

## Detected Box

Detected Box

Detected Box



## Question 6

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.

I copied the code and output from my `.mlx` file for this question for showing it here. Please find the live script in the attachments.

# Load Data

```
imds =
imageDatastore('input/question_6','IncludeSubfolders',true,'LabelSourc
tbl = countEachLabel(imds)
>>> tbl =

  5x2 table

    Label      Count
    _______    _____

    fork         10
    glass        10
    kettle       10
    knife        10
    spoon        10

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

## Prepare Data for Training

Separate the sets into training and validation data. Pick 60% of images from each set for the training data and the remainder, 40%, for the validation data. Randomize the split to avoid biasing the results.

```
[trainingSet, validationSet] = splitEachLabel(imds, 0.6,
'randomize');
```

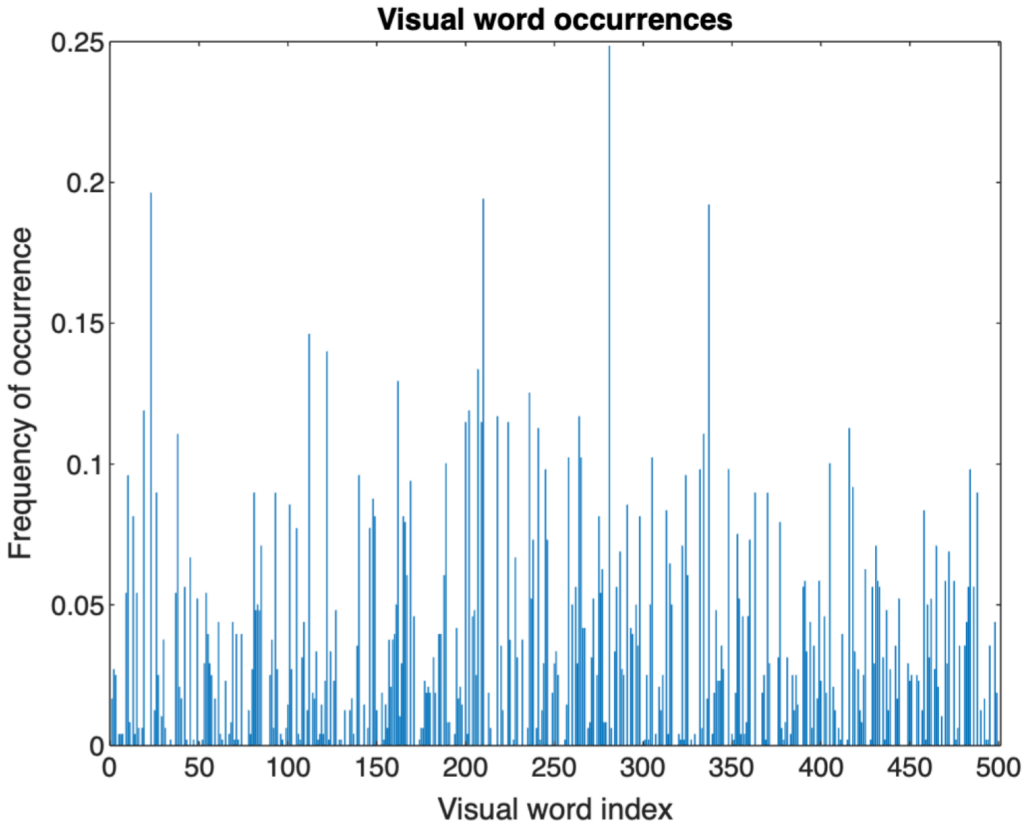
## Create a Visual Vocabulary and Train an Image Category Classifier

```
bag = bagOfFeatures(trainingSet);

>>> Creating Bag-Of-Features.
-----------------------
* Image category 1: fork
* Image category 2: glass
* Image category 3: kettle
* Image category 4: knife
* Image category 5: spoon
* Selecting feature point locations using the Grid method.
* Extracting SURF features from the selected feature point
locations.
** The GridStep is [8 8] and the BlockWidth is [32 64 96 128].

* Extracting features from 30 images...done. Extracted 192000
features.

* Keeping 80 percent of the strongest features from each category.

* Creating a 500 word visual vocabulary.
* Number of levels: 1
* Branching factor: 500
* Number of clustering steps: 1

* [Step 1/1] Clustering vocabulary level 1.
* Number of features          : 153600
* Number of clusters          : 500
* Initializing cluster centers...100.00%.
* Clustering...completed 49/100 iterations (~1.07
seconds/iteration)...converged in 49 iterations.

* Finished creating Bag-Of-Features
```

Additionally, the bagOfFeatures object provides an encode method for counting the visual word occurrences in an image. It produced a histogram that becomes a new and

reduced representation of an image.

```
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(trainingSet,
bag);
>>> Training an image category classifier for 5 categories.
--------------------------------------------------------
* Category 1: fork
* Category 2: glass
* Category 3: kettle
* Category 4: knife
* Category 5: spoon

* Encoding features for 30 images...done.

* Finished training the category classifier. Use evaluate to test
the classifier on a test set.
```

# Evaluate Classifier

## Training set

```
confMatrix = evaluate(categoryClassifier, trainingSet);
>>> Evaluating image category classifier for 5 categories.
-------------------------------------------------------

* Category 1: fork
* Category 2: glass
* Category 3: kettle
* Category 4: knife
* Category 5: spoon

* Evaluating 30 images...done.

* Finished evaluating all the test sets.

* The confusion matrix for this test set is:


                          PREDICTED
KNOWN     | fork   glass   kettle   knife   spoon
-------------------------------------------------
fork      | 1.00   0.00    0.00     0.00    0.00
glass     | 0.00   1.00    0.00     0.00    0.00
kettle    | 0.00   0.00    1.00     0.00    0.00
knife     | 0.00   0.00    0.00     1.00    0.00
spoon     | 0.00   0.00    0.00     0.00    1.00

* Average Accuracy is 1.00.
```

## Validation set

```
confMatrix = evaluate(categoryClassifier, validationSet);
>>> Evaluating image category classifier for 5 categories.
-------------------------------------------------------

* Category 1: fork
* Category 2: glass
* Category 3: kettle
* Category 4: knife
* Category 5: spoon

* Evaluating 20 images...done.

* Finished evaluating all the test sets.

* The confusion matrix for this test set is:
```

```
                          PREDICTED
KNOWN     | fork   glass   kettle   knife   spoon
-----------------------------------------------------
fork      | 0.75   0.00    0.00     0.00    0.25
glass     | 0.00   1.00    0.00     0.00    0.00
kettle    | 0.00   0.00    1.00     0.00    0.00
knife     | 0.00   0.00    0.00     1.00    0.00
spoon     | 0.00   0.75    0.00     0.00    0.25

* Average Accuracy is 0.80.
```

## Compute average accuracy

```
mean(diag(confMatrix))
>>> ans = 0.8000
```

## Classify Object in Images

```
img = imread(fullfile('input/question_6','spoon','IMG_6362
Small.jpeg'));
figure
imshow(img)
```

```
[labelIdx, scores] = predict(categoryClassifier, img);
>>> Encoding images using Bag-Of-Features.
```

```
-------------------------------------
* Encoding an image...done.
```

## Display the string label

```
categoryClassifier.Labels(labelIdx)
>>> ans = 1x1 cell array
    {'spoon'}
```

Another example:

```
img = imread(fullfile('input/question_6','kettle','IMG_6351
Small.jpeg'));
figure
imshow(img)
```

```
[labelIdx, scores] = predict(categoryClassifier, img);
>>> Encoding images using Bag-Of-Features.
-------------------------------------
* Encoding an image...done.
```

## Display the string label

```
categoryClassifier.Labels(labelIdx)
>>> ans = 1x1 cell array
    {'kettle'}
```

# Question 7

Repeat the image capture experiment from problem (3), 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 (MATLAB is mandatory for this exercise). Exercise this tutorial for the image pairs you have captured. You can make assumptions as necessary, however, justify them in your answers/description.

I copied the code and output from my `.mlx` file for this question for showing it here. Please find the live script in the attachments.

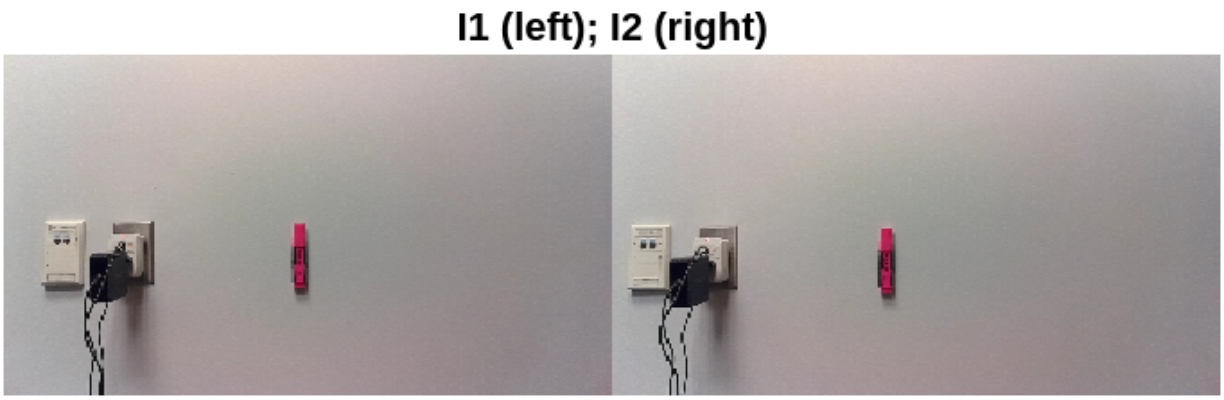
My experiment setup is: $T = 7$ cm, $\theta = 20^\circ$, $D = 53$ cm

## Step 1: Read Stereo Image Pair

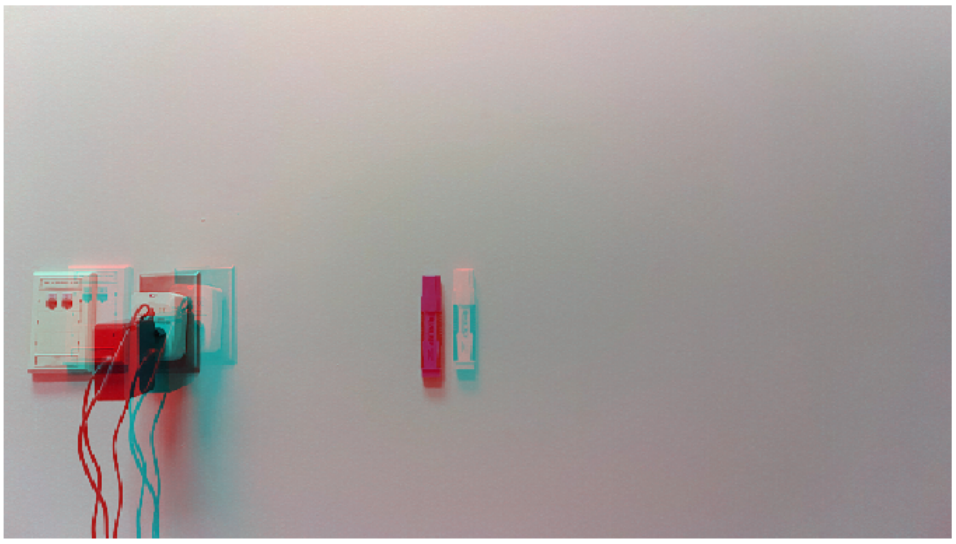
```
I1 = imread("input/question_7/left.jpg");
I2 = imread("input/question_7/right.jpg");

% Convert to grayscale.
I1gray = im2gray(I1);
I2gray = im2gray(I2);
```

Display both images side by side. Then, display a color composite demonstrating the pixel-wise differences between the images.

```
figure
imshowpair(I1,I2,"montage")
title("I1 (left); I2 (right)")
figure
imshow(stereoAnaglyph(I1,I2))
title("Composite Image (Red - Left Image, Cyan - Right Image)")
```



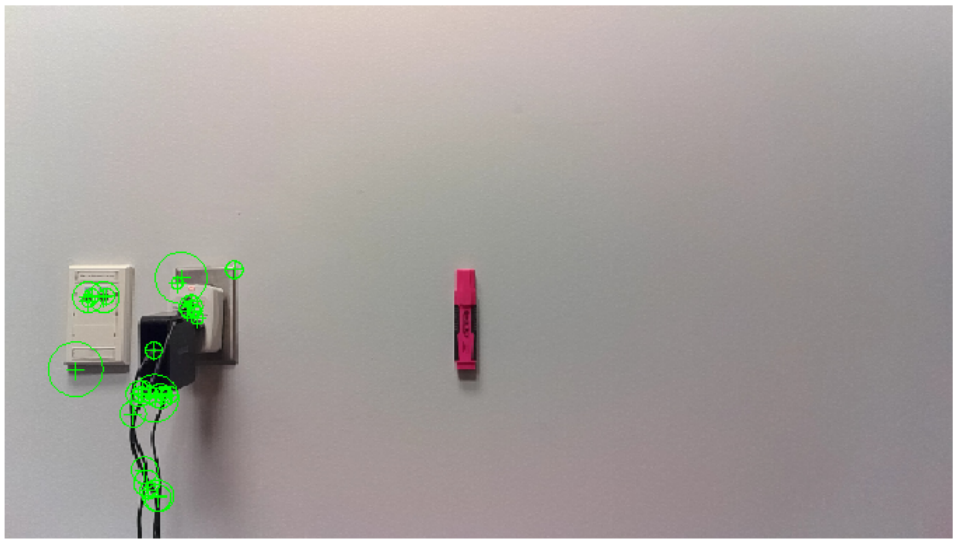
**Composite Image (Red - Left Image, Cyan - Right Image)**



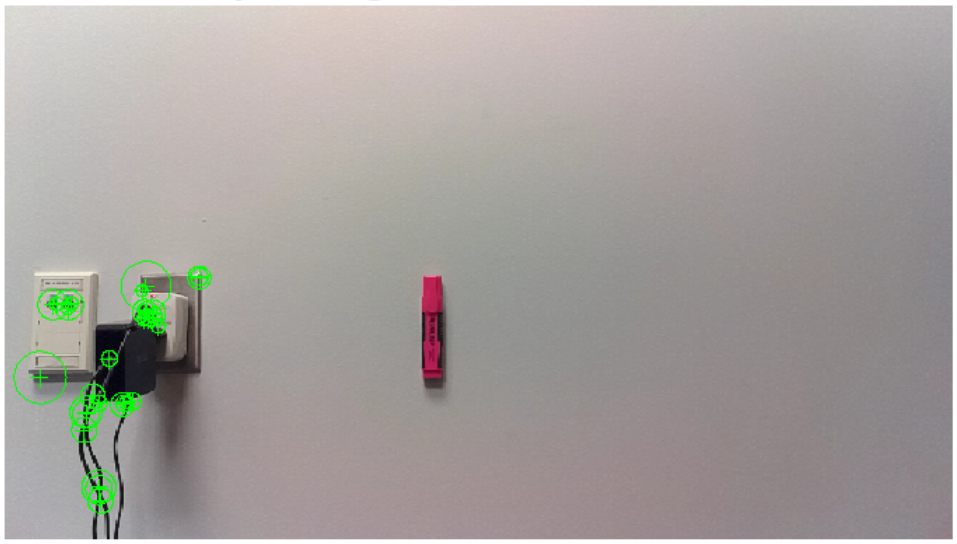
## Step 2: Collect Interest Points from Each Image

```
blobs1 = detectSURFFeatures(I1gray,MetricThreshold=2000);
blobs2 = detectSURFFeatures(I2gray,MetricThreshold=2000);

figure
imshow(I1)
hold on
plot(selectStrongest(blobs1,30))
title("Thirty Strongest SURF Features In I1")
figure
imshow(I2)
hold on
plot(selectStrongest(blobs2,30))
title("Thirty Strongest SURF Features In I2")
```

Thirty Strongest SURF Features In I1

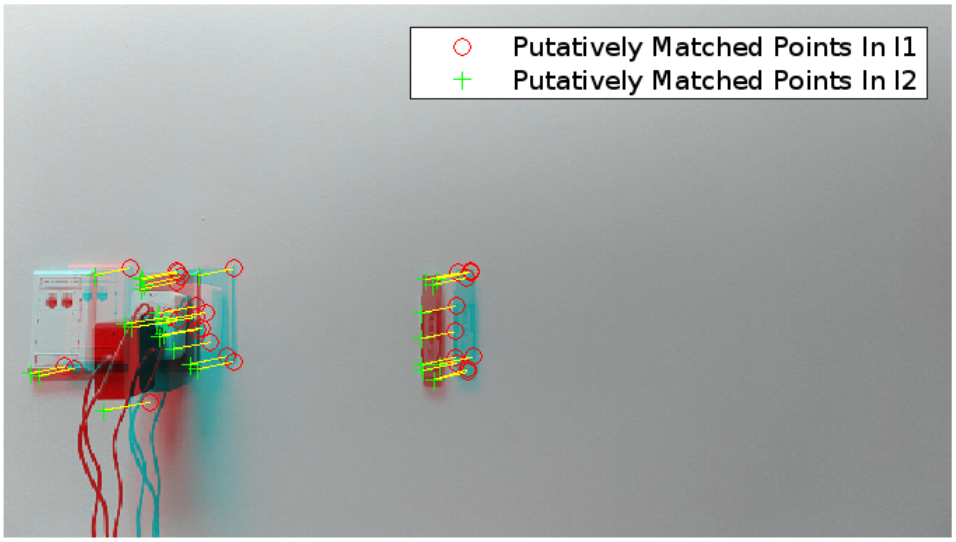


Thirty Strongest SURF Features In I2



## Step 3: Find Putative Point Correspondences

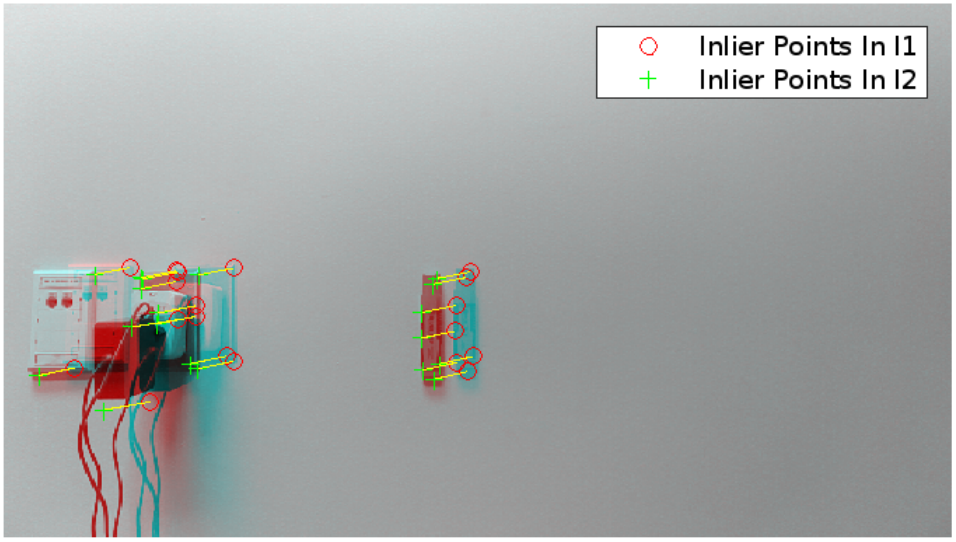
```
[features1,validBlobs1] = extractFeatures(I1gray,blobs1);
[features2,validBlobs2] = extractFeatures(I2gray,blobs2);

indexPairs = matchFeatures(features1,features2,Metric="SAD", ...
  MatchThreshold=5);

matchedPoints1 = validBlobs1(indexPairs(:,1),:);
matchedPoints2 = validBlobs2(indexPairs(:,2),:);

figure
showMatchedFeatures(I1, I2, matchedPoints1, matchedPoints2)
legend("Putatively Matched Points In I1","Putatively Matched
Points In I2")
```




## Step 4: Remove Outliers Using Epipolar Constraint

```
[fMatrix, epipolarInliers, status] = estimateFundamentalMatrix(...
  matchedPoints1,matchedPoints2,Method="RANSAC", ...
  NumTrials=10000,DistanceThreshold=0.1,Confidence=99.99);

if status ~= 0 || isEpipoleInImage(fMatrix,size(I1)) ...
  || isEpipoleInImage(fMatrix',size(I2))
```

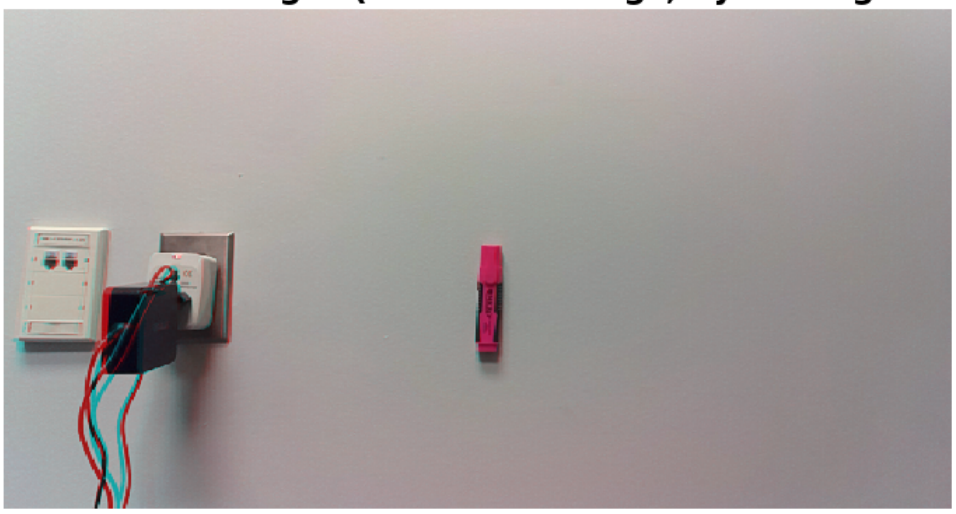
```
  error(["Not enough matching points were found or "...
         "the epipoles are inside the images. Inspect "...
         "and improve the quality of detected features ",...
         "and images."]);
end

inlierPoints1 = matchedPoints1(epipolarInliers, :);
inlierPoints2 = matchedPoints2(epipolarInliers, :);

figure
showMatchedFeatures(I1, I2, inlierPoints1, inlierPoints2)
legend("Inlier Points In I1","Inlier Points In I2")
```




## Step 5: Rectify Images

```
[tform1, tform2] = estimateStereoRectification(fMatrix, ...
  inlierPoints1.Location,inlierPoints2.Location,size(I2));

[I1Rect, I2Rect] = rectifyStereoImages(I1,I2,tform1,tform2);
figure
imshow(stereoAnaglyph(I1Rect,I2Rect))
title("Rectified Stereo Images (Red - Left Image, Cyan - Right
Image)")
```

**Rectified Stereo Images (Red - Left Image, Cyan - Right Image)**



## Step 6: Generalize The Rectification Process

```
cvexRectifyImages("parkinglot_left.png","parkinglot_right.png");
```

**Rectified Stereo Images (Red - Left Image, Cyan - Right Image)**

# Question 8

Implement a real-time object tracker (two versions) that (i) uses a marker (e.g. QR code or April tags), and (ii) does not use any marker and only relies on the object.

I only implemented the second version. The source code is attached with the assignment files. Please find the screen recordings in the attachments as well. Here is the code and a screenshot of its functioning.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

object = plt.imread('input/question_8.jpg')

cv2.namedWindow("Reza Mansouri - A3Q8", cv2.WINDOW_FREERATIO)
vc = cv2.VideoCapture(0)

if vc.isOpened():
    rval, frame = vc.read()
else:
    rval = False
```

```
while rval:
    ssds = []
    coords = []
    for i in range(0, frame.shape[0] - object.shape[0], 30):
        for j in range(0, frame.shape[1] - object.shape[1], 30):
            candidate = frame[i:i+object.shape[0],
j:j+object.shape[1]]
            ssd = np.sum(np.power(candidate-object, 2))
            ssds.append(ssd)
            coords.append((j, i))
    left_top = coords[np.argmin(ssds)]
    right_bottom = (left_top[0] + object.shape[1], left_top[1] +
object.shape[0])
    frame = cv2.rectangle(frame, left_top, right_bottom, color=(0,
255, 255), thickness=5)
    cv2.imshow("Reza Mansouri - A3Q8", frame)
    rval, frame = vc.read()
    key = cv2.waitKey(60)
    if key == 27:
        break

vc.release()
cv2.destroyWindow("Reza Mansouri - A3Q8")
```