Assignment3

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.

Solution:

The below snippet shows the python code for capturing the frames from the video and performing the normalization and then performing the comparison of the cropped image in the captured frames.

If it is detected it is shown as the rectangle.

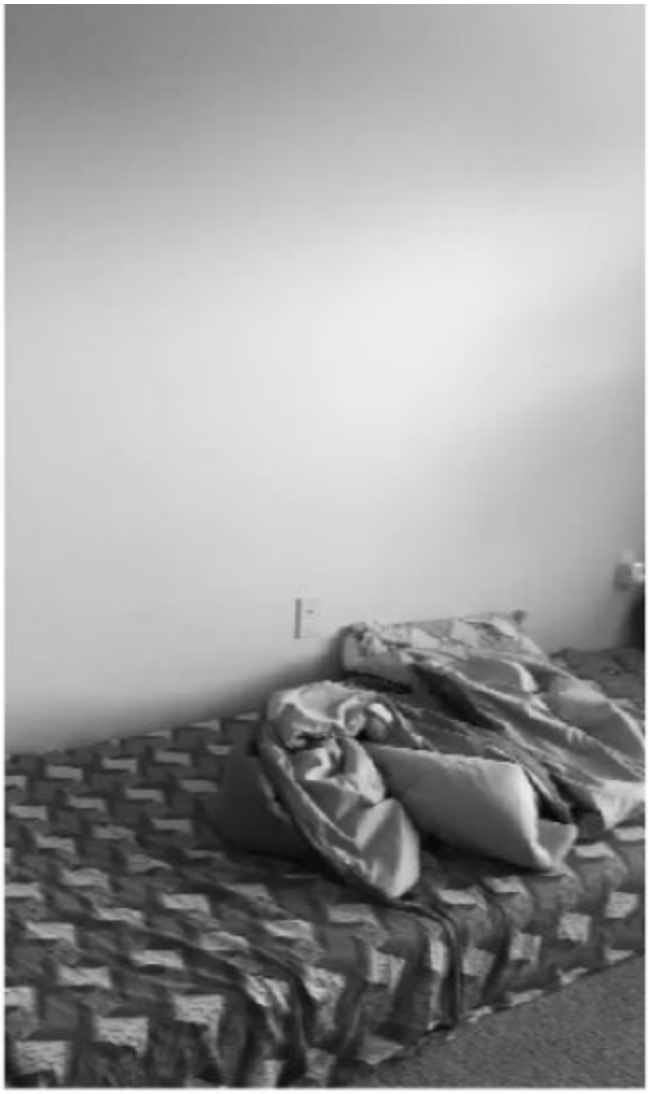
In [1]:
```
# import the necessary packages
import cv2
print("[INFO] loading images...")
#capturing video frames:
import cv2
import time
cap = cv2.VideoCapture('video_10sec_assign3.mp4')
i = 0
while(cap.isOpened()):
    ret, frame = cap.read()
    # This condition prevents from infinite looping
    # incase video ends.
    if ret == False:
        break
    # capture frames by Frame into disk using imwrite method
    if i%30==0:
        cv2.imwrite('Frame'+str(i)+'.jpg', frame,data)
    i += 1
    cap.release()
cv2.destroyAllWindows()
```

```
def norm_data(data):
    mean_data=np.mean(data)
    std_data=np.std(data, ddof=1)
print("[INFO] performing template matching...")
image = cv2.imread(args["fps90_croppedimage.jpg"])
imageGray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
result = cv2.matchTemplate(imageGray, templateGray,cv2.TM_CCOEFF_NORMED)
(minVal, maxVal, minLoc, maxLoc) = cv2.minMaxLoc(result)
# determine the starting and ending (x, y)-coordinates of the
# bounding box
(startX, startY) = maxLoc
endX = startX + template.shape[1]
endY = startY + template.shape[0]
# draw the bounding box on the image we want to detect from the video frames
cv2.rectangle(image, (startX, startY), (endX, endY), (255, 0, 0), 3)
# show the output image
cv2.imshow("Output", image)
cv2.waitKey(0)
```

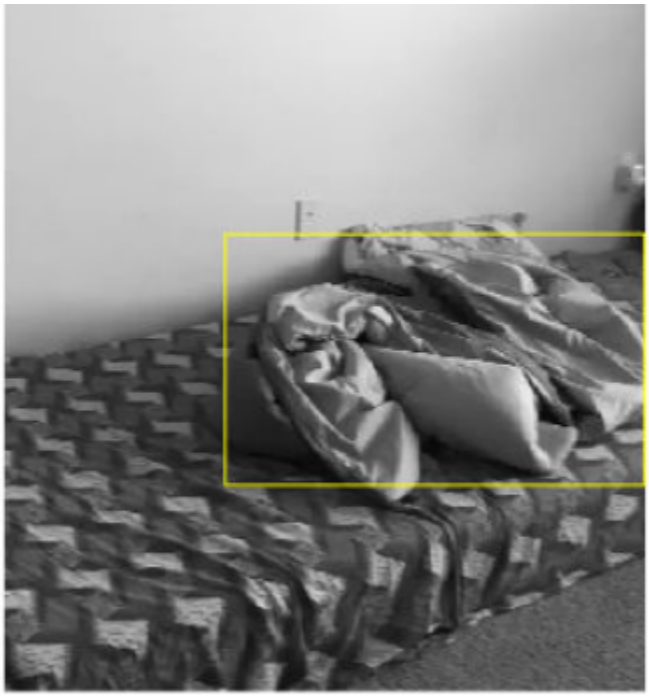
```
[INFO] loading images...
[INFO] performing template matching...
```

The cropped image is showm below:

And the below ones are the some of the frames captured in the video:(just for display)



Here we get the rectangular box on the cropped image, which is detected in the frames captured and is shown below:



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 equation

- We find optical flow constraint equn & then develop algorithm for the estimation of optical flow at each point
- Consider 2 images taken at time t, $t+\delta t$.
- Consider small window in the 2 images (at the same location) By focusing at the single point.

  the point will be moved to another location at δt which is $(x+\delta x, y+\delta y)$

We get $\boxed{(x, y) \quad \text{and} \quad (x+\delta x, y+\delta y)}$

Displacement is $\boxed{(\delta x, \delta y)}$ and optical flow $\boxed{u, v = \left(\frac{\delta x}{\delta t}, \frac{\delta y}{\delta t}\right)}$

- we assume brightness of Image point remains constant with time

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

- and also we assume displacement $(\delta x, \delta y)$ and time step δt are small
- expanding func^n as infinite sum of its derivatives:

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

δx is small

$$\Rightarrow f(x+\delta x) = f(x) + \frac{\partial f}{\partial x}\delta x + 0$$

then

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

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

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

——— (2)

① − ② we get

$$I_x \delta x + I_y \delta y + I_t \delta t = 0$$

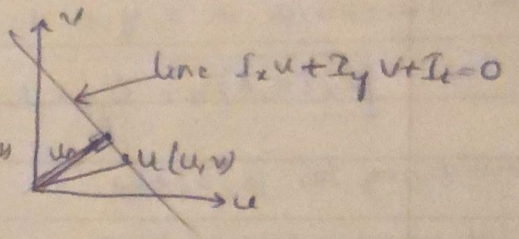
divide by $\delta t \rightarrow I_x \frac{\partial x}{\partial t} + I_y \frac{\partial y}{\partial t} + I_t = 0$

Constraint equation: $\boxed{I_x u + I_y v + I_t = 0}$ u,v optical flow

. I_x I_y I_t can be calculated from two frames

For any point (x,y) in the image, its optical flow (u,v) lies in the line

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



and $u = u_n + u_p$ ∵ 2 Components

u_n = Normal flow

u_p = parallel flow

$x_1 = 411.7500$ | $x_2 = 618.7500$

$y_1 = 361.2500$ | $y_2 = 380.7500$

$\delta x = 207$, $\delta y = 19.5$

15 fps $\Rightarrow \frac{\delta x}{\delta t} = \frac{207}{\delta t} = \frac{207}{1/15}$

for rgb → 15 fps

$1/15 =$

$\frac{\delta x}{\delta t} = 207(15) = 3105$

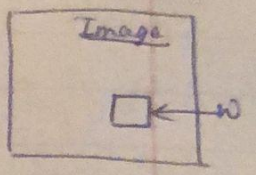
$\frac{\delta y}{\delta t} = 19.5(15) = 292.5$

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

Lucal Kanade algorithm

- It assumes that optical flow in a very small neighbourhood in the scene, is same for all the points in the neighbourhood.

Assumption : For each pixel, assume motion field and optical flow (u,v) is constant within a small neighbourhood ω.



- They produce same motion field & hence same optical flow in image.

- So for all the points $(k,l) \in \omega$

  the derivatives in x, y, t directions $= 0$

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

Considering for $(k,l) \in \omega$.

let window size be $n \times n$

In matrix form:

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

A	u	B
Known	unknown	Known
$n^2 \times 2$	2×1	$n^2 \times 1$

Considering $Au = B$

$$A^T A u = A^T B$$

matrix form

$$\begin{bmatrix} \Sigma_\omega I_x I_x & \Sigma_\omega I_x I_y \\ \Sigma_\omega I_x I_y & \Sigma_\omega I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -\Sigma_\omega I_x I_t \\ -\Sigma_\omega I_y I_t \end{bmatrix}$$

A^TA (known) $\quad u$ (unknown) $\quad$ Known A^TB

$$u = (A^T A)^{-1} A^T B$$

① $A^T A$ must be invertible. $\Rightarrow \det(A^T A) \neq 0$

② $A^T A$ must be well-conditioned

λ_1 and λ_2 eigenvalues of $A^T A$, then $\lambda_1 > \epsilon$ and $\lambda_2 \geq \epsilon$

$\lambda_1 \geq \lambda_2$ but not $\lambda_1 >> \lambda_2$

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.

Assignment 3

④

$u = f_x\left(\frac{x}{z}\right) + o_x \qquad (o_x\ o_y) \rightarrow$

$v = f_y\left(\frac{y}{z}\right) + o_y$

Baseline
= 355.6mm

4 equations

$u_L = f_x\left(\frac{x}{z}\right) + o_x \qquad v_L = f_y\left(\frac{y}{z}\right) + o_y$

$u_R = f_x\left(\frac{x-b}{z}\right) + o_x \qquad v_R = f_y\left(\frac{y}{z}\right) + o_y$

B = 14 in
= 355.6mm
Z = 25 inches
= 635mm

$x = \frac{b(u_L - o_x)}{u_L - u_R} \qquad y = \frac{bf_x(v_L - o_y)}{f_y(u_L - u_R)} \qquad z = \frac{bf_x}{u_L - u_R}$

$f_x = 1.6202 \times 10^3 \qquad o_x = 970.6671$

$f_y = 1.623 \times 10^3 \qquad o_y = 529.05$

left
x = 851.5000
y = 217.5000

right
x = 875.5000
y = 233.5000

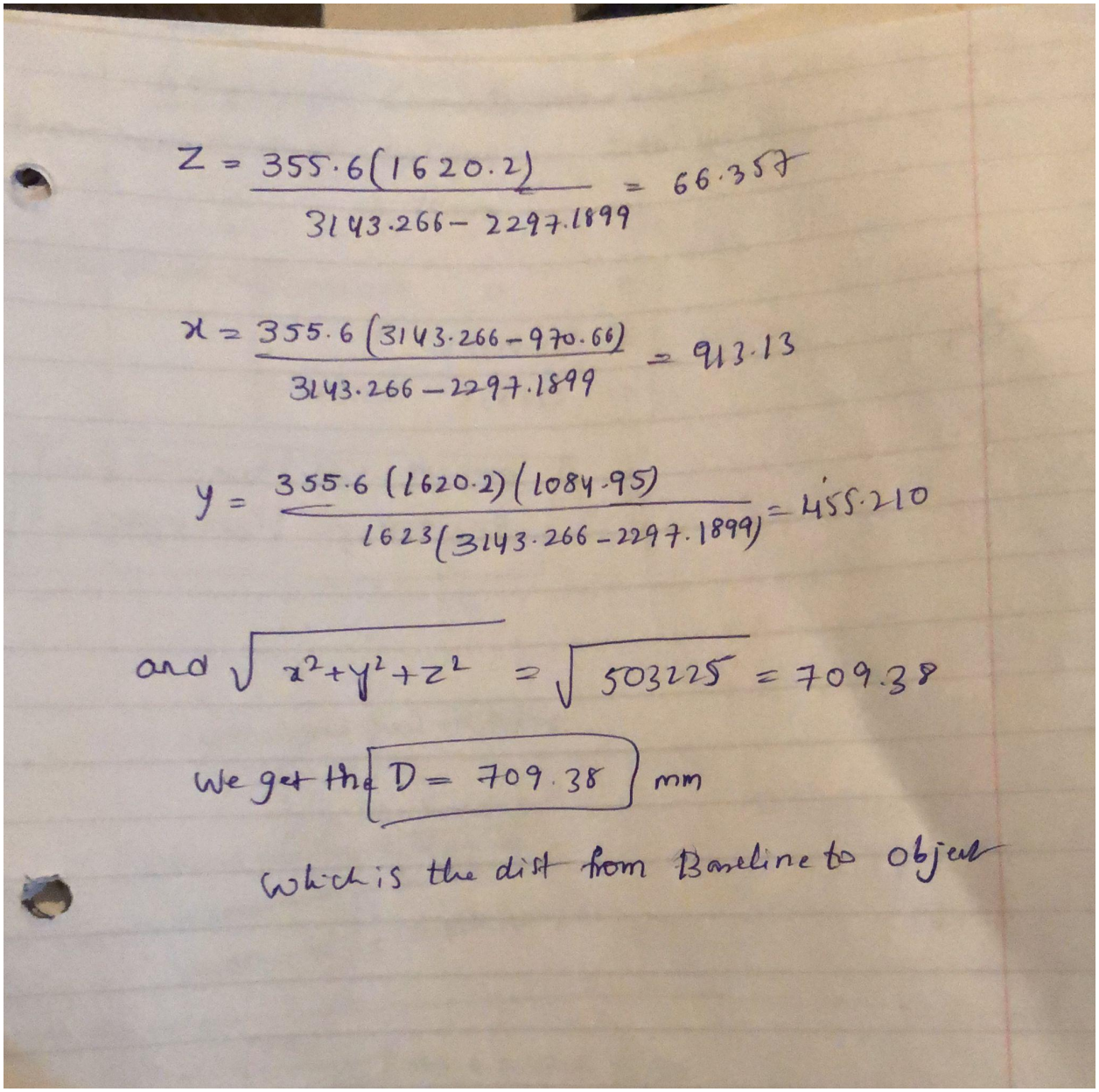
obtain → u_L, v_L ; obtain → u_R, v_R

$u_L = \frac{1620.2 \times 851.5}{635} + 970.6671 = (2172.5989 + 970.6671) = 3143.2$

$u_R = \frac{1620.2 \times (875.5 - 355.6)}{635} + 970.6671 = \frac{519.9)1620.2}{625} + 970.6 = 2297.189$

$v_L = \frac{1623 \times (217.5)}{635} + 529.05 = 555.90 + 52905 = 1084.95$

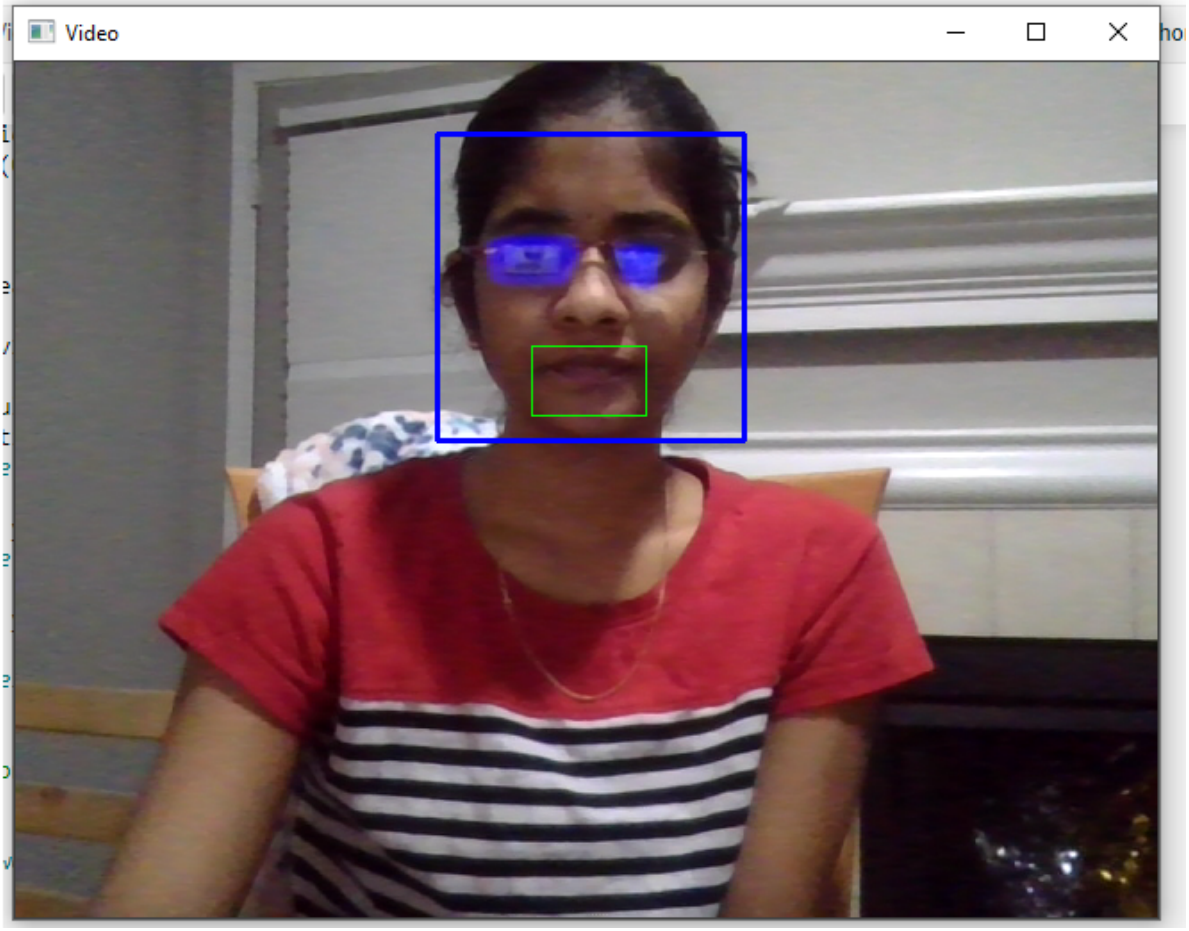
$v_R = \frac{1623 \times (233.5)}{635} + 529.05 = 1125.85$



9.#python program for faces and lip movement detection

```
#real time face and mouth tracking application
import cv2
import sys
mouthCascade = cv2.CascadeClassifier('mouth.xml')
faceCascade = cv2.CascadeClassifier('face.xml')
video_capture = cv2.VideoCapture(0)

while True:
```

```
    # Capture frame-by-frame
    ret, frame = video_capture.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    mouth = mouthCascade.detectMultiScale(gray, 1.3, 5)
    face = faceCascade.detectMultiScale(gray, 1.3, 5)
    # Draw a rectangle around the mouth
    for (x, y, w, h) in mouth:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 1)
    # Draw a rectangle around the face
    for (x, y, w, h) in face:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
    # Here we will Display the resulting frame
    cv2.imshow('Video', frame)
    iif cv2.waitKey(1) & 0xFF == ord('q'):
        break

# after everything is completed we an release the frame
video_capture.release()
cv2.destroyAllWindows
```



```
vidReader = VideoReader('video_10sec_assignment.mp4','CurrentTime',1);
h = figure;
movegui(h);
hViewPanel = uipanel(h,'Pos',[0 0 1 1],'heading','we get the optical vectors');
hPlot = axes(hViewPanel);
opticFlows = opticalFlowHS
```

```
opticFlows =
  opticalFlowHS with properties:

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

```
%floww = estimateFlow(opticFlow,frameGrayy);

while hasFrame(vidReader)
    frameRGB = readFrame(vidReader);
    frameGray = im2gray(frameRGB);
    img2 = im2double(frameGray);
    RGB = imread('Frame11.jpg');
    frameGrayy = im2gray(RGB);
    img1 = im2double(frameGrayy);
    opticFlow = opticalFlow(img1,img2);
    flow = estimateFlow(opticFlows,frameGray);
    imshow(frameRGB)
    hold on
    plot(flow,'DecimationFactor',[5 5],'Scalefactor',60,'Parent',hPlot);
    hold off
    pause(10^-3)
end
```

we get the optical vectors

```
imds = imageDatastore('5_images_set','IncludeSubfolders',true,'LabelSource','foldernames');
tbl = countEachLabel(imds)
```

tbl = 5×2 table

	Label	Count
1	bottle_bof1	5
2	knife_bof1	5
3	peeler_bof1	5
4	scissor_bof1	5
5	tape_bof1	5

```
figure
montage(imds.Files(1:6:end))
```


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

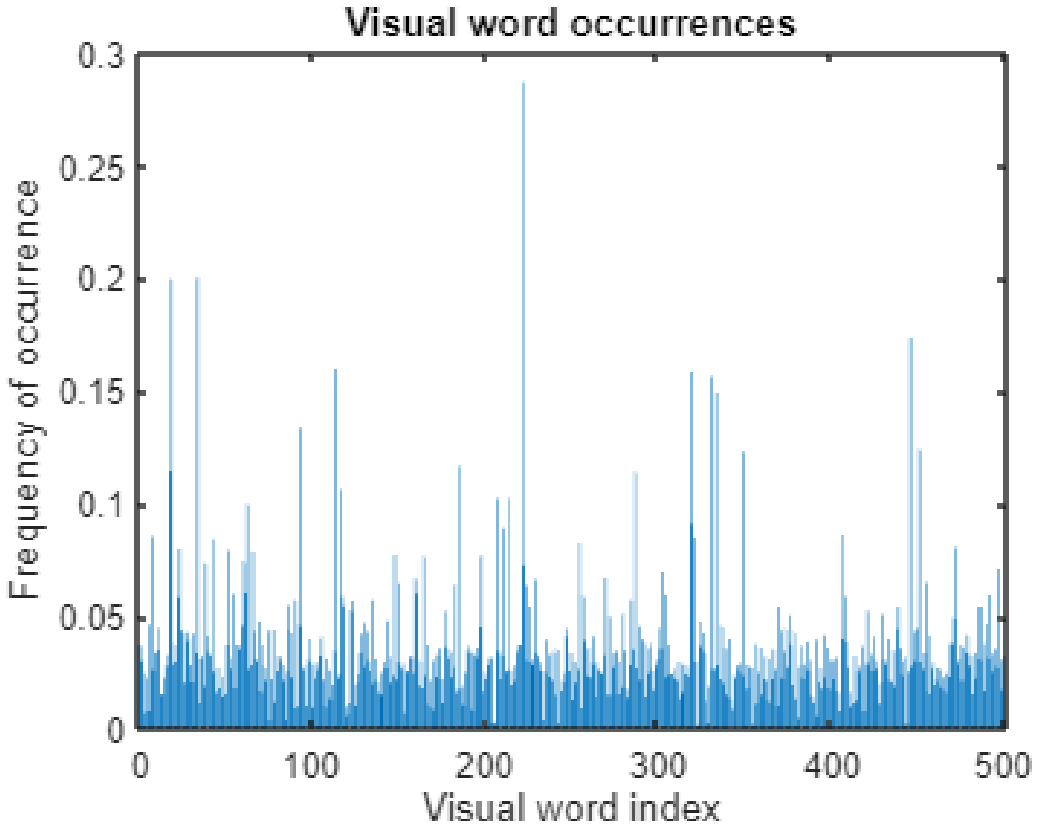
```
bag = bagOfFeatures(trainingSet);
```

```
Creating Bag-Of-Features.
-------------------------
* Image category 1: bottle_bof1
* Image category 2: knife_bof1
* Image category 3: peeler_bof1
* Image category 4: scissor_bof1
* Image category 5: tape_bof1
* 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 15 images...done. Extracted 2211840 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          : 1769470
* Number of clusters          : 500
* Initializing cluster centers...100.00%.
* Clustering...completed 28/100 iterations (~8.80 seconds/iteration)...converged in 28 iterations.

* Finished creating Bag-Of-Features
```

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

```
Encoding images using Bag-Of-Features.
--------------------------------------
* Encoding an image...done.
```

```
% 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);
```

```
Unrecognized function or variable 'bottle_bof1'.
```

```
confMatrix = evaluate(categoryClassifier, trainingSet);
confMatrix = evaluate(categoryClassifier, validationSet);
% Compute average accuracy
mean(diag(confMatrix))

img = imread(fullfile('5_images_zip','bottle_bof1','bottle_bof1.jpeg'));
figure
imshow(img)
[labelIdx, scores] = predict(categoryClassifier, img);

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

```
setDir  = fullfile('2_images_set');
imgSets = imageSet(setDir,'recursive');

trainingSets = partition(imgSets,2);

bag = bagOfFeatures(trainingSets,'Verbose',false);

img = read(imgSets(1),1);
featureVector = encode(bag,img);
```

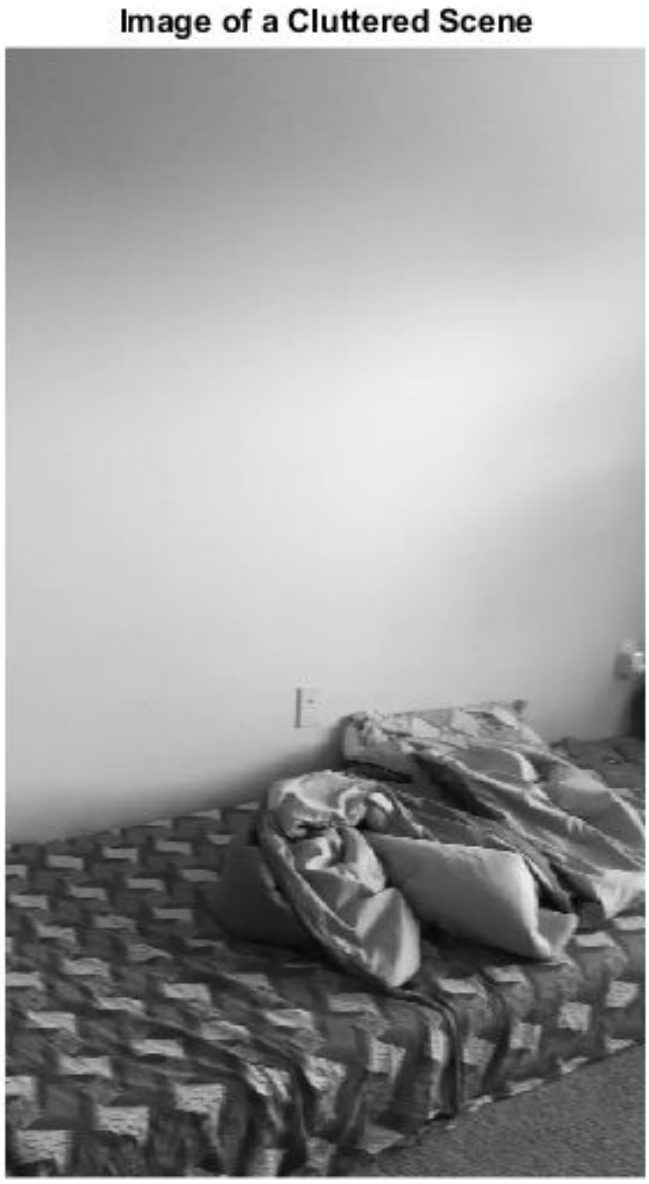
```
Encoding images using Bag-Of-Features.
--------------------------------------
* Encoding an image...done.
```

```
setDir  = fullfile('5_images_set');
imds = imageDatastore(setDir,'IncludeSubfolders',true,'LabelSource',...
    'foldernames');

extractor = @exampleBagOfFeaturesExtractor;
bag = bagOfFeatures(imds,'CustomExtractor',extractor)
```

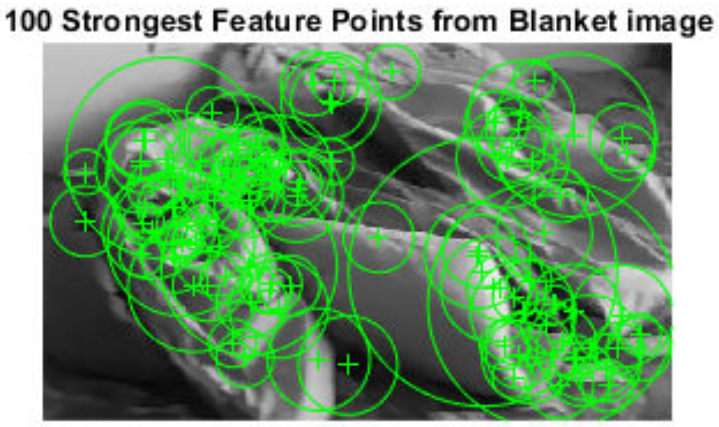
```
Creating Bag-Of-Features.
-------------------------
* Image category 1: bottle_bof1
* Image category 2: knife_bof1
* Image category 3: peeler_bof1
* Image category 4: scissor_bof1
* Image category 5: tape_bof1
* Extracting features using a custom feature extraction function: exampleBagOfFeaturesExtractor.

* Extracting features from 25 images...done. Extracted 3686400 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          : 2949120
* Number of clusters          : 500
* Initializing cluster centers...100.00%.
* Clustering...completed 31/100 iterations (~16.19 seconds/iteration)...converged in 31 iterations.

* Finished creating Bag-Of-Features
bag =
  bagOfFeatures with properties:

      CustomExtractor: @exampleBagOfFeaturesExtractor
       NumVisualWords: 500
       TreeProperties: [1 500]
    StrongestFeatures: 0.8000
```

```
RGB = imread('frame90_cropped.jpg');
boxImage = rgb2gray(RGB);
figure;
imshow(boxImage);
title('Image of a blanket');
```


```
RGB1 = imread('frame90.jpg');
sceneImage = rgb2gray(RGB1);
%sceneImage = imread(I1);
figure;
imshow(sceneImage);
title('Image of a Cluttered Scene');
```

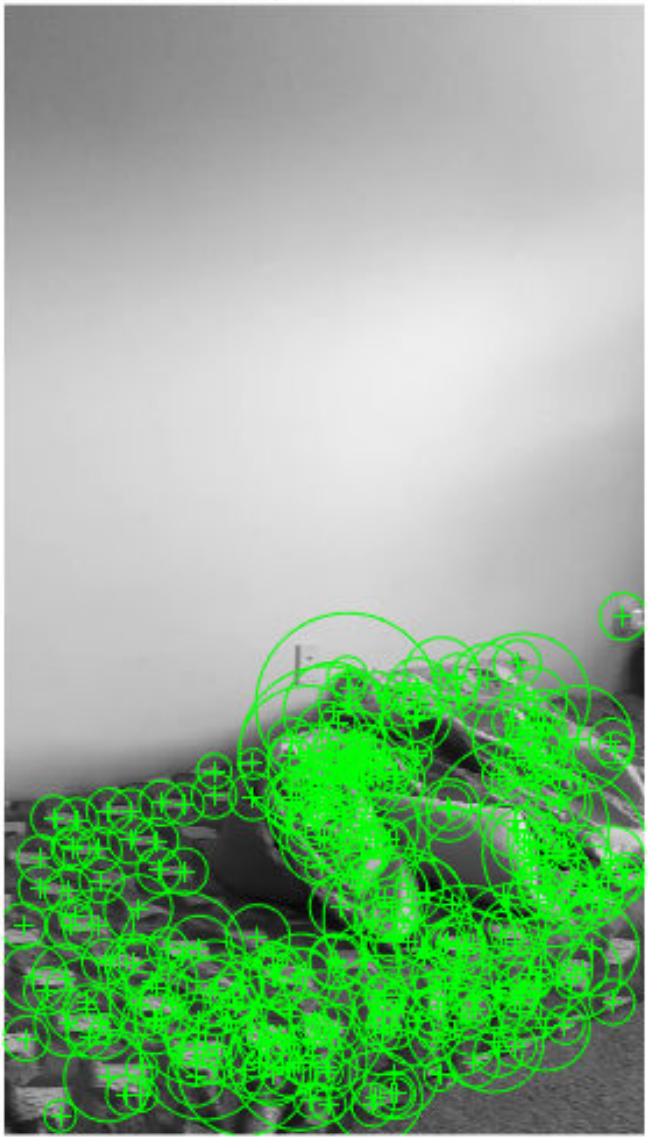



```
boxPoints = detectSURFFeatures(boxImage);
scenePoints = detectSURFFeatures(sceneImage);

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



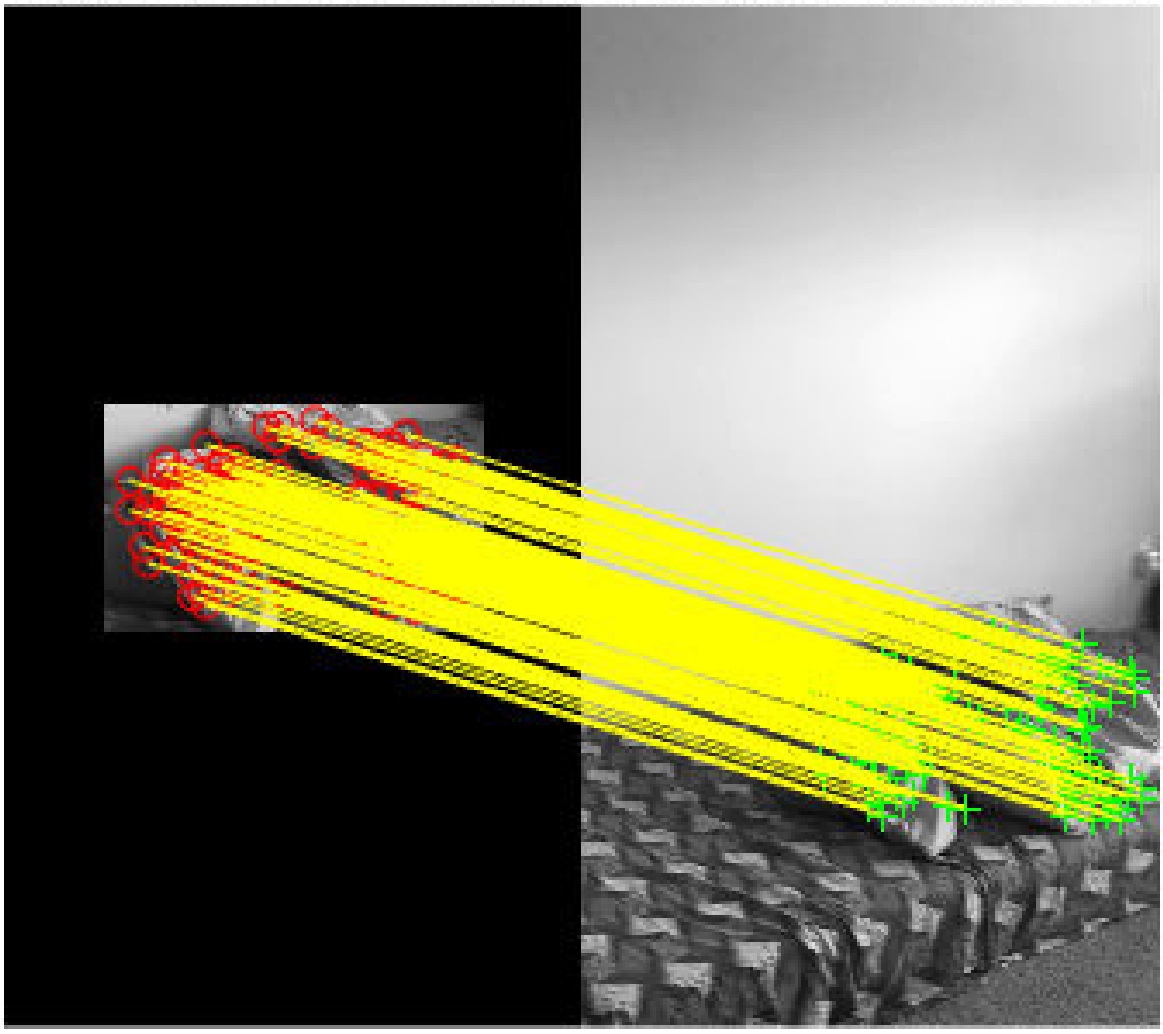

```
figure;
imshow(sceneImage);
title('300 Strongest Feature Points from blanket or clutter Image');
hold on;
plot(selectStrongest(scenePoints, 300));
```

300 Strongest Feature Points from blanket or clutter Image

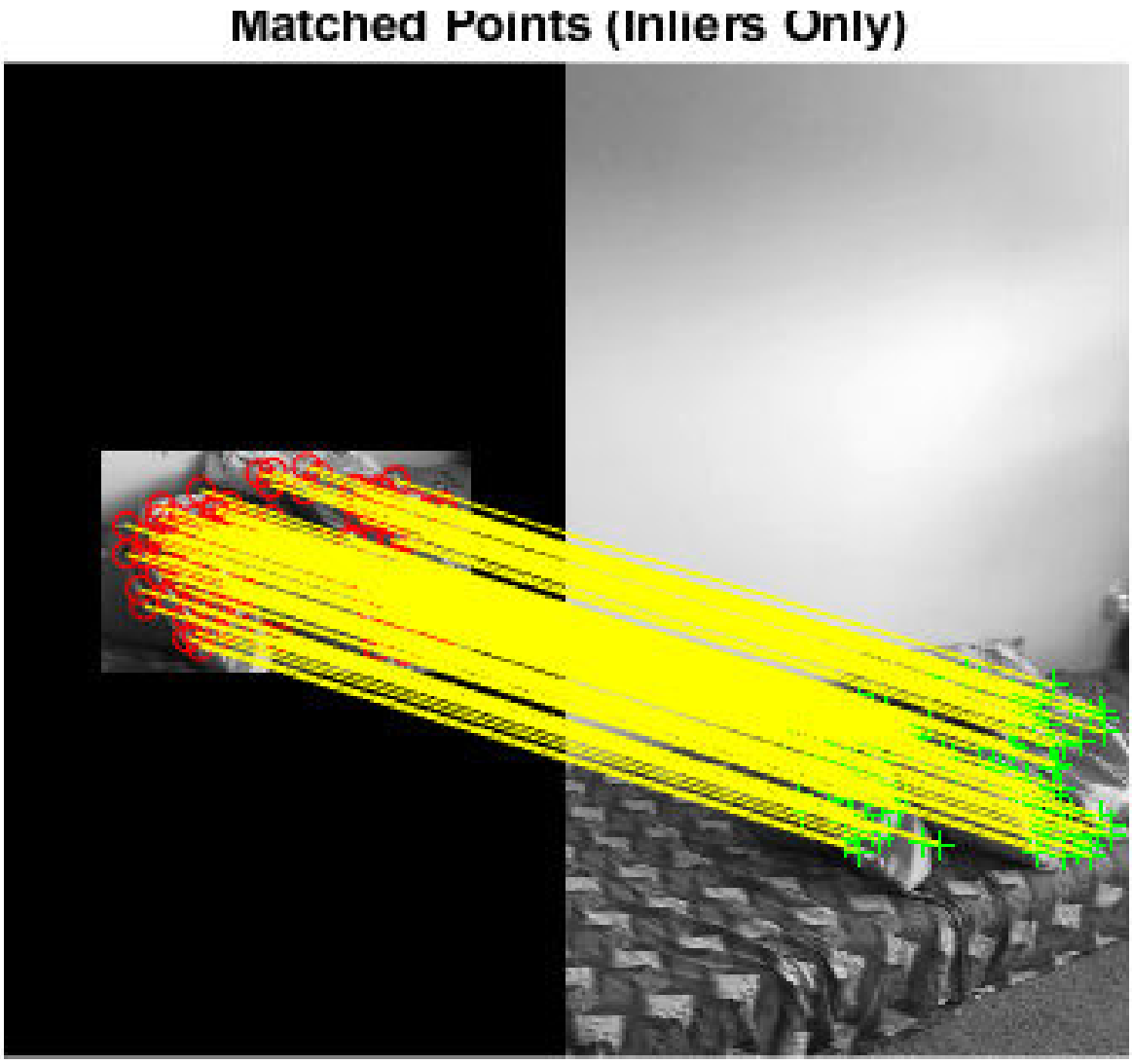


```
[boxFeatures, boxPoints] = extractFeatures(boxImage, boxPoints);
[sceneFeatures, scenePoints] = extractFeatures(sceneImage, scenePoints);

boxPairs = matchFeatures(boxFeatures, sceneFeatures);

matchedBoxPoints = boxPoints(boxPairs(:, 1), :);
matchedScenePoints = scenePoints(boxPairs(:, 2), :);
figure;
showMatchedFeatures(boxImage, sceneImage, matchedBoxPoints, ...
    matchedScenePoints, 'montage');
title(' (Including Outliers) the putatively matched points are shown');
```

(Including Outliers) the putatively matched points are shown



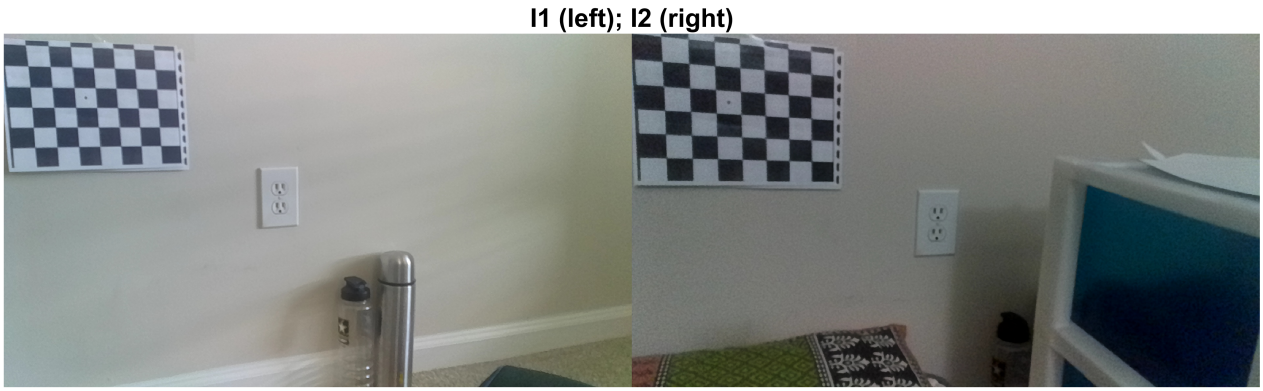
```
[tform, inlierIdx] = ...
    estimateGeometricTransform2D(matchedBoxPoints, matchedScenePoints, 'affine');
inlierBoxPoints   = matchedBoxPoints(inlierIdx, :);
inlierScenePoints = matchedScenePoints(inlierIdx, :);

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

newBoxPolygon = transformPointsForward(tform, boxPolygon);

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


Detected blanket

```
I1 = imread('cb_fps_image1.jpg');
I2 = imread('fps_image39.jpg');

% Convert to grayscale.
I1gray = rgb2gray(I1);
I2gray = rgb2gray(I2);

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)

```
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');
```


```
[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');
```


```
[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(['Either not enough matching points were found or '...
         'the epipoles are inside the images. You may need to '...
         'inspect and improve the quality of detected features ',...
         'and/or improve the quality of your images.']);
end
```

```
Either not enough matching points were found or the epipoles are inside the images. You may need to inspect and improve the quality of detected features and/or improve the quality of your images.
```

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

figure;
showMatchedFeatures(I1, I2, inlierPoints1, inlierPoints2);
legend('Inlier points in I1', 'Inlier points in I2');

[t1, t2] = estimateUncalibratedRectification(fMatrix, ...
  inlierPoints1.Location, inlierPoints2.Location, size(I2));
```

```
tform1 = projective2d(t1);
tform2 = projective2d(t2);

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

%RectifyImages('rgb_image1.jpg', 'rgb_image16.jpg');
```