ASSIGNMENT - 1
CBIVR

PRIYANSHU SHARMA
15BCE1282

QUESTION - 1

Detect SURF Interest points in a Gray scale image

THEORY

In computer vision, **speeded up robust features** (**SURF**) is a patented local feature detector and descriptor. It can be used for tasks such as object recognition, image registration, classification or 3D reconstruction. It is partly inspired by the scale-invariant feature transform (SIFT) descriptor. The standard version of SURF is several times faster than SIFT and claimed by its authors to be more robust against different image transformations than SIFT.

To detect interest points, SURF uses an integer approximation of the determinant of Hessian blob detector, which can be computed with 3 integer operations using a precomputed integral image. Its feature descriptor is based on the sum of the Haar wavelet response around the point of interest. These can also be computed with the aid of the integral image.

SURF descriptors have been used to locate and recognize objects, people or faces, to reconstruct 3D scenes, to track objects and to extract points of interest.

CODE -

```
srcFiles = dir('C:\Users\PRIYANSHU SHARMA\Desktop\PRIYANSHU\6 STUDY\MATLAB\ASSIGNMENT - 1\*.jpg');
srcFiles;
for i=1:5
    d = strcat('C:\Users\PRIYANSHU SHARMA\Desktop\PRIYANSHU\6 STUDY\MATLAB\ASSIGNMENT - 1\',srcFiles(i).name);
    di = imread(d);
    g = rgb2gray(di);
    figure, imshow(g);
    points = detectSURFFeatures(g);
    imshow(g);
    hold on;
    plot(points.selectStrongest(10));
end
```

FUNCTION

```
function Pts=detectSURFFeatures(I, varargin)
%detectSURFFeatures Finds SURF features.
%   points = detectSURFFeatures(I) returns a SURFPoints object, points,
%   containing information about SURF features detected in a 2-D grayscale
%   image I. detectSURFFeatures uses Speeded-Up Robust Features
%   (SURF) algorithm to find blob features.
%
%   points = detectSURFFeatures(I,Name,Value) specifies additional
%   name-value pair arguments described below:
%
%   'MetricThreshold'    A non-negative scalar which specifies a threshold
%                        for selecting the strongest features. Decrease it to
%                        return more blobs.
%
%                        Default: 1000.0
%
```

```
%     'NumOctaves'            Integer scalar, NumOctaves >= 1. Number of
octaves
%                               to use. Increase this value to detect larger
%                               blobs. Recommended values are between 1
and 4.
%
%                               Default: 3
%
%     'NumScaleLevels'      Integer scalar, NumScaleLevels >= 3. Number
of
%                               scale levels to compute per octave. Increase
%                               this number to detect more blobs at finer
scale
%                               increments. Recommended values are
between 3 and 6.
%
%                               Default: 4
%
%     'ROI'                   A vector of the format [X Y WIDTH HEIGHT],
%                               specifying a rectangular region in which
corners
%                               will be detected. [X Y] is the upper left
corner of
%                               the region.
%
%                               Default: [1 1 size(I,2) size(I,1)]
%
%     Notes
%     -----
%     - Each octave spans a number of scales that are analyzed using
varying
%       size filters:
%             octave         filter sizes
%             ------         ----------------------------
%               1              9x9,      15x15, 21x21, 27x27, ...
%               2              15x15, 27x27, 39x39, 51x51, ...
%               3              27x27, 51x51, 75x75, 99x99, ...
%               4              ....
%       Higher octaves use larger filters and sub-sample the image data.
%       Larger number of octaves will result in finding larger size blobs.
```

```
%         'NumOctaves' should be selected appropriately for the image
size.
%         For example, 50x50 image should not require NumOctaves > 2.
The
%         number of filters used per octave is controlled by the parameter
%         'NumScaleLevels'. To analyze the data in a single octave, at least
3
%         levels are required.
%
%     Class Support
%     -------------
%     The input image I can be logical, uint8, int16, uint16, single,
%     or double, and it must be real and nonsparse.
%
%     Example
%     -------
%     % Detect interest points and mark their locations
%     I = imread('cameraman.tif');
%     points = detectSURFFeatures(I);
%     imshow(I); hold on;
%     plot(points.selectStrongest(10));
%
%     See also SURFPoints, extractFeatures, matchFeatures,
%                   detectBRISKFeatures, detectFASTFeatures,
detectHarrisFeatures,
%                   detectMinEigenFeatures, detectMSERFeatures

%     Copyright 2010 The MathWorks, Inc.

%     References:
%         Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool
"SURF:
%         Speeded Up Robust Features", Computer Vision and Image
Understanding
%         (CVIU), Vol. 110, No. 3, pp. 346--359, 2008

%#codegen

checkImage(I);
```

```
Iu8 = im2uint8(I);

if isSimMode()
        [Iu8, params] = parseInputs(Iu8,varargin{:});
        PtsStruct=ocvFastHessianDetector(Iu8, params);

else
        [I_u8, params] = parseInputs_cg(Iu8,varargin{:});

        % get original image size
        nRows = size(I_u8, 1);
        nCols = size(I_u8, 2);
        numInDims = 2;

        % column-major (matlab) to row-major (opencv)
        Iu8 = I_u8';

        % output variable size and it's size cannot be determined here;
        % Inside OpenCV algorithm, vector is used to hold output;
        % Vector is grown by pushing element into it; Once OpenCV
computation is
        % done, output size is known, and we use that size to create output
        % memory using malloc; Then elements are copied from OpenCV
Vector to EML
        % output buffer

        [PtsStruct_Location, PtsStruct_Scale, PtsStruct_Metric,
PtsStruct_SignOfLaplacian] = ...

vision.internal.buildable.fastHessianDetectorBuildable.fastHessianDetect
or_uint8(Iu8, ...
                int32(nRows), int32(nCols), int32(numInDims), ...
                int32(params.nOctaveLayers), int32(params.nOctaves),
int32(params.hessianThreshold));

        PtsStruct.Location              = PtsStruct_Location;
        PtsStruct.Scale                 = PtsStruct_Scale;
        PtsStruct.Metric                = PtsStruct_Metric;
        PtsStruct.SignOfLaplacian = PtsStruct_SignOfLaplacian;
end
```

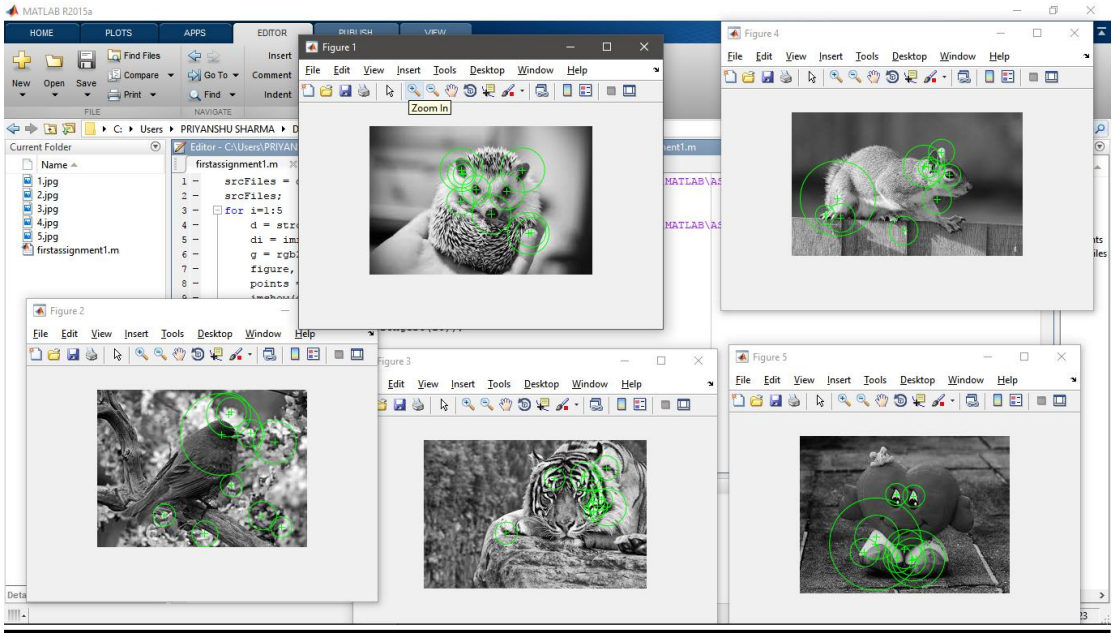
```
PtsStruct.Location =
vision.internal.detector.addOffsetForROI(PtsStruct.Location, params.ROI,
params.usingROI);

Pts = SURFPoints(PtsStruct.Location, PtsStruct);

%==========================================================
===============
function checkImage(I)
validateattributes(I,{'logical', 'uint8', 'int16', 'uint16', ...
      'single', 'double'}, {'2d', 'nonempty', 'nonsparse', 'real'},...
                           'detectSURFFeatures', 'I', 1); %#ok<EMCA>

%==========================================================
===============
function flag = isSimMode()

flag = isempty(coder.target);

%==========================================================
===============
% Parse and check inputs - simulation
%==========================================================
===============
function [img, params] = parseInputs(Iu8, varargin)

sz = size(Iu8);
defaults = getDefaultParametersVal(sz);

% Parse the PV pairs
parser = inputParser;
parser.addParameter('MetricThreshold', defaults.MetricThreshold,
@checkMetricThreshold);
parser.addParameter('NumOctaves',          defaults.NumOctaves,
@checkNumOctaves);
parser.addParameter('NumScaleLevels',    defaults.NumScaleLevels,
@checkNumScaleLevels);
parser.addParameter('ROI',                      defaults.ROI,
@(x)vision.internal.detector.checkROI(x,sz)); %#ok<*EMFH>
```

```
% Parse input
parser.parse(varargin{:});

% Populate the parameters to pass into OpenCV's
icvfastHessianDetector()
params.nOctaveLayers       = parser.Results.NumScaleLevels-2;
params.nOctaves            = parser.Results.NumOctaves;
params.hessianThreshold = parser.Results.MetricThreshold;
params.ROI                 = parser.Results.ROI;

params.usingROI = isempty(regexp([parser.UsingDefaults{:} ''],...
    'ROI','once')); %#ok<EMCA>

img = vision.internal.detector.cropImageIfRequested(Iu8, params.ROI,
params.usingROI);


%=========================================================
================
% Parse and check inputs - code-generation
%=========================================================
================
function [img, params] = parseInputs_cg(Iu8, varargin)

% varargin must be non-empty
defaultsVal   = getDefaultParametersVal(size(Iu8));
defaultsNoVal = getDefaultParametersNoVal();
properties    = getEmlParserProperties();
optarg = eml_parse_parameter_inputs(defaultsNoVal, properties,
varargin{:});
MetricThreshold = (eml_get_parameter_value( ...
        optarg.MetricThreshold, defaultsVal.MetricThreshold,
varargin{:}));
NumOctaves = (eml_get_parameter_value( ...
        optarg.NumOctaves, defaultsVal.NumOctaves, varargin{:}));
NumScaleLevels = (eml_get_parameter_value( ...
        optarg.NumScaleLevels, defaultsVal.NumScaleLevels,
varargin{:}));
ROI    = eml_get_parameter_value(optarg.ROI, ...
```

```
        defaultsVal.ROI, varargin{:});

checkMetricThreshold(MetricThreshold);
checkNumOctaves(NumOctaves);
checkNumScaleLevels(NumScaleLevels);

% check whether ROI parameter is specified
usingROI = optarg.ROI ~=uint32(0);

if usingROI
        vision.internal.detector.checkROI(ROI, size(Iu8));
end

params.nOctaveLayers        = uint32(NumScaleLevels)-uint32(2);
params.nOctaves             = uint32(NumOctaves);
params.hessianThreshold = uint32(MetricThreshold);
params.usingROI             = usingROI;
params.ROI                  = ROI;

img = vision.internal.detector.cropImageIfRequested(Iu8, params.ROI,
usingROI);

%=========================================================
================
function defaultsVal = getDefaultParametersVal(imgSize)

defaultsVal = struct(...
        'MetricThreshold', uint32(1000), ...
        'NumOctaves', uint32(3), ...
        'NumScaleLevels', uint32(4),...
        'ROI',int32([1 1 imgSize([2 1])]));

%=========================================================
================
function defaultsNoVal = getDefaultParametersNoVal()

defaultsNoVal = struct(...
        'MetricThreshold', uint32(0), ...
        'NumOctaves',           uint32(0), ...
        'NumScaleLevels',    uint32(0), ...
```

```
        'ROI',                      uint32(0));

%=========================================================
================
function properties = getEmlParserProperties()

properties = struct( ...
        'CaseSensitivity', false, ...
        'StructExpand',       true, ...
        'PartialMatching', false);

%=========================================================
================
function tf = checkMetricThreshold(threshold)
validateattributes(threshold, {'numeric'}, {'scalar','finite',...
        'nonsparse', 'real', 'nonnegative'},
'detectSURFFeatures'); %#ok<EMCA>
tf = true;

%=========================================================
================
function tf = checkNumOctaves(numOctaves)
validateattributes(numOctaves, {'numeric'}, {'integer',...
        'nonsparse', 'real', 'scalar', 'positive'},
'detectSURFFeatures'); %#ok<EMCA>
tf = true;

%=========================================================
================
function tf = checkNumScaleLevels(scales)
validateattributes(scales, {'numeric'}, {'integer',...
        'nonsparse', 'real', 'scalar', '>=', 3},
'detectSURFFeatures'); %#ok<EMCA>
tf = true;
```

OUTPUT



QUESTION - 2

Digit Classification using the HOG Feature

Theory

The histogram of oriented gradients (HOG) is a feature descriptor used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in localized portions of an image. This method is similar to that of edge orientation histograms, scale-invariant feature transform descriptors, and shape contexts, but differs in that it is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy.

CODE

```
% Load training and test data using |imageDatastore|.
syntheticDir      = fullfile(toolboxdir('vision'),
'visiondata','digits','synthetic');
handwrittenDir = fullfile(toolboxdir('vision'),
'visiondata','digits','handwritten');


% |imageDatastore| recursively scans the directory tree containing the
% images. Folder names are automatically used as labels for each image.
trainingSet = imageDatastore(syntheticDir,      'IncludeSubfolders', true,
'LabelSource', 'foldernames');
testSet         = imageDatastore(handwrittenDir, 'IncludeSubfolders', true,
'LabelSource', 'foldernames');
countEachLabel(trainingSet)
countEachLabel(testSet)
figure;


subplot(2,3,1);
imshow(trainingSet.Files{102});


subplot(2,3,2);
imshow(trainingSet.Files{304});


subplot(2,3,3);
imshow(trainingSet.Files{809});


subplot(2,3,4);
imshow(testSet.Files{13});


subplot(2,3,5);
imshow(testSet.Files{37});


subplot(2,3,6);
imshow(testSet.Files{97});
```

```
% Show pre-processing results
exTestImage = readimage(testSet,37);
processedImage = imbinarize(rgb2gray(exTestImage));

figure;

subplot(1,2,1)
imshow(exTestImage)

subplot(1,2,2)
imshow(processedImage)
```

OUTPUT

QUESTION - 3

Find corresponding interest points between the pairs of images

Theory

The Harris Corner Detector is a mathematical operator that finds features in an image. It is simple to compute, and is fast enough to work on computers. Also, it is popular because it is rotation, scale and illumination variation independent. However, the Shi-Tomasi corner detector, the one implemented in Open-CV, is an improvement of this corner detector.

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,y+v) - I(x,y)]^2$$

- E is the difference between the original and the moved window.
- u is the window's displacement in the x direction
- v is the window's displacement in the y direction
- w(x, y) is the window at position (x, y). This acts like a mask. Ensuring that only the desired window is used.
- I is the intensity of the image at a position (x, y)
- I(x+u, y+v) is the intensity of the moved window
- I(x, y) is the intensity of the original

FUNCTION

```
function pts = detectHarrisFeatures(I,varargin)
% detectHarrisFeatures Find corners using the Harris-Stephens algorithm
%     points = detectHarrisFeatures(I) returns a cornerPoints object,
%     points, containing information about the feature points detected in
a
```

```
%       2-D grayscale image I. detectHarrisFeatures uses the
Harris-Stephens
%       algorithm to find feature points.
%
%       points = detectHarrisFeatures(I,Name,Value) specifies additional
%       name-value pair arguments described below:
%
%       'MinQuality'    A scalar Q, 0 <= Q <= 1, specifying the minimum
accepted
%                       quality of corners as a fraction of the maximum
corner
%                       metric value in the image. Larger values of Q can
be used
%                       to remove erroneous corners.
%
%                       Default: 0.01
%
%       'FilterSize'    An odd integer, S >= 3, specifying a Gaussian filter
%                       which is used to smooth the gradient of the
image.
%                       The size of the filter is S-by-S and the standard
%                       deviation of the filter is (S/3).
%
%                       Default: 5
%
%       'ROI'               A vector of the format [X Y WIDTH HEIGHT],
specifying
%                       a rectangular region in which corners will be
detected.
%                       [X Y] is the upper left corner of the region.
%
%                       Default: [1 1 size(I,2) size(I,1)]
%
% Class Support
% -------------
% The input image I can be logical, uint8, int16, uint16, single, or
% double, and it must be real and nonsparse.
%
% Example
% -------
```

```
% % Find and plot corner points in an image.
% I = imread('cameraman.tif');
% corners = detectHarrisFeatures(I);
% imshow(I); hold on;
% plot(corners.selectStrongest(50));
%
% See also cornerPoints, detectMinEigenFeatures, detectFASTFeatures,
%                  detectBRISKFeatures, detectSURFFeatures,
detectMSERFeatures,
%                  extractFeatures, matchFeatures

% Reference
% ---------
% C. Harris and M. Stephens. "A Combined Corner and Edge Detector."
% Proceedings of the 4th Alvey Vision Conference. August 1988, pp.
147-151.

% Copyright    The MathWorks, Inc.

%#codegen
pts = vision.internal.detector.harrisMinEigen('Harris', I, varargin{:});


function hImage = showMatchedFeatures(I1, I2, matchedPoints1,
matchedPoints2, varargin)
%showMatchedFeatures Display corresponding feature points.
%    showMatchedFeatures(I1, I2, matchedPoints1, matchedPoints2)
displays a
%    falsecolor overlay of images I1 and I2, with a color-coded plot of the
%    corresponding points connected by a line. matchedPoints1 and
%    matchedPoints2 are the coordinates of corresponding points in I1
and
%    I2. Points can be an M-by-2 matrix of [x y] coordinates, a SURFPoints
%    an MSERRegions, a cornerPoints, or a BRISKPoints object.
%
%    showMatchedFeatures(I1, I2, matchedPoints1, matchedPoints2,
method)
%    displays images I1 and I2 using the visualization style specified by
%    method. Values of method can be:
%
```

```
%       'falsecolor' : Overlay the images by creating a composite red-cyan
%                              image showing I1 as red and I2 as cyan.
%       'blend'           : Overlay I1 and I2 using alpha blending.
%       'montage'        : Place I1 and I2 next to each other in the same
image.
%
%       Default: 'falsecolor'
%
%    hImage = showMatchedFeatures(...) returns the handle to the image
object
%    returned by showMatchedFeatures.
%
%    showMatchedFeatures(...,Name,Value) specifies additional
name-value pair
%    arguments described below:
%
%    'PlotOptions'    Specify custom plot options in a cell array containing
%                           three string values, {MarkerStyle1, MarkerStyle2,
LineStyle},
%                           corresponding to marker specification in I1,
marker
%                           specification in I2, and line style and color. Each of
%                           the specifiers is defined by the <a
href="matlab:doc('linespec')">LineSpec</a> of PLOT function.
%
%                           Default: {'ro','g+','y-'}
%
%     'Parent'          Specify an output axes for displaying the
visualization.
%
%    Class Support
%    -------------
%    I1 and I2 are numeric arrays.
%
%    Example 1
%    ---------
%    % Use Harris features to find corresponding points between two
images.
%    I1 = rgb2gray(imread('parkinglot_left.png'));
%    I2 = rgb2gray(imread('parkinglot_right.png'));
```

```
%
%    points1 = detectHarrisFeatures(I1);
%    points2 = detectHarrisFeatures(I2);
%
%    [f1, vpts1] = extractFeatures(I1, points1);
%    [f2, vpts2] = extractFeatures(I2, points2);
%
%    indexPairs = matchFeatures(f1, f2) ;
%    matchedPoints1 = vpts1(indexPairs(1:20, 1));
%    matchedPoints2 = vpts2(indexPairs(1:20, 2));
%
%    % Visualize putative matches
%    figure;
showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2,'montage')
;
%
%    title('Putative point matches');
%    legend('matchedPts1','matchedPts2');
%
%    Example 2
%    ---------
%    % Use SURF features to find corresponding points between two
images
%    % rotated and scaled with respect to each other
%    I1 = imread('cameraman.tif');
%    I2 = imresize(imrotate(I1,-20), 1.2);
%
%    points1 = detectSURFFeatures(I1);
%    points2 = detectSURFFeatures(I2);
%
%    [f1, vpts1] = extractFeatures(I1, points1);
%    [f2, vpts2] = extractFeatures(I2, points2);
%
%    indexPairs = matchFeatures(f1, f2) ;
%    matchedPoints1 = vpts1(indexPairs(:, 1));
%    matchedPoints2 = vpts2(indexPairs(:, 2));
%
%    % Visualize putative matches
%    figure;
showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2);
```

```
%
%    title('Putative point matches');
%    legend('matchedPts1','matchedPts2');
%
% See also matchFeatures, estimateGeometricTransform, imshowpair,
%          legend, SURFPoints, MSERRegions, cornerPoints

% Copyright 2011 The MathWorks, Inc.

narginchk(4,7);

[matchedPoints1, matchedPoints2, method, lineSpec, hAxes] = ...
       parseInputs(I1, I2, matchedPoints1, matchedPoints2, varargin{:});

% pad the smaller image
paddedSize = [max(size(I1,1), size(I2,1)), max(size(I1,2), size(I2,2))];
I1pad = [paddedSize(1) - size(I1,1), paddedSize(2) - size(I1,2)];
I2pad = [paddedSize(1) - size(I2,1), paddedSize(2) - size(I2,2)];
I1pre = round(I1pad/2);
I2pre = round(I2pad/2);
I1 = padarray(I1, I1pre, 0, 'pre');
I2 = padarray(I2, I2pre, 0, 'pre');
I1 = padarray(I1, I1pad-I1pre, 0, 'post');
I2 = padarray(I2, I2pad-I2pre, 0, 'post');

switch lower(method)
       case {'falsecolor'}
              imgOverlay = imfuse(I1, I2);
              % create red-cyan image instead of the imfuse default
              imgOverlay(:,:,1) = imgOverlay(:,:,2);
              imgOverlay(:,:,2) = imgOverlay(:,:,3);
       case {'blend'}
              imgOverlay = imfuse(I1,I2,'blend');
       case {'montage'}
              imgOverlay = imfuse(I1,I2,'montage');
end

% Display the composite image
if nargout > 0
       hImage = imshow(imgOverlay, 'Parent', hAxes);
```

```
else
    imshow(imgOverlay, 'Parent', hAxes);
end

holdState = get(hAxes,'NextPlot'); % store the state for 'hold' before
changing it
set(hAxes, 'NextPlot', 'add');

%============
% Plot points
%============
% Calculate the offsets needed to adjust plot after images were fused
offset1 = fliplr(I1pre);
offset2 = fliplr(I2pre);
if strcmp(method, 'montage')
    offset2 = offset2 + fliplr([0 size(I1,2)]);
end

matchedPoints1 = bsxfun(@plus, matchedPoints1, offset1);
matchedPoints2 = bsxfun(@plus, matchedPoints2, offset2);

if ~isempty(lineSpec{1})
    plot(hAxes, matchedPoints1(:,1), matchedPoints1(:,2),
lineSpec{1}); % marker 1
end
if ~isempty(lineSpec{2})
    plot(hAxes, matchedPoints2(:,1), matchedPoints2(:,2),
lineSpec{2}); % marker 2
end

% Plot by using a single line object with line segments broken by using
% NaNs. This is more efficient and makes it easier to customize the lines.
lineX = [matchedPoints1(:,1)'; matchedPoints2(:,1)'];
numPts = numel(lineX);
lineX = [lineX; NaN(1,numPts/2)];

lineY = [matchedPoints1(:,2)'; matchedPoints2(:,2)'];
lineY = [lineY; NaN(1,numPts/2)];

plot(hAxes, lineX(:), lineY(:), lineSpec{3}); % line
```

```
set(hAxes, 'NextPlot', holdState); % restore the hold state

drawnow();

%=========================================================
===============
% Input parser
%=========================================================
===============
function [matchedPoints1, matchedPoints2, method, lineSpec, hAxes]
= ...
        parseInputs(I1, I2, matchedPoints1, matchedPoints2, varargin)

% do only basic image validation; let padarray and imfuse take care of
% the rest
validateattributes(I1,{'numeric','logical'},{'real','nonsparse',...
        'nonempty'},mfilename,'I1',1)
validateattributes(I2,{'numeric','logical'},{'real','nonsparse',...
        'nonempty'},mfilename,'I2',2)

matchedPoints1 = parsePoints(matchedPoints1, 1);
matchedPoints2 = parsePoints(matchedPoints2, 2);

if size(matchedPoints1,1) ~= size(matchedPoints2,1)
        error(message('vision:showMatchedFeatures:numPtsMustMatch'));
end

% Process the rest of inputs
parser = inputParser;
parser.FunctionName    = mfilename;

parser.addOptional('Method', 'falsecolor', @checkMethod);
parser.addParameter('PlotOptions', {'ro','g+','y-'}, @checkPlotOptions);
parser.addParameter('Parent', [], ...
        @vision.internal.inputValidation.validateAxesHandle);

% Parse inputs
parser.parse(varargin{:});
```

```
% Calling validatestring again permits easy handling of partial string
matches
method = validatestring(parser.Results.Method,...
    {'falsecolor','blend','montage'},mfilename,'Method');

lineSpec = parser.Results.PlotOptions;

hAxes = newplot(parser.Results.Parent);

%=========================================================
================
function points=parsePoints(points, ptsInputNumber)

fcnInputVarNumber = 2 + ptsInputNumber;
varName = ['matchedPoints', num2str(ptsInputNumber)];

if ~isa(points, 'vision.internal.FeaturePoints') && ~isa(points,
'MSERRegions')
    validateattributes(points,{'int16', 'uint16', 'int32', 'uint32', ...
        'single', 'double'}, {'2d', 'nonsparse', 'real', 'size', [NaN 2]},...
        mfilename, varName, fcnInputVarNumber);
else
    points = points.Location;
end

points = double(points);

%=========================================================
================
function tf = checkMethod(method)

validatestring(method,{'falsecolor','blend','montage'},mfilename,'Metho
d');

tf = true;

%=========================================================
================
function tf = checkPlotOptions(options)
```

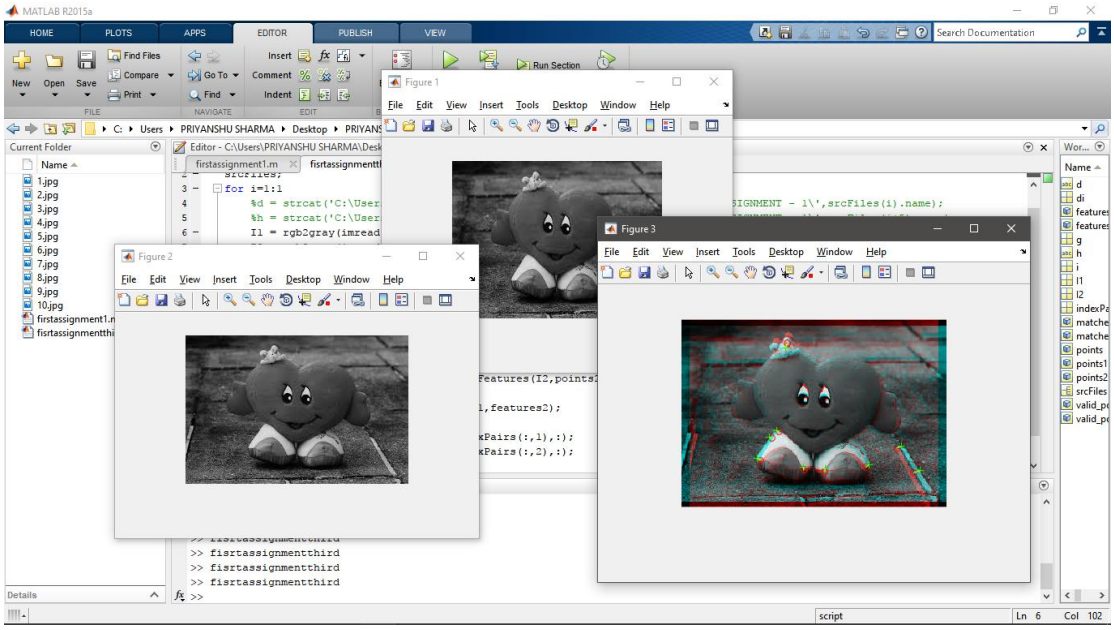
```
validateattributes(options,{'cell'}, {'size', [1 3]},...
        mfilename, 'PlotOptions');

validateattributes(options{1},{'char'},{},mfilename,'MarkerStyle1');
validateattributes(options{2},{'char'},{},mfilename,'MarkerStyle2');
validateattributes(options{3},{'char'},{},mfilename,'LineStyle');

% Now check valid strings
checkMarkerStyle(options{1}, 1);
checkMarkerStyle(options{2}, 2);

checkLineStyle(options{3});

tf = true;

%=========================================================
================
function style=eliminateColorSpec(style)

colorSpec = cell2mat({'r','g','b','c','m','y','k','w'});

% Color can be specified only at the beginning or end of the style string.
% Look for only one specifier. If color was specified twice, it will cause
% a failure in later stages of parsing
if ~isempty(style)
        if isempty(strfind(colorSpec, style(1)))
                % try the other end
                if ~isempty(strfind(colorSpec, style(end)))
                        style(end) = [];
                end
        else
                style(1) = [];
        end
end

%=========================================================
================
function checkMarkerStyle(style, id)

style = strtrim(style); % remove blanks from either end of the string
```

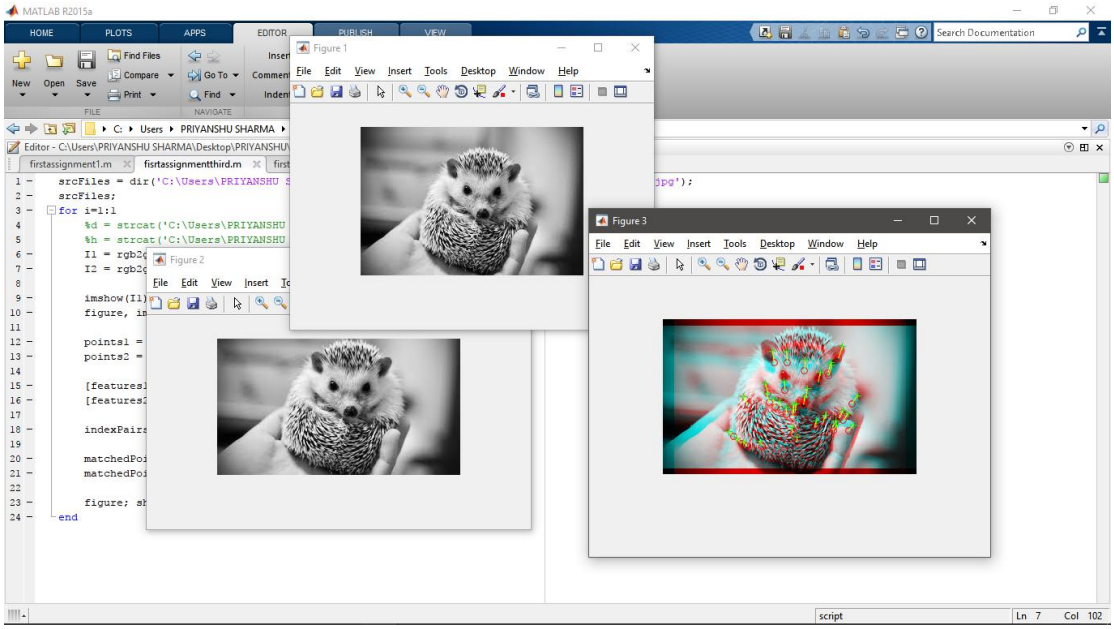
```
style = strtrim(eliminateColorSpec(style)); % pull out valid color spec

if isempty(style)
    % permit empty marker style, which amounts to marker not being
displayed
else
    markerSpec = {'+','o','*','.','x','square','s','diamond','d','^',...
        'v','>','<','pentagram','p','hexagram','hImage'};

    try
        validatestring(style,markerSpec);
    catch %#ok<CTCH>

error(message('vision:showMatchedFeatures:invalidMarkerStyle',id));
    end
end

%=========================================================
================
function checkLineStyle(style)

style = strtrim(style); % remove blanks from either end of the string
style = strtrim(eliminateColorSpec(style)); % pull out valid color spec

if isempty(style)
    % permit empty line style thus letting plot use its default settings
else
    lineSpec = {'-','--',':','-.'};

    try
        validatestring(style,lineSpec);
    catch %#ok<CTCH>

error(message('vision:showMatchedFeatures:invalidLineStyle'));
    end
end
```

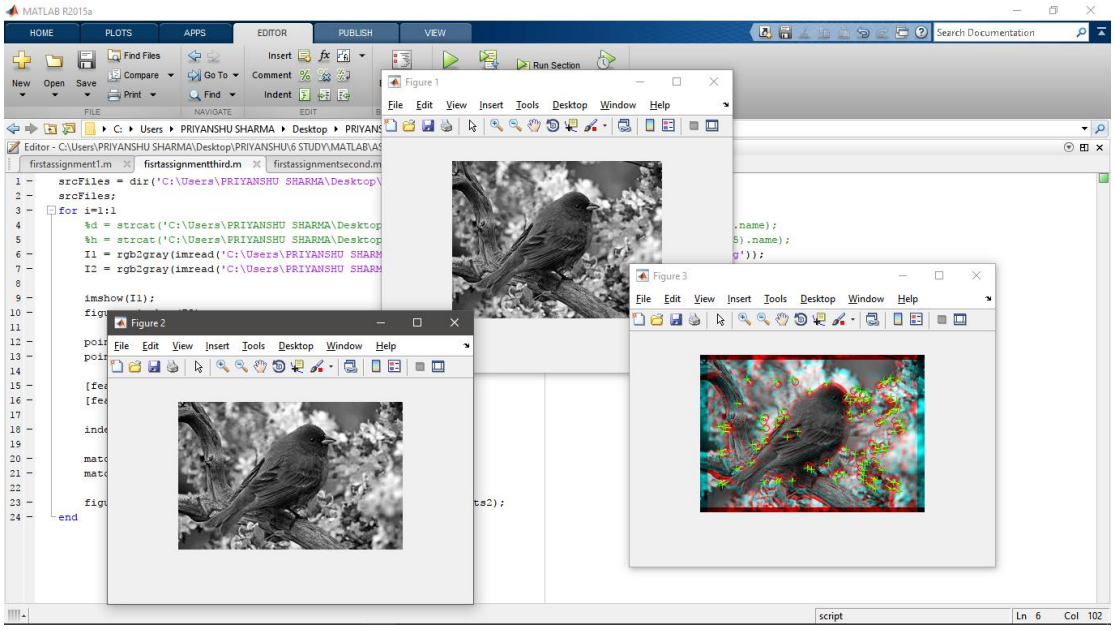
Code

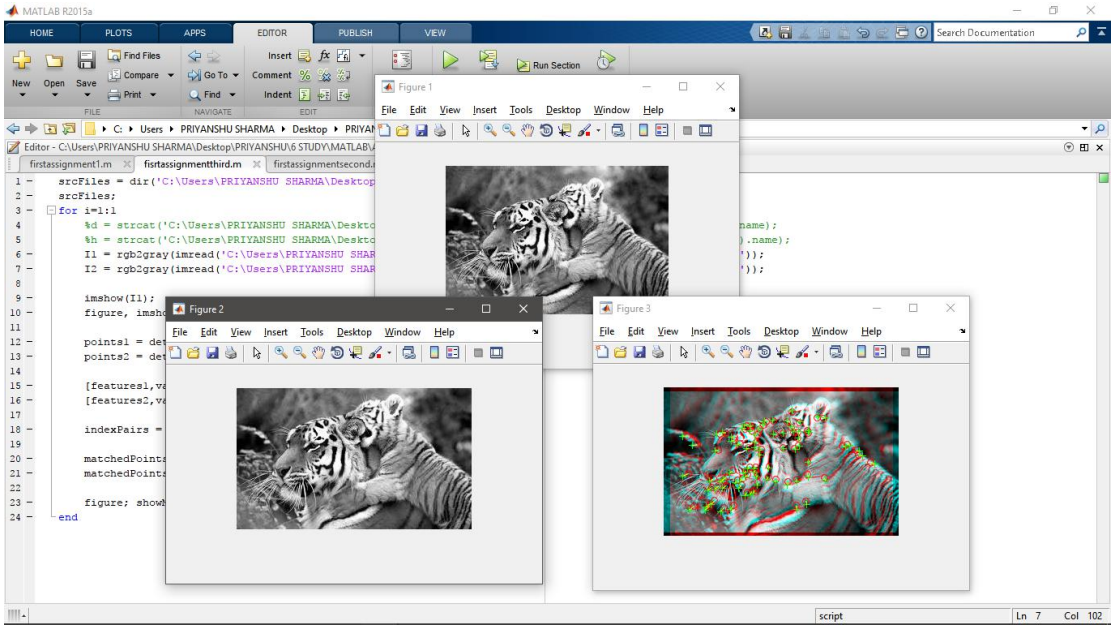
```
srcFiles = dir('C:\Users\PRIYANSHU SHARMA\Desktop\PRIYANSHU\6
STUDY\MATLAB\ASSIGNMENT - 1\*.jpg');
srcFiles;
for i=1:1

    I1 = rgb2gray(imread('C:\Users\PRIYANSHU SHARMA\Desktop\PRIYANSHU\6
STUDY\MATLAB\ASSIGNMENT - 1\5.jpg'));
    I2 = rgb2gray(imread('C:\Users\PRIYANSHU SHARMA\Desktop\PRIYANSHU\6
STUDY\MATLAB\ASSIGNMENT - 1\10.jpg'));

    imshow(I1);
    figure, imshow(I2);

    points1 = detectHarrisFeatures(I1);
    points2 = detectHarrisFeatures(I2);

    [features1,valid_points1] = extractFeatures(I1,points1);
    [features2,valid_points2] = extractFeatures(I2,points2);

    indexPairs = matchFeatures(features1,features2);

    matchedPoints1 = valid_points1(indexPairs(:,1),:);
    matchedPoints2 = valid_points2(indexPairs(:,2),:);

    figure; showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2);
end
```

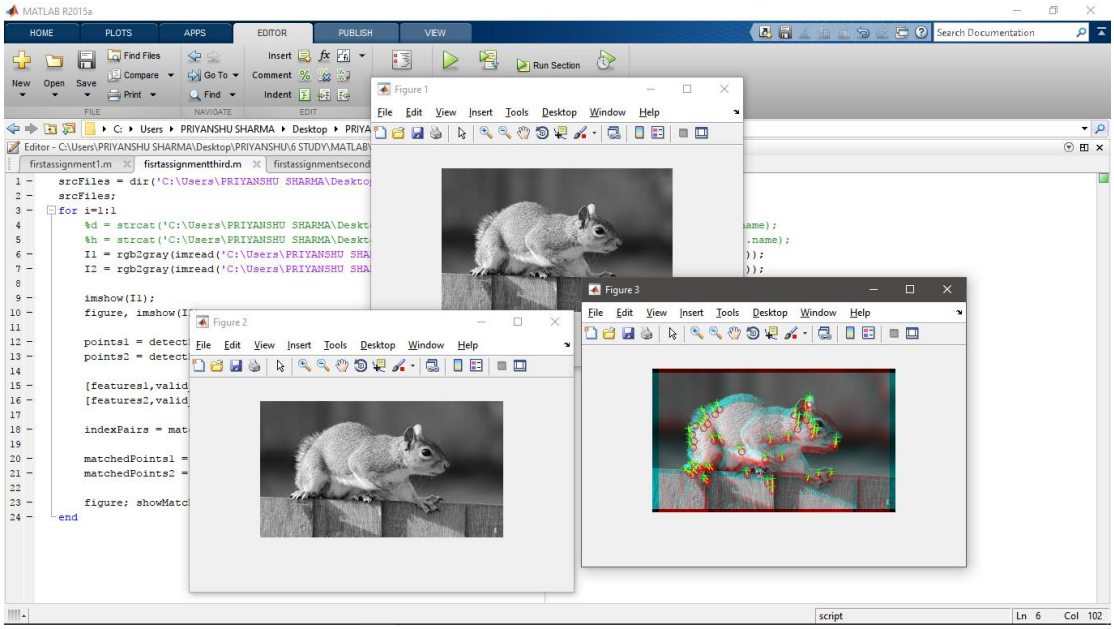
IMAGES 1


MATLAB R2015a
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
Figure 1
Figure 2
Figure 3
File Edit View Insert Tools Desktop Window Help
>> fisrtassignmentthird
>> fisrtassignmentthird
>> fisrtassignmentthird
>>
script
Ln 6
Col 102


MATLAB R2015a
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
firstassignment1.m
fisrtassignmentthird.m
Figure 1
Figure 2
Figure 3
File Edit View Insert Tools Desktop Window Help
srcFiles = dir('C:\Users\PRIYANSHU
srcFiles;
for i=1:1
%d = strcat('C:\Users\PRIYANSHU
%h = strcat('C:\Users\PRIYANSHU
I1 = rgb2
I2 = rgb2
imshow(I1)
figure, im
points1 =
points2 =
[features1
[features2
indexPairs
matchedPoi
matchedPoi
figure; sh
end
script
Ln 7
Col 102


MATLAB R2015a
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
New
Open
Save
Find Files
Compare
Print
Go To
Find
Insert
Comment
Indent
Run Section
FILE
NAVIGATE
EDIT
Figure 1
Figure 2
Figure 3
File Edit View Insert Tools Desktop Window Help
Editor - C:\Users\PRIYANSHU SHARMA\Desktop\PRIYANSHU\6 STUDY\MATLAB\
firstassignment1.m
fisrtassignmentthird.m
firstassignmentsecond.m
srcFiles = dir('C:\Users\PRIYANSHU SHARMA\Desktop\
srcFiles;
for i=1:1
%d = strcat('C:\Users\PRIYANSHU SHARMA\Desktop
%h = strcat('C:\Users\PRIYANSHU SHARMA\Desktop
I1 = rgb2gray(imread('C:\Users\PRIYANSHU SHARM
I2 = rgb2gray(imread('C:\Users\PRIYANSHU SHARM
imshow(I1);
end
script
Ln 6
Col 102


MATLAB R2015a
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
New
Open
Save
Find Files
Compare
Print
Go To
Find
Insert
Comment
Indent
Run Section
FILE
NAVIGATE
EDIT
Figure 1
Figure 2
Figure 3
File Edit View Insert Tools Desktop Window Help
Editor - C:\Users\PRIYANSHU SHARMA\Desktop\PRIYANSHU\6 STUDY\MATLAB\
firstassignment1.m
fisrtassignmentthird.m
srcFiles = dir('C:\Users\PRIYANSHU SHARMA\Desktop
srcFiles;
for i=1:1
%d = strcat('C:\Users\PRIYANSHU SHARMA\Deskto
%h = strcat('C:\Users\PRIYANSHU SHARMA\Deskto
I1 = rgb2gray(imread('C:\Users\PRIYANSHU SHAR
I2 = rgb2gray(imread('C:\Users\PRIYANSHU SHAR
imshow(I1);
figure, imsho
points1 = det
points2 = det
[features1,va
[features2,va
indexPairs =
matchedPoints
matchedPoints
figure; showM
end
script
Ln 7
Col 102


MATLAB R2015a
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
New
Open
Save
Find Files
Compare
Print
Go To
Find
Insert
Comment
Indent
Run Section
FILE
NAVIGATE
EDIT
C: ▸ Users ▸ PRIYANSHU SHARMA ▸ Desktop ▸ PRIYA
Editor - C:\Users\PRIYANSHU SHARMA\Desktop\PRIYANSHU\6 STUDY\MATLAB\
firstassignment1.m
fisrtassignmentthird.m
firstassignmentsecond
1 – srcFiles = dir('C:\Users\PRIYANSHU SHARMA\Deskto
2 – srcFiles;
3 – for i=1:l
4 %d = strcat('C:\Users\PRIYANSHU SHARMA\Deskt
5 %h = strcat('C:\Users\PRIYANSHU SHARMA\Deskt
6 – I1 = rgb2gray(imread('C:\Users\PRIYANSHU SHA
7 – I2 = rgb2gray(imread('C:\Users\PRIYANSHU SHA
8
9 – imshow(I1);
10 – figure, imshow(I
11
12 – points1 = detect
13 – points2 = detect
14
15 – [features1,valid
16 – [features2,valid
17
18 – indexPairs = mat
19
20 – matchedPoints1 =
21 – matchedPoints2 =
22
23 – figure; showMatc
24 – end
Figure 1
File Edit View Insert Tools Desktop Window Help
Figure 2
File Edit View Insert Tools Desktop Window Help
Figure 3
File Edit View Insert Tools Desktop Window Help
script
Ln 6
Col 102