CSC 428 Computer Vision

Project 4

Team Name: 2020 Vision

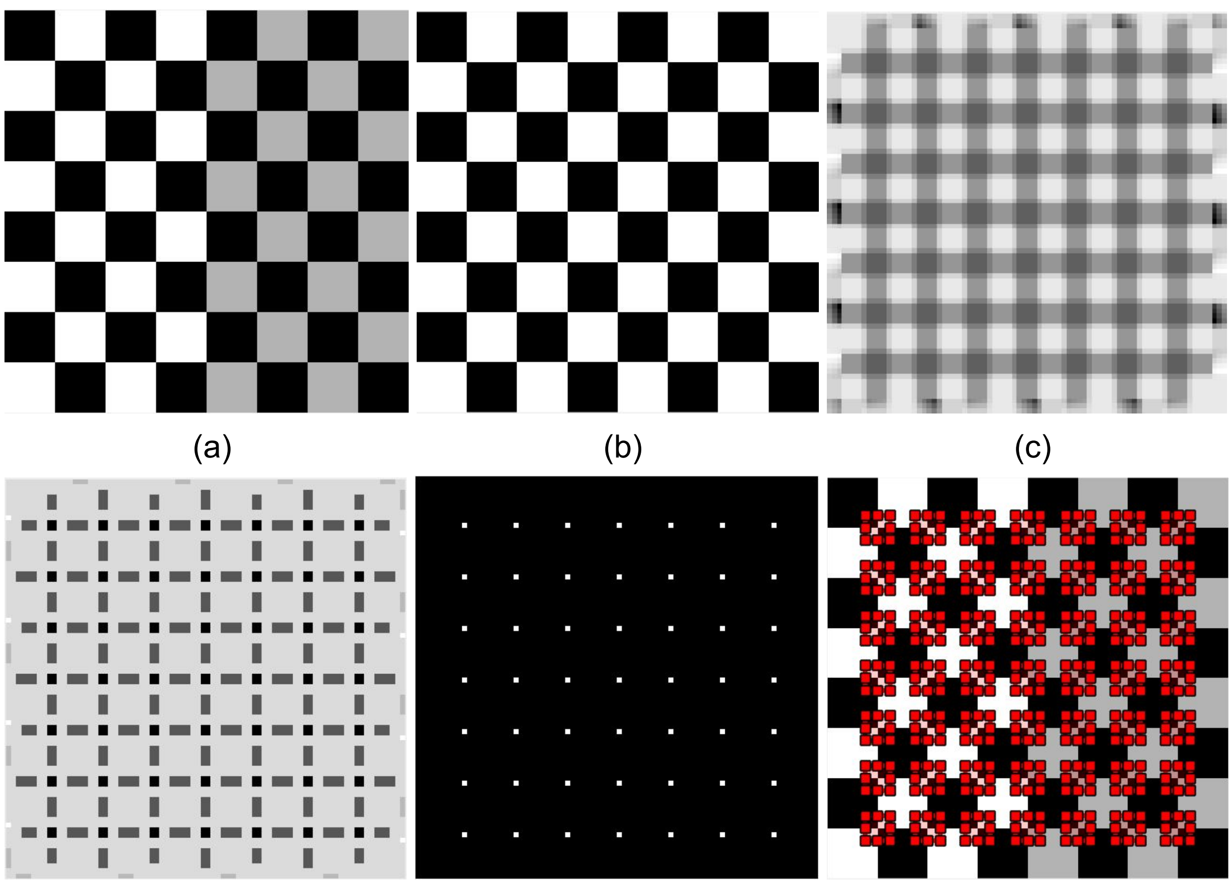
Danny Rivers, Neima Yeganeh

2/15/20

For this lab we used matlab to work on our feature detection and matching skills. This lab was broken down into three parts, Feature Detection, Feature Description, and Feature Matching. We broke this paper down into those sections below.

1. Feature Detection

For part 1 we worked on feature detection. In this part we implemented Harris corner detection and used it on the image shown in figure 1 (a). First we decided to turn the image into a binary image shown in figure 1 (b) using a threshold. This threshold was found using imhist to find the spikes in intensity corresponding to the grey area. Next we started to implement Harris corner detection. Next we computed the Gaussian derivative of every pixel. We did this by filter the image with the horizontal and vertical matrices. Next with these filtered images we compute the second moment matrix. Next we calculate the corner response function r, and calculate the local maxima. After that we use thresholding to get the perfect matches. Finally loop through all the peaks and draw the boxes around them using matlab's drawrectangles shown in figure 1 (f).



(a) (b) (c)

(d) (e) (f)

Figure 1: Checkerboard Image using Harris Corner Detection

(a) Original Checkerboard Image
(b) Checkerboard with Threshold
(c) Determinant and Trace Applied
(d) Local Maximum Applied
(e) Duplicates removed
(f) Bounding Boxes on Original Image

2. **Feature Description**

For part 2 we used Matlba's built in FAST function to detect the feature points. Then we implemented our own extractFeatures function to determine the feature descriptor for each keypoint. For the first extract features function, we grabbed the raw pixel data of a 5 by 5 matrix around the keypoint and used that as the descriptor. For the second function, we made a SIFT-like feature descriptor algorithm that returned the gradient with the directions. This SIFT-like algorithm used a 16 by 16 matrix and computes the directions in a 4 by 4 grid where each entry has a 4 by 4 pixel grid. This larger matrix with the gradient proved to be a much better feature descriptor than the 5 by 5 raw pixel data due to variations in brightness and contrast.

3. **Feature Matching**

For part 3 we continue working on feature matching. We first wrote a function to compute the distance between two key points for part a. For this we used euclidean distance and used the formula

$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

We implemented the formula in a function called euclideanDistance which took in the two vectors representing the key points and returned the distance between them.

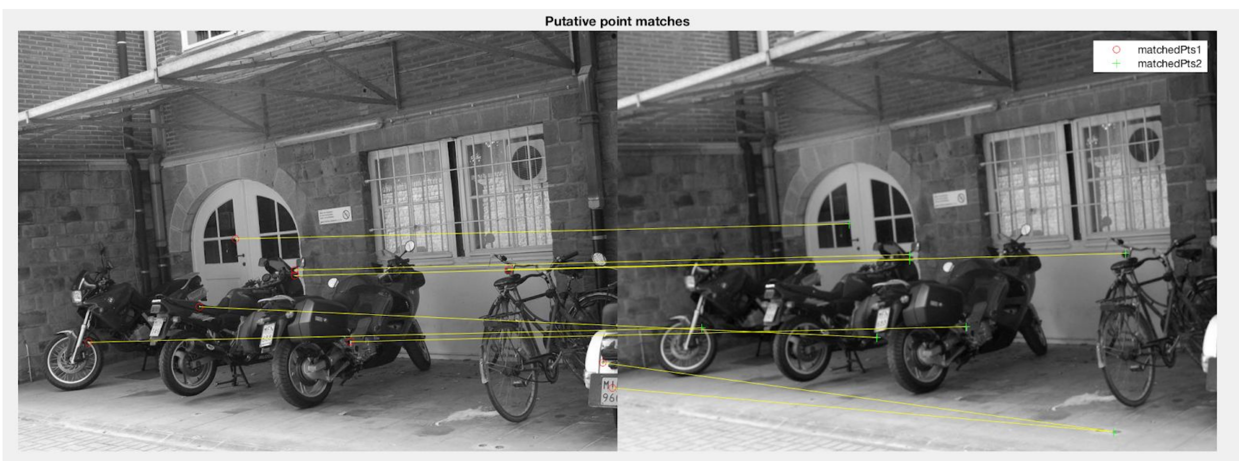
Next for part b, c and d we implemented feature matching using an algorithm we talked about in class and our euclidean distance formula. This function named featureMatching takes in two images and two thresholds. It uses both of our extractFeatures functions from part 2 to extract features then call our matchF function to match the features. This function is run twice, once for each extractFeatures function, and takes in one of the initial thresholds each. This function loops through each detected feature in one image and finds the two smallest distance partners in the corresponding feature list. It then checks the ratio of the two distances against the threshold parameter and if its good it is kept as a match. After featureMating calls matchF it then takes the matched pairs and displays them with showMatchedFeatures.

Figure 2: Grayscale Images and Feature Matching

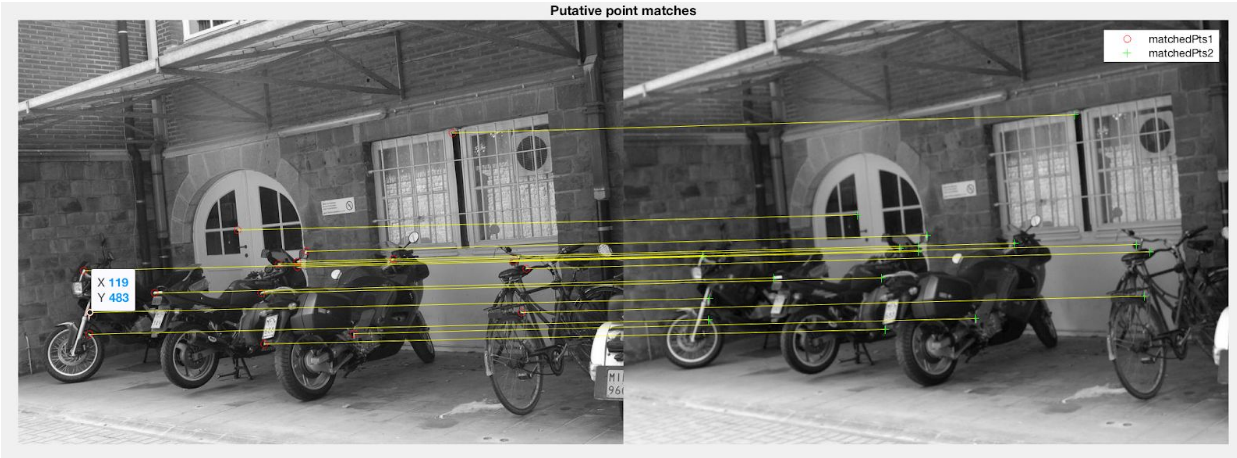
(a) Grayscale Bike Image 1
(b) Grayscale Bike Image 2
(c) Feature matching Bike Image 1
(d) Feature matching Bike Image 2
(e) Grayscale Car Image 1
(f) Grayscale Car Image 2
(g) Feature matching Car Image 1
(h) Feature matching Car Image 2
(i) Grayscale Wall Image 1
(j) Grayscale Wall Image 2
(k) Feature matching Wall Image 1
(l) Feature matching Wall Image 2

(a)

(b)



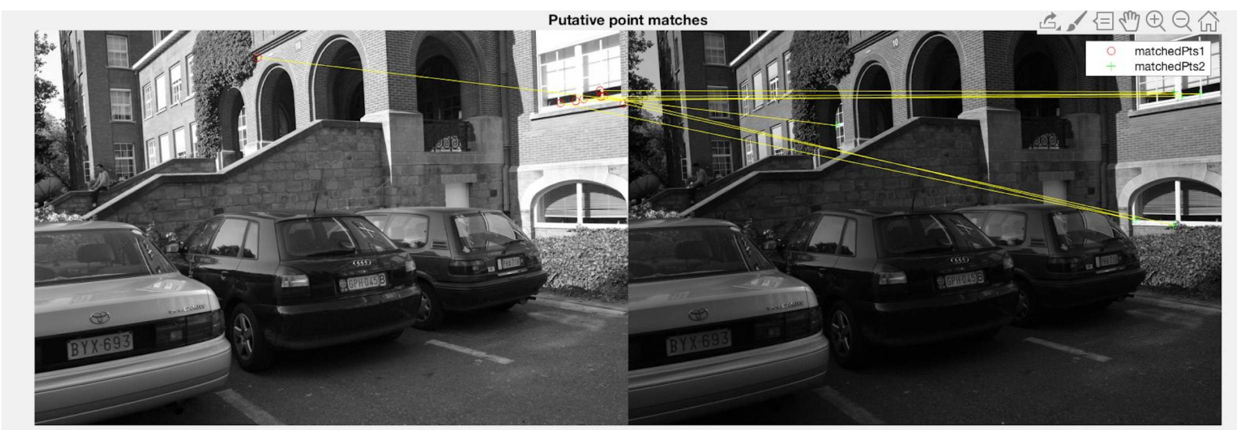
(c): Feature matching using my_extractFeatures_a



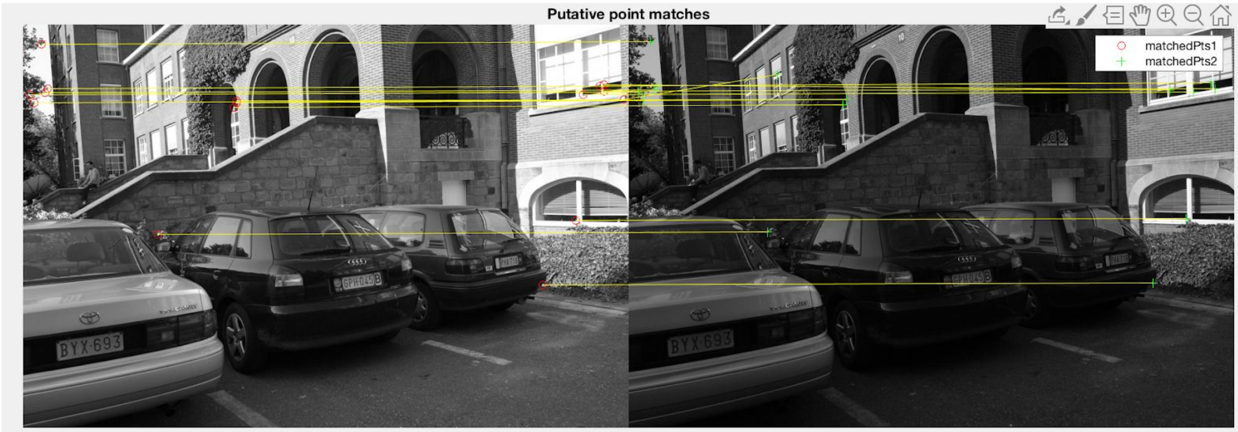
(d): Feature matching using my_extractFeatures_b

(e)

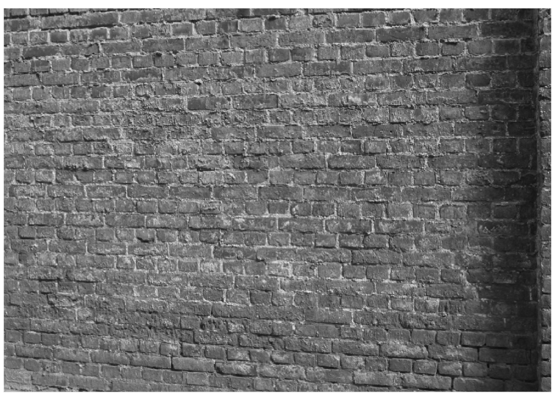
(f)



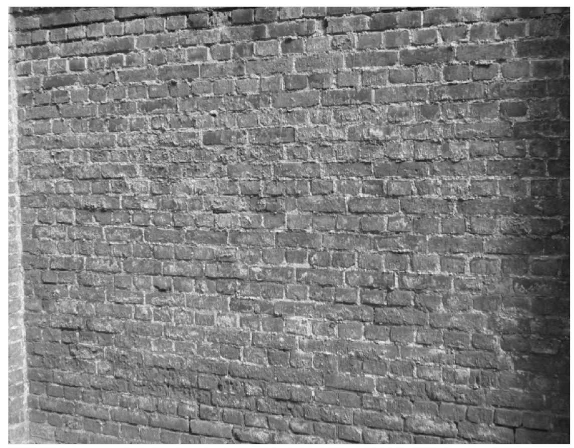
(g): Feature matching using my_extractFeatures_a



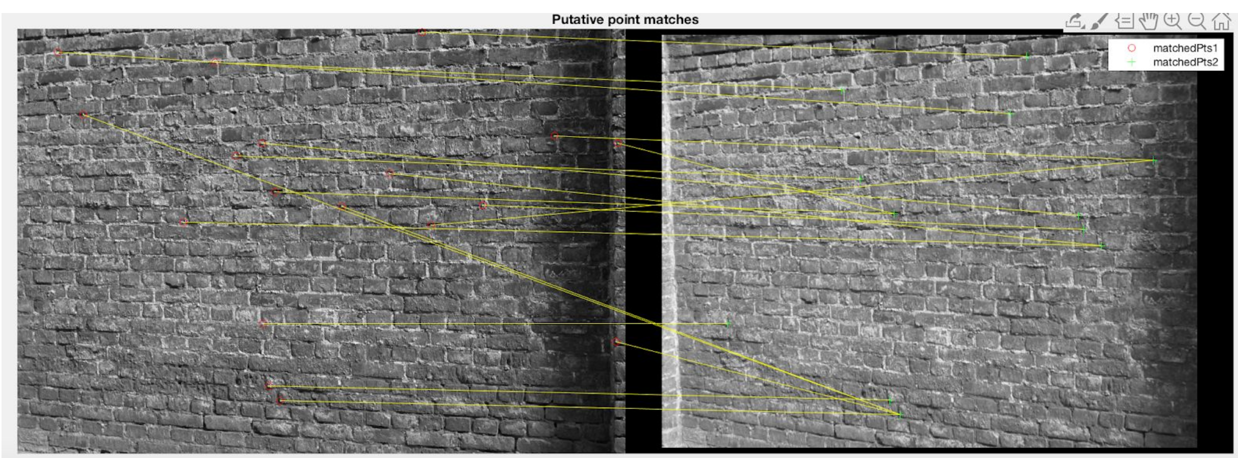
(h): Feature matching using my_extractFeatures_b



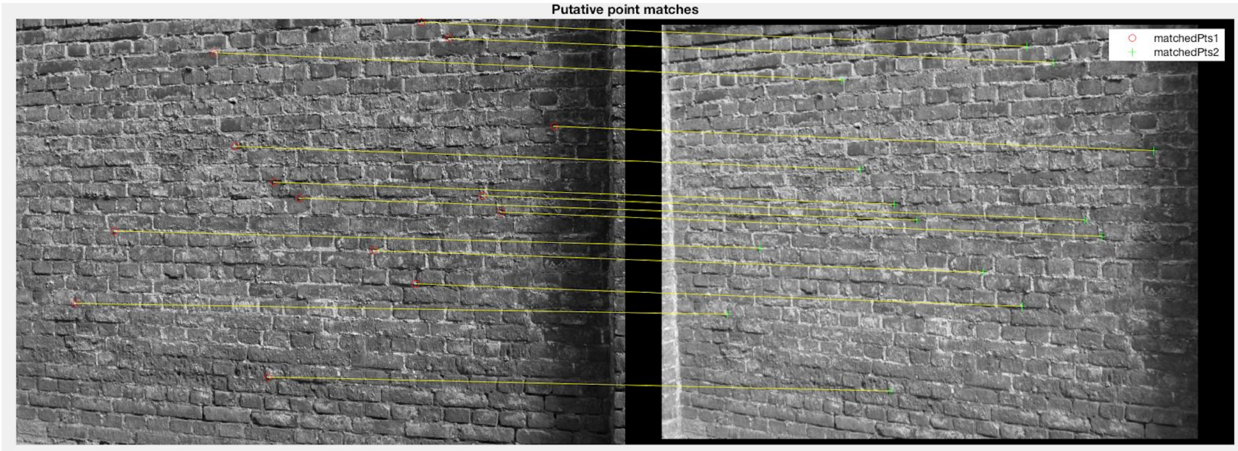
(i)



(j)



(k): Feature matching using my_extractFeatures_a



(l)**:** Feature matching using my_extractFeatures_b

What We Learned:

“I learned a lot about feature matching in this assignment. I learned that key points are important when implementing feature matching and that using algorithms like sift are more tolerant to pictures of things with different shading. This project was fun and very informative.” - Neima Yeganeh

“This lab was useful in understanding various feature detection algorithms and the descriptors necessary to identify them in two separate images. I learned alot about the feature descriptors and how to implement them. The gradient implementation with the angles was better than taking just the raw pixel data. I’ll look into other algorithms similar to SIFT that have less overhead with the computation.” - Danny Rivers

Appendix

```
% Part 1
im = checkerboard;
imshow(im)

% apply threshold
im_thresh = 0.6*max(im(:));
imT = double(im>im_thresh);
imshow(imT)

Sx = [-1 -1 -1; 0 0 0; 1 1 1];
Sy = Sx';

Ix = imfilter(imT, Sx);
Iy = imfilter(imT, Sy);

Ixx = Ix.*Ix;
Ixy = Ix.*Iy;
Iyy = Iy.*Iy;
```

```
N = 5; W = ones(N);
M11 = imfilter(Ixx, W);
M12 = imfilter(Ixy, W);
M22 = imfilter(Iyy, W);

alpha = 0.05;
detM = M11.*M12 - M12.^2;
traceM = M11 + M22;
R = detM - alpha*traceM.^2

% R = abs(R)

figure, imshow(R, [])

% imS = size(R);
% T = ones(imS);
% for i = 1:imS(1)
%     for j = 1:imS(2)
%         T(i,j) = (R(i,j)-min(R(:)))*(255)/(max(R(:))-min(R(:)));
%     end
% end
%
% figure, imshow(T)
%
% pause
R = localMaximum(R);
figure, imshow(R, [])

% R_thresh = mean(R(:));
% Lmax = (R == imdilate(R, strel('disk', 11)) & R>R_thresh);
% [rows, cols] = find(Lmax);

Rthreshold = 0.8*min(R(:));
T = R<Rthreshold;
figure, imshow(T)

T = removeDuplicates(T);
imshow(T)
```

```
[ypeak, xpeak] = find(T==1);

figure
hAx  = axes;
imshow(im,'Parent', hAx), hold on

for i = 1:size(ypeak)
    % Compute translation from max location in correlation matrix
    yoffSet = ypeak(i)-2;
    xoffSet = xpeak(i)-2;

    % Display matched area
    drawrectangle(hAx, 'Position', [xoffSet, yoffSet, 5, 5], 'Color', 'r');

end

hold off

% [rows, cols] = find(1);

function I2 = localMaximum(I)
    ss = size(I);
    height = ss(1);
    width = ss(2);

    hSize = 3;
    takeaway = hSize - 1;
    I2 = zeros(height-takeaway, width-takeaway);

    for i = 1:height-takeaway
        for j = 1:width-takeaway

            pixel = I(i+(takeaway/2), j+(takeaway/2));
            tmp = zeros(hSize, hSize);
            for k = 1:hSize*hSize
                tmp(mod((k-1), hSize) +1, floor((k-1)/hSize) +1) = I((i) + mod((k-1), hSize) , (j)
+ floor((k-1)/hSize));
            end
```

```
            if pixel == max(tmp(:))
                I2(i,j) = pixel;
            else
                I2(i,j) = 0;
            end
        end
    end
%     I2 = uint8(I2);
end

function I2 = removeDuplicates(I)
    ss = size(I);
    height = ss(1);
    width = ss(2);

    hSize = 3;
    takeaway = hSize - 1;
    I2 = I;

    for i = 1:height-takeaway
        for j = 1:width-takeaway

            pixel = I(i+(takeaway/2), j+(takeaway/2));
            if pixel == 1
                for k = 1:hSize*hSize
                    I2((i) + mod((k-1), hSize) , (j) + floor((k-1)/hSize)) = 0;
                end
                I2(i+(takeaway/2), j+(takeaway/2)) = 1;
            end
        end
    end
%     I2 = uint8(I2);
end

% Part 2

% Load images
b1   = rgb2gray(imread('bikes1.ppm'));
b2   = rgb2gray(imread('bikes2.ppm'));
```

```
% Show the images
imshow(b1)
imshow(b2)

% Detected keypoints using FAST
detected_pts1 = detectFASTFeatures(b1);
detected_pts1 = detected_pts1.selectStrongest(50);

detected_pts2 = detectFASTFeatures(b2);
detected_pts2 = detected_pts2.selectStrongest(50);

% Show detected points
imshow(b1); hold on;
plot(detected_pts1); hold off;

imshow(b2); hold on;
plot(detected_pts2); hold off;

% Extract features using raw data
[extracted_features1a] = my_extractFeatures_a(b1, detected_pts1);
[extracted_features2a] = my_extractFeatures_a(b2, detected_pts2);

% NEEDS WORK: Extract features using SIFT-like feature descriptor
[extracted_features1b] = my_extractFeatures_b(b1, detected_pts1);
[extracted_features2b] = my_extractFeatures_b(b2, detected_pts2);


function [extracted_features] = my_extractFeatures_a(im, detected_pts)
    numPts = size(detected_pts);
    numPts = numPts(1);

    hSize = 5;
    hDiff = (hSize - 1) / 2;

    im = padarray(im,[hDiff hDiff],0,'both');

    [extracted_features] = zeros(numPts, hSize*hSize);
```

```
    for i = 1:numPts
        pt = detected_pts(i).Location;
        y = pt(1) + hDiff;
        x = pt(2) + hDiff;
        features = zeros(hSize*hSize, 1);
        for j = 1:hSize*hSize
            features(j, 1) = im((x) + mod((j-1), hSize) - hDiff, (y) + floor((j-1)/hSize) - hDiff);
        end
        extracted_features(i, :) = features;
    end
end

function [extracted_features] = my_extractFeatures_b(im, detected_pts)

    % Prewitt Operators
    Sx = [-1 -1 -1; 0 0 0; 1 1 1];
    Sy = Sx';

    numPts = size(detected_pts);
    numPts = numPts(1);

    hSize = 16;
    hDiff = (hSize) / 2;

    im = padarray(im,[hDiff hDiff],0,'both');

    [extracted_features] = zeros(numPts, hSize*hSize);

    for i = 1:numPts
        pt = detected_pts(i).Location;
        y = pt(1) + hDiff;
        x = pt(2) + hDiff;
        features = zeros(hSize, hSize);
        for j = 1:hSize*hSize
            features(mod((j-1), hSize) +1, floor((j-1)/hSize) +1) = im((x) + mod((j-1), hSize) -
hDiff, (y) + floor((j-1)/hSize) - hDiff);
        end

        % Apply the Prewitt Filter to the Image
```

```
        Gx = imfilter(double(features), Sx);
        Gy = imfilter(double(features), Sy);

        G = (Gx.^2 + Gy.^2).^0.5;

        extracted_features(i, :) = G(:);
    end
end

% Part 3

% Load bike images
b1   = rgb2gray(imread('bikes1.ppm'));
b2   = rgb2gray(imread('bikes2.ppm'));

featureMatching(b1, b2, 0.8, 0.8);

% Load car images
c1   = rgb2gray(imread('cars1.ppm'));
c2   = rgb2gray(imread('cars2.ppm'));

featureMatching(c1, c2, 0.8, 0.8);

% Load wall images
w1   = rgb2gray(imread('wall1.ppm'));
w2   = rgb2gray(imread('wall2.ppm'));

featureMatching(w1, w2, 0.8, 0.8);


% thresh1 is raw, thresh2 is SIFT
function featureMatching(b1, b2, thresh1, thresh2)
    % Show the images
    imshow(b1)
    imshow(b2)

    % Detected keypoints using FAST
    detected_pts1 = detectFASTFeatures(b1);
    detected_pts1 = detected_pts1.selectStrongest(50);
```

```
detected_pts2 = detectFASTFeatures(b2);
detected_pts2 = detected_pts2.selectStrongest(50);

% Show detected points
imshow(b1); hold on;
plot(detected_pts1); hold off;

imshow(b2); hold on;
plot(detected_pts2); hold off;

% Extract features using raw data
[extracted_features1a] = my_extractFeatures_a(b1, detected_pts1);
[extracted_features2a] = my_extractFeatures_a(b2, detected_pts2);

indexPairs = matchF(extracted_features1a, extracted_features2a, thresh1);
len = size(indexPairs);
len = len(1);

matchedPoints1 = detected_pts1(indexPairs(1:len, 1));
matchedPoints2 = detected_pts2(indexPairs(1:len, 2));

% Visualize putative matches
figure; ax = axes;

showMatchedFeatures(b1,b2,matchedPoints1,matchedPoints2,'montage','Parent',ax);

title(ax, 'Putative point matches');
legend(ax, 'matchedPts1','matchedPts2');

% NEEDS WORK: Extract features using SIFT-like feature descriptor
[extracted_features1b] = my_extractFeatures_b(b1, detected_pts1);
[extracted_features2b] = my_extractFeatures_b(b2, detected_pts2);

indexPairs = matchF(extracted_features1b, extracted_features2b, thresh2);
len = size(indexPairs);
len = len(1);

matchedPoints1 = detected_pts1(indexPairs(1:len, 1));
```

```
    matchedPoints2 = detected_pts2(indexPairs(1:len, 2));

    % Visualize putative matches
    figure; ax = axes;

showMatchedFeatures(b1,b2,matchedPoints1,matchedPoints2,'montage','Parent',ax);

    title(ax, 'Putative point matches');
    legend(ax, 'matchedPts1','matchedPts2');

end



function [extracted_features] = my_extractFeatures_a(im, detected_pts)
    numPts = size(detected_pts);
    numPts = numPts(1);

    hSize = 5;
    hDiff = (hSize - 1) / 2;

    im = padarray(im,[hDiff hDiff],0,'both');

    [extracted_features] = zeros(numPts, hSize*hSize);

    for i = 1:numPts
        pt = detected_pts(i).Location;
        y = pt(1) + hDiff;
        x = pt(2) + hDiff;
        features = zeros(hSize*hSize, 1);
        for j = 1:hSize*hSize
            features(j, 1) = im((x) + mod((j-1), hSize) - hDiff, (y) + floor((j-1)/hSize) - hDiff);
        end
        extracted_features(i, :) = features;
    end
end

function [extracted_features] = my_extractFeatures_b(im, detected_pts)
```

```
    % Prewitt Operators
    Sx = [-1 -1 -1; 0 0 0; 1 1 1];
    Sy = Sx';

    numPts = size(detected_pts);
    numPts = numPts(1);

    hSize = 16;
    hDiff = (hSize) / 2;

    im = padarray(im,[hDiff hDiff],0,'both');

    [extracted_features] = zeros(numPts, hSize*hSize);

    for i = 1:numPts
        pt = detected_pts(i).Location;
        y = pt(1) + hDiff;
        x = pt(2) + hDiff;
        features = zeros(hSize, hSize);
        for j = 1:hSize*hSize
            features(mod((j-1), hSize) +1, floor((j-1)/hSize) +1) = im((x) + mod((j-1), hSize) -
hDiff, (y) + floor((j-1)/hSize) - hDiff);
        end

        % Apply the Prewitt Filter to the Image
        Gx = imfilter(double(features), Sx);
        Gy = imfilter(double(features), Sy);

        G = (Gx.^2 + Gy.^2).^0.5;

        extracted_features(i, :) = G(:);
    end
end


function indexPairs = matchF(f1, f2, thresh)
    indexPairs = [];
    len1 = size(f1);
    len1 = len1(1);
```

```
    len2 = size(f2);
    len2 = len2(1);
    for fi = 1 : len1
        min1 = 0;
        min1v = [];
        min2 = 0;
        min2v = [];
        i1 = euclideanDistance(f1(fi, :), f2(1, :));
        i2 = euclideanDistance(f1(fi, :), f2(2, :));
        if (i1 >= i2)
            min1v = [fi 2];
            min1 = i2;
            min2v = [fi 1];
            min2 = i1;
        else
            min1 = i1;
            min1v = [fi 1];
            min2v = [fi 2];
            min2 = i2;
        end
        for gi = 3 : len2
            d = euclideanDistance(f1(fi, :), f2(gi, :));
            if d <= min1
                min2 = min1;
                min2v = min1v;
                min1v = [fi gi];
                min1 = d;
            elseif d <= min2
                min2 = d;
                min2v = [fi gi];
            end
        end
        rat = min1/min2;
        if rat < thresh
          indexPairs = [indexPairs; (min1v)];
        end
    end
end
```

```matlab
function a = euclideanDistance(vect1, vect2)
    a = (sum((vect1-vect2).^2)).^0.5;
end
```