Local features

Chao Ni

October 4, 2020

1 Detection

1.1 Image gradients

In MATLAB, the convolution is defined as: $C(j,k) = \sum_p \sum_q A(p,q)B(j-p+1, k-q+1)$ for $C = \text{conv}(A,B)$, where the base matrix B doesn't start from the middle. We adjust the base matrix such that the convolution is more intuitive.

```
[n,m] = size(img);
% construct an auxilary matrix to allow the convolution coincide with
% exercise notation
D = zeros(n+1,m+1);
D(2:n+1,2:m+1) = img;
% constuct kernels
K_x = [  0, 0,    0;
       0.5, 0, -0.5;
         0, 0,   0];
K_y = K_x';
Ix = conv2(K_x, D);
Ix = Ix(2:n+1,2:n+1);
Iy = conv2(K_y, D);
Iy = Iy(2:n+1,2:n+1);
```

1.2 Local auto-correlation matrix

For all pixels, we need to compute elements needed for the local auto-correlation matrix M.

```
% local auto-correlation matrix
% need to enumerate over all pixels to compute the matrix M and eigen
% values
Ix2 = imgaussfilt(Ix.^2, sigma);
Iy2 = imgaussfilt(Iy.^2, sigma);
Ixy = imgaussfilt(Ix.*Iy, sigma);
```

1.3 Harris response function

Then we compute the Harris response function C based on the element of the local auto-correlation matrix:

```
% Harris response function
R = Ix2.*Iy2 - Ixy.^2 - k * (Ix2+Iy2).^2;
C = R;
```

1.4 Detection criteria

To make a valid corner point, it has to satisfy two conditions: be the local maximum among its 3-cell neighborhood and the value of response function has to be above a predefined threshold.

```
% detection criteria: above threshold and local maximum in 3*3 neighbor
regmax = imregionalmax(R,8);
R = R.*regmax;
idx = find(R>thresh);
[rows,cols] = ind2sub(size(R),idx);
corners = [rows';cols'];
```

One possible issue is that the algorithm tends to also include points close to the edge of the image, which are actually not a real corner in the picture.

2 Description and Matching

2.1 Local descriptors

We need to filter out keypoints that are too close to edges of the image.

```
% filter our keypoints that are too close to the edge
[n, m] = size(img);
patch_size = int32(9);
half_size = idivide(patch_size,2);
new_corners = [];
descriptors = [];
for ii = 1:size(keypoints,2)
    if keypoints(1,ii)≤half_size || keypoints(1,ii)≥n−half_size+1 || ...
        keypoints(2,ii)≤half_size || keypoints(2,ii)≥m−half_size+1
        continue;
    end
    new_corners = [new_corners, keypoints(:,ii)];
end
keypoints = new_corners;
descriptors = extractPatches(img, keypoints, patch_size);
```

2.2 SSD one-way nearest neighbors matching

We need to compute a distance matrix which memorizes distance among all possible feature pairs.

```
function distances = ssd(descr1, descr2)
    distances = pdist2(descr1', descr2', 'squaredeuclidean');
end
```

Then the one-way nearest neighbor can by simply found by row-wise minimization of the distance matrix:

```
if strcmp(matching, 'one−way')
        [¬,idx] = min(distances,[],2);
        matches = [1:1:size(descr1,2);idx'];
```

2.3 Mutual nearest neighbors/Ratio test

The rest two methods are analogous to the first one-way method and we plot the related code in a whole here:

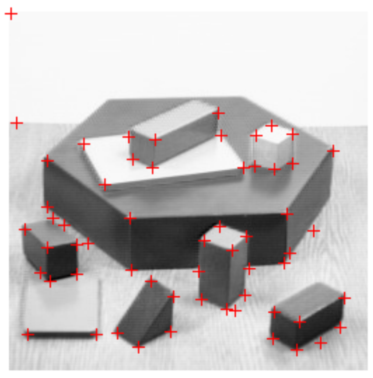
```
function matches = matchDescriptors(descr1, descr2, matching)
    % distances(i,j) denotes the distance between i−th feature in descri1
    % and the j−th feature in descri2
    distances = ssd(descr1, descr2);

    if strcmp(matching, 'one−way')
        [¬,idx] = min(distances,[],2);
        matches = [1:1:size(descr1,2);idx'];
    elseif strcmp(matching, 'mutual')
        [¬, idx1] = min(distances,[],2);
        [¬, idx2] = min(distances,[],1);
        matches = [];
        for ii=1:size(idx1)
            if idx2(idx1(ii))==ii
                matches = [matches, [ii;idx1(ii)]];
            end
        end
    elseif strcmp(matching, 'ratio')
        thresh = 0.5;
        [¬,idx] = mink(distances',2);
        matches = [];
        for ii=1:size(idx,2)
            if distances(ii,idx(1,ii))<distances(ii,idx(2,ii))*thresh
                matches = [matches, [ii;idx(1,ii)]];
            end
        end
    else
        error('Unknown matching type.');
    end
end
```

3 Results

3.1 Detection

By trail and error, we use the parameter: $T = 1e-6$, $k = 0.06$, $\sigma = 2$. We plot the example pictures as shown in Fig. 1:


(a) block

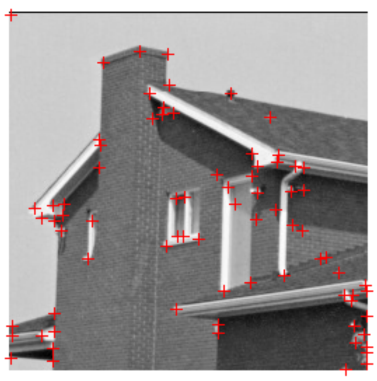

(b) house

Figure 1: corners illustration

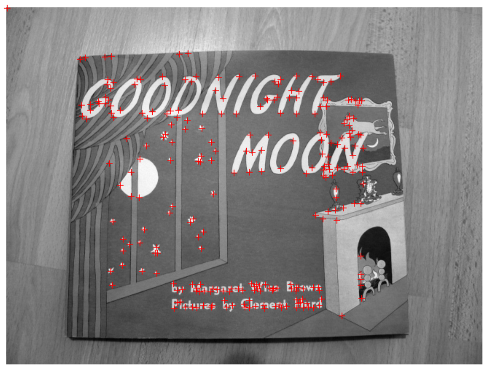
One issue with this method is that it tends to detect the points in the edge of the image as corners.

3.2 Matching

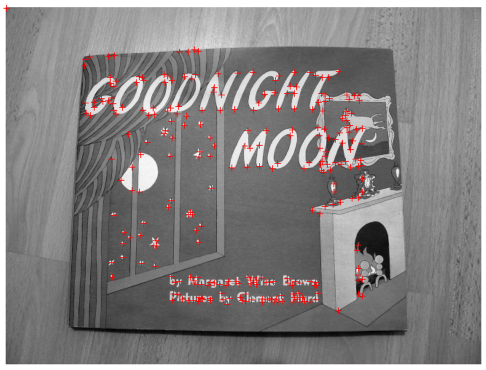
First, we use the previous detection setting to find the corners, and the results are shown in Fig. 2.

With one-way nearest neighbors matching, we obtain the result as shown in Fig 3. We can see that there are many crossing lines, which is not ideal in our case. As in a perfect match, all lines should be parallel and connect the same area in the image.

We then implemented the mutual nearest neighbors and ratio test and have the result in Fig. 4 and Fig. 5 respectively. We can see the crossling lines are less in both images. This shows that the matching performance is higher compared to the naive one-way nearest neighbors matching. However, there are still some mismatched pairs, for example, corner in the capitalized O in MOON in the left image can be confused with the O in GOODNIGHT from the right image, and we can see some crossing lines are actually due to this reason.



(a) I1



(b) I2

Figure 2: corners illustration

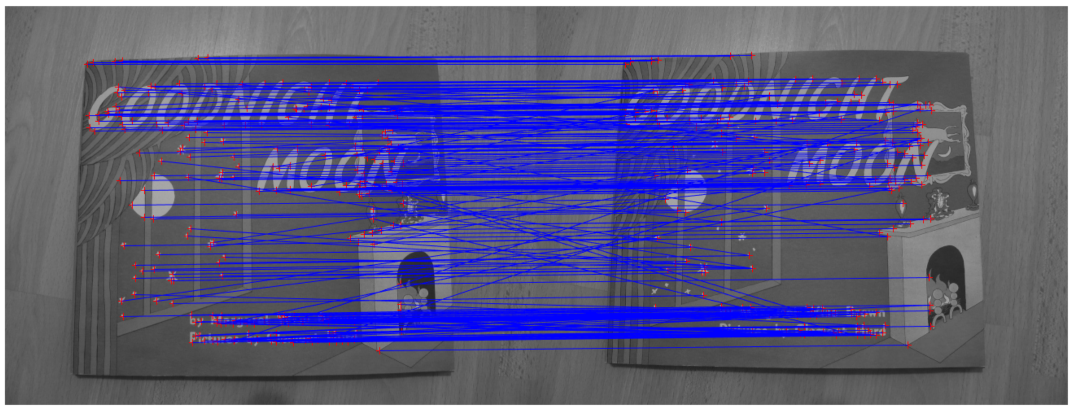


Figure 3: One-way nearest neighbors matching

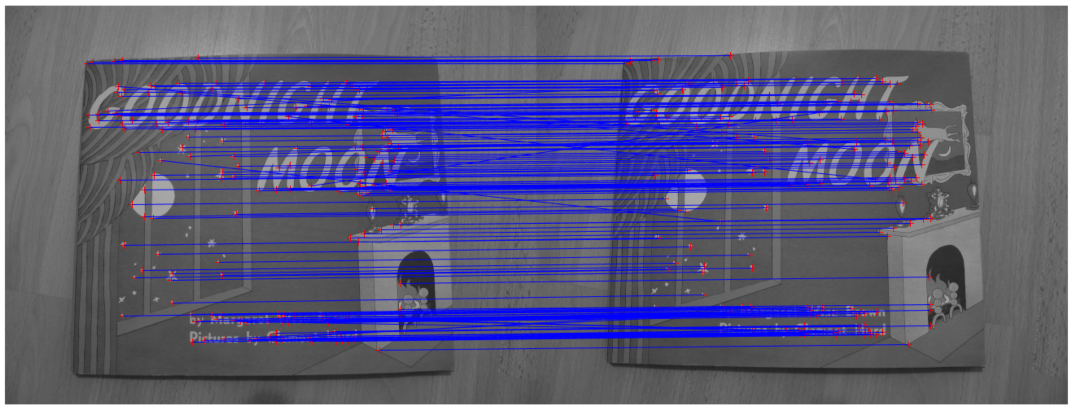


Figure 4: Mutual nearest neighbors matching

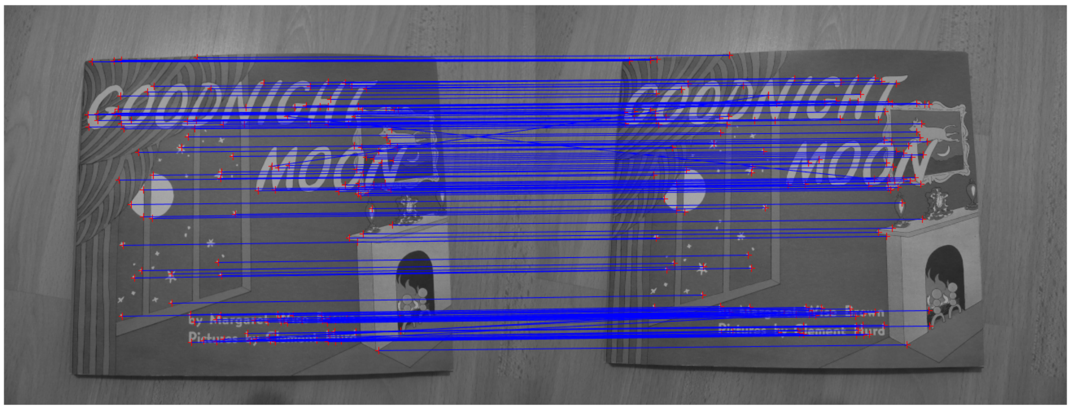


Figure 5: Ratio test