Computer Vision
Jacobs University Bremen
Dragi Kamov

Course: CO22-320671
October 31st, 2018

Homework 3

Problem 1

Solution:

I will include only my implementations of the codes in this document and the full solutions can be found in the appropriate files.

Problem 2

Solution:

For this problem check *SIFTDescriptor.m*.

First part

```
% gradient image, for gradients in x direction.
img_dx = zeros(size(currentImage));
% gradients in y direction.
img_dy = zeros(size(currentImage));

% Filtering
img_dx = filter2([-1,0,1], currentImage, 'same');
img_dy = filter2([-1;0;1], currentImage, 'same');
```

Second part

```
% Calculate the magnitude and orientation of the gradient.
grad_mag{scale} = zeros(size(currentImage));
grad_theta{scale} = zeros(size(currentImage));

grad_mag{scale} = sqrt((img_dx.^2) + (img_dy.^2));
grad_theta{scale} = atan2(img_dy, img_dx);
```

Third part

```
% Step 1: compute the dominant gradient direction of the patch
patch_angle_offset = 0;

patch_angle_offset = ComputeDominantDirection(patch_mag(:), patch_theta(:));

% Step 2: change patch_theta so it's relative to the dominant direction
patch_theta = patch_theta - patch_angle_offset;
```

Fourth part

```
% The patch we have extracted should be subdivided into
% num_histograms x num_histograms cells, each of which is size
% pixelsPerHistogram x pixelsPerHistogram.

% Compute a gradient histogram for each cell, and concatenate
% the histograms into a single feature vector of length 128.

% Please traverse the patch row by row, starting in the top left,
% in order to match the given solution. E.g. if you use two
% nested 'for' loops, the loop over x should  be the inner loop.

% (Note: Unlike the SIFT paper, we will not smooth a gradient across
% nearby histograms. For simplicity, we will just assign all
% gradient pixels within a pixelsPerHistogram x pixelsPerHistogram
% square to the same histogram.)

% Initializing the feature vector to size 0. Hint: you can
% concatenate the histogram descriptors to it like this:
```

```
% feature = [feature, histogram]
feature = [];

nBins=8;
for p = 1:num_histograms^2
    x = mod(4*p-3, 16);
    y = 4*ceil(p/4)-3;
    mag = patch_mag(y:y+3,x:x+3);
    angle = patch_theta(y:y+3,x:x+3);
    histogram = ComputeGradientHistogram(nBins, mag, angle);
    feature = [feature, histogram];
end
```

Fifth part

```
histogram = zeros(1, num_bins);
for i = 1:num_bins
    gradientD = (angles(i) < gradient_angles) & (angles(i) + angle_step >
       gradient_angles);
    histogram(i) = sum(gradient_magnitudes(gradientD));
end
```

Sixth part

```
% Compute a gradient histogram using the weighted gradient magnitudes.
% In David Lowe's paper he suggests using 36 bins for this histogram.
num_bins = 36;
% Step 1:
% compute the 36-bin histogram of angles using ComputeGradientHistogram()
[histogram, angles] = ComputeGradientHistogram(num_bins, gradient_magnitudes,
   gradient_angles);
% Step 2:
% Find the maximum value of the gradient histogram, and set "direction"
% to the angle corresponding to the maximum. (To match our solutions,
% just use the lower-bound angle of the max histogram bin. (E.g. return
% 0 radians if it's bin 1.)
maxA = histogram == max(histogram);
direction = angles(maxA);
```

Problem 3

Solution:

For this problem check *SIFTSimpleMatcher.m*.

```
n = size(descriptor1, 1);
m = size(descriptor2, 1);

for i = 1:n
    desc1 = repmat(descriptor1(i,:), [m,1]);

    summ = sum(((desc1 - descriptor2).^2), 2);
    dist = sqrt(summ);

    [X,Y] = sort(dist);

    if X(1) < thresh * X(2)
        match = [match; i Y(1)];
    end
end
```

Problem 4

Solution:

For this problem check *ComputeAffineMatrix.m*.

```
P1_t = P1';
P2_t = P2';

H(1,:) = P1_t \ (P2_t(:,1));
H(2,:) = P1_t \ (P2_t(:,2));
```

Problem 5

Solution:

For this problem check *RANSACFit.m*.

```
% hint: If you have an array of indices, MATLAB can directly use it to
% index into another array. For example, pt1(match(:, 1),:) returns a
% matrix whose first row is pt1(match(1,1),:), second row is
% pt1(match(2,1),:), etc. (You may use 'for' loops if this is too
% confusing, but understanding it will make your code simple and fast.)
dists = zeros(size(match,1),1);

pt_1 = [pt1(match(:,1),:)'; ones(1,(size(match,1)))];
pt_2 = [pt2(match(:,2),:)'; ones(1,(size(match,1)))];

dists = (sqrt(sum((pt_2 - H * pt_1).^ 2,1)))';
```

Problem 6

Solution:

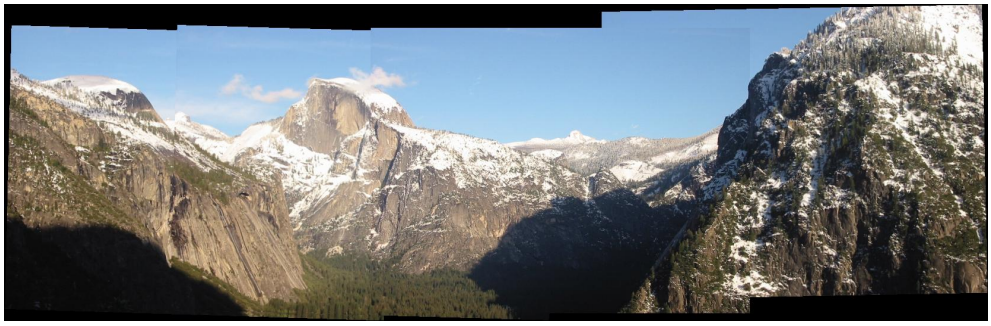
For this problem check *MultipleStitch.m*.

```
% HINT 1: There are two separate cases to consider: currentFrameIndex <
% refFrameIndex (this is the easier case), and currentFrameIndex >
% refFrameIndex (this is the harder case).

% HINT 2: You can use the pinv function to invert a transformation.

T = eye(3);

if(currentFrameIndex > refFrameIndex)
    for i=refFrameIndex:refFrameIndex
        T=T*pinv(i_To_iPlusOne_Transform{i});
    end
elseif (currentFrameIndex < refFrameIndex)
    for i=currentFrameIndex:refFrameIndex-1
        T=T*i_To_iPlusOne_Transform{i};
    end
end
```



Problem 7

Solution:

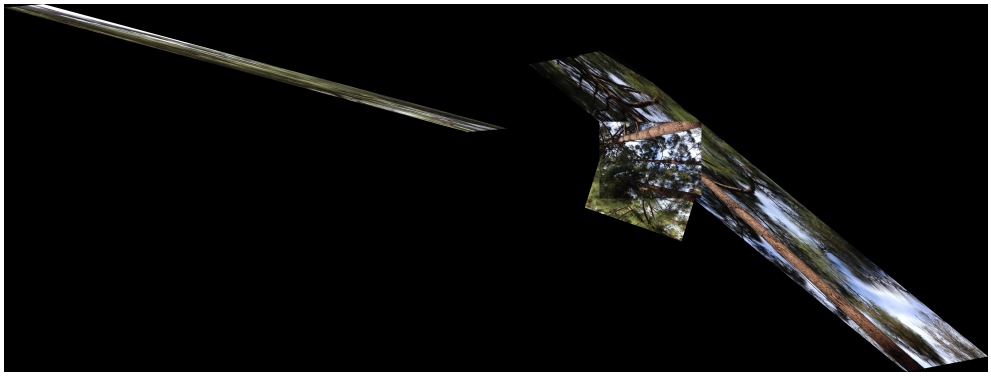


Figure 1: This picture is displayed at 17%.

Problem 8

Solution:

- Lowe's method for image feature generation transforms an image into a large collection of feature vectors, each of which is invariant to image translation, scaling, and rotation, partially invariant to illumination changes and robust to local geometric distortion. The features are highly distinctive, in the sense that a single feature can be correctly matched with high probability against a large database of features from many images.
- Key locations are defined as maxima and minima of the result of difference of Gaussians function applied in scale space to a series of smoothed and resampled images. Low-contrast candidate points and edge response points along an edge are discarded. Dominant orientations are assigned to localized keypoints. These steps ensure that the keypoints are more stable for matching and recognition. SIFT descriptors robust to local affine distortion are then obtained by considering pixels around a radius of the key location, blurring and resampling of local image orientation planes.
- With Difference of Gaussian function we are identifying potential interest points that are invariant to scale and orientation, and by random 5 pixels over the image we will not get that.
- After making a list of matching keypoints homographies between pairs of images are then computed using RANSAC and a probabilistic model is used for verification. Because there is no restriction on the input images, graph search is applied to find connected components of image matches such that each connected component will correspond to a panorama. Finally for each connected component bundle adjustment is performed to solve for joint camera parameters, and the panorama is rendered using multi-band blending.
- The image-stitching algorithm had problems with the "pine" image, it needed too much memory (26 GB) in order to compute the algorithm. At the end I got an image, but it was too big to be shown on my screen and it was reduced to 17%.