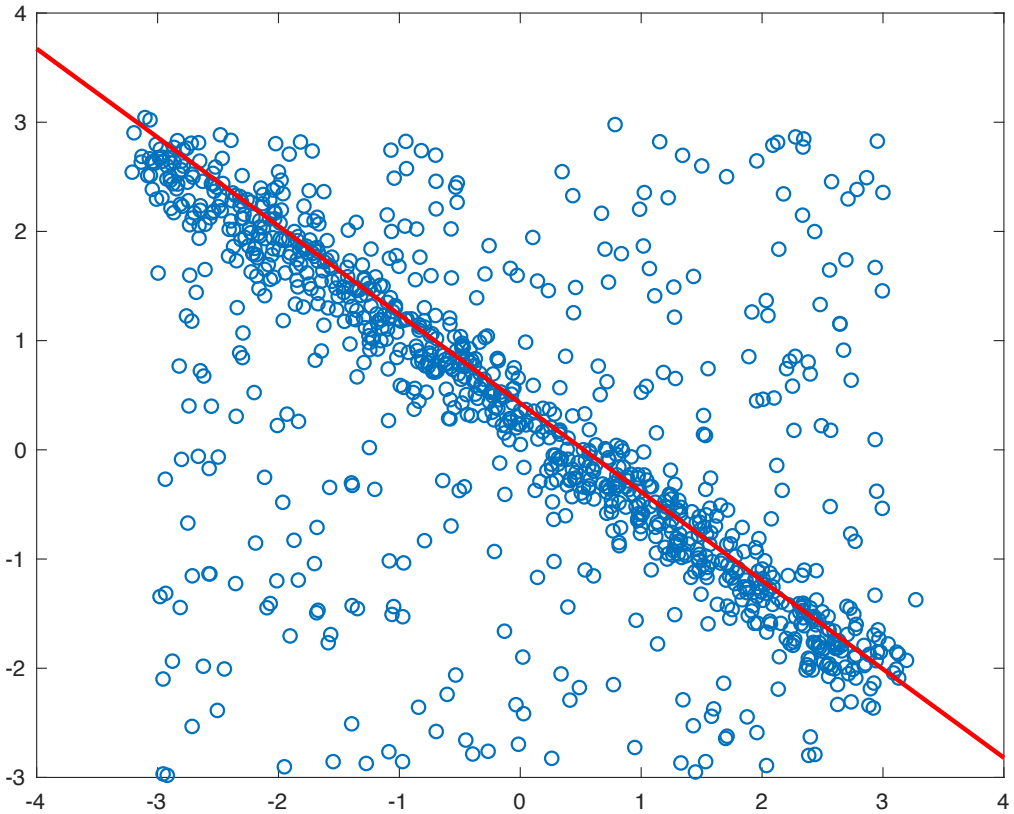
Computer Vision HW3

Chien-Cheng Lai
ChBE

I. Understanding RANSAC

- Ransac for Line Fitting



```
number = length(data.x);
bestInNum = 0; % Best fitting line with largest number of inliers
bestParameter1=0;bestParameter2=0; % parameters for best fitting line
DATA = [data.x;data.y];
for i = 1:iter
    %Randomly select 2 points
    idx = randperm(number,2);
    sample = DATA(:,idx);
    % Compute the distances between all points with the fitting line
    kLine = sample(:,2)-sample(:,1);% two points relative distance
    kLineNorm = kLine/norm(kLine);
    normVector = [-kLineNorm(2),kLineNorm(1)];%Ax+By+C=0 A=-kLineNorm(2),B=kLineNorm(1)
    distance = normVector*(DATA - repmat(sample(:,1),1,number));
    % Compute the inliers with distances smaller than the threshold
    inlierIdx = find(abs(distance)<=threshDist);
    inlierNum = length(inlierIdx);
    % Update the number of inliers and fitting model if better model is found
     if inlierNum>=round(inlierRatio*number) && inlierNum>bestInNum
         bestInNum = inlierNum;
         parameter1 = (sample(2,2)-sample(2,1))/(sample(1,2)-sample(1,1));
         parameter2 = sample(2,1)-parameter1*sample(1,1);
         bestParameter1=parameter1; bestParameter2=parameter2;
     end
end
```

Ransac for Line Fitting

1. I think initial number of points should be 2 better, because it has most number of possible fitting lines and also has less chance to include the outliers.
2. According to the lecture material, t^2 = 3.84*sigma^2 by Zero-mean Gaussian noise with std. dev. After lots of trials, I think 0.4 is the best threshold number.
3. According to the lecture material, we have this equation to choose N.

$$N = \log(1-p)/\log\left(1-(1-e)^s\right)$$

I choose p = 0.99, e = 0.6. So the result N = 26.4.

- Ransac for Line Fitting

  Firstly, I found the numbers of A matrix corresponding to each pixel by the giving feature data. Then by Ax=b, I apply Ransac method to compute the best fitting A.

```
a = zeros(6,6);

for i=1:iter
    idx = randperm(size(au,2),3);

    a(1:2,:) = kron(eye(2,2),[au(1:2,idx(1))' 1]);
    a(3:4,:) = kron(eye(2,2),[au(1:2,idx(2))' 1]);
    a(5:6,:) = kron(eye(2,2),[au(1:2,idx(3))' 1]);
    b = [av(1:2,idx(1))' av(1:2,idx(2))' av(1:2,idx(3))'];

    % Ax = b
    x=inv(a)*b';

    AT(1,:) = x(1:3);
    AT(2,:) = x(4:6);
    result = AT*[au;ones(1,size(au,2))];
    dist = sqrt(sum((av - result).^2));
    summation = sum(dist < threshold);
    if summation > subsum
        subsum = summation;
        outputx = x;
    end
end
Afinal = zeros(3,3);
Afinal(:,1) = outputx(1:3);
Afinal(:,2) = outputx(4:6);
Afinal(3,3) = 1;
```

As the result,the original picture transforming shows,

After finding the A matrix and apply to the original picture. It shows,

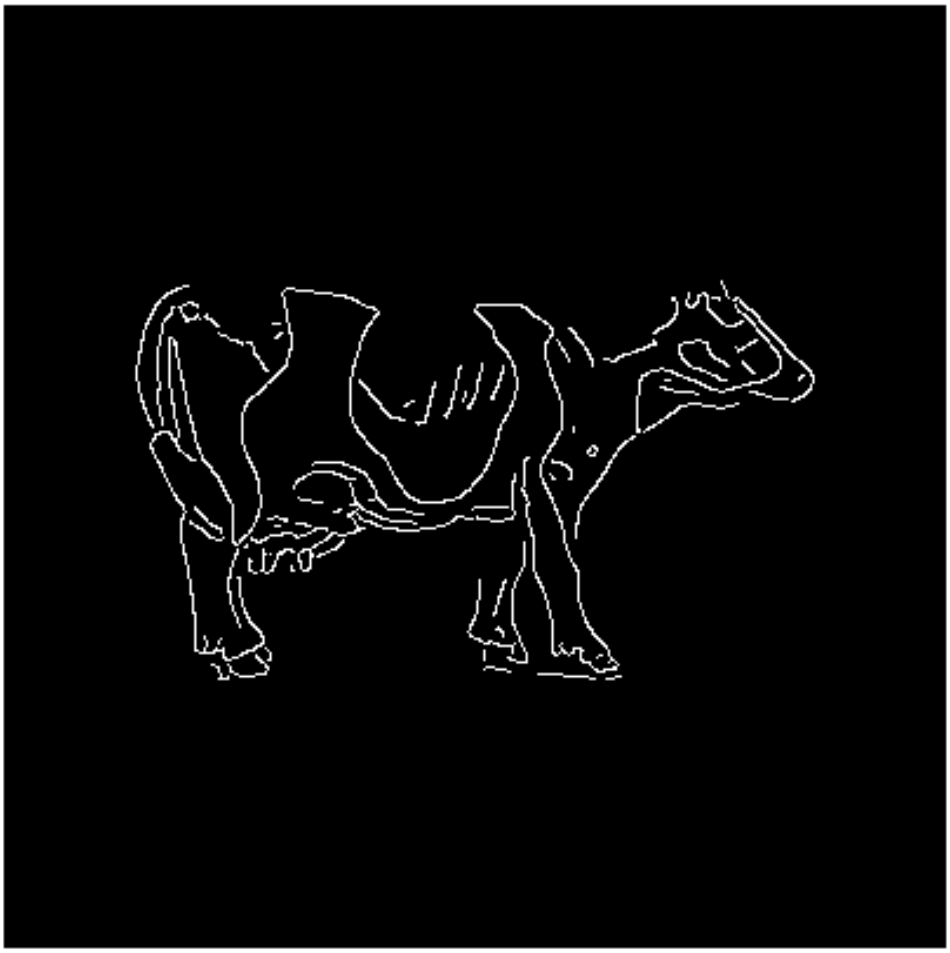
I could see that there is no much difference between these two pictures. That is to say, the parameter I found earlier is good enough to fitting the transforming features.

1. I think 3 is a better number because A is a 6x6 matrix. Each point can provide a equation with 2 unknowns and total is 6.
2. As question 1, after trying lots of numbers, I think 0.4 is a good number.
3. As question 1, by setting p = 0.99, e = 0.6, I think N = 26 is a good parameter to compute.

II. Distance Transforms and Chamfer Matching

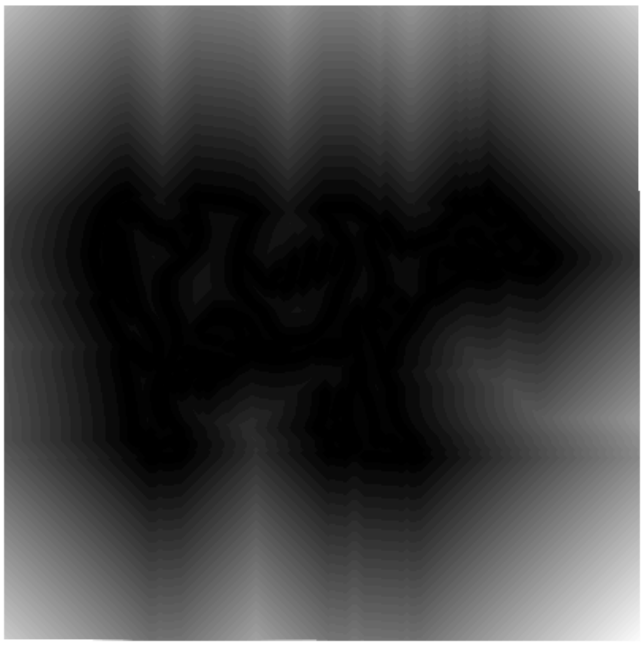
- Distance Transform
  1. Use edge function in matlab to compute the canny edge map.

```
%% Problem2-1-1
cow = imread('/Users/aaron/Desktop/Computer Vision/HW3/DistanceTransform_ChamferMatching
grcow = rgb2gray(cow);
edgecow = edge(grcow, 'canny', [0.14,0.16]);
imshow(edgecow)
```

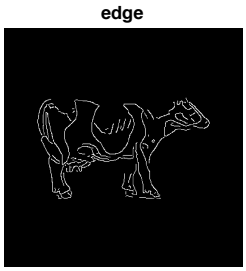


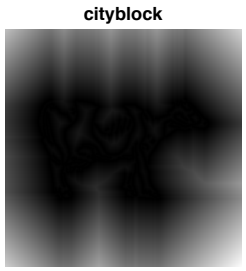
2. Use cityblock method to compute the distance from edge.

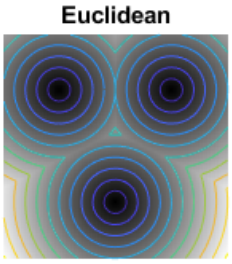
```
%% Problem2-1-2
d = zeros(size(edgecow));
d(find(edgecow == 0)) = Inf;
d(find(edgecow == 1)) = 0;

[r,c] = size(edgecow);

%forward
for i = 2:r
    for j = 2:c
        d(i,j) = min([d(i,j),d(i,j-1)+1,d(i-1,j)+1]);
    end
end

%backward
for i = fliplr(1:r-1)
    for j = fliplr(1:c-1)
        d(i,j) = min([d(i,j),d(i,j+1)+1,d(i+1,j)+1]);
    end
end
imshow(uint8(d))
```

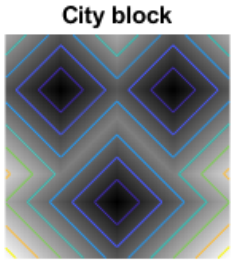


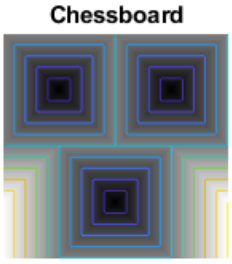
3.

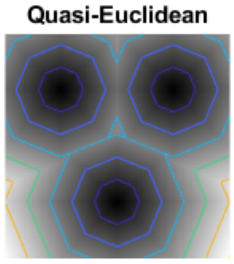












I think city block is faster in computing but less accurate in x-y direction and chessboard is also faster in computing but less accurate in diagonal direction.

- Chamfer Matching

  Place the template on the distance map and compute the sum of the corresponding distance where the location is at the edge.

After computing every sum numbers, pick the minimum one and it is exactly where the template should put on for the first pixel. As the result,

```
tem = imread('/Users/aaron/Desktop/Computer Vision/HW3/DistanceTransform_ChamferMatchin

r1 = size(d,1) - size(tem,1) + 1;
c1 = size(d,2) - size(tem,2) + 1;
diff = zeros(r1,c1);
for i = 1:r1
    for j = 1:c1
        diff(i,j) = sum(sum(abs(d(i:i+size(tem,1)-1,j:j+size(tem,2)-1)./256.*tem)));
        charmd = min(min(diff));
    end
end

[r2,c2]=min(diff(:));
[r3,c3]=ind2sub(size(diff),c2)

for i = 1:size(tem,1)
    for j = 1:size(tem,2)
        if (tem(i,j)==1)
            cow(i+r3,j+c3)=255;
        end
    end
end
imshow(cow)
```

III. Fast Directional Chamfer Matching

This essay is talking about improving the performance the accuracy of Chamfer Matching while reducing the computational time. The key point is that putting edge orientation information in the matching algorithm. So the resulting cost function is piecewise smooth and the cost variation is tightly bounded. Moreover, it presents a sublinear time algorithm for exact computation of the directional chamfer matching score using techniques from 3D distance transforms and directional integral images

Chamfer matching is a popular technique to find the best alignment between two edge maps. Chamfer matching provides a fairly smooth measure of fitness, and can tolerate small rotations, misalignments, occlusions, and deformations.

Chamfer matching becomes less reliable in the presence of background clutter. Instead of an explicit formulation of the orientation mismatch, we generalize the chamfer distance to points for matching directional edge pixels. Each edge point x is augmented with a direction term φ(x) and the directional chamfer matching (DCM) score is given by

$$d_{DCM}(U,V) = \frac{1}{n}\sum_{\mathbf{u}_i \in U} \min_{\mathbf{v}_j \in V} |\mathbf{u}_i - \mathbf{v}_j| + \lambda |\phi(\mathbf{u}_i) - \phi(\mathbf{v}_j)|$$

where λ is a weighting factor between location and orientation terms.

It also presents a three-dimensional distance transform representation to compute the matching cost in linear time.

In conclusion, it presented a novel approach for improving the accuracy of chamfer matching while significant reducing its computational cost. It provides an alternative approach for incorporating the edge orientation in the cost function and solved the matching problem in the orientation augmented space. The novel cost function is smooth and can be computed in sublinear time in the size of the shape template.