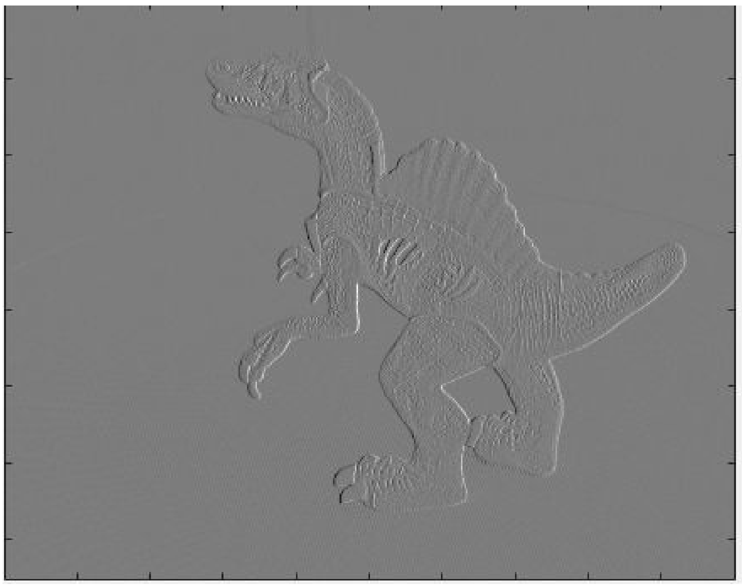
1)

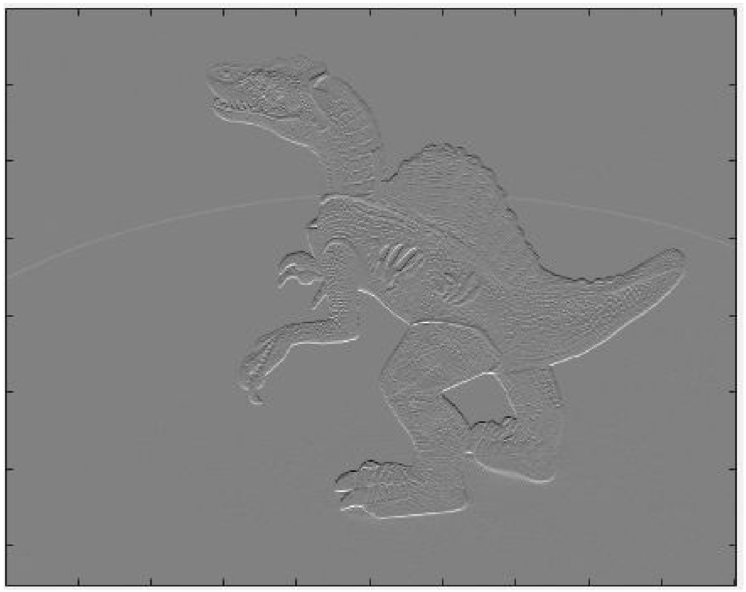
1.1) In order to make it easier to detect edges, a gaussian kernel is used to implement gaussian blur before calculating the gradient. For this image it seems like a sigma of 3 would result in the best edge detection since a sigma of 1 results in faint edges and a sigma of 5 results in a blurred gradient image.

Sigma = 1

xGradient

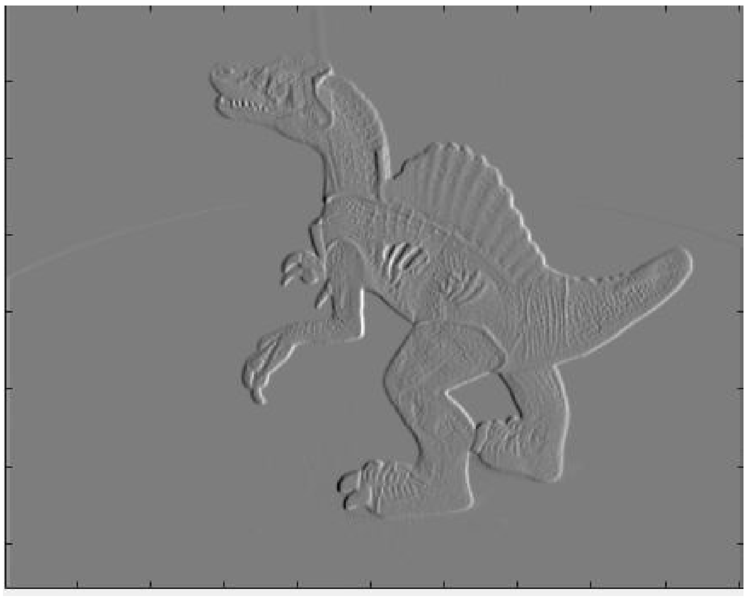


yGradient

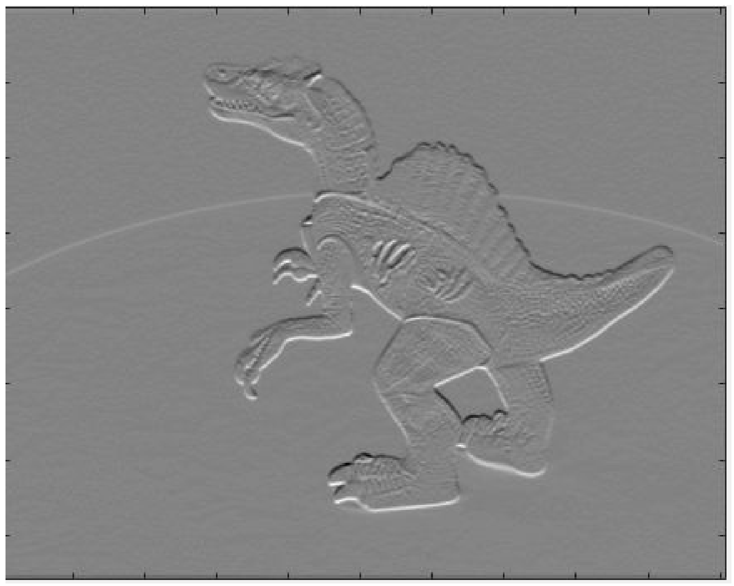


Sigma = 3

xGradient

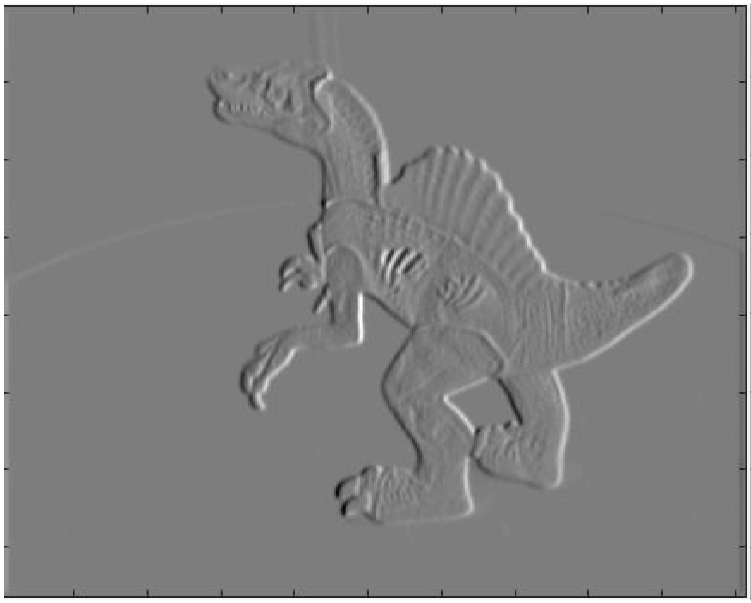


yGradient

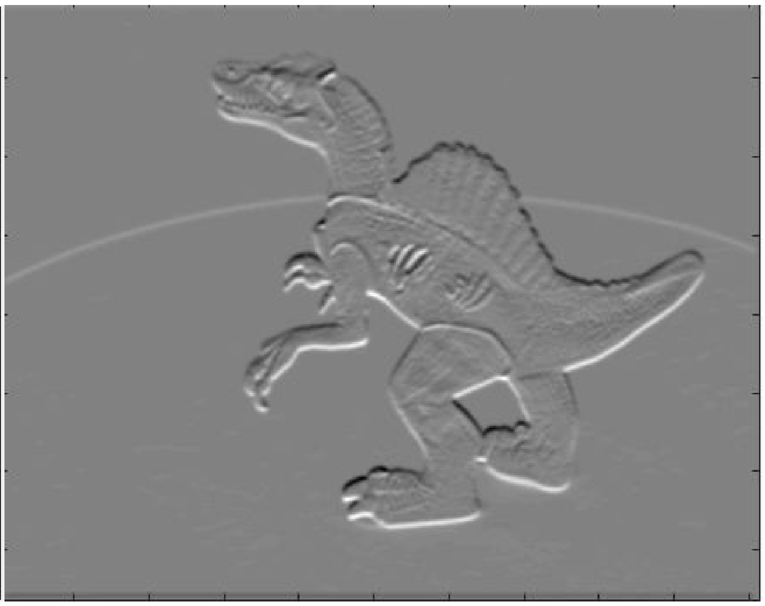


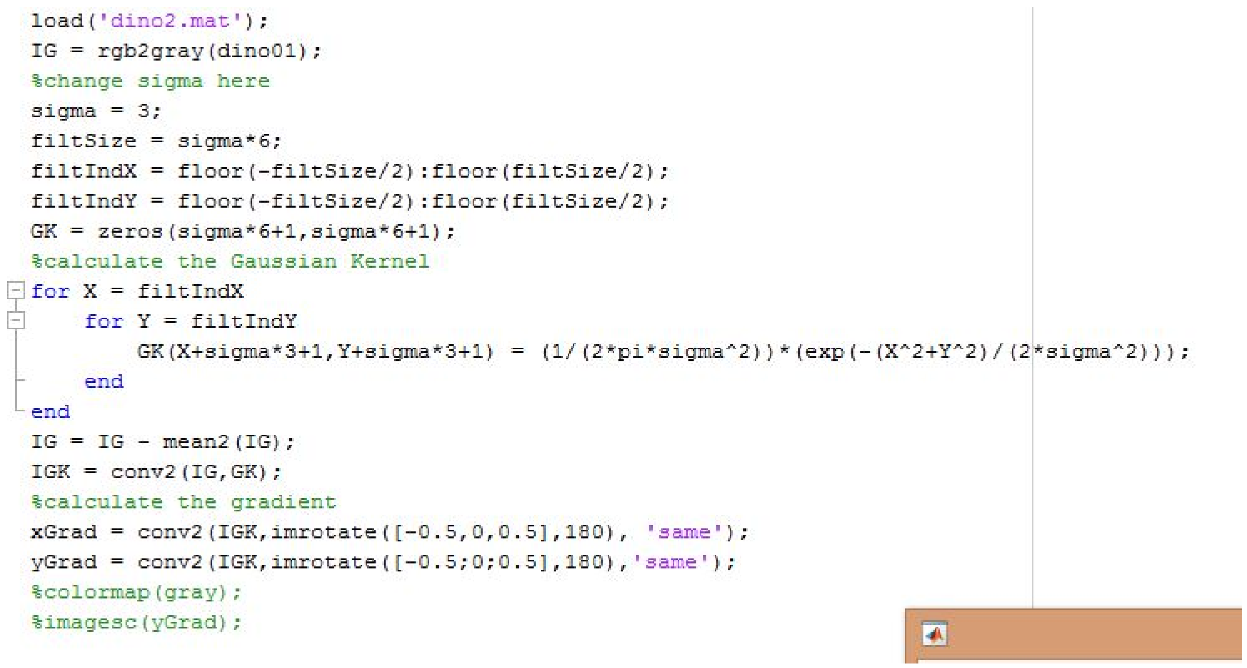
Sigma = 5

xGradient



yGradient

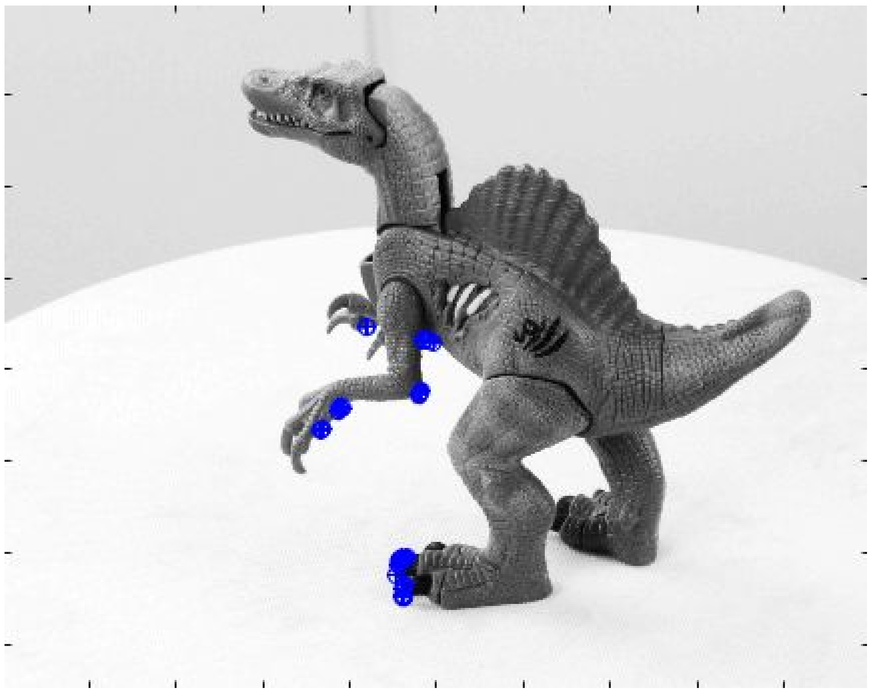


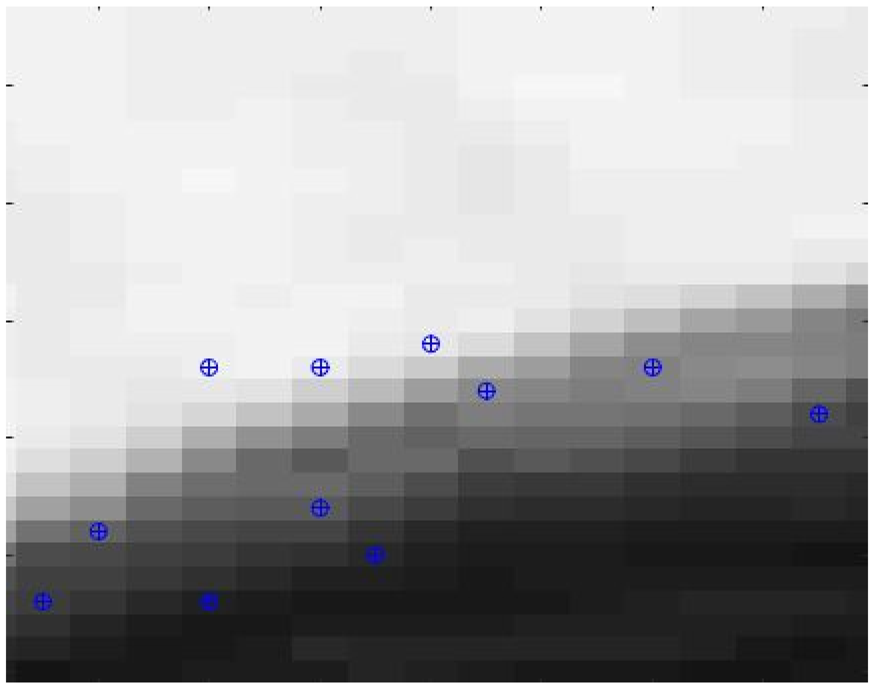


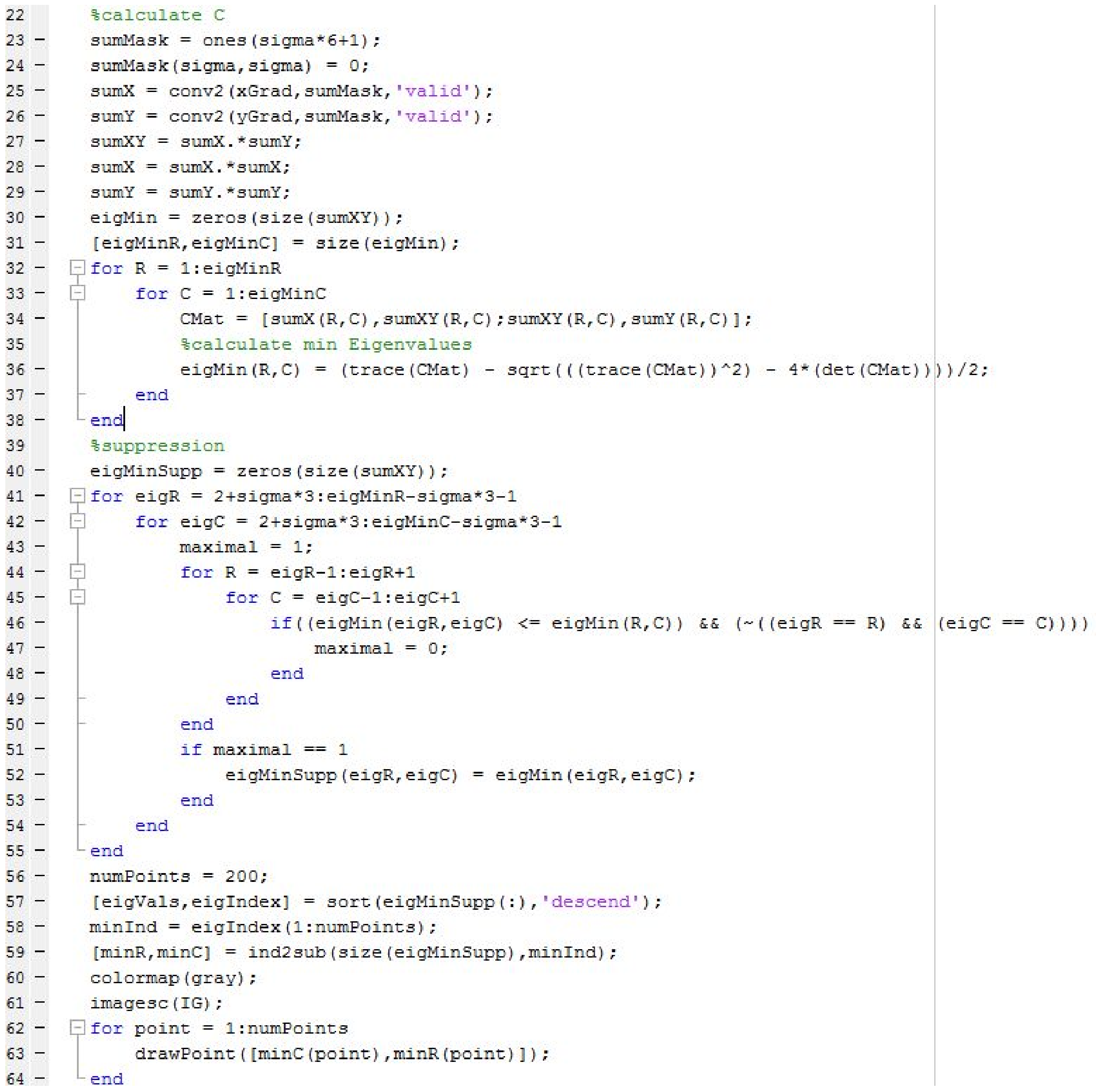
```
load('dino2.mat');
IG = rgb2gray(dino01);
%change sigma here
sigma = 3;
filtSize = sigma*6;
filtIndX = floor(-filtSize/2):floor(filtSize/2);
filtIndY = floor(-filtSize/2):floor(filtSize/2);
GK = zeros(sigma*6+1,sigma*6+1);
%calculate the Gaussian Kernel
for X = filtIndX
    for Y = filtIndY
        GK(X+sigma*3+1,Y+sigma*3+1) = (1/(2*pi*sigma^2))*(exp(-(X^2+Y^2)/(2*sigma^2)));
    end
end
IG = IG - mean2(IG);
IGK = conv2(IG,GK);
%calculate the gradient
xGrad = conv2(IGK,imrotate([-0.5,0,0.5],180), 'same');
yGrad = conv2(IGK,imrotate([-0.5;0;0.5],180),'same');
%colormap(gray);
%imagesc(yGrad);
```

1.2) The matrix C was calculated at every pixel in the image using the X and Y gradient matrices. Each C is composed of the sum of neighboring pixels in X squared, the sum of neighboring pixels in Y squared, and the two sums multiplied together. This 2x2 matrix is then used to quickly compute the minimum eigenvalue at each pixel and the indices of the smallest n points are the indices of the detected corners.

Corners for sigma=3



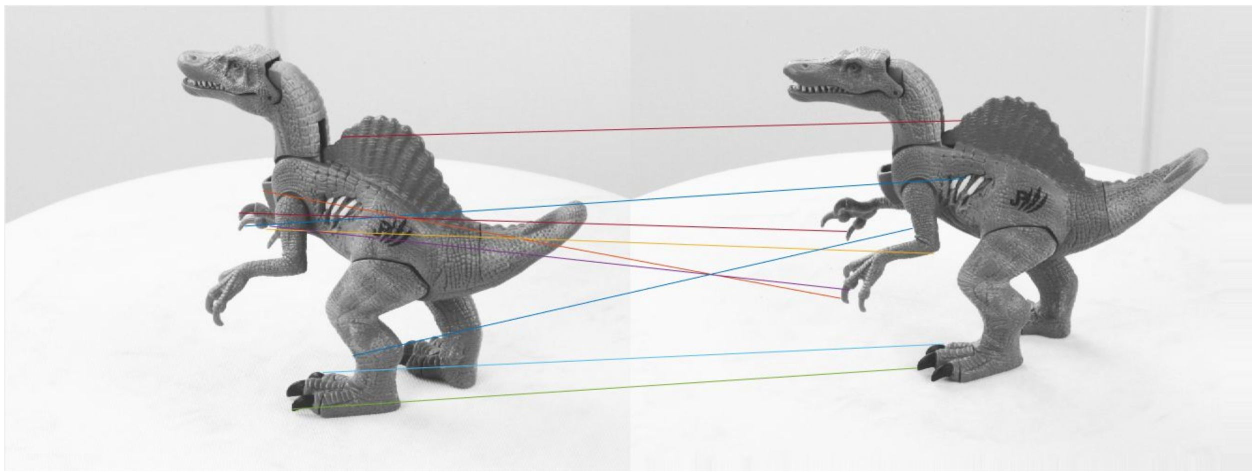


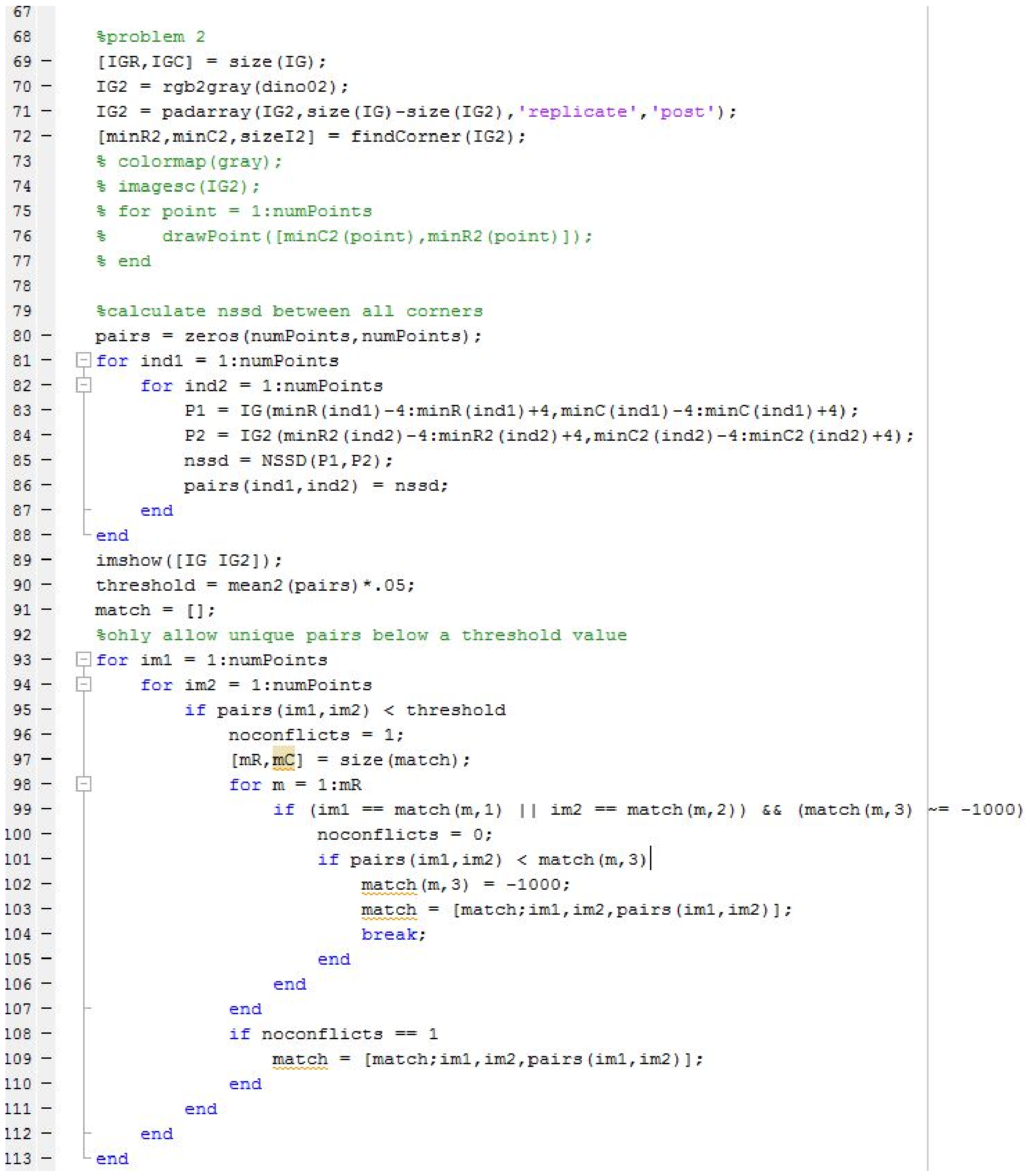


```matlab
%calculate C
sumMask = ones(sigma*6+1);
sumMask(sigma,sigma) = 0;
sumX = conv2(xGrad,sumMask,'valid');
sumY = conv2(yGrad,sumMask,'valid');
sumXY = sumX.*sumY;
sumX = sumX.*sumX;
sumY = sumY.*sumY;
eigMin = zeros(size(sumXY));
[eigMinR,eigMinC] = size(eigMin);
for R = 1:eigMinR
    for C = 1:eigMinC
        CMat = [sumX(R,C),sumXY(R,C);sumXY(R,C),sumY(R,C)];
        %calculate min Eigenvalues
        eigMin(R,C) = (trace(CMat) - sqrt(((trace(CMat))^2) - 4*(det(CMat))))/2;
    end
end
%suppression
eigMinSupp = zeros(size(sumXY));
for eigR = 2+sigma*3:eigMinR-sigma*3-1
    for eigC = 2+sigma*3:eigMinC-sigma*3-1
        maximal = 1;
        for R = eigR-1:eigR+1
            for C = eigC-1:eigC+1
                if((eigMin(eigR,eigC) <= eigMin(R,C)) && (~((eigR == R) && (eigC == C))))
                    maximal = 0;
                end
            end
        end
        if maximal == 1
            eigMinSupp(eigR,eigC) = eigMin(eigR,eigC);
        end
    end
end
numPoints = 200;
[eigVals,eigIndex] = sort(eigMinSupp(:),'descend');
minInd = eigIndex(1:numPoints);
[minR,minC] = ind2sub(size(eigMinSupp),minInd);
colormap(gray);
imagesc(IG);
for point = 1:numPoints
    drawPoint([minC(point),minR(point)]);
end
```

2) In this problem, the corners in part one are used to find the corresponding corners between the two slightly different dino images. The NSSD is calculated based on the patch of pixels around the a corner in each image. The lower the NSSD is the stronger the correlation between the patches is. If the NSSD was not lower than a threshold value than the line won't be drawn. Also, the point pair with the smallest unique NSSD value was chosen so that each point would not draw to multiple other points. Before drawing, the ratio between the unique best matches and the second best match was checked in order to further filter out non unique points. I think that because the claws on each of the four hands were so similar, there was a lot of crossing between the two.

Threshold value used = (mean of all threshold pairs)*0.04
Rmatch value used = 0.49

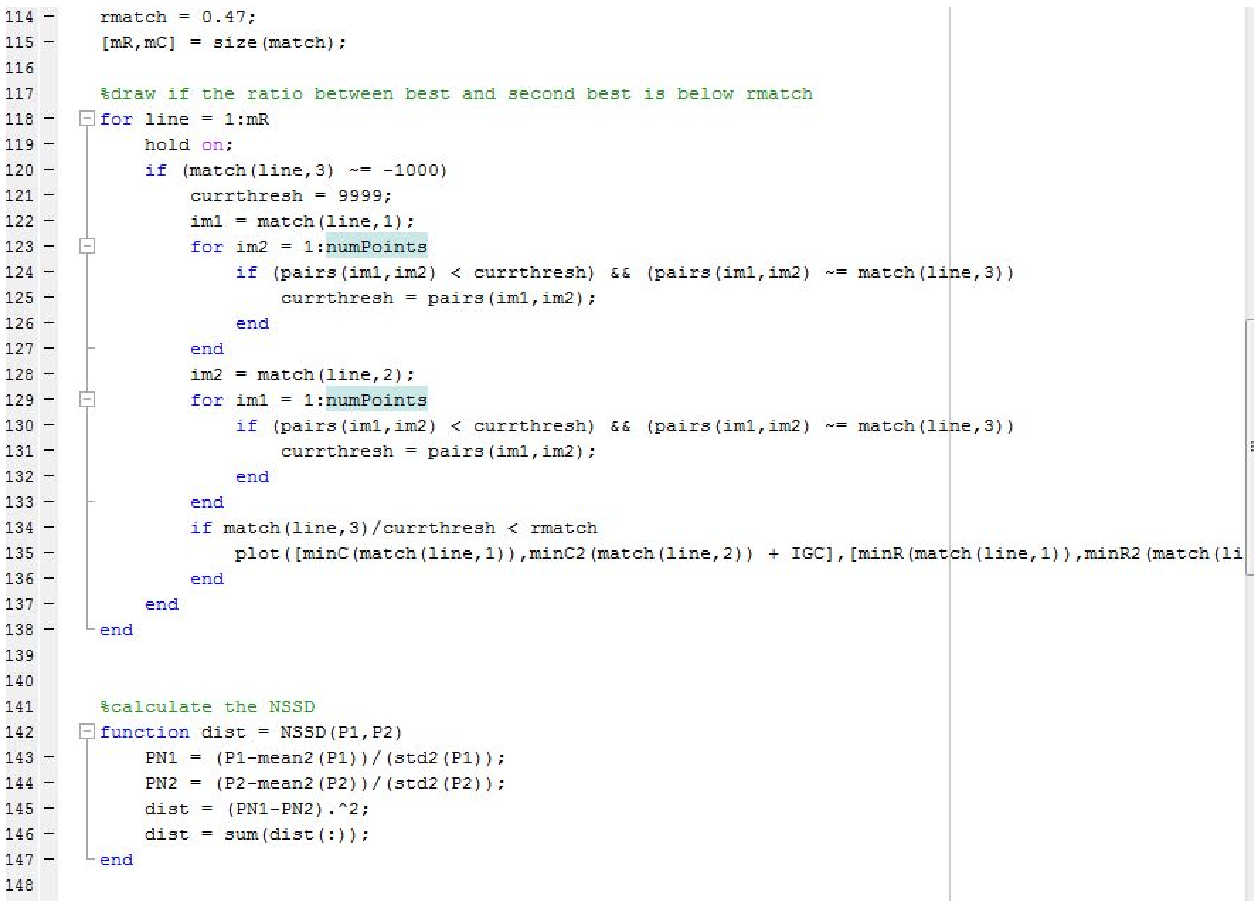




```matlab

%problem 2
[IGR,IGC] = size(IG);
IG2 = rgb2gray(dino02);
IG2 = padarray(IG2,size(IG)-size(IG2),'replicate','post');
[minR2,minC2,sizeI2] = findCorner(IG2);
% colormap(gray);
% imagesc(IG2);
% for point = 1:numPoints
%     drawPoint([minC2(point),minR2(point)]);
% end

%calculate nssd between all corners
pairs = zeros(numPoints,numPoints);
for ind1 = 1:numPoints
    for ind2 = 1:numPoints
        P1 = IG(minR(ind1)-4:minR(ind1)+4,minC(ind1)-4:minC(ind1)+4);
        P2 = IG2(minR2(ind2)-4:minR2(ind2)+4,minC2(ind2)-4:minC2(ind2)+4);
        nssd = NSSD(P1,P2);
        pairs(ind1,ind2) = nssd;
    end
end
imshow([IG IG2]);
threshold = mean2(pairs)*.05;
match = [];
%ohly allow unique pairs below a threshold value
for im1 = 1:numPoints
    for im2 = 1:numPoints
        if pairs(im1,im2) < threshold
            noconflicts = 1;
            [mR,mC] = size(match);
            for m = 1:mR
                if (im1 == match(m,1) || im2 == match(m,2)) && (match(m,3) ~= -1000)
                    noconflicts = 0;
                    if pairs(im1,im2) < match(m,3)
                        match(m,3) = -1000;
                        match = [match;im1,im2,pairs(im1,im2)];
                        break;
                    end
                end
            end
            if noconflicts == 1
                match = [match;im1,im2,pairs(im1,im2)];
            end
        end
    end
end
```



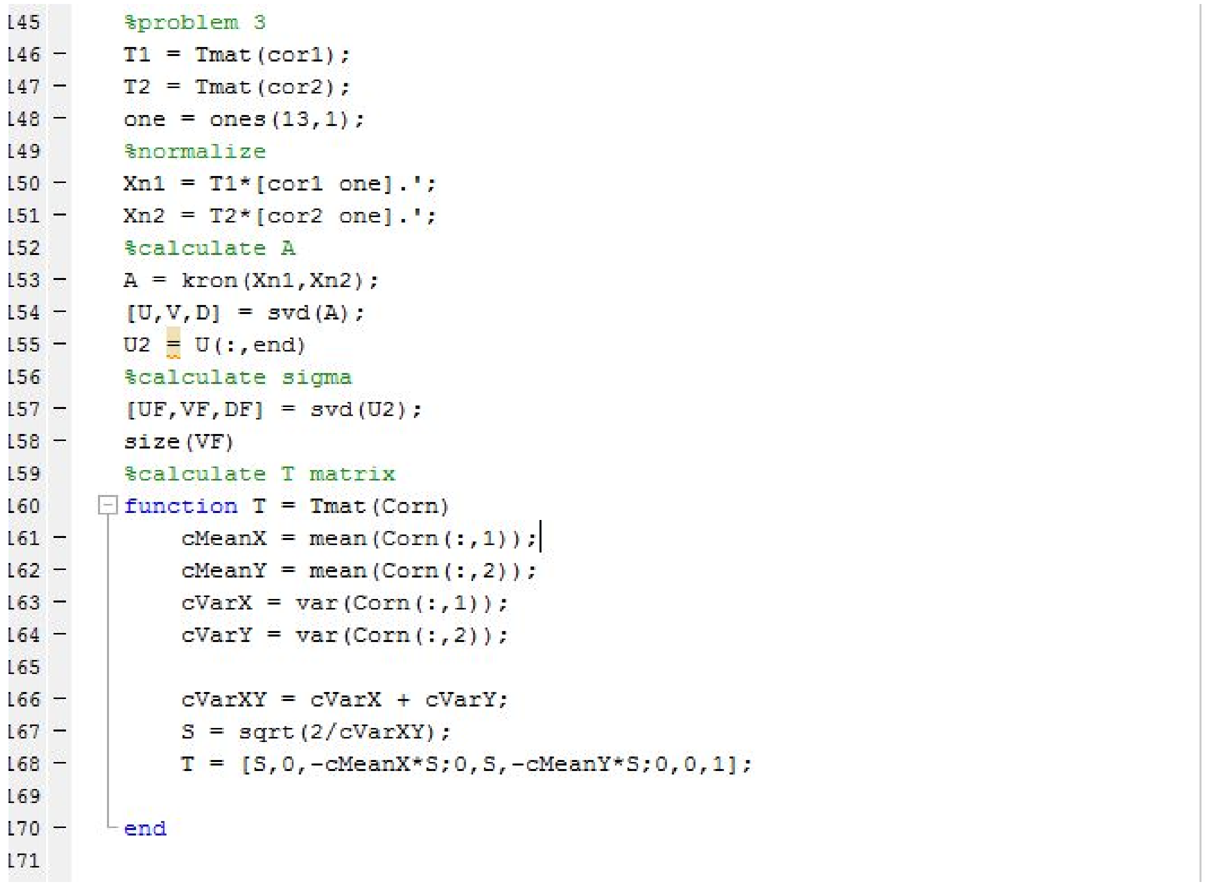
```
rmatch = 0.47;
[mR,mC] = size(match);

%draw if the ratio between best and second best is below rmatch
for line = 1:mR
    hold on;
    if (match(line,3) ~= -1000)
        currthresh = 9999;
        im1 = match(line,1);
        for im2 = 1:numPoints
            if (pairs(im1,im2) < currthresh) && (pairs(im1,im2) ~= match(line,3))
                currthresh = pairs(im1,im2);
            end
        end
        im2 = match(line,2);
        for im1 = 1:numPoints
            if (pairs(im1,im2) < currthresh) && (pairs(im1,im2) ~= match(line,3))
                currthresh = pairs(im1,im2);
            end
        end
        if match(line,3)/currthresh < rmatch
            plot([minC(match(line,1)),minC2(match(line,2)) + IGC],[minR(match(line,1)),minR2(match(li
        end
    end
end


%calculate the NSSD
function dist = NSSD(P1,P2)
    PN1 = (P1-mean2(P1))/(std2(P1));
    PN2 = (P2-mean2(P2))/(std2(P2));
    dist = (PN1-PN2).^2;
    dist = sum(dist(:));
end
```

3.1)

I did not complete the function for calculating the fundamental matrix. In order to calculate each of the T's I used the function Tmat which takes in the 13x2 corners and uses the mean and variance. This T us then used to normalized the cor1 and cor2. The two normalized corners are put into the function kron to construct the matrix A. The matrix A put in the function svd to construct U V and D. The last column of the 9x9 matrix U is reconstructed into a 3x3 matrix by using svd on the column and changing the new V's corner to 0. I was not sure where to go from here in calculating the fundamental matrix.



```
%problem 3
T1 = Tmat(cor1);
T2 = Tmat(cor2);
one = ones(13,1);
%normalize
Xn1 = T1*[cor1 one].';
Xn2 = T2*[cor2 one].';
%calculate A
A = kron(Xn1,Xn2);
[U,V,D] = svd(A);
U2 = U(:,end)
%calculate sigma
[UF,VF,DF] = svd(U2);
size(VF)
%calculate T matrix
function T = Tmat(Corn)
    cMeanX = mean(Corn(:,1));
    cMeanY = mean(Corn(:,2));
    cVarX = var(Corn(:,1));
    cVarY = var(Corn(:,2));

    cVarXY = cVarX + cVarY;
    S = sqrt(2/cVarXY);
    T = [S,0,-cMeanX*S;0,S,-cMeanY*S;0,0,1];

end
```