EEE 178 Final Exam Part 3

Curtis Muntz

I. Problem 1: Calibrate Camera

Using the provided data set and our understanding of matrices and the camera calibration process, we solved for the problem $Ax = 0$ using MATLAB's SVD funciton. The solution is below.

```
clear all; close all; clc;
%DATASET is in the form: r, c, x, y, z;
DATASET=[927 425 126 13.4 531.3
1113 417 202 13.4 531.3
200 490 -178 29.2 551.3
779 469 69 29.2 551.3
870 466 107 29.2 551.3
1053 457 183 29.2 551.3
226 561 -178 60.7 591.4
564 550 -26 60.7 591.4
731 543 50 60.7 591.4
1070 532 202 60.7 591.4
403 589 -102 76.5 611.4
527 582 -45 76.5 611.4
608 582 -7 76.5 611.4
932 568 145 76.5 611.4
292 620 -159 92.2 631.5
414 618 -102 92.2 631.5
530 613 -45 92.2 631.5
726 605 50 92.2 631.5
1003 596 183 92.2 631.5
343 648 -140 108 651.5
647 640 12 108 651.5
685 638 31 108 651.5
800 634 88 108 651.5
1029 623 202 108 651.5
279 679 -178 123.8 671.6
388 672 -121 123.8 671.6
557 669 -45 123.8 671.6
611 667 -7 123.8 671.6
396 700 -140 139.5 691.6
471 698 -83 139.5 691.6
612 691 -7 139.5 691.6];

r=DATASET(:,1);
c=DATASET(:,2);
x=DATASET(:,3);
y=DATASET(:,4);
z=DATASET(:,5);


rline=[(r(:).*z(:)),-z(:),-x(:)];
cline=[c(:).*z(:),-z(:),-y(:)];
A=[rline];
[U,D,V]=svd(A);
X_1=V(:,end)

A=[cline];
[U,D,V]=svd(A);
X_2=V(:,end)

Sx=X_1(1)
Sy=X_2(1)
u0=X_1(2)/X_1(1)
v0=X_2(2)/X_2(1)
lambda=(X_1(3)-(X_2(3)))/2
```

```
X_1 =

    0.0007
    0.4347
    0.9006


X_2 =

   -0.0006
   -0.2419
   -0.9703


Sx =

   6.9570e-04


Sy =

  -6.2794e-04


u0 =

  624.7794


v0 =

  385.2259


lambda =

    0.9354
```

II. Problem 2: Stereo Vision

A. Part 1

Because they were needed, the solution to α_u, α_v, u_0, and v_0 are provided through the pseudo-inverse solution to the $Ax = b$ equation from the data set given in problem 1. These values are used to build a stereoParams object that I will use later.

```
Left=imread('../../../Pictures/
    ph20140515_122217.jpg');
Right=imread('../../../Pictures/
    ph20140515_122100.jpg');
```

```
close all; clc; %clearvars -except r c x y z
    DATASET;
Br=r(:).*z(:); % r*Z
Bc=c(:).*z(:);
A1=[z(:),x(:)];
A2=[z(:),y(:)];

X1=(inv(A1'*A1)*A1'*Br)
X2=(inv(A2'*A2)*A2'*Bc)

%create a stereoParams object with known values
stereoParams.baseline=20e-2;
stereoParams.lambda=lambda;
stereoParams.u0=floor(u0);
stereoParams.v0=floor(v0);
stereoParams.Sx=Sx;
stereoParams.Sy=Sy
```

```
X1 =

   1.0e+03 *

    0.6248
    1.2934


X2 =

   1.0e+03 *

    0.3877
    1.5277


stereoParams =

    baseline: 0.2000
      lambda: 0.9354
          u0: 624
          v0: 385
          Sx: 6.9570e-04
          Sy: -6.2794e-04
```

B. Part 2

By selecting the same point on the left and right camera images, I was able to estimate the real world distance to the cantaloupe, Kleenex box, and the marker. The units shown are meters.

```
% The following values are found by zooming
    into similar points on objects within the
    frame.
%IMAGE COORDINATES ARE IN FORM [leftR, leftC,
    rightR, rightC]
cantelope_coords=[756,579, 452,544];
kleenex_coords=[1129,598, 859,564];
marker_coords=[562,513, 353,482];

subplot(231);
imshow(Left); title('Left Cam');
subplot(232);
imshow(Right); title('Right Cam');

cantelopeDistance = testPhysics(stereoParams,
    cantelope_coords)
kleenexDistance= testPhysics(stereoParams,
    kleenex_coords)
markerDistance = testPhysics(stereoParams,
    marker_coords)
```

```
cantelopeDistance =

    0.0868   -0.1152    0.8846


kleenexDistance =

    0.3741   -0.1424    0.9960


markerDistance =

   -0.0593   -0.1106    1.2867
```

C. Part 3

Finding the coordinates of the edges of the visible side of the box, I was able to find the area of the kleenex boxe's side. Finding the points for three of the four corners on the box enables me to be able to find the distances between the points, and then estimate the real world units of length and width of the box. The answer given is in square meters.

```
upperLeft_coords=[955,541, 687,507];
upperRight_coords=[1260,551, 992,516];
lowerLeft_coords=[953,662, 683,628];

upperLeft  = testPhysics(stereoParams,
    upperLeft_coords);
upperRight = testPhysics(stereoParams,
    upperRight_coords);
lowerLeft  = testPhysics(stereoParams,
    lowerLeft_coords);

Length=sqrt((upperLeft(1)-upperRight(1))^2+(
    upperLeft(2)-upperRight(2))^2+(upperLeft(3)
    -upperRight(3))^2);
Width=sqrt((upperLeft(1)-lowerLeft(1))^2+(
    upperLeft(2)-lowerLeft(2))^2+(upperLeft(3)-
    lowerLeft(3))^2);

kleenexArea=Length*Width
```

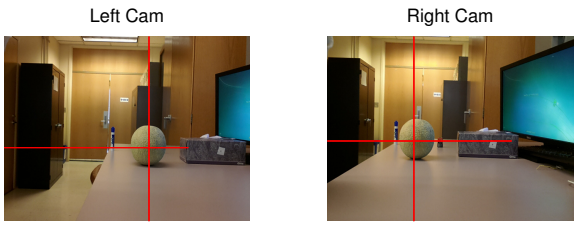
```
kleenexArea =

    0.0183
```

D. Part 4

Assuming the cantelope is a perfect sphere, we can estimate its volume by using the mathetical model of a sphere. By definition, the volume of a sphere is $\frac{4}{3}\pi r^3$. Assuming the cantelope is perfectly spherical, no matter which way we look at it, we will see a circle without knowing the depth. We can therefore estimate the diameter, and then the radius, and then the volume of the cantelope by finding the distance from one point along the cantelope's circumference and its antipode. The answer given is in cubic meters.

```
[M,N]=size(Left);
cantelope_coords=[756,579, 452,544];
subplot(231);
imshow(Left); title('Left Cam'); hold on;
plot(cantelope_coords(1),(1:N),'-','color','r')
    ; hold on;
plot((1:M), cantelope_coords(2),'color','r');
    hold off;
subplot(232);
imshow(Right); title('Right Cam'); hold on;
plot(cantelope_coords(3),(1:N),'LineStyle', '-'
    ,'color','r'); hold on;
plot((1:M),cantelope_coords(4),'LineStyle', '-'
    ,'color','r'); hold off;

cantelopeCirc=[756,465, 452,430];
cantelopeAntipode=[756,673, 452,638];

cantelopeLeft= testPhysics(stereoParams,
    cantelopeCirc);
cantelopeRight= testPhysics(stereoParams,
    cantelopeAntipode);
radius=(1/2)*sqrt((cantelopeLeft(1)-
    cantelopeRight(1))^2+(cantelopeLeft(2)-
    cantelopeRight(2))^2+(cantelopeLeft(3)-
    cantelopeRight(3))^2);
volume = (4/3)*pi*radius^3
```

```
volume =

    9.8664e-04
```

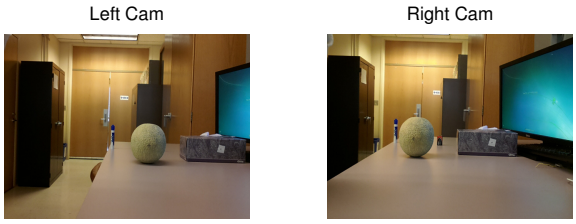


E. Part 5

Finding the height of the marker is as simple as finding the points on the bottom of the marker and the top, and calculating the distance between the points. The answer given is in meters.

```
subplot(231);
imshow(Left); title('Left Cam');
subplot(232);
imshow(Right); title('Right Cam');

bottom_coord=[563,596, 356,564];
top_coord=[567,471, 360,440];

bottom = testPhysics(stereoParams, bottom_coord
    );
top    = testPhysics(stereoParams, top_coord);

Height=sqrt((top(1)-bottom(1))^2+(top(2)-bottom
    (2))^2+(top(3)-bottom(3))^2)
```

```
Height =

    0.1091
```



III. Problem 3: Correcting For Bad Data

A. Part 1

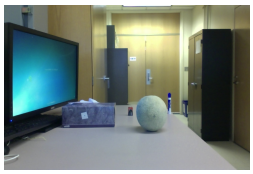
Using the provided intrinsic parameters of the camera, I had to redefine some of the stereo triangulation equations in order to solve for the distances. However, it was important to note that the photo relating to the left image is labeled as the right image in the problem statement. I corrected for this error and the appropriate titles can be seen in the figures. The solution for distance is in meters.

```
%clearvars -except Left Right; close all; clc;
clear stereoParams
LeftNEW=imread('../../../Pictures/leftNu.jpg');
RightNEW=imread('../../../Pictures/rightNu.jpg'
    );
subplot(231);
imshow(LeftNEW); title('Leftbad Cam');
subplot(232);
imshow(RightNEW); title('Rightbad Cam');
subplot(234);
imshow(Left); title('Left Cam');
subplot(235);
imshow(Right); title('Right Cam');


% Data in form: r c x y z
P3DATA=[179.0000 49.0000 -18.0000 -235.0000
    549.0000;
%843.0000 56.0000 -18.0000 64.6189 929.7477;
%318.0000 513.0000 362.0000 -235.0000 549.0000;
721.0000 519.0000 362.0000 64.6189 929.7477;
423.0000 159.0000 115.0000 -187.6918 609.1181;
731.0000 130.0000 305.0000 -219.2306 569.0394;
300.0000 277.0000 1.0000 -124.6141 689.2755;
625.0000 254.0000 248.0000 -140.3835 669.2361;
739.0000 353.0000 343.0000 -77.3059 749.3935;
412.0000 305.0000 96.0000 -108.8447 709.3148;
248.0000 188.0000 -18.0000 -171.9223 629.1574;
```

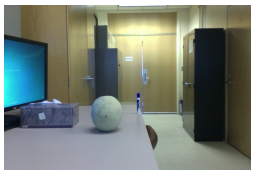
```
707.0000 396.0000 324.0000 -45.7670 789.4722;
%410.0000 449.0000 267.0000 -156.1529 649.1968;
684.0000 395.0000 305.0000 -45.7670 789.4722;
498.0000 450.0000 153.0000 1.5412 849.5903;
424.0000 372.0000 96.0000 -61.5364 769.4329;
221.0000 87.0000 1.0000 -219.2306 569.0394;
801.0000 93.0000 343.0000 -219.2306 569.0394;
336.0000 499.0000 1.0000 48.8495 909.7083;
707.0000 503.0000 343.0000 48.8495 909.7083;];

%given params
alpha_u=476.83;
alpha_v=433.45;
u0=433.45;
v0=385.64;

stereoParams.baseline=20e-2;
stereoParams.u0=ceil(u0); %I NEED TO ASK
    BELKHOUCHE ABOUT THIS
stereoParams.v0=ceil(v0);
stereoParams.alpha_u=alpha_u;
stereoParams.alpha_v=alpha_v;



cantelope_coords = [660,486, 446,487];
cantelopeDistance_bad = testPhysics_alphas(
    stereoParams,cantelope_coords)
```

```
cantelopeDistance_bad =

    0.2112    0.1028    0.4456
```

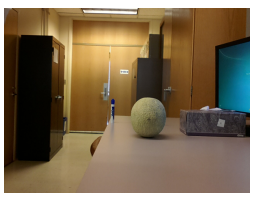
Leftbad Cam



Rightbad Cam



Left Cam

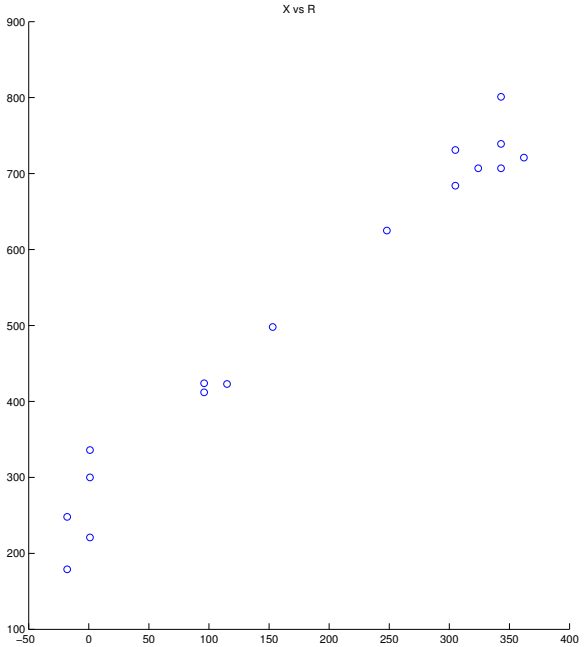


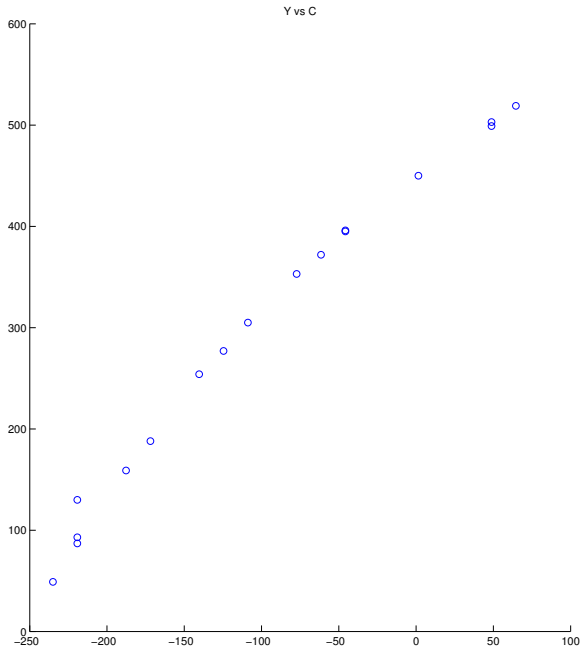
Right Cam

B. *Part 2*

Some bad data points were used to calibrate these cameras. By plotting the data points, we can clearly see some outliers. I elimiated the points [843.0000 56.0000 -18.0000 64.6189 929.7477], [318.0000 513.0000 362.0000 -235.0000 549.0000;], and [410.0000 449.0000 267.0000 -156.1529 649.1968;] (the outlier points decided based on the graphs below).

```
%r(rowtoremove,:)=[]

r=P3DATA(:,1);
c=P3DATA(:,2);
x=P3DATA(:,3);
y=P3DATA(:,4);
z=P3DATA(:,5);

figure;
%plot(X, R,'color','r','Marker','s');
scatter(x,r);
title('X vs R');
figure;
scatter(y,c);
title('Y vs C');
%scatter(Y, C,'color','g','Marker','s');
```





C. Part 3

After the points were elimiated (they were commented out in part 1), the parameters were recalculated using the pseudo-inverse solution to $Ax = b$. This solution uses 17 points.

```
Br=r(:).*z(:); % r*Z
Bc=c(:).*z(:);
A1=[z(:),x(:)];
A2=[z(:),y(:)];

X1=(inv(A1'*A1)*A1'*Br)
X2=(inv(A2'*A2)*A2'*Bc)
```

```
X1 =

  294.4145
  975.2616


X2 =

  449.7802
  925.4220
```

D. Part 4

Using the new calibration parameters, the new distance to the cantelope is calculated (answer is in meters) below. This is a larger distance than part 1 gave, but I trust this measurement much more because the bad calibration parameters were eliminated.

```
stereoParams.baseline=20e-2;
stereoParams.u0=ceil(X1(1)); %I NEED TO ASK
    BELKHOUCHE ABOUT THIS
stereoParams.v0=ceil(X2(1));
stereoParams.alpha_u=X1(2);
stereoParams.alpha_v=X2(2);

cantelopeDistance_bad = testPhysics_alphas(
    stereoParams,cantelope_coords)
```

```
cantelopeDistance_bad =

    0.3411    0.0355    0.9115
```

IV. Problem 4: Robust Camera Calibrator

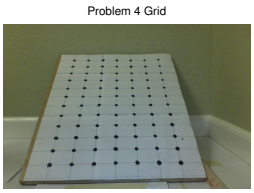
A. Part 1

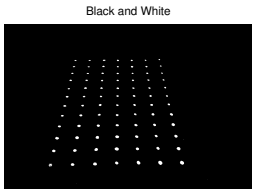
The real world coordinates of the grid are given by the following for loop. It is noted that the Z distance is given by the equation $Z = \frac{Y}{\tan\beta}$.

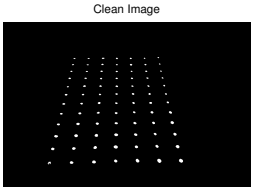
```
clear X Y Z Board dX dY realWorld bw clean S
    stats; close all; clc;
Board=imread('../../../Pictures/phMAC3.jpg');


beta=38.2; %degrees
dX=50.4e-3;
dY=37.8e-3;
% Z=Y/tan(beta);
realWorld=zeros(84,3);
X=0;
Y=0;
Z=0;
for i=1:(84-1)
    if (mod(i,7)==0)
        Y=Y+dY;
        X=-dX;
    end
    realWorld(i+1,1)=X+dX; %[X,Y,Z];
    realWorld(i+1,2)=Y;
    realWorld(i+1,3)=Y/tan(beta);
    X=X+dX;
end


realWorld

figure('name', 'Problem 4');
subplot(221);
imshow(Board), title('Problem 4 Grid');

thresh=0.16; %this is... the best threshold
    value ever recorded by mankind.

bw=im2bw(Board,thresh);
bw=imcomplement(bw);
subplot(222);
imshow(bw); title('Black and White');

S=strel('disk',1);
clean=imopen(bw,S);

subplot(223);
imshow(clean); title('Clean Image');

stats=regionprops(clean);

point=zeros(84,2);
for i=1:length(stats)
```

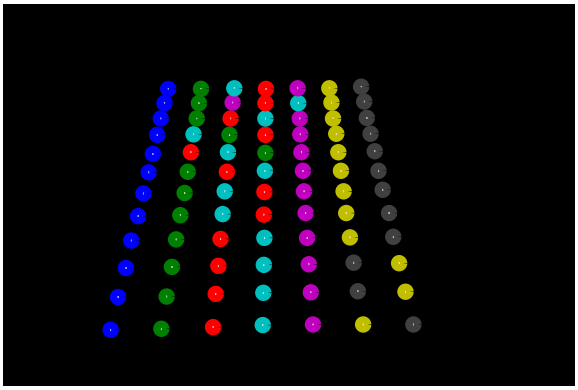
```
    r(i)=round(stats(i).Centroid(1));
    c(i)=round(stats(i).Centroid(2));
    point(i,:)=[r(i),c(i)];
end

%now we have to sort these lists! this is going
     to suck!
pointCopy=point;
sorted=[];

for j=1:7:(12*7)
    for i=1:7;
        [biggestC,I]=max(pointCopy(:,2));
        sorted((i+j)-1,:)=pointCopy(I,:);
        pointCopy(I,:)=[];
    end
end

figure('name','testing ordering');
imshow(clean); hold on;

for i=1:length(sorted)
    scatter(sorted(i,1),sorted(i,2),'LineWidth'
    ,8)
    %pause(0.2);
end
hold off;
% As you can see, the ordering is imperfect,
    but it was the best I could
% do.
for i=1:84
    correspondingMatrix=[sorted(i,1),sorted(i
    ,2),realWorld(i,1),realWorld(i,2),realWorld
    (i,3)]
end
correspondingMatrix;
```

```
realWorld =

         0         0         0
    0.0504         0         0
    0.1008         0         0
    0.1512         0         0
    0.2016         0         0
    0.2520         0         0
    0.3024         0         0
         0    0.0378    0.0690
    0.0504    0.0378    0.0690
    0.1008    0.0378    0.0690
    0.1512    0.0378    0.0690
    0.2016    0.0378    0.0690
    0.2520    0.0378    0.0690
    0.3024    0.0378    0.0690
         0    0.0756    0.1381
    0.0504    0.0756    0.1381
    0.1008    0.0756    0.1381
    0.1512    0.0756    0.1381
    0.2016    0.0756    0.1381
    0.2520    0.0756    0.1381
    0.3024    0.0756    0.1381
         0    0.1134    0.2071
    0.0504    0.1134    0.2071
    0.1008    0.1134    0.2071
    0.1512    0.1134    0.2071
    0.2016    0.1134    0.2071
    0.2520    0.1134    0.2071
    0.3024    0.1134    0.2071
         0    0.1512    0.2762
    0.0504    0.1512    0.2762
    0.1008    0.1512    0.2762
    0.1512    0.1512    0.2762
    0.2016    0.1512    0.2762
    0.2520    0.1512    0.2762
    0.3024    0.1512    0.2762
         0    0.1890    0.3452
    0.0504    0.1890    0.3452
    0.1008    0.1890    0.3452
    0.1512    0.1890    0.3452
    0.2016    0.1890    0.3452
    0.2520    0.1890    0.3452
    0.3024    0.1890    0.3452
         0    0.2268    0.4143
    0.0504    0.2268    0.4143
    0.1008    0.2268    0.4143
    0.1512    0.2268    0.4143
    0.2016    0.2268    0.4143
    0.2520    0.2268    0.4143
    0.3024    0.2268    0.4143
         0    0.2646    0.4833
    0.0504    0.2646    0.4833
    0.1008    0.2646    0.4833
    0.1512    0.2646    0.4833
    0.2016    0.2646    0.4833
    0.2520    0.2646    0.4833
    0.3024    0.2646    0.4833
         0    0.3024    0.5524
    0.0504    0.3024    0.5524
    0.1008    0.3024    0.5524
    0.1512    0.3024    0.5524
    0.2016    0.3024    0.5524
    0.2520    0.3024    0.5524
    0.3024    0.3024    0.5524
         0    0.3402    0.6214
    0.0504    0.3402    0.6214
    0.1008    0.3402    0.6214
    0.1512    0.3402    0.6214
    0.2016    0.3402    0.6214
    0.2520    0.3402    0.6214
    0.3024    0.3402    0.6214
         0    0.3780    0.6905
    0.0504    0.3780    0.6905
    0.1008    0.3780    0.6905
    0.1512    0.3780    0.6905
    0.2016    0.3780    0.6905
    0.2520    0.3780    0.6905
    0.3024    0.3780    0.6905
         0    0.4158    0.7595
    0.0504    0.4158    0.7595
    0.1008    0.4158    0.7595
    0.1512    0.4158    0.7595
    0.2016    0.4158    0.7595
    0.2520    0.4158    0.7595
    0.3024    0.4158    0.7595
```









V. Problem 5

class 1: x class 1: y class 2: x class 2: y This data is not linearly seperable, so Neural Networks will not find a solution. The best option is for us to use the Kernel trick to make the data seperable.

```
close all;
classDATA=[3.0619 3.1131 1.9640 2.5211;
        2.8916 3.0156 2.5730 1.5873;
        3.0245 3.0161 3.0777 3.9750;
        3.1925 3.1589 4.7909 3.2897;
        2.9492 3.2087 3.7973 4.1654;
        3.3784 3.1604 2.5575 2.0913;
        2.7559 2.9062 2.3709 3.2406;
        2.9248 2.9907 4.5338 2.3460;
        3.0105 3.0707 5.7335 4.2750;
        2.6092 2.7486 3.1674 2.4488;]

trainingSet=zeros(2*length(classDATA),2);
for i=1:10
    trainingSet(i,1)   = classDATA(i,1);
    trainingSet(i,2) = classDATA(i,2);
    resultSet(i)       = 1;
end
for i=11:20
    trainingSet(i,1)   = classDATA(i-10,3);
    trainingSet(i,2) = classDATA(i-10,4);
    resultSet(i)=0;
end

plot(classDATA(:,1), classDATA(:,2),'*','Color'
    ,'r'); hold on;
plot(classDATA(:,3), classDATA(:,4),'*','Color'
    ,'g'); hold off;

[weightVectorB, biasB] = customPerceptron(
    trainingSet, resultSet);

disp(['weights: ' num2str(weightVectorB)]);
disp(['bias: ' num2str(biasB)]);

t=0:.001:6;
figure('name','Seperation of class B');
plot(classDATA(1:10,1), classDATA(1:10,2), 's',
    'color','green'), hold on
plot(classDATA(1:10,3), classDATA(1:10,4), 's',
    'color','red'), hold on
plot(t,(-weightVectorB(1)*t-biasB)/
    weightVectorB(2),'color','cyan'); %NN
    solution
legend('class1','class2','NN sol'), hold off
```

```
classDATA =

    3.0619    3.1131    1.9640    2.5211
    2.8916    3.0156    2.5730    1.5873
    3.0245    3.0161    3.0777    3.9750
    3.1925    3.1589    4.7909    3.2897
    2.9492    3.2087    3.7973    4.1654
    3.3784    3.1604    2.5575    2.0913
    2.7559    2.9062    2.3709    3.2406
    2.9248    2.9907    4.5338    2.3460
    3.0105    3.0707    5.7335    4.2750
    2.6092    2.7486    3.1674    2.4488

Neuron input calculation couldn't completed
in timely fashion.

weights: 0.0004626  -0.0017771
bias: -0.014
```

VI. Problem 6

Because this problem uses the same images as Problem 1, I used the same corresponding points in order to define my matrices. For some reason, my epipolar lines are not coming out as expected on my images. I have reason to believe that this is coming from an imperfection in my image selection choices. Perhaps all points need to be taken from the same line?

```
close all;
%reload stereo parameters
r=DATASET(:,1);
c=DATASET(:,2);
x=DATASET(:,3);
y=DATASET(:,4);
z=DATASET(:,5);
rline=[(r(:).*z(:)),-z(:),-x(:)];
cline=[c(:).*z(:),-z(:),-y(:)];
A=[rline];
[U,D,V]=svd(A);
X_1=V(:,end);

A=[cline];
```

```
[U,D,V]=svd(A);
X_2=V(:,end);

Sx=X_1(1);
Sy=abs(X_2(1));
u0=X_1(2)/X_1(1);
v0=abs(X_2(2)/X_2(1));
lambda=(X_1(3)-(X_2(3)))/2;

cantelope_coords=[756,579, 452,544];
kleenex_coords=[1129,598, 859,564];
marker_coords=[562,513, 353,482];
upperLeft_coords=[955,541, 687,507];
upperRight_coords=[1260,551, 992,516];
lowerLeft_coords=[953,662, 683,628];
cantelopeCirc=[756,465, 452,430];
cantelopeAntipode=[756,673, 452,638];


randomPoints=[cantelope_coords;kleenex_coords;
    marker_coords;upperLeft_coords;
                upperRight_coords;
    lowerLeft_coords; cantelopeCirc;
    cantelopeAntipode;];
Rl=randomPoints(:,1);
Cl=randomPoints(:,2);
Rr=randomPoints(:,3);
Cr=randomPoints(:,4);


%build the W matrix
W=zeros(8,9);
for i=1:8%length(DATASET)
    Ul=(Rl(i)-u0)*Sx;
    Ur=(Rr(i)-u0)*Sx;
    Vl=(Cl(i)-v0)*Sy;
    Vr=(Cr(i)-v0)*Sy;
    W(i,:)=[Ul*Ur, Ul*Vr, Ul, Vl*Ur, Vl*Vr, Vl,
     Ur, Vr, 1];
end
[U,D,V]=svd(W);
fundamentalMatrixSVD=V(:,end)
fundamentalMatrixNull=null(W)
F=zeros(3,3);
i=1; j=1;
for n=1:length(fundamentalMatrixNull)
    F(i,j)=fundamentalMatrixNull(n);
    j=j+1;
  if (mod(n,3)==0)
        j=1;
        i=i+1;
    end
end
F

%cantelope line
coords=cantelope_coords;
coords=cantelope_coords;
Rl=coords(1);
Cl=coords(2);
Rr=coords(3);
Cr=coords(4);

Ul=(Rl-u0)*Sx;
Ur=(Rr-u0)*Sx;
Vl=(Cl-v0)*Sy;
Vr=(Cr-v0)*Sy;


% ql=[Ur;Vr;1];
% qr=[Ul;Vl;1];
ql=[Ul;Vl;1];
qr=[Ur;Vr;1];

Lr=F*ql
Ll=F'*qr

[M,N]=size(Left);
figure('name','epipolar line')
imshow(rgb2gray(Left)); hold on;
x=0:1:N;
yl=((-(Lr(1)*x)-(Lr(3)))/Lr(2));
yr=((-(Ll(1)*x)-(Ll(3)))/Ll(1));
plot(x,yl,'Color','g'); hold on;
plot(x,yr,'Color','r');hold off;
```

APPENDIX
FUNCTIONS CALLED

```
function [p] = testPhysics_alphas(stereoParams,
     imageCoords);
    %knowns (from calibration)
    alpha_v=stereoParams.alpha_v;
    alpha_u=stereoParams.alpha_u;
    u0=stereoParams.u0;
    v0=stereoParams.v0;
    baseline=stereoParams.baseline;

    % IMAGE COORDINATES ARE IN FORM [leftR,
    leftC, rightR, rightC]
    Rl=imageCoords(1);
    Cl=imageCoords(2);
    Rr=imageCoords(3);
    %Cr=imageCoords(4);


    %triangulations
    nUl=(Rl-u0)*(alpha_v/alpha_u);
    nUr=(Rr-u0)*(alpha_v/alpha_u);
    nVl=(Cl-v0)*(alpha_u/alpha_v);
    %Vr=(Cr-v0)*Sy;
    Z=(alpha_v*baseline)/(nUl-nUr);
    X=((Rl-u0)*Z)/alpha_u;
    Y=((Cl-v0)*Z)/alpha_v;
    p=[X,Y,Z];


end
```

```
function [p] = testPhysics(stereoParams,
    imageCoords);
    %knowns (from calibration)
    lambda=stereoParams.lambda;
    Sx=stereoParams.Sx;
    Sy=stereoParams.Sy;
    u0=stereoParams.u0;
    v0=stereoParams.v0;
    baseline=stereoParams.baseline;

    % IMAGE COORDINATES ARE IN FORM [leftR,
    leftC, rightR, rightC]
    Rl=imageCoords(1);
    Cl=imageCoords(2);
    Rr=imageCoords(3);
    %Cr=imageCoords(4);


    %triangulations
    Ul=(Rl-u0)*Sx;
    Ur=(Rr-u0)*Sx;
```

```
    Vl=(Cl-v0)*Sy;
    %Vr=(Cr-v0)*Sy;

    X=(baseline*Ul)/(Ul-Ur);
    Y=(baseline*Vl)/(Ul-Ur);
    Z=(lambda*baseline)/(Ul-Ur);


    p=[X,Y,Z];


end
```

```
function [weightVector, bias] =
    customPerceptron(trainingSet, resultSet)
    %build training set
    [m, n] = size(trainingSet);
    weightVector = ones(1,n);
    for i=1:n
        weightVector(i) = weightVector(i)./(5);
     %initialize to small numbers. 12
    guaranteed to be random, it was chosen
    through a set of dice rolls.
    end

    %weightVector = zeros(1,n);
    threshold = 0;%threshold to decide if the
    output is good or bad. usually this is 0
    error_count = 1;
    bias = 0.1;
    iterationNo = 1;
    learningRate = 0.001;
    % training phase
    while (error_count > 0)
        error_count = 0;
        for i=1:m
            gx=dot(weightVector,trainingSet(i
    ,:))+bias;
                if (gx > threshold)
                    result = 1;
                else
                    result = 0;
                end

                error = resultSet(i)-result;
                if (error ~= 0)
                    error_count = error_count +
     1;
                    weightVector = weightVector
     + (learningRate*(error))*trainingSet(i,1:n
    );
                    bias = bias + learningRate*
    error;
                end
        end

        if (iterationNo >= 1000)
            disp('Neuron input calculation
    couldn''t completed in timely fashion.');
            break
        end
        iterationNo = iterationNo +1;
    end
    disp(['answer converged in ' num2str(
    iterationNo) ' iterations']);
end
```