Assignment 5

Arian Nedjabat BA 20661686

Problem 1

a & b)

```
clc, clear

p1 = [5,4];
p2 = [1,7];
p3 = [6,3];
p4 = [-3,8];

%slope1 = (p2(2)-p1(2))/(p2(1)-p1(1));


denom = (p1(1)-p2(1))*(p3(2)-p4(2)) - (p1(2)-p2(2))*(p3(1)-p4(1));
numerx = (p1(1)*p2(2)-p1(2)*p2(1))*(p3(1)-p4(1)) - (p1(1)-p2(1))*(p3(1)*p4(2)-
p3(2)*p4(1));
numery = (p1(1)*p2(2)-p1(2)*p2(1))*(p3(2)-p4(2)) - (p1(2)-p2(2))*(p3(1)*p4(2)-
p3(2)*p4(1));

px = numerx/denom
py = numery/denom



l1 = cross([p1 1],[p2 1]);
l2 = cross([p3 1],[p4 1]);
intercept = cross(l1,l2);

px_hc = intercept(1)/intercept(3)
py_hc = intercept(2)/intercept(3)
```

```
px = 7.2857
py = 2.2857
```

```
px_hc = 7.2857
py_hc = 2.2857
```

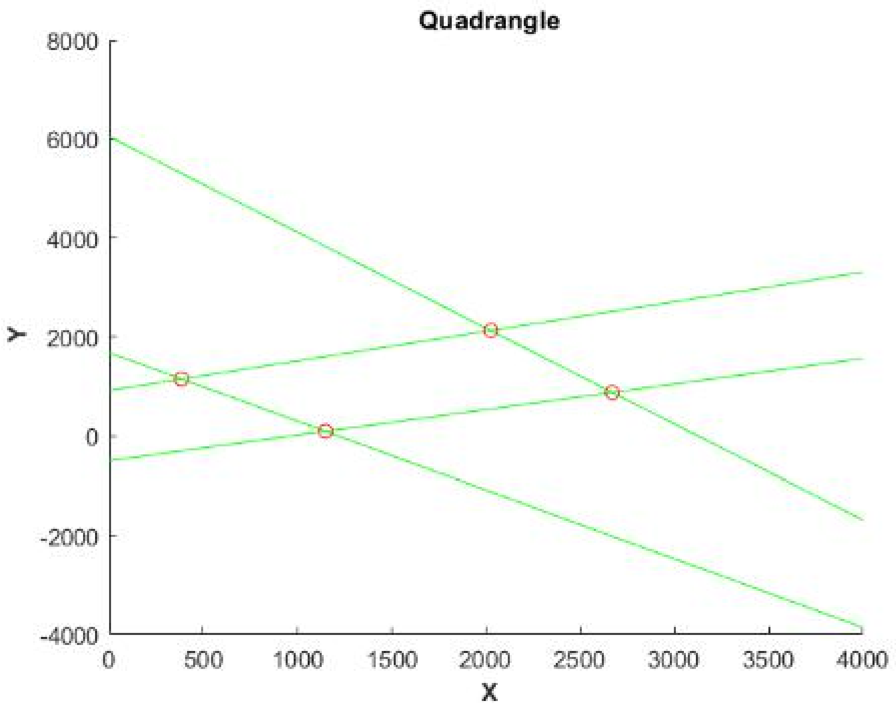
Problem 2

```
clc, clear, clf

l1 = [0.59756 -1 921.94];
l2 = [-1.935 -1 6050.3];
l3 = [0.51575 -1 -495.56];
l4 = [-1.3858 -1 1685.5];
global line_list point_list quadrangle

line_list = [l1; l2; l3; l4];


intercept = cross(l1,l2);
p1 = [intercept(1)/intercept(3), intercept(2)/intercept(3)];

intercept = cross(l1,l3);
p2 = [intercept(1)/intercept(3), intercept(2)/intercept(3)];

intercept = cross(l1,l4);
p3 = [intercept(1)/intercept(3), intercept(2)/intercept(3)];

intercept = cross(l2,l3);
p4 = [intercept(1)/intercept(3), intercept(2)/intercept(3)];

intercept = cross(l2,l4);
p5 = [intercept(1)/intercept(3), intercept(2)/intercept(3)];

intercept = cross(l3,l4);
p6 = [intercept(1)/intercept(3), intercept(2)/intercept(3)];

point_list = [p1; p2; p3; p4; p5; p6]

for ii = [1:3]
    point1 = point_list(ii,:); %pick random point to start. If need 4 pts but
have 6, max loop 3 times
    quad_linesP1 = find_lines(point1); %find lines that make up intersection
point 1
    quad_p2 = find_pts(quad_linesP1(1,:),point1);%pick one of the two lines
found earlier and find the points on the new line (candidates for point2)
    for jj = 1:size(quad_p2,1) %loop through point 2 candidates
        point2 = quad_p2(jj,:);
        quad_linesP2 = find_lines(point2); %find lines connected with point2
        line2 =
quad_linesP2(~ismember(quad_linesP2,quad_linesP1(1,:),'rows'),:); %drop the line
we selected earlier from the list

        quad_p3 = find_pts(line2,point2);% Point 3 candidates
        for kk = 1:size(quad_p3,1) %loop through point 3 candidates
            point3 = quad_p3(kk,:);
            quad_linesP3 = find_lines(point3);%find lines connected with point3
            line3 = quad_linesP3(~ismember(quad_linesP3,line2,'rows'),:); %drop
the line we selected earlier from the list

            quad_p4 = find_pts(line3,point3);% Point 4 candidates
            for mm = 1:size(quad_p3,1) %loop through point 4 candidates
```

```
                point4 = quad_p4(mm,:);
                if (point4 == point1)
                    "Triangle";
                    continue
                end

                quad_linesP4 = find_lines(point4);%find lines connected with
point3
                line4 = quad_linesP4(~ismember(quad_linesP4,line3,'rows'),:);
%drop the line we selected earlier from the list
                if (line4 == quad_linesP1(2,:)) %if point 4 belong to the other
line that intersected point 1 but was not chosen, quadrangle found
                    quadrangle = [point1;point2;point3;point4]
                end
            end

        end
    end
end

syms x

y1 = l1(1,1)*x+l1(1,3);
y2 = l2(1,1)*x+l2(1,3);
y3 = l3(1,1)*x+l3(1,3);
y4 = l4(1,1)*x+l4(1,3);

hold on;
xlabel('\bf X');
ylabel('\bf Y');
title('\bf Quadrangle');
xlim([0,4000]);
fplot([y1,y2,y3,y4],'g');
plot(quadrangle(:,1),quadrangle(:,2),'ro');
```

```
function lines = find_lines(pt) % FINDS THE LINES THAT MAKE UP PT INTERSECTION
    lines = [];
    global line_list
    for ii = 1:length(line_list)
        if (abs(sum(line_list(ii,:)*[pt 1]')) < 1*10^-8 ) %Checking if pt is on
line
            lines = [lines; line_list(ii,:)];
        end
    end
end

function pts = find_pts(line,poi) %FIND ALL INTERSECTION PTS THAT BELONG TO A
SPECIFIC LINE. ONLY RETURN NEAREST PTS IN EACH DIRECTION ALONG LINE
    pts = [];
    global point_list
    for ii = 1:length(point_list)
        if (abs(sum([point_list(ii,:) 1].*line)) < 1*10^-8 & point_list(ii,:) ~=
poi) %Checking if pt is on line and avoid including previous point (poi)
            pts = [pts; point_list(ii,:)];
        end
    end
    if((pts(1,1)> poi(1) && pts(2,1)> poi(1)) || (pts(1,1)< poi(1) && pts(2,1)<
poi(1))) %checking for nearest point
        dist1 = norm([pts(1,:)-poi]);
        dist2 = norm([pts(2,:)-poi]);
        if (dist1 < dist2)
            pts = pts(1,:);
        else
             pts = pts(2,:);
        end
    end

end
```

```
quadrangle = 4×2
10^3  ×
      0.3850    1.1520
      2.0250    2.1320
      2.6710    0.8820
      1.1470    0.0960
```



Problem 3

```
%Problem 3

clf

H = [0.4493 -0.3531 1147.5; 0.2448 0.41441 96.5; 0 0 1];
pts_transformed = [];

for ii = 1:size(quadrangle,1)
    pts_transformed = [pts_transformed; [H*[quadrangle(ii,:) 1]']'];
end
quadrangle
pts_transformed
```

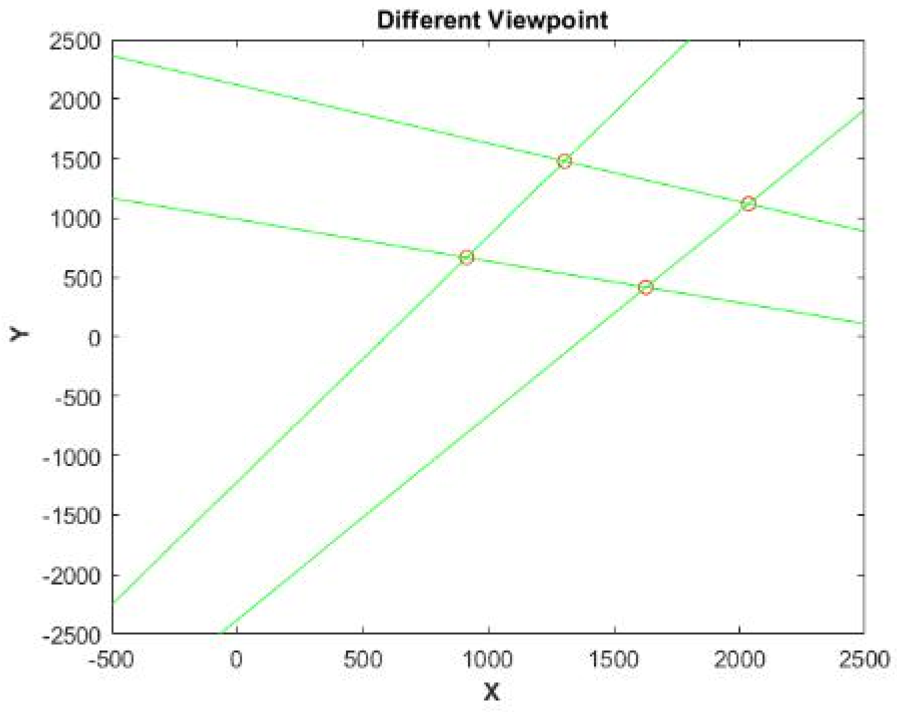
```
line1 = cross(pts_transformed(1,:),pts_transformed(2,:));
line2 = cross(pts_transformed(1,:),pts_transformed(4,:));
line3 = cross(pts_transformed(3,:),pts_transformed(2,:));
line4 = cross(pts_transformed(3,:),pts_transformed(4,:));
lines = [line1;line2;line3;line4];


syms x

y1 = (lines(1,1)*x+lines(1,3))/-lines(1,2)
y2 = (lines(2,1)*x+lines(2,3))/-lines(2,2)
y3 = (lines(3,1)*x+lines(3,3))/-lines(3,2)
y4 = (lines(4,1)*x+lines(4,3))/-lines(4,2)


figure(2)
plot(pts_transformed(:,1),pts_transformed(:,2),'ro');
hold on;
fplot([y1,y2,y3,y4],'g');
xlabel('\bf X');
ylabel('\bf Y');
title('\bf Different Viewpoint');
```

y1 = $2.07\,x - 1220.0$

y2 = $989.0 - 0.351\,x$

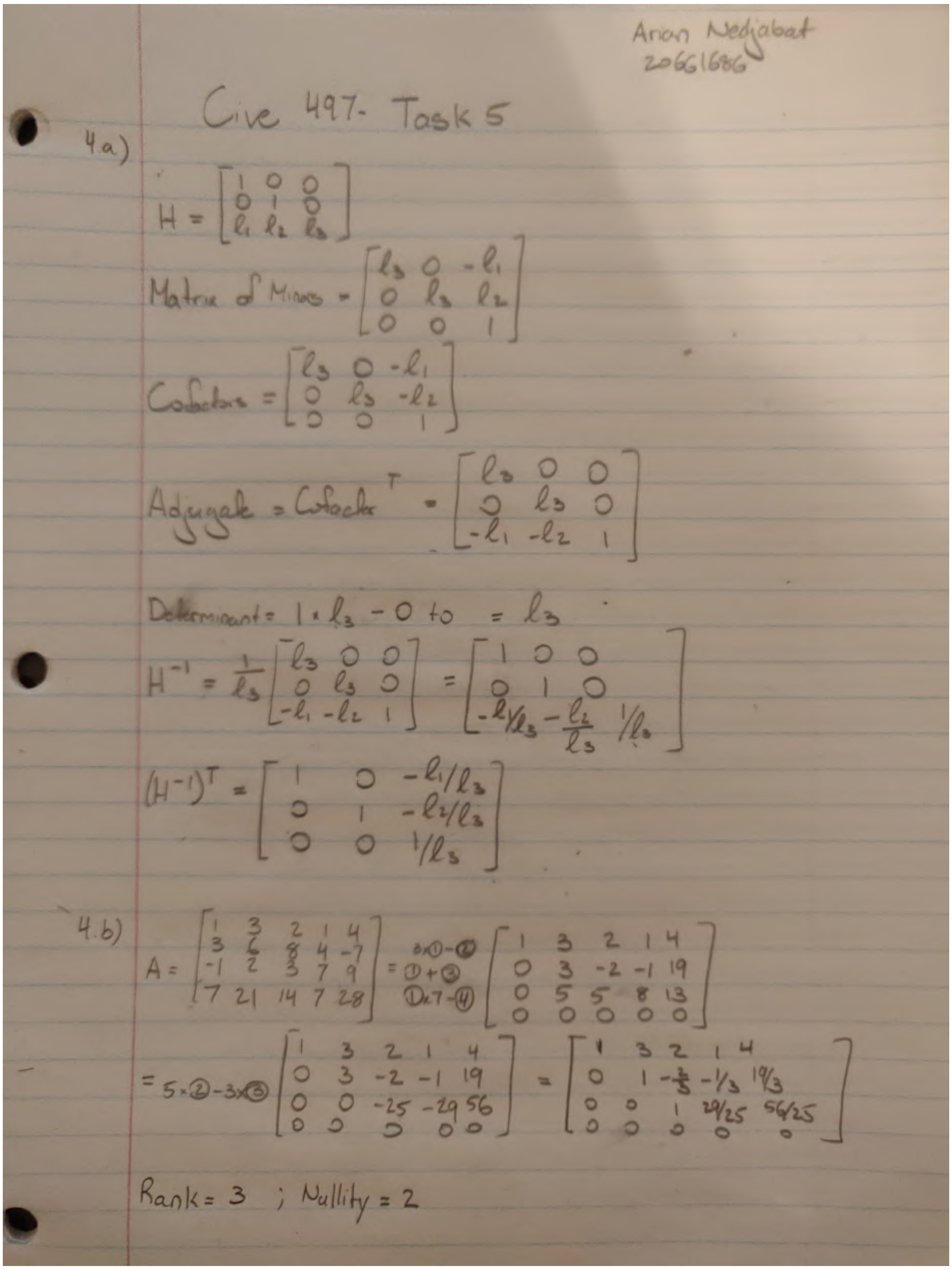
y3 = $2120.0 - 0.492\,x$

y4 = $1.72\,x - 2380.0$



Problem 4

a & b)



Arian Nedjabat
20661686

Cive 497- Task 5

4.a)

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \ell_1 & \ell_2 & \ell_3 \end{bmatrix}$$

$$\text{Matrix of Minors} = \begin{bmatrix} \ell_3 & 0 & -\ell_1 \\ 0 & \ell_3 & \ell_2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{Cofactors} = \begin{bmatrix} \ell_3 & 0 & -\ell_1 \\ 0 & \ell_3 & -\ell_2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{Adjugate} = \text{Cofactor}^T = \begin{bmatrix} \ell_3 & 0 & 0 \\ 0 & \ell_3 & 0 \\ -\ell_1 & -\ell_2 & 1 \end{bmatrix}$$

$$\text{Determinant} = 1 \cdot \ell_3 - 0 + 0 = \ell_3$$

$$H^{-1} = \frac{1}{\ell_3} \begin{bmatrix} \ell_3 & 0 & 0 \\ 0 & \ell_3 & 0 \\ -\ell_1 & -\ell_2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -\ell_1/\ell_3 & -\frac{\ell_2}{\ell_3} & 1/\ell_3 \end{bmatrix}$$

$$(H^{-1})^T = \begin{bmatrix} 1 & 0 & -\ell_1/\ell_3 \\ 0 & 1 & -\ell_2/\ell_3 \\ 0 & 0 & 1/\ell_3 \end{bmatrix}$$

4.b)

$$A = \begin{bmatrix} 1 & 3 & 2 & 1 & 4 \\ 3 & 6 & 8 & 4 & -7 \\ -1 & 2 & 3 & 7 & 9 \\ 7 & 21 & 14 & 7 & 28 \end{bmatrix} = \begin{matrix} 3\times① - ② \\ ① + ③ \\ ①\times 7 - ④ \end{matrix} \begin{bmatrix} 1 & 3 & 2 & 1 & 4 \\ 0 & 3 & -2 & -1 & 19 \\ 0 & 5 & 5 & 8 & 13 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$= 5\times② - 3\times③ \begin{bmatrix} 1 & 3 & 2 & 1 & 4 \\ 0 & 3 & -2 & -1 & 19 \\ 0 & 0 & -25 & -29 & 56 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 2 & 1 & 4 \\ 0 & 1 & -\frac{2}{3} & -1/3 & 19/3 \\ 0 & 0 & 1 & 29/25 & 56/25 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Rank = 3 ; Nullity = 2

Check with Matlab

```
clc, clear

A = [1 3 2 1 4; 3 6 8 4 -7; -1 2 3 7 9; 7 21 14 7 28];

A_rref = rref(A)
null_space = null(A)
nullity = size(null_space,2)
A_rank = rank(A)
```

```
A_rref = 4x5
    1.0000         0         0   -2.6400   -6.0400
         0    1.0000         0    0.4400    4.8400
         0         0    1.0000    1.1600   -2.2400
         0         0         0         0         0

null_space = 5x2
    0.8981    0.0315
   -0.3121   -0.5835
   -0.1873    0.7244
    0.2433   -0.3332
    0.0424    0.1508

nullity = 2
A_rank = 3
```

Rank is the dimensions of the row space of a matrix. It can be seen as the count of the number of columns that have a value of 1 in rref. It represents the minimum number of vectors required to make up that subspace. Row space are those vectors. Nullity is the number of vectors that can be placed that are orthogonal to both the subspace and each other. The null space are those orthogonal vectors. The nullity is the count of the columns which do not have a value of 1.

c)

```
A2 = [1 3 2 1; 3 6 8 4; 7 21 14 7; 1 21 14 -3; -1 2 3 7]
null_space2 = null(A2)
A2_rank = rank(A2)
A2_nullity = size(null_space2,2)
fprintf('Vector x does not exist.')

A3 = [1 3 2 1; 3 6 8 4; 1 21 14 -3; -1 2 3 7]
null_space3 = null(A3)
A3_rank = rank(A3)
A3_nullity = size(null_space3,2)
fprintf('Vector x does not exist.')

A4 = [1 3 2 1; 3 6 8 4; 1 21 14 -3]
null_space4 = null(A4)
A4_rank = rank(A4)
A4_nullity = size(null_space4,2)
fprintf('Vector x does exist.')
```

```
A2 = 5x4
       1     3     2     1
       3     6     8     4
       7    21    14     7
       1    21    14    -3
      -1     2     3     7

null_space2 =

  4x0 empty double matrix
A2_rank = 4
A2_nullity = 0
Vector x does not exist.
A3 = 4x4
       1     3     2     1
       3     6     8     4
       1    21    14    -3
      -1     2     3     7

null_space3 =

  4x0 empty double matrix
A3_rank = 4
A3_nullity = 0
Vector x does not exist.
A4 = 3x4
       1     3     2     1
       3     6     8     4
       1    21    14    -3

null_space4 = 4x1
     -0.8481
      0.1414
     -0.0424
      0.5089

A4_rank = 3
A4_nullity = 1
Vector x does exist.
```

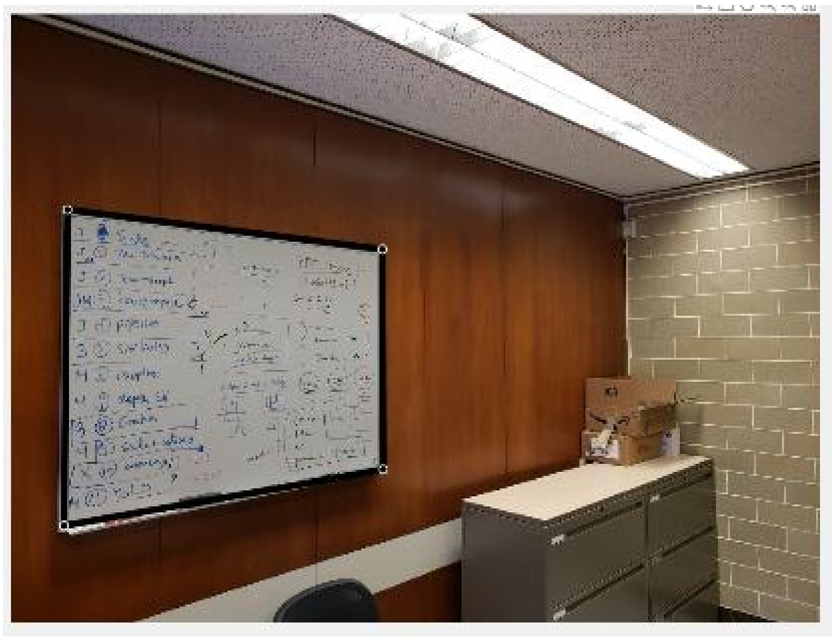
Problem 5

a)

```
clear; close all; clc; format shortG;

%Parameter
imgBoardFile = 'prob5_1.JPG';
imgPicFile = 'jail.jpg';

info = imfinfo(imgPicFile);
sizePic = [info.Width info.Height];

%Step1: Pick four corners of your white board in (a)
imgBoard = imread(imgBoardFile);
figure(1); imshow(imgBoard);
p = drawpolygon('LineWidth',5,'Color','black');
```

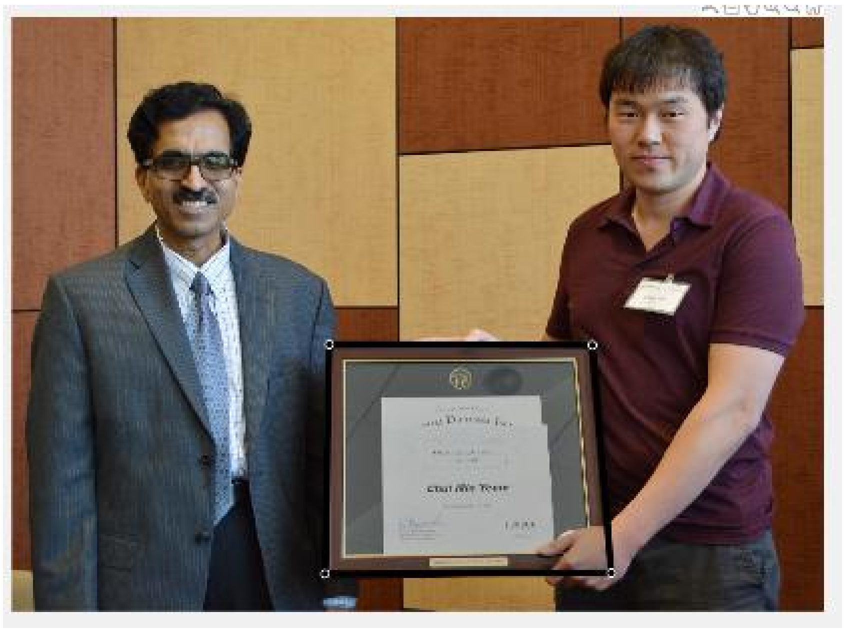
```
corner = p.Position;

%Step2: Compute H (Your Section)
original = [0 0; sizePic(1,1) 0; sizePic(1,1) sizePic(1,2); 0 sizePic(1,2)];
A = [];
for ii = 1:size(original,1)
    row1 = [original(ii,1) original(ii,2) 1 0 0 0 -original(ii,1)*corner(ii,1) -
corner(ii,1)*original(ii,2) -corner(ii,1)];
    row2 = [0 0 0 original(ii,1) original(ii,2) 1 -original(ii,1)*corner(ii,2) -
corner(ii,2)*original(ii,2) -corner(ii,2)];
    A = [A; row1 ; row2];
end
null_space = null(A);
H2 = [null_space(1:3)'; null_space(4:6)'; null_space(7:9)'];

%Step3: Overlay your picture (I think there is a better way to do this)
imgPic = imread(imgPicFile);
[imgPicTran, RB] = imwarp(imgPic, projective2d(H2'));
BWPic = roipoly(imgPicTran, corner(:,1)-RB.XWorldLimits(1), corner(:,2)-
RB.YWorldLimits(1));


BWBoard = ~roipoly(imgBoard, corner(:,1), corner(:,2));
RA = imref2d(size(BWBoard));

imgBoardMask = bsxfun(@times, imgBoard, cast(BWBoard, 'like', imgBoard));
imgPicTranMask = bsxfun(@times, imgPicTran, cast(BWPic, 'like', imgPicTran));

imgFinal(:,:,1) = imfuse(imgBoardMask(:,:,1),RA,
imgPicTranMask(:,:,1),RB,'diff');
imgFinal(:,:,2) = imfuse(imgBoardMask(:,:,2),RA,
imgPicTranMask(:,:,2),RB,'diff');
imgFinal(:,:,3) = imfuse(imgBoardMask(:,:,3),RA,
imgPicTranMask(:,:,3),RB,'diff');

imshow(imgFinal);

imgPicFile = 'clock_wall.PNG';

info = imfinfo(imgPicFile);
sizePic = [info.Width info.Height];

%Step1: Pick four corners of your white board in (a) (repeat)
imgBoard = imgFinal;
figure(2); imshow(imgBoard);
p = drawpolygon('LineWidth',5,'Color','black');
corner = p.Position;

%Step2: Compute H (Your Section) (repeat)
original = [0 0; sizePic(1,1) 0; sizePic(1,1) sizePic(1,2); 0 sizePic(1,2)];
A = [];
for ii = 1:size(original,1)
    row1 = [original(ii,1) original(ii,2) 1 0 0 0 -original(ii,1)*corner(ii,1) -
corner(ii,1)*original(ii,2) -corner(ii,1)];
    row2 = [0 0 0 original(ii,1) original(ii,2) 1 -original(ii,1)*corner(ii,2) -
corner(ii,2)*original(ii,2) -corner(ii,2)];
    A = [A; row1 ; row2];
end
```

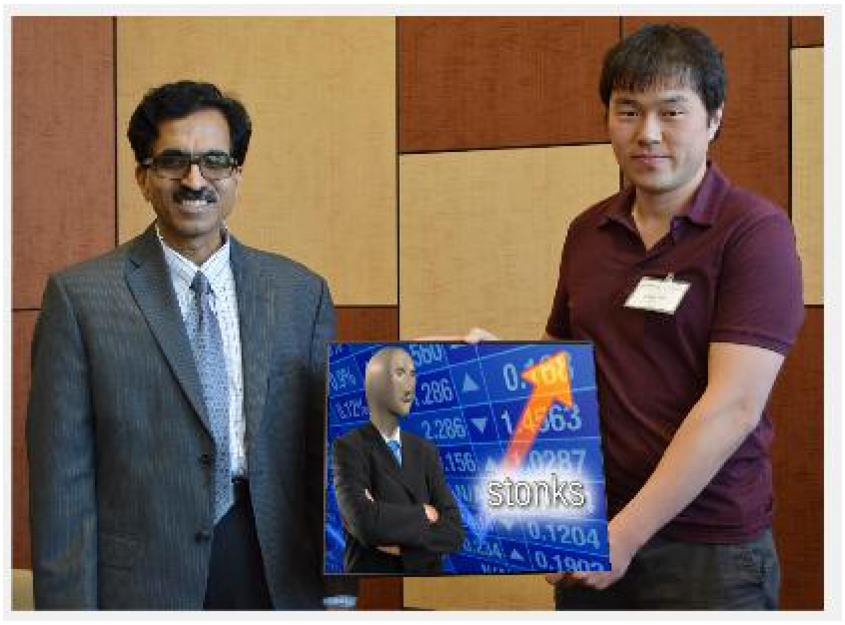
```
null_space = null(A);
H2 = [null_space(1:3)'; null_space(4:6)'; null_space(7:9)'];

%Step3: Overlay your picture (Repeat)
imgPic = imread(imgPicFile);
[imgPicTran, RB] = imwarp(imgPic, projective2d(H2'));
BWPic = roipoly(imgPicTran, corner(:,1)-RB.XWorldLimits(1), corner(:,2)-
RB.YWorldLimits(1));


BWBoard = ~roipoly(imgBoard, corner(:,1), corner(:,2));
RA = imref2d(size(BWBoard));

imgBoardMask = bsxfun(@times, imgBoard, cast(BWBoard, 'like', imgBoard));
imgPicTranMask = bsxfun(@times, imgPicTran, cast(BWPic, 'like', imgPicTran));

imgFinal(:,:,1) = imfuse(imgBoardMask(:,:,1),RA,
imgPicTranMask(:,:,1),RB,'diff');
imgFinal(:,:,2) = imfuse(imgBoardMask(:,:,2),RA,
imgPicTranMask(:,:,2),RB,'diff');
imgFinal(:,:,3) = imfuse(imgBoardMask(:,:,3),RA,
imgPicTranMask(:,:,3),RB,'diff');

imshow(imgFinal); imwrite(imgFinal, 'result_5_1.jpg');
```



b)

```
clear; close all; clc; format shortG;

%Parameter
imgBoardFile = 'prob5_2.JPG';
imgPicFile = 'stonks.PNG';

info = imfinfo(imgPicFile);
sizePic = [info.Width info.Height];

%Step1: Pick four corners of your white board in (a)
imgBoard = imread(imgBoardFile);
figure(1); imshow(imgBoard);
p = drawpolygon('LineWidth',5,'Color','black');
corner = p.Position;

%Step2: Compute H (Your Section)
original = [0 0; sizePic(1,1) 0; sizePic(1,1) sizePic(1,2); 0 sizePic(1,2)];
A = [];
for ii = 1:size(original,1)
    row1 = [original(ii,1) original(ii,2) 1 0 0 0 -original(ii,1)*corner(ii,1) -
corner(ii,1)*original(ii,2) -corner(ii,1)];
    row2 = [0 0 0 original(ii,1) original(ii,2) 1 -original(ii,1)*corner(ii,2) -
corner(ii,2)*original(ii,2) -corner(ii,2)];
    A = [A; row1 ; row2];
end
null_space = null(A);
H2 = [null_space(1:3)'; null_space(4:6)'; null_space(7:9)'];

%H = ComputeH(corner, sizePic);

%Step3: Overlay your picture (I think there is a better way to do this)
imgPic = imread(imgPicFile);
[imgPicTran, RB] = imwarp(imgPic, projective2d(H2'));
BWPic = roipoly(imgPicTran, corner(:,1)-RB.XWorldLimits(1), corner(:,2)-
RB.YWorldLimits(1));


BWBoard = ~roipoly(imgBoard, corner(:,1), corner(:,2));
RA = imref2d(size(BWBoard));

imgBoardMask = bsxfun(@times, imgBoard, cast(BWBoard, 'like', imgBoard));
imgPicTranMask = bsxfun(@times, imgPicTran, cast(BWPic, 'like', imgPicTran));

imgFinal(:,:,1) = imfuse(imgBoardMask(:,:,1),RA,
imgPicTranMask(:,:,1),RB,'diff');
imgFinal(:,:,2) = imfuse(imgBoardMask(:,:,2),RA,
imgPicTranMask(:,:,2),RB,'diff');
imgFinal(:,:,3) = imfuse(imgBoardMask(:,:,3),RA,
imgPicTranMask(:,:,3),RB,'diff');

imshow(imgFinal); imwrite(imgFinal, 'result_5_2.jpg');
```





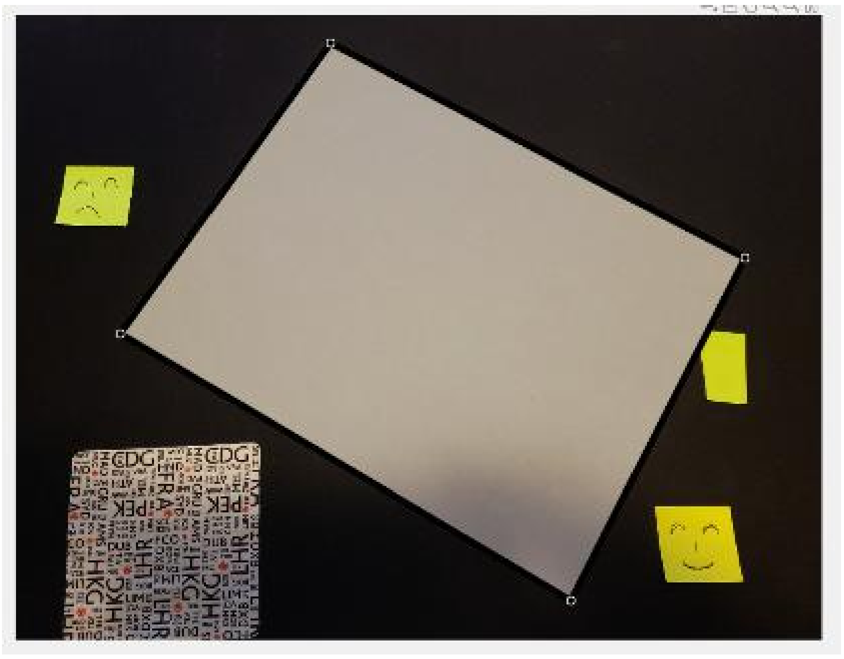
Problem 6

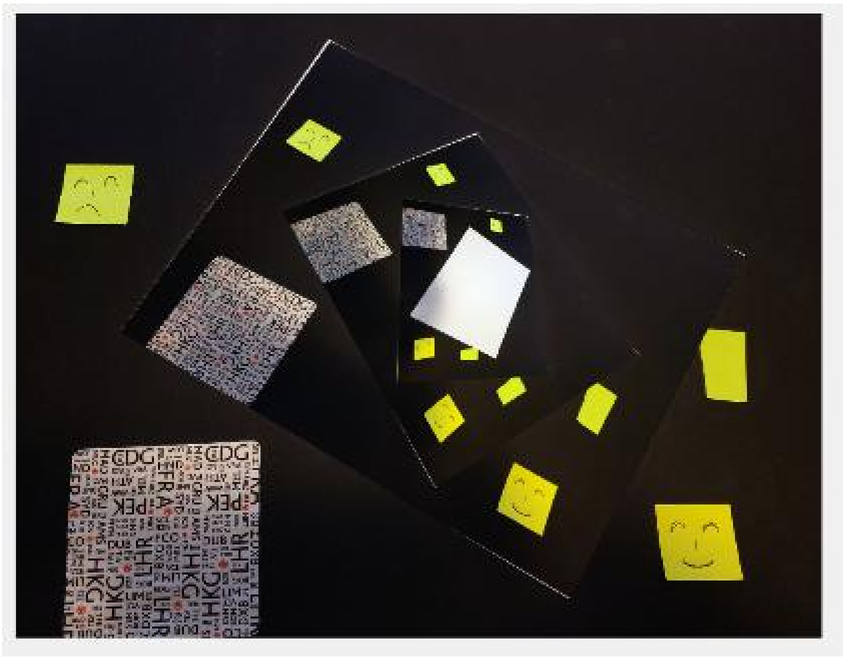
a)

```
clear; close all; clc; format shortG;

%Parameter
imgBoardFile = 'prob6.JPG';
imgPicFile = 'prob6.JPG';

info = imfinfo(imgPicFile);
sizePic = [info.Width info.Height];

%Step1: Pick four corners of your white board in (a)
imgBoard = imread(imgBoardFile);
figure(1); imshow(imgBoard);
p = drawpolygon('LineWidth',5,'Color','black');
corner = p.Position;

%Step2: Compute H (Your Section)
```

```
original = [0 0; sizePic(1,1) 0; sizePic(1,1) sizePic(1,2); 0 sizePic(1,2)];
A = [];
for ii = 1:size(original,1)
    row1 = [original(ii,1) original(ii,2) 1 0 0 0 -original(ii,1)*corner(ii,1) -
corner(ii,1)*original(ii,2) -corner(ii,1)];
    row2 = [0 0 0 original(ii,1) original(ii,2) 1 -original(ii,1)*corner(ii,2) -
corner(ii,2)*original(ii,2) -corner(ii,2)];
    A = [A; row1 ; row2];
end
null_space = null(A);
H2 = [null_space(1:3)'; null_space(4:6)'; null_space(7:9)'];

%H = ComputeH(corner, sizePic);

%Step3: Overlay your picture (I think there is a better way to do this)
imgPic = imread(imgPicFile);
[imgPicTran, RB] = imwarp(imgPic, projective2d(H2'));
BWPic = roipoly(imgPicTran, corner(:,1)-RB.XWorldLimits(1), corner(:,2)-
RB.YWorldLimits(1));


BWBoard = ~roipoly(imgBoard, corner(:,1), corner(:,2));
RA = imref2d(size(BWBoard));

imgBoardMask = bsxfun(@times, imgBoard, cast(BWBoard, 'like', imgBoard));
imgPicTranMask = bsxfun(@times, imgPicTran, cast(BWPic, 'like', imgPicTran));

imgFinal(:,:,1) = imfuse(imgBoardMask(:,:,1),RA,
imgPicTranMask(:,:,1),RB,'diff');
imgFinal(:,:,2) = imfuse(imgBoardMask(:,:,2),RA,
imgPicTranMask(:,:,2),RB,'diff');
imgFinal(:,:,3) = imfuse(imgBoardMask(:,:,3),RA,
imgPicTranMask(:,:,3),RB,'diff');

imshow(imgFinal);

imgPic = imgFinal;
[imgPicTran, RB] = imwarp(imgPic, projective2d(H2'));
BWPic = roipoly(imgPicTran, corner(:,1)-RB.XWorldLimits(1), corner(:,2)-
RB.YWorldLimits(1));


BWBoard = ~roipoly(imgBoard, corner(:,1), corner(:,2));
RA = imref2d(size(BWBoard));

imgBoardMask = bsxfun(@times, imgBoard, cast(BWBoard, 'like', imgBoard));
imgPicTranMask = bsxfun(@times, imgPicTran, cast(BWPic, 'like', imgPicTran));

imgFinal2(:,:,1) = imfuse(imgBoardMask(:,:,1),RA,
imgPicTranMask(:,:,1),RB,'diff');
imgFinal2(:,:,2) = imfuse(imgBoardMask(:,:,2),RA,
imgPicTranMask(:,:,2),RB,'diff');
imgFinal2(:,:,3) = imfuse(imgBoardMask(:,:,3),RA,
imgPicTranMask(:,:,3),RB,'diff');

imshow(imgFinal2);

imgPic = imgFinal2;
```

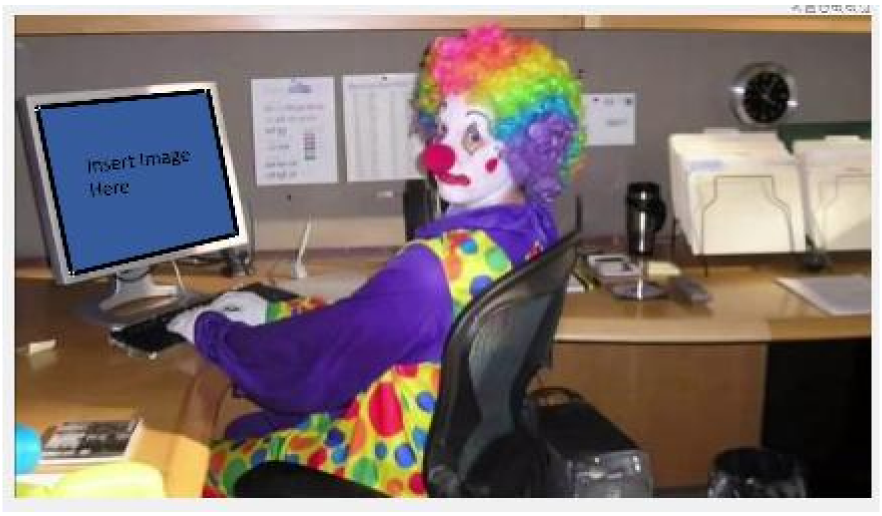
```
[imgPicTran, RB] = imwarp(imgPic, projective2d(H2'));
BWPic = roipoly(imgPicTran, corner(:,1)-RB.XWorldLimits(1), corner(:,2)-
RB.YWorldLimits(1));


BWBoard = ~roipoly(imgBoard, corner(:,1), corner(:,2));
RA = imref2d(size(BWBoard));

imgBoardMask = bsxfun(@times, imgBoard, cast(BWBoard, 'like', imgBoard));
imgPicTranMask = bsxfun(@times, imgPicTran, cast(BWPic, 'like', imgPicTran));

imgFinal3(:,:,1) = imfuse(imgBoardMask(:,:,1),RA,
imgPicTranMask(:,:,1),RB,'diff');
imgFinal3(:,:,2) = imfuse(imgBoardMask(:,:,2),RA,
imgPicTranMask(:,:,2),RB,'diff');
imgFinal3(:,:,3) = imfuse(imgBoardMask(:,:,3),RA,
imgPicTranMask(:,:,3),RB,'diff');

imshow(imgFinal3); imwrite(imgFinal3, 'result_6.jpg');
```

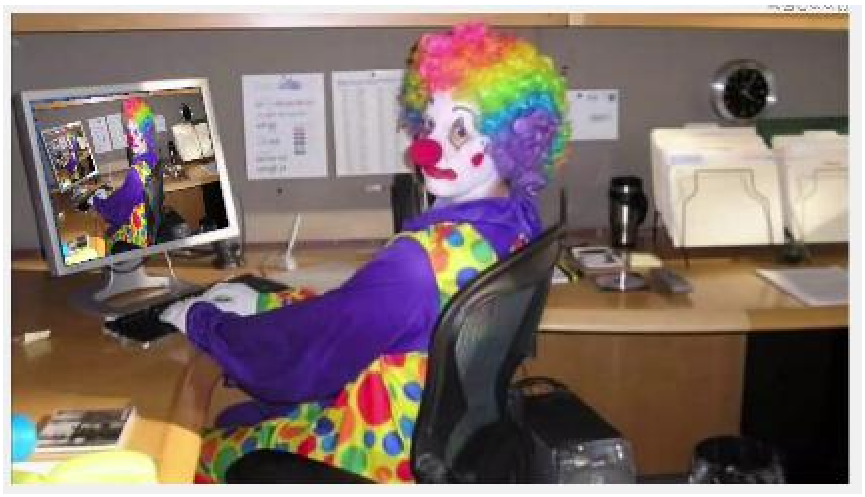




b)

The same code found in a) was used in b).

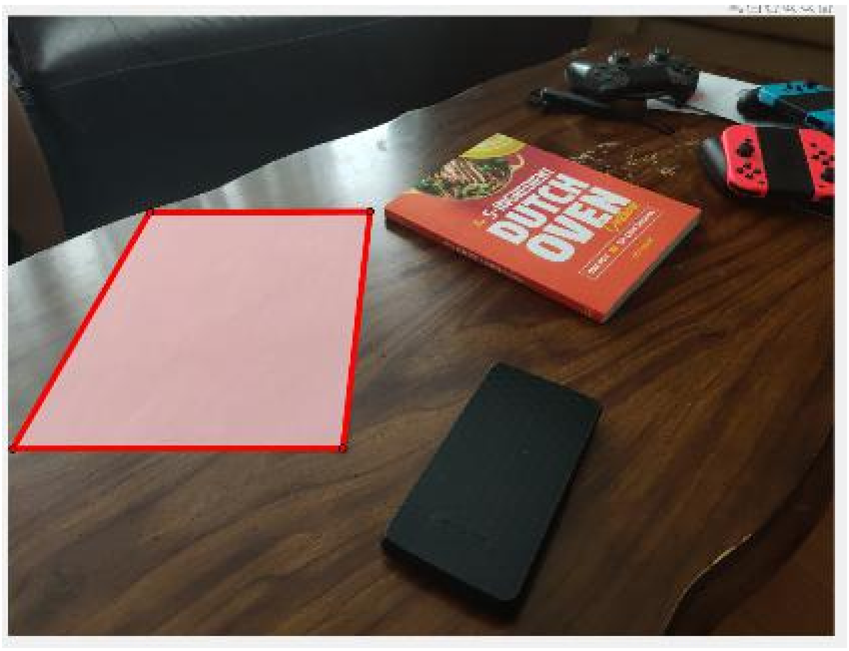


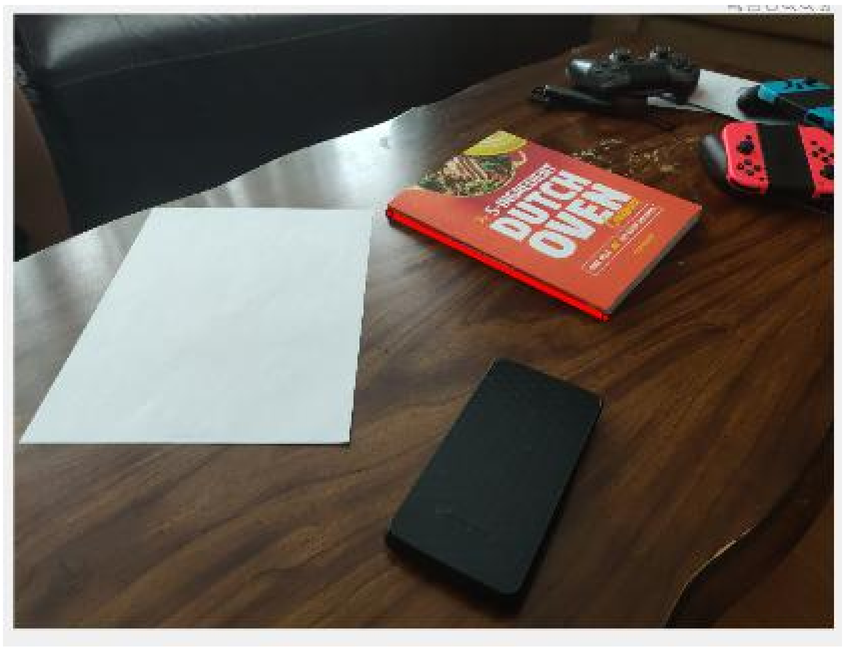


Problem 7

```
clear; close all; clc; format shortG;
%% Parameter

imgBoardFile = 'prob7.JPG';
% imgPicFile = 'stonks.PNG';

% info = imfinfo(imgPicFile);
% sizePic = [info.Width info.Height];
%% Step1: Pick four corners of your white board in (a)

imgBoard = imread(imgBoardFile);
figure(1); imshow(imgBoard);
p = drawpolygon('LineWidth',5,'Color','red');
corner = p.Position;
%% Step2: Compute H (Your Section)

original = [0 0; 216 0; 216 279; 0 279]; %hard coded letter paper dimensions in
mm
A = [];
for ii = 1:size(original,1)
```

```
    row1 = [original(ii,1) original(ii,2) 1 0 0 0 -original(ii,1)*corner(ii,1) -
corner(ii,1)*original(ii,2) -corner(ii,1)];
    row2 = [0 0 0 original(ii,1) original(ii,2) 1 -original(ii,1)*corner(ii,2) -
corner(ii,2)*original(ii,2) -corner(ii,2)];
    A = [A; row1 ; row2];
end
null_space = null(A);
H = [null_space(1:3)'; null_space(4:6)'; null_space(7:9)'];
%% Step3: Measure Distance in Distorted Space

imshow(imgBoard);
p2 = drawpolygon('LineWidth',5,'Color','red');
distance = p2.Position;

H_inverse = inv(H);
points = [];

for ii = 1:size(distance,1)
    col = [distance(ii,:) 1];
    transformed = [H_inverse*col']';
    points = [ points;
transformed(1,1)/transformed(1,3),transformed(1,2)/transformed(1,3)];
end

length = norm(points(1,:)-points(2,:));
fprintf("The length is %.2f mm\n",length)
```





The length is 234.43 mm