Julio Soldevilla
EECS 504 Winter 2018 — Problem Set 1

Problem 1 Problem 1

Proof:

1. For this problm we can begin by just substituting the points into the equation $w\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$. If we do this, we end up having n matrices of the form

$$w\begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix}\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \quad \text{for } i = 1, ..., n \tag{1}$$

which we can rewrite to have the following matrices:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} x_i h_{11} + y_i h_{12} + h_{13} - w x'_i \\ x_i h_{21} + y_i h_{22} + h_{23} - w y'_i \\ x_i h_{31} + y_i h_{32} + 1 - w \end{bmatrix} \quad \text{for } i = 1, ..., n \tag{2}$$

. Recall that in this problem we know the points $\{(x_i, y_i\}$ and $\{(x'_i, y'_i\}$ but don't know the matrix with entries h_{ij}. This is nothing more than a giant system of equations and so we can rewrite more conveniently by considering the matrix $X = \begin{bmatrix} \frac{h_{11}}{w} \\ \frac{h_{12}}{w} \\ \frac{h_{13}}{w} \\ \frac{h_{21}}{w} \\ \frac{h_{22}}{w} \\ \frac{h_{23}}{w} \\ \frac{h_{31}}{w} \\ \frac{h_{32}}{w} \\ \frac{1}{w} \end{bmatrix}$ which is the matrix we want to find, setting the matrix of known output values $Y = \begin{bmatrix} x'_1 \\ y'_1 \\ 1 \\ x'_2 \\ y'_2 \\ 1 \\ \vdots\ x'_n \\ y'_n \\ 1 \end{bmatrix}$ and finally setting the matrix

$$A = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_n & y_n & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_n & y_n & 1 \end{bmatrix}$$

. THen, with this equations, we can reformulate the affine transformation estimation to minimize the standard least squares problem

$$\min \|Ax - y\|_2^2$$

with the matrices A, x and y given above.

However, it seems in the formulation above we could still do some simplification, since we don't really care about the value of w and we have a way of getting rid of it. Particularly, we can take equation 2 above and take the last entry. This entry tells us that $x_i h_{31} + y_i h_{32} + 1 - w = 0 \implies w = x_i h_{31} + y_i h_{32} + 1$ for $i = 1, ..., n$. Substituting thisv alue of w into th erows above, we find that we have the relationships $x_i h_{11} + y_i h_{12} + h_{13} - x'_i x_i h_{31} - x'_i y_i h_{32} = -x'_i$ and $x_i h_{21} + y_i h_{22} + h_{23} - y'_i x_i h_{31} - y'_i y_i h_{32} = -y'_i$. This equations capture the same information as the formulation above but is just presented differently because we represented differently the relations represented the value w. Thus, we can rewrite this equation in matrix form again by considering the following matrices $X' = \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix}$ which is the matrix of unkowns that we want to find to find the homography transformation matrix, the matrix $Y' = \begin{bmatrix} -x'_1 \\ -y'_1 \\ -x'_2 \\ -y'_2 \\ -x'_3 \\ -y'_3 \\ \vdots \\ -x'_n \\ -y'_n \end{bmatrix}$ and finally the matrix $A' = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x'_2x_2 & -x'_2y_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -y'_2x_2 & -y'_2y_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_nx_n & -x'_ny_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_nx_n & -y'_ny_n \end{bmatrix}$. With these matrices, we can again reformulate the problem as the standard least-squares problem and just try to minimize

$$\min \|A'x' - y'\|_2^2$$

with the matrices given in the second reformulation.

2. In any case of the reformulations above, we need to solve the least squares problem $\min \|Ax - y\|_2^2 = \min(Ax-y)^T(Ax-y)$. In this case, we will just the second set of matrices we found above. Recall that in this case the matrix x' is the matrix $\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix}$ and so when taking a partial derivative with respect to x_i, we are taking with respect to the variable of entry x_i in this matrix. To solve this, let $e = (Ax - y)$, then the problem to solve is $\min e^Te$ which we can do by taking the first derivative of this expression, setting it equal to 0 and solving for it. Letting $f(x) = e^Te$ we have that $f'(x) = 2\frac{\partial e}{\partial x_i}e = 0 \overset{1}{\implies} c_i^T(Ax-y) = 0$ for all $i = 1, ..., n$ where c_i are the columns of matrix A and $\frac{\partial e}{\partial x_i} = c_i$ and this derivative comes just from writing out the product $Ax - y$ and then differentiating every row and concatenating the resulting columns. Hence we have that the first derivative implies $c_i^T(Ax-y) = 0$, then concatenating for taking all the derivatives of the rows of the expression $Ax - y$ with respect to all the x_i we have the expression $A^T(Ax-y) = 0 \implies A^TAx - A^Ty = 0 \implies A^TAx = A^Ty \implies x = (A^TA)^{-1}A^Ty$, and so this is the result to the estimation problem from above.

I claim we need 8 points (counting both x_i and y_i) to obtain a unique solution, so we require 4 pairs of points. I think this is the answer because we want to determine the 8 entries h_{ij} in the transformation matrix. TO do this, we need to solve the system of equations with 8 equations given in the second formulation of the problem above. Since with 4 points this matrix becomes a square matrix, we have that the rank of the matrix is 8 with no relationships between the rows or columns, namely it would have full rank and so it would be invertible, allowing us to find a unique solution by just finding the inverse of A. Notice that this if we consider less than 3 pairs of points, then the matrix might not be full rank.

3. In the following we present the code used to solve the problem.

```
%
% W18 EECS 504 HW1p1 Homography Estimation
%

close all;

% Load image
inpath1 = 'football1.jpg';
inpath2 = 'football2.jpg';

im1 = imread(inpath1);
im2 = imread(inpath2);

% Display the yellow line in the first image
figure;
imshow(im1); title('football image 1');
hold on;
u=[1210,1701];v=[126,939];  % marker 33
%u=[942,1294];v=[138,939];
plot(u,v,'y','LineWidth',2);
hold off;

%------------------------------------------------
% FILL BLANK HERE
% Specify the number of pairs of points you need.
n = 4;
%------------------------------------------------

% Get n correspondences
baseName = regexp(inpath1,'^\D+','match','once');
pointsPath = sprintf('%s_points%i.mat',baseName,n);
if exist(pointsPath,'file')
   % Load saved points
   load(pointsPath);
else
   % Get correspondences
   [XY1, XY2] = getCorrespondences(im1,im2,n);
   save(pointsPath,'XY1','XY2');
end

```

```
%--------------------------------------------------
% FILL YOUR CODE HERE
% Your code should estimate the homography and draw the
% corresponding yellow line in the second image.

%We first form matrix A:

A = zeros(3*n,9);

for i = 1:n
    %disp(i)
    k = 1 + 3*(i-1);
    %disp(k)
    for j = 1:9
        %disp(j)
        if mod(j,3) < 3 && mod(j,3) ~= 0
            %disp(XY1(i,mod(j,3)))
            A(k,j) = XY1(i,mod(j,3));
        end
        if mod(j,3) == 0
            A(k,j) = 1;
            k = k+1;
        end
    end
end

% Now, we form the vector Y

Y = zeros(3*n,1);

for i = 1:n
    k = 1 + 3*(i-1);
    %disp(i);
    for j = 1:3
        if mod(j,3) < 3 && mod(j,3) ~= 0
            Y(k,1) = XY2(i,mod(j,3));
            k = k + 1;
        end
        if mod(j,3) == 0
            Y(k,1) = 1;
            k = k+1;
        end
    end
end

%Computing vector X (vector where every entry is one of the entries of
%transformation matrix

Ainv = inv(A'*A);
H = Ainv * A' * Y;

%The last step is to make the entries in H form the transformation matrix
```

```
.

Hmatrix = vec2mat(H,3);

%Now we take the points u and v from the original yellow line and apply the
%transformation H to them.

point1 = [u(1), v(1)];
point2 = [u(2),v(2)];

point1_transform = Hmatrix * [point1,1]';
point2_transform = Hmatrix * [point2,1]';

u_transform = [point1_transform(1),point2_transform(1)];
v_transform =[point1_transform(2),point2_transform(2)];

% Now, we display the yellow line in the second image
figure;
imshow(im2); title('football image 2');
hold on;
%u=[1210,1701];v=[126,939];  % marker 33
%u=[942,1294];v=[138,939];

plot(u_transform,v_transform,'y','LineWidth',2);
hold off;

%---------------------------------------------------------------
```

Now, we show the image that results as output of the code above in figure 1. It is important to notice that the points obtained for the correspondences are $XY1 = \begin{bmatrix} 1134.65 & 517.9 \\ 1134.7 & 589.9 \\ 1614.2 & 561.9 \\ 966.9 & 314.2 \end{bmatrix}$ and $XY2 = \begin{bmatrix} 615.2638 & 613.8468 \\ 415.4766 & 689.7660 \\ 863.000 & 629.8298 \\ 231.6723 & 426.0468 \end{bmatrix}$ and with these vectors we get the transformation matrix $H = \begin{bmatrix} 0.9147 & 0.0949 & -707.4416 \\ -0.0633 & 0.9968 & 175.5697 \\ -0 & -0 & 1 \end{bmatrix}$:

To finish, we also present the original image with the 33 yard line in figure 2:

■

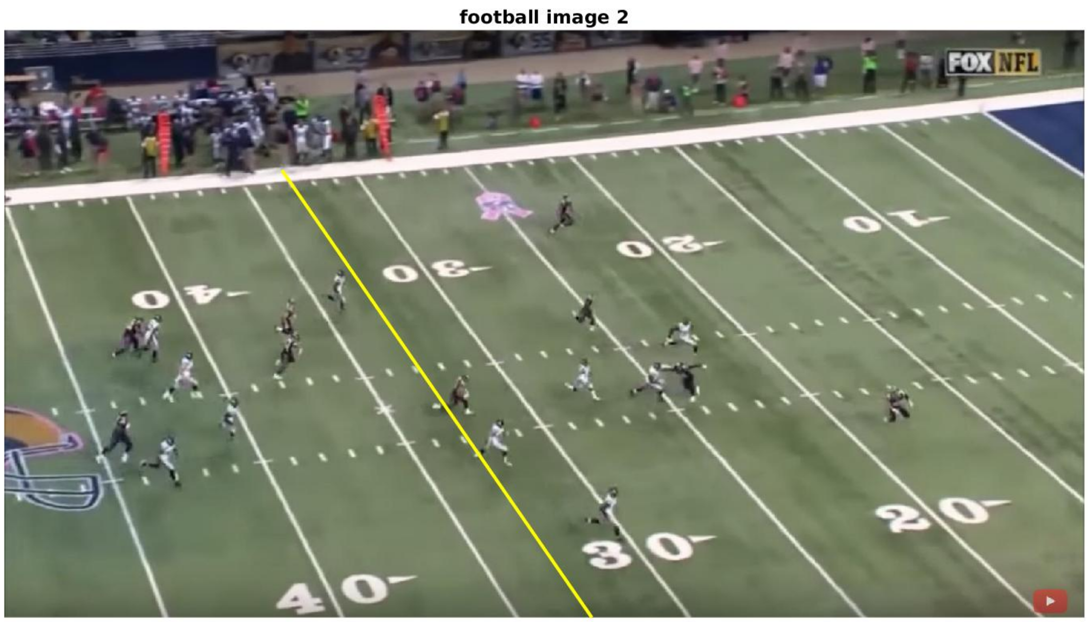

Figure 1: This is the image that we get as output after running the code above using the points XY1 and XY2

Problem 2 *Problem 2*

Proof: The issue with the given model on vertical lines is that in this case we are trying to compute the vertical distance of the points to the line, but that is impossible in this case since the line is vertical and there is no vertical distance between the data points and the line we compute (additionally, the slope of such line is $m = \infty$ and so we cannot do the substraction required in the least squares problem). So, to avoid this issue (and any issue with horizontal lines too), first we can reparameterize the line as $Ax + By + C = 0$. With this reformulation, we can also reformulate the least squares problem to be $E(A, B, C) = \sum_{i=1}^{n}(Ax_i + By_i + C)^2$, so the least squares problem would be

$$\begin{aligned} \underset{A,B,C}{\text{minimize}} \quad & \sum_{i=1}^{n}(Ax_i + By_i + C)^2 \\ \text{subject to} \quad & A^2 + B^2 = 1 \end{aligned}$$

where we have the last constraint to ensure that we are not considering repeated lines, or lines that are the same but the coefficients are just multiplied by some constant, i.e. lines with coefficients A, B, C and then lines with coefficients $2A, 2B, 2C$. Furthermore, we can interpret the equation $Ax_i + By_i + C = d_i$ as the distance between the lime we want estimate and the points we have, then the minimization problem is to minimize the sum of this distance with respect to all the

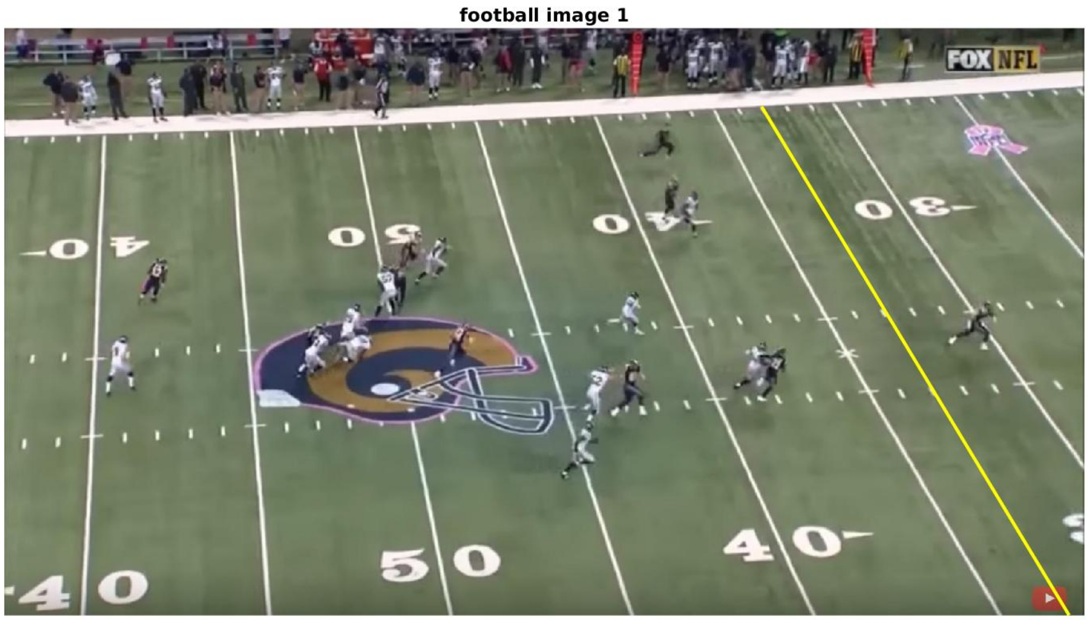

Figure 2: Original image with 33 yard line, no transformation.

points in the data set. Notice that this reformulation of the least squares problem solves the issue with vertical and horizontal lines, since when we have vertical lines, we just set the coefficient of Y to be 0 and we will be able to minimize the horizontal distances between the points and the line we are estimating, namely minimize the expression $\sum_{i=1}^{n}(Ax_i + C)^2$ and when we have horizontal lines, the coefficient of X will just turn out to be 0, and we will be able to minimize the horizontal distances between the points and the line we are estimating, namely minimize the expression $\sum_{i=1}^{n}(By_i + C)^2$. Thus this reformulation solves the problem with the least squares estimation of horizontal and vertical lines. Now, we can also do a matrix formulation of the minimization problem by rewriting the line equation as $Ax + By + C = 0 \iff [A \quad B \quad C]\begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ 1 & 1 & \dots & 1 \end{bmatrix} = 0$. THen, the minimization problem would be $\min_{A,B,C}\left\|[A \quad B \quad C]\begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ 1 & 1 & \dots & 1 \end{bmatrix}\right\|_2^2 = \min_{A,B,C}\left([A \quad B \quad C]\begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ 1 & 1 & \dots & 1 \end{bmatrix}\right)^T\left([A \quad B \quad C]\begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ 1 & 1 & \dots & 1 \end{bmatrix}\right)$. Now, in this case, if we try to find the solution of this problem by finding the first derivative and setting this equation equal to 0, letting $e = [A \quad B \quad C]\begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ 1 & 1 & \dots & 1 \end{bmatrix}$ we end up having that such derivative $f'(x) = 2\frac{\partial e}{\partial x_i}e = 2A[A \quad B \quad C]\begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ 1 & 1 & \dots & 1 \end{bmatrix} = 0$. However, if we try to solve this (therefore getting the pseudo inverse expression) we just end up having that $\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = 0$, which is not a solutio to this problem. This issue when obtaining the solution to the least squares via the first derivative implies that we **cannot** use the pseudo-inverse to solve the minimization problem. ■

Problem 3 *Problem 3*

Proof:

1. I claim that both A and B will observe the reflection at point x with the same strength. First of all notice that the lambertian model of reflectance $R(x) = \rho(x)^T n(x)$ is a formula that contains information about the surface normal with $n(x)$, about the direction of incident light source $\mathcal{L}(x)$, hence it contains no information of the strength that someone will observe the reflection and thus we can concldue that the radiance emitted from a Lambertian surface is not a function of outgoing direction, hence since the formula for the Lambertian surface has no information about the strength of the reflection, it must be that both A and B perceive the strength of the reflection at x equally.

2. One major drawback of Lambertian surface model is that it only captures diffuse characteristics of the surface, but not ambient lighting, specular reflections or other more complex radiometric functions. One object in the real world whose reflectance is not welll modeled by the Lambertian model is a mirror or glass.

■

Problem 4 *Problem 4*

Proof: Here we present the code of the function mydemosaic.m

```
function I = mydemosaic(I_gray)

%
% W18 EECS 504 HW1p4 Bayer Demosaicking
%
% function mydemosaic recovers the original color image (M*N*3)
% from the Bayer encoded image I_gray (M*N).
%

%-----------------------------------
```

```
% IMPLEMENT THE FUNCTION HERE

% Generating Bayern pattern filter

%R = [0.25, 0.5, 0.25 : 0.5, 1, 0.5: 0.25, 0.5, 0.25 ];
%B = [ 0.25, 0.5, 0.5; 0.5,; ];
%G = [ ; ; ];

% Generating the Bayer encoded image.
%[I,I_grey] = bayer_filter(im1);
double_I_gray = im2double(I_gray);
[M,N] = size(double_I_gray);
T = zeros(M,N,3);
%T(:,:,1) Red layer
%T(:,:,2) Green layer
%T(:,:,3) Blue layer

for i = 2:M-1
    for j = 2:N-1
        if mod(i,2) == 0  && mod(j,2) == 0 % Green pixels on even row
            T(i,j,1) = (double_I_gray(i,j-1) + double_I_gray(i,j+1))/2; %Red part
            T(i,j,2) = (double_I_gray(i,j)); % Green part
            T(i,j,3) = (double_I_gray(i+1,j) + double_I_gray(i-1,j))/2; % Blue Part
        elseif mod(i,2) == 1 && mod(j,2) == 1 %Green layer on odd row
            T(i,j,1) = (double_I_gray(i+1,j) + double_I_gray(i-1,j))/2; %Red part
            T(i,j,2) = double_I_gray(i,j); %Green part
            T(i,j,3) = (double_I_gray(i,j-1) + double_I_gray(i,j+1))/2; %Blue part
        elseif mod(i,2) == 1 && mod(j,2) == 0 % First selecting the blue pixels
            T(i,j,1) = (double_I_gray(i-1,j+1) + double_I_gray(i-1,j-1) + double_I_gray(i+1,j-1) + double_I_gray(i+1,j+1))/4; %Red layer
            T(i,j,2) = (double_I_gray(i-1,j) + double_I_gray(i+1,j) + double_I_gray(i,j-1) + double_I_gray(i,j+1))/4; %Green Layer
            T(i,j,3) = double_I_gray(i,j);%Blue layer
        elseif mod(i,2) == 0 &&  mod(j,2)== 1 % Select red pixels
            T(i,j,1) = double_I_gray(i,j); %Red Layer
            T(i,j,2) = (double_I_gray(i,j-1) + double_I_gray(i,j+1) + double_I_gray(i-1,j) + double_I_gray(i+1,j))/4; %Green Layer
            T(i,j,3) =  (double_I_gray(i-1,j-1) + double_I_gray(i-1,j+1)+ double_I_gray(i+1,j-1) + double_I_gray(i+1,j+1))/4; %Blue part
        end
    end
end

% Now we correct for the borders. We divided the borders into 4 corners and
% 4 edges.

%-------------------------------------------------------------------
```

```
% Checking corners:

% UL corner:
for i = [1,M]
    for j = [1,N]
        if i == 1 && j == 1 %UL corner (green corner)
            T(i,j,1) = double_I_gray(i+1,j); % Red part
            T(i,j,2) = double_I_gray(i,j); % Green part
            T(i,j,3) = double_I_gray(i,j+1); % Blue part

            %Two possibilities for UR corner
            %1) for even N (blue corner)
        elseif i == 1 && j == N && mod(N,2) == 0
            T(i,j,1) = double_I_gray(i+1,j-1); % Red part
            T(i,j,2) = (double_I_gray(i,j-1) + double_I_gray(i+1,j))/2; % Green part
            T(i,j,3) = double_I_gray(i,j); % Blue part
            %2) for odd N (green corner)
        elseif i == 1 && j == N && mod(N,2) == 1
            T(i,j,1) = double_I_gray(i+1,j); % Red part
            T(i,j,2) = double_I_gray(i,j); % Green part
            T(i,j,3) = double_I_gray(i,j-1); % Blue part

            %DL corner, two possibilities:
            %1) for odd M (green corner)
        elseif j == 1 && i == M && mod(M,2) == 1
            T(i,j,1) = double_I_gray(i-1,j); % Red part
            T(i,j,2) = double_I_gray(i,j); % Green part
            T(i,j,3) = double_I_gray(i,j+1); % Blue part
            %2) for even M (red corner)
        elseif j == 1 && i == M && mod(M,2) == 0
            T(i,j,1) = double_I_gray(i,j); % Red part
            T(i,j,2) = (double_I_gray(i-1,j) + double_I_gray(i,j+1))/2; % Green part
            T(i,j,3) = double_I_gray(i-1,j+1); % Blue part

            %DR corner, four possibilities:
            %M odd and N odd (green corner)
        elseif j == N && i == M && mod(N,2) == 1 && mod(M,2) == 1
            T(i,j,1) = double_I_gray(i-1,j); % Red part
            T(i,j,2) = double_I_gray(i,j); % Green part
            T(i,j,3) = double_I_gray(i,j-1); % Blue part
            %M even and N even (green corner)
        elseif j == N && i == M && mod(N,2) == 0 && mod(M,2) == 0
            T(i,j,1) = double_I_gray(i,j-1); % Red part
            T(i,j,2) = double_I_gray(i,j); % Green part
            T(i,j,3) = double_I_gray(i-1,j); % Blue part
            % M even and N odd (red corner)
        elseif j == N && i == M && mod(N,2) == 1 && mod(M,2) == 0
            T(i,j,1) = double_I_gray(i,j); % Red part
```

```
            T(i,j,2) = (double_I_gray(i-1,j) + double_I_gray(i,j-1))/2; % Green part
            T(i,j,3) = double_I_gray(i-1,j-1); % Blue part
            % M odd and N even (blue corner)
        elseif j == N && i == M && mod(N,2) == 0 && mod(M,2) == 1
            T(i,j,1) = double_I_gray(i-1,j-1); % Red part
            T(i,j,2) = (double_I_gray(i-1,j) + double_I_gray(i,j-1))/2; % Green part
            T(i,j,3) = double_I_gray(i,j); % Blue part
        end
    end
end

%----------------------------------------

%Edges of the picture

%Upper and lower edge

%Upper edge
for i = [1,M]
    for j = 2:N-1
        if mod(j,2) == 0 && i == 1 % Blue upper pixels
            T(i,j,1) = (double_I_gray(i+1,j-1) + double_I_gray(i+1,j+1))/2; % Red part
            T(i,j,2) = (double_I_gray(i,j-1) + double_I_gray(i+1,j) + double_I_gray(i,j+1))/3; % Green part
            T(i,j,3) = double_I_gray(i,j); % Blue part
        elseif mod(j,2) == 1 && i == 1% Green upper pixels
            T(i,j,1) = (double_I_gray(i+1,j)); % Red part
            T(i,j,2) = double_I_gray(i,j); % Green part
            T(i,j,3) = (double_I_gray(i,j-1) + double_I_gray(i,j+1))/2; % Blue part
        elseif i == M && mod(j,2) == 0 && mod(M,2) == 1 % Blue lower pixels and M odd
            T(i,j,1) = (double_I_gray(i-1,j-1) + double_I_gray(i-1,j+1))/2; % Red part
            T(i,j,2) = (double_I_gray(i-1,j) + double_I_gray(i,j-1) + double_I_gray(i,j+1))/3; % Green part
            T(i,j,3) = double_I_gray(i,j); % Blue part
        elseif i == M && mod(j,2) == 1 && mod(M,2) == 1 % Green lower pixels and M odd
            T(i,j,1) = (double_I_gray(i-1,j)); % Red part
            T(i,j,2) = double_I_gray(i,j); % Green part
            T(i,j,3) = (double_I_gray(i,j+1) + double_I_gray(i,j-1))/2; % Blue part
        elseif i == M && mod(j,2) == 0 && mod(M,2) == 0 %Lower part green pixels and even M
            T(i,j,1) = (double_I_gray(i,j-1) + double_I_gray(i,j+1))/2; % Red part
            T(i,j,2) = double_I_gray(i,j); % Green part
            T(i,j,3) = (double_I_gray(i-1,j)); % Blue part
```

```matlab
        elseif i == M && mod(j,2) == 1 && mod(M,2) == 0 % Lower part red pixels and even M
            T(i,j,1) = double_I_gray(i,j); % Red part
            T(i,j,2) = (double_I_gray(i,j-1) + double_I_gray(i,j+1) + double_I_gray(i-1,j))/3; % Green part
            T(i,j,3) = (double_I_gray(i-1,j-1) + double_I_gray(i-1,j+1))/2; % Blue part
        end
    end
end

%Left and right edge

for j = [1,N]
    for i = 2:M-1
        if j == 1 && mod(i,2) == 0 %Left edge red pixel
            T(i,j,1) = double_I_gray(i,j); % Red part
            T(i,j,2) = (double_I_gray(i-1,j) + double_I_gray(i+1,j) + double_I_gray(i,j+1))/3; % Green part
            T(i,j,3) = (double_I_gray(i-1,j+1) + double_I_gray(i+1,j+1))/2; % Blue part
        elseif j == 1 && mod(i,2) == 1 %Left edge green pixel
            T(i,j,1) = (double_I_gray(i+1,j) + double_I_gray(i-1,j))/2; % Red part
            T(i,j,2) = double_I_gray(i,j); % Green part
            T(i,j,3) = double_I_gray(i,j+1); % Blue part
        elseif j == N && mod(i,2) == 0 && mod(N,2) == 1 %Right edge, red pixel with N odd
            T(i,j,1) = double_I_gray(i,j); % Red part
            T(i,j,2) = (double_I_gray(i-1,j) + double_I_gray(i+1,j) + doubl_I_gray(i,j-1))/3; % Green part
            T(i,j,3) = (double_I_gray(i-1,j-1) + double_I_gray(i+1,j-1))/2; % Blue part
        elseif j == N && mod(i,2) == 1 && mod(N,2) == 1 % Right edge, green pixel with N odd
            T(i,j,1) = (double_I_gray(i+1,j) + double_I_gray(i-1,j))/2; % Red part
            T(i,j,2) = double_I_gray(i,j); % Green part
            T(i,j,3) = double_I_gray(i,j-1); % Blue part
        elseif j == N && mod(i,2) == 0 && mod(N,2) == 0 %Right edge, green pixel with N even
            T(i,j,1) = double_I_gray(i,j-1); % Red part
            T(i,j,2) = double_I_gray(i,j); % Green part
            T(i,j,3) = (double_I_gray(i-1,j) + double_I_gray(i+1,j))/2; % Blue part
        elseif j == N && mod(i,2) == 1 && mod(N,2) == 0 %Right edge, blue pixel with N even
            T(i,j,1) = (double_I_gray(i-1,j-1) + double_I_gray(i+1,j-1))/2; % Red part
            T(i,j,2) = (double_I_gray(i-1,j) + double_I_gray(i+1,j) + double_I_gray(i,j-1))/3; % Green part
            T(i,j,3) = double_I_gray(i,j); % Blue part
```

```
            end
        end
end

T;
figure; imshow(T);
end
```

The original image we are considering for this problem is the following one, in figure 3:

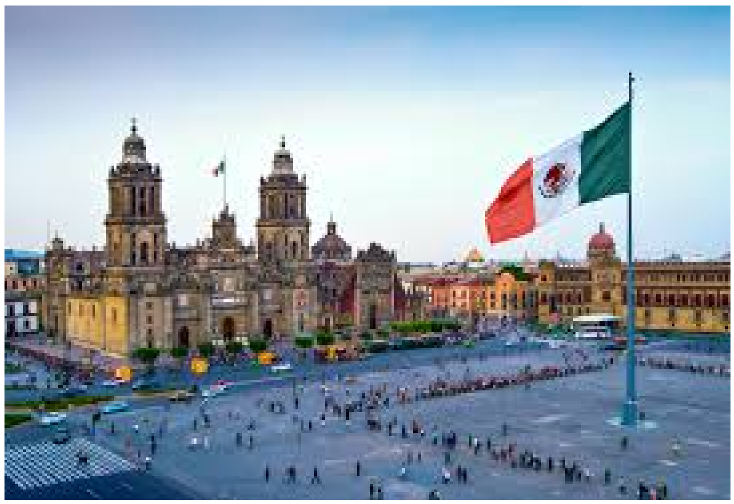

Figure 3: Original image considered for this problem.

Then, the gray-scaled image obtained from the Bayer filter function is seen in figure 4 and in figure 5 we see the other result from the bayer filter function.

Finally, this is the image we get as output from the mydemosaic.m function we implemented, we see such image in figure 6.

■

Figure 4: Gray-scaled image obtained from Bayer filter function.

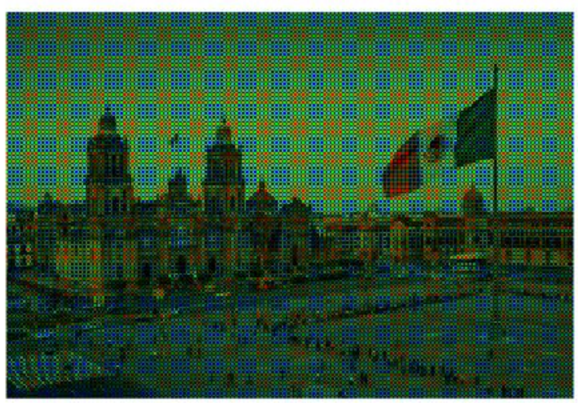

Figure 5: Image obtained from bayer filter (non-gray scale).

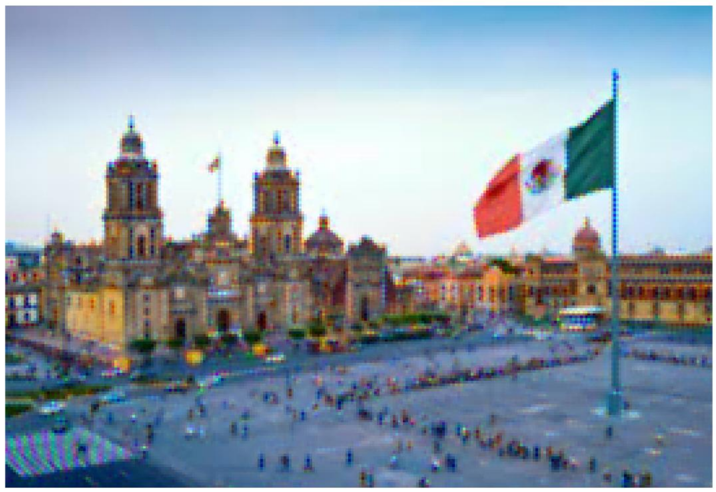

Figure 6: Output image from function mydemosaic.