Assignment 1 Solutions

CS283, Computer Vision

1. (a) Using the MATLAB function `size`, we see that the original image has a height of 512 and a width of 512 pixels.

   (b) In Figure 1, we see the four images, generated using `subplot` to display them and `print` to print the overall figure to an `.eps` file.

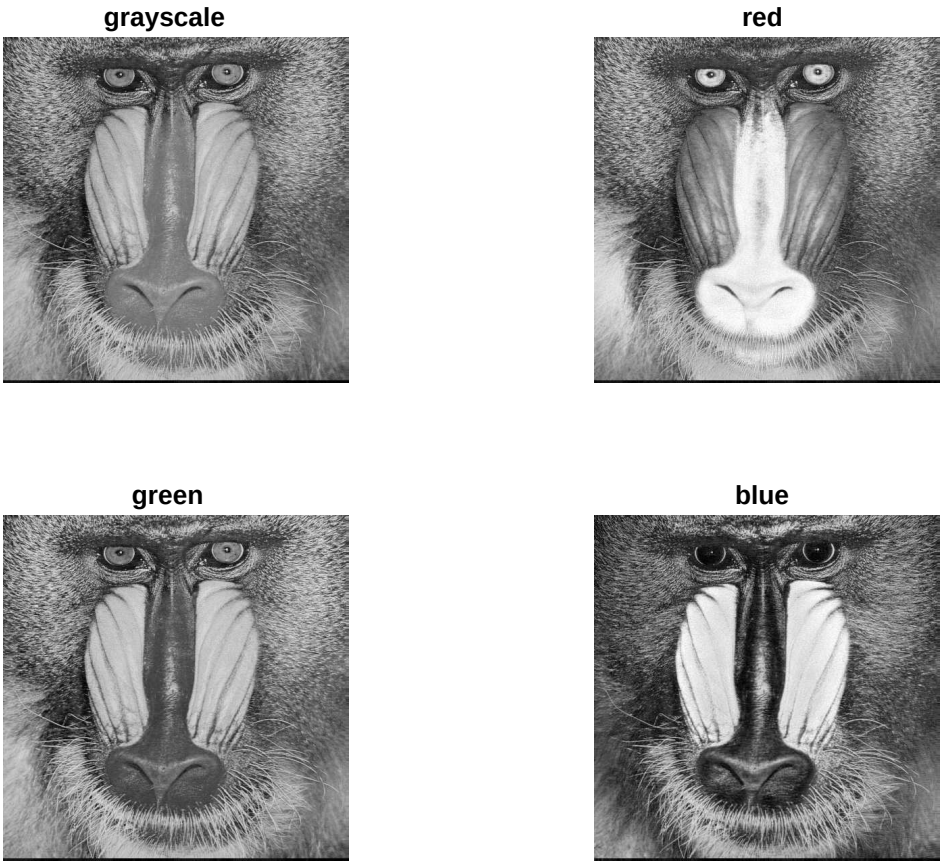

Figure 1: From top to bottom and from left to right: grayscale image; red channel; green channel; blue channel.

   (c) In Figure 2, we show the original image and its JPEG compressed version, produced using `imwrite` with a quality setting of 95%. The two images look very similar. To convince you that they are not identical, in Figure 2 we also show the absolute value of the relative difference of the two images.

   The compression ratio with respect to the original `.tif` file is

$$\frac{793580\,\text{bytes}}{190017\,\text{bytes}} = 4.18. \tag{1}$$

   (d) Deciding whether two images are indistinguishable is a very subjective task. Using decrements of 5%, we may say that the compressed image is practically indistinguishable from the original for

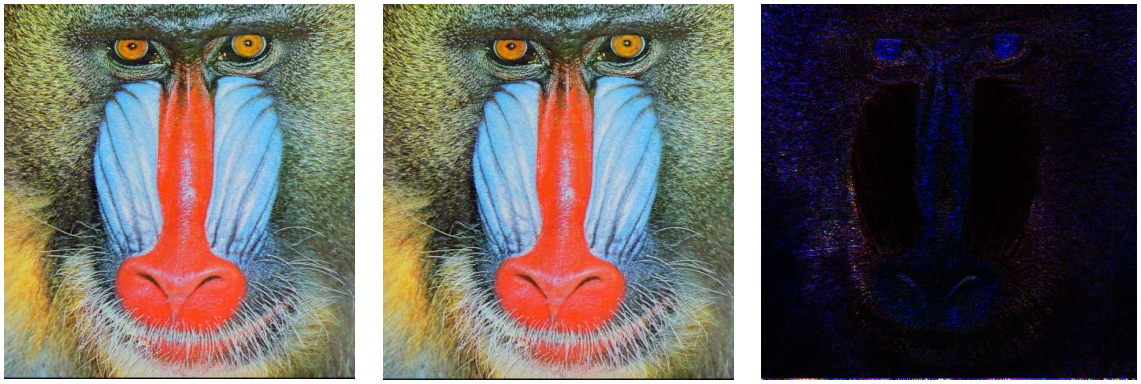

Figure 2: From left to right: original image; JPEG compressed version with quality setting 95%; relative absolute difference between the two.

a quality setting as low as 45%. The compression ratio in this case is

$$\frac{798292\,\text{bytes}}{47444\,\text{bytes}} = 16.73. \tag{2}$$

2. (a) In Figure 3, we can see the cameraman image with the intensities of a random 25% subset of its pixels replaced by random integers.

Figure 3: Baboon image with noise.

Below you can see one possible version of a script that solves this question.

```
% CS283 Fall 2015
% Student: Ioannis Gkioulekas
% Homework assignment 1, question 2
%
% Read image and replace intensities of randomly selected 25 percent of the
```

```
% pixels with random values drawn uniformly from 0–255.

% read image
im = imread('../data/cameraman.tif');

% get number of pixels
N = numel(im);

% randomly select exactly 25% of the pixels
percentage = .25;
numReplace = ceil(percentage * N);
indsReplace = randperm(N, numReplace);

% create random values to replace intensities with
valuesReplace = floor(256 * rand(numReplace, 1));

% replace values
im(indsReplace) = uint8(valuesReplace);

% show result
figure; imshow(im);
```

The random selection of 25% of the image pixels is done by creating a random permutation of the indices corresponding the pixels, and then selecting the first 25% of them. This method achieves both the objectives of the question description: that exactly 25% of the pixels are selected; and that all the pixels have equal probability of being selected.

Note that this is different from uniformly sampling $512 \cdot 512 \cdot 0.25$ numbers between $[1, 512 \cdot 512]$, as then it is possible to have some pixels selected more than once.

(b) As before, the answer to this question is subjective, and it is very hard to define a threshold that would hold for everyone. At 25%, the picture is still recognizable. As we increase the percentage of pixels that have their intensities replaced by random values, we observe that there is a gradual degradation of the quality of the image.

3. (a) In order to write the equations in a matrix form, we have

$$\left.\begin{array}{c} ax_1 + b = y_1 \\ ax_2 + b = y_2 \\ \vdots \\ ax_N + b = y_N \end{array}\right\} \Rightarrow \underbrace{\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_N & 1 \end{bmatrix}}_{\mathbf{A}} \cdot \underbrace{\begin{bmatrix} a \\ b \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\mathbf{b}}. \tag{3}$$

When $\mathbf{A}^T\mathbf{A}$ is invertible, a condition that is always satisfied when we have three or more non-collinera points, the solution to the above optimization problem is given by

$$\mathbf{x} = \mathbf{A}^{\dagger}\mathbf{b}, \tag{4}$$

where $\mathbf{A}^{\dagger}$ is the *pseudoinverse* of the matrix $\mathbf{A}$,

$$\mathbf{A}^{\dagger} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T. \tag{5}$$

The pseudoinverse is used *only for notational purposes*, in order to express the solution in closed-form. As the above problem is equivalent to solving an over-determined linear system of equations, in practice it is preferred to find the solution using factorization methods, such as QR factorization. Such methods offer significant advantages both in terms of computational efficiency and numerical stability.

The above solution minimizes the objective

$$L\left(a, b\right) = \sum_{i=1}^{N} \left(y_i - ax_i - b\right)^2, \tag{6}$$

where N is the number of points $p_i = (x_i, y_i)$ we have. Geometrically, the above solution minimizes sum of the square of the *vertical* distance of each point from the straight line, or equivalently the sum of $(y_i - f(x_i))^2$, where by $f(\cdot)$ we mean the straight line. To make things clearer, we refer to Figure 4. The above solution is the one that minimizes the sum of the square of the lengths of the green line segments drawn on the figure.

Whether fitting a line using this optimization criterion is appropriate depends on the application. In a setting where we want to perform linear regression (prediction) under the assumption of Gaussian noise *only at the vertical coordinate*, the above procedure gives an optimal solution under various notions of optimality, for example maximum likelihood. On the other hand, from a geometric perspective, the above procedure emphasizes fitting only the vertical distances, which is equivalent to assuming that the horizontal coordinates are known exactly. This assumption is not reasonable for the case of fitting a line to arbitrary points on the plane.

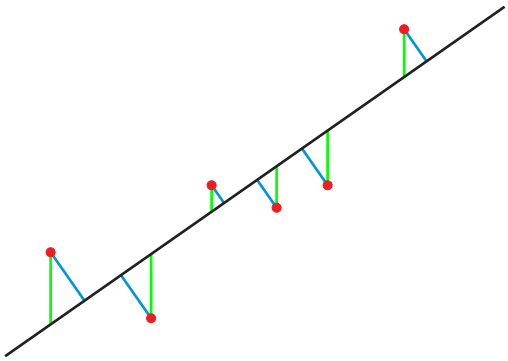

Figure 4: Geometrical interpretation of optimization problems solved in parts (a) (green lines) and (b) (blue lines).

(b) Using homogeneous coordinates, we can write the problem as

$$\underbrace{\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \\ x_N & y_N & 1 \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} a \\ b \\ c \end{bmatrix}}_{\mathbf{l}} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{\mathbf{0}}. \tag{7}$$

The vector $[a, b, c]^T$ characterizing the line we fit is defined only up to scale. Therefore, for the above problem to have a unique solution, we need to enforce some constraint on the solution space. The objective minimized by this approach depends on the constraint we select. To decide what constraint to enforce, it is worth remembering that the perpendicular distance between a point $[x, y]^T$ and a line $[a, b, c]^T$ is equal to

$$\frac{|ax + by + c|}{\sqrt{a^2 + b^2}}. \tag{8}$$

Therefore, if we use the constraint

$$a^2 + b^2 = 1, \tag{9}$$

then the objective we minize is the the sum of the squares of perpendicular distances of all sample points from the fitted line. Referring again to Figure 4, this problem minimizes the sum of the squares of the blue line segments. An alternative approach is to use instead the constraint

$$a^2 + b^2 + c^2 = 1, \tag{10}$$

that is, that the vector characterizing the line is of unit norm. While with this second approach we are no longer minimizing exactly the sum of squares of perpendicular distances, it is still a better one than the approach in part (a), as it takes into account both vertical and horizontal distances. Furthermore, as we discussed in class, this second problem can be solved easily using the singular value decomposition of $\mathbf{A}$. Note however that this does not mean that this problem can be solved easier or more efficiently than the problem in part (a): As discussed there, that problem can be solved using appropriate factorization algorithms that can be even more efficient than the SVD used here. In fact, the problem of part (a) can also be solved using the SVD of $\mathbf{A}$, though we will not show this here. We adopt the second constraint for this problem, but see part (d) for related discussion.

(c) A possible script for solving both parts (c) and (d) is shown below.

(d) Using duality, we can formulate a homogeneous linear system similar to part (b),

$$\underbrace{\begin{bmatrix}\mathbf{l}_R^T\\ \mathbf{l}_G^T\\ \mathbf{l}_B^T\end{bmatrix}}_{\mathbf{A}}\underbrace{\begin{bmatrix}x\\ y\\ z\end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix}0\\ 0\\ 0\end{bmatrix}}_{\mathbf{0}}, \tag{11}$$

where $\mathbf{l}_R$, $\mathbf{l}_G$, and $\mathbf{l}_B$ are the (3×1) homogeneous vectors representing the three lines fitted in part (b). This solution returns a homogeneous point defined up to scale: to obtain the corresponding inhomogeneous solution, we need to divide by z.

Note that, as the homogeneous vectors for the lines are defined only up to scale, the least-squares solution of the above problem is different depending on how we scale the vectors. If we use exactly the vectors found in part (c), then the choice of scale we make is that the three vectors are unit vectors. A more reasonable choice in this case would be to normalize each vector $\mathbf{l}$ representing a line such that

$$l_1^2 + l_2^2 = 1. \tag{12}$$

Then, for each point on the two-dimensional place represented by a homogeneous vector $\mathbf{x}$ normalized such that its third coordinate is equal to 1, the inner product $\mathbf{l}^T\mathbf{x}$ is exactly equal to the perpendicular distance of the point from the line. Figure 5 shows the resulting lines and points, using both of the scaling approaches we discussed above.

As in part (b), from a geometric point of view, the solution that would make the most sense would be the point for which the sum of squared distances from the three lines is minimized. Using the discussion above, we can do this by fixing the third coordinate of the point we are searching for to 1, and reformulating our problem as a heterogeneous system,

$$\begin{bmatrix}l_{R,1} & l_{R,2} & l_{R,3}\\ l_{G,1} & l_{G,2} & l_{G,3}\\ l_{B,1} & l_{B,2} & l_{B,3}\end{bmatrix}\begin{bmatrix}x\\ y\\ 1\end{bmatrix} = \begin{bmatrix}0\\ 0\\ 0\end{bmatrix} \Rightarrow \underbrace{\begin{bmatrix}l_{R,1} & l_{R,2}\\ l_{G,1} & l_{G,2}\\ l_{B,1} & l_{B,2}\end{bmatrix}}_{\mathbf{A}}\underbrace{\begin{bmatrix}x\\ y\end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix}-l_{R,3}\\ -l_{G,3}\\ -l_{B,3}\end{bmatrix}}_{\mathbf{b}}. \tag{13}$$

The least-squares solution to the new system $\mathbf{Ax} = \mathbf{b}$, which can be obtained as discussed in part (a), gives the desired 2D point.

```
% CS283 Fall 2015
% Student: Ioannis Gkioulekas
% Homework assignment 1, question 3
%
% Fit lines to points.

%% Part c

% read "dots" file
im = imread('../data/dots.tif');
dims = size(im);
```

```
12
13 % read coordinates of dots
14 % red
15 indsRed = find(im(:, :, 2) == 0 & im(:, :, 3) == 0);
16 [yRed xRed] = ind2sub(dims(1:2), indsRed);
17
18 % green
19 indsGreen = find(im(:, :, 1) == 0 & im(:, :, 3) == 0);
20 [yGreen xGreen] = ind2sub(dims(1:2), indsGreen);
21
22 % blue
23 indsBlue = find(im(:, :, 1) == 0 & im(:, :, 2) == 0);
24 [yBlue xBlue] = ind2sub(dims(1:2), indsBlue);
25
26 % construct A matrices for each color channel and find least-squares
27 % solutions of the corresponding homogeneous linear systems
28 % red
29 numRed = length(xRed);
30 ARed = [xRed, yRed, ones(numRed, 1)];
31 [¬, ¬, VRed] = svd(ARed);
32 lineRed = VRed(:, 3);
33
34 % green
35 numGreen = length(xGreen);
36 AGreen = [xGreen, yGreen, ones(numGreen, 1)];
37 [¬, ¬, VGreen] = svd(AGreen);
38 lineGreen = VGreen(:, 3);
39
40 % blue
41 numBlue = length(xBlue);
42 ABlue = [xBlue, yBlue, ones(numBlue, 1)];
43 [¬, ¬, VBlue] = svd(ABlue);
44 lineBlue = VBlue(:, 3);
45
46 % create extreme points of calculated lines
47
48 % x coordinates of extreme points
49 xCoords = [1 dims(2)];
50
51 % red
52 yRedCoords = - (lineRed(1) * xCoords + lineRed(3)) / lineRed(2);
53
54 % green
55 yGreenCoords = - (lineGreen(1) * xCoords + lineGreen(3)) / lineGreen(2);
56
57 % blue
58 yBlueCoords = - (lineBlue(1) * xCoords + lineBlue(3)) / lineBlue(2);
59
60 % plot resuts on top of "dots"
61 figure; imshow(im); hold on;
62 plot(xCoords, yRedCoords, 'r-', 'LineWidth', 2);
63 plot(xCoords, yGreenCoords, 'g-', 'LineWidth', 2);
64 plot(xCoords, yBlueCoords, 'b-', 'LineWidth', 2);
65
66 %% Part d
67
68 % stack parameters of three lines together
69 APoint = [lineRed'; lineGreen'; lineBlue'];
70 [¬, ¬, VPoint] = svd(APoint);
71 point = VPoint(:, 3);
72 plot(point(1) / point(3), point(2) / point(3), 'm.', 'MarkerSize', 20)
73
74 % alternative: normalize x and y multipliers in lines, to minimize exactly
75 % perpendicular distances
76 % NOTE: use hypot when doing operations of the form sqrt(x ^ 2 + y ^ 2),
77 % for numerical robustness:
78 % http://www.mathworks.com/help/matlab/ref/hypot.html
79 APoint = APoint ./ repmat(hypot(APoint(:, 1), APoint(:, 2)), [1 3]);
```

```
[¬, ¬, VPoint] = svd(APoint);
point = VPoint(:, 3);
plot(point(1) / point(3), point(2) / point(3), 'k.', 'MarkerSize', 20)
```

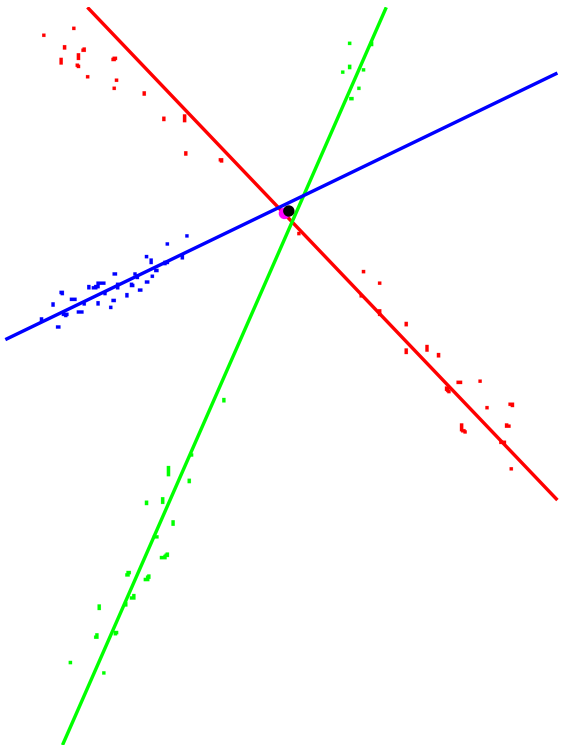

Figure 5: Figure depicting the three fitted lines and the points closest to their intersection (the magenta point corresponds to scaling vectors representing lines to have unit norm, and the black point to the alternative scaling discussed in (d)).

4. A possible script for solving both parts (a) and (b) of this question is shown below. Note that in part (b) we select two points in each iteration, as that is the minimum amount of points that can uniquely determine a line.

```
% CS283 Fall 2015
% Student: Ioannis Gkioulekas
% Homework assignment 1, question 4
%
% Fit line to points with outliers.

%% Part a
% Fit to all points.

% read "dots_outliers" file
im = imread('../data/dots_outliers.tif');
dims = size(im);

% read coordinates of dots:
inds = find(im);
[y x] = ind2sub(dims(1:2), inds);

% construct A and y matrices for each color channel and find least-squares
% solution of corresponding homogeneous linear system:

```

```
% red
numDots = length(x);
A = [x, y, ones(numDots, 1)];
[¬, ¬, V] = svd(A);
lineAll = V(:, 3);

% create extreme points of calculated lines
xCoords = [1 dims(2)];
yCoords = — (lineAll(1) * xCoords + lineAll(3)) / lineAll(2);

% plot resuts on top of "dots_outliers"
figure; imshow(im); hold on;
plot(xCoords, yCoords, 'r—', 'LineWidth', 2);

%% Part b
% Use RANSAC to prune outliers.

% RANSAC parameters
threshold = 20;
numIters = 100;

N = length(x);
bestConsensusSet = [];
sizeBestConsensusSet = length(bestConsensusSet);
objBestConsensusset = 0;

for iter = 1:numIters,

    sampledInds = randperm(N, 2);

    % make corresponding homogeneous vectors
    X1 = [x(sampledInds(1)); y(sampledInds(1)); 1];
    X2 = [x(sampledInds(2)); y(sampledInds(2)); 1];

    % find line as a homogeneous vector
    l = cross(X1, X2);

    % normalize line so that its inner product with another homogeneous vector
    % is equal to the perpendicular distance between the line and the point
    % corresponding to that vector
    scaleFactor = hypot(l(1), l(2));
    l = l / scaleFactor;

    % find distances of all points from line
    distances = abs(l(1) * x + l(2) * y + l(3));

    % threshold and check if a better consensus set has been found
    consensusSet = find(distances < threshold);
    if (length(consensusSet) < sizeBestConsensusSet),
        continue;
    end;

    if ((length(consensusSet) == sizeBestConsensusSet) && ...
        (sum(distances(consensusSet)) > objBestConsensusset))
        continue;
    end;

    bestConsensusSet = consensusSet;
    sizeBestConsensusSet = length(consensusSet);
    objBestConsensusset = sum(distances(consensusSet));
end;

% fit line to selected points
xConsensus = x(bestConsensusSet);
yConsensus = y(bestConsensusSet);
AConsensus = [xConsensus, yConsensus, ones(sizeBestConsensusSet, 1)];
[¬, ¬, VConsensus] = svd(AConsensus);
lineConsensus = VConsensus(:, 3);
```

```
88
89  % create extreme points of calculated lines
90  xCoords = [1 dims(2)];
91  yCoords = - (lineConsensus(1) * xCoords + lineConsensus(3)) / lineConsensus(2);
92
93  % plot resuts on top of "dots_outliers"
94  plot(xCoords, yCoords, 'g-', 'LineWidth', 2);
```

Figure 6 shows the fitted lines with and without using RANSAC to do outlier detection.

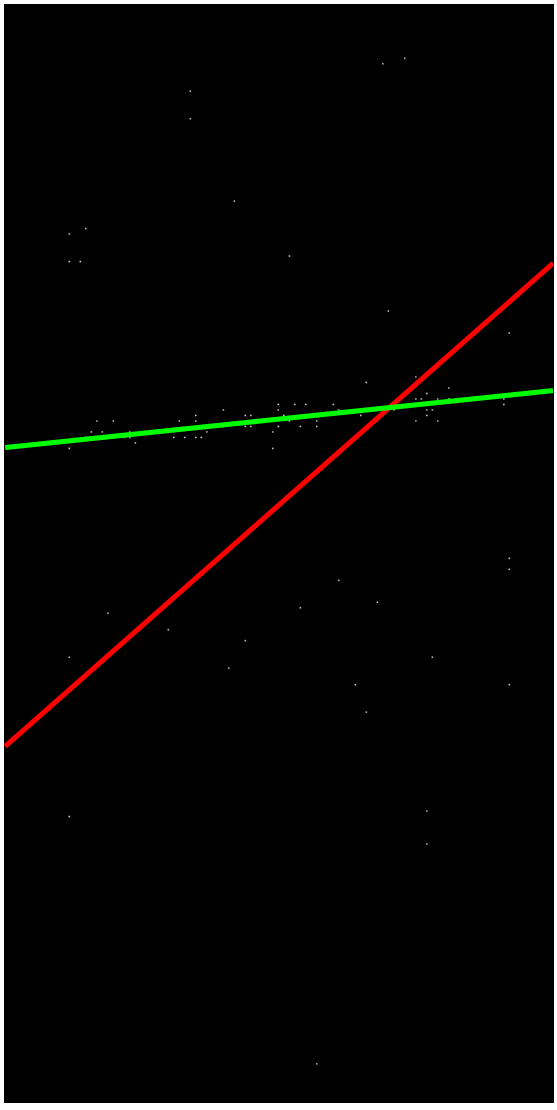

Figure 6: Figure depicting the fitted lines with (green) and without (red) using RANSAC.

5. The area of the triangle with vertices $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$ is given by

$$\frac{1}{2}\left|\det(\boldsymbol{x}_2 - \boldsymbol{x}_1 \quad \boldsymbol{x}_3 - \boldsymbol{x}_1)\right| = \frac{1}{2}\left|\det\begin{pmatrix} m & 0 \\ 0 & m \end{pmatrix}\right| = \frac{m^2}{2}. \tag{14}$$

Now when the affine transformation is applied to all coordinates, the area becomes

$$\frac{1}{2}\left|\det(\boldsymbol{x}_2' - \boldsymbol{x}_1' \quad \boldsymbol{x}_3' - \boldsymbol{x}_1')\right| = \frac{1}{2}\left|\det\begin{pmatrix} ma_{11} & ma_{12} \\ ma_{21} & ma_{22} \end{pmatrix}\right| = \frac{m^2}{2}\left|a_{11}a_{22} - a_{21}a_{12}\right|. \tag{15}$$

The ratios of the two right angled triangles before warping is

$$R = \frac{A_1}{A_2} = \frac{m_1^2/2}{m_2^2/2} = \frac{m_1^2}{m_2^2}. \tag{16}$$

After warping, the ratio is given by

$$R' = \frac{A_1'}{A_2'} = \frac{m_1^2 \left|a_{11}a_{22} - a_{21}a_{12}\right|/2}{m_2^2 \left|a_{11}a_{22} - a_{21}a_{12}\right|/2} = \frac{m_1^2}{m_2^2}. \tag{17}$$

Therefore $R = R'$ and the ratio of areas is preserved by an affine transformation.