ECE661: Computer Vision (Fall 2014)

Shaobo Fang: s-fang@purdue

September 18, 2014

Contents

1 Theory: Projective Distortion and Affine Distortion Removal Using 2-Step Method

Introduction

In two step method, we first need to estimate homography matrix $H_{projective}$ that can remove projective distortion. After the projective distortion was removed, we then estimate the homography matrix H_{affine} which will remove affine distortion accordingly.

Estimation of Homography: $H_{projective}$

To estimate the 3 by 3 homography matrix H that can remove the projective distortion, we need to obtain the **pixel coordinates of the four corners of a rectangle**. This will be implemented in `Matlab` using function `'ginput'`, so that all desired points could be picked conveniently by clicking at the points using computer mice. As projective distortion can be removed by estimating the `Vanishing Line`, we need to have at least **TWO SETS** of parallel lines in real world plane.

As those four coordinates we pick $(x'_1, y'_1), (x'_2, y'_2), (x'_3, y'_3), (x'_4, y'_4)$ are from four corners of a rectangle, it means **in IMAGE plane**:

$$l'_1 = \begin{pmatrix} x'_1 \\ y'_1 \\ 1 \end{pmatrix} \times \begin{pmatrix} x'_2 \\ y'_2 \\ 1 \end{pmatrix}$$

$$l_2 = \begin{pmatrix} x'_3 \\ y'_3 \\ 1 \end{pmatrix} \times \begin{pmatrix} x'_4 \\ y'_4 \\ 1 \end{pmatrix}$$

$$l'_3 = \begin{pmatrix} x'_1 \\ y'_1 \\ 1 \end{pmatrix} \times \begin{pmatrix} x'_3 \\ y'_3 \\ 1 \end{pmatrix}$$

$$l'_4 = \begin{pmatrix} x'_2 \\ y'_2 \\ 1 \end{pmatrix} \times \begin{pmatrix} x'_4 \\ y'_4 \\ 1 \end{pmatrix}$$

As in real world plane we know, $l_1 \parallel l_2$ *and* $l_3 \parallel l_4$. In image plane Vanishing Points at:

$$P' = l'_1 \times l'_2$$

$$Q' = l'_3 \times l'_4$$

Vanishing Line:

$$l^{VL} = P' \times Q' = \begin{pmatrix} l_1^{VL} \\ l_2^{VL} \\ l_3^{VL} \end{pmatrix}$$

Accordingly, we can form:

$$H_{projetive} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1^{VL} & l_2^{VL} & l_3^{VL} \end{bmatrix}$$

Estimation of Homography: H_{affine}

Assume that the Dual Degenerate Conic is:

$$C_\infty^* = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Thus, the angles θ between two lines $l = \begin{pmatrix} l_1 \\ l_2 \\ l_3 \end{pmatrix}$, and $m = \begin{pmatrix} m_1 \\ m_2 \\ m_3 \end{pmatrix}$ would have the following relationship:

$$\cos\theta = \frac{l_1 m_1 + l_2 m_2}{\sqrt{(l_1^2 + l_2^2)(m_1^2 + m_2^2)}}$$

From lecture notes, substitute the C_∞^* into the equation:

$$\cos\theta = \frac{l^T C_\infty^* m}{\sqrt{(l^T C_\infty^* l)(m^T C_\infty^* m)}}$$

Similarly, in image plane as:

$$C^{*'} = H C^* H^T$$

$$\cos\theta|_{numerator} = (l'^T H)(H^{-1} C_\infty^{*'} H^{-T})(H^T m') = l'^T C_\infty^{*'} m'$$

Now assume that the angle the real world plane is 90 degrees, this means:

$$l'^T H C_\infty^* H^T m' = 0$$

Define:

$$H_{affine} = \begin{bmatrix} A & \vec{t} \\ \vec{0}^T & 1 \end{bmatrix}$$

$$(l_1', l_2', l_3') \begin{bmatrix} A & \vec{t} \\ \vec{0}^T & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} A^T & \vec{0} \\ \vec{t}^T & 1 \end{bmatrix} \begin{pmatrix} m_1' \\ m_2' \\ m_3' \end{pmatrix} = 0$$

Collapses Into:

$$(l'_1, l'_2, l'_3) \begin{bmatrix} AA^T & \vec{0} \\ \vec{0}^T & 0 \end{bmatrix} \begin{pmatrix} m'_1 \\ m'_2 \\ m'_3 \end{pmatrix} = 0$$

Set $S = AA^T$:

$$(l'_1, l'_2) \begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix} \begin{pmatrix} m'_1 \\ m'_2 \end{pmatrix} = 0, \ with \ s_{12} = s_{21}$$

Which Implies:

$$s_1 l'_1 m'_1 + s_{12}(l'_1 m'_2 + l'_2 m'_1) + s_{22} l'_2 m'_2 = 0$$

Set $s_{22} = 1$, we need two pair of orthogonal lines to solve for matrix S. It would be better if there two pairs of orthogonal lines are not pairwise parallel. We can pick a $SQUARE$ in our image plane so the diagonal lines would also be orthogonal. After the matrix S is solved, applying SVD:

$$[U, D, U^t] = svd(A)$$
$$S = AA^T = UDU^tUDU^t = UD^2U^t$$
$$[U, D^2, U^t] = svd(S)$$

After A is obtained using singular values decomposition, homography matrix H_{affine} can be constructed.

Conclusion of Two Step Method To conclude, in order to remove both affine and projective distortion, we can form matrix $H = H^{-1}_{projective} H_{affine}$.

2 Theory: Projective Distortion and Affine Distortion Removal Using Single Step Method

For single step method, assume the homography:

$$H = \begin{bmatrix} A & \vec{0} \\ \vec{v}^T & 1 \end{bmatrix}$$

Similarly as in two step method:

$$C^*_\infty = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$C^{*'}_\infty = HC^*_\infty H^T = \begin{bmatrix} AA^T & A\vec{v} \\ \vec{v}^T A^T & \vec{v}^T \vec{v} \end{bmatrix}$$

Now writhe the $C^{*'}_\infty$ in general form:

$$C_\infty^{*'} = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix}$$

If $l \perp m$ in real world plane, then in image plane:

$$(l_1', l_2', l_3') \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} \begin{pmatrix} m_1' \\ m_2' \\ m_3' \end{pmatrix} = 0$$

$$(l_1'm_1' \quad \frac{l_2'm_1' + l_1'm_2'}{2} \quad l_2'm_2' \quad \frac{l_1'm_3' + l_3'm_1'}{2} \quad \frac{l_2'm_3' + l_3'm_2'}{2} \quad l_3'm_3') \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix} = 0$$

Set $f = 1$, **THIS MEANS WE NEED AT LEAST FIVE PAIRS OF ORTHOGONAL LINES**

Recall that:

$$C_\infty^{*'} = HC_\infty^* H^T = \begin{bmatrix} AA^T & A\vec{v} \\ \vec{v}^T A^T & \vec{v}^T\vec{v} \end{bmatrix}$$

$$AA^T = \begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix}$$

As in previous steps matrix A could be solved by applying the SVD. Then, easily $\vec{v}$ can also be solved by:

$$A\vec{v} = \begin{pmatrix} d/2 \\ e/2 \end{pmatrix}$$

.

3 Debugging Technique

Debugging Technique For Single Step Method

Note that, after we solved both A and $\vec{v}$, we should be able to verify that:
A. The Product Should Match:

$$C_\infty^{*'} = HC_\infty^* H^T = \begin{bmatrix} AA^T & A\vec{v} \\ \vec{v}^T A^T & \vec{v}^T\vec{v} \end{bmatrix}$$

B. In Image Plane:

$$(l'_1, l'_2, l'_3) \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} \begin{pmatrix} m'_1 \\ m'_2 \\ m'_3 \end{pmatrix} = 0$$

C. In Real World Plane (After Inverse Projection):

$$l \perp m$$

4 Comparison of Two Methods

1. **Computation Efficiency:** As the single step method would only need to apply inverse homography H once, theoretically single step method should be better in computational efficiency.

2. **Algorithms Complexity:** Single step method is more straight forward and is easier to understand theoretically.

3. **Implementation Complexity:** Personally I believe two step method is easier to implement as we need significant less points selected from images compared to single step method.

4. **System Stability:** Generally, the two-step method is more stable. It means although the results would not be very ideal if `bad set of points` were selected but the results should still be acceptable. Single step method need significantly more `orthogonal` features compared to two-step method. This has also been mentioned by student `Sriram Karthik Badam` in his sample solutions from `Fall 2012`. Especially in `Matlab`, if the 5 pairs of orthogonal lines were not 'good', the result would be unacceptable. This problem might be solved by implementing the algorithms in C or C++ language.

5. **Results:** For Matlab I believe `Sriram Karthik Badam`'s conclusion also fit. With the right sets of orthogonal lines, the results from single step method is comparable to two step method.

5 Point Selection Features

1. To make it easier for the user to pick points, Matlab function ginput was utilized.

2. For those four points on the rectangle, the points should be picked according to the following order: Upper Left → Upper Right → Lower Left → Lower Right.

6 Results

6.1 Original Images

Fig 1. Img1.jpg of Set1

Fig 2. Img2.jpg of Set1

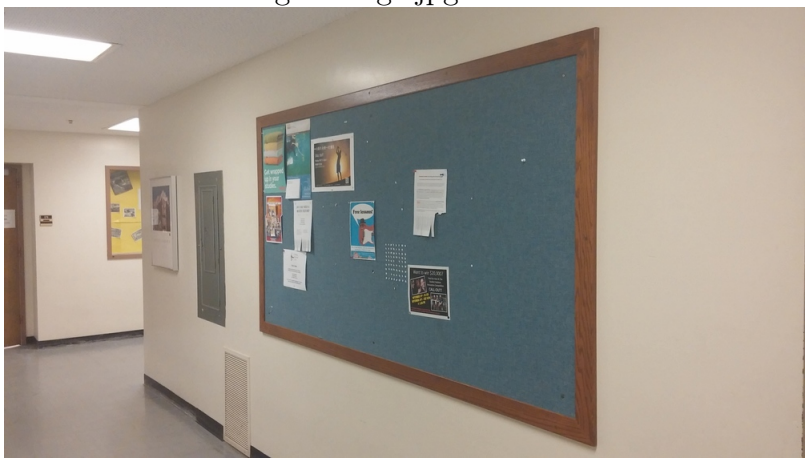

Fig 3. Img1.jpg of Set2

Fig 4. Img2.jpg of Set2

Fig 5. Img1.jpg of Set3

Fig 6. Img2.jpg of Set3

Fig 7. Img1.jpg of Set4

Fig 8. Img1.jpg of Set4

Fig 9. Image1 of My Selection

Fig 10. Image2 of My Selection

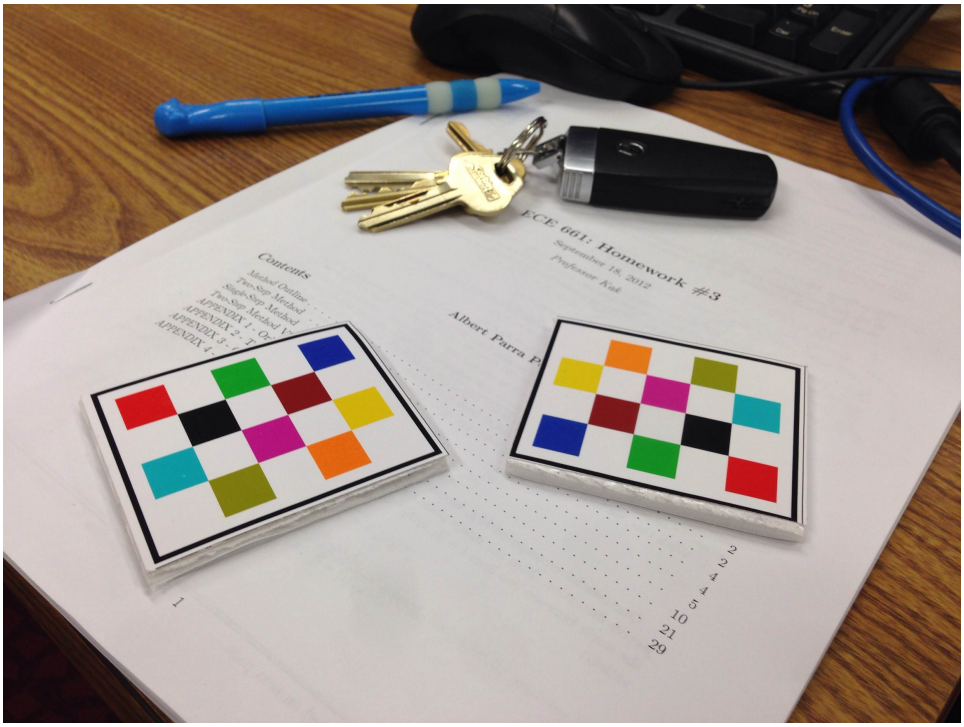

Fig 11. Image3 of My Selection

6.2 Results For Two-Step Method

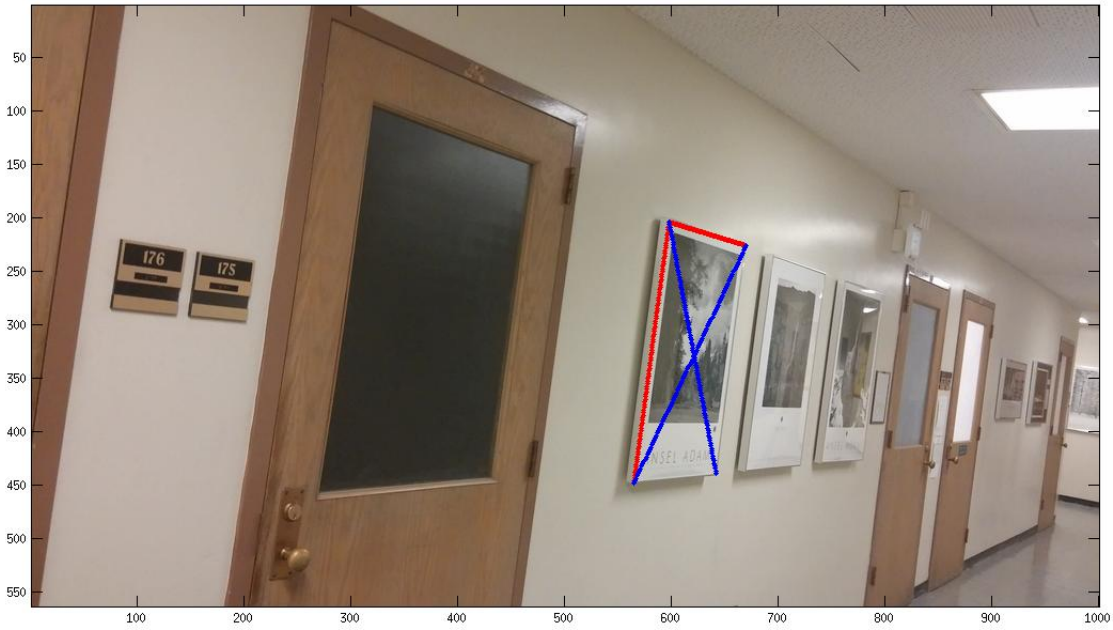

Fig 12. `Set1-Img1` The Rectangle Area Selected To Estimate Homography $H_{projective}$. **Note that although diagonal lines were demonstrated it was not used for removing projective distortion**

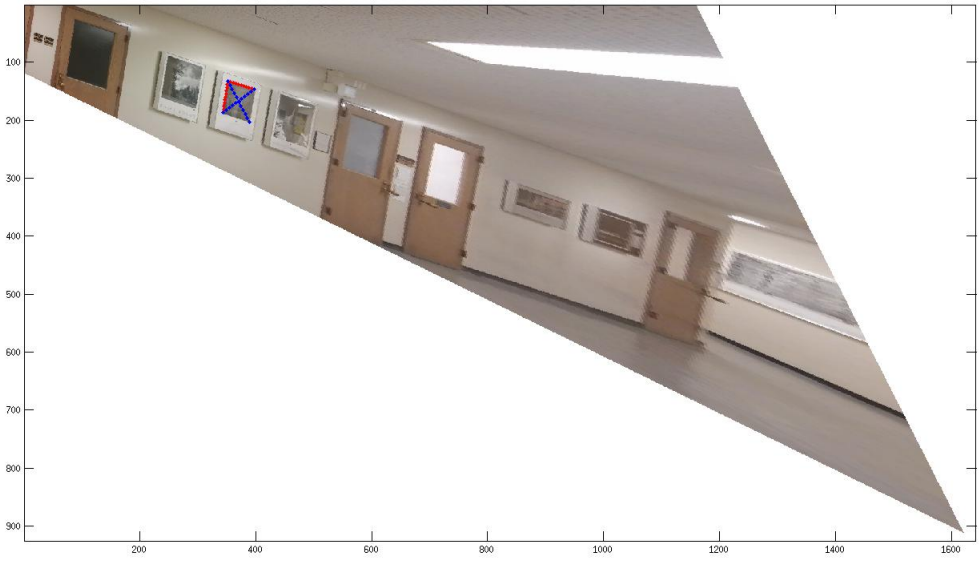

Fig 13. `Set1-Img1` Projective Distortion Removed. The Squared Area Selected To Estimate Homography. H_{affine}. **The squared area with diagonal marked is the the**

area used to estimate H_{affine}

Fig 14. `Set1-Img1` Corrected Image With Both Projective Distortion and Affine Distortion Removed. **Note that the squared area selected in previous plot is more squared now**

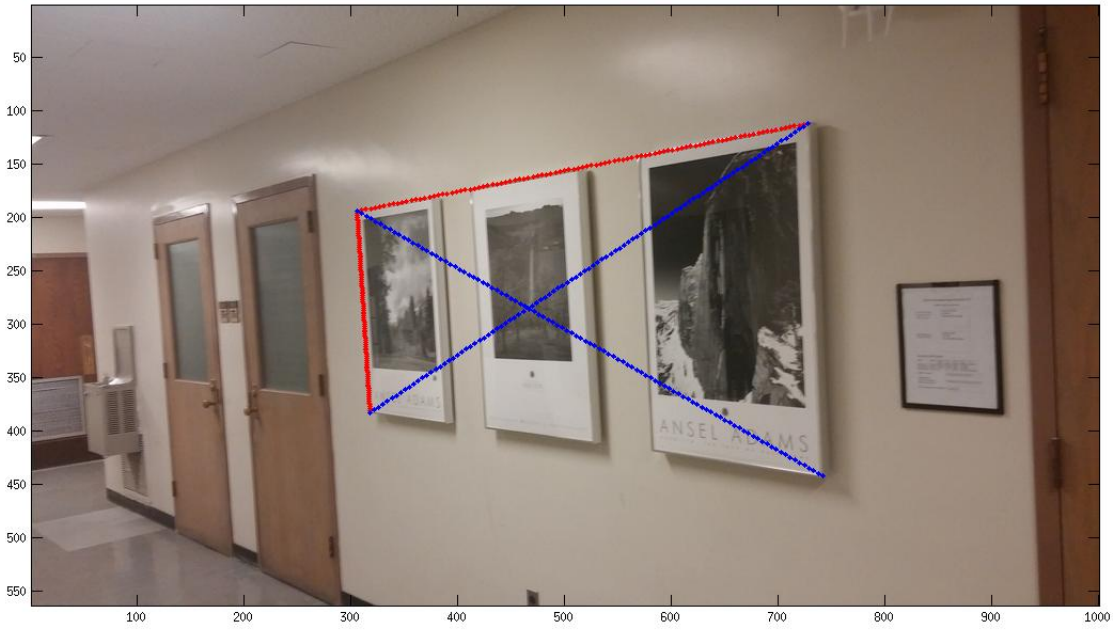

Fig 15. `Set1-Img2` The Rectangle Area Selected To Estimate Homography $H_{projective}$. **Note that although diagonal lines were demonstrated it was not used for removing projective distortion**

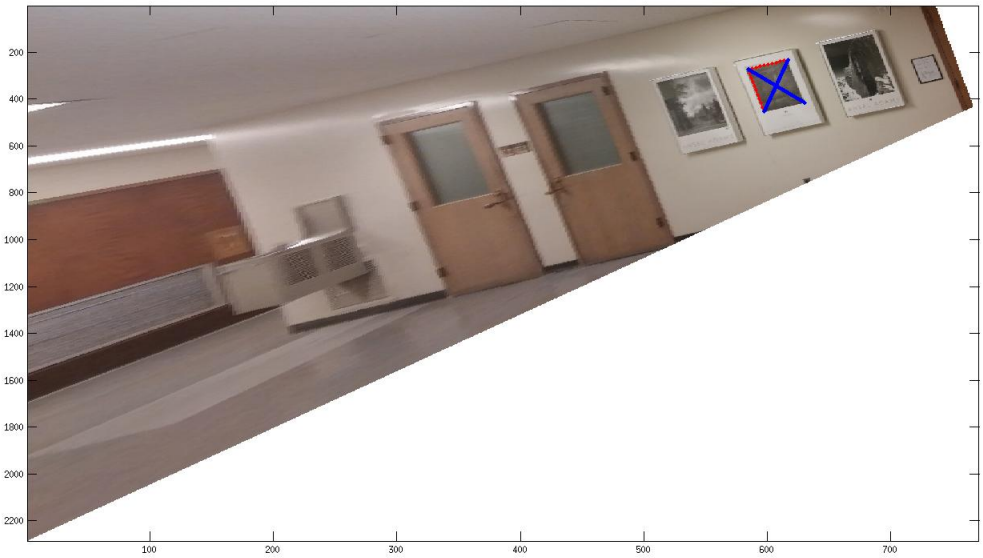

Fig 16. `Set1-Img2` Projective Distortion Removed. The Squared Area Selected To Estimate Homography. H_{affine}. **The squared area with diagonal marked is the the area used to estimate** H_{affine}

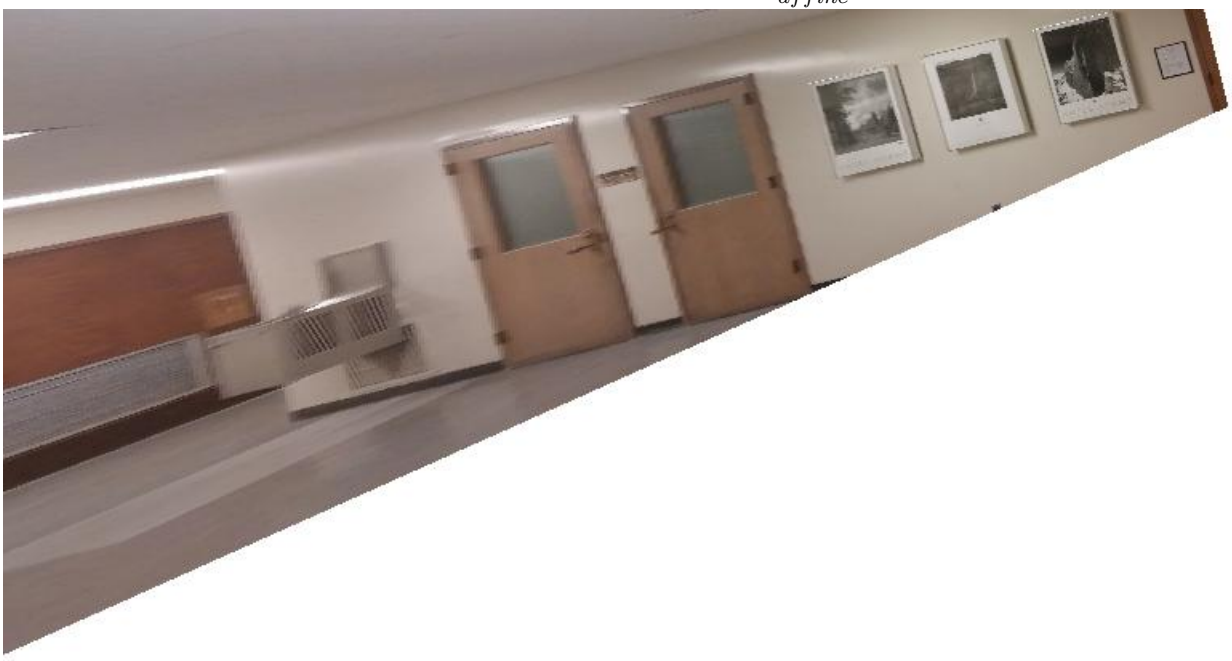

Fig 17. `Set1-Img2` Corrected Image With Both Projective Distortion and Affine Distortion Removed. **Note that the squared area selected in previous plot is more squared now**

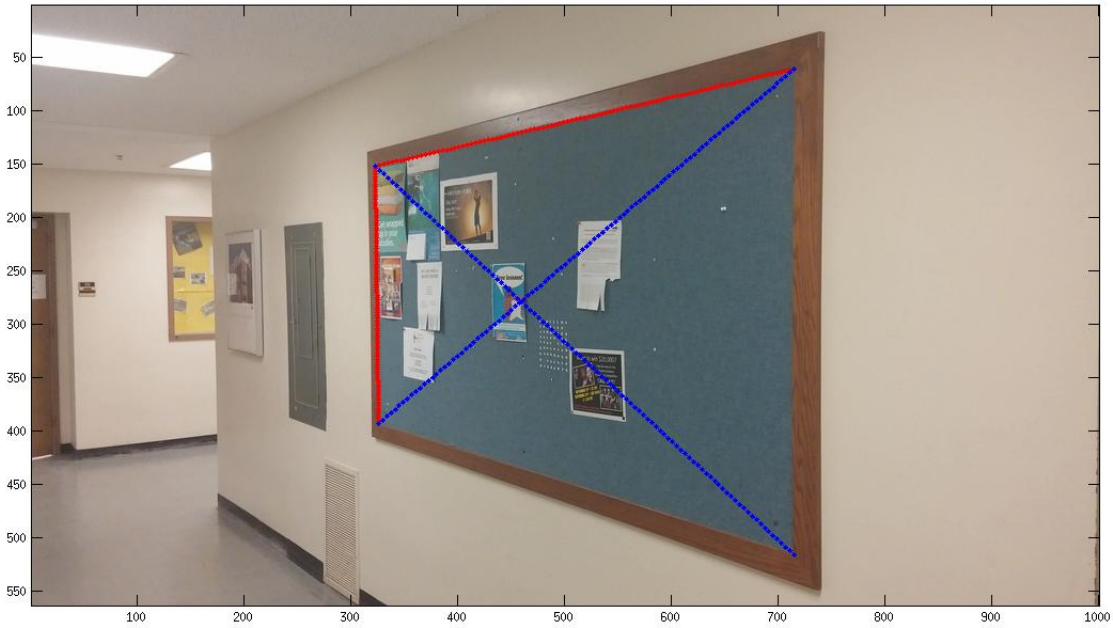

Fig 18. `Set2-Img1` The Rectangle Area Selected To Estimate Homography $H_{projective}$. **Note that although diagonal lines were demonstrated it was not used for removing projective distortion**

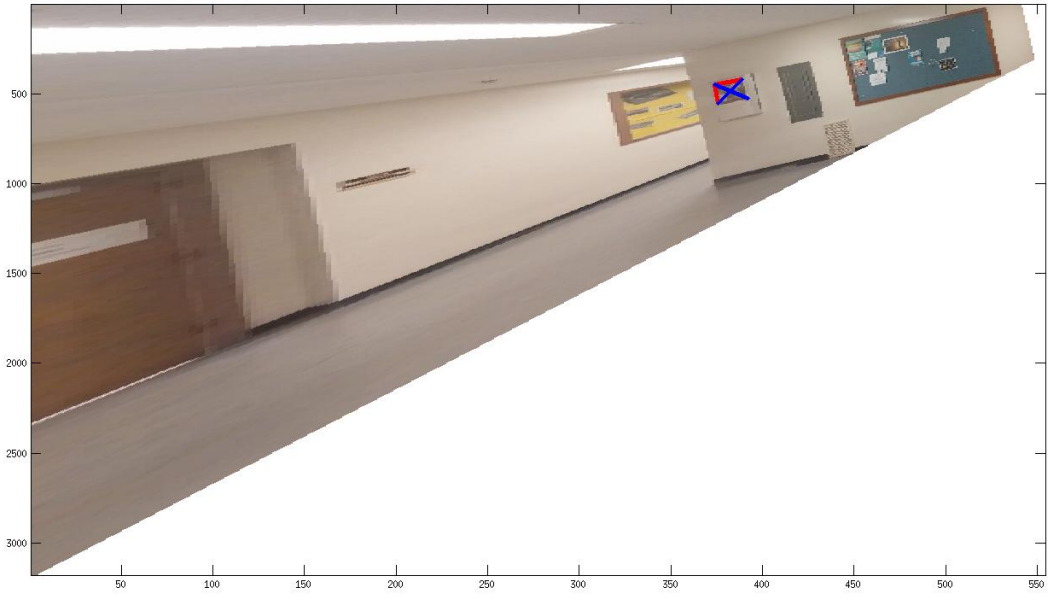

Fig 19. `Set2-Img1` Projective Distortion Removed. The Squared Area Selected To Estimate Homography. H_{affine}. **The squared area with diagonal marked is the the**

area used to estimate H_{affine}

Fig 20. `Set2-Img1` Corrected Image With Both Projective Distortion and Affine Distortion Removed. **Note that the squared area selected in previous plot is more squared now**

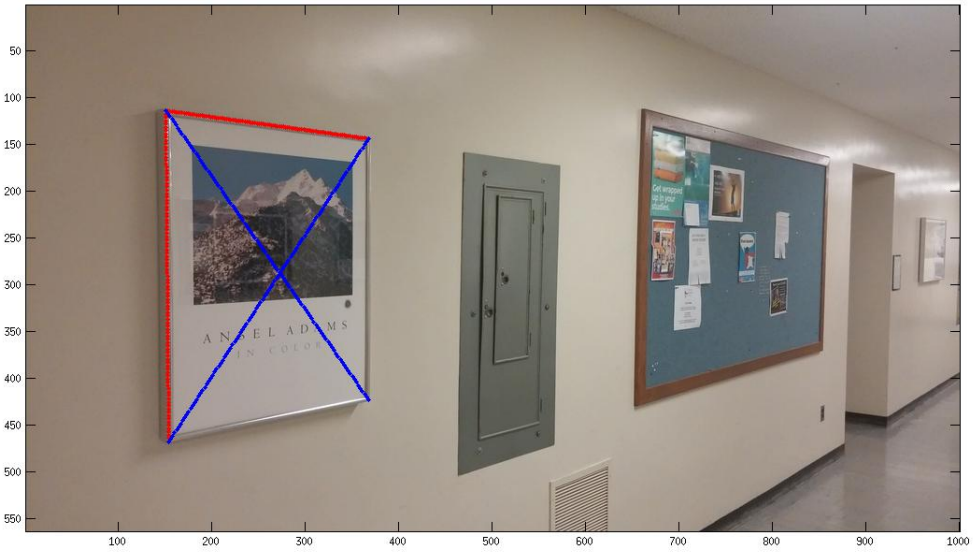

Fig 21. `Set2-Img2` The Rectangle Area Selected To Estimate Homography $H_{projective}$. **Note that although diagonal lines were demonstrated it was not used for removing projective distortion**

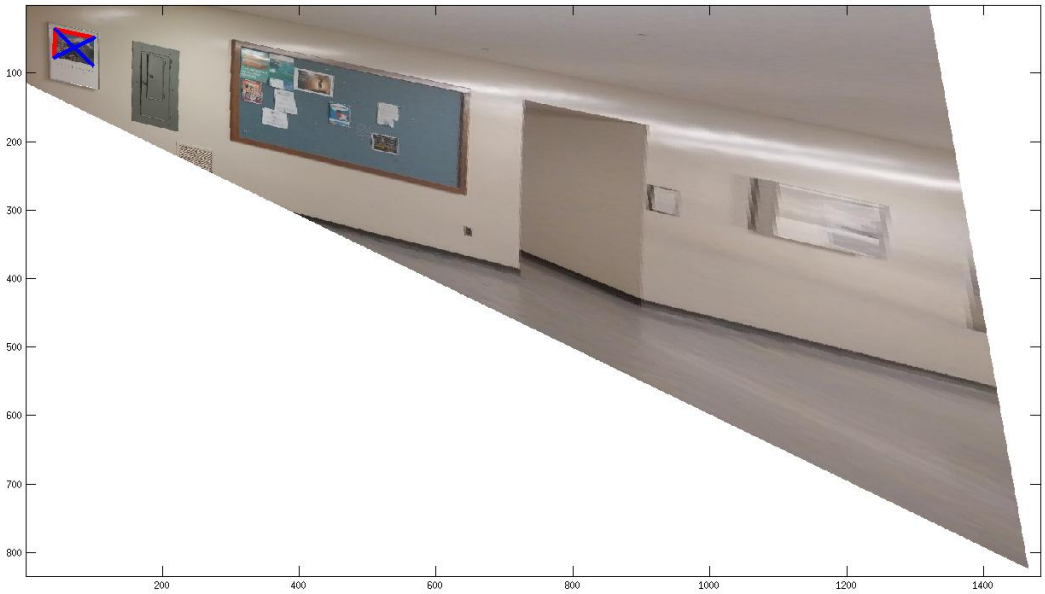

Fig 22. `Set2-Img2` Projective Distortion Removed. The Squared Area Selected To Estimate Homography. H_{affine}. **The squared area with diagonal marked is the the area used to estimate** H_{affine}

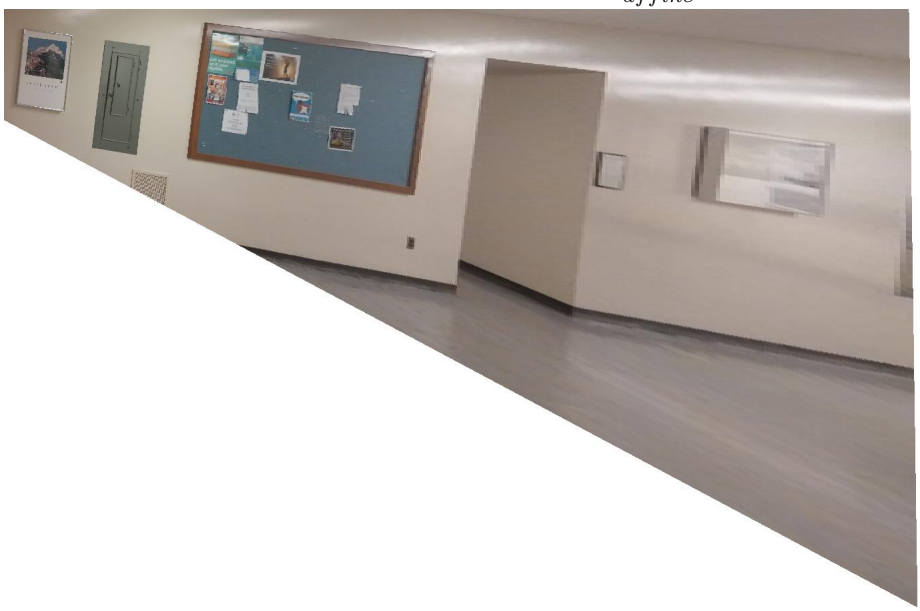

Fig 23. `Set2-Img2` Corrected Image With Both Projective Distortion and Affine Distortion

Removed. **Note that the squared area selected in previous plot is more squared now.**

Fig 24. `Set3-Img1` The Rectangle Area Selected To Estimate Homography $H_{projective}$. **Note that although diagonal lines were demonstrated it was not used for removing projective distortion**

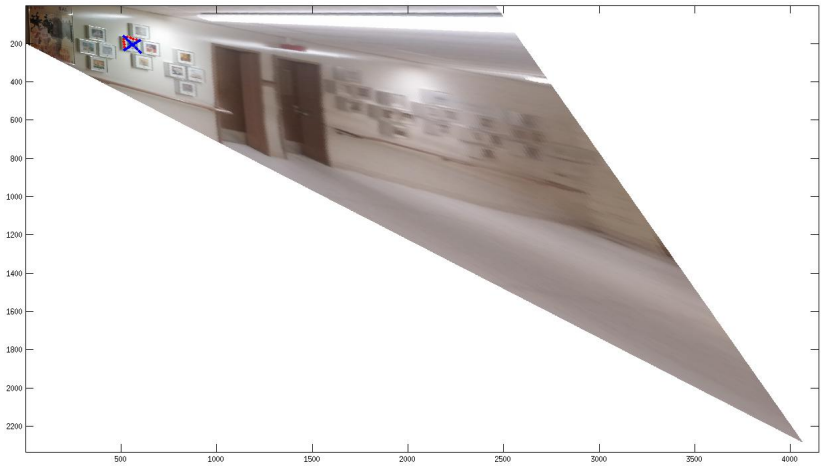

Fig 25. `Set3-Img1` Projective Distortion Removed. The Squared Area Selected To Estimate Homography. H_{affine}. **The squared area with diagonal marked is the the area used to estimate** H_{affine}

Fig 26. `Set3-Img1` Corrected Image With Both Projective Distortion and Affine Distortion Removed. **Note that the squared area selected in previous plot is more squared now.**

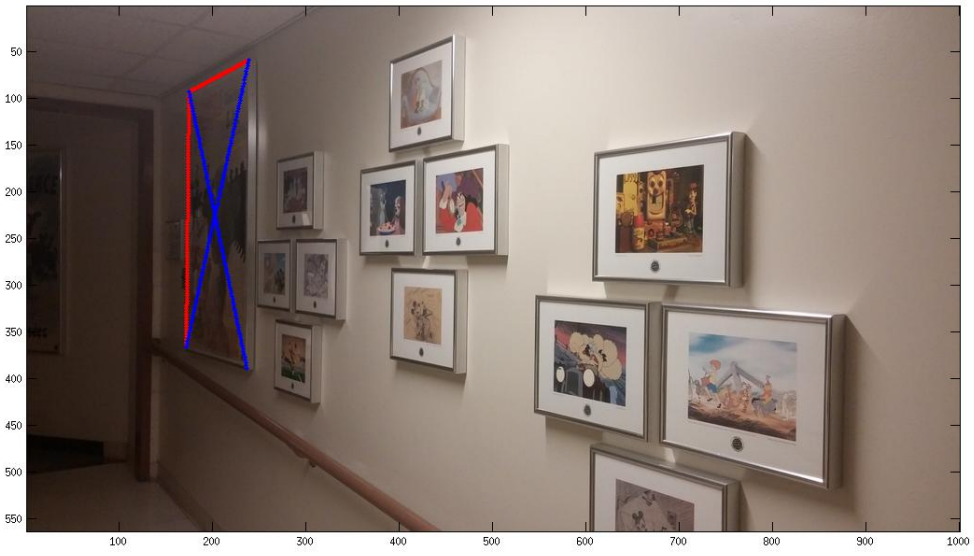

Fig 27. `Set3-Img2` The Rectangle Area Selected To Estimate Homography $H_{projective}$. **Note that although diagonal lines were demonstrated it was not used for removing projective distortion**

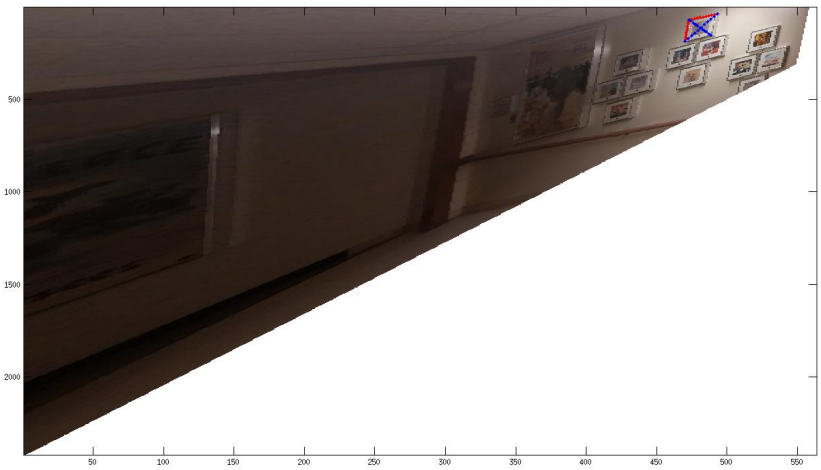

Fig 28. `Set3-Img2` Projective Distortion Removed. The Squared Area Selected To Estimate Homography. H_{affine}. **The squared area with diagonal marked is the the area used to estimate** H_{affine}

Fig 29. `Set3-Img2` Corrected Image With Both Projective Distortion and Affine Distortion Removed. **Note that the squared area selected in previous plot is more squared now.**

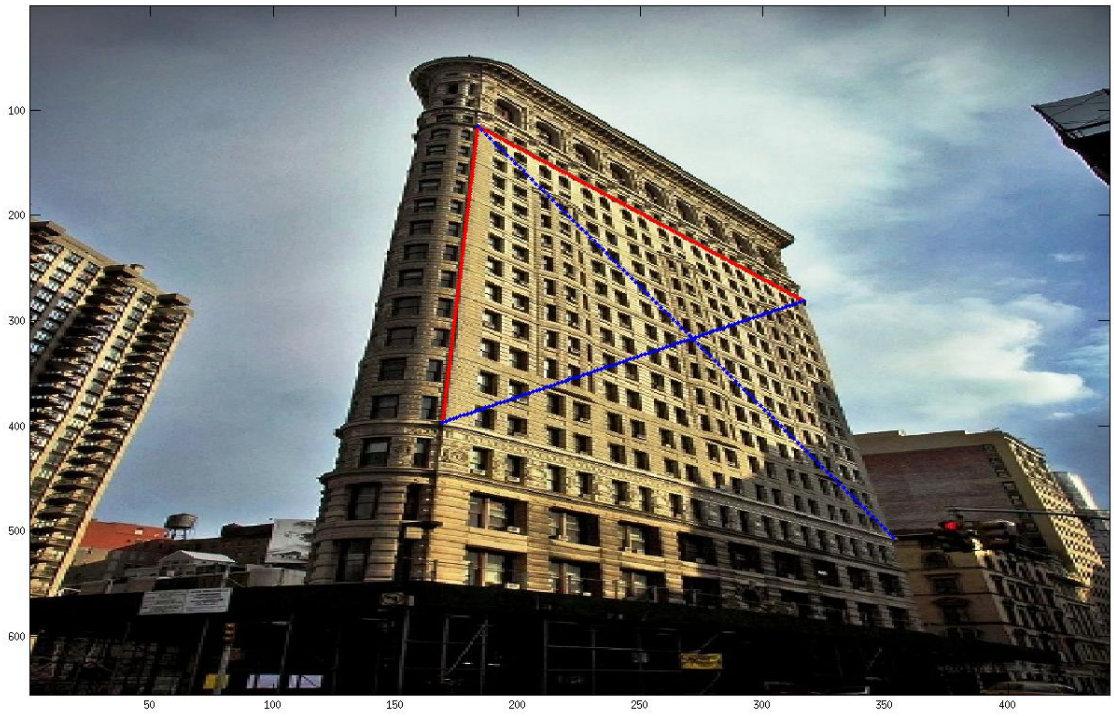

Fig 30. `Set4-Img1` The Rectangle Area Selected To Estimate Homography $H_{projective}$. **Note that although diagonal lines were demonstrated it was not used for removing projective distortion**

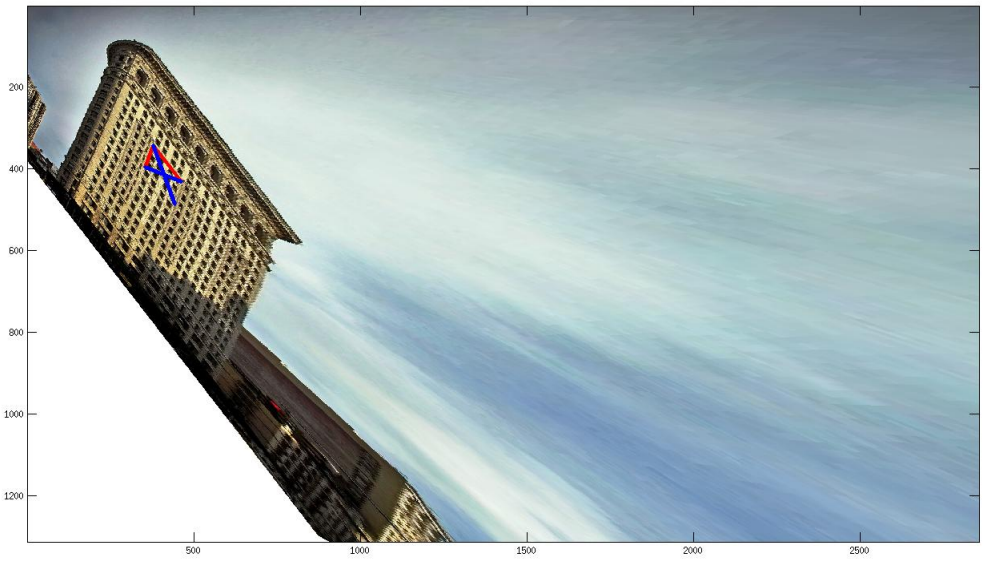

Fig 31. `Set4-Img1` Projective Distortion Removed. The Squared Area Selected To

Estimate Homography. H_{affine}. **The squared area with diagonal marked is the the area used to estimate** H_{affine}

Fig 32. `Set4-Img1` Corrected Image With Both Projective Distortion and Affine Distortion Removed. **Note that the squared area selected in previous plot is more squared now.**

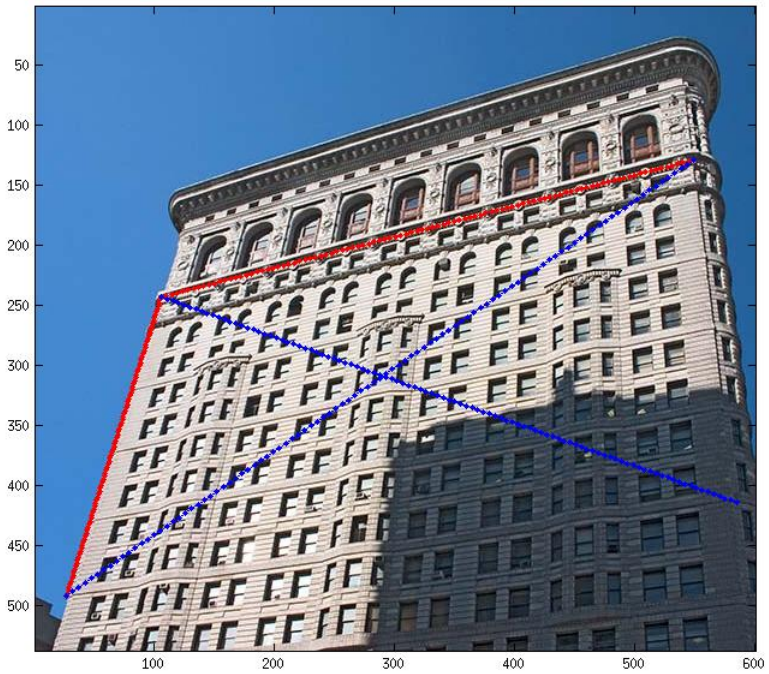

Fig 33. `Set4-Img2` The Rectangle Area Selected To Estimate Homography $H_{projective}$. **Note that although diagonal lines were demonstrated it was not used for removing projective distortion**

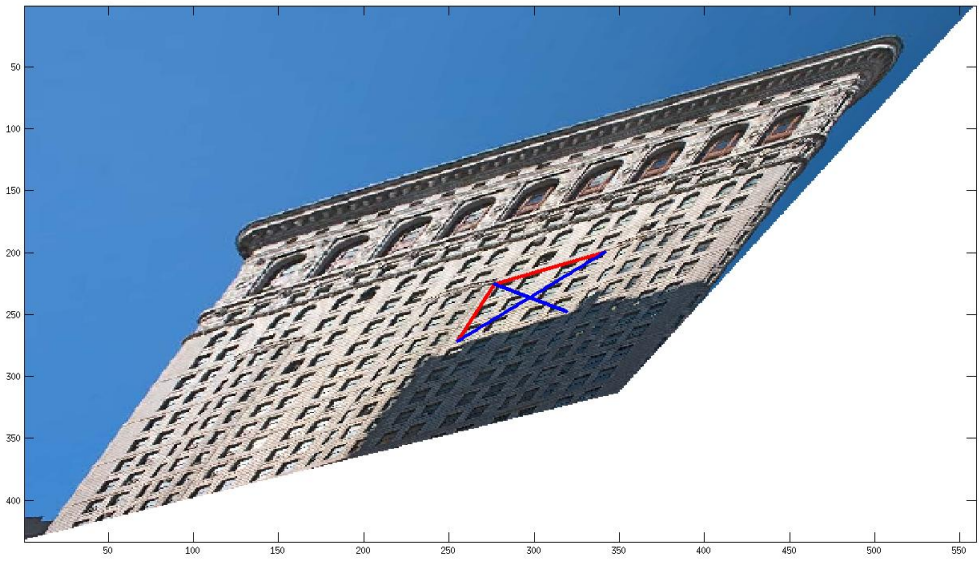

Fig 34. `Set4-Img2` Projective Distortion Removed. The Squared Area Selected To Estimate Homography. H_{affine}. **The squared area with diagonal marked is the the area used to estimate** H_{affine}

Fig 35. `Set4-Img2` Corrected Image With Both Projective Distortion and Affine Distortion Removed. **Note that the squared area selected in previous plot is more squared now.**

Fig 36. `My Image 1` The Rectangle Area Selected To Estimate Homography $H_{projective}$. **Note that although diagonal lines were demonstrated it was not used for removing projective distortion**

Fig 37. `My Image 1` Projective Distortion Removed. The Squared Area Selected To Estimate Homography. H_{affine}. **The squared area with diagonal marked is the the area used to estimate** H_{affine}

Fig 38. `My Image 1` Corrected Image With Both Projective Distortion and Affine Distortion Removed. **Note that the squared area selected in previous plot is more squared now.**

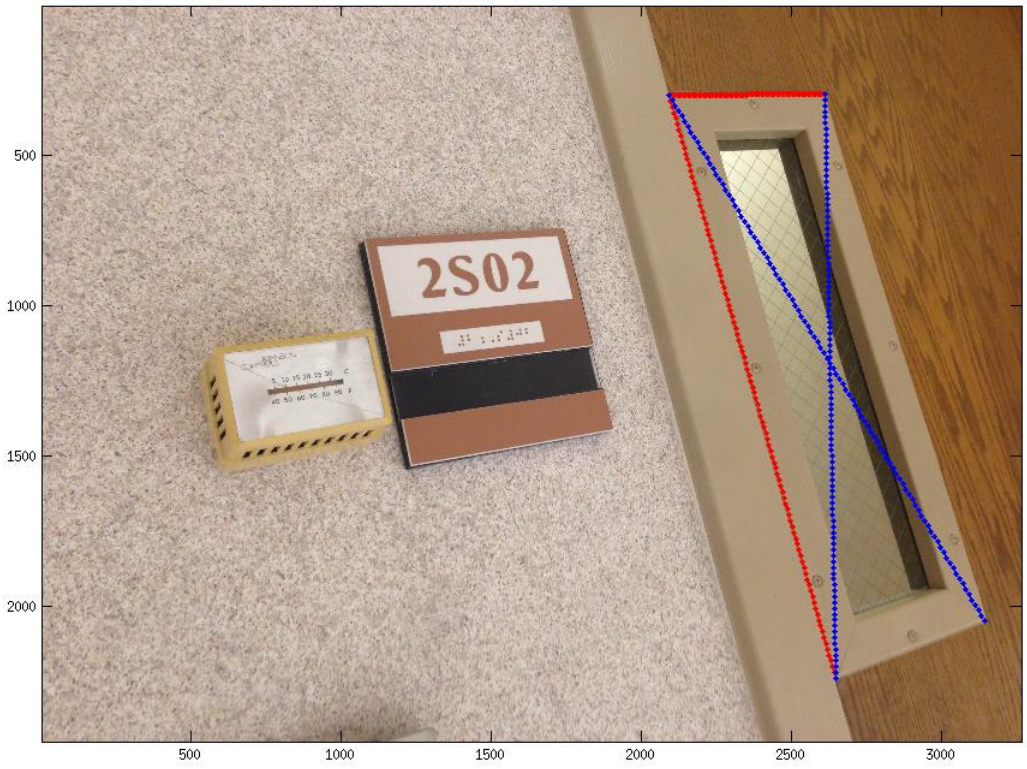

Fig 39. `My Image 2` The Rectangle Area Selected To Estimate Homography $H_{projective}$. **Note that although diagonal lines were demonstrated it was not used for removing projective distortion**

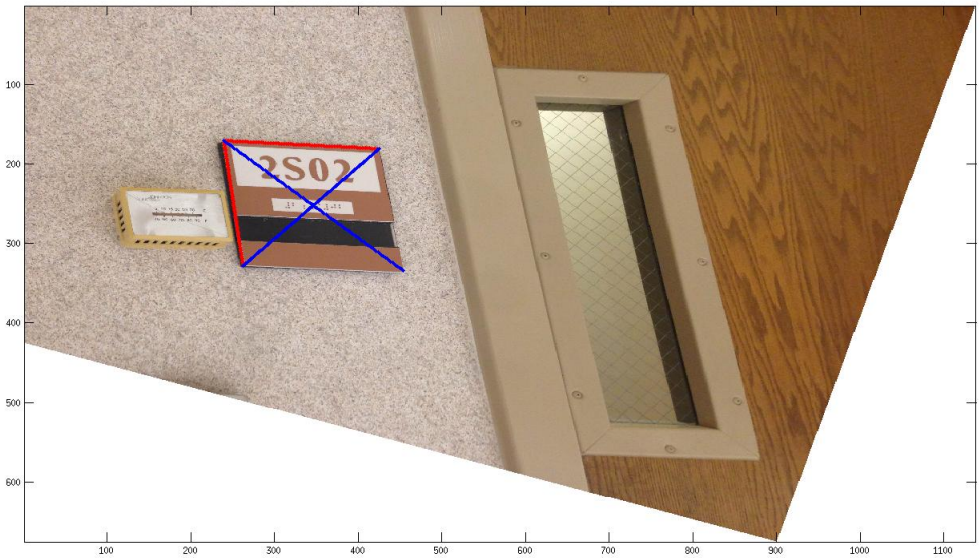

Fig 40. `My Image 2` Projective Distortion Removed. The Squared Area Selected To Estimate Homography. H_{affine}. **The squared area with diagonal marked is the the area used to estimate** H_{affine}

Fig 41. `My Image 2` Corrected Image With Both Projective Distortion and Affine Distortion Removed. **Note that the squared area selected in previous plot is more squared now.**

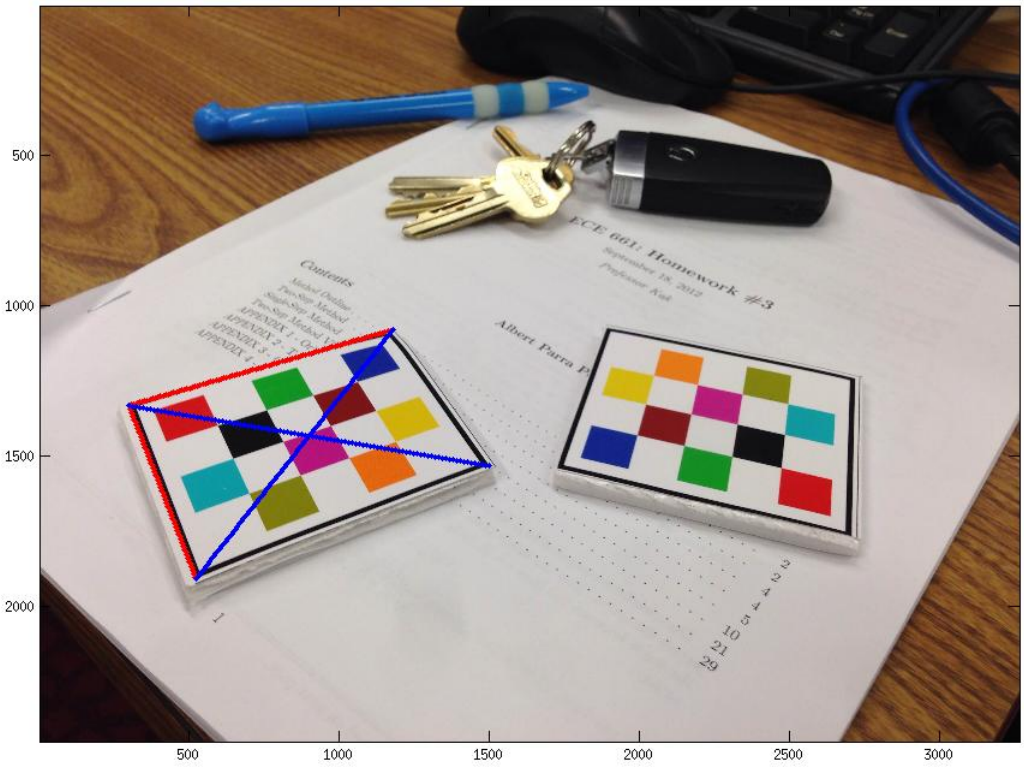

Fig 42. `My Image 3` The Rectangle Area Selected To Estimate Homography $H_{projective}$. **Note that although diagonal lines were demonstrated it was not used for removing projective distortion**

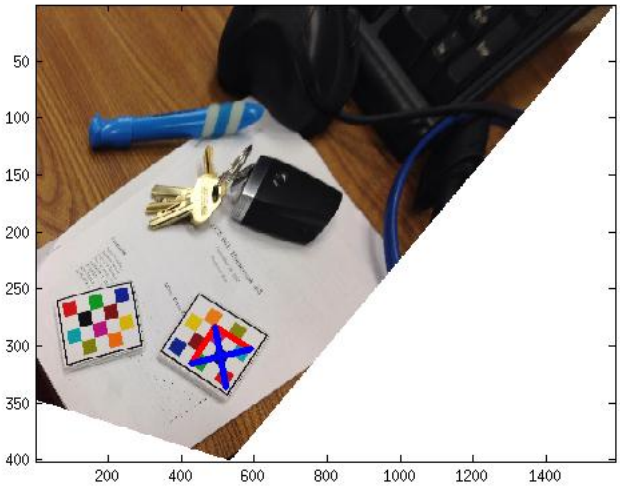

Fig 43. `My Image 3` Projective Distortion Removed. The Squared Area Selected To Estimate Homography. H_{affine}. **The squared area with diagonal marked is the the**

area used to estimate H_{affine}

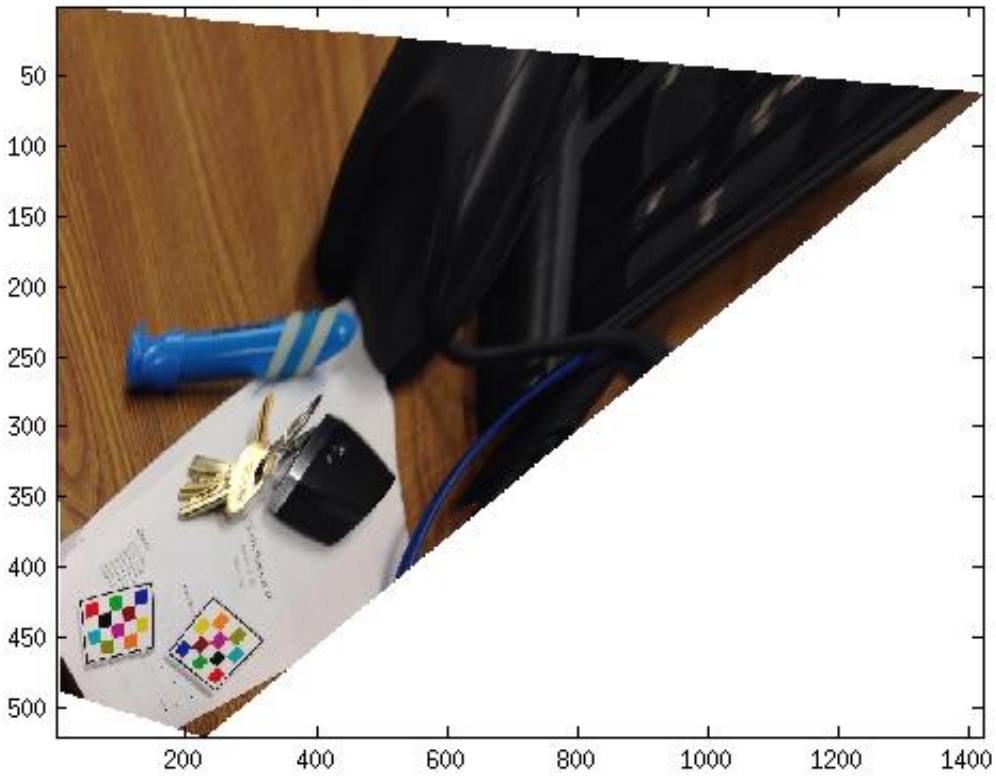

Fig 44. `My Image 3` Corrected Image With Both Projective Distortion and Affine Distortion Removed. **Note that the squared area selected in previous plot is more squared now.**

6.3 Results For Single Step Method

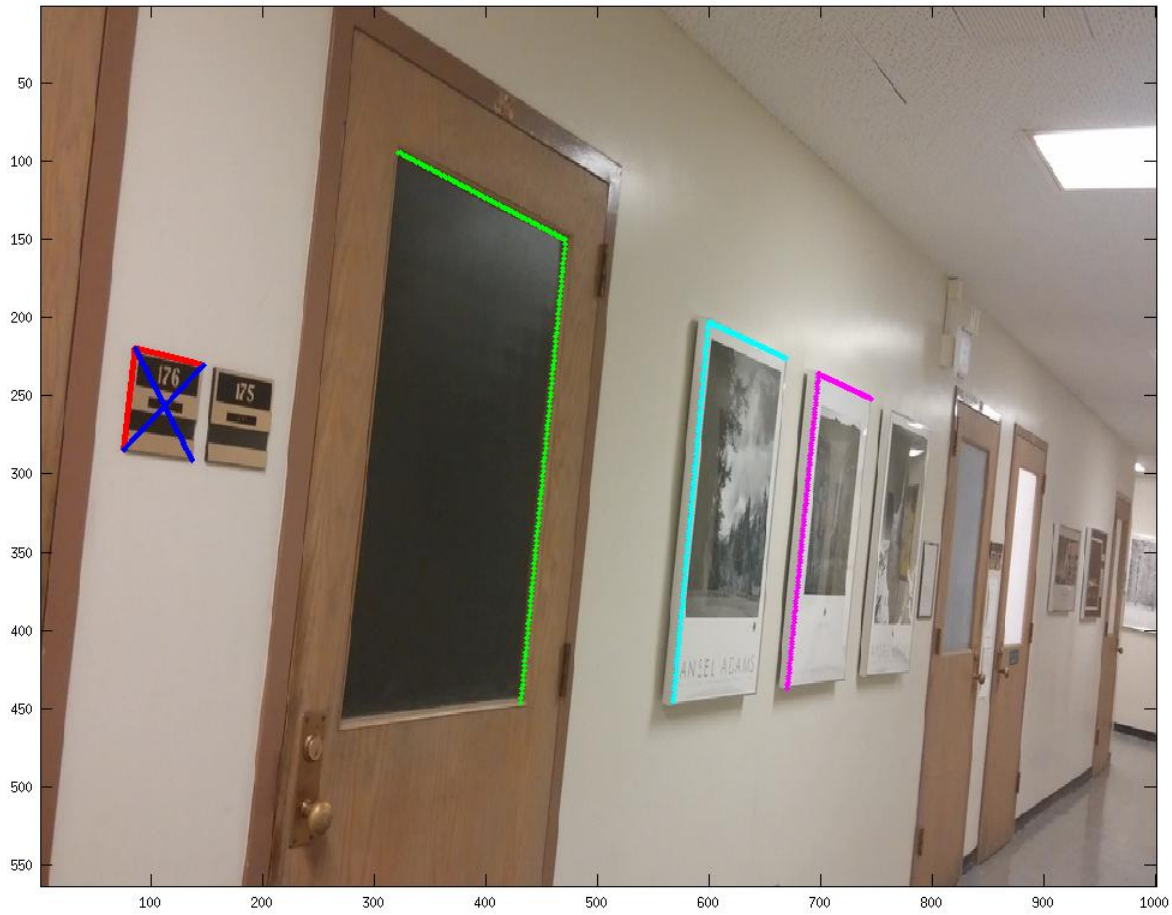

Fig 45. `Set 1 Img 1` Single Step Method, With Orthogonal Lines Selected In Original Image. **Note that the area with diagonal lines crossed is the area we assume to be squared**.

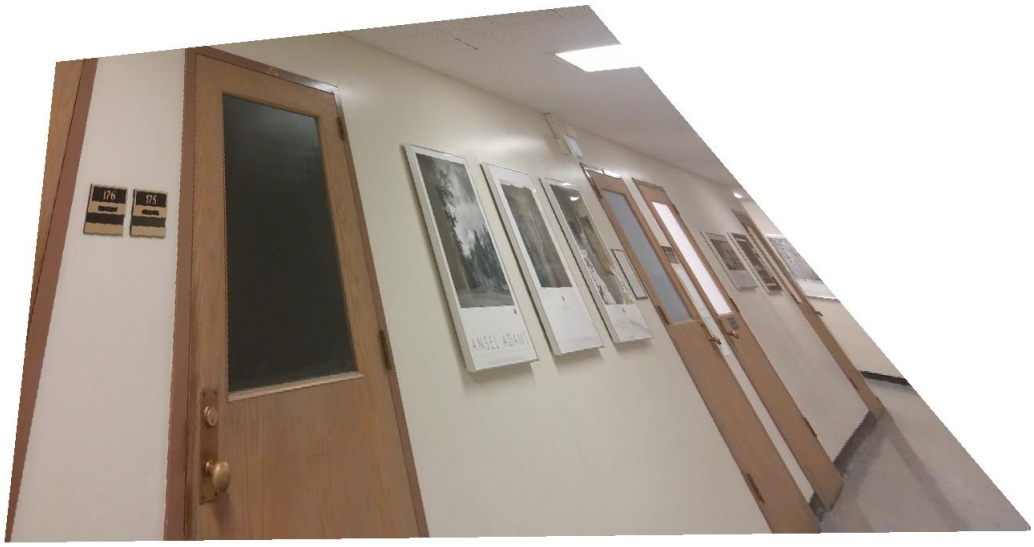

Fig. 46 `Set 1 Img 1` Single Step Method with distortion removed. **Note that since the squared area selected in previous image is too small some distortion could still be observed. But the selected area is more like a 'square' and all lines assumed to be orthogonal are orthogonal.**

Fig 47. `Set 1 Img 2` Single Step Method, With Orthogonal Lines Selected In Original Image. **Note that the area with diagonal lines crossed is the area we assume to be squared**.

Fig. 48 `Set 1 Img 2` Single Step Method with distortion removed. **Note that THIS IS A GREAT RESULT! The selected area is a 'square' and all lines assumed to be orthogonal are orthogonal after correction!**

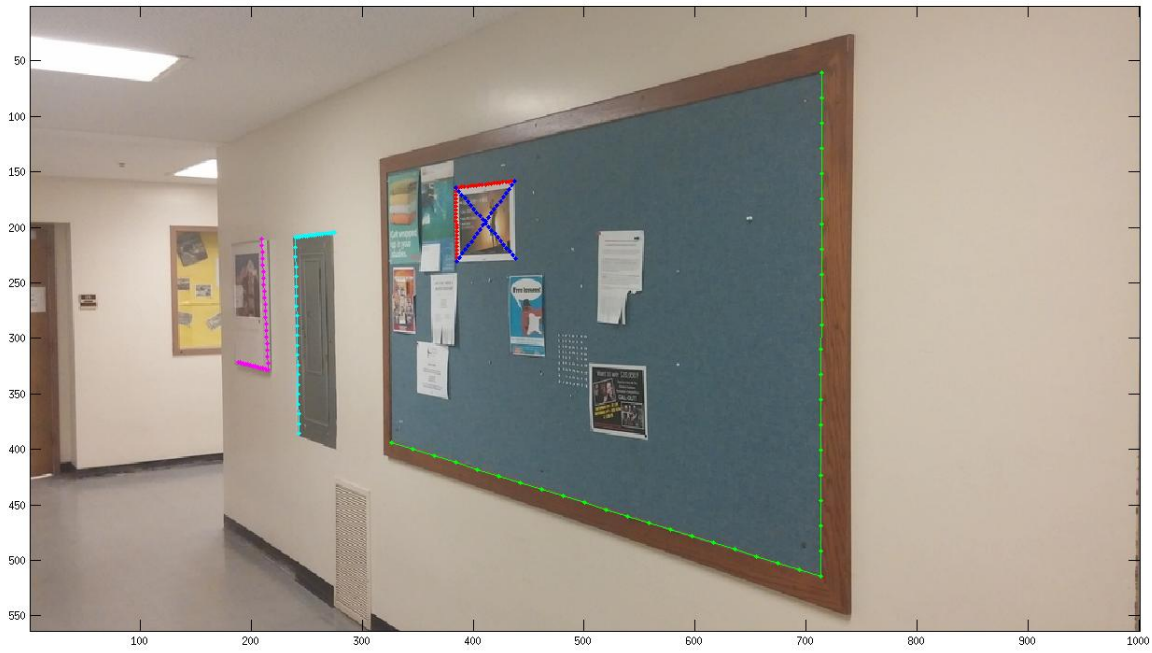

Fig 49. `Set 2 Img 1` Single Step Method, With Orthogonal Lines Selected In Original Image. **Note that the area with diagonal lines crossed is the area we assume to**

be squared.

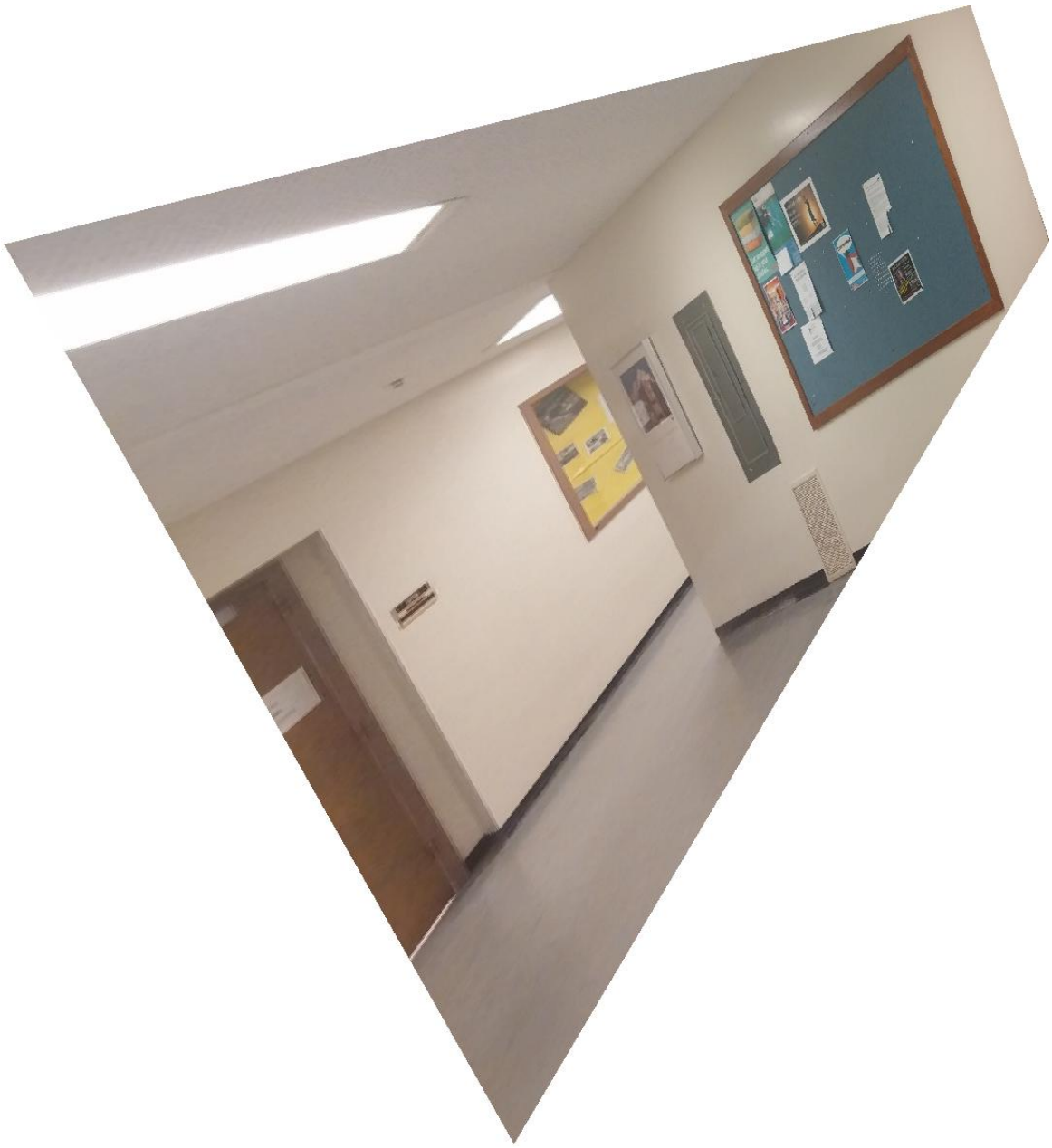

Fig. 50 `Set 2 Img 1` Single Step Method with distortion removed. **Note that THIS IS A GREAT RESULT! The selected area is a 'square' and all lines assumed to be orthogonal are orthogonal after correction!**

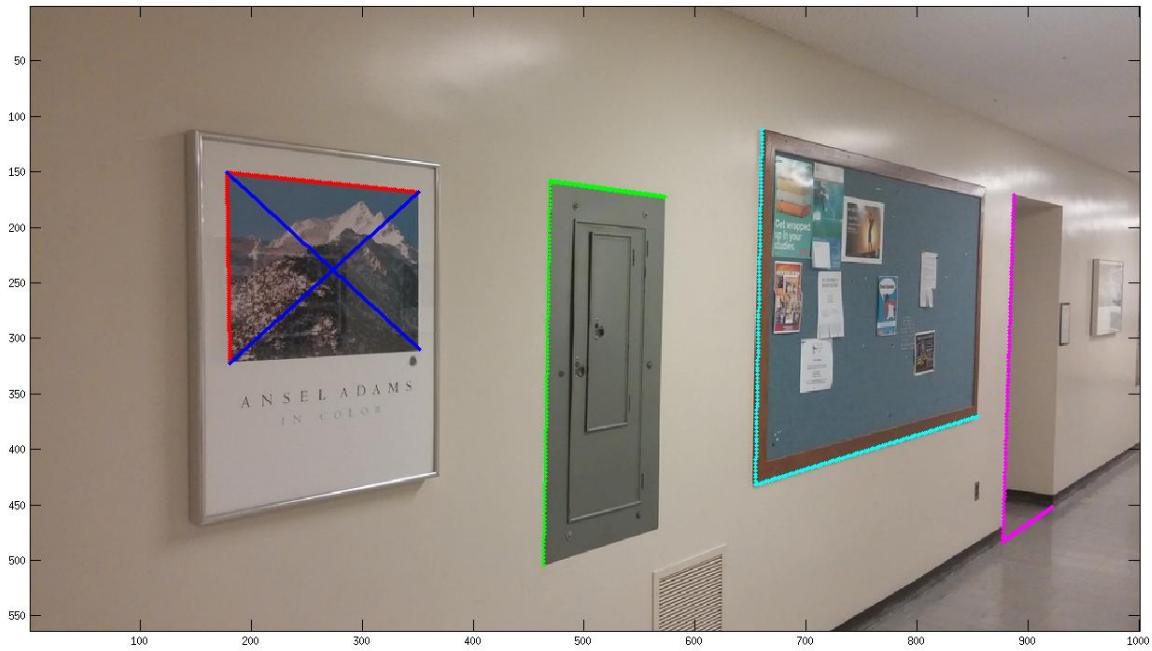

Fig 49. `Set 2 Img 2` Single Step Method, With Orthogonal Lines Selected In Original Image. **Note that the area with diagonal lines crossed is the area we assume to be squared**.

Fig. 50 `Set 2 Img 2` Single Step Method with distortion removed. **Note that THIS IS A GREAT RESULT! The selected area is a 'square' and all lines assumed to be orthogonal are orthogonal after correction!**

Fig 51. `Set 3 Img 1` Single Step Method, With Orthogonal Lines Selected In Original Image. **Note that the area with diagonal lines crossed is the area we assume to be squared**.

Fig. 52 `Set 3 Img 1` Single Step Method with distortion removed. **Note that as the 'squared' area selected from previous image is not a perfect square(it was roughly estimated on Micky Mouse Portrait), the result is subject to relatively noticeable distortion**

Fig 53. `Set 3 Img 2` Single Step Method, With Orthogonal Lines Selected In Original Image. **Note that the area with diagonal lines crossed is the area we assume to be squared**.

Fig. 54 `Set 3 Img 2` Single Step Method with distortion removed. **Note that THIS IS A GREAT RESULT! The selected area is a 'square' and all lines assumed to be orthogonal are orthogonal after correction!**

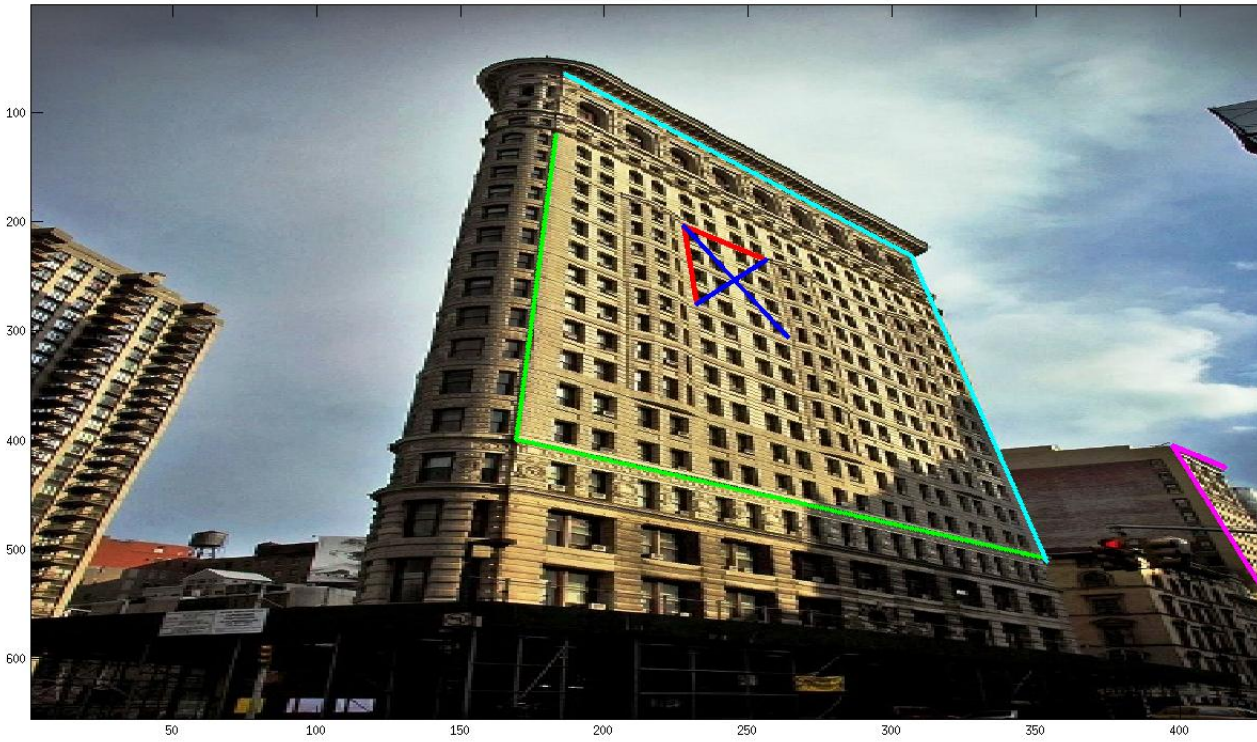

Fig 55. `Set 4 Img 1` Single Step Method, With Orthogonal Lines Selected In Original Image. **Note that the area with diagonal lines crossed is the area we assume to be squared**.

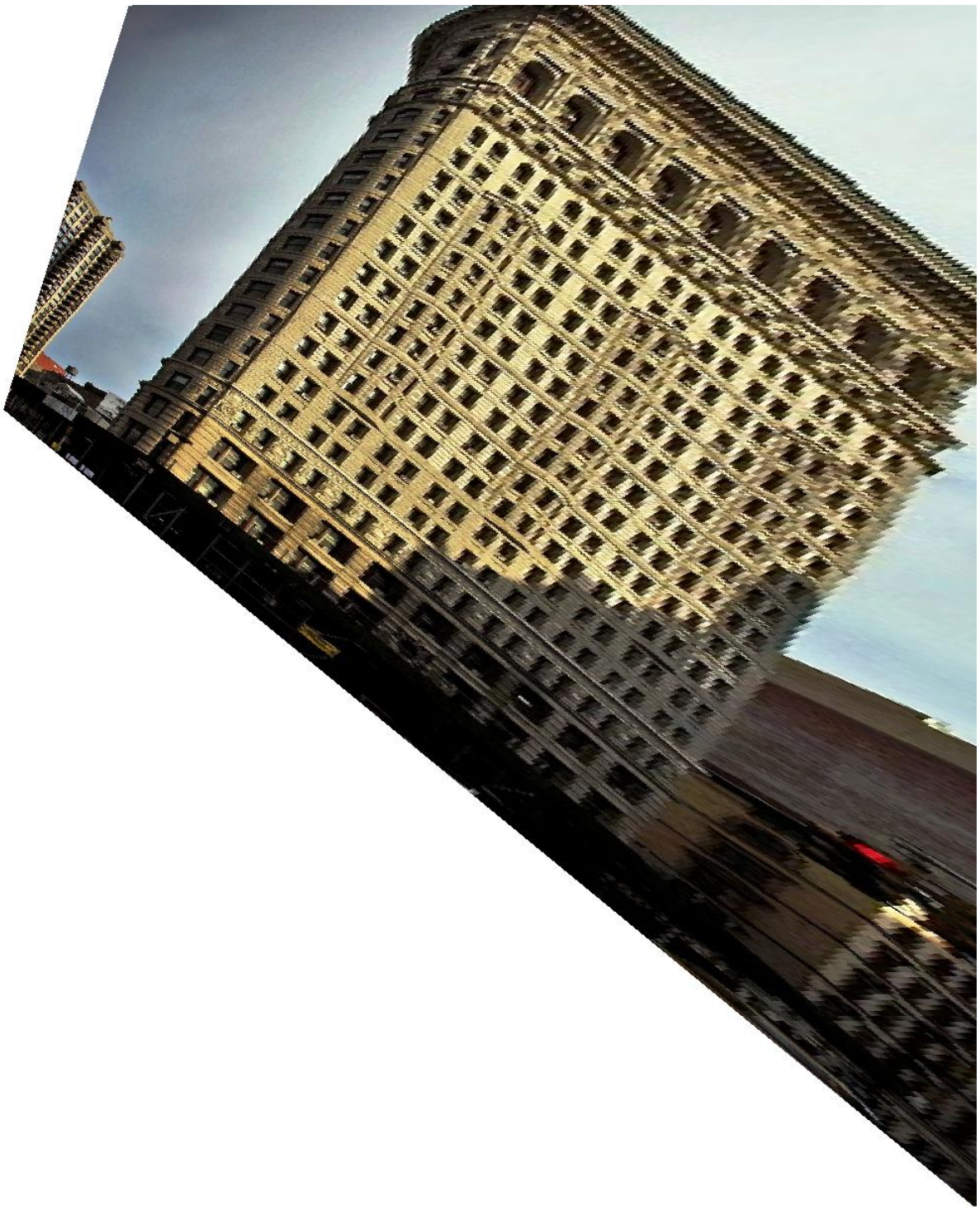

Fig. 56 `Set 4 Img 1` Single Step Method with distortion removed. **Note that THIS IS A GREAT RESULT! The selected area is a 'square' and all lines assumed to be orthogonal are orthogonal after correction!**

Fig 57. `Set 4 Img 2` Single Step Method, With Orthogonal Lines Selected In Original Image. **Note that the area with diagonal lines crossed is the area we assume to be squared**.

Fig. 58 `Set 4 Img 2` Single Step Method with distortion removed. **Note that THIS IS A GREAT RESULT! The selected area is a 'square' and all lines assumed to be orthogonal are orthogonal after correction!**

Fig 59. `My Own Image1` Single Step Method, With Orthogonal Lines Selected In Original Image. **Note that the area with diagonal lines crossed is the area we assume to be squared**.

Fig. 60 `My Own Image1` Single Step Method with distortion removed. **Note that THIS IS A GREAT RESULT! The selected area is a 'square' and all lines assumed to be orthogonal are orthogonal after correction!**

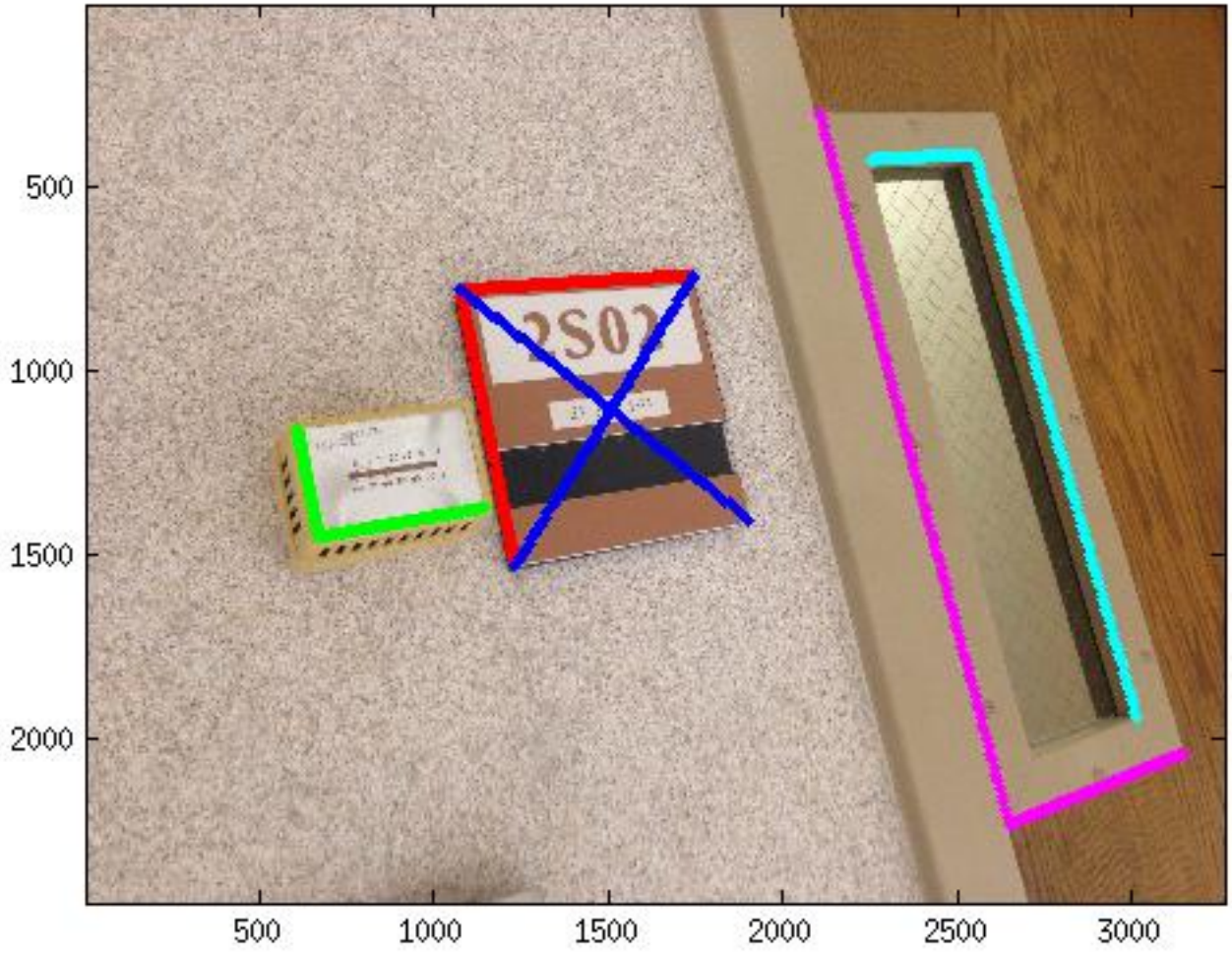

Fig 61. `My Own Image2` Single Step Method, With Orthogonal Lines Selected In Original Image. **Note that the area with diagonal lines crossed is the area we assume to be squared**.

Fig. 62 `My Own Image2` Single Step Method with distortion removed. **Note that THIS IS A GREAT RESULT! The selected area is a 'square' and all lines assumed to be orthogonal are orthogonal after correction!**

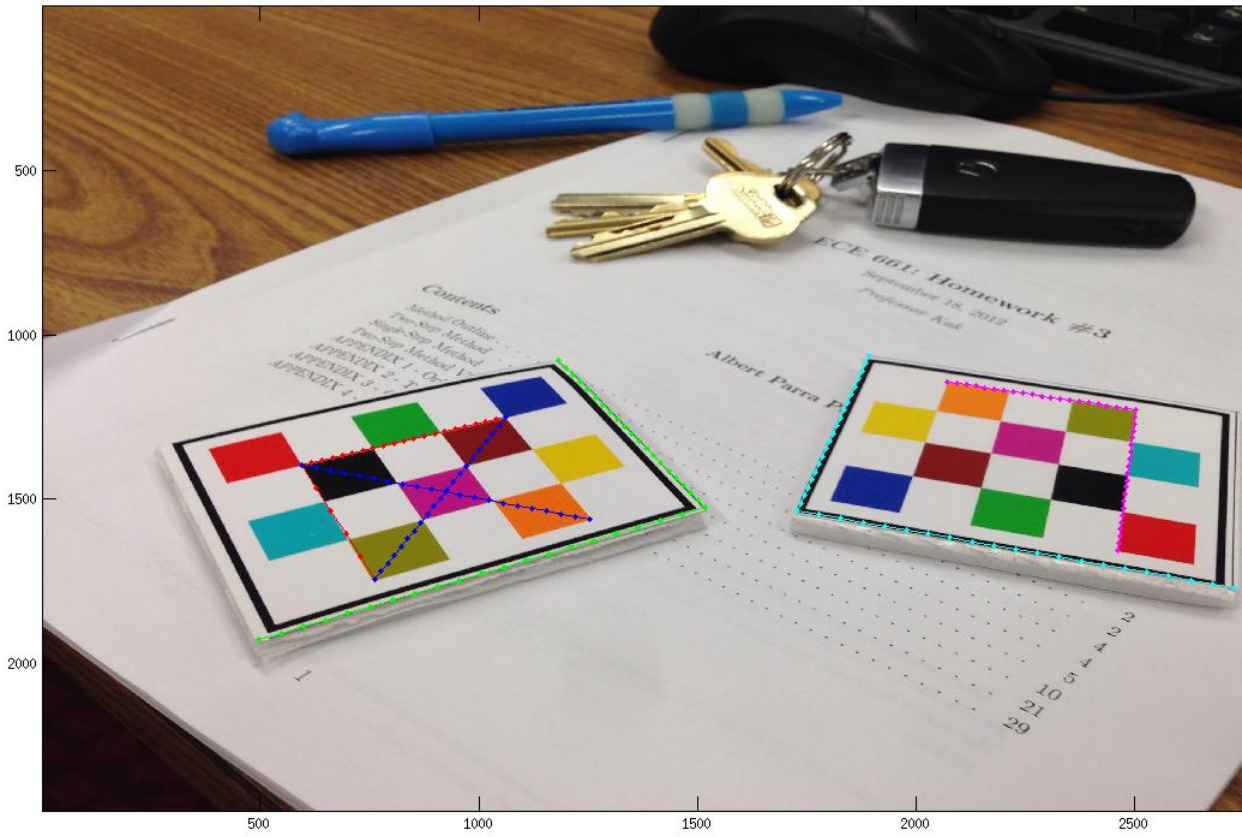

Fig 63. `My Own Image3` Single Step Method, With Orthogonal Lines Selected In Original Image. **Note that the area with diagonal lines crossed is the area we assume to be squared**.

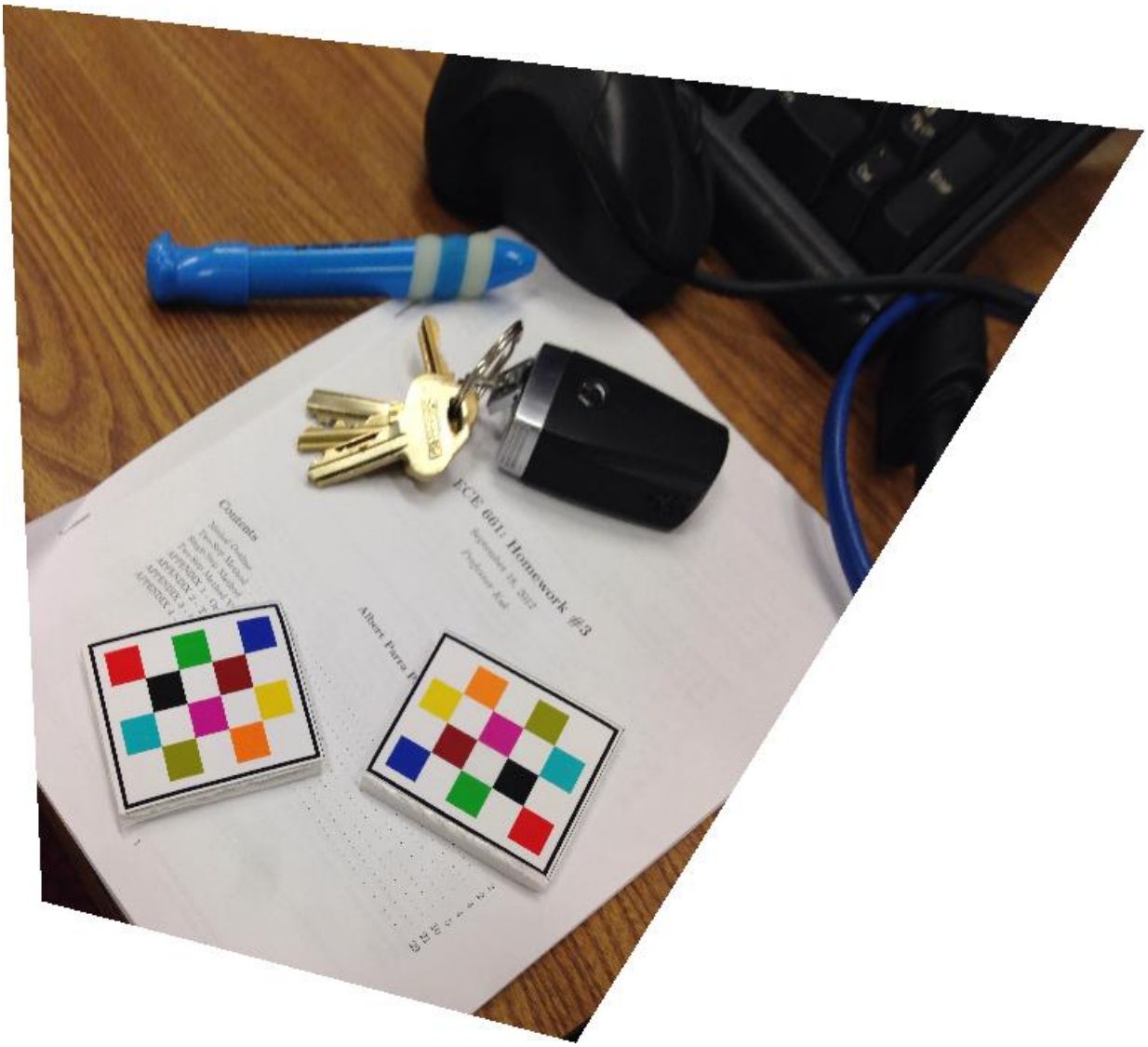

Fig. 64 `My Own Image3` Single Step Method with distortion removed. **Note that THIS IS A GREAT RESULT! The selected area is a 'square' and all lines assumed to be orthogonal are orthogonal after correction!**

END OF RESULTS!

7 Matlab Code For Two-Step Method

```
clc; close all; clear all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% First, move to the targeted folder
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Img = imread('Img_Name.jpg');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 1: Remove The Projective Distortion
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure
image(Img)
truesize
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Get the coordinates we need to remove projective distortion
% The order of picking the points could not be changed otherwise the system
% will not work.
% Order: Upper Left(R) -> Upper Right(G) -> Lower Left(B) -> Lower Right(C)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[x,y] = ginput(4);
x1 = y(1);
y1 = x(1);
x2 = y(2);
y2 = x(2);
x3 = y(3);
y3 = x(3);
x4 = y(4);
y4 = x(4);
hold on;
x = [x1:(x2 - x1)/100:x2];
y = [y1:(y2 - y1)/100:y2];
hold on
plot(y,x, 'r.-')
x = [x1:(x3 - x1)/100:x3];
y = [y1:(y3 - y1)/100:y3];
hold on
plot(y,x, 'r.-')


x = [x1:(x4 - x1)/100:x4];
y = [y1:(y4 - y1)/100:y4];
hold on
plot(y,x, 'b.-')
x = [x3:(x2 - x3)/100:x2];
y = [y3:(y2 - y3)/100:y2];
hold on
plot(y,x, 'b.-')
a = 5;  %%% This is a scaler, with a scaler a lot of time could be saved!
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Size_Img = size(Img);
```

```
51
52 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
53 % Find the vanishing line in the image plane so we can construct the
      homography
54 % that removes the projective distortion.
55 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
56
57 l1 = cross([x1;y1;1],[x2;y2;1]);
58 l2 = cross([x3;y3;1],[x4;y4;1]);
59 l3 = cross([x1;y1;1],[x3;y3;1]);
60 l4 = cross([x2;y2;1],[x4;y4;1]);
61 P = cross(l1,l2);
62 Q = cross(l3,l4);
63 L_VL = cross(P,Q);
64 L_VL(1) = L_VL(1)/L_VL(3);
65 L_VL(2) = L_VL(2)/L_VL(3);
66 L_VL(3) = L_VL(3)/L_VL(3);
67 H_projective = [1,0,0;0,1,0;a*L_VL'];
68
69 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
70 % Compute the image size required when the pixel coordinates are
71 % projected back into world plane.
72 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
73
74 xw1 = 1;
75 yw1 = 1;
76 xw2 = 1;
77 yw2 = Size_Img(2);
78 xw3 = Size_Img(1);
79 yw3 = 1;
80 xw4 = Size_Img(1);
81 yw4 = Size_Img(2);
82
83 pw1 = (H_projective)*[xw1;yw1;1];
84 pw2 = (H_projective)*[xw2;yw2;1];
85 pw3 = (H_projective)*[xw3;yw3;1];
86 pw4 = (H_projective)*[xw4;yw4;1];
87
88 pw1 = pw1./pw1(3);
89 pw2 = pw2./pw2(3);
90 pw3 = pw3./pw3(3);
91 pw4 = pw4./pw4(3);
92
93 New_Size(1) = round(max([pw1(1),pw2(1),pw3(1),pw4(1)]) + ...
94     abs(min([pw1(1),pw2(1),pw3(1),pw4(1)])) );
95
96 New_Size(2) = round(max([pw1(2),pw2(2),pw3(2),pw4(2)]) + ...
97     abs(min([pw1(2),pw2(2),pw3(2),pw4(2)])) )
98
99 New_Img(1:New_Size(1),1:New_Size(2),1:3) = 255;
100
101 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
102 % Project the pixel coordinates back to the real world plane. And Save
103 % The Results.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for m = 1:1:New_Size(1)
    m
    for n = 1:1:New_Size(2)
        p = inv(H_projective)*[m;n;1];
        p = round(p/p(3)) + [1;1;0];
        if((p(1)<Size_Img(1))&&(p(2)<Size_Img(2))&&(p(1)>0)&&(p(2)>0))
            New_Img(m,n,:) = Img(p(1),p(2),:);
        else
        end
    end
end
New_Img = uint8(New_Img);

imwrite(New_Img,'SetName_ImgName_Projective.jpg','jpeg')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 2: Remove The Affine Distortion
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure
image(New_Img)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% First, we need to selected a squared area from the figure.
% Order: Upper Left(R) -> Upper Right(G) -> Lower Left(B) -> Lower Right(C)
% And plot the diagonal lines on the square.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[x,y] = ginput(4);
x1 = y(1);
y1 = x(1);
x2 = y(2);
y2 = x(2);
x3 = y(3);
y3 = x(3);
x4 = y(4);
y4 = x(4);
hold on;
x = [x1:(x2 - x1)/10:x2];
y = [y1:(y2 - y1)/10:y2];
hold on
plot(y,x, 'r.-')
x = [x1:(x3 - x1)/10:x3];
y = [y1:(y3 - y1)/10:y3];
hold on
plot(y,x, 'r.-')

x = [x1:(x4 - x1)/10:x4];
y = [y1:(y4 - y1)/10:y4];
hold on
plot(y,x, 'b.-')
x = [x3:(x2 - x3)/10:x2];
y = [y3:(y2 - y3)/10:y2];
hold on
plot(y,x, 'b.-')
```

```
158
159 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
160 % Find the Degeneric Conic Matrix, using SVD to compute for H_affine
161 % Accordingly.
162
163 l1 = cross([x1;y1;1],[x2;y2;1]);
164 m1 = cross([x1;y1;1],[x3;y3;1]);
165 l2 = cross([x1;y1;1],[x4;y4;1]);
166 m2 = cross([x3;y3;1],[x2;y2;1]);
167 M = [l1(1)*m1(1), (l1(1)*m1(2) + l1(2)*m1(1)); l2(1)*m2(1), ...
168     (l2(1)*m2(2) + l2(2)*m2(1))];
169 b = [-l1(2)*m1(2); -l2(2)*m2(2)];
170 s = inv(M)*b
171 s11 = s(1);
172 s12 = s(2);
173 S = [s11,s12; s12,1];
174 [U,S,V] = svd(S);
175 A = U*(S^(1/2))*V;
176 H_affine = [A(1,1),A(1,2),0; A(2,1),A(2,2),0; 0,0,1];
177
178 %%%%%%%% Overall Homography%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
179 H = inv(H_affine)*H_projective;
180 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
181 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
182 % Compute the image size required when the pixel coordinates are
183 % projected back into world plane.
184 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
185
186 xw1 = 1;
187 yw1 = 1;
188 xw2 = 1;
189 yw2 = Size_Img(2);
190 xw3 = Size_Img(1);
191 yw3 = 1;
192 xw4 = Size_Img(1);
193 yw4 = Size_Img(2);
194
195 pw1 = (H)*[xw1;yw1;1]
196 pw2 = (H)*[xw2;yw2;1]
197 pw3 = (H)*[xw3;yw3;1]
198 pw4 = (H)*[xw4;yw4;1]
199
200 pw1 = pw1./pw1(3);
201 pw2 = pw2./pw2(3);
202 pw3 = pw3./pw3(3);
203 pw4 = pw4./pw4(3);
204
205 New_Size(1) = round(max([pw1(1),pw2(1),pw3(1),pw4(1)]) + ...
206     abs(min([pw1(1),pw2(1),pw3(1),pw4(1)])) );
207
208 New_Size(2) = round(max([pw1(2),pw2(2),pw3(2),pw4(2)]) + ...
209     abs(min([pw1(2),pw2(2),pw3(2),pw4(2)])) )
210 clear New_Img
211 New_Img(1:New_Size(1),1:New_Size(2),1:3) = 255;
```

```
212
213 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
214 % Project the pixel coordinates back to the real world plane. And Save
215 % The Results.
216 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
217
218 for m = 1:1:New_Size(1)
219     m
220     for n = 1:1:New_Size(2)
221         p = inv(H)*[m;n;1];
222         p = round(p/p(3)) + [1;1;0];
223         if((p(1)<Size_Img(1))&&(p(2)<Size_Img(2))&&(p(1)>0)&&(p(2)>0))
224             New_Img(m,n,:) = Img(p(1),p(2),:);
225         else
226         end
227     end
228 end
229 New_Img = uint8(New_Img);
230 figure
231 image(New_Img)
232 imwrite(New_Img,'Set_Name_Img_Name_Projective_Affine.jpg','jpeg')
233
234
235
236 %%%%%% Process For Img2.jpg, This is  the same step       %%%%%%%%%%%%%%%%%
237 %%%%%% all over again. Please refer to the first half of the code. %%%%%%%%
238 %%%%%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
239 clc; clear all;
240
241 Img = imread('Img2.jpg');
242 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
243 % Step 1: Remove The Projective Distortion
244 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
245
246 figure
247 image(Img)
248 truesize
249 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
250 % Get the coordinates we need to remove projective distortion
251 % The order of picking the points could not be changed otherwise the system
252 % will not work.
253 % Order: Upper Left(R) -> Upper Right(G) -> Lower Left(B) -> Lower Right(C)
254 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
255 [x,y] = ginput(4);
256 x1 = y(1);
257 y1 = x(1);
258 x2 = y(2);
259 y2 = x(2);
260 x3 = y(3);
261 y3 = x(3);
262 x4 = y(4);
263 y4 = x(4);
264 hold on;
265 x = [x1:(x2 - x1)/100:x2];
```

```
y = [y1:(y2 - y1)/100:y2];
hold on
plot(y,x, 'r.-')
x = [x1:(x3 - x1)/100:x3];
y = [y1:(y3 - y1)/100:y3];
hold on
plot(y,x, 'r.-')


x = [x1:(x4 - x1)/100:x4];
y = [y1:(y4 - y1)/100:y4];
hold on
plot(y,x, 'b.-')
x = [x3:(x2 - x3)/100:x2];
y = [y3:(y2 - y3)/100:y2];
hold on
plot(y,x, 'b.-')
a = 0.3;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Size_Img = size(Img);

l1 = cross([x1;y1;1],[x2;y2;1]);
l2 = cross([x3;y3;1],[x4;y4;1]);
l3 = cross([x1;y1;1],[x3;y3;1]);
l4 = cross([x2;y2;1],[x4;y4;1]);
P = cross(l1,l2);
Q = cross(l3,l4);
L_VL = cross(P,Q);
L_VL(1) = L_VL(1)/L_VL(3);
L_VL(2) = L_VL(2)/L_VL(3);
L_VL(3) = L_VL(3)/L_VL(3);
H_projective = [1,0,0;0,1,0;a*L_VL'];
xw1 = 1;
yw1 = 1;
xw2 = 1;
yw2 = Size_Img(2);
xw3 = Size_Img(1);
yw3 = 1;
xw4 = Size_Img(1);
yw4 = Size_Img(2);

pw1 = (H_projective)*[xw1;yw1;1];
pw2 = (H_projective)*[xw2;yw2;1];
pw3 = (H_projective)*[xw3;yw3;1];
pw4 = (H_projective)*[xw4;yw4;1];

pw1 = pw1./pw1(3);
pw2 = pw2./pw2(3);
pw3 = pw3./pw3(3);
pw4 = pw4./pw4(3);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% p1 = (H)*[x1;y1;1]
% p2 = (H)*[x2;y2;1]
```

```
% p3 = (H)*[x3;y3;1]
% p4 = (H)*[x4;y4;1]
%
% p1 = p1./p1(3)
% p2 = p2./p2(3)
% p3 = p3./p3(3)
% p4 = p4./p4(3)
% p1 - p2
% p3 - p4
% p1 - p3
% p2 - p4

New_Size(1) = round(max([pw1(1),pw2(1),pw3(1),pw4(1)]) + ...
    abs(min([pw1(1),pw2(1),pw3(1),pw4(1)])) );

New_Size(2) = round(max([pw1(2),pw2(2),pw3(2),pw4(2)]) + ...
    abs(min([pw1(2),pw2(2),pw3(2),pw4(2)])) )

New_Img(1:New_Size(1),1:New_Size(2),1:3) = 255;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for m = 1:1:New_Size(1)
    m
    for n = 1:1:New_Size(2)
        p = inv(H_projective)*[m;n;1];
        p = round(p/p(3)) + [1;1;0];
        if((p(1)<Size_Img(1))&&(p(2)<Size_Img(2))&&(p(1)>0)&&(p(2)>0))
            New_Img(m,n,:) = Img(p(1),p(2),:);
        else
        end
    end
end
New_Img = uint8(New_Img);

imwrite(New_Img,'Set1_Img2_Projective.jpg','jpeg')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 2: Remove The Affine Distortion
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure
image(New_Img)
[x,y] = ginput(4);
x1 = y(1);
y1 = x(1);
x2 = y(2);
y2 = x(2);
x3 = y(3);
y3 = x(3);
x4 = y(4);
y4 = x(4);
hold on;
x = [x1:(x2 - x1)/10:x2];
y = [y1:(y2 - y1)/10:y2];
hold on
plot(y,x, 'r.-')
```

```
x = [x1:(x3 - x1)/10:x3];
y = [y1:(y3 - y1)/10:y3];
hold on
plot(y,x, 'r.-')


x = [x1:(x4 - x1)/100:x4];
y = [y1:(y4 - y1)/100:y4];
hold on
plot(y,x, 'b.-')
x = [x3:(x2 - x3)/100:x2];
y = [y3:(y2 - y3)/100:y2];
hold on
plot(y,x, 'b.-')

l1 = cross([x1;y1;1],[x2;y2;1]);
m1 = cross([x1;y1;1],[x3;y3;1]);
l2 = cross([x1;y1;1],[x4;y4;1]);
m2 = cross([x3;y3;1],[x2;y2;1]);
M = [l1(1)*m1(1), (l1(1)*m1(2) + l1(2)*m1(1)); l2(1)*m2(1), ...
    (l2(1)*m2(2) + l2(2)*m2(1))];
b = [-l1(2)*m1(2); -l2(2)*m2(2)];
s = inv(M)*b
s11 = s(1);
s12 = s(2);
S = [s11,s12; s12,1];
[U,S,V] = svd(S);
A = U*(S^(1/2))*V;
H_affine = [A(1,1),A(1,2),0; A(2,1),A(2,2),0; 0,0,1];

H = inv(H_affine)*H_projective;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


xw1 = 1;
yw1 = 1;
xw2 = 1;
yw2 = Size_Img(2);
xw3 = Size_Img(1);
yw3 = 1;
xw4 = Size_Img(1);
yw4 = Size_Img(2);

pw1 = (H)*[xw1;yw1;1]
pw2 = (H)*[xw2;yw2;1]
pw3 = (H)*[xw3;yw3;1]
pw4 = (H)*[xw4;yw4;1]

pw1 = pw1./pw1(3);
pw2 = pw2./pw2(3);
pw3 = pw3./pw3(3);
pw4 = pw4./pw4(3);

```

```
New_Size(1) = round(max([pw1(1),pw2(1),pw3(1),pw4(1)]) + ...
    abs(min([pw1(1),pw2(1),pw3(1),pw4(1)])) );

New_Size(2) = round(max([pw1(2),pw2(2),pw3(2),pw4(2)]) + ...
    abs(min([pw1(2),pw2(2),pw3(2),pw4(2)])) )
clear New_Img
New_Img(1:New_Size(1),1:New_Size(2),1:3) = 255;

for m = 1:1:New_Size(1)
    m
    for n = 1:1:New_Size(2)
        p = inv(H)*[m;n;1];
        p = round(p/p(3)) + [1;1;0];
        if((p(1)<Size_Img(1))&&(p(2)<Size_Img(2))&&(p(1)>0)&&(p(2)>0))
            New_Img(m,n,:) = Img(p(1),p(2),:);
        else
        end
    end
end
New_Img = uint8(New_Img);
figure
image(New_Img)
truesize
imwrite(New_Img,'Set1_Img2_Projective_Affine.jpg','jpeg')
```

8 Matlab Code For Single-Step Method

```
clc; close all; clear all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% First, move to the targeted folder
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
scaler = 50;
Img = imread('Image_Name.jpg');
Size_Img = size(Img);
a = 1;
R = 1;
figure
image(Img)
[x,y] = ginput(13);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Get the 13 coordinates we need to remove projective distortion
% The order of picking the points could not be changed otherwise the system
% will not work.
% Order: Upper Left(R) -> Upper Right(G) -> Lower Left(B) -> Lower Right(C)
% Why 13 points.
% Squared: 4 points, Three Pairs of Orthogonal Lines: 3 points * 3 = 9
% 9 + 4 = 13. For more details refer to the image in single step method.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x1 = y(1);
y1 = x(1);
x2 = y(2);
y2 = x(2);
x3 = y(3);
y3 = x(3);
x4 = y(4);
y4 = x(4);
x5 = y(5);
y5 = x(5);
x6 = y(6);
y6 = x(6);
x7 = y(7);
y7 = x(7);
x8 = y(8);
y8 = x(8);
x9 = y(9);
y9 = x(9);
x10 = y(10);
y10 = x(10);
x11 = y(11);
y11 = x(11);
x12 = y(12);
y12 = x(12);
x13 = y(13);
y13 = x(13);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Mark those pairs of orthogonal lines used to compute Homography on
% original image.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x = [x1:(x2 - x1)/20:x2];
y = [y1:(y2 - y1)/20:y2];
hold on
plot(y,x, 'r.-')
x = [x1:(x3 - x1)/5:x3];
y = [y1:(y3 - y1)/5:y3];
hold on
plot(y,x, 'r.-')



x = [x1:(x4 - x1)/20:x4];
y = [y1:(y4 - y1)/20:y4];
hold on
plot(y,x, 'b.-')
x = [x3:(x2 - x3)/20:x2];
y = [y3:(y2 - y3)/20:y2];
hold on
plot(y,x, 'b.-')


x = [x5:(x6 - x5)/20:x6];
y = [y5:(y6 - y5)/20:y6];
hold on
plot(y,x, 'g.-')
x = [x5:(x7 - x5)/20:x7];
y = [y5:(y7 - y5)/20:y7];
hold on
plot(y,x, 'g.-')

x = [x8:(x9 - x8)/20:x9];
y = [y8:(y9 - y8)/20:y9];
hold on
plot(y,x, 'c.-')
x = [x8:(x10 - x8)/20:x10];
y = [y8:(y10 - y8)/20:y10];
hold on
plot(y,x, 'c.-')

x = [x11:(x12 - x11)/20:x12];
y = [y11:(y12 - y11)/20:y12];
hold on
plot(y,x, 'm.-')
x = [x11:(x13 - x11)/20:x13];
y = [y11:(y13 - y11)/20:y13];
hold on
plot(y,x, 'm.-')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Find all pairs of orthogonal lines by finding the cross product of points
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
l1 = cross([x1;y1;R],[x2;y2;R]);
m1 = cross([x1;y1;R],[x3;y3;R]);
l2 = cross([x1;y1;R],[x4;y4;R]);
m2 = cross([x2;y2;R],[x3;y3;R]);
l3 = cross([x5;y5;R],[x6;y6;R]);
m3 = cross([x5;y5;R],[x7;y7;R]);
l4 = cross([x8;y8;R],[x9;y9;R]);
m4 = cross([x8;y8;R],[x10;y10;R]);
l5 = cross([x11;y11;R],[x12;y12;R]);
m5 = cross([x11;y11;R],[x13;y13;R]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Solve for the degeneric conic matrix. After that using SVD to find the
% Homography matrix accordingly.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
M = [l1(1)*m1(1), (l1(2)*m1(1) + l1(1)*m1(2))/2, l1(2)*m1(2), ...
     (l1(1)*m1(3) + l1(3)*m1(1))/2 , (l1(2)*m1(3) + l1(3)*m1(2))/2;
     l2(1)*m2(1), (l2(2)*m2(1) + l2(1)*m2(2))/2, l2(2)*m2(2), ...
     (l2(1)*m2(3) + l2(3)*m2(1))/2 , (l2(2)*m2(3) + l2(3)*m2(2))/2;
     l3(1)*m3(1), (l3(2)*m3(1) + l3(1)*m3(2))/2, l3(2)*m3(2), ...
     (l3(1)*m3(3) + l3(3)*m3(1))/2 , (l3(2)*m3(3) + l3(3)*m3(2))/2;
     l4(1)*m4(1), (l4(2)*m4(1) + l4(1)*m4(2))/2, l4(2)*m4(2), ...
     (l4(1)*m4(3) + l4(3)*m4(1))/2 , (l4(2)*m4(3) + l4(3)*m4(2))/2;
     l5(1)*m5(1), (l5(2)*m5(1) + l5(1)*m5(2))/2, l5(2)*m5(2), ...
     (l5(1)*m5(3) + l5(3)*m5(1))/2 , (l5(2)*m5(3) + l5(3)*m5(2))/2];

b = [-l1(3)*m1(3);
      -l2(3)*m2(3);
      -l3(3)*m3(3);
      -l4(3)*m4(3);
      -l5(3)*m5(3)];
s = pinv(M)*b;
S = [s(1),s(2)/2;s(2)/2,s(3)];
C = [s(1),s(2)/2,s(4)/2;s(2)/2,s(3),s(5)/2;s(4)/2,s(5)/2,1]
[U1,D1,V1] = svd(C);

S = C(1:2,1:2);
[U,D,V] = svd(S);
A = U*(D^(1/2))*V'
v = inv(A)*[s(4)/2;s(5)/2];

H = [A(1,1),A(1,2),0;
     A(2,1),A(2,2),0;
     v(1), v(2) , 1]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Verify that in image plane transpose(l')*C'*m'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 l1'*C*m1
 l2'*C*m2
 l3'*C*m3
 l4'*C*m4
 l5'*C*m5

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Verify that the desired area to be square is now square
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 pw1 = inv(H)*[x1;y1;R];
 pw2 = inv(H)*[x2;y2;R];
 pw3 = inv(H)*[x3;y3;R];
 pw4 = inv(H)*[x4;y4;R];
 lw1 = cross(pw1,pw2);
 mw1 = cross(pw1,pw3);
 lw2 = cross(pw1,pw4);
 mw2 = cross(pw2,pw3);
 lw1'*mw1
 lw2'*mw2
 pw1 = pw1./(pw1(3))
 pw2 = pw2./(pw2(3))
 pw3 = pw3./(pw3(3))
 pw4 = pw4./(pw4(3))

 pdist([pw1(1),pw1(2);pw2(1),pw2(2)])
 pdist([pw1(1),pw1(2);pw3(1),pw3(2)])
 pdist([pw2(1),pw2(2);pw4(1),pw4(2)])
 pdist([pw3(1),pw3(2);pw4(1),pw4(2)])
 pdist([pw1(1),pw1(2);pw4(1),pw4(2)])
 pdist([pw2(1),pw2(2);pw3(1),pw3(2)])

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute the image size required when the pixel coordinates are
% projected back into world plane.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 pver1 = [1;1;R];
 pver2 = [1;Size_Img(2);R];
 pver3 = [Size_Img(1);1;R];
 pver4 = [Size_Img(1);Size_Img(2);R];
 pwver1 = inv(H)*[1;1;R];
 pwver2 = inv(H)*[1;Size_Img(2);R];
 pwver3 = inv(H)*[Size_Img(1);1;R];
 pwver4 = inv(H)*[Size_Img(1);Size_Img(2);R];
 pwver1 = pwver1/pwver1(3);
 pwver2 = pwver2/pwver2(3);
 pwver3 = pwver3/pwver3(3);
 pwver4 = pwver4/pwver4(3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Verify that our homography matrix is correct by mapping back to image plane
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
p_image1 = H*pwver1
p_image2 = H*pwver2
p_image3 = H*pwver3
p_image4 = H*pwver4
p_image1 = p_image1/p_image1(3)
p_image2 = p_image2/p_image2(3)
p_image3 = p_image3/p_image3(3)
p_image4 = p_image4/p_image4(3)

```

```
213
214 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
215 % Project the pixel coordinates back to the real world plane. And Save
216 % The Results.
217 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
218
219
220 clear New_Img
221
222 range_x = max([pwver1(1),pwver2(1),pwver3(1),pwver4(1)]) - ...
223     min([pwver1(1),pwver2(1),pwver3(1),pwver4(1)])
224 range_y = max([pwver1(2),pwver2(2),pwver3(2),pwver4(2)]) - ...
225     min([pwver1(2),pwver2(2),pwver3(2),pwver4(2)])
226 range_m = round(scaler*range_x)
227 range_n = round(scaler*range_y)
228 New_Img(1:1:range_m,1:1:range_n,1:3) = 255;
229 tx = scaler*min([pwver1(1),pwver2(1),pwver3(1),pwver4(1)]);
230 ty = scaler*min([pwver1(2),pwver2(2),pwver3(2),pwver4(2)]);
231 pnew1 = pwver1(1:2)*scaler - [tx;ty] +[1;1]
232 pnew2 = pwver2(1:2)*scaler - [tx;ty] +[1;1]
233 pnew3 = pwver3(1:2)*scaler - [tx;ty] +[1;1]
234 pnew4 = pwver4(1:2)*scaler - [tx;ty] +[1;1]
235
236
237
238 for i = 1:1:range_m
239     i
240     for j = 1:1:range_n
241         px = i/scaler + (min([pwver1(1),pwver2(1),pwver3(1),pwver4(1)]));
242         py = j/scaler + (min([pwver1(2),pwver2(2),pwver3(2),pwver4(2)]));
243         p_img = H*[px;py;R];
244         p_img = p_img/p_img(3);
245         px_img = round(p_img(1));
246         py_img = round(p_img(2));
247         if ((px_img>1) && (px_img < Size_Img(1)) && (py_img > 1) && (py_img <
                Size_Img(2)))
248             New_Img(i,j,:) = Img(px_img,py_img,:);
249         else
250         end
251     end
252 end
253
254 New_Img = uint8(New_Img);
255 figure
256 image(New_Img)
257 truesize
258 imwrite(New_Img,'Img_Name_Results.jpg','jpeg')
```