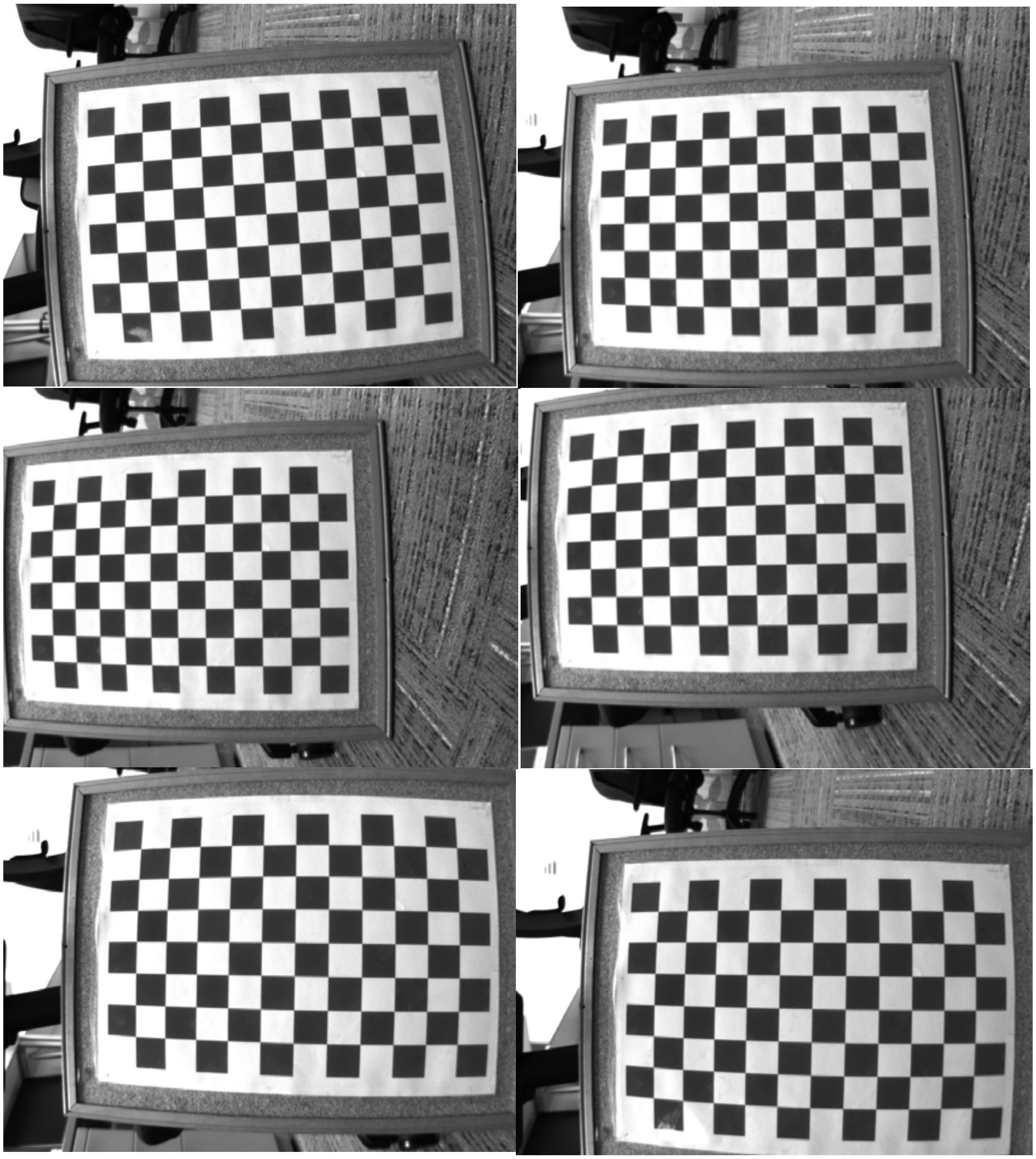
Final Project CIVILEN 6451 ZEYU WANG

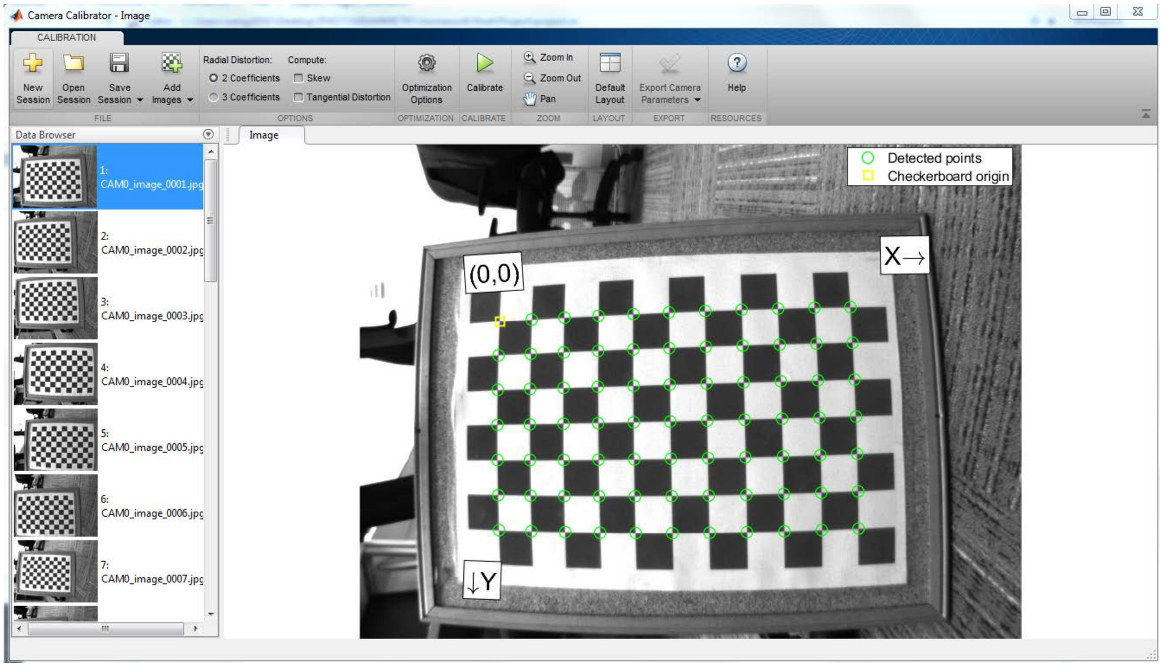
Calibrate the camera and augmented reality:

1. Camera calibration

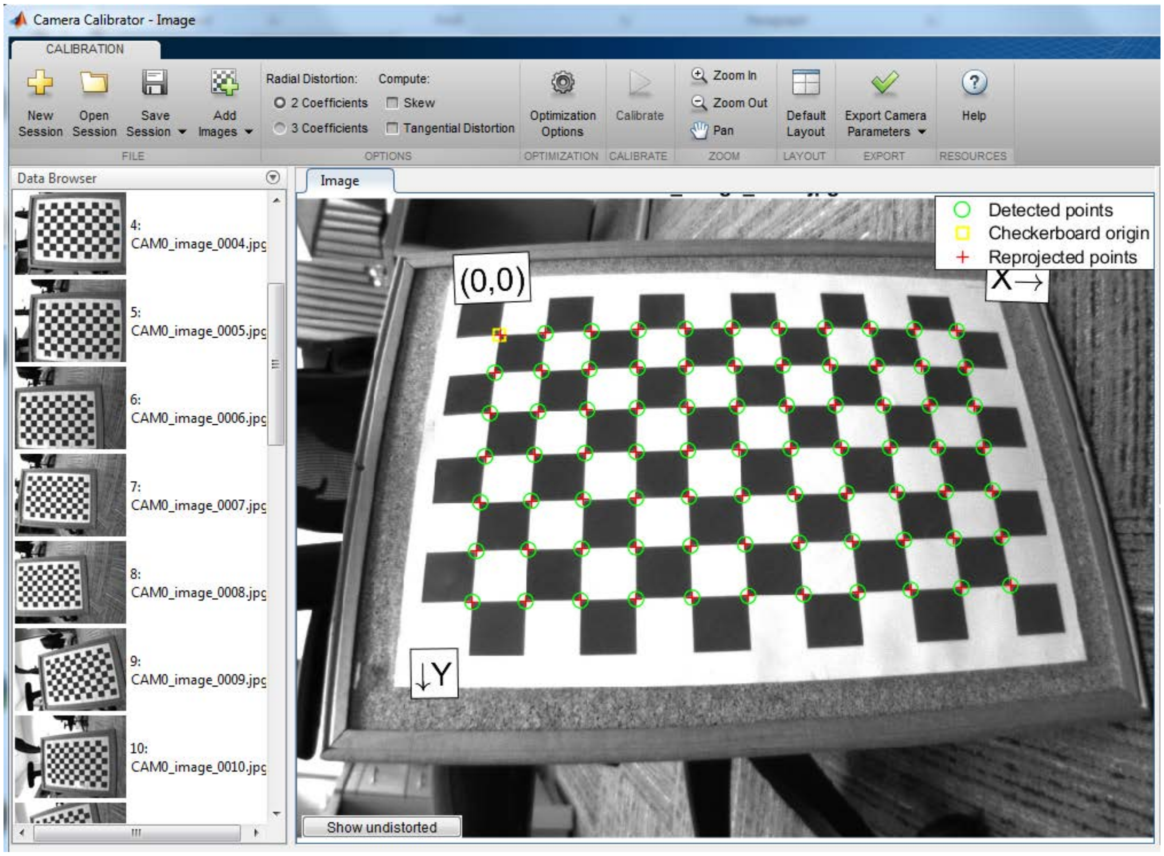
In this step, I used the first provided datasets to calibrate the datasets and found the camera parameters:



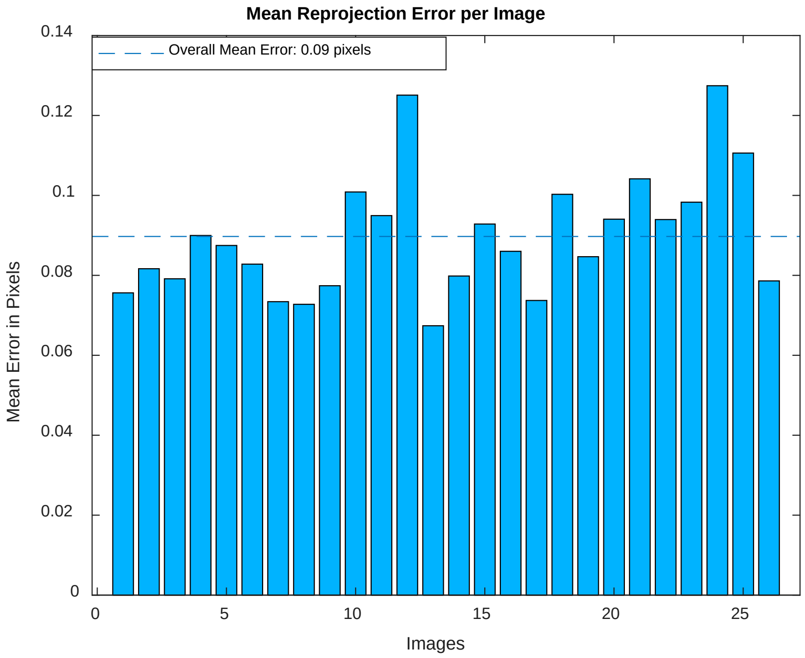
Then I used the camera calibration program in the MATLAB:



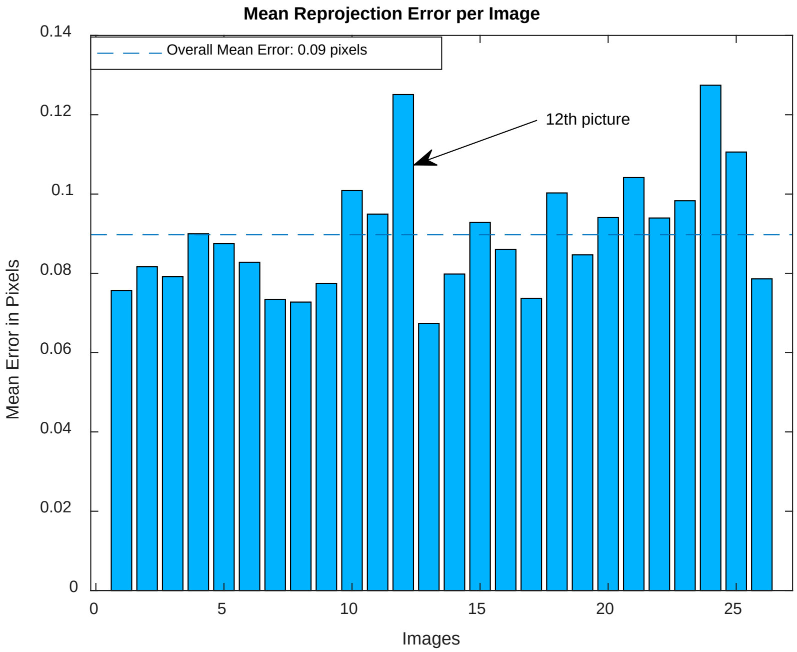
The calibrated framework:

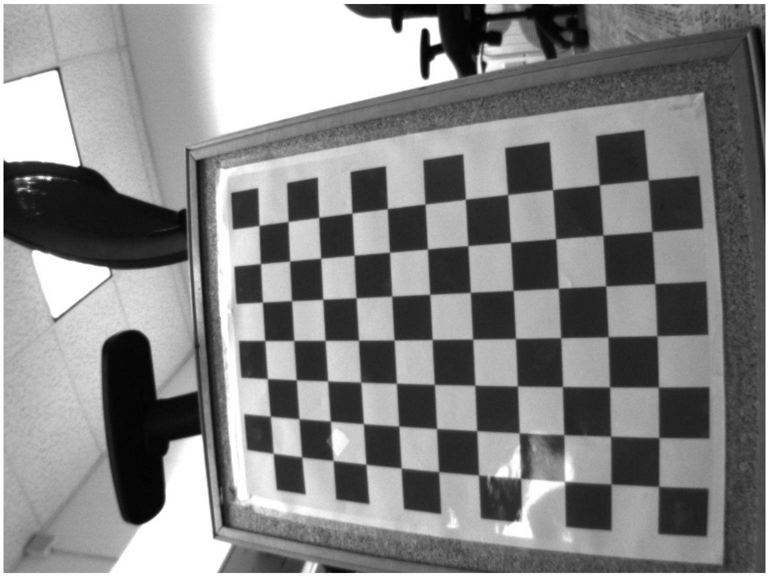


The mean reprojection error for each Image:

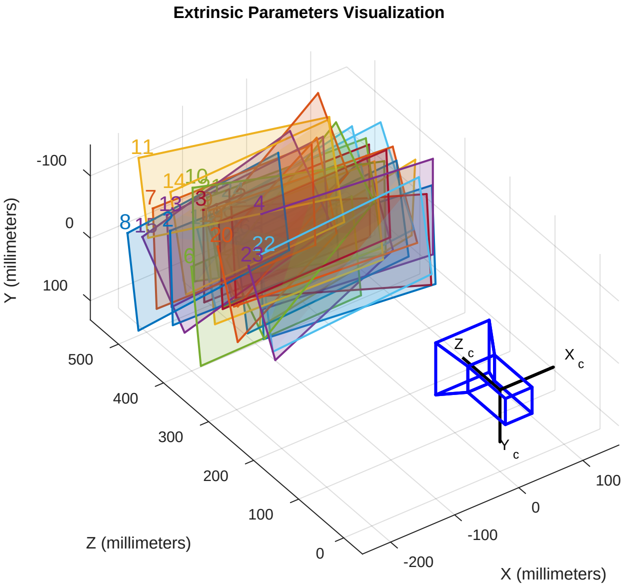


For example, if we pick the picture:





The visualization of the extrinsic parameters for each pictures:



The output of the camera parameters:

```
Standard Errors of Estimated Camera Parameters

                              -----------------------------------------------

Intrinsics
--------------------------------------------------------------------------------------------------------------------
Focal length (pixels):   [  736.2635 +/- 0.2911      737.2685 +/- 0.2885  ]
Principal point (pixels):[  404.4557 +/- 0.2295      295.1041 +/- 0.1549  ]
Radial distortion:       [   -0.3243 +/- 0.0006        0.1573 +/- 0.0020  ]


Extrinsics
--------------------------------------------------------------------------------------------------------------------
Rotation vectors:
                 [   -0.1003 +/- 0.0005       0.0769 +/- 0.0004      -0.0233 +/- 0.0001  ]
                 [   -0.0233 +/- 0.0005       0.0467 +/- 0.0004      -0.0121 +/- 0.0001  ]
                 [    0.0078 +/- 0.0004       0.0523 +/- 0.0004      -0.0162 +/- 0.0001  ]
                 [   -0.0250 +/- 0.0004       0.1082 +/- 0.0004       0.0144 +/- 0.0001  ]
                 [   -0.0809 +/- 0.0004       0.1127 +/- 0.0004       0.0146 +/- 0.0001  ]
                 [   -0.1182 +/- 0.0005       0.0007 +/- 0.0004      -0.0160 +/- 0.0001  ]
                 [   -0.1038 +/- 0.0006      -0.0140 +/- 0.0005       0.0343 +/- 0.0001  ]
                 [   -0.1082 +/- 0.0006      -0.0770 +/- 0.0005      -0.0354 +/- 0.0001  ]
                 [   -0.1563 +/- 0.0004       0.3096 +/- 0.0004      -0.1581 +/- 0.0001  ]
                 [   -0.0059 +/- 0.0004       0.4037 +/- 0.0004      -0.0122 +/- 0.0001  ]
                 [    0.1231 +/- 0.0004       0.4053 +/- 0.0004      -0.1441 +/- 0.0001  ]
                 [    0.0219 +/- 0.0003       0.5573 +/- 0.0003      -0.0499 +/- 0.0001  ]
                 [   -0.0079 +/- 0.0006      -0.0331 +/- 0.0005      -0.0456 +/- 0.0001  ]
                 [   -0.2055 +/- 0.0005      -0.0162 +/- 0.0005      -0.0190 +/- 0.0001  ]
                 [   -0.3278 +/- 0.0004      -0.0437 +/- 0.0004      -0.2171 +/- 0.0001  ]
```

```
                    [    0.0512 +/- 0.0006       0.0713 +/- 0.0005      -0.0737 +/- 0.0001  ]
                    [    0.0335 +/- 0.0006      -0.0178 +/- 0.0005      -0.2209 +/- 0.0001  ]
                    [   -0.1425 +/- 0.0004      -0.3013 +/- 0.0004      -0.3228 +/- 0.0001  ]
                    [   -0.0843 +/- 0.0004      -0.3198 +/- 0.0004      -0.2771 +/- 0.0001  ]
                    [   -0.1615 +/- 0.0004      -0.4229 +/- 0.0004      -0.0789 +/- 0.0001  ]
                    [   -0.3263 +/- 0.0004      -0.0203 +/- 0.0004       0.0603 +/- 0.0001  ]
                    [   -0.1084 +/- 0.0003      -0.0068 +/- 0.0003      -0.0572 +/- 0.0000  ]
                    [   -0.1937 +/- 0.0003      -0.2968 +/- 0.0004      -0.1355 +/- 0.0001  ]
                    [   -0.4312 +/- 0.0003      -0.0672 +/- 0.0004      -0.0403 +/- 0.0001  ]
                    [    0.1957 +/- 0.0004      -0.0733 +/- 0.0004      -0.1103 +/- 0.0001  ]
                    [   -0.0918 +/- 0.0005       0.0130 +/- 0.0004      -0.0629 +/- 0.0001  ]


Translation vectors (millimeters):
                    [ -149.2130 +/- 0.1373      -54.3810 +/- 0.0944      439.7155 +/- 0.1835  ]
                    [ -208.9597 +/- 0.1367      -75.8970 +/- 0.0964      435.0173 +/- 0.1917  ]
                    [ -176.1227 +/- 0.1294     -111.5422 +/- 0.0895      408.3160 +/- 0.1801  ]
                    [ -105.5126 +/- 0.1197      -92.0603 +/- 0.0807      379.5525 +/- 0.1582  ]
                    [ -102.4676 +/- 0.1208      -43.9154 +/- 0.0820      386.7757 +/- 0.1584  ]
                    [ -193.3730 +/- 0.1285      -26.9014 +/- 0.0909      409.9630 +/- 0.1789  ]
                    [ -197.9707 +/- 0.1534      -73.4516 +/- 0.1070      488.9410 +/- 0.2120  ]
                    [ -243.8083 +/- 0.1514      -59.3659 +/- 0.1077      480.1239 +/- 0.2166  ]
                    [ -106.1338 +/- 0.1532      -27.0256 +/- 0.1041      494.0212 +/- 0.1818  ]
                    [ -109.5873 +/- 0.1631      -44.6262 +/- 0.1113      526.4086 +/- 0.1904  ]
                    [ -178.1718 +/- 0.1730     -108.5095 +/- 0.1185      548.9422 +/- 0.2121  ]
                    [  -92.2188 +/- 0.1436      -44.3335 +/- 0.0978      465.2959 +/- 0.1536  ]
                    [ -199.5236 +/- 0.1438      -84.1001 +/- 0.1001      455.8670 +/- 0.2033  ]
                    [ -169.8490 +/- 0.1538      -86.7597 +/- 0.1059      490.0718 +/- 0.2053  ]
                    [ -218.1140 +/- 0.1514      -39.9547 +/- 0.1070      483.5833 +/- 0.2041  ]
                    [ -103.8273 +/- 0.1491      -18.4949 +/- 0.1013      476.6838 +/- 0.2015  ]
                    [ -170.8315 +/- 0.1356     -66.8104 +/- 0.0932     429.1202 +/- 0.1883  ]
                    [ -142.0920 +/- 0.1284     -19.3127 +/- 0.0873     405.1125 +/- 0.1734  ]
                    [ -171.0435 +/- 0.1297     -85.6485 +/- 0.0882     404.5159 +/- 0.1821  ]
                    [ -182.5285 +/- 0.1194     -81.6461 +/- 0.0816     366.9050 +/- 0.1663  ]
                    [ -142.1107 +/- 0.1405     -91.9970 +/- 0.0963     448.1258 +/- 0.1808  ]
                    [ -141.3167 +/- 0.1048     -71.7012 +/- 0.0718     331.7695 +/- 0.1423  ]
                    [ -158.6241 +/- 0.1065     -61.1602 +/- 0.0731     332.7586 +/- 0.1443  ]
                    [ -130.5395 +/- 0.1311    -101.1634 +/- 0.0892     415.6192 +/- 0.1658  ]
                    [ -120.9955 +/- 0.1120     -55.2924 +/- 0.0764     353.1335 +/- 0.1533  ]
                    [ -132.6247 +/- 0.1252     -54.2858 +/- 0.0853     398.6261 +/- 0.1682  ]
```

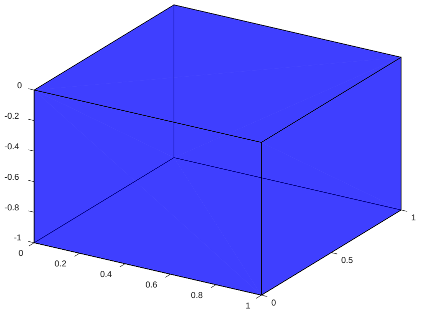
All the parameters are stored in the cameraParams.

2. Augmented Reality

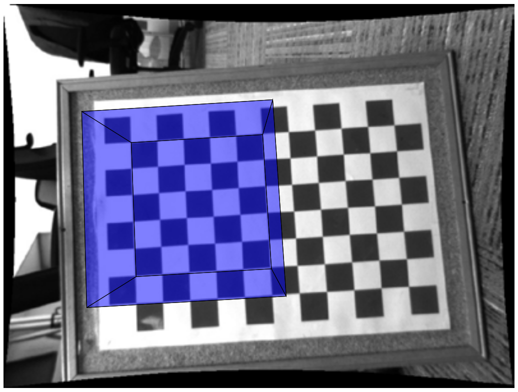
In this section, two objects are displayed in the figure: 1) A simple geometric cubic 2) a complex 3D teapot. The steps of Augmented Reality can be summarized as below:

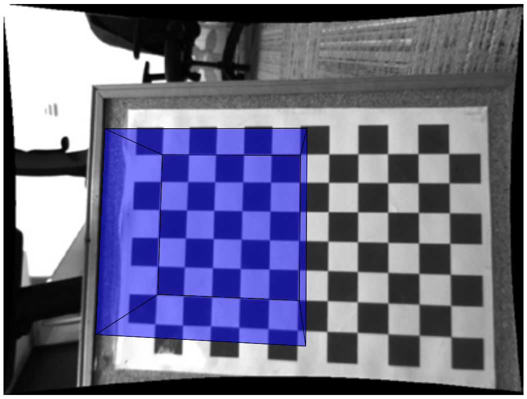
- Read the Image
- Eliminate the distortion of picture
- Find the fiducial marker in the Image
- Generate the checkerboard in real world
- Generate the extrinsics vector (Rotation and translation)
- Find the camera matrix form the rotation and translation vector
- Show the virtual object in the picture

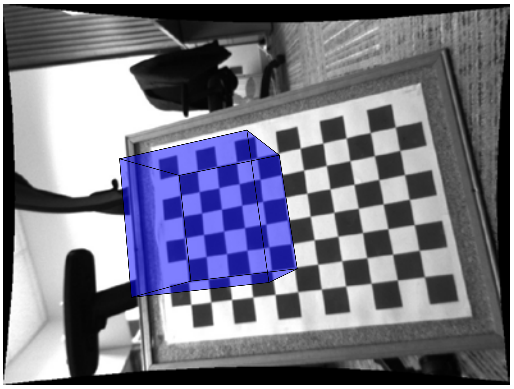
1). A simple geometric cubic

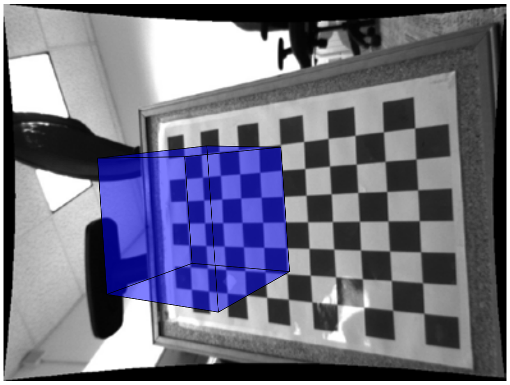


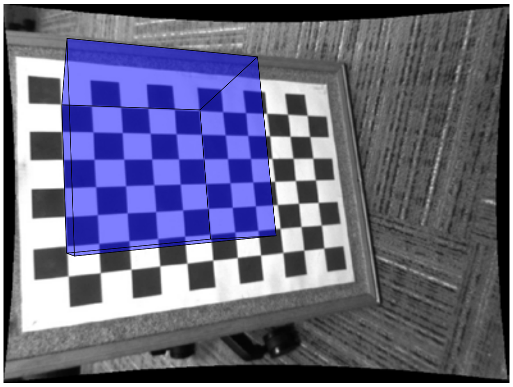
The selected augmented reality:

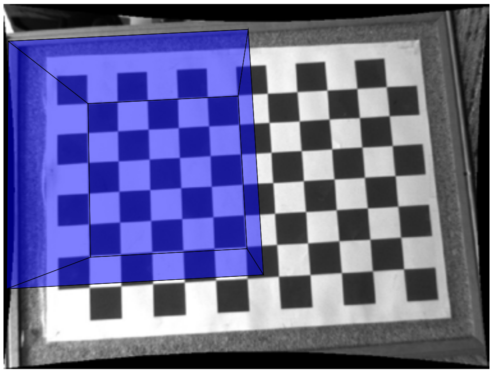




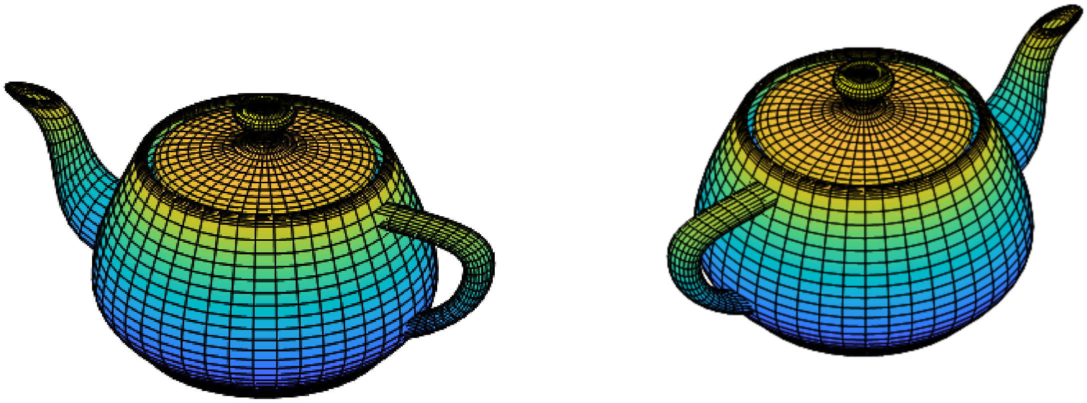




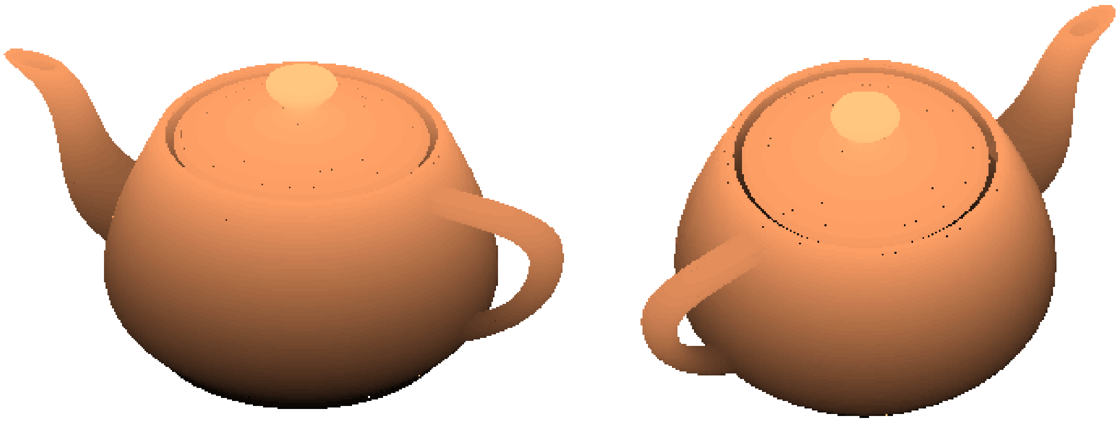




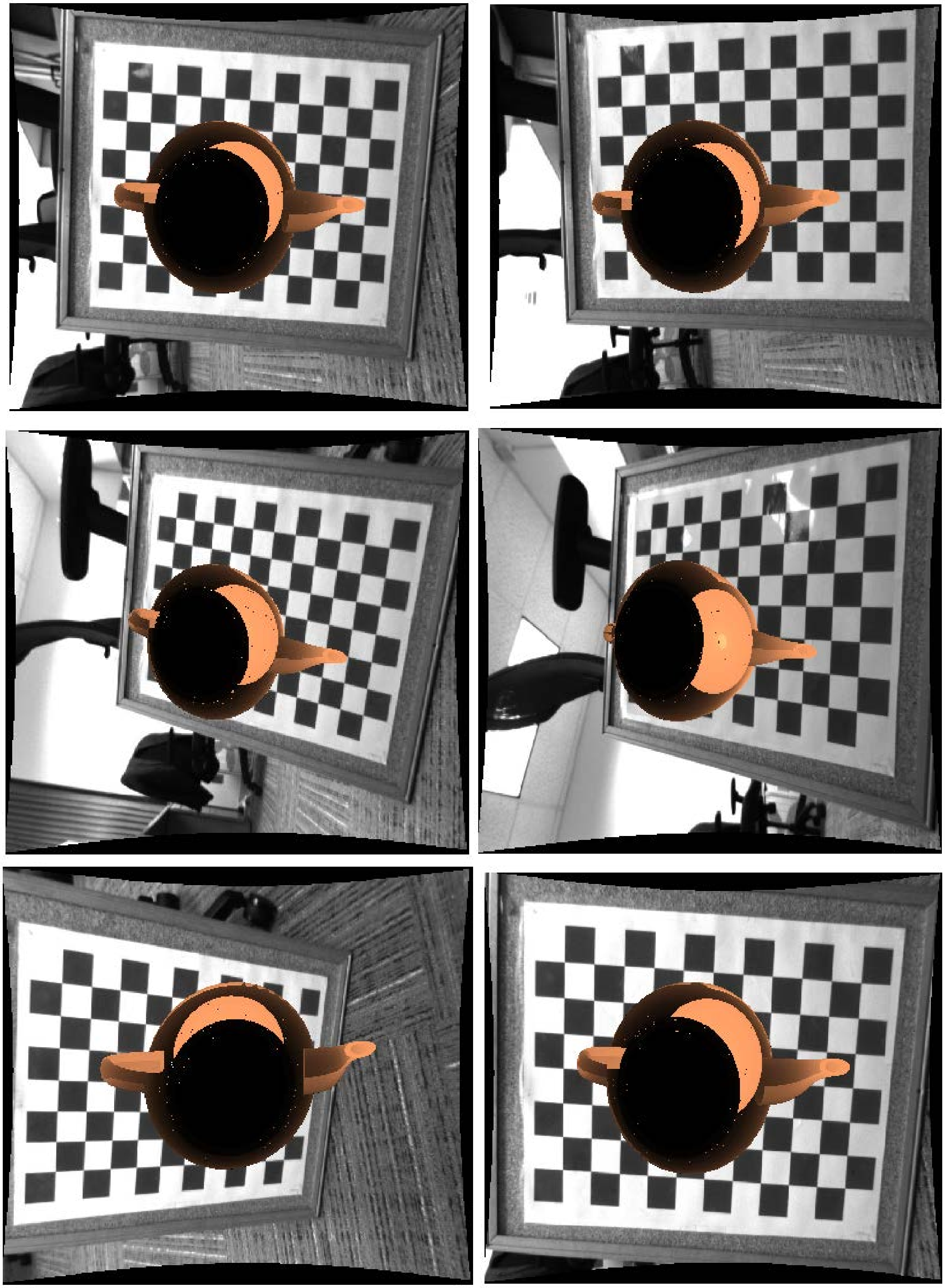
1). A complex 3D teapot



With copper color:



The selected Augmented Reality:



Observations:

1). The simple cubic works very well and the position is well captured.

2). The position of the teapot in the picture is not well captured accurately. Some improvements for the accurate determination of the position should be done.

Appendix

Camera calibration:

```
% Define images to process
% Detect checkerboards in images
[imagePoints, boardSize, imagesUsed] =
detectCheckerboardPoints(imageFileNames);
imageFileNames = imageFileNames(imagesUsed);

% Read the first image to obtain image size
originalImage = imread(imageFileNames{1});
[mrows, ncols, ~] = size(originalImage);

% Generate world coordinates of the corners of the squares
squareSize = 25;  % in units of 'millimeters'
worldPoints = generateCheckerboardPoints(boardSize, squareSize);
% Calibrate the camera
[cameraParams, imagesUsed, estimationErrors] =
estimateCameraParameters(imagePoints, worldPoints, ...
    'EstimateSkew', false, 'EstimateTangentialDistortion', false, ...
    'NumRadialDistortionCoefficients', 2, 'WorldUnits', 'millimeters', ...
    'InitialIntrinsicMatrix', [], 'InitialRadialDistortion', [], ...
    'ImageSize', [mrows, ncols]);
% View reprojection errors
h1=figure; showReprojectionErrors(cameraParams);

% Visualize pattern locations
h2=figure; showExtrinsics(cameraParams, 'CameraCentric');

% Display parameter estimation errors
displayErrors(estimationErrors, cameraParams);

% For example, you can use the calibration data to remove effects of lens
distortion.
undistortedImage = undistortImage(originalImage, cameraParams);
```

Augmented reality:

```
figure(1);
I = imread('myImage.jpg');
[im, newOrigin] = undistortImage(I, cameraParams, 'OutputView', 'full'); %
Eliminate the distortion of picture
[pts,boardSize] = detectCheckerboardPoints(im); % Find the fiducial marker in
the Image marker
scale = 25;
worldPoints = generateCheckerboardPoints(boardSize,scale); %Generate the
checkerboard in real world
[rotationMat,translation]=extrinsics(pts,worldPoints,cameraParams);%Generate
the extrinsics vector (Rotation and translation)
camMat = cameraMatrix(cameraParams,rotationMat,translation); % Find the
camera matrix form the rotation and translation vector
cube = Cubic(5*25);  % Define the cubic
imshow(im);
cube.render(gca,camMat); % Show the virtual object in the picture

figure(2);
I = imread('CAM0_image_0005.jpg');
[im, newOrigin] = undistortImage(I, cameraParams, 'OutputView', 'full'); %
Eliminate the distortion of picture
[pts,boardSize] = detectCheckerboardPoints(im); % Find the fiducial marker in
the Image marker
scale = 25;
worldPoints = generateCheckerboardPoints(boardSize,scale); %Generate the
checkerboard in real world
[rotationMat,translation]=extrinsics(pts,worldPoints,cameraParams);%Generate
the extrinsics vector (Rotation and translation)
camMat = cameraMatrix(cameraParams,rotationMat,translation); % Find the
camera matrix form the rotation and translation vector
cube = Cubic(5*25);  % Define the cubic
imshow(im);
cube.render(gca,camMat); % Show the virtual object in the picture

figure(3);
I = imread('CAM0_image_0009.jpg');
[im, newOrigin] = undistortImage(I, cameraParams, 'OutputView', 'full'); %
Eliminate the distortion of picture
[pts,boardSize] = detectCheckerboardPoints(im); % Find the fiducial marker in
the Image marker
scale = 25;
worldPoints = generateCheckerboardPoints(boardSize,scale); %Generate the
checkerboard in real world
[rotationMat,translation]=extrinsics(pts,worldPoints,cameraParams);%Generate
the extrinsics vector (Rotation and translation)
camMat = cameraMatrix(cameraParams,rotationMat,translation); % Find the
camera matrix form the rotation and translation vector
cube = Cubic(5*25);  % Define the cubic
imshow(im);
cube.render(gca,camMat); % Show the virtual object in the picture

figure(4);
I = imread('CAM0_image_0012.jpg');
```

```
[im, newOrigin] = undistortImage(I, cameraParams, 'OutputView', 'full'); %
Eliminate the distortion of picture
[pts,boardSize] = detectCheckerboardPoints(im); % Find the fiducial marker in
the Image marker
scale = 25;
worldPoints = generateCheckerboardPoints(boardSize,scale); %Generate the
checkerboard in real world
[rotationMat,translation]=extrinsics(pts,worldPoints,cameraParams);%Generate
the extrinsics vector (Rotation and translation)
camMat = cameraMatrix(cameraParams,rotationMat,translation); % Find the
camera matrix form the rotation and translation vector
cube = Cubic(5*25);  % Define the cubic
imshow(im);
cube.render(gca,camMat); % Show the virtual object in the picture

figure(5);
I = imread('CAM0_image_0020.jpg');
[im, newOrigin] = undistortImage(I, cameraParams, 'OutputView', 'full'); %
Eliminate the distortion of picture
[pts,boardSize] = detectCheckerboardPoints(im); % Find the fiducial marker in
the Image marker
scale = 25;
worldPoints = generateCheckerboardPoints(boardSize,scale); %Generate the
checkerboard in real world
[rotationMat,translation]=extrinsics(pts,worldPoints,cameraParams);%Generate
the extrinsics vector (Rotation and translation)
camMat = cameraMatrix(cameraParams,rotationMat,translation); % Find the
camera matrix form the rotation and translation vector
cube = Cubic(5*25);  % Define the cubic
imshow(im);
cube.render(gca,camMat); % Show the virtual object in the picture

figure(6);
I = imread('CAM0_image_0022.jpg');
[im, newOrigin] = undistortImage(I, cameraParams, 'OutputView', 'full'); %
Eliminate the distortion of picture
[pts,boardSize] = detectCheckerboardPoints(im); % Find the fiducial marker in
the Image marker
scale = 25;
worldPoints = generateCheckerboardPoints(boardSize,scale); %Generate the
checkerboard in real world
[rotationMat,translation]=extrinsics(pts,worldPoints,cameraParams);%Generate
the extrinsics vector (Rotation and translation)
camMat = cameraMatrix(cameraParams,rotationMat,translation); % Find the
camera matrix form the rotation and translation vector
cube = Cubic(5*25);  % Define the cubic
imshow(im);
cube.render(gca,camMat); % Show the virtual object in the picture
```

Augmented reality of the teapot:

```
figure(1);
I = imread('myImage.jpg');
[im, newOrigin] = undistortImage(I, cameraParams, 'OutputView', 'full');
[pts,boardSize] = detectCheckerboardPoints(im);
scale = 25;
worldPoints = generateCheckerboardPoints(boardSize,scale); %
[rotationMat,translation]=extrinsics(pts,worldPoints,cameraParams);%
camMat = cameraMatrix(cameraParams,rotationMat,translation); %
[verts, faces, cindex] = teapotGeometry; vert =
[verts,ones(size(verts,1),1)];
cindex = 0.1*cindex;
scale = 1;
verts = scale*verts;
vert2d = vert*camMat;
vertImage =  vert2d(:,1:2)./vert2d(:,3);
image('CData',im,'XData',[min(vertImage(:,1))-5
max(vertImage(:,1))+5],'YData',[min(vertImage(:,2))-5 max(vertImage(:,2))+5])
hold on;
cindex = 0.2+cindex;
patch('Faces',faces,'Vertices',vertImage,'FaceVertexCData',cindex,'FaceColor'
,'interp','LineStyle','none');
colormap(copper);
axis off;

figure(2);
I = imread('CAM0_image_0005.jpg');
[im, newOrigin] = undistortImage(I, cameraParams, 'OutputView', 'full');
[pts,boardSize] = detectCheckerboardPoints(im);
scale = 25;
worldPoints = generateCheckerboardPoints(boardSize,scale); %
[rotationMat,translation]=extrinsics(pts,worldPoints,cameraParams);%
camMat = cameraMatrix(cameraParams,rotationMat,translation); %
[verts, faces, cindex] = teapotGeometry; vert =
[verts,ones(size(verts,1),1)];
cindex = 0.1*cindex;
scale = 1;
verts = scale*verts;
vert2d = vert*camMat;
vertImage =  vert2d(:,1:2)./vert2d(:,3);
image('CData',im,'XData',[min(vertImage(:,1))-5
max(vertImage(:,1))+5],'YData',[min(vertImage(:,2))-5 max(vertImage(:,2))+5])
hold on;
cindex = 0.2+cindex;
patch('Faces',faces,'Vertices',vertImage,'FaceVertexCData',cindex,'FaceColor'
,'interp','LineStyle','none');
colormap(copper);
axis off;

figure(3);
I = imread('CAM0_image_0009.jpg');
[im, newOrigin] = undistortImage(I, cameraParams, 'OutputView', 'full');
[pts,boardSize] = detectCheckerboardPoints(im);
scale = 25;
worldPoints = generateCheckerboardPoints(boardSize,scale); %
[rotationMat,translation]=extrinsics(pts,worldPoints,cameraParams);%
camMat = cameraMatrix(cameraParams,rotationMat,translation); %
```

```
[verts, faces, cindex] = teapotGeometry; vert =
[verts,ones(size(verts,1),1)];
cindex = 0.1*cindex;
scale = 1;
verts = scale*verts;
vert2d = vert*camMat;
vertImage =  vert2d(:,1:2)./vert2d(:,3);
image('CData',im,'XData',[min(vertImage(:,1))-5
max(vertImage(:,1))+5],'YData',[min(vertImage(:,2))-5 max(vertImage(:,2))+5])
hold on;
cindex = 0.2+cindex;
patch('Faces',faces,'Vertices',vertImage,'FaceVertexCData',cindex,'FaceColor'
,'interp','LineStyle','none');
colormap(copper);
axis off;

figure(4);
I = imread('CAM0_image_0012.jpg');
[im, newOrigin] = undistortImage(I, cameraParams, 'OutputView', 'full');
[pts,boardSize] = detectCheckerboardPoints(im);
scale = 25;
worldPoints = generateCheckerboardPoints(boardSize,scale); %
[rotationMat,translation]=extrinsics(pts,worldPoints,cameraParams);%
camMat = cameraMatrix(cameraParams,rotationMat,translation); %
[verts, faces, cindex] = teapotGeometry; vert =
[verts,ones(size(verts,1),1)];
cindex = 0.1*cindex;
scale = 1;
verts = scale*verts;
vert2d = vert*camMat;
vertImage =  vert2d(:,1:2)./vert2d(:,3);
image('CData',im,'XData',[min(vertImage(:,1))-5
max(vertImage(:,1))+5],'YData',[min(vertImage(:,2))-5 max(vertImage(:,2))+5])
hold on;
cindex = 0.2+cindex;
patch('Faces',faces,'Vertices',vertImage,'FaceVertexCData',cindex,'FaceColor'
,'interp','LineStyle','none');
colormap(copper);
axis off;

figure(5);
I = imread('CAM0_image_0020.jpg');
[im, newOrigin] = undistortImage(I, cameraParams, 'OutputView', 'full');
[pts,boardSize] = detectCheckerboardPoints(im);
scale = 25;
worldPoints = generateCheckerboardPoints(boardSize,scale); %
[rotationMat,translation]=extrinsics(pts,worldPoints,cameraParams);%
camMat = cameraMatrix(cameraParams,rotationMat,translation); %
[verts, faces, cindex] = teapotGeometry; vert =
[verts,ones(size(verts,1),1)];
cindex = 0.1*cindex;
scale = 1;
verts = scale*verts;
vert2d = vert*camMat;
vertImage =  vert2d(:,1:2)./vert2d(:,3);
```

```
image('CData',im,'XData',[min(vertImage(:,1))-5
max(vertImage(:,1))+5],'YData',[min(vertImage(:,2))-5 max(vertImage(:,2))+5])
hold on;
cindex = 0.2+cindex;
patch('Faces',faces,'Vertices',vertImage,'FaceVertexCData',cindex,'FaceColor'
,'interp','LineStyle','none');
colormap(copper);
axis off;

figure(6);
I = imread('CAM0_image_0022.jpg');
[im, newOrigin] = undistortImage(I, cameraParams, 'OutputView', 'full');
[pts,boardSize] = detectCheckerboardPoints(im);
scale = 25;
worldPoints = generateCheckerboardPoints(boardSize,scale); %
[rotationMat,translation]=extrinsics(pts,worldPoints,cameraParams);%
camMat = cameraMatrix(cameraParams,rotationMat,translation); %
[verts, faces, cindex] = teapotGeometry; vert =
[verts,ones(size(verts,1),1)];
cindex = 0.1*cindex;
scale = 1;
verts = scale*verts;
vert2d = vert*camMat;
vertImage =  vert2d(:,1:2)./vert2d(:,3);
image('CData',im,'XData',[min(vertImage(:,1))-5
max(vertImage(:,1))+5],'YData',[min(vertImage(:,2))-5 max(vertImage(:,2))+5])
hold on;
cindex = 0.2+cindex;
patch('Faces',faces,'Vertices',vertImage,'FaceVertexCData',cindex,'FaceColor'
,'interp','LineStyle','none');
colormap(copper);
axis off;
```

Definition of the cubic:

```
classdef Cubic < handle
    properties
        vertices
        faces
        scale
        patch
    end

    methods
        function this = Cubic(scale)
            this.scale = scale; %??
            this.vertices = scale*...
               [0 0  0;
                0 0 -1;
                0 1 -1;
                0 1  0;
                1 0  0;
                1 1  0;
                1 0 -1;
                1 1 -1;]; %8???
            this.faces = ...
               [1 4 3 2;
                7 8 3 2;
                3 4 6 8;
                5 6 8 7;
                1 4 6 5;
                1 2 7 5;]; %6??
        end

        function render(this,ax,camMat)
            % Homogeneous coordinates (x,y,1)
            vert = [this.vertices,ones(size(this.vertices,1),1)];
            % Transform vertices from 3d world into 2d image plane
            vert2d = vert*camMat;
            % Homogenous coordinates (x,y,1) to (x,y)
            for i = 1:size(vert2d,1)
                vertImage(i,1:2) =  vert2d(i,1:2)/vert2d(i,3);
            end
            % draw patch
            if isempty(this.patch) || ~isvalid(this.patch)
                hold(ax,'on');
                this.patch =
patch('Faces',this.faces,'Vertices',vertImage,'facecolor','b','faceAlpha',0.5
);
                hold(ax,'off');
            else

                this.patch.Vertices= vertImage;
            end
            drawnow;

        end

        function clear(this)
            delete(this.patch)
        end
    end
end
```