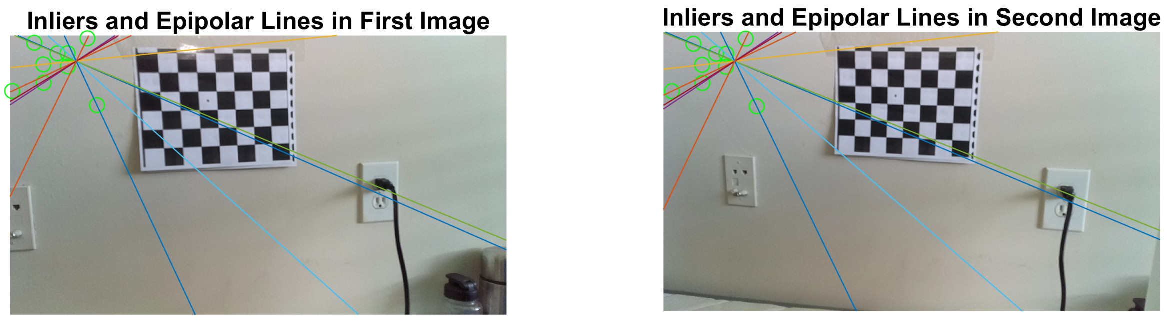
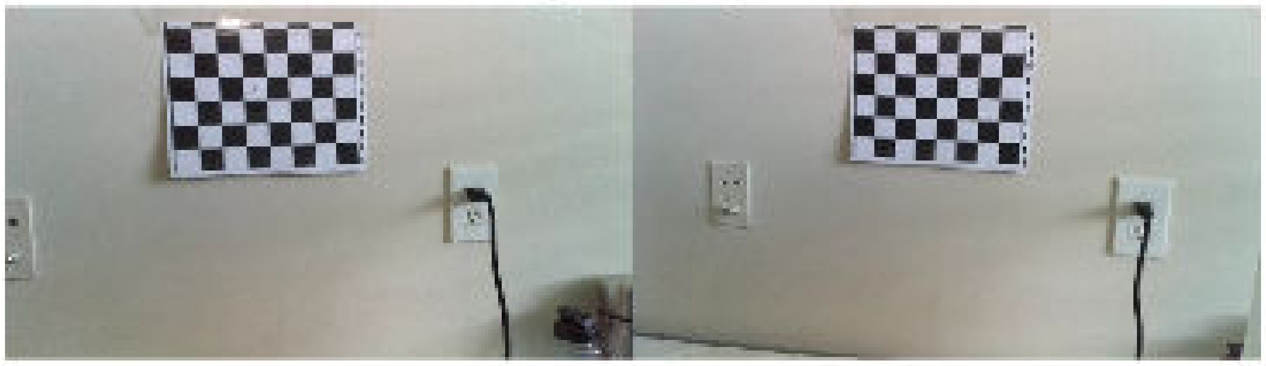
```
load stereoPointPairs
[fLMedS,inliers] = estimateFundamentalMatrix(matchedPoints1,...
    matchedPoints2,'NumTrials',4000);
I1 = imread('fps_image13.jpg');
figure;
subplot(121);
imshow(I1);
title('Inliers and Epipolar Lines in First Image'); hold on;
plot(matchedPoints1(inliers,1),matchedPoints1(inliers,2),'go')
epiLines = epipolarLine(fLMedS',matchedPoints2(inliers,:));
points = lineToBorderPoints(epiLines,size(I1));
line(points(:,[1,3])',points(:,[2,4])');
I2 = imread('fps_image32.jpg');
subplot(122);
imshow(I2);
title('Inliers and Epipolar Lines in Second Image'); hold on;
plot(matchedPoints2(inliers,1),matchedPoints2(inliers,2),'go')
epiLines = epipolarLine(fLMedS,matchedPoints1(inliers,:));
points = lineToBorderPoints(epiLines,size(I2));
line(points(:,[1,3])',points(:,[2,4])');
truesize;
```


Inliers and Epipolar Lines in First Image
Inliers and Epipolar Lines in Second Image

```
Warning: Image is too big to fit on screen; displaying at 23% scale.
```

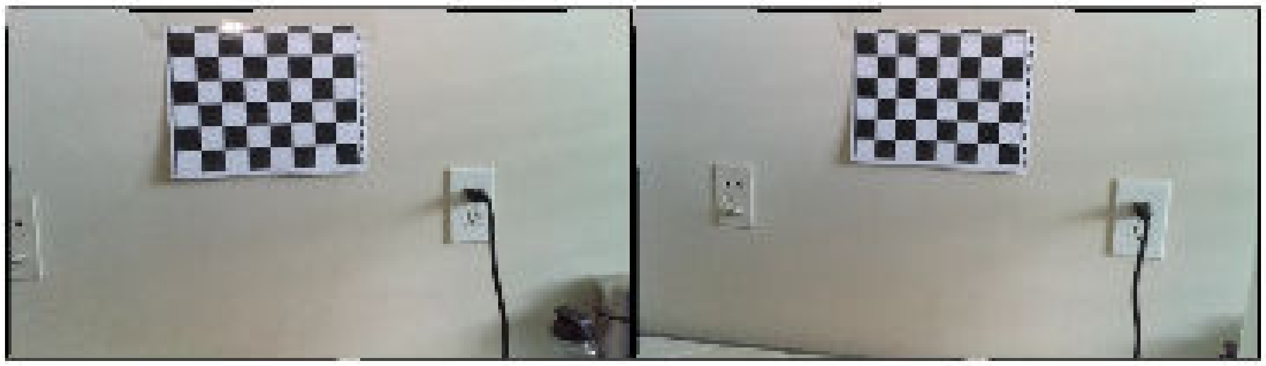
```
imageDir = fullfile('motion_2_views');
images = imageDatastore(imageDir);
I1 = readimage(images, 1);
I2 = readimage(images, 2);
figure
imshowpair(I1, I2, 'montage');
title('Original Images');
```

Original Images



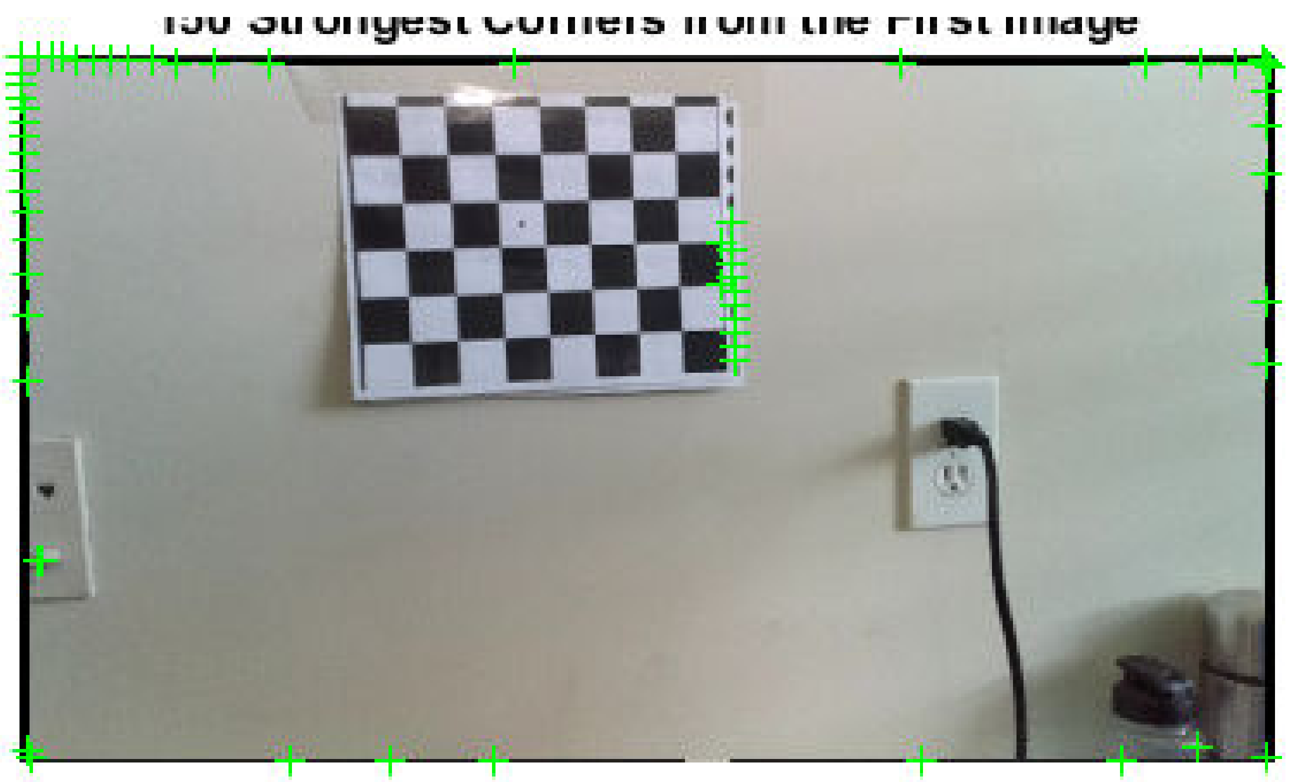
```
% Load precomputed camera parameters
load upToScaleReconstructionCameraParameters.mat
I1 = undistortImage(I1, cameraParams);
I2 = undistortImage(I2, cameraParams);
figure
imshowpair(I1, I2, 'montage');
title('Undistorted Images');
```

Undistorted Images

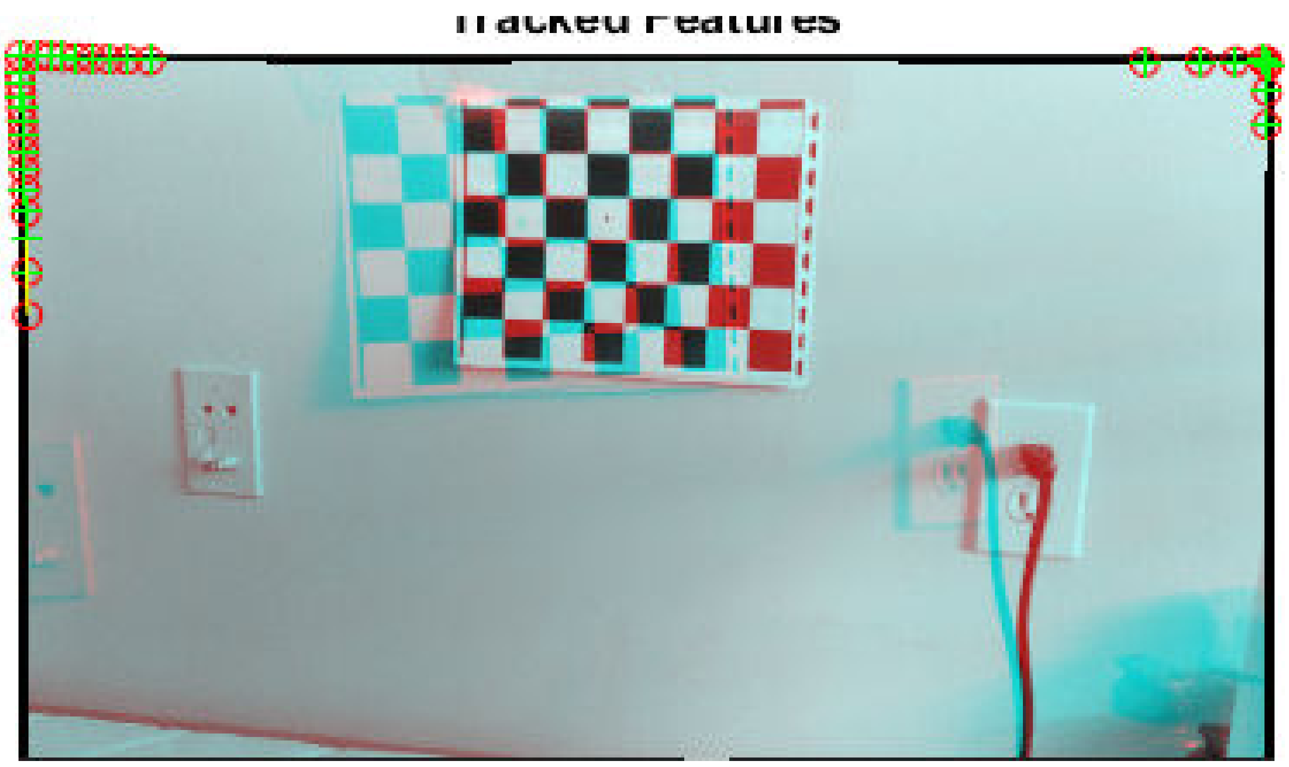


```
% Detect feature points
imagePoints1 = detectMinEigenFeatures(im2gray(I1), 'MinQuality', 0.1);

% Visualize detected points
figure
imshow(I1, 'InitialMagnification', 50);
title('150 Strongest Corners from the First Image');
hold on
```

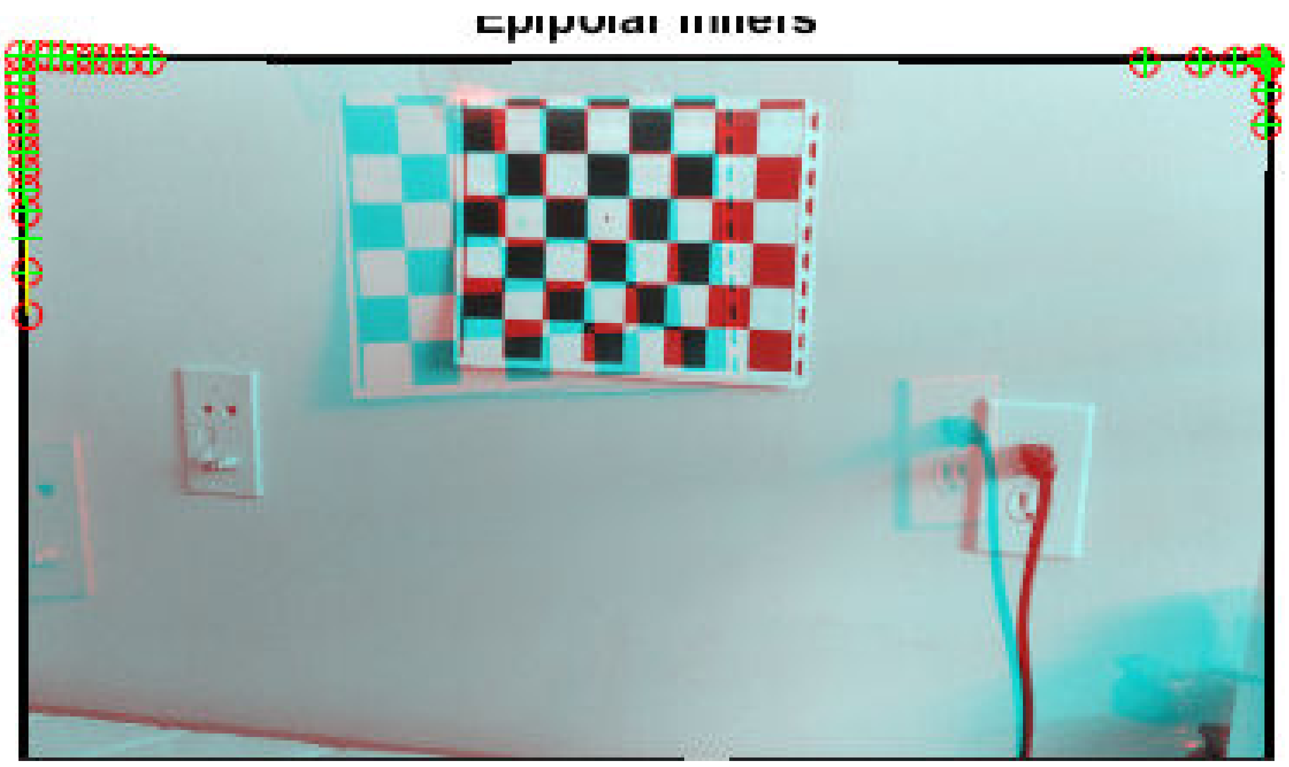
```
plot(selectStrongest(imagePoints1, 150));
```



```
% Create the point tracker
tracker = vision.PointTracker('MaxBidirectionalError', 1, 'NumPyramidLevels', 5);

% Initialize the point tracker
imagePoints1 = imagePoints1.Location;
initialize(tracker, imagePoints1, I1);

% Track the points
[imagePoints2, validIdx] = step(tracker, I2);
matchedPoints1 = imagePoints1(validIdx, :);
matchedPoints2 = imagePoints2(validIdx, :);

% Visualize correspondences
figure
showMatchedFeatures(I1, I2, matchedPoints1, matchedPoints2);
title('Tracked Features');
```

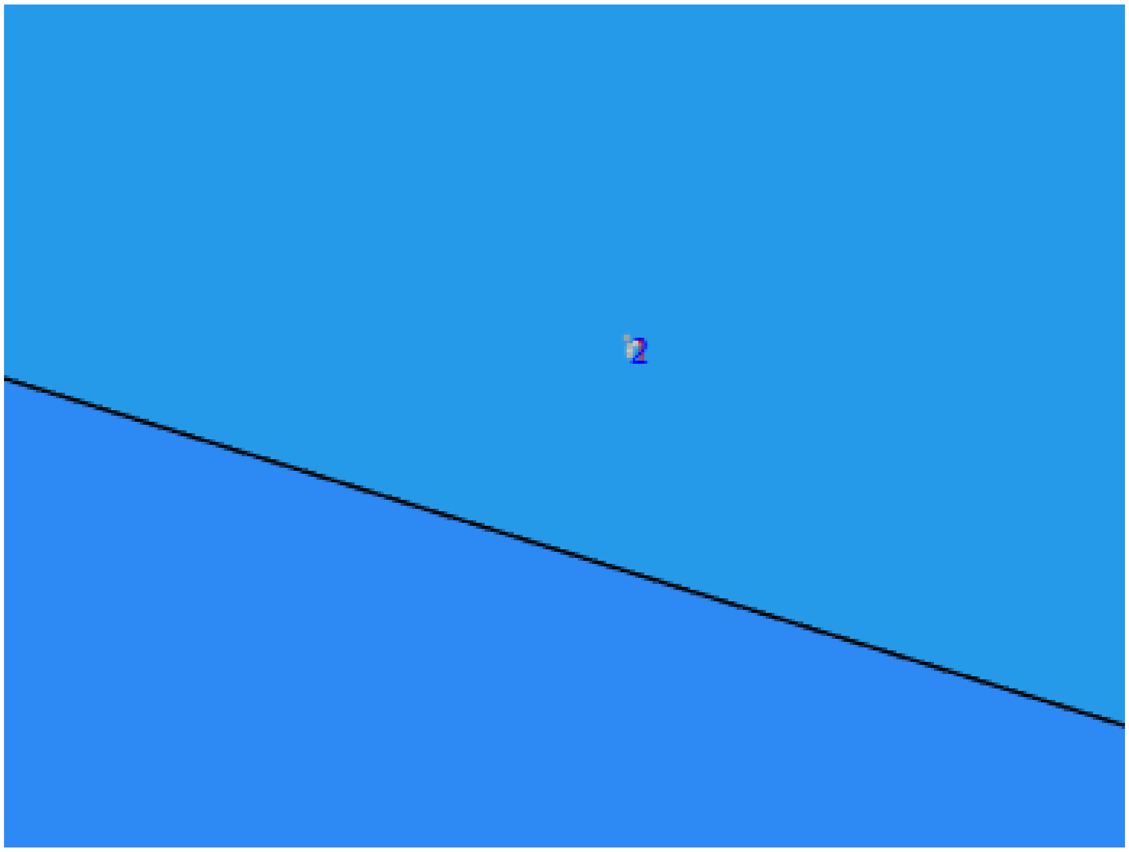


```
% Estimate the fundamental matrix
[E, epipolarInliers] = estimateEssentialMatrix(...
    matchedPoints1, matchedPoints2, cameraParams, 'Confidence', 99.99);

% Find epipolar inliers
inlierPoints1 = matchedPoints1(epipolarInliers, :);
inlierPoints2 = matchedPoints2(epipolarInliers, :);

% Display inlier matches
figure
showMatchedFeatures(I1, I2, inlierPoints1, inlierPoints2);
title('Epipolar Inliers');
```

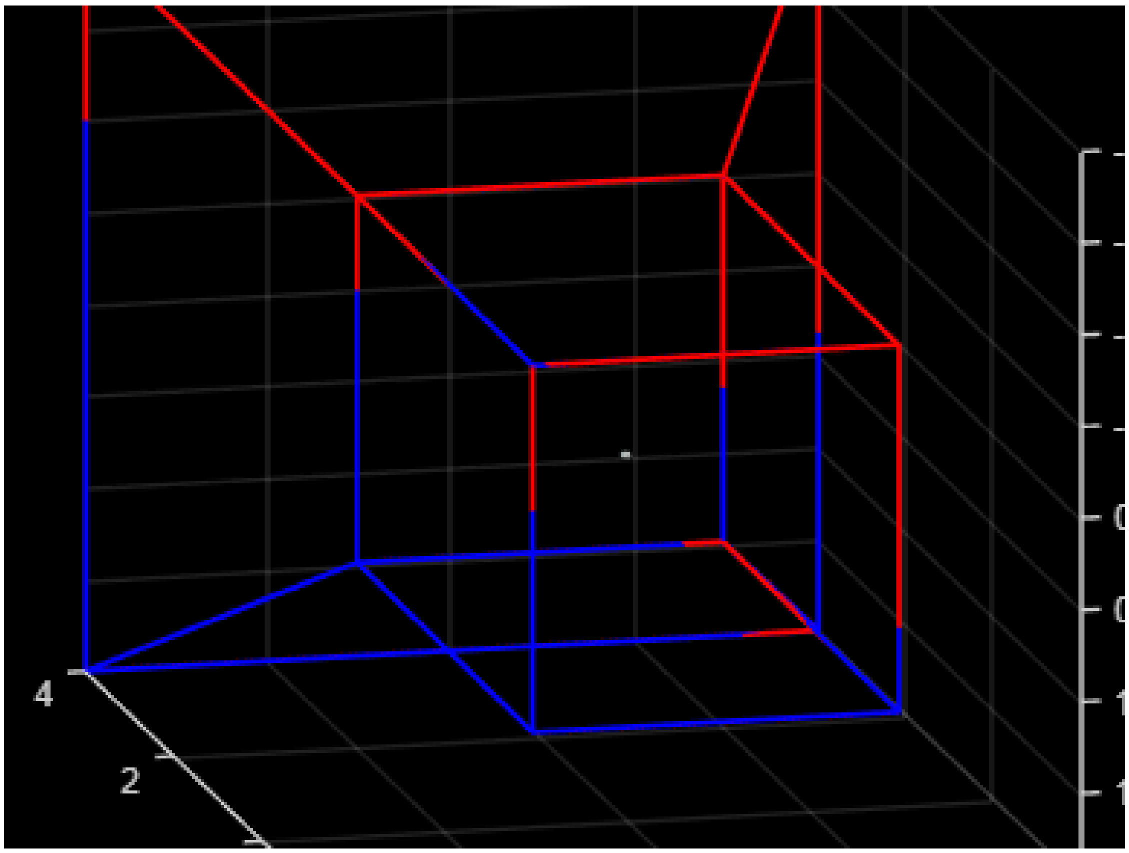


```
[orient, loc] = relativeCameraPose(E, cameraParams, inlierPoints1, inlierPoints2);
% Detect dense feature points. Use an ROI to exclude points close to the
% image edges.
roi = [30, 30, size(I1, 2) - 30, size(I1, 1) - 30];
imagePoints1 = detectMinEigenFeatures(im2gray(I1), 'ROI', roi, ...
    'MinQuality', 0.001);

% Create the point tracker
tracker = vision.PointTracker('MaxBidirectionalError', 1, 'NumPyramidLevels', 5);

% Initialize the point tracker
imagePoints1 = imagePoints1.Location;
initialize(tracker, imagePoints1, I1);

% Track the points
[imagePoints2, validIdx] = step(tracker, I2);
matchedPoints1 = imagePoints1(validIdx, :);
matchedPoints2 = imagePoints2(validIdx, :);

% Compute the camera matrices for each position of the camera
% The first camera is at the origin looking along the Z-axis. Thus, its
% transformation is identity.
tform1 = rigid3d;
camMatrix1 = cameraMatrix(cameraParams, tform1);

% Compute extrinsics of the second camera
cameraPose = rigid3d(orient, loc);
tform2 = cameraPoseToExtrinsics(cameraPose);
```

```
camMatrix2 = cameraMatrix(cameraParams, tform2);

% Compute the 3-D points
points3D = triangulate(matchedPoints1, matchedPoints2, camMatrix1, camMatrix2);

% Get the color of each reconstructed point
numPixels = size(I1, 1) * size(I1, 2);
allColors = reshape(I1, [numPixels, 3]);
colorIdx = sub2ind([size(I1, 1), size(I1, 2)], round(matchedPoints1(:,2)), ...
    round(matchedPoints1(:, 1)));
color = allColors(colorIdx, :);

% Create the point cloud
ptCloud = pointCloud(points3D, 'Color', color);
% Visualize the camera locations and orientations
cameraSize = 0.3;
figure
plotCamera('Size', cameraSize, 'Color', 'r', 'Label', '1', 'Opacity', 0);
hold on
grid on
plotCamera('Location', loc, 'Orientation', orient, 'Size', cameraSize, ...
    'Color', 'b', 'Label', '2', 'Opacity', 0);

% Visualize the point cloud
pcshow(ptCloud, 'VerticalAxis', 'y', 'VerticalAxisDir', 'down', ...
    'MarkerSize', 45);

% Rotate and zoom the plot
camorbit(0, -30);
camzoom(1.5);

% Label the axes
xlabel('x-axis');
ylabel('y-axis');
zlabel('z-axis')

title('Up to Scale Reconstruction of the Scene');
% Detect the globe
globe = pcfitsphere(ptCloud, 0.1);

% Display the surface of the globe
plot(globe);
title('Estimated Location and Size of the Globe');
hold off
```



```
% Determine the scale factor
scaleFactor = 10 / globe.Radius;

% Scale the point cloud
ptCloud = pointCloud(points3D * scaleFactor, 'Color', color);
loc = loc * scaleFactor;

% Visualize the point cloud in centimeters
cameraSize = 2;
figure
plotCamera('Size', cameraSize, 'Color', 'r', 'Label', '1', 'Opacity', 0);
hold on
grid on
plotCamera('Location', loc, 'Orientation', orient, 'Size', cameraSize, ...
    'Color', 'b', 'Label', '2', 'Opacity', 0);

% Visualize the point cloud
pcshow(ptCloud, 'VerticalAxis', 'y', 'VerticalAxisDir', 'down', ...
    'MarkerSize', 45);
camorbit(0, -30);
camzoom(1.5);

% Label the axes
xlabel('x-axis (cm)');
ylabel('y-axis (cm)');
zlabel('z-axis (cm)')
title('Metric Reconstruction of the Scene');
```


4
2