Camera Calibration Using AprilTag Markers

AprilTags are widely used as visual markers for applications in object detection, localization, and as a target for camera calibration [1]. AprilTags are similar to QR codes, but are designed to encode less data, and can therefore be decoded faster which is useful, for example, for real-time robotics applications.

This example uses the `readAprilTag` function to detect and localize AprilTags in a calibration pattern. The `readAprilTag` function supports all official tag families. The example also uses additional Computer Vision Toolbox™ functions to perform end-to-end camera calibration. The default checkerboard pattern is replaced by a grid of evenly spaced AprilTags. For an example of using a checkerboard pattern for calibration, refer to Single Camera Calibration.

The advantages of using AprilTags as a calibration pattern are: more robust feature point detection, consistent and repeatable detections. This example can also serve as a template for using other custom calibration patters such as a field of circles instead of a typical checkerboard pattern.

Step 1: Generate the calibration pattern

Download and prepare tag images

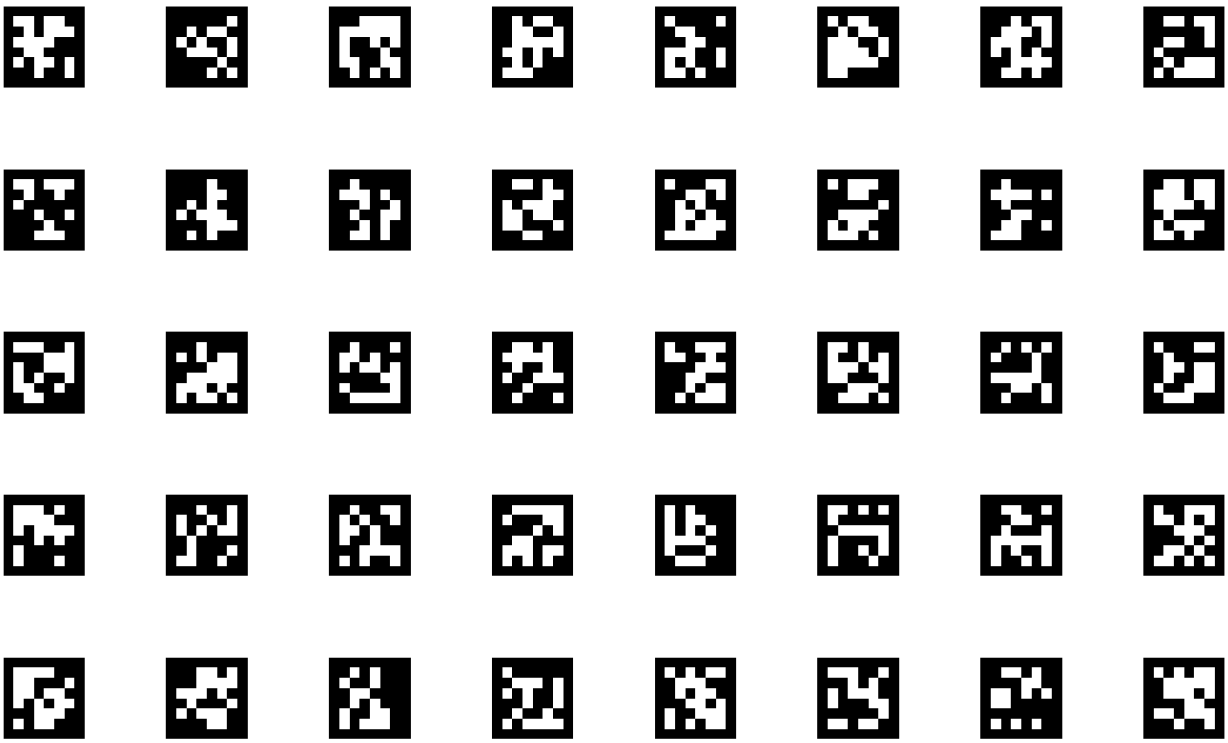
Pre-generated tags for all the supported families can be downloaded from here using a web browser or by running the following code:

```
downloadURL  = 'https://github.com/AprilRobotics/apriltag-imgs/archive/master.zip';
dataFolder   = fullfile(tempdir, 'apriltag-imgs', filesep);
options      = weboptions('Timeout', Inf);
zipFileName  = [dataFolder, 'apriltag-imgs-master.zip'];
folderExists = exist(dataFolder, 'dir');

% Create a folder in a temporary directory to save the downloaded file
if ~folderExists
    mkdir(dataFolder);
    disp('Downloading apriltag-imgs-master.zip (60.1 MB)...')
    websave(zipFileName, downloadURL, options);

    % Extract contents of the downloaded file
    disp('Extracting apriltag-imgs-master.zip...')
    unzip(zipFileName, dataFolder);
end
```

The helperGenerateAprilTagPattern function at the end of the end of the exmaple can be used to generate a calibration target using the tag images for a specific arrangement of tags. The pattern image is contained in `calibPattern`, which can be used to print the pattern (from MATLAB). The example uses the **tag36h11** family, which provides a reasonable trade-off between detection performance and robustness to false-positive detections.

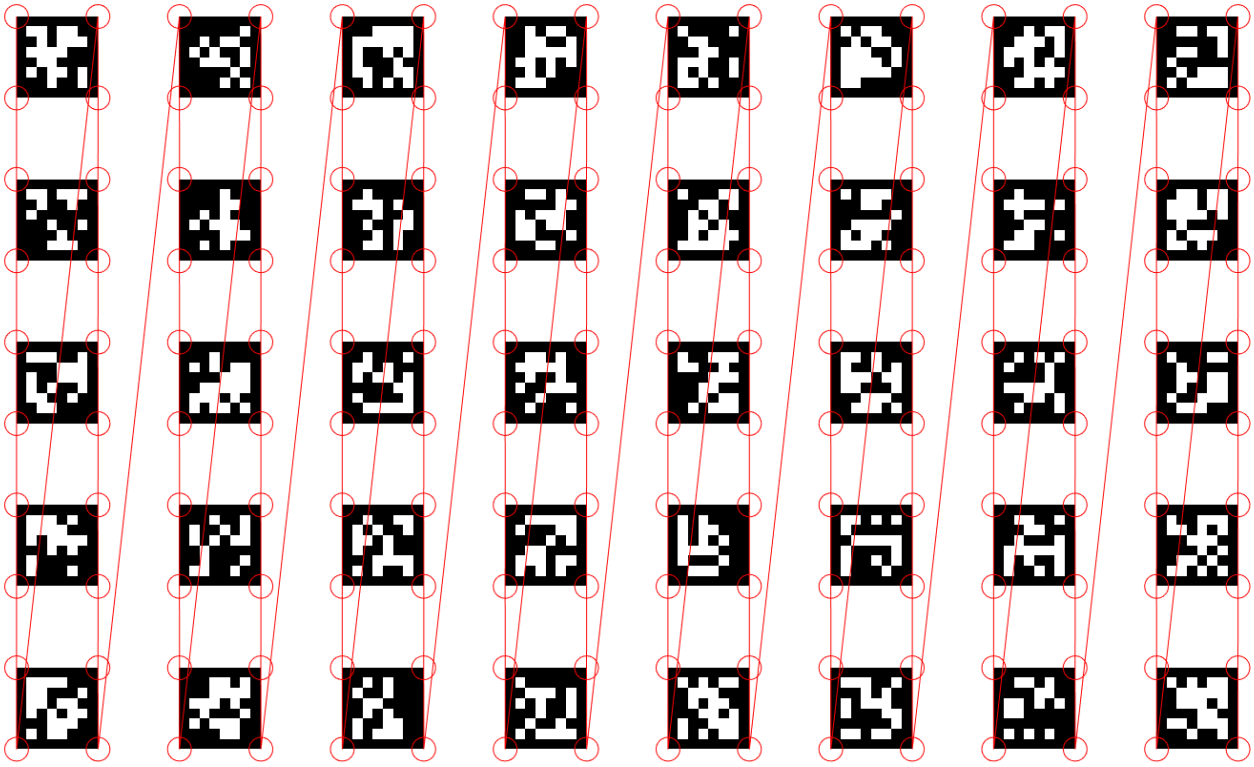
```
% Set the properties of the calibration pattern.
tagArrangement = [5,8];
tagFamily = 'tag36h11';

% Generate the calibration pattern using AprilTags.
```

```
tagImageFolder = [dataFolder 'apriltag-imgs-master/' tagFamily];
imdsTags = imageDatastore(tagImageFolder);
calibPattern = helperGenerateAprilTagPattern(imdsTags, tagArrangement, tagFamily);
```



Using the `readAprilTag` function on this pattern results in detections with the corner locations of the individual tags grouped together. The helperAprilTagToCheckerLocations function can be used to convert this arrangement to a column-major arrangement similar to a checkerboard.

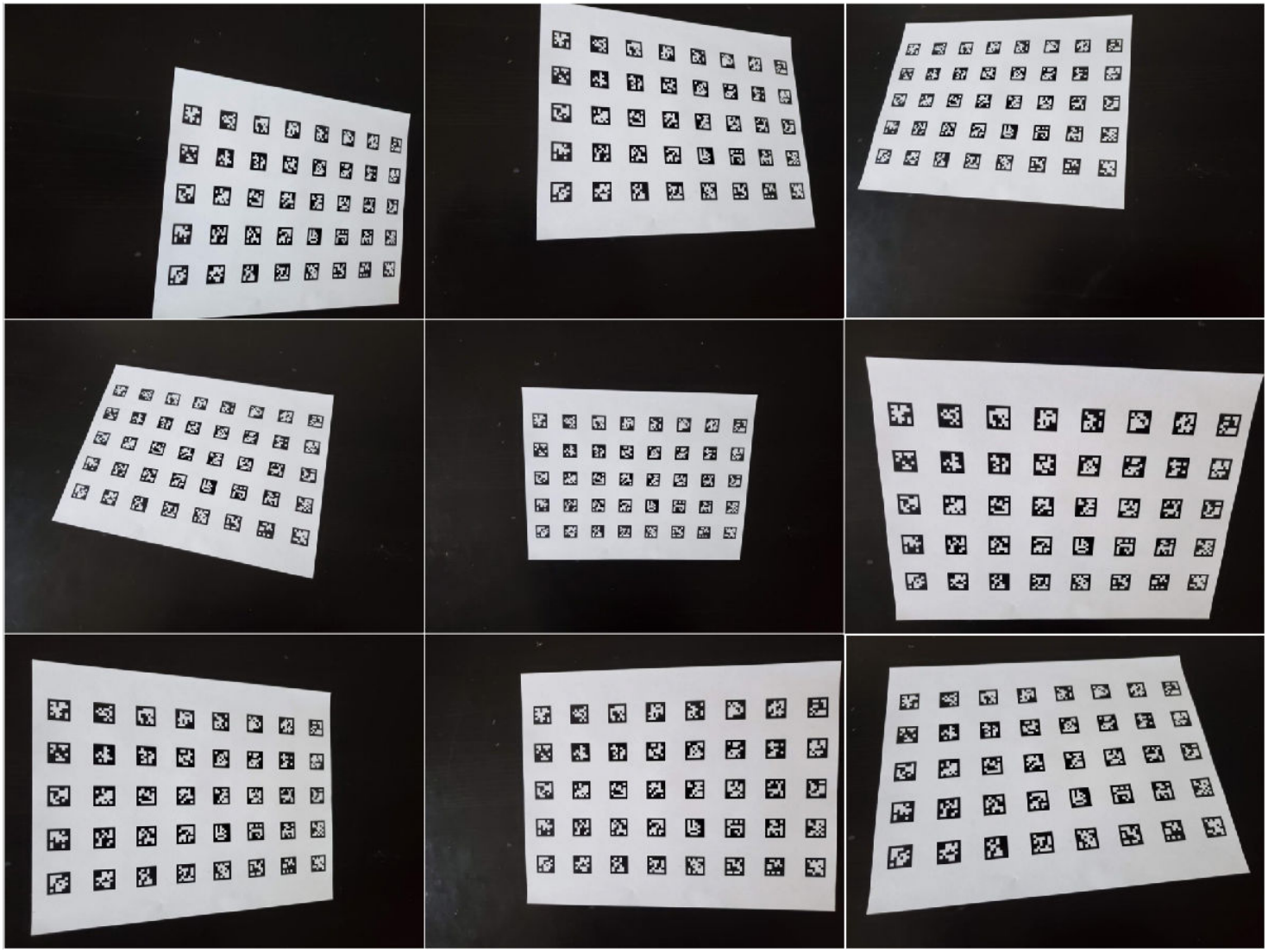
```
% Read and localize the tags in the calibration pattern.
[tagIds, tagLocs] = readAprilTag(calibPattern, tagFamily);

% Sort the tags based on their ID values.
[~, sortIdx] = sort(tagIds);
tagLocs = tagLocs(:,:,sortIdx);

% Reshape the tag corner locations into an M-by-2 array.
tagLocs = reshape(permute(tagLocs,[1,3,2]), [], 2);

% Convert the AprilTag corner locations to checkerboard corner locations.
checkerIdx = helperAprilTagToCheckerLocations(tagArrangement);
imagePoints = tagLocs(checkerIdx(:),:);

% Display corner locations.
figure; imshow(calibPattern); hold on
plot(imagePoints(:,1), imagePoints(:,2), 'ro-', 'MarkerSize',15)
```



Prepare images for calibration

The generated calibration pattern must be printed on a flat surface and the camera that needs to be calibrated is used to capture images of the pattern. A few points to note while preparing images for calibration:

- While the pattern is printed on a paper in this example, consider printing it on a surface that remains flat, and is not subject to deformations due to moisture, etc.
- Since the calibration procedure assumes that the pattern is planar, any imperfections in the pattern (eg. an uneven surface) can reduce the accuracy of the calibration.
- The calibration procedure requires at least 2 images of the pattern but using between 10 and 20 images produces more accurate results.
- Capture a variety of images of the pattern such that the pattern fills most of the image, thus covering the entire field of view. For example, to best capture the lens distortion, have images of the pattern at all edges of the image frame.
- Make sure the pattern is completely visible in the captured images, since images with partially visible patterns will be rejected.
- For more information on preparing images of the calibration pattern, see additional tips in documentation.



Step 2: Detect and localize the AprilTags

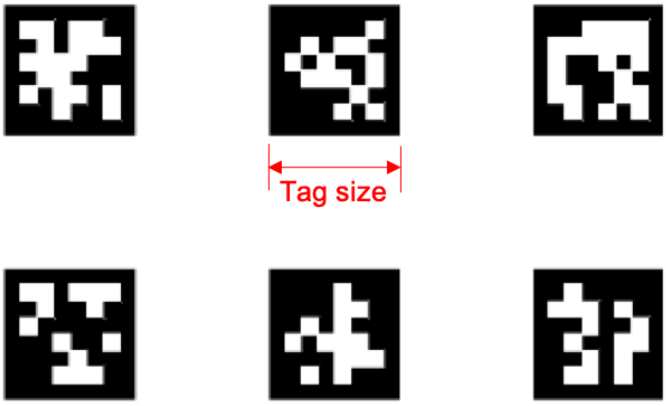
The helperDetectAprilTagCorners function, included at the end of the example, is used to detect and localize the tags from the captured images and arrange them in a checkerboard fashion to be used as key points in the calibration procedure.

```
% Create an imageDatastore object to store the captured images.
imdsCalib = imageDatastore("aprilTagCalibImages/");

% Detect the calibration pattern from the images.
[imagePoints, boardSize] = helperDetectAprilTagCorners(imdsCalib, tagArrangement, tagFamily);
```

Step 3: Generate world points for the calibration pattern

The generated AprilTag pattern is such that the tags are located in a checkerboard fashion, and so the world coordinates for the corresponding image coordinates determined above (in `imagePoints`) can be obtained using the `generateCheckerboardPoints` function.

Here, the size of the square is replaced by the size of the tag and the size of the board is obtained from the previous step. Measure the tag size between the outer black edges of one side of the tag.




```
% Generate world point coordinates for the pattern.
tagSize = 14.732; % in millimeters (0.58")
worldPoints = generateCheckerboardPoints(boardSize, tagSize);
```

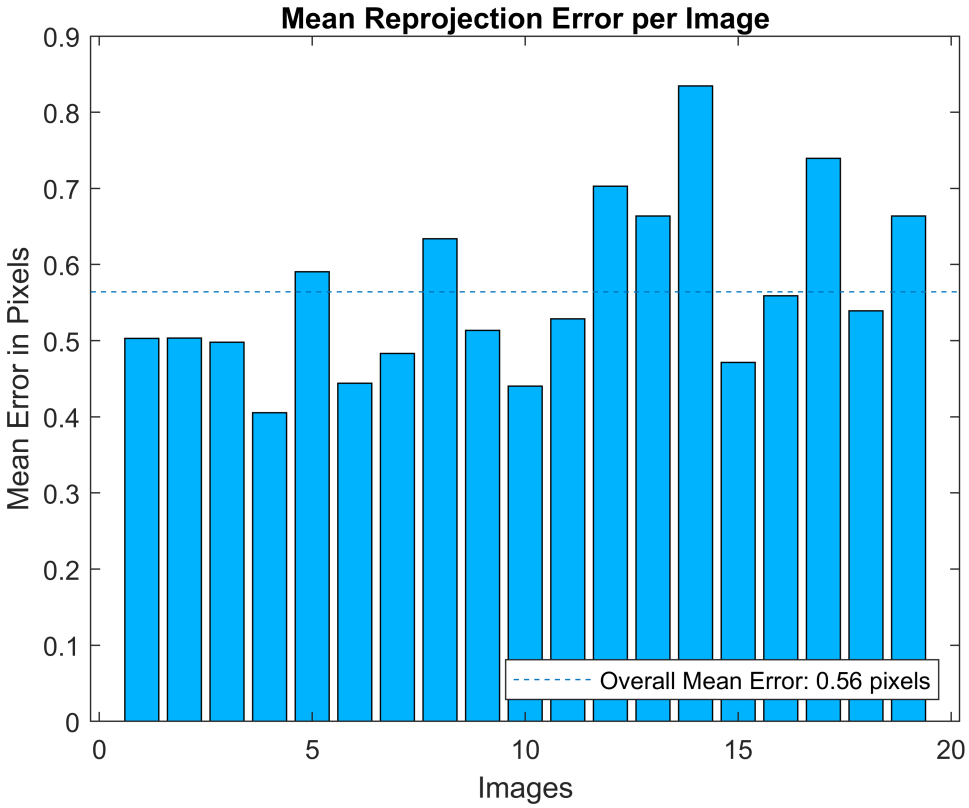
Step 4: Estimate camera parameters

With the image and world point correspondences, estimate the camera parameters using the `estimateCameraParameters` function.

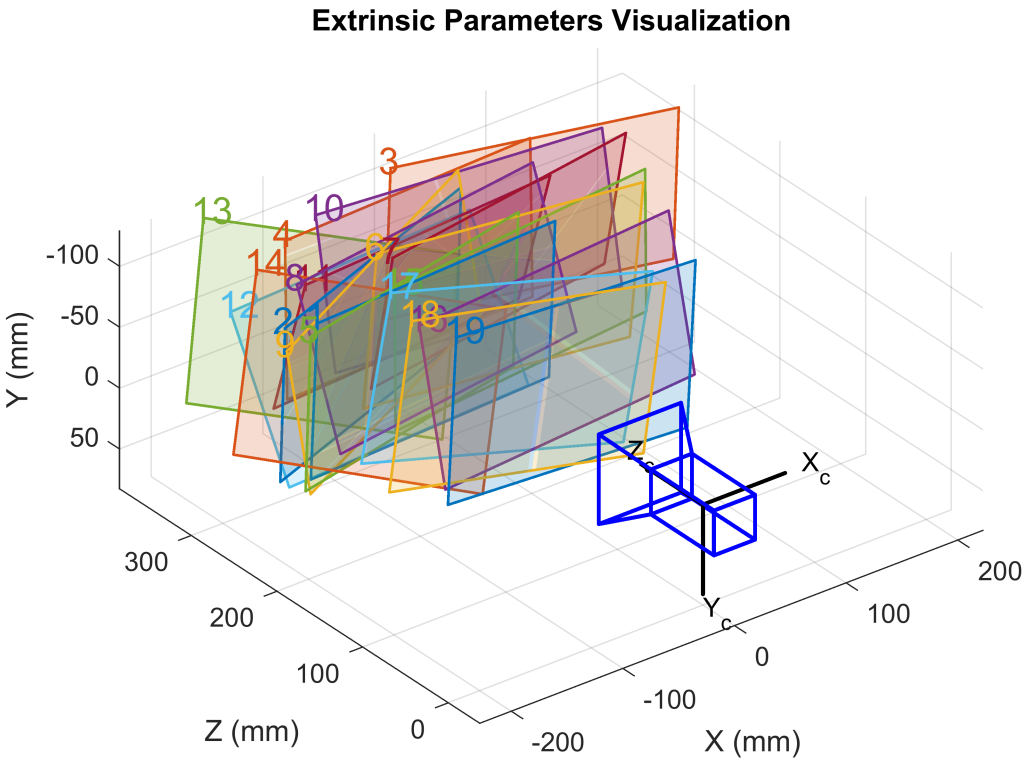
```
% Determine the size of the images.
I = readimage(imdsCalib, 1);
imageSize = [size(I,1), size(I,2)];

% Estimate the camera parameters.
params = estimateCameraParameters(imagePoints, worldPoints, 'ImageSize', imageSize);
```

Visualize the accuracy of the calibration and the extrinsic camera parameters showing the planes of the calibration pattern in the captured images.

```
% Display the reprojection errors.
figure
showReprojectionErrors(params)
```

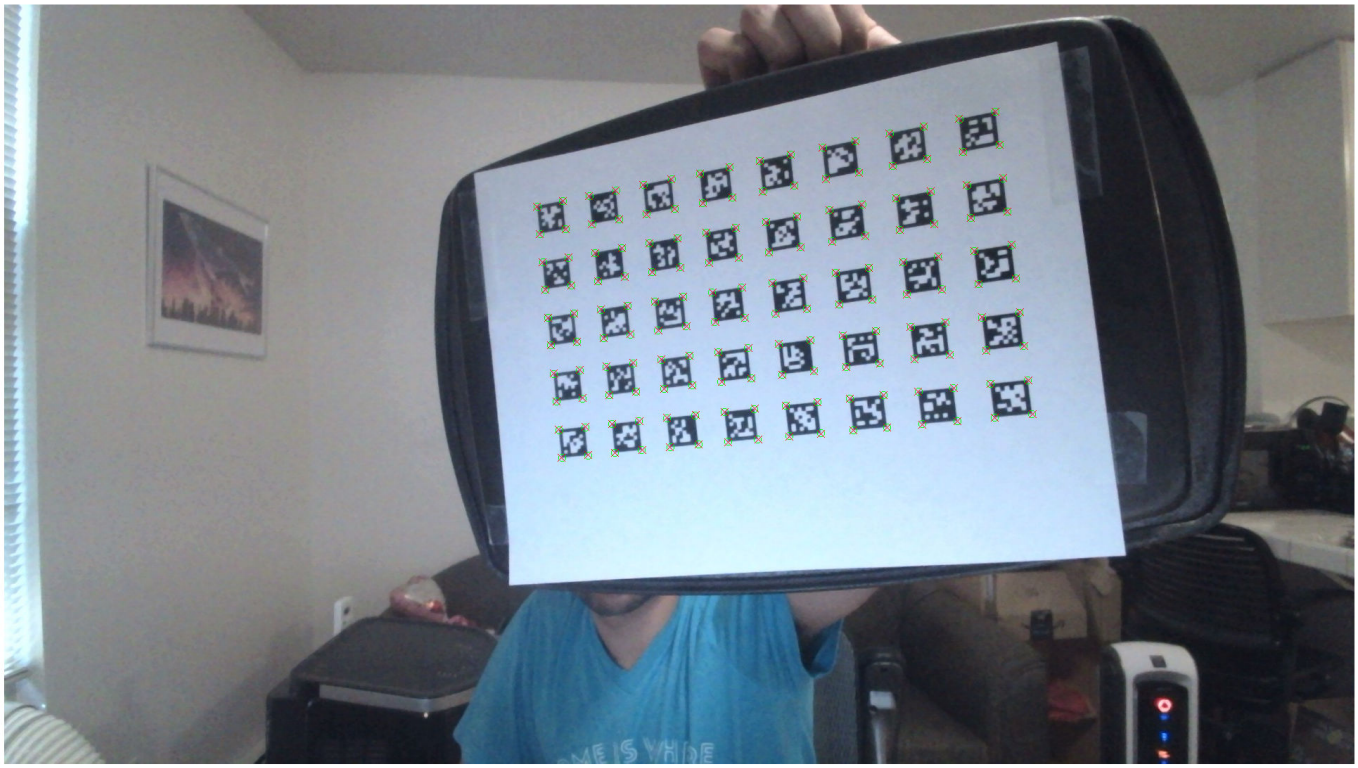



```
% Display the extrinsics.
figure
showExtrinsics(params)
```



Inspect the locations of the detected image points and the reprojected points obtained using the estimated camera parameters.

```
% Read a calibration image.
I = readimage(imdsCalib, 10);

% Insert markers for the detected and reprojected points.
I = insertMarker(I, imagePoints(:,:,10), 'o', 'Color', 'g', 'Size', 5);
I = insertMarker(I, params.ReprojectedPoints(:,:,10), 'x', 'Color', 'r', 'Size', 5);

% Display the image.
figure
imshow(I)
```



Using other Calibration Patterns

While this example uses AprilTags markers in the calibration pattern, the same workflow can be extended to other planar patterns as well. The `estimateCameraParameters` used to obtain the camera parameters requires:

- **imagePoints**: Key points in the calibration pattern in image coordinates obtained from the captured images.
- **worldPoints**: Corresponding world point coordinates of the key points in the calibration pattern.

Provided there is a way to obtain these key points, the rest of the calibration workflow remains the same.

```
% Display params
params
```

```
params =
  cameraParameters with properties:

    Camera Intrinsics
                        Intrinsics: [1×1 cameraIntrinsics]

    Camera Extrinsics
                  RotationMatrices: [3×3×19 double]
                 TranslationVectors: [19×3 double]

    Accuracy of Estimation
              MeanReprojectionError: 0.5642
                 ReprojectionErrors: [160×2×19 double]
                  ReprojectedPoints: [160×2×19 double]
```

```
  Calibration Settings
                       NumPatterns: 19
                 DetectedKeypoints: [160×19 logical]
                       WorldPoints: [160×2 double]
                        WorldUnits: 'mm'
                      EstimateSkew: 0
   NumRadialDistortionCoefficients: 2
      EstimateTangentialDistortion: 0
```

```
params.Intrinsics
```

```
ans =
  cameraIntrinsics with properties:

            FocalLength: [958.9126 956.1364]
         PrincipalPoint: [957.4814 557.8223]
              ImageSize: [1080 1920]
       RadialDistortion: [0.0833 -0.0986]
    TangentialDistortion: [0 0]
                    Skew: 0
          IntrinsicMatrix: [3×3 double]
```

Supporting functions

helperGenerateAprilTagPattern generates an AprilTag based calibration pattern.

```
function calibPattern = helperGenerateAprilTagPattern(imdsTags, tagArragement, tagFamily)

numTags = tagArragement(1)*tagArragement(2);
tagIds = zeros(1,numTags);

% Read the first image.
I = readimage(imdsTags, 3);
Igray = im2gray(I);

% Scale up the thumbnail tag image.
Ires = imresize(Igray, 15, 'nearest');

% Detect the tag ID and location (in image coordinates).
[tagIds(1), tagLoc] = readAprilTag(Ires, tagFamily);

% Pad image with white boundaries (ensures the tags replace the black
% portions of the checkerboard).
tagSize = round(max(tagLoc(:,2)) - min(tagLoc(:,2)));
padSize = round(tagSize/2 - (size(Ires,2) - tagSize)/2);
Ires = padarray(Ires, [padSize,padSize], 255);

% Initialize tagImages array to hold the scaled tags.
tagImages = zeros(size(Ires,1), size(Ires,2), numTags);
tagImages(:,:,1) = Ires;

for idx = 2:numTags

    I = readimage(imdsTags, idx + 2);
    Igray = im2gray(I);
```

```
    Ires = imresize(Igray, 15, 'nearest');
    Ires = padarray(Ires, [padSize,padSize], 255);

    tagIds(idx) = readAprilTag(Ires, tagFamily);

    % Store the tag images.
    tagImages(:,:,idx) = Ires;

end

% Sort the tag images based on their IDs.
[~, sortIdx] = sort(tagIds);
tagImages = tagImages(:,:,sortIdx);

% Reshape the tag images to ensure that they appear in column-major order
% (montage function places image in row-major order).
columnMajIdx = reshape(1:numTags, tagArragement)';
tagImages = tagImages(:,:,columnMajIdx(:));

% Create the pattern using 'montage'.
imgData = montage(tagImages, 'Size', tagArragement);
calibPattern = imgData.CData;

end
```

helperDetectAprilTagCorners detects AprilTag calibration pattern in images.

```
function [imagePoints, boardSize, imagesUsed] = helperDetectAprilTagCorners(imdsCalib, tagArra

% Get the pattern size from tagArrangement.
boardSize = tagArrangement*2 + 1;

% Initialize number of images and tags.
numImages = length(imdsCalib.Files);
numTags = tagArrangement(1)*tagArrangement(2);

% Initialize number of corners in AprilTag pattern.
imagePoints = zeros(numTags*4,2,numImages);
imagesUsed = zeros(1, numImages);

% Get checkerboard corner indices from AprilTag corners.
checkerIdx = helperAprilTagToCheckerLocations(tagArrangement);

for idx = 1:numImages

    % Read and detect AprilTags in image.
    I = readimage(imdsCalib, idx);
    [tagIds, tagLocs] = readAprilTag(I, tagFamily);

    % Accept images if all tags are detected.
    if numel(tagIds) == numTags
        % Sort detected tags using ID values.
        [~, sortIdx] = sort(tagIds);
        tagLocs = tagLocs(:,:,sortIdx);
```

```
        % Reshape tag corner locations into a M-by-2 array.
        tagLocs = reshape(permute(tagLocs,[1,3,2]), [], 2);

        % Populate imagePoints using checkerboard corner indices.
        imagePoints(:,:,idx) = tagLocs(checkerIdx(:),:);
        imagesUsed(idx) = true;
    else
        imagePoints(:,:,idx) = [];
    end

end

end
```

helperAprilTagToCheckerLocations converts AprilTag corners to checkerboard corners.

```
function checkerIdx = helperAprilTagToCheckerLocations(tagArrangement)

numTagRows = tagArrangement(1);
numTagCols = tagArrangement(2);
numTags = numTagRows * numTagCols;

% Row index offsets.
rowIdxOffset = [0:numTagRows - 1; 0:numTagRows - 1];

% Row indices for first and second columns in board.
col1Idx = repmat([4 1]', numTagRows, 1);
col2Idx = repmat([3 2]', numTagRows, 1);
col1Idx = col1Idx + rowIdxOffset(:)*4;
col2Idx = col2Idx + rowIdxOffset(:)*4;

% Column index offsets
colIdxOffset = 0:4*numTagRows:numTags*4 - 1;

% Implicit expansion to get all indices in order.
checkerIdx = [col1Idx;col2Idx] + colIdxOffset;

end
```

Reference

[1] E. Olson, "AprilTag: A robust and flexible visual fiducial system," *2011 IEEE International Conference on Robotics and Automation*, Shanghai, 2011, pp. 3400-3407, doi: 10.1109/ICRA.2011.5979561.