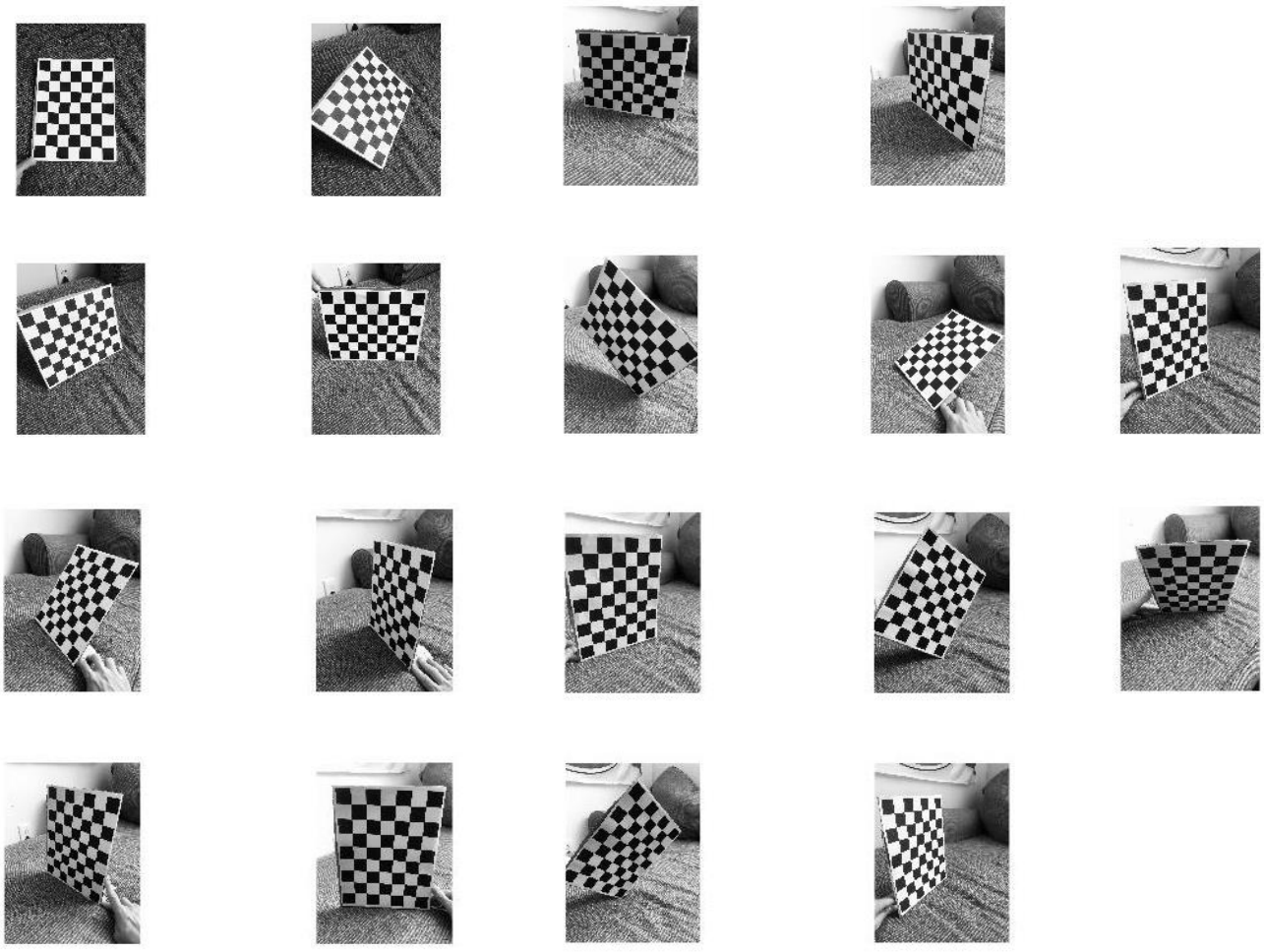
CSCE 590 Introduction to Image Processing

TO: Professor Ioannis Rekleitis
FROM: Adam Einstein

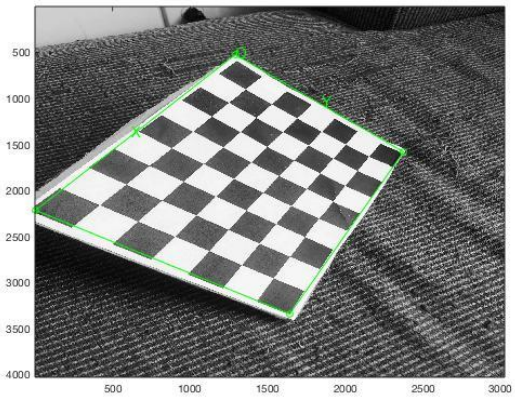
DATE: April 20, 2021
SUBJECT: Assignment 4

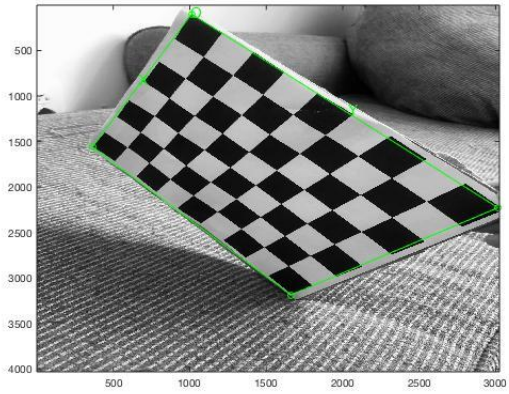
1. Camera Calibration (50.0%)

Using any calibration toolbox (OpenCV or MATLAB) print the recommended calibration target and perform camera calibration for a camera (web cam, cellphone cam, digital point and shoot camera, or DSLR. Ensure that images are captured in all possible orientations to cover the field of view. Produce a write up in the report for the intrinsic parameters together with the error estimates and examples of the images used in the calibration procedure.

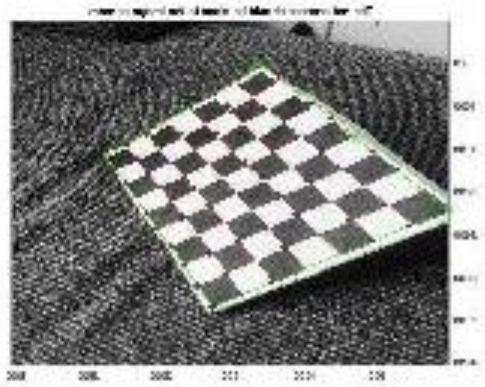


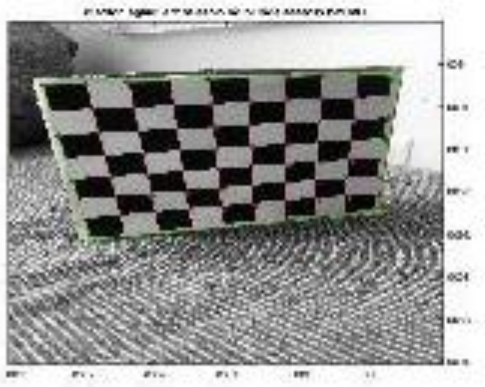
Using my iPhone camera, I took eighteen photos of this checkerboard pattern I printed offline. I made sure to capture as many angles as possible to cover all fields of view of the checkerboard.

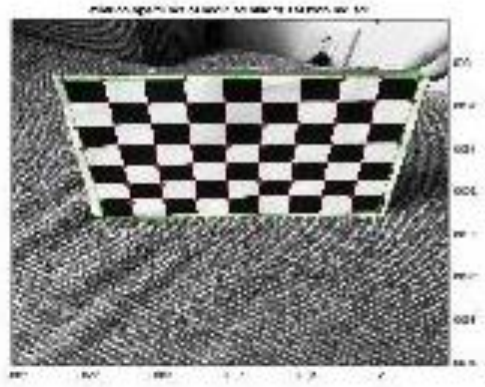


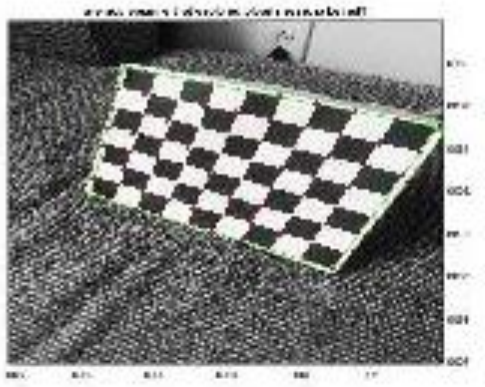


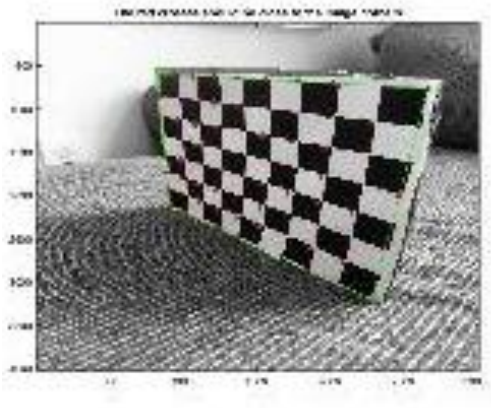
The next step during calibration was labelling the boundaries of the rectangular object in each of the eighteen images. After clicking all four corners you had to confirm that each square was a size of 30mm. This is the default size for a checkerboard square, but I measured the squares to be sure that printing it out didn't alter the size. After that it asked how many squares ran along the x-axis, and then how many squares were on the y-axis. After these numbers these next images were produced.

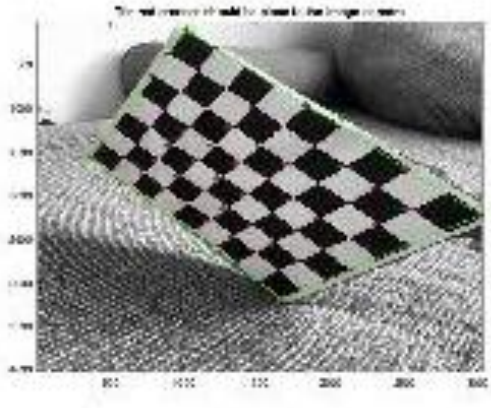


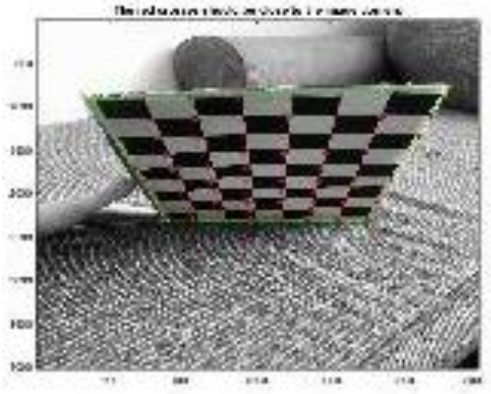






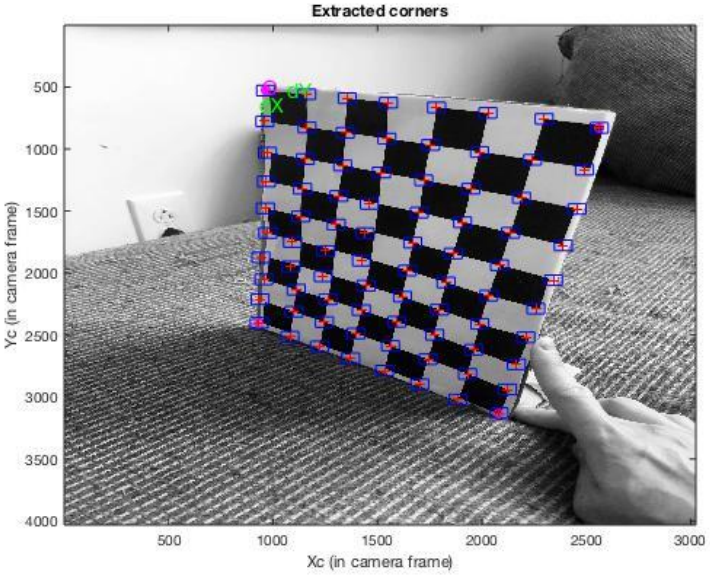


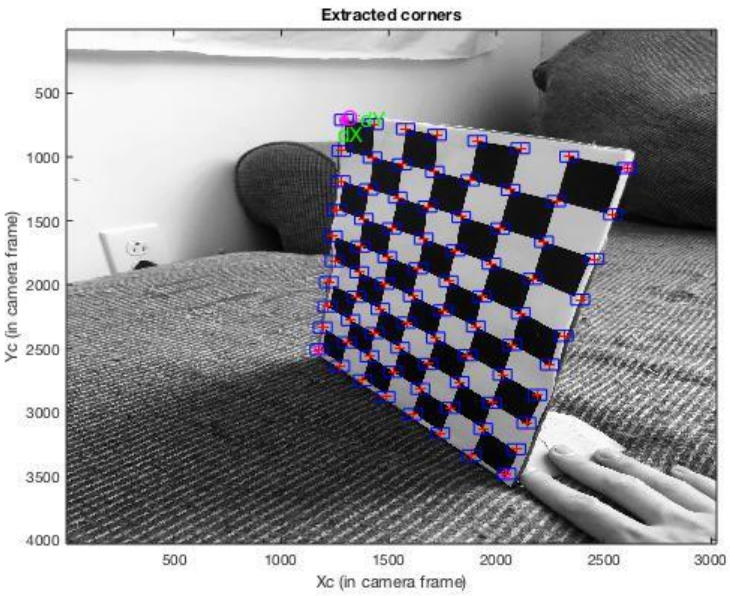


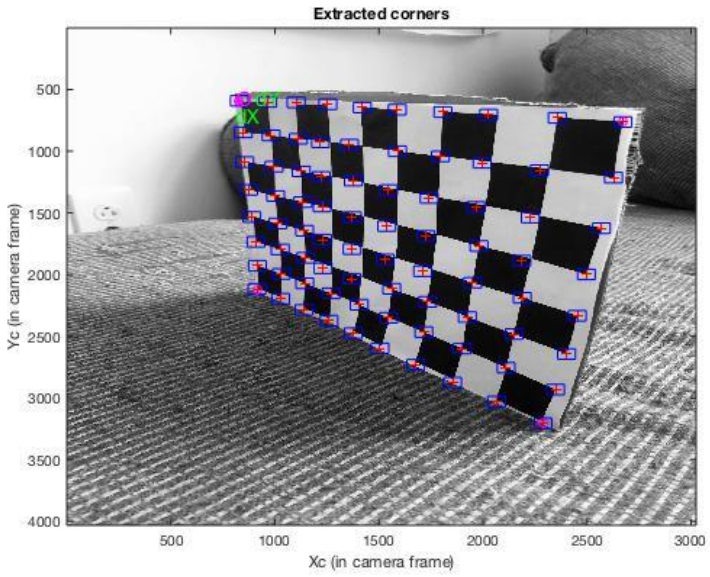


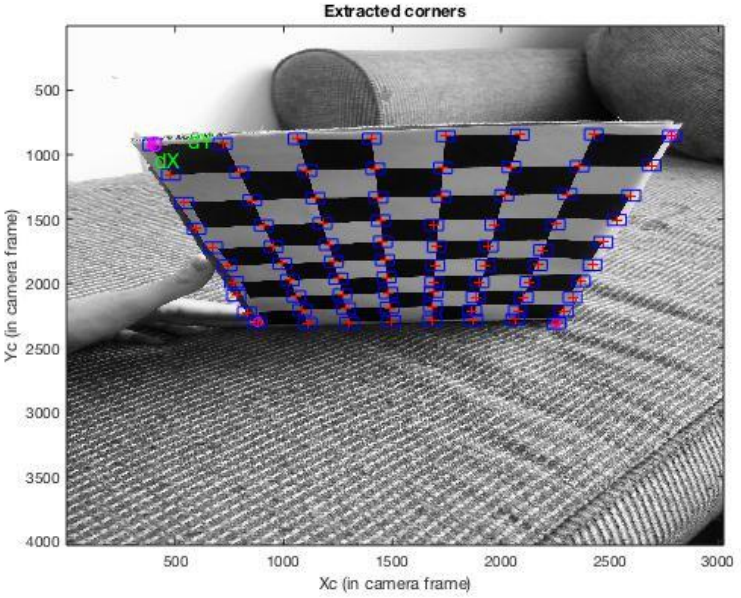
These images display the predicted grid corners of the checkerboard given the four identified corners, single square size in millimeters, and square length of the x-axis and y-axis. In most cases this software was able to predict the corners accurately enough, but it definitely depended on how accurately I place the boundary region for the edges of the checkerboard. As you can see in the last predicted image above the red grid corners are not very close because the boundary was not as accurate as the others.

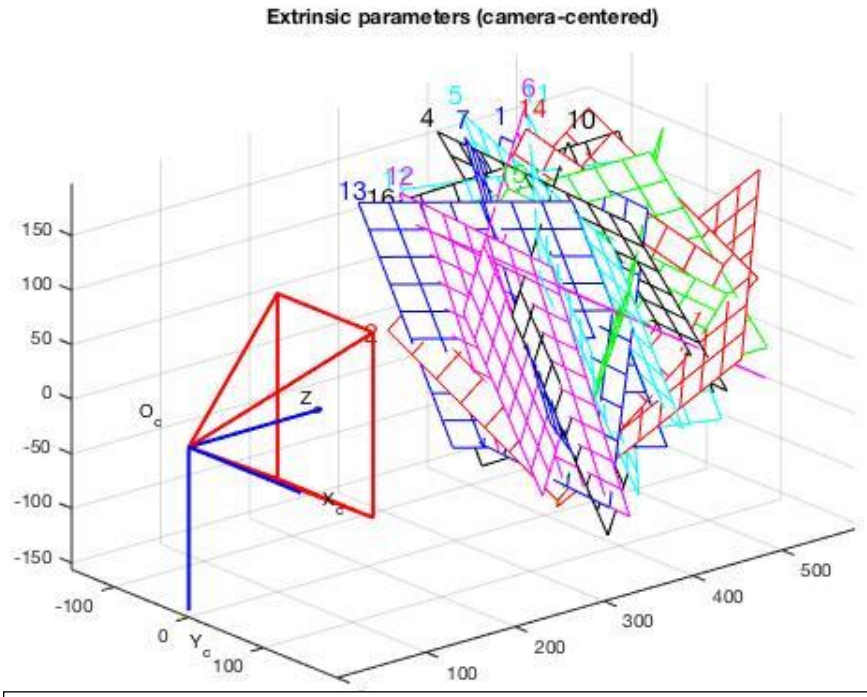
After this step in calibration the software then extracts the corners of the image if you accept the image that was produced with the red crosses above. Below in the extracted corner images, the blue squares identify the limits to the corner finder window.


Extracted corners
Yc (in camera frame)
Xc (in camera frame)
500
1000
1500
2000
2500
3000
3500
4000


Extracted corners
Yc (in camera frame)
Xc (in camera frame)
500
1000
1500
2000
2500
3000
3500
4000


Extracted corners
Yc (in camera frame)
Xc (in camera frame)
500
1000
1500
2000
2500
3000
3500
4000


Extracted corners
Yc (in camera frame)
Xc (in camera frame)
500
1000
1500
2000
2500
3000
3500
4000



The extrinsic parameters display the relative positions of each checkerboard image that was taken from my iPhone

After performing all the corner extractions for all eighteen images I ran the main camera calibration procedure. The first step of calibration is initialization, and second is optimization. Optimization is designed to minimize the total reprojection error in all calibrations. Below are captured images of the completed calibration windows. It shows the calibration initialization results as well as optimization results.

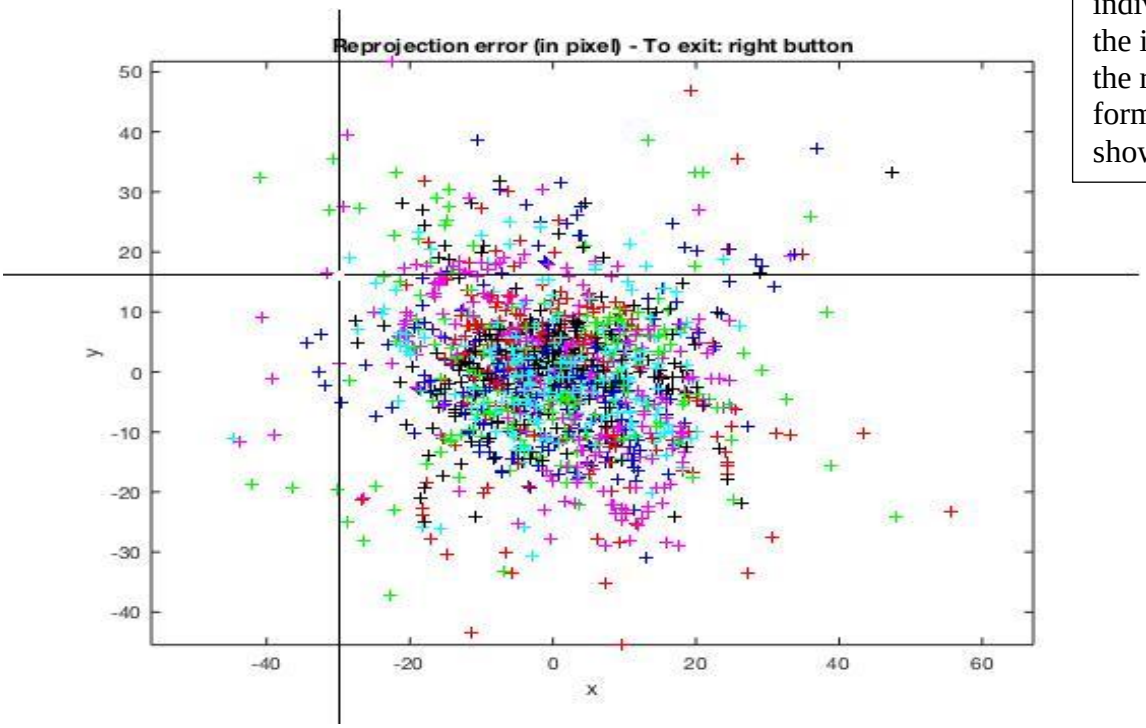
Editor - /Users/adameinstein/Downloads/MATLAB/590-hw-4/TOOLBOX_calib/calib.m

Command Window

```
Distortion not fully estimated (defined by the variable est_dist):
     Sixth order distortion not estimated (est_dist(5)=0) - (DEFAULT) .
Initialization of the principal point at the center of the image.
Initialization of the intrinsic parameters using the vanishing points of planar patterns.

Initialization of the intrinsic parameters - Number of images: 18


Calibration parameters after initialization:

Focal Length:          fc = [ 3488.02681   3488.02681 ]
Principal point:       cc = [ 1511.50000   2015.50000 ]
Skew:             alpha_c = [ 0.00000 ]   => angle of pixel = 90.00000 degrees
Distortion:            kc = [ 0.00000   0.00000   0.00000   0.00000   0.00000 ]

Main calibration optimization procedure - Number of images: 18
Gradient descent iterations: 1...2...3...4...5...6...7...8...9...10...11...12...13...14...15...16...17...18...19...20...21...22...23...24...
Estimation of uncertainties...done


Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 3536.16529   3561.71251 ] +/- [ 47.96904   45.86011 ]
Principal point:       cc = [ 1417.17979   2110.11884 ] +/- [ 67.96053   83.58869 ]
Skew:             alpha_c = [ 0.00000 ] +/- [ 0.00000  ]   => angle of pixel axes = 90.00000 +/- 0.00000 degrees
Distortion:            kc = [ 0.08209   -0.18421   0.01143   -0.00397  0.00000 ] +/- [ 0.05717   0.18049   0.00927   0.00689  0.00000 ]
Pixel error:          err = [ 12.47398   11.59018 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).
```

Command Window

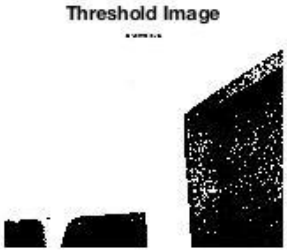
```
Generating the matlab script file Calib_Results.m containing the intrinsic and extrinsic parameters...
done
Pixel error:          err = [ 12.47398   11.59018] (all active images)


Selected image: 16
Selected point index: 77
Pattern coordinates (in units of (dX,dY)): (X,Y)=(6,0)
Image coordinates (in pixel): (438.31,2661.87)
Pixel error = (-21.15797,28.09253)
Window size: (wintx,winty) = (42,42)

Selected image: 17
Selected point index: 1
Pattern coordinates (in units of (dX,dY)): (X,Y)=(0,7)
Image coordinates (in pixel): (1840.88,759.61)
Pixel error = (10.85440,21.25867)
Window size: (wintx,winty) = (42,42)

Selected image: 17
Selected point index: 69
Pattern coordinates (in units of (dX,dY)): (X,Y)=(8,1)
Image coordinates (in pixel): (770.19,2905.97)
Pixel error = (7.36033,3.36183)
Window size: (wintx,winty) = (42,42)
Pixel error:          err = [ 12.47398   11.59018] (all active images)
```

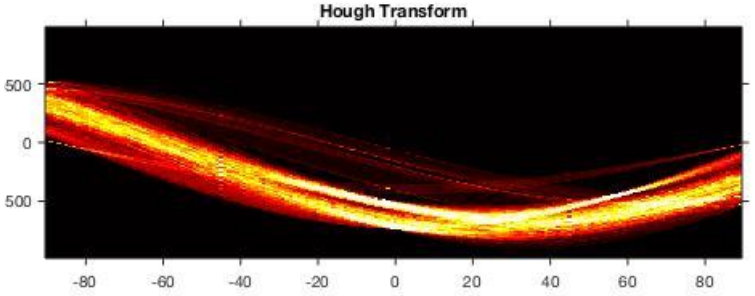


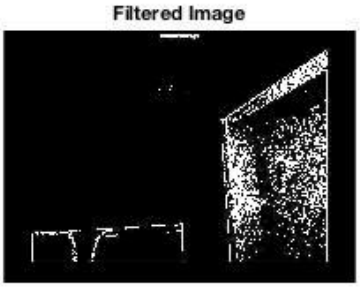
These two figures both display the reprojection error of the calibration. The first figure of the command window prints the pixel error of all images as well as the individual error sizes for a few of the images that were ran. Below the reprojection error is in the form of color-coded crosses shown in a histogram.

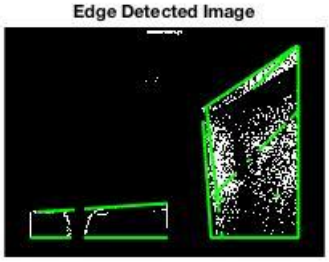
2. Line Detection (50.0%)

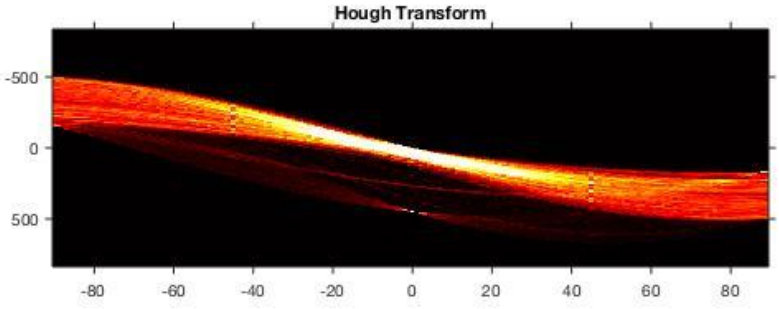
Take some pictures with clearly identifiable lines. Use your edge detector to find edges and threshold the image. Use the Hough Transform to detect lines by first examine the counts of accumulator cell for high pixel concentrations and then for each chosen cell, link the pixels based on the continuity.

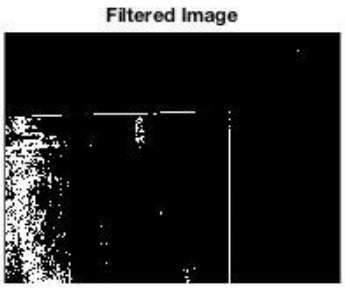


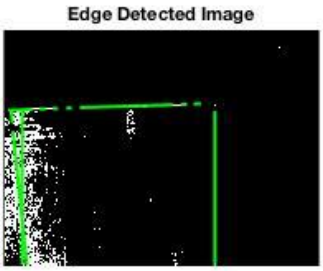






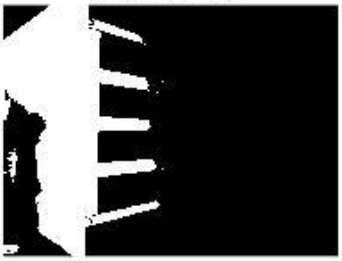




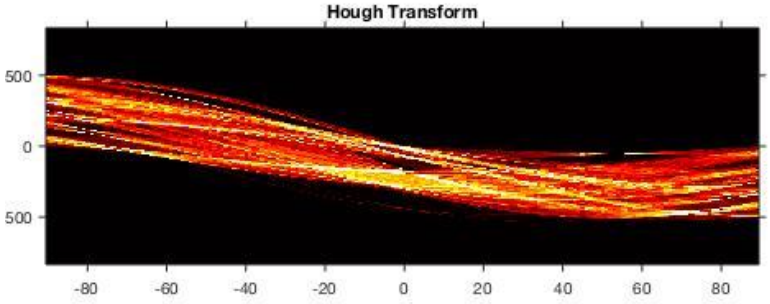


Original Image

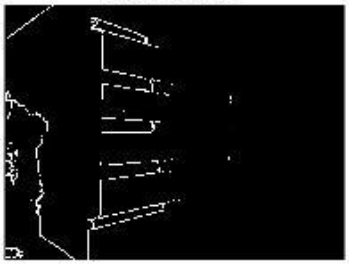
Threshold Image



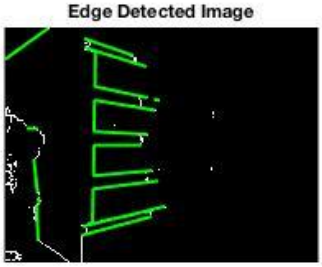
Hough Transform



Filtered Image

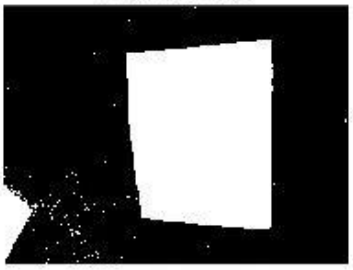


Edge Detected Image

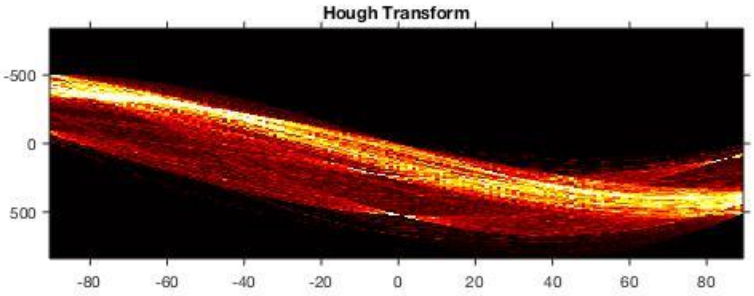


Original Image

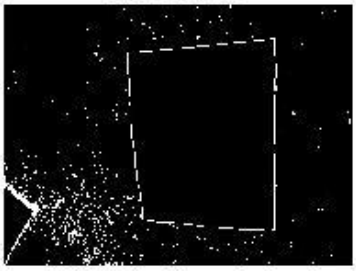
Threshold Image



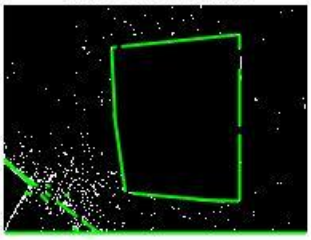
Hough Transform



Filtered Image



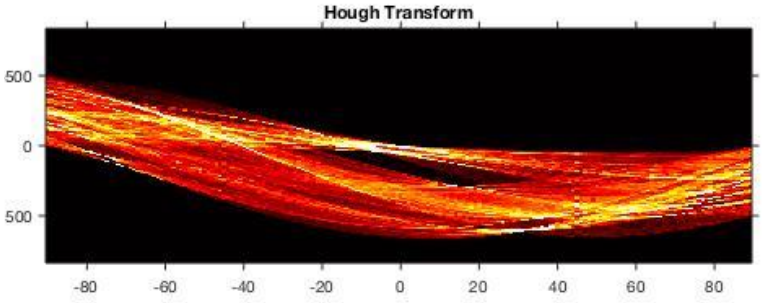
Edge Detected Image



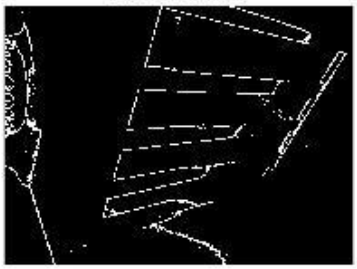
Original Image

Threshold Image

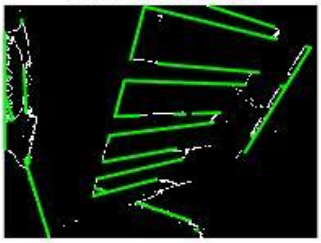
Hough Transform



Filtered Image

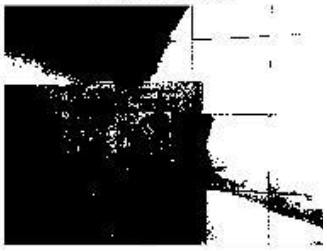


Edge Detected Image

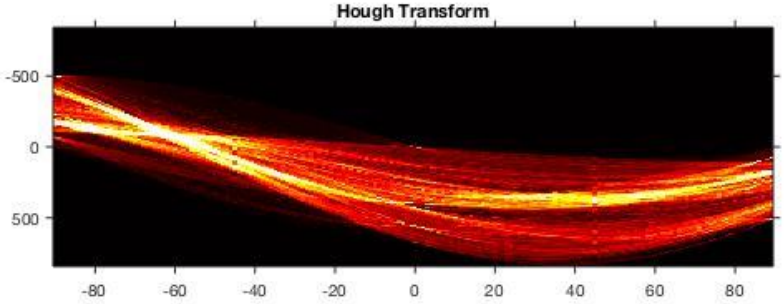


Original Image

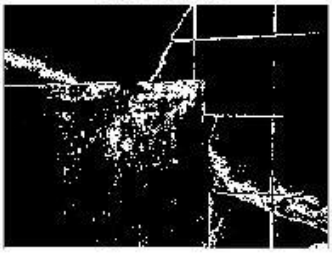
Threshold Image



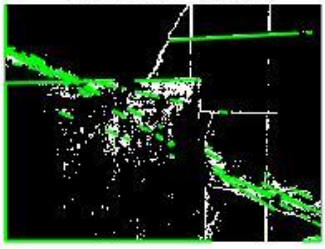
Hough Transform



Filtered Image



Edge Detected Image



Code:

1.

Toolbox: http://www.vision.caltech.edu/bouguetj/calib_doc/

2.

```
function threshholder
B=imread('7.jpg');
V1=hist(B,0:255);
G=reshape(V1,[],1);

Ind=0:255;
I1=reshape(Ind,[],1);
res=zeros(size([1 256]));

for A=0:255
    [weightb,varb]=calculate(1,A);
 [weightf,varf]=calculate(A+1,255);
res(A+1)=(weightb*varb)+(weightf*va
rf);
end
[~,val]=min(res);
threshval = (val-1)/256;
thresh = im2bw(B,threshval);
figure,
imshow(thresh);
title('Threshold Image');
function
[weight,variance]=calculate(m,n)
weight=sum(G(m:n))/sum(G);
v=G(m:n).*I1(m:n);
total=sum(v);
mean=total/sum(G(m:n));
val2=(I1(m:n)-mean).^2;
num=sum(val2.*G(m:n));
variance=num/sum(G(m:n));
end
end
%///
a = imread('4.jpg');
figure
subplot(2,2,1)
imshow(a); title('Original Image');
a = imread('4thresh.jpg');
a = uint8(a);
subplot(2,2,2)
imshow(a); title('Threshold
Image');

a = rgb2gray(a);
a = double(a);
fImage = zeros(size(a));

% Sobel Operator Mask
Mx = [-1 0 1; -2 0 2; -1 0 1];
My = [-1 -2 -1; 0 0 0; 1 2 1];

for i = 1:size(a, 1) - 2
    for j = 1:size(a, 2) - 2

        Gx = sum(sum(Mx.*a(i:i+2,
j:j+2)));
        Gy = sum(sum(My.*a(i:i+2,
j:j+2)));

        fImage(i+1, j+1) =
sqrt(Gx.^2 + Gy.^2);
    end
end

fImage = uint8(fImage);
subplot(2,2,[3,4])
imshow(fImage); title('Filtered
Image');
outImage = im2bw(fImage);
% Hough transform
[H, theta, rho] = hough(outImage);
figure
subplot(2,1,1),
imshow(imadjust(rescale(H)),'XData'
,theta,'YData',rho,'InitialMagnific
ation', 'fit'); title('Hough
Transform');
axis on
axis normal
hold on;
colormap(gca, hot);
A =
houghpeaks(H,10,'threshold',ceil(0.
2*max(H(:))));
lines =
houghlines(outImage,theta,rho,A,'Fi
llGap',10,'MinLength',9);
subplot(2,1,2), imshow(outImage);
title('Edge Detected Image');
hold on
max_len = 0;
for k = 1:length(lines)
    xy = [lines(k).point1;
lines(k).point2];

plot(xy(:,1),xy(:,2),'LineWidth',2,
'Color', 'green');
end
```