Assignment A3: Camera Calibration

Clinton Fernandes

u1016390

02/11/2016

A3

CS 6320

1. Introduction

The purpose of this assignment is to estimate the intrinsic and extrinsic parameters of a camera for the image positions of points, whose positions are known in some fixed world coordinate system. The assignment has two parts, the first one in which we develop a calibration function that takes a corresponding set of image points and world points and produces both the intrinsic and extrinsic calibration parameters. Also, the sensitivity of calibration to noise in the image locations should be studied. In the second part we apply the calibration function to the image provided. Here, we need to develop a method to extract the points from the image.

The following questions can be answered from this exercise:

- How sensitive is the calibration to noise in the image locations?

2. Method

Part 1: develop CS5320_calibrate function

- Input of this function are image and world points (in homogenous coordinates)

Algorithm:

- We know that, to calculate the values of x and y for a point in the camera, we use the following equations:

$$x_i = \frac{m_1(\varepsilon).P_i}{m_3(\varepsilon).P_i} \qquad y_i = \frac{m_2(\varepsilon).P_i}{m_3(\varepsilon).P_i}$$

- Here, $m_i(\varepsilon)^T$ is the i^{th} row of the M matrix.
- ε is the set of parameters including: α, β, θ, x0, y0, R, t
- Each of the above equation yields one equation in m_1, m_2, m_3

$$x_i = \frac{m_1.P_i}{m_3.P_i} \quad => \quad x_i.m_3.P_i = m_1.P_i \quad => \quad m_1.P_i - x_i.m_3.P_i = 0$$

$$(m_1 - x_i.m_3).P_i = 0 \quad => \quad {P_i}^T m_1 + 0^T m_2 - x_i.{P_i}^T.m_3 = 0$$

$=> \ [P_{i1} \ \ P_{i2} \ \ P_{i3} \ \ 1 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ -x_i.P_{i1} \ \ -x_i.P_{i2} \ \ -x_i.P_{i3} \ \ -x_i] * [m]$

Similarly from $y = \frac{m_2.P_i}{m_3.P_i}$, We have

$=> \ [0 \ \ 0 \ \ 0 \ \ 0 \ \ P_{i1} \ \ P_{i2} \ \ P_{i3} \ \ 1 \ \ -y_i.P_{i1} \ \ -y.P_{i2} \ \ -y_i.P_{i3} \ \ -y_i] * [m]$

$$m = \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ , \\ , \\ \\ m_{33} \\ m_{34} \end{bmatrix}$$

- Put all equations in matrix P, Pm=0 and solve
- Find Eigen values of $P^T P$ then the Eigen values associated with the smallest eigenvalue is m solution.

  [V, D] = eigs ($P^T P$, 12)

  [v_old, indexes] = sort(diag(D))

  index = indexes (1)

  m = [V (1:4, index)'; V (5:8, index)'; V (9:12, index)'];

- now write M = (A, b)

  $a_1{}^T = A(1,1:3) \quad a_2{}^T = A(2,1:3) \quad a_3{}^T = A(3,1:3)$

- using the following equations, we can find the parameters:

  $\rho = 1/[\![a_3]\!]$

  $r_3 = \rho a_3$

  $x_0 = \rho^2(a_1.a_3) \qquad\qquad y_0 = \rho^2(a_1.a_3)$

  $\theta = cos^{-1}(-\frac{(a_1 \times a_3)\cdot(a_2 \times a_3)}{[\![a_1 \times a_3]\!][\![a_2 \times a_3]\!]})$

  $\alpha = \rho^2[\![a_1 \times a_3]\!]sin\theta \qquad\qquad \beta = \rho^2[\![a_2 \times a_3]\!]sin\theta$

  $r_1 = \frac{(a_2 \times a_3)}{[\![a_2 \times a_3]\!]}$

  $r_1 = r_3 \times r_1$

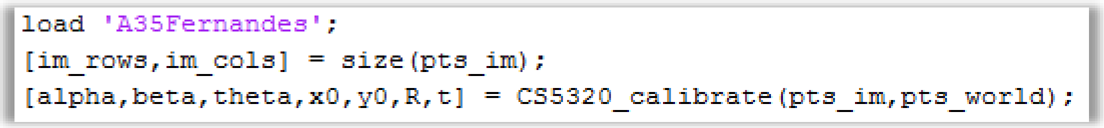
  $$K = \begin{bmatrix} \alpha & -\alpha cot\theta & x_0 \\ 0 & \beta/sin\theta & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

  $t = \rho K^{-1} b$ where b= M (1:3,4)

Part 1: Study of sensitivity of calibration to noise (Gaussian) in the image locations.

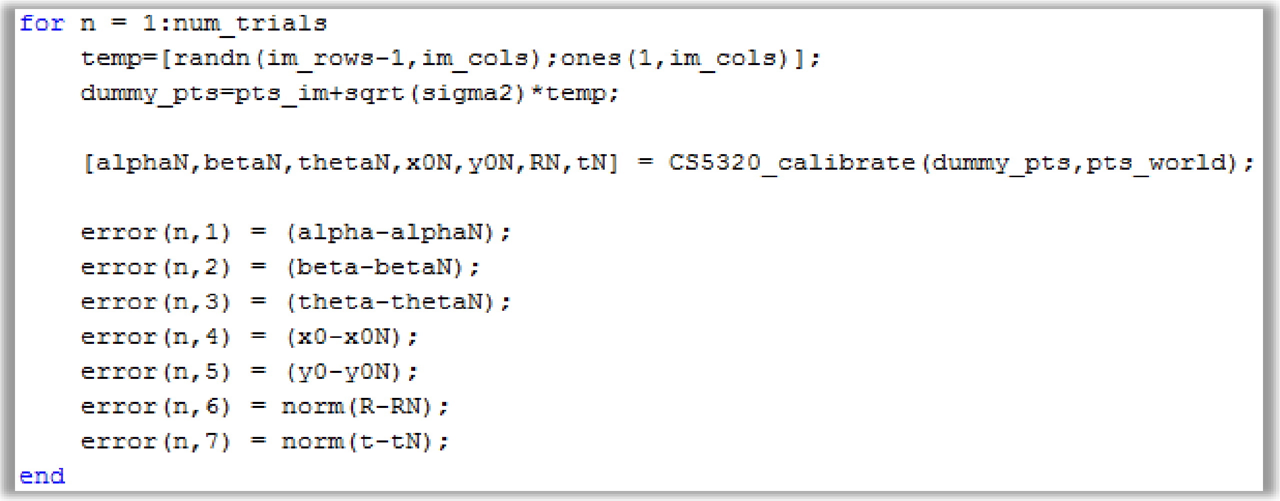
Here, the sensitivity of calibration can be studied by adding noise to the image locations and observing the mean error in the parameters for different number of trials.

- We can first load the image and world point data. Then, run the calibrate function for this data and save the output parameters.



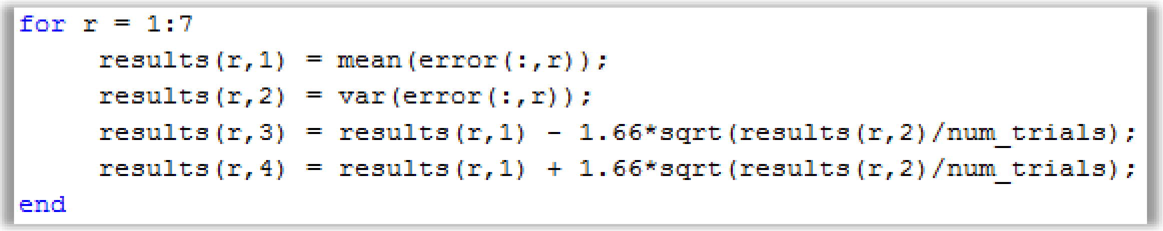
```
load 'A35Fernandes';
[im_rows,im_cols] = size(pts_im);
[alpha,beta,theta,x0,y0,R,t] = CS5320_calibrate(pts_im,pts_world);
```

- Now, for each of the 100 trials, create a matrix of noise using randn(2,n) for n locations and add noise to the original image locations.
- Then run the calibrate function for the new image locations and world points and store the output parameters by a different name like alphaN, betaN.
- Compute error for each parameter by comparing the old and new parameters.
- Repeat for 100 trials.



```
for n = 1:num_trials
    temp=[randn(im_rows-1,im_cols);ones(1,im_cols)];
    dummy_pts=pts_im+sqrt(sigma2)*temp;

    [alphaN,betaN,thetaN,x0N,y0N,RN,tN] = CS5320_calibrate(dummy_pts,pts_world);

    error(n,1) = (alpha-alphaN);
    error(n,2) = (beta-betaN);
    error(n,3) = (theta-thetaN);
    error(n,4) = (x0-x0N);
    error(n,5) = (y0-y0N);
    error(n,6) = norm(R-RN);
    error(n,7) = norm(t-tN);
end
```

- Compute mean error, variance and confidence interval at 95% confidence level for 100 trials.



```
for r = 1:7
     results(r,1) = mean(error(:,r));
     results(r,2) = var(error(:,r));
     results(r,3) = results(r,1) - 1.66*sqrt(results(r,2)/num_trials);
     results(r,4) = results(r,1) + 1.66*sqrt(results(r,2)/num_trials);
end
```

- Also, the mean error in the parameters can be checked by changing the variance of Gaussian noise added to the image locations.
- A plot of mean error in parameters vs variance (0 to 0.9) will help to study the sensitivity.

Part 2: Apply calibration function to the provided image and develop a method to extract the points from the image.

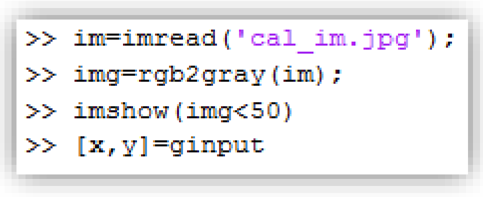
We can extract the points from the image in the following manner:

First, upload/read the image using imread('cal_im.jpg') function. Then convert this RGB image to grayscale using rgb2gray function. Then call imshow(img<50) to clearly observe the edges/corners of the speaker in the image.

Now, we can extract the points from the image using the [x,y]=ginput() function of Matlab. i.e., Graphical input from mouse or cursor.

[x,y] = ginput enables you to identify n points from the current axes and returns their x- and y-coordinates in the x and y column vectors. Press the Return key to terminate the input after clicking onto the 6 required locations in the image (i.e., the corners of the speaker)

And we can now apply the calibration function using the world points provided and the computed image points as the input.



```
>> im=imread('cal_im.jpg');
>> img=rgb2gray(im);
>> imshow(img<50)
>> [x,y]=ginput
```

Figure 1: Method to extract points from the image.

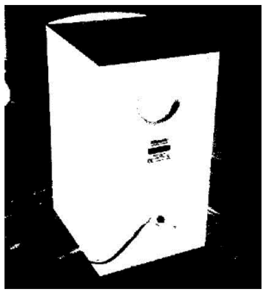


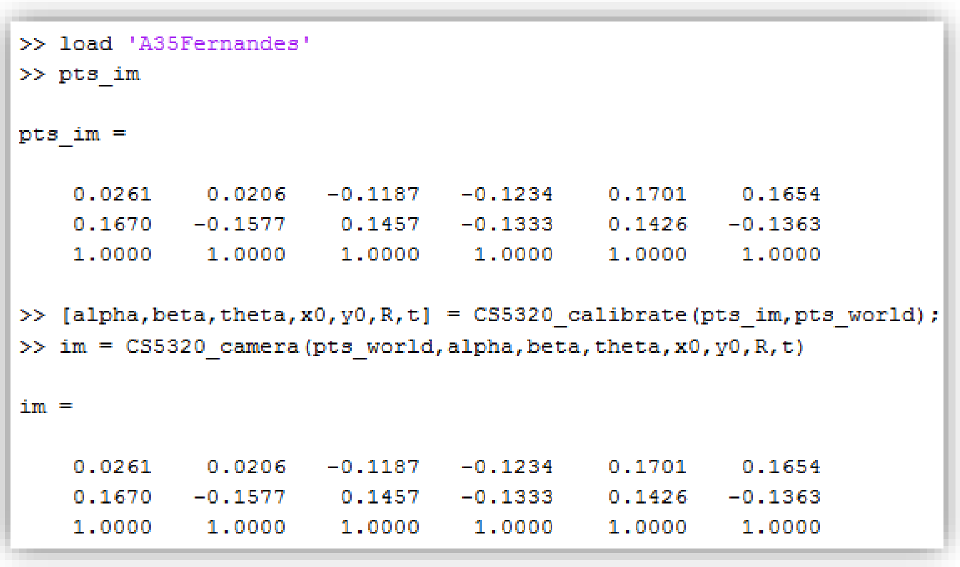
Figure 2: imshow(img<50)

3. Verification

To verify that the CS5320_calibrate function works, we will use the given image and world points as the input to the function and get the output i.e., the set of parameter values.

Then we will use these parameters values in the CS5320_camera function of the assignment A1. With these parameters and the world points, we will get the image points as output of the camera function.

Then we will compare the image points obtained from the camera function and the initially provided image points and verify whether the CS5320_calibrate function works properly.



```
>> load 'A35Fernandes'
>> pts_im

pts_im =

    0.0261    0.0206   -0.1187   -0.1234    0.1701    0.1654
    0.1670   -0.1577    0.1457   -0.1333    0.1426   -0.1363
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000

>> [alpha,beta,theta,x0,y0,R,t] = CS5320_calibrate(pts_im,pts_world);
>> im = CS5320_camera(pts_world,alpha,beta,theta,x0,y0,R,t)

im =

    0.0261    0.0206   -0.1187   -0.1234    0.1701    0.1654
    0.1670   -0.1577    0.1457   -0.1333    0.1426   -0.1363
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
```

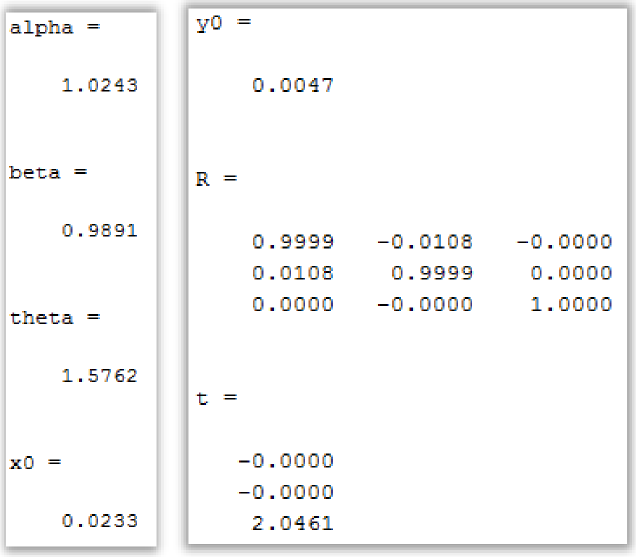
Figure 3: Comparison of provided image points and points obtained from CS5320_camera

We can see from the above figure that the image points obtained from the camera function and the initially provided image points match. Hence, it is verified that the CS5320_calibrate function works.

4. Data

Part 1: CS5320_calibrate

When we use the specific image-world point set found in. mat file (‘A35Fernandes’) provided in Canvas, as an input to the CS5320_calibrate function, we get the following parameters as the output:



```
alpha =

    1.0243


beta =

    0.9891


theta =

    1.5762


x0 =

    0.0233
```

```
y0 =

    0.0047


R =

    0.9999   -0.0108   -0.0000
    0.0108    0.9999    0.0000
    0.0000   -0.0000    1.0000


t =

   -0.0000
   -0.0000
    2.0461
```

Figure 4: Output of CS5320_calibrate function for inputs of pts_im and pts_world.

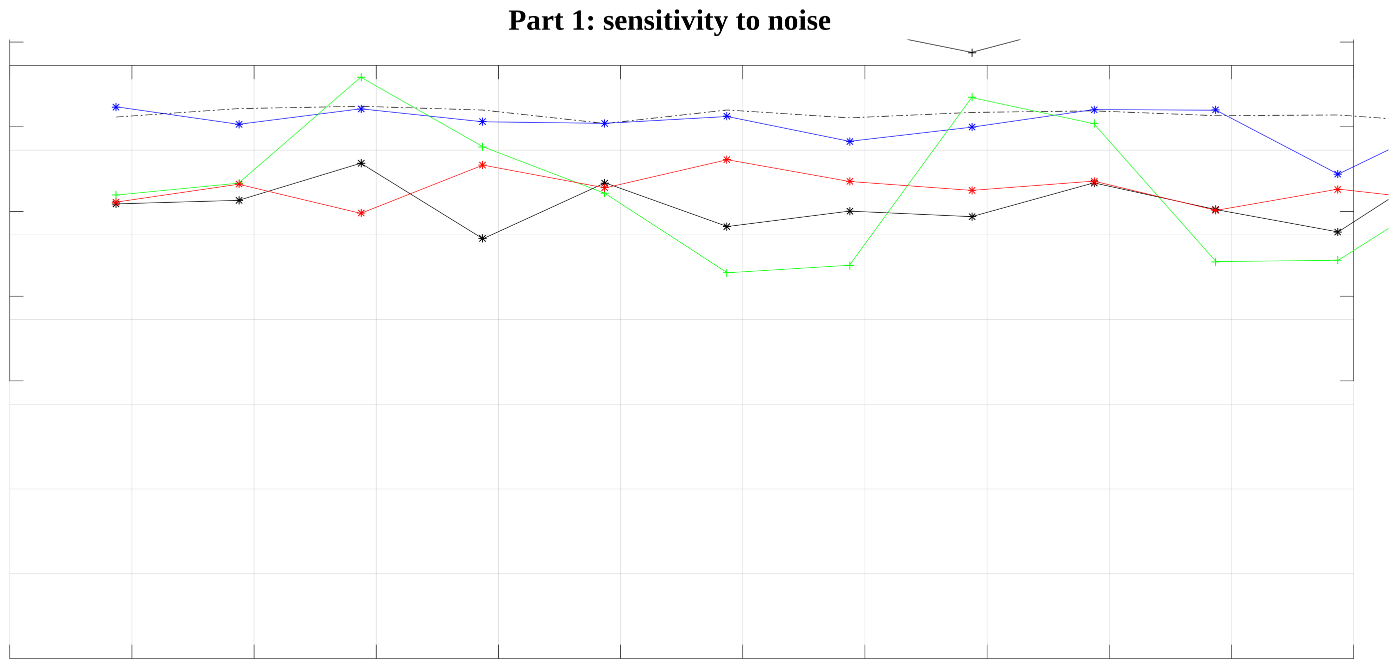


Figure 5: Mean error vs Number of trials (10 to 21) plot of the parameters.

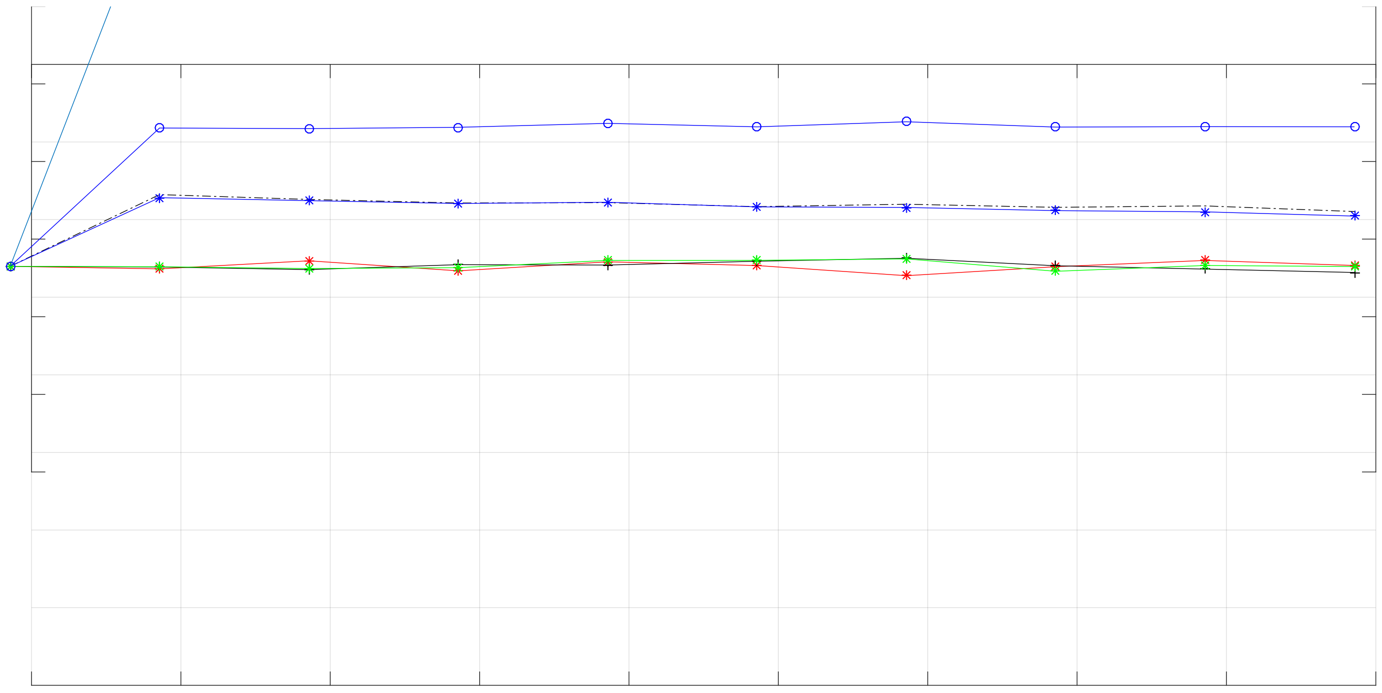
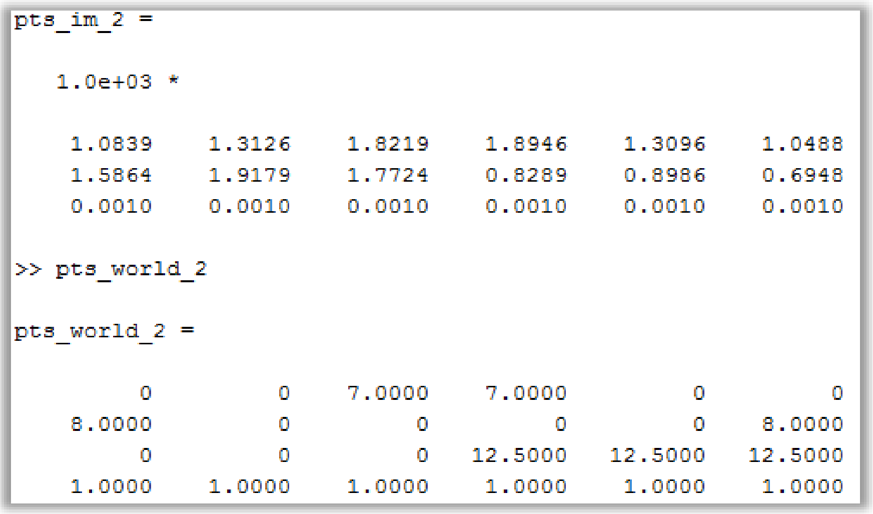


Figure 6: Mean error vs Variance in the Gaussian noise.

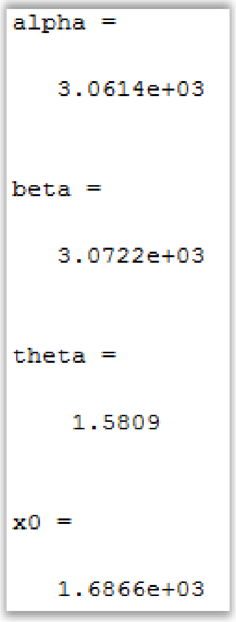
Part 2: Applying calibration function to the provided image.

Here, the required points from the image can be extracted using the ginput function in Matlab. The extracted points were saved in pts_im_2 as a 3x6 array (homogenous coordinates). Also the world points for the features of this image (corners of the speaker) were saved in pts_world_2.

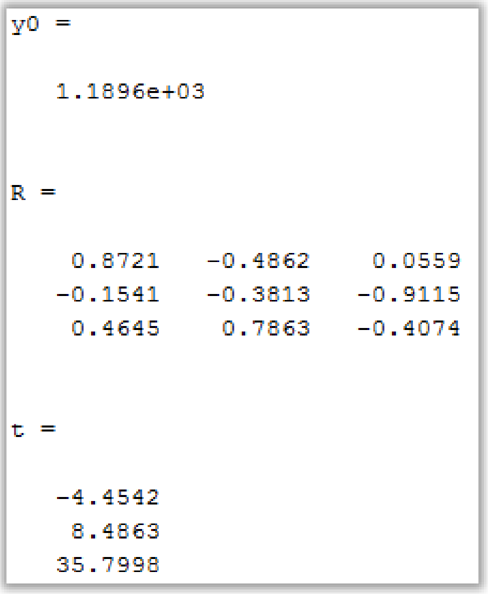


```
pts_im_2 =

   1.0e+03 *

    1.0839    1.3126    1.8219    1.8946    1.3096    1.0488
    1.5864    1.9179    1.7724    0.8289    0.8986    0.6948
    0.0010    0.0010    0.0010    0.0010    0.0010    0.0010

>> pts_world_2

pts_world_2 =

         0         0    7.0000    7.0000         0         0
    8.0000         0         0         0         0    8.0000
         0         0         0   12.5000   12.5000   12.5000
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
```

Figure 7: contents of pts_im_2 and pts_world_2



```
alpha =

   3.0614e+03


beta =

   3.0722e+03


theta =

    1.5809


x0 =

   1.6866e+03
```



```
y0 =

   1.1896e+03


R =

    0.8721   -0.4862    0.0559
   -0.1541   -0.3813   -0.9115
    0.4645    0.7863   -0.4074


t =

   -4.4542
    8.4863
   35.7998
```

Figure 8: Output of CS5320_calibrate function for inputs of pts_im_2 and pts_world_2.

5. Analysis

	mean error	variance	low conf int	up conf int
alpha	0.909436	0.017728	0.887334	0.931538
beta	0.864001	0.016927	0.842403	0.885598
theta	-0.04795	0.82426	-0.19866	0.10276
x0	0.029056	0.107507	-0.02537	0.083484
y0	0.03396	0.08709	-0.01503	0.082948
R	1.803145	0.12161	1.745256	1.861033
t	3.674354	1.09268	3.500832	3.847877

Table 1: Statistics for variance 0.2 and 100 trials

6. Interpretation

It can be seen from the above plot of error vs mean number of trials (Figure 5), that there is no particular trend in the error of any of the 7 parameters. Although the plotted error is from to 10 to 21 trials, the plot for 1-100 trials also resembles the same, where the pattern of the change in the error with the trials is random.

However, from the plot and also from the table 1 which gives the statistics, it can be seen that the mean error in the translation parameter is much higher than compared to the other parameters. In this set of 100 trials it was found that the maximum value of the translation error was about 8, sometimes, for another 100 different trials, the error reaches to even 90 or so.

Error in the alpha and the beta parameter followed a similar pattern with increasing number of trials.

Error in x0, y0 and theta were the lowest amongst the other parameters. and were centered about zero. Their values of mean error were also found to be about zero.

Looking at the plot of mean error vs variance in the Gaussian noise (figure 6), it can be seen that mean error for x0, y0 and theta would fluctuate about zero with the increase in variance.

Mean error in alpha and beta was within 1, but it was found to be decreasing at a slow rate with increase in variance. Error in Rotation matrix was almost constant.

While the error curve for translation was found to be random.

7. Critique

As it was found that the error in certain parameters was decreasing with the increase in the variance of noise and also for some parameters, it was constant, it would be better to investigate what might be causing this trend.

Adding a uniformly distributed noise (using rand function in matlab) in instead of Gaussian (randn), it was found that the error in the parameters, for example, alpha, was increasing with increase in variance, as opposed to the decreasing trend which was seen when randn was used.

Also, using other methods to compute the error in the rotation parameter can be looked at. The error may be calculated in the following ways:

$$\text{Error} = \text{norm}\,(\text{R\_old} - \text{R\_new})$$

OR

$$\text{Error} = \text{sqrt}\,(\text{trace}((\text{R\_old} - \text{R\_new}) * (\text{R\_old} - \text{R\_new})^{T}\ )$$

8. Log

Coding/debugging: 18 hrs

Report: 4hrs