# Assignment 4: Camera Calibration

Ana Maria Martinez Sidera
Department of Computer Science
Illinois Institute of Technology

November 19, 2017

## 1. Problem statement

The problem that we are going to solve is implementation of a camera calibration algorithm, estimating the camera matrix parameters under the assumption of noisy data. We are going to implement non-coplanar calibration. Successive we will use a Ransac algorithm to eliminate the outliers, therefore we will try to obtain a robust estimation.The steps we are going to take to perform the problem are:

1. Extract features points from the calibration target and show them on the image. Save the image points detected in a file. With the 2D points, we should enter the 3d points of corresponding 2D points manually in a file.

2. We have the 3D points and 2D points form the first step and we are going to estimate the projection matrix with this points. We are going to use a non-planar calibration so we will need just one point for it. To know how good is our estimated project matrix we are going to compute the MSE of the 2D points estimated and the real 2D points. The 2D points estimated are obtained from computed the projection matrix with the 3D points.

3. To perform a better estimated projection matrix we are going to look for outliers in our data using Ransac algorithm. To execute the Ransac algorithm we must iterate over a number n different points of our system, calculate the parameters estimate, calculate the error we obtain with these paramaters and at the end of an iterative number of times we will extract the model that has the least error.

   (a) We will perform this step using different data points. They will have noisy data so we could see how the Ransac is executed.

## 2. Proposed solution

1. Program: In this part we have to read all the images of the folder where we execute the code and convert them to images in gray. Therefore, we have to find pattern in chess board. We use the function, cv2.findChessboardCorners(). We also need to pass what kind of pattern we are looking, like 8x8 grid, 5x5 grid etc. It returns the corner points and retval which will be True if pattern is obtained. Once we find the corners, we can increase their accuracy using cv2.cornerSubPix(). We can also draw the pattern using cv2.drawChessboardCorners().Then with all the points obtained we will save them in a txt file where each line will be a point with two coordinates.

```
    gray = cv2.cvtColor(imagen_final,cv2.COLOR_BGR2GRAY)
    ret, corners = cv2.findChessboardCorners(gray, (7,6),None)
    fo = open("corners.txt", "w")
    if ret == True:
```

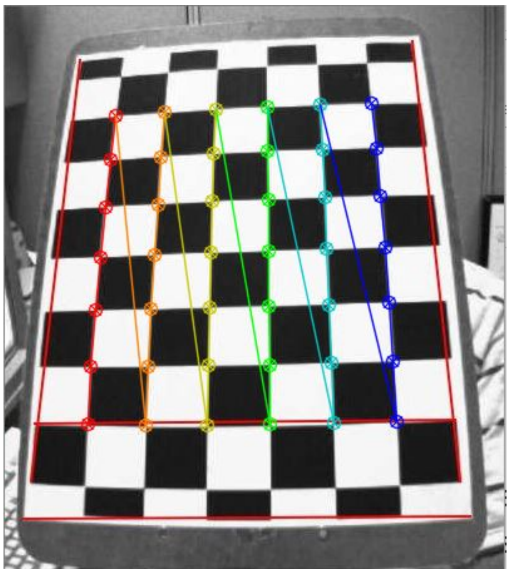
Figure 1: Image with corners draw.

```
        objpoints.append(objp)

        corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
        imagen_final = cv2.drawChessboardCorners(imagen_final, (7,6), corners2,ret)
    for item in corners2:
        fo.write("%s\n" %item)
    fo.close()
```

In order to know the 3D points we visualize the points along with the row number and we put the third coordinate by hand in the file.

```
    fig, ax = plt.subplots()
    ax.scatter(imgpointsx, imgpointsy)

    for i, txt in enumerate(names):
        ax.annotate(int(txt), (imgpointsx[i],imgpointsy[i]))
```

2. Program: In this part of the program we will estimate a projection matrix from the 3D points and the 2D points of our photo. To do this, we must first read all the files and pass them as a float-type parameter to our program.

```
            def read_txt(name,number):
                result = []
                resultx = []
                resulty = []
                file = open(name, "r")
                for line in file:
                    a = [x for x in line.split()]
                    if number == 1:
                        result.append([float(a[0]), float(a[1]), float(a[2]), 1])
```

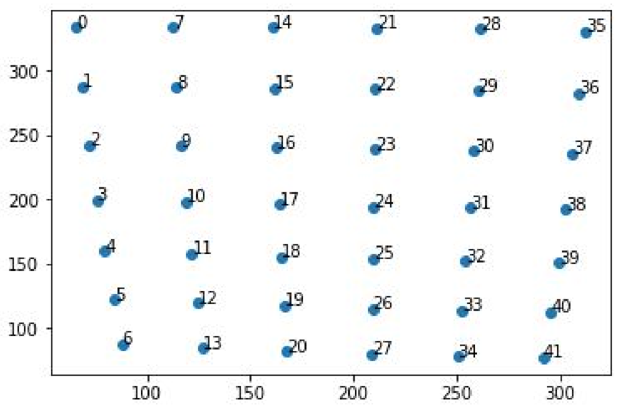

Figure 2: Points with row number label.

```
            elif number == 0:
                result.append([float(a[0]), float(a[1])])
                resultx.append([float(a[0])])
                resulty.append([float(a[1])])
        file.close()
        return result, resultx, resulty
```

When we have at least 6 points 3D points and 2D points we can calculate the projection matrix of our system. For this we calculate the SVD of the matrix A. In this case the matrix A has for each two rows these parameters:

$$[Px, Py, Pz, 1, 0, 0, 0, 0, -uPx, -uPy, -uPz, -u]$$
$$[0, 0, 0, 0, Px, Py, Pz, 1, -vPx, -vPy, -vPz, -v]$$

```
def calculateparameters(worldpoints, image_points):

    Q = np.zeros(shape=(len(worldpoints)*2,12))
    j=0
    for i in range(0,len(worldpoints)):
        Q[j]= [worldpoints[i][0],
                worldpoints[i][1],
                worldpoints[i][2],
                1,
                0,0,0,0,
                -image_points[i][0] * worldpoints[i][0],
                -image_points[i][0] * worldpoints[i][1],
                -image_points[i][0] * worldpoints[i][2],
                -image_points[i][0]]
        Q[j+1] =  [0,0,0,0,
                    worldpoints[i][0],
                    worldpoints[i][1],
                    worldpoints[i][2],
                    1,
                    -image_points[i][1] * worldpoints[i][0],
                    -image_points[i][1] * worldpoints[i][1],
                    -image_points[i][1] * worldpoints[i][2],
                    -image_points[i][1]]
        j+= 2
    Q = np.dot(Q.T, Q)
    U, s, V = np.linalg.svd(Q, full_matrices=True)
```

We will keep the last row of the matrix V. We will have to create a 3x4 matrix with these parameters and we can calculate our non-planar parameters.

```
        for j in range(0,3):
            for k in range(0,4):
                M[j][k]= V[i][11]
                i += 1
```

$$M = \begin{pmatrix} a1_1 & a1_2 & a1_3 & b1_1 \\ a2_1 & a2_2 & a2_3 & b1_2 \\ a3_1 & a3_2 & a3_3 & b1_3 \end{pmatrix}$$

```
        rho = 1/np.linalg.norm(M[2,0:3])
        a1 = M[0,0:3]
        a2 = M[1,0:3]
        a3 = M[2,0:3]
        b = M[0:3, 3].reshape(3,1)
```

$$u_o = |\rho|^2 a_1 \cdot a_3 \qquad v_o = |\rho|^2 a_2 \cdot a_3$$

```
        u0 = np.abs(rho)**2*np.dot(a1, a3)
        v0 = np.abs(rho)**2*np.dot(a2, a3)
```

$$\alpha_v = \sqrt{\rho|^2 a_2 \cdot a_2 - v_o^2} \qquad s = \frac{|\rho|^4}{\alpha_v(a_1 \times a_3) \cdot (a_2 \times a_3)}$$

$$\alpha_u = \sqrt{\rho|^2 a_1 \cdot a_1 - u_o^2}$$

```
        alphav = np.sqrt(np.abs(rho)**2*(np.dot(a2, a2))-v0**2)
        s = (np.abs(rho)**4)/alphav*np.dot(np.cross(a1, a3),np.cross(a2, a3) )
        alphau = np.sqrt(np.abs(rho)**2*np.dot(a1, a1)-s**2-u0**2)
```

$$r_3 = a_3 \qquad r_1 = \frac{|\rho|^2}{\alpha_v(a_2 \times a_3)} \qquad r_2 = r_3 \times r_1$$

```python
        r1 = np.cross(a2, a3)  / np.linalg.norm(np.cross(a2, a3) )
        r3 = a3
        r2 = np.cross(r3, r1)
```

$$K = \begin{pmatrix} alphau & s & u0 \\ 0 & alphav & v0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$M = \begin{pmatrix} r1_1 & r2_1 & r3_1 & t1_1 \\ r1_2 & r2_2 & r3_2 & t1_2 \\ r1_3 & r2_3 & r3_3 & t1_3 \end{pmatrix}$$

```python
        K = np.matrix([[alphau, s, u0],[0,   alphav, v0],[0, 0, 1]])
        invK = np.linalg.inv(K)
        sigma = np.sign(b[2])
        b = np.array([b[0][0],b[1][0],b[2][0]]).reshape(1,3)
        t = sigma*rho*np.dot(invK,b[0]).tolist()

        extrinsicMatrix = np.matrix([[r1[0], r2[0], r3[0], t[0][0]],
                [r1[1], r2[1], r3[1], t[0][1]],
                [r1[2], r2[2], r3[2], t[0][2]]])
```

To finish we have to see how good our estimation is. For this we are going to project the 3D points with the projection matrix that we have obtained. With the estimated 2D points and the 2D real points we can obtain an MSE that will tell us how well our estimation has been.

```python
        def estimate_2dpoints(matrix,worldpoints, name):
            image_pointsestimated = []
            image_points2x = []
            image_points2y = []
            file = open(name, "w")
            for i in range(0,len(worldpoints)):
                result = np.dot(matrix, worldpoints[i]).tolist()
                image_points2x.append(result[0][0]/result[0][2])
                image_points2y.append(result[0][1]/result[0][2])
                image_pointsestimated.append([result[0][0]/result[0][2],
                            result[0][1]/result[0][2]])
                file.write("%s %s\n" %(result[0][0]/result[0][2],
                        result[0][1]/result[0][2]))
            file.close()
            return image_pointsestimated, image_points2x, image_points2y
        msex = np.mean((image_pointsestimatedx - image_pointsx)**2)
        msey = np.mean((image_pointsestimatedy - image_pointsy)**2)
        msetotal = msex + msey
        print(msetotal)
```

The main of our program are the names of the files, calculate the parameters and estimate the 2D points.

```
    def main():
        name3d = "3D_withoutnoise.txt"
        worldpoints, worldpointsx,worldpointsy = read_txt(name3d,1)
        name2d = "2D_withoutnoise.txt"
        image_points, image_pointsx , image_pointsy   = read_txt(name2d,0)

        K, extrinsicMatrix = calculateparameters(worldpoints, image_points )
        matrix = np.dot(K, extrinsicMatrix)
        image_pointsestimated,image_pointsestimatedx, image_pointsestimatedy  =
            estimate_2dpoints(matrix,worldpoints, "2D_new.txt")
```

3. Program: We have to read the parameters of the Ransac algorithm from a .config file. So it develops in a different way to a .txt.

```
    def ConfigSectionMap():
        parser = configparser.ConfigParser()
        parser.read_file(open('ransac.config'))
        n = int(parser['data']['n'])
        k = int(parser['data']['k'])
        t = int(parser['data']['t'])
        return n,k,t
```

This is the Ransac algorithm that I explained earlier. First of all, we take a series of points from our world points and image points. We are left with points that are below the average of the Euclidean distance. And we do the whole process of estimating the parameters with only those points. We repeat this entire process k times with different points and we are left with the best model (the one that has given us the least error).

```
    def ransac(worldpoints,image_points,n,k,t,debug=False):
        iterations = 0
        besterr = np.inf

        while iterations < k:
            maybe_idxs_world, test_idxs_world
                = random_partition(n,worldpoints.shape[0])
            maybe_idxs_image, test_idxs_image
```

```python
                = random_partition(n,image_points.shape[0])

            maybeinliers_world = worldpoints[maybe_idxs_world,:]
            maybeinliers_image = image_points[maybe_idxs_image,:]

            K, extrinsicMatrix = calculateparameters(maybeinliers_world,
                maybeinliers_image)
            matrix = np.dot(K, extrinsicMatrix)
            image_pointsestimated, image_pointsestimatedx,
                image_pointsestimatedy  = estimate_2dpoints(matrix,
                    maybeinliers_world)
            msex = (np.sqrt((image_pointsestimated - maybeinliers_image)**2))
            distance = np.zeros(shape=(len(msex),1))

            for i in range(0,len(msex)):
                distance[i] = msex[i][0]+msex[i][1]
            mean_distance = np.mean(distance)
            j = 0
            k = 0
            for i in range(0,len(msex)):
                if distance[i]< t*mean_distance:
                    j +=1
                else:
                    k += 1

            inliers_world = np.zeros(shape=(j,4))
            inliers_image = np.zeros(shape=(j,3))
            j = 0
            for i in range(0,len(msex)):
                if distance[i]< mean_distance:
                    inliers_world[j][0] = maybeinliers_world[i][0]
                    inliers_world[j][1] = maybeinliers_world[i][1]
                    inliers_world[j][2] = maybeinliers_world[i][2]
                    inliers_world[j][3] = maybeinliers_world[i][3]
                    inliers_image[j][0] = maybeinliers_image[i][0]
                    inliers_image[j][1] = maybeinliers_image[i][1]
                    inliers_image[j][2] = maybeinliers_image[i][2]
                    j +=1
            K, extrinsicMatrix = calculateparameters(inliers_world,
                inliers_image)
            matrix = np.dot(K, extrinsicMatrix)
            image_pointsestimated,image_pointsestimatedx,
                image_pointsestimatedy  = estimate_2dpoints(matrix,
                    inliers_world)
            msex = np.mean((image_pointsestimated - inliers_image)**2)
            if msex < besterr:
                bestlenght = len(inliers_world)
                besterr = msex
                iterations+=1
```

```
        return besterr, bestlenght
```

In this part of the code we create the indices of this part of points with which we will perform the estimation of parameters within the Ransac algorithm.

```
    def random_partition(n,n_data):
        all_idxs = np.arange( n_data )
        np.random.shuffle(all_idxs)
        idxs1 = all_idxs[:n]
        idxs2 = all_idxs[n:]
        return idxs1, idxs2
```

The main function of our last program first removes the parameters of the Ransac algorithm, reads all the documents we have and inserts them in the ransac function. Once finished, we take by parameters the probability that a point is an inlier depending on the parameters inserted in the function ransac and we extract the minimum error that we have been able to achieve in our model.

```
    n1,k1,t= ConfigSectionMap()
    name3d = ["3D_noise1.txt","3D_noise2.txt"]
    name2d = ["2D_noise1.txt","2D_noise2.txt"]
    name = ["noise1.txt","noise2.txt"]
    for i in range(0,len(name3d)):
        worldpoints, worldpointsx,worldpointsy, worldpointsz
            = read_txt3D(name3d[i])
        worldpoints1 = np.ones(shape=(len(worldpoints),1))
        imagepoints1 = np.ones(shape=(len(worldpoints),1))
        image_points, image_pointsx , image_pointsy   = read_txt2D(name2d[i])

        all_data1 = np.hstack( (worldpointsx,worldpointsy,
            worldpointsz,worldpoints1) )
        all_data2 = np.hstack( (image_pointsx,image_pointsy, imagepoints1) )
        debug = False

        msex, n = ransac(all_data1, all_data2,n1,k1,t,debug=debug)
        res = np.power((-1*(10**(np.log10(0.01)/k1)-1)), np.float64(1/n))
        print(name[i])
        print("Number of points at each evaluation")
        print(n1)
        print("Number of trials")
        print(k1)
        print("Treshold to determine is one point is an inlier")
        print(t)
        print("Probability that a point is an inlier:")
        print(res)
        print("Best model MSE:")
        print(msex)
```

## 3. Implementation details

- **Problems found during the implementation of the program:**

**Problem 1** : The MSE achieved in the second program is too high for a normal estimation.

**Problem 2** : The probability that a point is an inlier taking into account the two documents is sometimes higher in the second document, when it should not be so. I think that having higher noise in the second document the probability would have to decrease with respect to the first document and this does not happen in the program that I created.

**Problem 3** : Sometimes the second document gets a minor error, this is simply because of the points we choose in our ransac model but I do not know if it really can be that way.

- **Instructions for using the program:**

**Program 1** : In total we have three different programs that work separately. In the first, we read the images that are in the same folder. And we save the files in 3D.txt and 2D.txt. To finish visualizing the image we must press 's'. And then the points with their row are displayed to introduce the third parameter in the created file.

**Program 2** : In the second we need to have the files in the same folder and read them. The input parameters are the names of the files and the output that we will obtain are the parameters obtained during the estimation, the total MSE obtained and a visualization of the estimated points.

**Program 3** : In this program we need as parameters the names of the files. We also have to have the files in the same folder both the two 3D and 2D files and the ransac.config. As output we will obtain the two visualizations with the the probability that a point is an inlier and the best model error MSE.

## 4. Results and discussion

**Program 1** : We obtain the corners of the images, along with their position. To obtain the coordinate third we have to enter it manually. Which makes it a very expensive job, with luck they were little points. But it would be a part of the program that can be improved to automate this process, in case instead of having 6 points we have 2000.

**Program 2** :The MSE obtained in this case is very high for an estimated process. This questions whether this is the process to estimate the projection matrix. We have many outliers in our estimated points that could be eliminated.

**Program 3** : In this program the obtained results fluctuate considerably depending on the time we execute it, in any case it is always better than in program 2. But the results that we obtain taking into account the two documents do not convince me much. The first document should have a minor error or a lower probability but it does not always happen that way.