LAB

VISUAL PERCEPTION

May 20, 2019

Submitted to
David FOFI

ASRAF ALI Abdul Salam Rasmi
Masters in Computer Vision
Centre Universitaire Condorcet
Universite de Bourgogne

Calibration and Triangulation

1 CALIBRATION

I have used MATLAB built-in Camera Calibration Toolbox which uses Zhang Calibration Technique. The real intrinsic Parameters are as follows.

```
Intrinsic Matrix
  736.7111         0  309.4650
         0  738.2868  234.4135
         0         0    1.0000
```

Figure 1: Real Intrinsic Parameters

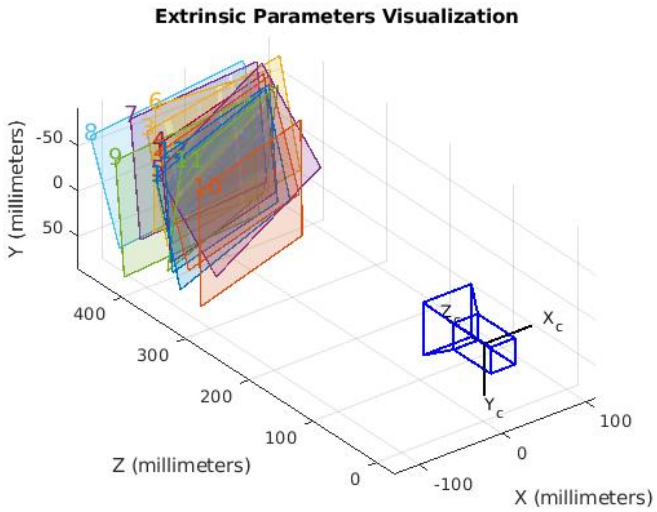

Figure 2: Orientation of Calibration Patterns

2 SIMULATION OF CAMERA AND 3D SCENE

I simulated a 3D scene above the Camera's Focal plane so that the scene can be projected on to the Camera plane. A MATLAB function *drawCube3D* takes in the origin of the Cube and length along each axis and plots a cube in a 3D plot. The coordinates of the cube is given below.

Simulated 3D Points		
20	10	800
100	10	800
100	50	800
20	50	800
20	10	900
100	10	900
100	50	900
20	50	900

Figure 3: Simulated 3D Points

The camera is simulated with real Intrinsic Parameters obtained from the Calibration and simulated Extrinsic Parameters. The simulation is shown below.

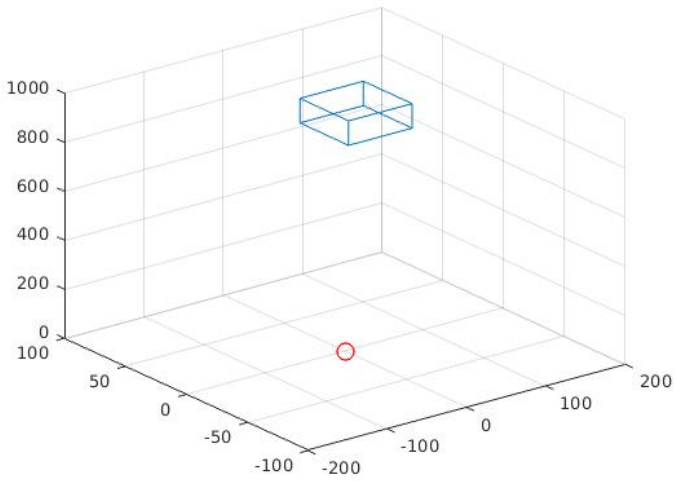

Figure 4: Simulated Camera and 3D scene

3 PROJECTION OF 3D POINT ONTO THE CAMERA PLANE

Using the known Intrinsic Matrix and simulated Extrinsic Matrix, I have computed the Projection Matrix using the function *getProjectionMatrix*. The Projection Matrix is given by,

$$P = K \times C \times [R|t]$$

where, K is the Intrinsic Matrix, C is the Normalized Camera Matrix and $[R|t]$ is the Extrinsic Matrix. With this Matrix I have computed the corresponding 2D points in homogeneous coordinate for each 3D point using the function *project2D*.

In order to show the points we need to create the camera plane (along Z-axis). For this step I have used some online sources to create a Camera plane and to project the 3D Points onto it. The functions doing this task are *imagept2plane, camstruct, decomposecamera, projmatrix2camstruct, rq3* [1]. The Output of the Projection and the Image in 2D are shown in the following figures.

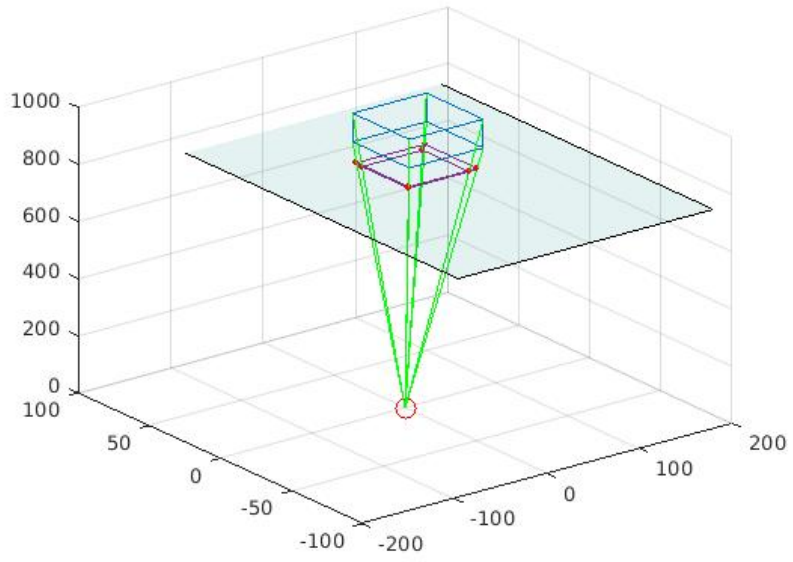

Figure 5: Figure showing the Camera Plane, Projection Points & Projection Lines

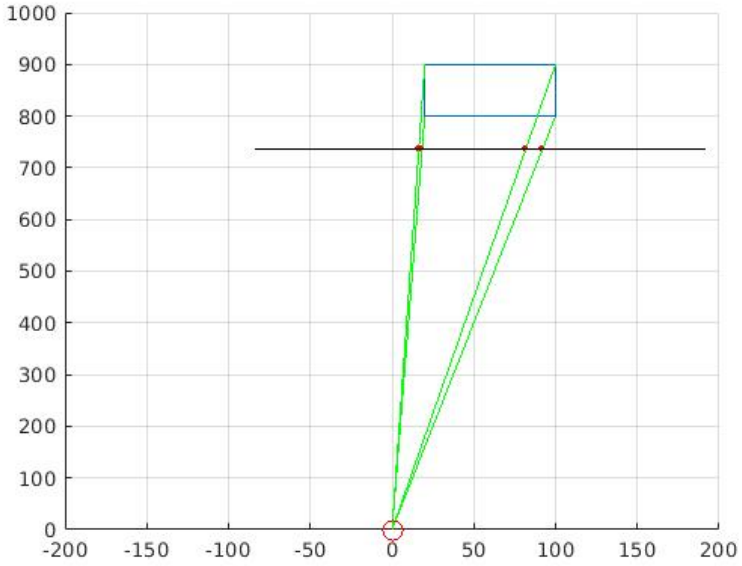

Figure 6: X - Z view

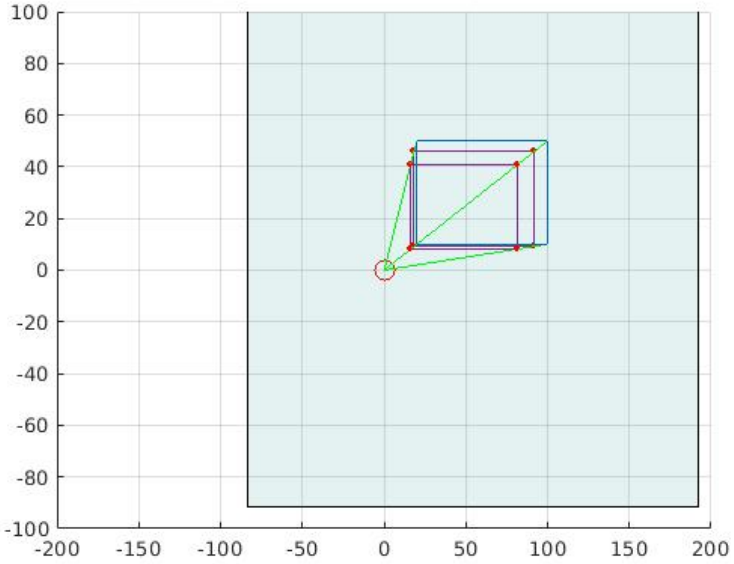

Figure 7: X - Y view

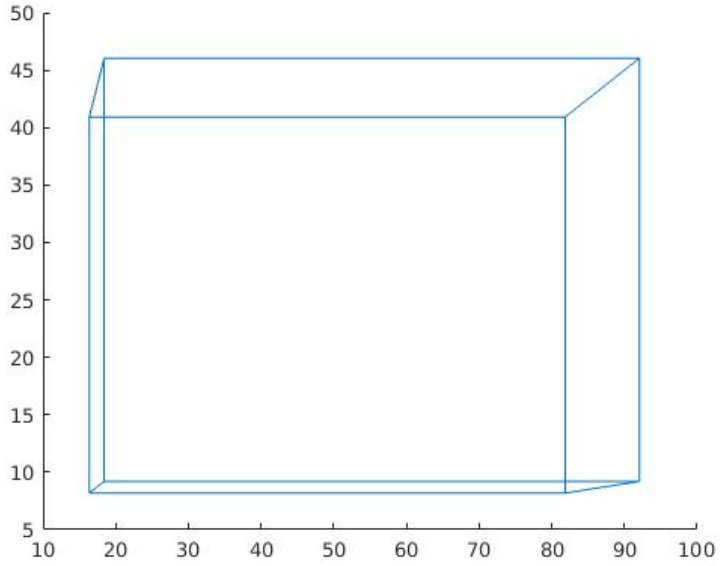

Figure 8: The Image Projected onto the Camera Plane

4 IMPLEMENTATION OF DLT FOR CALIBRATION

I have implemented the DLT to compute the Projection Matrix 'P' given the 3D and corresponding 2D points. The steps are as follows.

1. Take a 3D Point (X, Y, Z) and its corresponding 2D point (u, v) and transform them into a 2×12 Matrix as follows.

$$\begin{bmatrix} -X & -Y & -Z & -1 & 0 & 0 & 0 & 0 & uX & uY & uZ & u \\ 0 & 0 & 0 & 0 & -X & -Y & -Z & -1 & vX & vY & vZ & v \end{bmatrix}$$

2. Repeat the above step for N points to form $2N \times 12$ Matrix.

3. Compute the SVD of the above matrix and take the Last Eigen Vector (12×1)

4. Reshape this vector into a 3×4 Matrix. This gives us the **Projection Matrix**.

5. After computing the Projection Matrix, use the function *decomposecamera* to get the Intrinsic Parameters.

6. We can see that the the scale is lost, so we should make the the last element of the matrix to 1, to get the scale back as shown in figure 9

```
Intrinsic Matrix computed with DLT
    0.6620   -0.0000    0.2781
   -0.0000    0.6634    0.2106
         0         0    0.0009

Intrinsic Matrix computed with DLT (Rescaled)
  736.7111   -0.0000  309.4650
   -0.0000  738.2868  234.4135
         0         0    1.0000
```

Figure 9: Computed Intrinsic Matrix (using DLT)

5 SIMULATION OF THE SECOND CAMERA

I simulated the Second Camera by translating the First Camera along the X-axis by some value and then I repeated the same procedure to compute Projection Matrix and to project the 3D Points onto the second Camera Plane. Th Output are as follows.

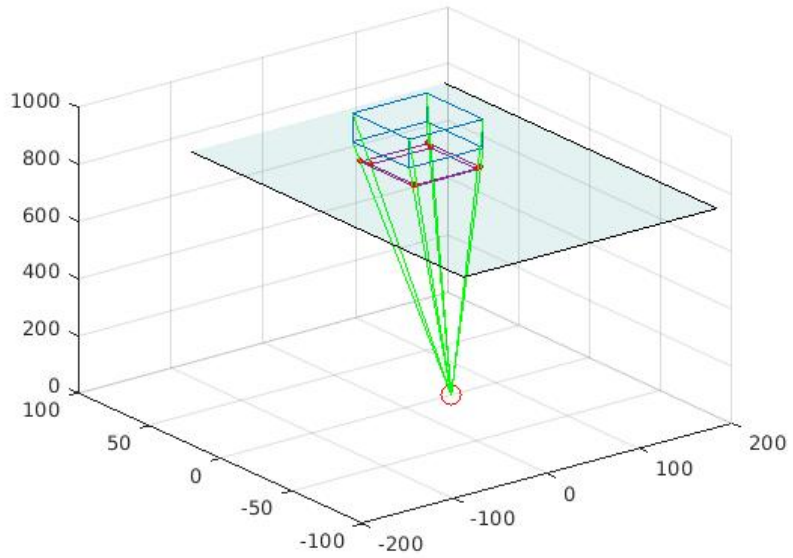

Figure 10: Figure showing the Camera Plane, Projection Points & Projection Lines

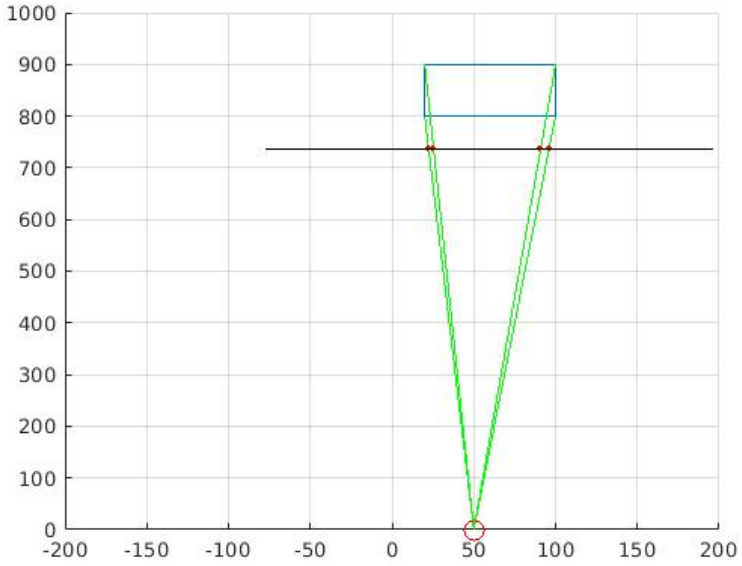

Figure 11: X - Z view

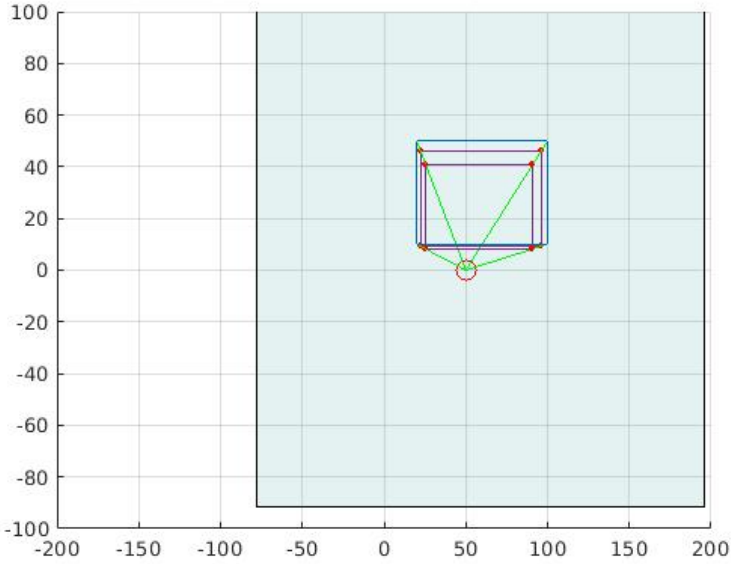

Figure 12: X - Y view

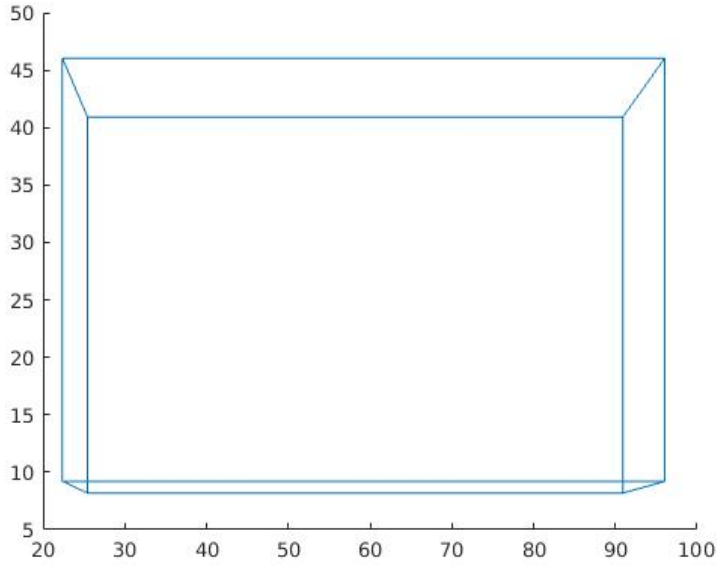

Figure 13: The Image Projected onto the Camera Plane

6 TRIANGULATION

After projecting the Points onto the second camera Plane, I tried to Triangulate the 2D Points from both Camera Plane to retrieve the 3D Points. The function *triangulate3D* does this task. The steps involved are as follows.

1. For two corresponding 2D Points, $x_1 = (u_1, v_1)$ & $x_2 = (u_2, v_2)$ design a (3×3) Matrix as follows.
$$X_1 = \begin{bmatrix} 0 & 1 & -v_1 \\ -1 & 0 & u_1 \\ v_1 & -u_1 & 0 \end{bmatrix} \quad X_2 = \begin{bmatrix} 0 & 1 & -v_2 \\ -1 & 0 & u_2 \\ v_2 & -u_2 & 0 \end{bmatrix}$$
2. Multiply the above Matrices with corresponding Projection Matrices P_1, P_2
3. Append the Results and compute SVD for the Matrix.
4. The Last Eigen Vector is the required 3D Point upto scale.
5. Make the Points Homogeneous and consider the first 3 values (X,Y,Z).

The Triangulated 3D Points are shown below. We can notice that it gave exact 3D Points. This is because both the Camera and the 3D Scene are Simulated. However, with Real scenes we will experience some error in computation as 3D Points in the Real scenes are Random.

```
Triangulated 3D Points
   20.0000   10.0000  800.0000
  100.0000   10.0000  800.0000
  100.0000   50.0000  800.0000
   20.0000   50.0000  800.0000
   20.0000   10.0000  900.0000
  100.0000   10.0000  900.0000
  100.0000   50.0000  900.0000
   20.0000   50.0000  900.0000
```

Figure 14: Triangulated 3D Points

Bibliography

[1] P. D. Kovesi, “MATLAB and Octave functions for computer vision and image processing.” Available from: <http://www.peterkovesi.com/matlabfns/>.