

CSE578 Computer Vision

Assignment 1 : Camera Calibration

Karnik Ram
2018701007

1 Direct Linear Transform

The Direct Linear Transform (DLT) algorithm is used to estimate the camera matrix (11 dof) from a set of atleast 6 2D-3D point correspondences. The algorithm is based on solving the system of linear equations $x = PX$, which is then simply rearranged to form a system $Mp = 0$. p is a 12×1 vector and M is a $2N \times 12$ matrix where N is the number of correspondences. The solution p is obtained using SVD which is then rearranged to form back the matrix P . P is then decomposed into K, R, C using RQ decomposition.

1.1 Code/Algorithm

The following is a snippet from Listing 2 (Section 11) which implements DLT.

```
1
2 void Calibrator::calibrateByDlt(const std::vector<int> &sample_indices)
3 {
4     if(sample_indices.size() < 6)
5     {
6         std::cout << "Cannot run DLT! Require at least 6 correspondences.\n";
7         return;
8     }
9
10    Eigen::MatrixXf X_samples(sample_indices.size(),4);
11    Eigen::MatrixXf x_samples(sample_indices.size(),3);
12
13    if(sample_indices.size() == X.rows())
14    {
15        X_samples = X;
16        x_samples = x;
17    }
18
19    else
20    { int j = 0;
21      for (int i : sample_indices)
22      {
23          X_samples.row(j) = X.row(i);
24          x_samples.row(j++) = x.row(i);
25      }
26    }
27
28    Eigen::ArrayXf p(12);
29    Eigen::MatrixXf M(2*x_samples.rows(),12);
30
```

```

31 // Build M
32 for (int i = 0, j=0; i < M.rows(); i+=2, j++)
33 {
34     M.row(i) << X_samples.row(j) * -1, Eigen::MatrixXf::Zero(1,4), X_samples.row(j)
        * x_samples(j,0);
35     M.row(i+1) << Eigen::MatrixXf::Zero(1,4), X_samples.row(j) * -1, X_samples.row
        (j) * x_samples(j,1);
36 }
37
38 // Estimate p
39 Eigen::JacobiSVD<Eigen::MatrixXf> svd(M, Eigen::ComputeFullU | Eigen::
    ComputeFullV);
40 Eigen::MatrixXf V;
41 V = svd.matrixV();
42 p = V.col(V.cols()-1);
43
44 // Build P
45 P.resize(3,4);
46 P.row(0) = p.segment(0,4);
47 P.row(1) = p.segment(4,4);
48 P.row(2) = p.segment(8,4);
49 std::cout << "P:\n" << P << std::endl;
50 std::cout << "Average reprojection error:\n" << calcAvgReprojectionError() <<
    std::endl;
51 }
52

```

2 DLT with RANSAC

Random sample consensus (RANSAC) is a robust estimation algorithm. The idea is to determine a set of inliers from the set of correspondences, which are then used to estimate the calibration parameters. This is done by repeatedly sampling minimal sets (6 correspondences) and estimating the model, and then choosing the model which is supported by the maximum number of inliers. Inliers are determined using user specified distance and ratio thresholds.

2.1 Code/Algorithm

The following is a snippet from Listing 2 (Section 11) which implements DLT inside a RANSAC scheme.

```

1
2 void Calibrator::calibrateByDltRansac(const float &dist_threshold, const float &
    inlier_ratio, std::vector<int> &inlier_indices)
3 {
4     std::cout << "Running RANSAC..." << std::endl;
5     std::vector<int> largest_support;
6
7     for (int n = 0; n < 500; n++)

```

```

8 {
9     std::cout << "\n\nIteration #" << n+1 << std::endl;
10    std::vector<int> sample_indices = utils::generateRandomVector(0,X.rows()-1,6);
11    calibrateByDlt(sample_indices);
12
13    for(int i = 0; i < X.rows(); i++)
14    {
15        float dist = calcReprojectionError(X.row(i),x.row(i));
16        if(dist < dist_threshold)
17            inlier_indices.push_back(i);
18    }
19
20    if(inlier_indices.size() >= inlier_ratio * X.rows())
21    {
22        std::cout << "Found a model!\n" << "Number of inliers: " << inlier_indices.
size() << std::endl;
23        std::cout << "Inliers: ";
24        for(int i : inlier_indices)
25            std::cout << i << " ";
26
27        std::cout << "\n";
28        calibrateByDlt(inlier_indices);
29        return;
30    }
31
32    else
33    {
34        if(largest_support.size() <= inlier_indices.size())
35            largest_support = inlier_indices;
36
37        inlier_indices.clear();
38    }
39 }
40
41 if(largest_support.size() >= 6)
42 {
43     std::cout << "Could not find a model according to threshold!\nSo using largest
inlier set instead." << std::endl;
44     calibrateByDlt(largest_support);
45     inlier_indices = largest_support;
46     std::cout << "Number of inliers: " << inlier_indices.size() << std::endl;
47     for(int i : inlier_indices)
48         std::cout << i << " ";
49
50     std::cout << "\n";
51 }
52
53 else
54     std::cout << "Could not find a model!" << std::endl;
55 }
56
57

```

3 Results using the provided setup

The 2D and 3D points were measured manually from the provided images. The Z-axis of the world frame was taken to be pointing into the object to make it a right-handed coordinate frame for convenience. The estimated parameters were validated using the average reprojection error as a metric.

DLT

$$\text{Calibration matrix : } \begin{pmatrix} 12077.4 & -149.151 & 3008.17 \\ 0 & 12200.9 & 2970.19 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\text{Rotation matrix : } \begin{pmatrix} -0.9780 & 0.0072 & 0.2084 \\ -0.0916 & -0.9126 & -0.3983 \\ 0.1873 & -0.4086 & 0.8932 \end{pmatrix}$$

$$\text{Camera center : } \begin{pmatrix} -0.1435 \\ -0.0648 \\ -2.99 \times 10^{-5} \end{pmatrix}$$

Average reprojection error : 310.596

DLT with RANSAC

$$\text{Calibration matrix : } \begin{pmatrix} 12578 & -216.138 & 2778.59 \\ 0 & 12633.2 & 2080.56 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\text{Rotation matrix : } \begin{pmatrix} -0.9745 & -0.0016 & 0.2242 \\ -0.0762 & -0.9380 & -0.3380 \\ 0.2109 & -0.3465 & 0.9140 \end{pmatrix}$$

$$\text{Camera center : } \begin{pmatrix} -0.1447 \\ -0.0648 \\ -3.0195 \times 10^{-5} \end{pmatrix}$$

Average reprojection error : 447.887

Reprojection error threshold : 250

Inlier ratio threshold : 0.8

The thresholds for RANSAC were set by trial-and-error. The maximum number of iterations was set to 500.

4 Undistortion

The radial and tangential distortion parameters were estimated using Matlab's `estimateCameraParameters` function. The checkerboard images were used for this purpose. Using these parameters, the image of the 3D object was undistorted using OpenCV's `undistort` function. I assumed the 3D object, and the checkerboard were both captured using the same camera.

The following are the estimated distortion parameters:

$$\begin{aligned}\text{Radial} &: 0.2963, -2.999, 20.5691 \\ \text{Tangential} &: 0.0017, 0.0112\end{aligned}$$

The three radial distortion values correspond to k_1, k_2, k_3 , and tangential to p_1, p_2 . The distortion parameters are usually estimated by finding the straight line in the image, and then doing a search over the values of p_1, p_2, k_1, k_2, k_3 that minimizes the distance between the midpoint of these lines and the average of their endpoints.

$$\begin{aligned}x_{\text{distortion}} &= x + (2p_1xy + p_2(r^2 + 2x^2)) \\ y_{\text{distortion}} &= y + (2p_2xy + p_1(r^2 + 2y^2))\end{aligned}$$

$$\begin{aligned}x_{\text{distortion}} &= x(1 + k_1r^2 + k_2r^4 + k_3r^6) \\ y_{\text{distortion}} &= y(1 + k_1r^2 + k_2r^4 + k_3r^6)\end{aligned}$$

The following are the estimated camera parameters after undistorting the 3D calibration object image:

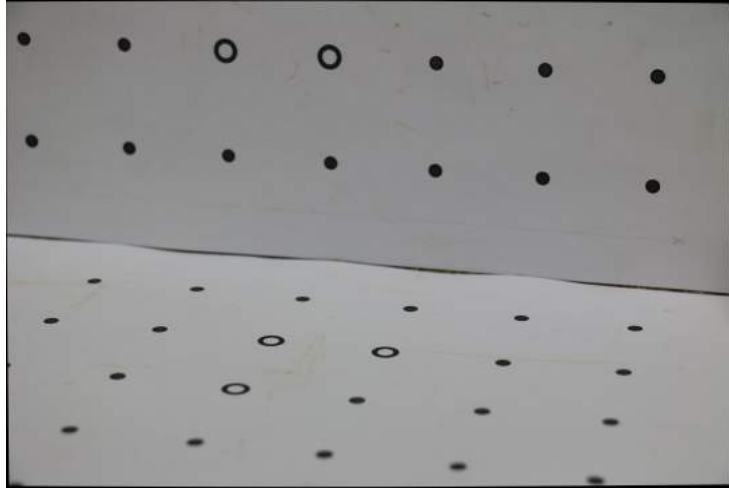


Figure 1: Undistorted image

DLT

$$\text{Calibration matrix : } \begin{pmatrix} 12768 & -131.677 & 3283.4 \\ 0 & 12653.8 & 2589.14 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\text{Rotation matrix : } \begin{pmatrix} -0.9816 & 0.0143 & 0.1902 \\ -0.0875 & -0.9199 & -0.3820 \\ 0.1965 & -0.3916 & 0.9043 \end{pmatrix}$$

$$\text{Camera center : } \begin{pmatrix} -0.1458 \\ -0.0653 \\ -3.04 \times 10^{-5} \end{pmatrix}$$

Average reprojection error : 318.676

DLT with RANSAC

$$\text{Calibration matrix : } \begin{pmatrix} 12668 & -31.677 & 3000.4 \\ 0 & 12633.8 & 2386.14 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\text{Rotation matrix : } \begin{pmatrix} -0.9776 & 0.0162 & 0.1928 \\ -0.0834 & -0.9171 & -0.3811 \\ 0.1915 & -0.3626 & 0.9123 \end{pmatrix}$$

$$\text{Camera center : } \begin{pmatrix} -0.1248 \\ -0.0723 - 3.64 \times 10^{-5} \end{pmatrix}$$

Average reprojection error : 503.276

Reprojection error threshold : 200

Inlier ratio threshold : 0.7

5 Wireframe overlay

DLT

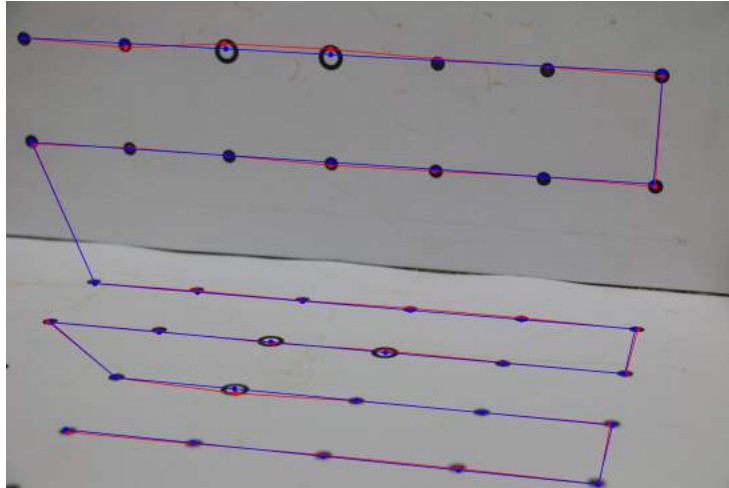


Figure 2: Wireframe overlayed over the 3D object after DLT calibration. Blue indicates the estimated points, red indicates the measured points.

DLT with RANSAC

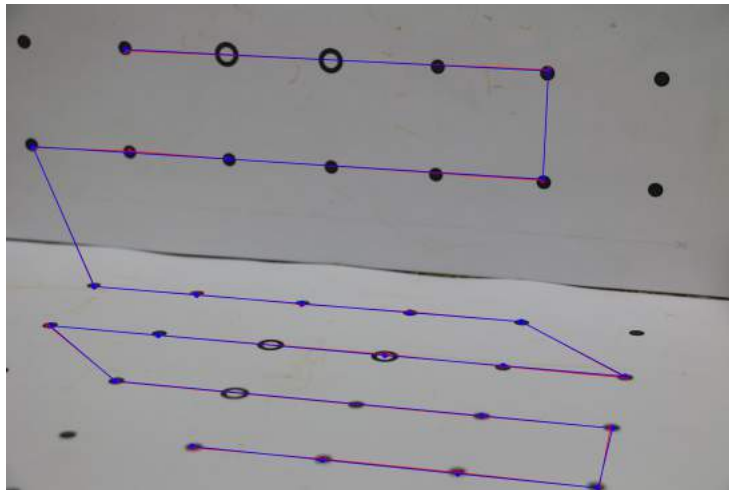


Figure 3: Wireframe of only the inlier points overlayed over the 3D object after DLT - RANSAC calibration. Blue indicates the estimated points, red indicates the measured points.

6 Zhang's method

Zhang's method was tested on the provided checkerboard images using Matlab's implementation, available in the Computer Vision toolbox. Code is provided in Listing 6. The following are the estimated values:

$$\text{Calibration matrix : } \begin{pmatrix} 13459 & 78 & 2983 \\ 0 & 13507 & 1849 \\ 0 & 0 & 1 \end{pmatrix}$$

Average reprojection error : 1.6417

The results are obviously much better than the ones obtained using my implementation of DLT. One can observe that the c_x, c_y values almost exactly overlap with the center of the image. The reprojection error is also less than 2 pixels.

7 Wireframe overlay using Zhang's

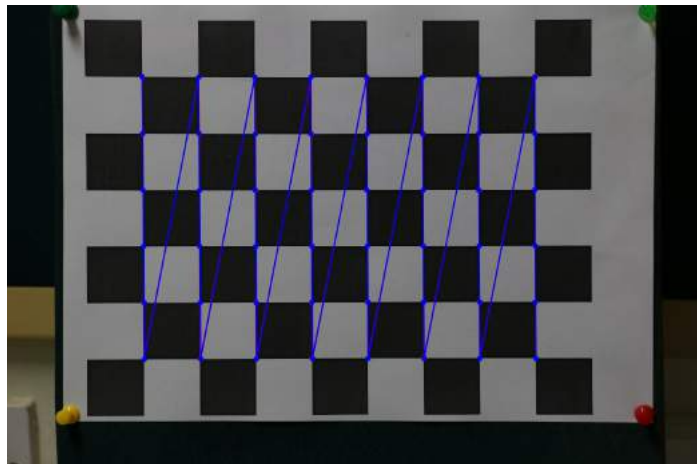


Figure 4: Wireframe overlayed over the checkerboard after calibration. Blue indicates the estimated points, red indicates the measured points.

8 Image of world origin

The world origin when multiplied by the camera matrix gets projected onto the corresponding location in the image as expected.

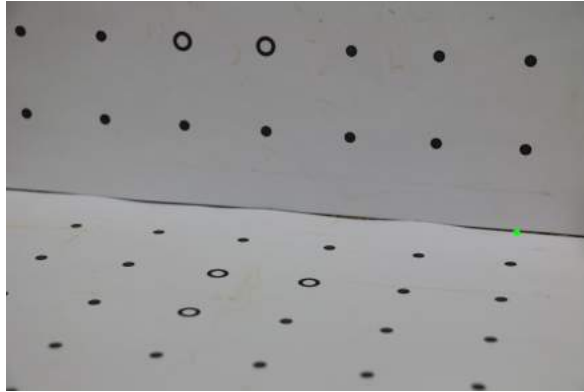


Figure 5: Image of the world origin

9 Own setup

For testing the code on my setup, I used my phone camera after fixing its focal length. I used a Rubik's cube as my calibration object since it is of known dimensions and the points are easy to measure. For Zhang's method I printed out a 12X7 checkerboard and took multiple images of it from different positions.



Figure 6: Calibration object for DLT, captured with phone camera.

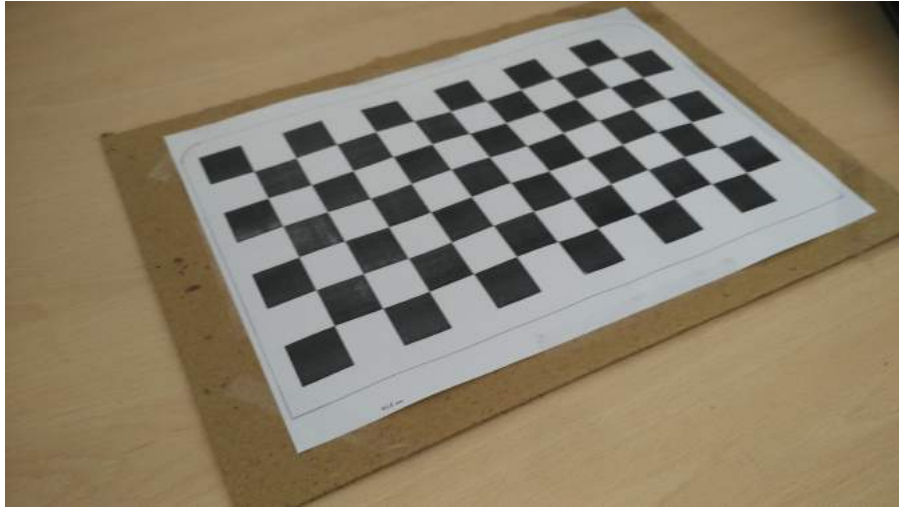


Figure 7: Calibration object for Zhang's, captured with phone camera.

10 Results using own setup

DLT

$$\text{Calibration matrix : } \begin{pmatrix} -4209.56 & 22.7655 & 1331.44 \\ 0 & -4186.34 & 2174.04 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\text{Rotation matrix : } \begin{pmatrix} -0.7607 & -0.0113 & 0.6489 \\ 0.4814 & -0.6802 & 0.5526 \\ 0.4351 & 0.7329 & 0.5229 \end{pmatrix}$$

$$\text{Camera center : } \begin{pmatrix} 0.0311 \\ 0.0550 \\ 3.0356 \times 10^{-5} \end{pmatrix}$$

Average reprojection error : 86.6504

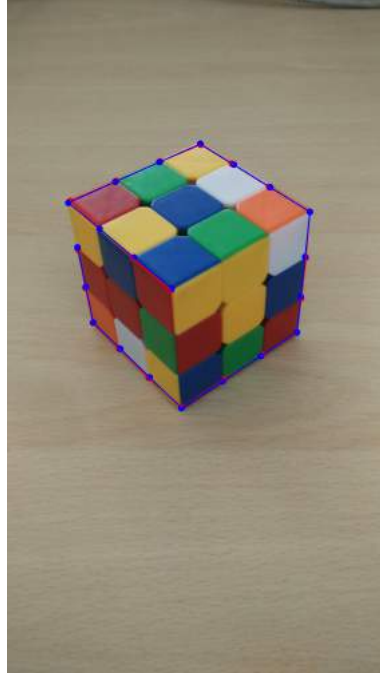


Figure 8: Wireframe overlayed over the cube object after calibration. Blue indicates projected points, red indicates measured points.

DLT with RANSAC

$$\text{Calibration matrix : } \begin{pmatrix} -4342.95 & -24.8555 & 911.788 \\ 0 & -4377.99 & 1868.58 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\text{Rotation matrix : } \begin{pmatrix} -0.8037 & -0.0652 & 0.5864 \\ 0.4369 & -0.7340 & 0.5199 \\ 0.3964 & 0.676 & 0.6211 \end{pmatrix}$$

$$\text{Camera center : } \begin{pmatrix} 0.0319 \\ 0.0564 \\ 3.1294 \times 10^{-5} \end{pmatrix}$$

Average reprojection error : 198.293

Reprojection error threshold : 50

Inlier ratio threshold : 0.5

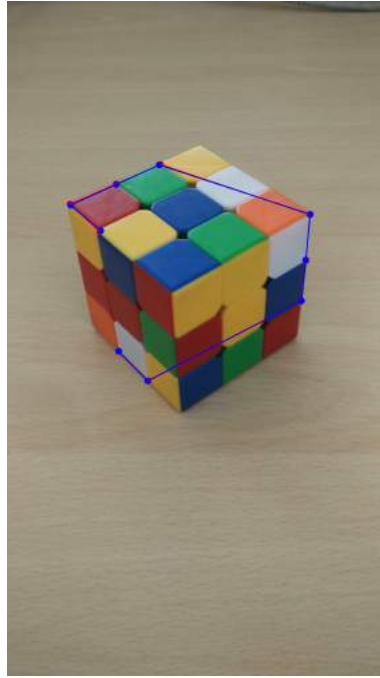


Figure 9: Wireframe of only the inlier points overlayed over the cube object after calibration. Blue indicates projected points, red indicates measured points.

Zhang's

$$\text{Calibration matrix : } \begin{pmatrix} 3765.8 & -4.1 & 2067.7 \\ 0 & 3757.1 & 1204.7 \\ 0 & 0 & 1 \end{pmatrix}$$

Average reprojection error : 1.5617

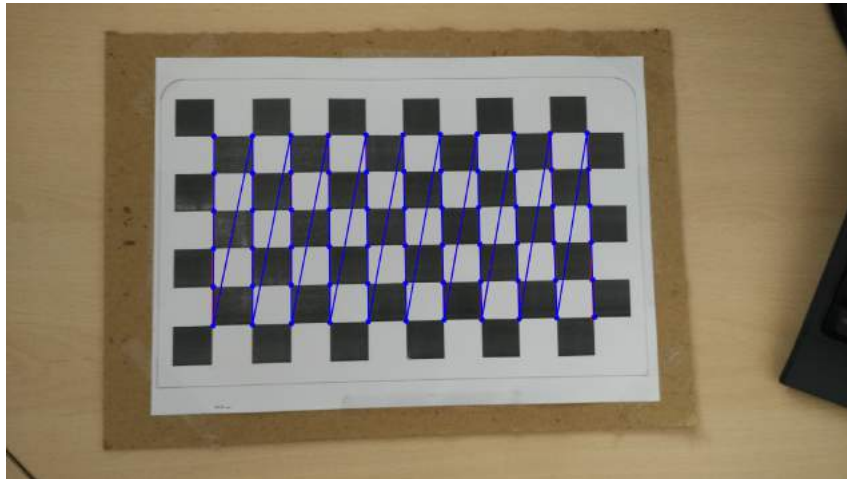


Figure 10: Wireframe overlaid over the checkerboard after calibration. Blue indicates the estimated points, red indicates the measured points.

11 Code

All the code files can be found [here](#). The main algorithms are implemented in Listing 2.

Listing 1: calibrator.hpp

```

1 #pragma once
2
3 #include <iostream>
4
5 #include <opencv2/core.hpp>
6 #include <opencv2/highgui.hpp>
7 #include <opencv2/imgproc.hpp>
8
9 #include <Eigen/Core>
10 #include <Eigen/SVD>
11 #include <Eigen/QR>
12 #include <Eigen/LU>
13
14 #include "utils.hpp"
15
16 class Calibrator
17 {
18 public:
19     Calibrator(const Eigen::MatrixXf &points_3d, const Eigen::MatrixXf &points_2d)
20     ;
21     void calibrateByDlt(const std::vector<int> &sample_indices);
22     void calibrateByDltRansac(const float &dist_threshold, const float &
        inlier_ratio, std::vector<int> &inlier_indices);
23     float calcReprojectionError(const Eigen::Vector4f &pt_3d, const Eigen::
        Vector3f &pt_img);

```

```

23     float calcSetReprojectionError(const Eigen::MatrixXf &pts_3d, const Eigen::
MatrixXf &pts_img);
24     float calcAvgReprojectionError();
25     void decomposePMatrix(Eigen::MatrixXf &K, Eigen::MatrixXf &R, Eigen::MatrixXf
&c);
26     void drawOverlay(const std::vector<int> &sample_indices, cv::Mat &frame);
27     Eigen::MatrixXf getPMatrix();
28
29 private:
30     Eigen::MatrixXf X;
31     Eigen::MatrixXf x;
32     Eigen::MatrixXf P;
33 };

```

Listing 2: calibrator.cpp

```

1 #include "calibrator.hpp"
2 #include <stdlib.h>
3
4 Calibrator::Calibrator(const Eigen::MatrixXf &points_3d, const Eigen::MatrixXf &
points_2d)
5 {
6     x = points_2d;
7     x.conservativeResize(x.rows(), x.cols() + 1);
8     x.col(x.cols() - 1).setOnes();
9
10    X = points_3d;
11    X = X * 0.01; // assumes points are in cm, converting to m
12    X.conservativeResize(X.rows(), X.cols() + 1);
13    X.col(X.cols() - 1).setOnes();
14 }
15
16 void Calibrator::calibrateByDlt(const std::vector<int> &sample_indices)
17 {
18     if(sample_indices.size() < 6)
19     {
20         std::cout << "Cannot run DLT! Require at least 6 correspondences.\n";
21         return;
22     }
23
24     Eigen::MatrixXf X_samples(sample_indices.size(), 4);
25     Eigen::MatrixXf x_samples(sample_indices.size(), 3);
26
27     if(sample_indices.size() == X.rows())
28     {
29         X_samples = X;
30         x_samples = x;
31     }
32
33     else
34     {
35         int j = 0;
36         for (int i : sample_indices)
37         {
38             X_samples.row(j) = X.row(i);
39             x_samples.row(j) = x.row(i);
40             j++;
41         }
42     }
43 }

```

```

38     x_samples.row(j++) = x.row(i);
39 }
40 }
41
42 Eigen::ArrayXf p(12);
43 Eigen::MatrixXf M(2*x_samples.rows(),12);
44
45 // Build M
46 for(int i = 0, j=0; i < M.rows(); i+=2, j++)
47 {
48     M.row(i) << X_samples.row(j) * -1, Eigen::MatrixXf::Zero(1,4), X_samples.row(j)
49     * x_samples(j,0);
50     M.row(i+1) << Eigen::MatrixXf::Zero(1,4), X_samples.row(j) * -1, X_samples.row
51     (j) * x_samples(j,1);
52 }
53
54 // Estimate p
55 Eigen::JacobiSVD<Eigen::MatrixXf> svd(M, Eigen::ComputeFullU | Eigen::
56     ComputeFullV);
57 Eigen::MatrixXf V;
58 V = svd.matrixV();
59 p = V.col(V.cols()-1);
60
61 // Build P
62 P.resize(3,4);
63 P.row(0) = p.segment(0,4);
64 P.row(1) = p.segment(4,4);
65 P.row(2) = p.segment(8,4);
66 std::cout << "P:\n" << P << std::endl;
67 std::cout << "Average reprojection error:\n" << calcAvgReprojectionError() <<
68     std::endl;
69 }
70
71 void Calibrator::calibrateByDltRansac(const float &dist_threshold, const float &
72     inlier_ratio, std::vector<int> &inlier_indices)
73 {
74     std::cout << "Running RANSAC..." << std::endl;
75     std::vector<int> largest_support;
76
77     for(int n = 0; n < 500; n++)
78     {
79         std::cout << "\n\nIteration #" << n+1 << std::endl;
80         std::vector<int> sample_indices = utils::generateRandomVector(0, X.rows()-1, 6);
81         calibrateByDlt(sample_indices);
82
83         for(int i = 0; i < X.rows(); i++)
84         {
85             float dist = calcReprojectionError(X.row(i), x.row(i));
86             if(dist < dist_threshold)
87                 inlier_indices.push_back(i);
88         }
89
90         if(inlier_indices.size() >= inlier_ratio * X.rows())

```

```

86     {
87         std::cout << "Found a model!\n" << "Number of inliers: " << inlier_indices.
size() << std::endl;
88         std::cout << "Inliers: ";
89         for(int i : inlier_indices)
90             std::cout << i << " ";
91
92         std::cout << "\n";
93         calibrateByDlt(inlier_indices);
94         return;
95     }
96
97     else
98     {
99         if(largest_support.size() <= inlier_indices.size())
100             largest_support = inlier_indices;
101
102         inlier_indices.clear();
103     }
104 }
105
106 if(largest_support.size() >= 6)
107 {
108     std::cout << "Could not find a model according to threshold!\nSo using largest
inlier set instead." << std::endl;
109     calibrateByDlt(largest_support);
110     inlier_indices = largest_support;
111     std::cout << "Number of inliers: " << inlier_indices.size() << std::endl;
112     for(int i : inlier_indices)
113         std::cout << i << " ";
114
115     std::cout << "\n";
116 }
117
118 else
119     std::cout << "Could not find a model!" << std::endl;
120 }
121
122 float Calibrator::calcReprojectionError(const Eigen::Vector4f &pt_3d, const Eigen
::Vector3f &pt_img)
123 {
124     Eigen::Vector3f est_pt_img = P * pt_3d;
125     est_pt_img = est_pt_img / est_pt_img(2);
126     float error = (est_pt_img - pt_img).squaredNorm();
127
128     return error;
129 }
130
131
132 float Calibrator::calcSetReprojectionError(const Eigen::MatrixXf &pts_3d, const
Eigen::MatrixXf &pts_img)
133 {
134     Eigen::MatrixXf est_pts_img = P * pts_3d.transpose();

```



```

135 Eigen::MatrixXf scale = est_pts_img.row(2).replicate(3,1);
136 est_pts_img = est_pts_img.array() / scale.array();
137
138 float total_error = 0;
139
140 for(int i = 0; i < pts_3d.rows(); i++)
141 {
142     total_error += (est_pts_img.col(i) - pts_img.row(i).transpose()).squaredNorm();
143 }
144
145 float avg_error = total_error/pts_3d.rows();
146 return avg_error;
147 }
148
149 float Calibrator::calcAvgReprojectionError()
150 {
151     Eigen::MatrixXf est_pts_img = P * X.transpose();
152     Eigen::MatrixXf scale = est_pts_img.row(2).replicate(3,1);
153     est_pts_img = est_pts_img.array() / scale.array();
154
155     float total_error = 0;
156
157     for(int i = 0; i < X.rows(); i++)
158     {
159         total_error += (est_pts_img.col(i) - x.row(i).transpose()).squaredNorm();
160     }
161
162     float avg_error = total_error/X.rows();
163     return avg_error;
164 }
165
166 void Calibrator::decomposePMatrix(Eigen::MatrixXf &K, Eigen::MatrixXf &R, Eigen::
    MatrixXf &c)
167 {
168     Eigen::MatrixXf H,p4;
169     H = P.block(0,0,3,3);
170     p4 = P.col(3);
171
172     std::cout << "H:\n" << H << std::endl;
173     std::cout << "p4:\n" << p4 << std::endl;
174
175     //Camera center
176     c = -1 * H.inverse() * p4;
177
178     //Calibration matrix, rotation matrix
179     Eigen::HouseholderQR<Eigen::MatrixXf> qr(H.inverse());
180     R = qr.householderQ();
181     K = qr.matrixQR().triangularView<Eigen::Upper>();
182
183     R = R.inverse();
184     K = K.inverse();
185     K = K / K(2,2);

```

```

186
187     std::cout << "R:\n" << R << std::endl;
188     std::cout << "K:\n" << K << std::endl;
189 }
190
191 void Calibrator::drawOverlay(const std::vector<int> &sample_indices, cv::Mat &
    frame)
192 {
193     Eigen::MatrixXf X_samples(sample_indices.size(),4);
194     Eigen::MatrixXf x_samples(sample_indices.size(),3);
195
196     if(sample_indices.size() == X.rows())
197     {
198         X_samples = X;
199         x_samples = x;
200     }
201
202     else
203     { int j = 0;
204       for (int i : sample_indices)
205       {
206           X_samples.row(j) = X.row(i);
207           x_samples.row(j++) = x.row(i);
208       }
209     }
210
211     Eigen::MatrixXf est_x, est_u, est_v;
212     est_x = P * X_samples.transpose();
213
214     //std::cout << "image: \n" << x << std::endl;
215
216     est_u = est_x.row(0).array() / est_x.row(2).array();
217     est_v = est_x.row(1).array() / est_x.row(2).array();
218
219     //std::cout << "u:\n" << u << std::endl;
220     //std::cout << "test: u(2) \n" << u(2) << std::endl;
221
222     //Mark points
223     for(int i = 0; i < x_samples.rows(); i++)
224     {
225         cv::circle(frame, cv::Point(x_samples(i,0), x_samples(i,1)), 20, cv::Scalar
            (0,0,255), -1, CV_AA);
226         cv::circle(frame, cv::Point(est_u(i), est_v(i)), 20, cv::Scalar(255,0,0), -1, CV_AA)
            ;
227     }
228
229     //Draw lines
230     for(int i = 0; i < x_samples.rows() - 1; i++)
231     {
232         cv::line(frame, cv::Point(x_samples(i,0), x_samples(i,1)), cv::Point(x_samples(i
            +1,0), x_samples(i+1,1)), cv::Scalar(0,0,255), 5, CV_AA);
233         cv::line(frame, cv::Point(est_u(i), est_v(i)), cv::Point(est_u(i+1), est_v(i+1)),
            cv::Scalar(255,0,0), 5, CV_AA);

```

```

234 }
235 }
236
237 Eigen::MatrixXf Calibrator::getPMatrix()
238 {
239     return P;
240 }

```

Listing 3: run_dlt.cpp

```

1 #include <iostream>
2 #include <string>
3
4 #include "utils.hpp"
5 #include "calibrator.hpp"
6 #include <opencv2/core/eigen.hpp>
7
8 int main(int argc, char *argv[])
9 {
10     // std::string frame_path = "../data/dlt/IMG_5455.JPG";
11     // std::string points_2d_path = "../data/dlt/2d-points.txt";
12     // std::string points_3d_path = "../data/dlt/3d-points.txt";
13     // std::string points_2d_path = "../data/dlt/undistort-2d-points.txt";
14
15     std::string frame_path = "../data/phone-camera/dlt/cube.jpg";
16     std::string points_2d_path = "../data/phone-camera/dlt/2d-points.txt";
17     std::string points_3d_path = "../data/phone-camera/dlt/3d-points.txt";
18
19     Eigen::MatrixXf points_2d;
20     utils::loadMatrix(points_2d_path, utils::getNumberOfLines(points_2d_path), 2,
21         points_2d);
22
23     Eigen::MatrixXf points_3d;
24     utils::loadMatrix(points_3d_path, utils::getNumberOfLines(points_3d_path), 3,
25         points_3d);
26
27     if(points_2d.rows() != points_3d.rows())
28     {
29         std::cout << "Number of correspondences don't match!\n";
30         return -1;
31     }
32
33     cv::Mat frame;
34     frame = cv::imread(frame_path, 1);
35
36     /* Undistortion
37
38     cv::Mat undistorted_frame;
39     cv::Mat distortion = (cv::Mat_<double>(1,5) << 0.2963, -2.999, 0.0017, 0.0112,
40         20.5691); //obtained from matlab (Zhang's)
41     cv::Mat cv_K = (cv::Mat_<double>(3,3) << 13459, 78, 2983, 0, 13507, 1849, 0, 0,
42         1); //obtained from matlab (Zhang's)
43     cv::undistort(frame, undistorted_frame, cv_K, distortion);
44 */

```

```

41
42     // DLT
43     Calibrator calib(points_3d, points_2d);
44     std::vector<int> sample_indices(points_3d.rows());
45     std::iota(std::begin(sample_indices), std::end(sample_indices), 0);
46     calib.calibrateByDlt(sample_indices);
47     calib.drawOverlay(sample_indices, frame);
48
49     /* Image of world origin
50
51     Eigen::Vector4f X_origin;
52     Eigen::Vector3f x_origin;
53     X_origin << 0, 0, 0, 1;
54     x_origin = calib.getPMatrix() * X_origin;
55     x_origin = x_origin/x_origin(2);
56     std::cout << "Image of world origin: " << x_origin << std::endl;
57     cv::circle(frame, cv::Point(x_origin(0), x_origin(1)), 30, cv::Scalar(0, 255, 0), -1,
58                CV_AA);
59 */
60     // Decompose P
61     Eigen::MatrixXf K, R, c;
62     calib.decomposePMatrix(K, R, c);
63
64     cv::namedWindow("frame", CV_WINDOW_NORMAL);
65     cv::imshow("frame", frame);
66     cv::imwrite("../report/imgs/cube-overlay.png", frame);
67     cv::waitKey(0);
68
69     return 0;
70 }

```

Listing 4: run_ransac_dlt.cpp

```

1 #include <iostream>
2 #include <string>
3
4 #include "utils.hpp"
5 #include "calibrator.hpp"
6 #include <opencv2/core/eigen.hpp>
7
8 int main(int argc, char *argv[])
9 {
10     // std::string frame_path = "../data/dlt/IMG_5455.JPG";
11     // std::string points_2d_path = "../data/dlt/2d-points.txt";
12     // std::string points_3d_path = "../data/dlt/3d-points.txt";
13     // std::string points_2d_path = "../data/dlt/undistort-2d-points.txt";
14
15     std::string frame_path = "../data/phone-camera/dlt/cube.jpg";
16     std::string points_2d_path = "../data/phone-camera/dlt/2d-points.txt";
17     std::string points_3d_path = "../data/phone-camera/dlt/3d-points.txt";
18
19     Eigen::MatrixXf points_2d;
20     utils::loadMatrix(points_2d_path, utils::getNumberOfLines(points_2d_path), 2,

```

```

    points_2d);
21
22 Eigen::MatrixXf points_3d;
23 utils::loadMatrix(points_3d_path, utils::getNumberOfLines(points_3d_path), 3,
    points_3d);
24
25 if(points_2d.rows() != points_3d.rows())
26 {
27     std::cout << "Number of correspondences don't match!\n";
28     return -1;
29 }
30
31 cv::Mat frame;
32 frame = cv::imread(frame_path, 1);
33
34 /* Undistortion
35
36 cv::Mat undistorted_frame;
37 cv::Mat distortion = (cv::Mat_<double>(1,5) << 0.2963, -2.999, 0.0017, 0.0112,
    20.5691); //obtained from matlab (Zhang's)
38 cv::Mat cv_K = (cv::Mat_<double>(3,3) << 13459, 78, 2983, 0, 13507, 1849, 0, 0,
    1); //obtained from matlab (Zhang's)
39 cv::undistort(frame, undistorted_frame, cv_K, distortion);
40 */
41
42 //DLT with RANSAC calibration
43 Calibrator calib(points_3d, points_2d);
44 std::vector<int> inlier_indices;
45 calib.calibrateByDltRansac(40, 0.5, inlier_indices);
46 calib.drawOverlay(inlier_indices, frame);
47
48 // Decompose P
49 Eigen::MatrixXf K, R, c;
50 calib.decomposePMatrix(K, R, c);
51
52 cv::namedWindow("frame", CV_WINDOW_NORMAL);
53 cv::imshow("frame", frame);
54 cv::imwrite("../report/imgs/cube-ransac.png", frame);
55 cv::waitKey(0);
56
57 return 0;
58 }

```

Listing 5: CMakeLists.txt

```

1 cmake_minimum_required(VERSION 3.11)
2
3 project(calibration)
4
5 set(CMAKE_CXX_STANDARD 17)
6 set(CMAKE_BUILD_TYPE Debug)
7
8 find_package(OpenCV REQUIRED)
9

```

```

10 add_executable(run_dlt src/run_dlt.cpp src/utils.cpp src/calibrator.cpp)
11 add_executable(run_ransac_dlt src/run_ransac_dlt.cpp src/utils.cpp src/calibrator.
    cpp)
12 add_executable(run_undistorted_dlt src/run_undistorted_dlt.cpp src/utils.cpp src/
    calibrator.cpp)
13
14 target_link_libraries(run_dlt ${OpenCV_LIBS})
15 target_link_libraries(run_ransac_dlt ${OpenCV_LIBS})
16 target_link_libraries(run_undistorted_dlt ${OpenCV_LIBS})

```

Listing 6: run_zhangs.m

```

1 clear variables
2 clc
3 close all
4
5 for i = 1:15
6     %imageFileName = sprintf('img%d.jpg',i);
7     %imageFileNames{i} = fullfile(' ../data/phone-camera/zhang',imageFileName);
8
9     imageFileName = sprintf('img%d.JPG',i);
10    imageFileNames{i} = fullfile(' ../data/zhang',imageFileName);
11 end
12
13 [imagePoints,boardSize] = detectCheckerboardPoints(imageFileNames);
14
15 worldPoints = generateCheckerboardPoints(boardSize, 0.029);
16
17 [params,imagesUsed,estimataionErrors] = estimateCameraParameters(imagePoints,
    worldPoints, ...
18 'EstimateSkew',true,'NumRadialDistortionCoefficients',3, '
    EstimateTangentialDistortion',true);
19
20 figure, imshow(imageFileNames{1}), hold on
21 plot(imagePoints(:,1,1),imagePoints(:,2,1),'ro-','LineWidth',2,'MarkerFaceColor','
    r');
22
23 plot(params.ReprojectedPoints(:,1,1),params.ReprojectedPoints(:,2,1),'bo-','
    LineWidth',2,'MarkerFaceColor','b');
24
25 disp('Average reprojection error:');
26 disp(params.MeanReprojectionError);
27
28 disp('Calibration matrix:');
29 disp(params.IntrinsicMatrix);
30
31 disp('Radial distortion:');
32 disp(params.RadialDistortion);
33
34 disp('Tangential distortion:');
35 disp(params.TangentialDistortion);

```