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 corespondences. 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 $2N * 12$ 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.

```

 void Calibrator::calibrateByDlt(const std::vector<int> &sample_indices)
{
  if(sample_indices.size() < 6)
  {
    std::cout << "Cannot run DLT! Require at least 6 correspondences.\n";
    return;
  }

  Eigen::MatrixXf X_samples(sample_indices.size(),4);
  Eigen::MatrixXf x_samples(sample_indices.size(),3);

  if(sample_indices.size() == X.rows())
  {
    X_samples = X;
    x_samples = x;
  }

  else
  { int j = 0;
    for (int i : sample_indices)
    {
      X_samples.row(j) = X.row(i);
      x_samples.row(j++) = x.row(i);
    }
  }

  Eigen::ArrayXf p(12);
  Eigen::MatrixXf M(2*x_samples.rows(),12);

```

```
  // Build M
  for(int i = 0,j=0; i < M.rows(); i+=2,j++)
  {
    M.row(i) << X_samples.row(j) * -1, Eigen::MatrixXf::Zero(1,4), X_samples.row(j) * x_samples(j,0);
    M.row(i+1) << Eigen::MatrixXf::Zero(1,4), X_samples.row(j) * -1, X_samples.row(j) * x_samples(j,1);
  }

  // Estimate p
  Eigen::JacobiSVD<Eigen::MatrixXf> svd(M, Eigen::ComputeFullU | Eigen::ComputeFullV);
  Eigen::MatrixXf V;
  V = svd.matrixV();
  p = V.col(V.cols()-1);

  // Build P
  P.resize(3,4);
  P.row(0) = p.segment(0,4);
  P.row(1) = p.segment(4,4);
  P.row(2) = p.segment(8,4);
  std::cout << "P:\n" << P << std::endl;
  std::cout << "Average reprojection error:\n" << calcAvgReprojectionError() << std::endl;
}

```

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.

```

void Calibrator::calibrateByDltRansac(const float &dist_threshold, const float &inlier_ratio, std::vector<int> &inlier_indices)
{
  std::cout << "Running RANSAC...." << std::endl;
  std::vector<int> largest_support;

  for(int n = 0; n < 500; n++)
```

```cpp
  {
    std::cout << "\n\nIteration #" << n+1 << std::endl;
    std::vector<int> sample_indices = utils::generateRandomVector(0,X.rows()-1,6);
    calibrateByDlt(sample_indices);

    for(int i = 0; i < X.rows(); i++)
    {
      float dist = calcReprojectionError(X.row(i),x.row(i));
      if(dist < dist_threshold)
        inlier_indices.push_back(i);
    }

    if(inlier_indices.size() >= inlier_ratio * X.rows())
    {
      std::cout << "Found a model!\n" << "Number of inliers: " << inlier_indices.
     size() << std::endl;
      std::cout << "Inliers: ";
      for(int i : inlier_indices)
        std::cout << i << " ";

      std::cout << "\n";
      calibrateByDlt(inlier_indices);
      return;
    }

    else
    {
      if(largest_support.size() <= inlier_indices.size())
        largest_support = inlier_indices;

      inlier_indices.clear();
    }
  }

  if(largest_support.size() >= 6)
  {
    std::cout << "Could not find a model according to threshold!\nSo using largest
     inlier set instead." << std::endl;
    calibrateByDlt(largest_support);
    inlier_indices = largest_support;
    std::cout << "Number of inliers: " << inlier_indices.size() << std::endl;
    for(int i : inlier_indices)
      std::cout << i << " ";

    std::cout << "\n";
  }

  else
    std::cout << "Could not find a model!" << std::endl;
}
```

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

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

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

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

$$\text{Average reprojection error}: 310.596$$

DLT with RANSAC

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

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

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

$$\text{Average reprojection error}: 447.887$$

$$\text{Reprojection error threshold}: 250$$

$$\text{Inlier ratio threshold}: 0.8$$

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 `estimateCamera-Parameters` 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:

$$\text{Radial} : 0.2963, -2.999, 20.5691$$
$$\text{Tangential} : 0.0017, 0.0112$$

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.

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

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

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

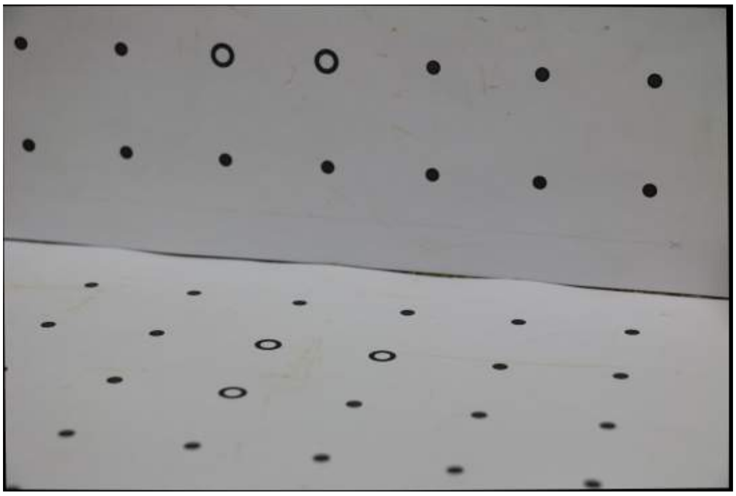

Figure 1: Undistorted image

DLT

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

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

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

$$\text{Average reprojection error}: 318.676$$

DLT with RANSAC

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

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

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

$$\text{Average reprojection error}: 503.276$$

$$\text{Reprojection error threshold}: 200$$

$$\text{Inlier ratio threshold}: 0.7$$

5 Wireframe overlay

DLT

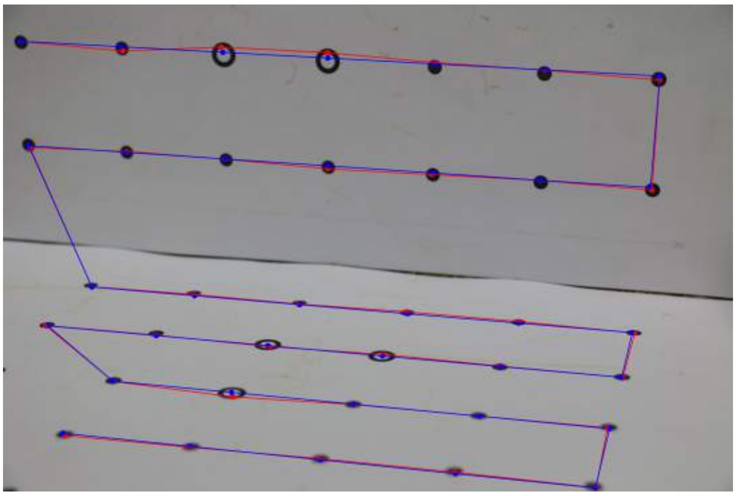

Figure 2: Wireframe overlayed over the 3D object after DLT calibration. Blue indicates the estimated points, red indicates the measured points.

DLT with RANSAC

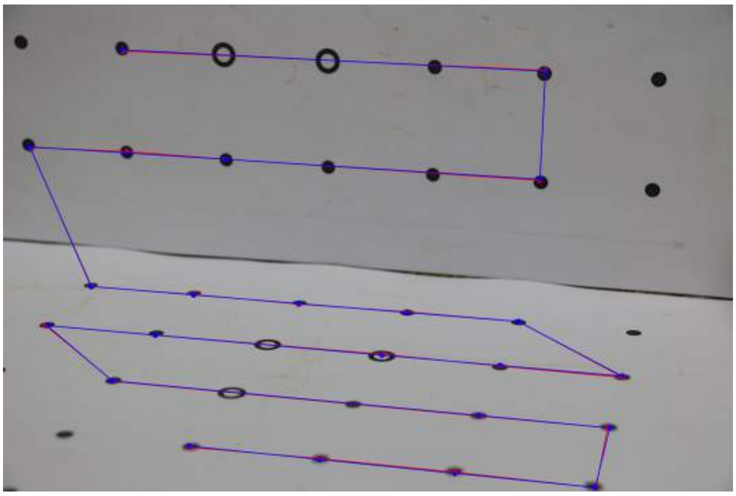

Figure 3: Wireframe of only the inlier points overlayed over the 3D object after DLT - RANSAC calibration. Blue indicates the estimated points, red indicates the measured points.

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:

$$\text{Calibration matrix : } \begin{pmatrix} 13459 & 78 & 2983 \\ 0 & 13507 & 1849 \\ 0 & 0 & 1 \end{pmatrix}$$
$$\text{Average reprojection error : } 1.6417$$

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

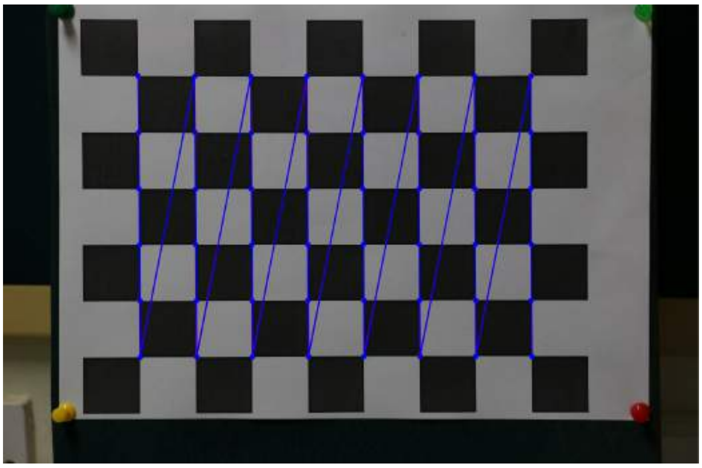

Figure 4: Wireframe overlayed over the checkerboard after calibration. Blue indicates the estimated points, red indicates the measured points.

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.

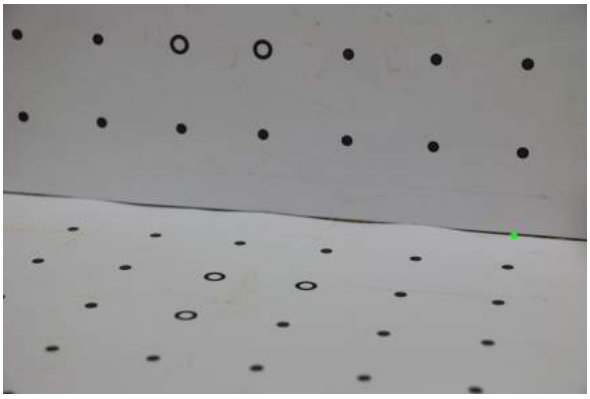

Figure 5: Image of the world origin

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.

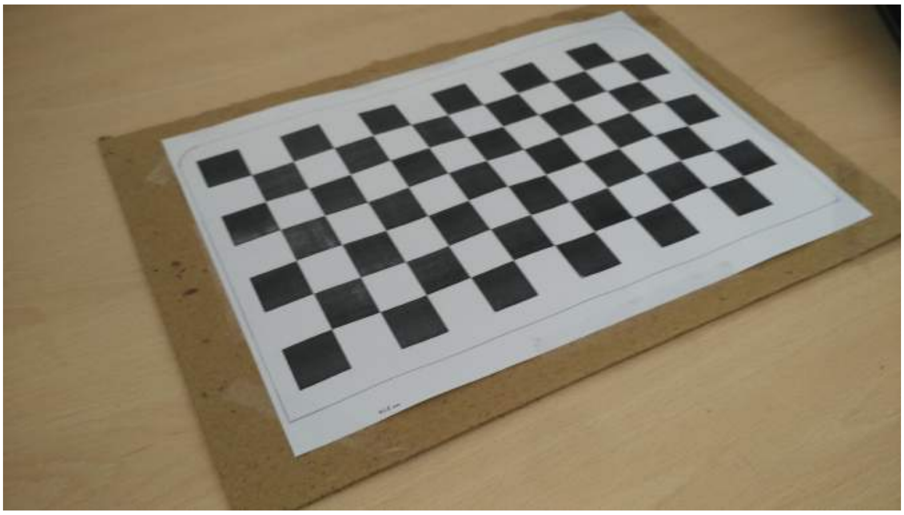

Figure 7: Calibration object for Zhang's, captured with phone camera.

10 Results using own setup

DLT

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

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

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

$$\text{Average reprojection error}: 86.6504$$

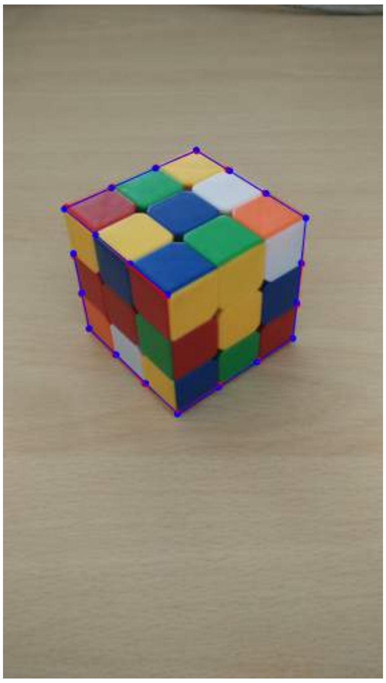

Figure 8: Wireframe overlayed over the cube object after calibration. Blue indicates projected points, red indicates measured points.

DLT with RANSAC

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

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

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

$$\text{Average reprojection error}: 198.293$$

$$\text{Reprojection error threshold}: 50$$

$$\text{Inlier ratio threshold}: 0.5$$

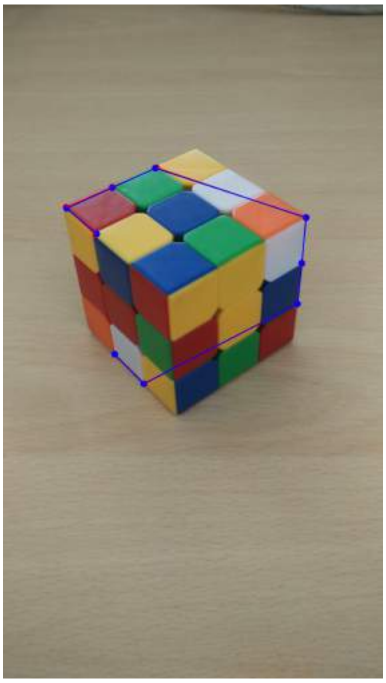

Figure 9: Wireframe of only the inlier points overlayed over the cube object after calibration. Blue indicates projected points, red indicates measured points.

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

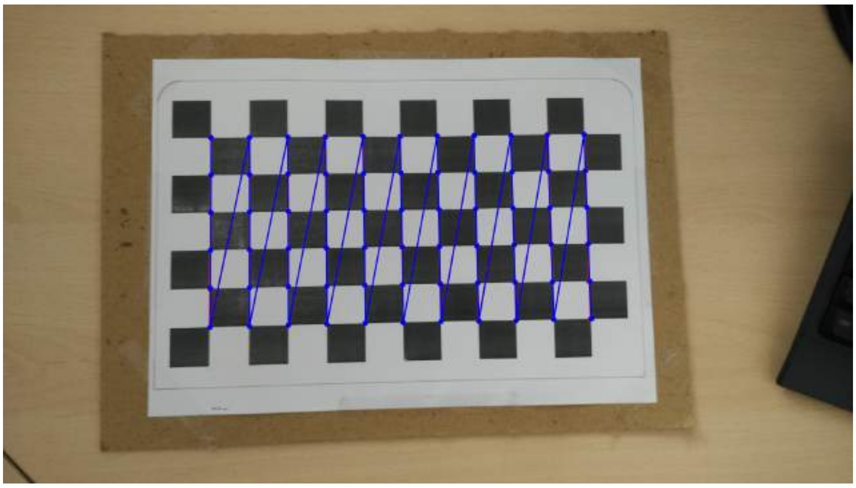

Figure 10: Wireframe overlayed over the checkerboard after calibration. Blue indicates the estimated points, red indicates the measured points.

11 Code

All the code files can be found here. The main algorithms are implemented in Listing 2.

Listing 1: calibrator.hpp

```
#pragma once

#include <iostream>

#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>

#include <Eigen/Core>
#include <Eigen/SVD>
#include <Eigen/QR>
#include <Eigen/LU>

#include "utils.hpp"

class Calibrator
{
  public:
    Calibrator(const Eigen::MatrixXf &points_3d, const Eigen::MatrixXf &points_2d);
    void calibrateByDlt(const std::vector<int> &sample_indices);
    void calibrateByDltRansac(const float &dist_threshold, const float &inlier_ratio, std::vector<int> &inlier_indices);
    float calcReprojectionError(const Eigen::Vector4f &pt_3d, const Eigen::Vector3f &pt_img);
```

```
    float calcSetReprojectionError(const Eigen::MatrixXf &pts_3d, const Eigen::
    MatrixXf &pts_img);
    float calcAvgReprojectionError();
    void decomposePMatrix(Eigen::MatrixXf &K, Eigen::MatrixXf &R, Eigen::MatrixXf
    &c);
    void drawOverlay(const std::vector<int> &sample_indices, cv::Mat &frame);
    Eigen::MatrixXf getPMatrix();

  private:
    Eigen::MatrixXf X;
    Eigen::MatrixXf x;
    Eigen::MatrixXf P;
};
```

Listing 2: calibrator.cpp

```
#include "calibrator.hpp"
#include <stdlib.h>

Calibrator::Calibrator(const Eigen::MatrixXf &points_3d, const Eigen::MatrixXf &
    points_2d)
{
  x = points_2d;
  x.conservativeResize(x.rows(),x.cols()+1);
  x.col(x.cols()-1).setOnes();

  X = points_3d;
  X = X * 0.01; // assumes points are in cm, converting to m
  X.conservativeResize(X.rows(),X.cols()+1);
  X.col(X.cols()-1).setOnes();
}

void Calibrator::calibrateByDlt(const std::vector<int> &sample_indices)
{
  if(sample_indices.size() < 6)
  {
    std::cout << "Cannot run DLT! Require at least 6 correspondences.\n";
    return;
  }

  Eigen::MatrixXf X_samples(sample_indices.size(),4);
  Eigen::MatrixXf x_samples(sample_indices.size(),3);

  if(sample_indices.size() == X.rows())
  {
    X_samples = X;
    x_samples = x;
  }

  else
  { int j = 0;
    for (int i : sample_indices)
    {
      X_samples.row(j) = X.row(i);
```

```
        x_samples.row(j++) = x.row(i);
      }
    }

    Eigen::ArrayXf p(12);
    Eigen::MatrixXf M(2*x_samples.rows(),12);

    // Build M
    for(int i = 0,j=0; i < M.rows(); i+=2,j++)
    {
      M.row(i) << X_samples.row(j) * -1, Eigen::MatrixXf::Zero(1,4), X_samples.row(j) * x_samples(j,0);
      M.row(i+1) << Eigen::MatrixXf::Zero(1,4), X_samples.row(j) * -1, X_samples.row(j) * x_samples(j,1);
    }

    // Estimate p
    Eigen::JacobiSVD<Eigen::MatrixXf> svd(M,Eigen::ComputeFullU | Eigen::ComputeFullV);
    Eigen::MatrixXf V;
    V = svd.matrixV();
    p = V.col(V.cols()-1);

    // Build P
    P.resize(3,4);
    P.row(0) = p.segment(0,4);
    P.row(1) = p.segment(4,4);
    P.row(2) = p.segment(8,4);
    std::cout << "P:\n" << P << std::endl;
    std::cout << "Average reprojection error:\n" << calcAvgReprojectionError() << std::endl;
}

void Calibrator::calibrateByDltRansac(const float &dist_threshold, const float &inlier_ratio, std::vector<int> &inlier_indices)
{
    std::cout << "Running RANSAC...." << std::endl;
    std::vector<int> largest_support;

    for(int n = 0; n < 500; n++)
    {
      std::cout << "\n\nIteration #" << n+1 << std::endl;
      std::vector<int> sample_indices = utils::generateRandomVector(0,X.rows()-1,6);
      calibrateByDlt(sample_indices);

      for(int i = 0; i < X.rows(); i++)
      {
        float dist = calcReprojectionError(X.row(i),x.row(i));
        if(dist < dist_threshold)
          inlier_indices.push_back(i);
      }

      if(inlier_indices.size() >= inlier_ratio * X.rows())
```

```
    {
      std::cout << "Found a model!\n" << "Number of inliers: " << inlier_indices.size() << std::endl;
      std::cout << "Inliers: ";
      for(int i : inlier_indices)
        std::cout << i << " ";

      std::cout << "\n";
      calibrateByDlt(inlier_indices);
      return;
    }

    else
    {
      if(largest_support.size() <= inlier_indices.size())
        largest_support = inlier_indices;

      inlier_indices.clear();
    }
  }

  if(largest_support.size() >= 6)
  {
    std::cout << "Could not find a model according to threshold!\nSo using largest inlier set instead." << std::endl;
    calibrateByDlt(largest_support);
    inlier_indices = largest_support;
    std::cout << "Number of inliers: " << inlier_indices.size() << std::endl;
    for(int i : inlier_indices)
      std::cout << i << " ";

    std::cout << "\n";
  }

  else
    std::cout << "Could not find a model!" << std::endl;
}

float Calibrator::calcReprojectionError(const Eigen::Vector4f &pt_3d, const Eigen::Vector3f &pt_img)
{
  Eigen::Vector3f est_pt_img = P * pt_3d;
  est_pt_img = est_pt_img / est_pt_img(2);
  float error = (est_pt_img - pt_img).squaredNorm();

  return error;
}


float Calibrator::calcSetReprojectionError(const Eigen::MatrixXf &pts_3d, const Eigen::MatrixXf &pts_img)
{
  Eigen::MatrixXf est_pts_img = P * pts_3d.transpose();
```

```
  Eigen::MatrixXf scale = est_pts_img.row(2).replicate(3,1);
  est_pts_img = est_pts_img.array() / scale.array();

  float total_error = 0;

  for(int i = 0; i < pts_3d.rows(); i++)
  {
    total_error += (est_pts_img.col(i) - pts_img.row(i).transpose()).squaredNorm()
    ;
  }

  float avg_error = total_error/pts_3d.rows();
  return avg_error;
}

float Calibrator::calcAvgReprojectionError()
{
  Eigen::MatrixXf est_pts_img = P * X.transpose();
  Eigen::MatrixXf scale = est_pts_img.row(2).replicate(3,1);
  est_pts_img = est_pts_img.array() / scale.array();

  float total_error = 0;

  for(int i = 0; i < X.rows(); i++)
  {
    total_error += (est_pts_img.col(i) - x.row(i).transpose()).squaredNorm();
  }

  float avg_error = total_error/X.rows();
  return avg_error;
 }

void Calibrator::decomposePMatrix(Eigen::MatrixXf &K, Eigen::MatrixXf &R, Eigen::
    MatrixXf &c)
{
  Eigen::MatrixXf H,p4;
  H = P.block(0,0,3,3);
  p4 = P.col(3);

  std::cout << "H:\n" << H << std::endl;
  std::cout << "p4:\n" << p4 << std::endl;

  //Camera center
  c = -1 * H.inverse() * p4;

  //Calibration matrix, rotation matrix
  Eigen::HouseholderQR<Eigen::MatrixXf> qr(H.inverse());
  R = qr.householderQ();
  K = qr.matrixQR().triangularView<Eigen::Upper>();

  R = R.inverse();
  K = K.inverse();
  K = K / K(2,2);
```

```cpp

  std::cout << "R:\n" << R << std::endl;
  std::cout << "K:\n" << K << std::endl;
}

void Calibrator::drawOverlay(const std::vector<int> &sample_indices, cv::Mat &
    frame)
{
  Eigen::MatrixXf X_samples(sample_indices.size(),4);
  Eigen::MatrixXf x_samples(sample_indices.size(),3);

  if(sample_indices.size() == X.rows())
  {
    X_samples = X;
    x_samples = x;
  }

  else
  { int j = 0;
    for (int i : sample_indices)
    {
      X_samples.row(j) = X.row(i);
      x_samples.row(j++) = x.row(i);
    }
  }

  Eigen::MatrixXf est_x, est_u, est_v;
  est_x = P * X_samples.transpose();

  //std::cout << "image: \n" << x << std::endl;

  est_u = est_x.row(0).array() / est_x.row(2).array();
  est_v = est_x.row(1).array() / est_x.row(2).array();

  //std::cout << "u:\n" << u << std::endl;
  //std::cout << "test: u(2) \n"<< u(2) << std::endl;

  //Mark points
  for(int i = 0; i < x_samples.rows(); i++)
  {
    cv::circle(frame,cv::Point(x_samples(i,0),x_samples(i,1)),20,cv::Scalar
    (0,0,255),-1,CV_AA);
    cv::circle(frame,cv::Point(est_u(i),est_v(i)),20,cv::Scalar(255,0,0),-1,CV_AA)
    ;
  }

  //Draw lines
  for(int i = 0; i < x_samples.rows() - 1; i++)
  {
    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);
    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);
```

```cpp
  }
}

Eigen::MatrixXf Calibrator::getPMatrix()
{
  return P;
}
```

Listing 3: run_dlt.cpp

```cpp
#include <iostream>
#include <string>

#include "utils.hpp"
#include "calibrator.hpp"
#include <opencv2/core/eigen.hpp>

int main(int argc, char *argv[])
{
  // std::string frame_path = "../data/dlt/IMG_5455.JPG";
  // std::string points_2d_path = "../data/dlt/2d-points.txt";
  // std::string points_3d_path = "../data/dlt/3d-points.txt";
    // std::string points_2d_path = "../data/dlt/undistort-2d-points.txt";

  std::string frame_path = "../data/phone-camera/dlt/cube.jpg";
  std::string points_2d_path = "../data/phone-camera/dlt/2d-points.txt";
  std::string points_3d_path = "../data/phone-camera/dlt/3d-points.txt";

  Eigen::MatrixXf points_2d;
  utils::loadMatrix(points_2d_path, utils::getNumberofLines(points_2d_path),2, points_2d);

  Eigen::MatrixXf points_3d;
  utils::loadMatrix(points_3d_path, utils::getNumberofLines(points_3d_path),3, points_3d);

  if(points_2d.rows() != points_3d.rows())
  {
    std::cout << "Number of correspondences don't match!\n";
    return -1;
  }

  cv::Mat frame;
  frame = cv::imread(frame_path,1);

/* Undistortion

  cv::Mat undistorted_frame;
  cv::Mat distortion = (cv::Mat_<double>(1,5) << 0.2963, -2.999, 0.0017, 0.0112, 20.5691); //obtained from matlab (Zhang's)
  cv::Mat cv_K = (cv::Mat_<double>(3,3) << 13459, 78, 2983, 0, 13507, 1849, 0, 0, 1); //obtained from matlab (Zhang's)
  cv::undistort(frame, undistorted_frame, cv_K, distortion);
 */
```

```

    // DLT
  Calibrator calib(points_3d, points_2d);
  std::vector <int> sample_indices(points_3d.rows());
  std::iota(std::begin(sample_indices), std::end(sample_indices), 0);
  calib.calibrateByDlt(sample_indices);
  calib.drawOverlay(sample_indices,frame);

/*  Image of world origin

  Eigen::Vector4f X_origin;
  Eigen::Vector3f x_origin;
  X_origin << 0, 0, 0, 1;
  x_origin = calib.getPMatrix() * X_origin;
  x_origin = x_origin/x_origin(2);
  std::cout << "Image of world origin: " << x_origin << std::endl;
  cv::circle(frame,cv::Point(x_origin(0),x_origin(1)),30,cv::Scalar(0,255,0),-1,
    CV_AA);
 */

  // Decompose P
  Eigen::MatrixXf K, R, c;
  calib.decomposePMatrix(K,R,c);

  cv::namedWindow("frame",CV_WINDOW_NORMAL);
  cv::imshow("frame",frame);
    cv::imwrite("../report/imgs/cube-overlay.png",frame);
  cv::waitKey(0);

  return 0;
}
```

Listing 4: run_ransac_dlt.cpp

```
#include <iostream>
#include <string>

#include "utils.hpp"
#include "calibrator.hpp"
#include <opencv2/core/eigen.hpp>

int main(int argc, char *argv[])
{
  // std::string frame_path = "../data/dlt/IMG_5455.JPG";
  // std::string points_2d_path = "../data/dlt/2d-points.txt";
  // std::string points_3d_path = "../data/dlt/3d-points.txt";
    // std::string points_2d_path = "../data/dlt/undistort-2d-points.txt";

  std::string frame_path = "../data/phone-camera/dlt/cube.jpg";
  std::string points_2d_path = "../data/phone-camera/dlt/2d-points.txt";
  std::string points_3d_path = "../data/phone-camera/dlt/3d-points.txt";

  Eigen::MatrixXf points_2d;
  utils::loadMatrix(points_2d_path,utils::getNumberofLines(points_2d_path),2,
```

```
    points_2d);

  Eigen::MatrixXf points_3d;
  utils::loadMatrix(points_3d_path,utils::getNumberofLines(points_3d_path),3,
    points_3d);

  if(points_2d.rows() != points_3d.rows())
  {
    std::cout << "Number of correspondences don't match!\n";
    return -1;
  }

  cv::Mat frame;
  frame = cv::imread(frame_path,1);

/* Undistortion

  cv::Mat undistorted_frame;
  cv::Mat distortion = (cv::Mat_<double>(1,5) << 0.2963, -2.999, 0.0017, 0.0112,
    20.5691); //obtained from matlab (Zhang's)
  cv::Mat cv_K = (cv::Mat_<double>(3,3) << 13459, 78, 2983, 0, 13507, 1849, 0, 0,
    1); //obtained from matlab (Zhang's)
  cv::undistort(frame,undistorted_frame,cv_K,distortion);
 */

    //DLT with RANSAC calibration
  Calibrator calib(points_3d, points_2d);
    std::vector<int> inlier_indices;
    calib.calibrateByDltRansac(40, 0.5, inlier_indices);
  calib.drawOverlay(inlier_indices,frame);

  // Decompose P
  Eigen::MatrixXf K, R, c;
  calib.decomposePMatrix(K,R,c);

  cv::namedWindow("frame",CV_WINDOW_NORMAL);
  cv::imshow("frame",frame);
    cv::imwrite("../report/imgs/cube-ransac.png",frame);
  cv::waitKey(0);

  return 0;
}
```

Listing 5: CMakeLists.txt

```
cmake_minimum_required(VERSION 3.11)

project(calibration)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_BUILD_TYPE Debug)

find_package(OpenCV REQUIRED)

```

```
add_executable(run_dlt src/run_dlt.cpp src/utils.cpp src/calibrator.cpp)
add_executable(run_ransac_dlt src/run_ransac_dlt.cpp src/utils.cpp src/calibrator.cpp)
add_executable(run_undistorted_dlt src/run_undistorted_dlt.cpp src/utils.cpp src/calibrator.cpp)

target_link_libraries(run_dlt ${OpenCV_LIBS})
target_link_libraries(run_ransac_dlt ${OpenCV_LIBS})
target_link_libraries(run_undistorted_dlt ${OpenCV_LIBS})
```

Listing 6: run_zhangs.m

```
clear variables
clc
close all

for i = 1:15
    %imageFileName = sprintf('img%d.jpg',i);
    %imageFileNames{i} = fullfile('../data/phone-camera/zhang',imageFileName);

    imageFileName = sprintf('img%d.JPG',i);
    imageFileNames{i} = fullfile('../data/zhang',imageFileName);
end

[imagePoints,boardSize] = detectCheckerboardPoints(imageFileNames);

worldPoints = generateCheckerboardPoints(boardSize, 0.029);

[params,imagesUsed,estimataionErrors] = estimateCameraParameters(imagePoints, worldPoints, ...
'EstimateSkew',true','NumRadialDistortionCoefficients',3, 'EstimateTangentialDistortion',true);

figure, imshow(imageFileNames{1}), hold on
plot(imagePoints(:,1,1),imagePoints(:,2,1),'ro-','LineWidth',2,'MarkerFaceColor','r');

plot(params.ReprojectedPoints(:,1,1),params.ReprojectedPoints(:,2,1),'bo-','LineWidth',2,'MarkerFaceColor','b');

disp('Average reprojection error:');
disp(params.MeanReprojectionError);

disp('Calibration matrix:');
disp(params.IntrinsicMatrix');

disp('Radial distortion:');
disp(params.RadialDistortion);

disp('Tangential distortion:');
disp(params.TangentialDistortion);
```