Exercise 5: Geometric transforms

This week includes 3 exercises: The first is derivation of equations by hand. The second is the use of OpenCV methods estimate a projective transform. The third is the use of OpenCV to calibrate a camera.

- The estimation of the perspective transform is done by solving the equation Ax=b. This equation is formulated from the homogenous equations for four sets of image point pairs. The first exercise is to follow the derivation in the slides (or BB ch. 21.1.4) and do it yourself by hand. Express the solution x of the equation Ax=b using the pseudo inverse.
  - The purpose of this exercise is to become familiar with:
    1) homogeneous coordinates
    2) expressing point correspondences as linear equations, and rearranging and stacking them into a single matrix equation Ax=b
    3) Solving Ax=b using the pseudo inverse
- Load the image chessboard.jpg as grayscale. Manually define four point in the image and define four corresponding points in the output image such that the chessboard becomes a square. You can e.g. use mouse input to select these points. Estimate the perspective transform using OpenCV or make your own implementation. Apply the estimated transformation to the image.
- Load the calibration/image*.png images and calibrate the camera. You can use the provided calibration code as a starting point. Note that you only have to write code at the places in the framework marked with numbers. It is similar to the code provided for the semester project.

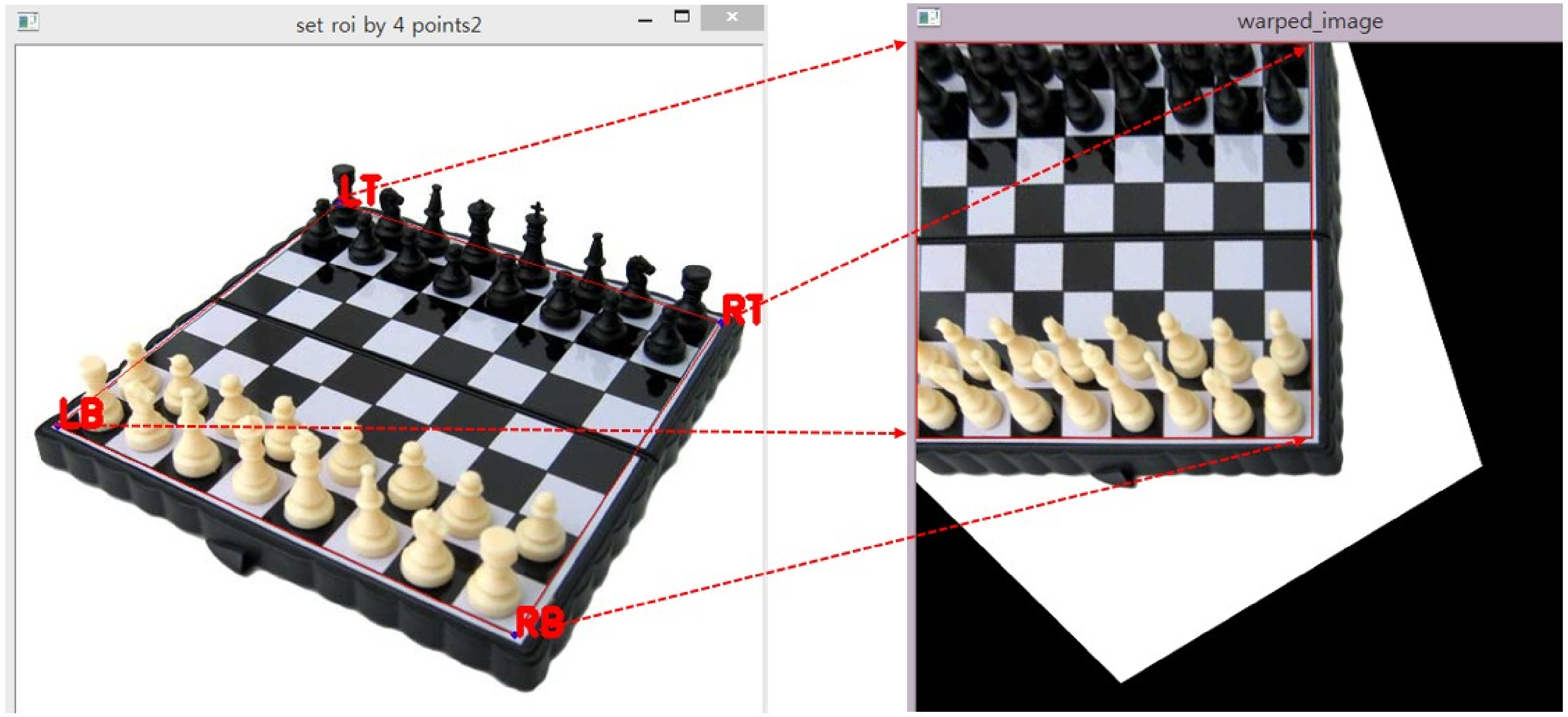
Derivation

$$\begin{bmatrix} i\cdot w \\ j\cdot w \\ w \end{bmatrix} = \begin{bmatrix} p_{00} & p_{01} & p_{02} \\ p_{10} & p_{11} & p_{12} \\ p_{20} & p_{21} & 1 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} i_1 \\ j_1 \\ i_2 \\ j_2 \\ i_3 \\ j_3 \\ i_4 \\ j_4 \end{bmatrix} = \begin{bmatrix} i'_1 & j'_1 & 1 & 0 & 0 & 0 & -i_1 i'_1 & -i_1 j'_1 \\ 0 & 0 & 0 & i'_1 & j'_1 & 1 & -j_1 i'_1 & -j_1 j'_1 \\ i'_2 & j'_2 & 1 & 0 & 0 & 0 & -i_2 i'_2 & -i_2 j'_2 \\ 0 & 0 & 0 & i'_2 & j'_2 & 1 & -j_2 i'_2 & -j_2 j'_2 \\ i'_3 & j'_3 & 1 & 0 & 0 & 0 & -i_3 i'_3 & -i_3 j'_3 \\ 0 & 0 & 0 & i'_3 & j'_3 & 1 & -j_3 i'_3 & -j_3 j'_3 \\ i'_4 & j'_4 & 1 & 0 & 0 & 0 & -i_4 i'_4 & -i_4 j'_4 \\ 0 & 0 & 0 & i'_4 & j'_4 & 1 & -j_4 i'_4 & -j_4 j'_4 \end{bmatrix} \begin{bmatrix} p_{00} \\ p_{01} \\ p_{02} \\ p_{10} \\ p_{11} \\ p_{12} \\ p_{20} \\ p_{21} \end{bmatrix}$$

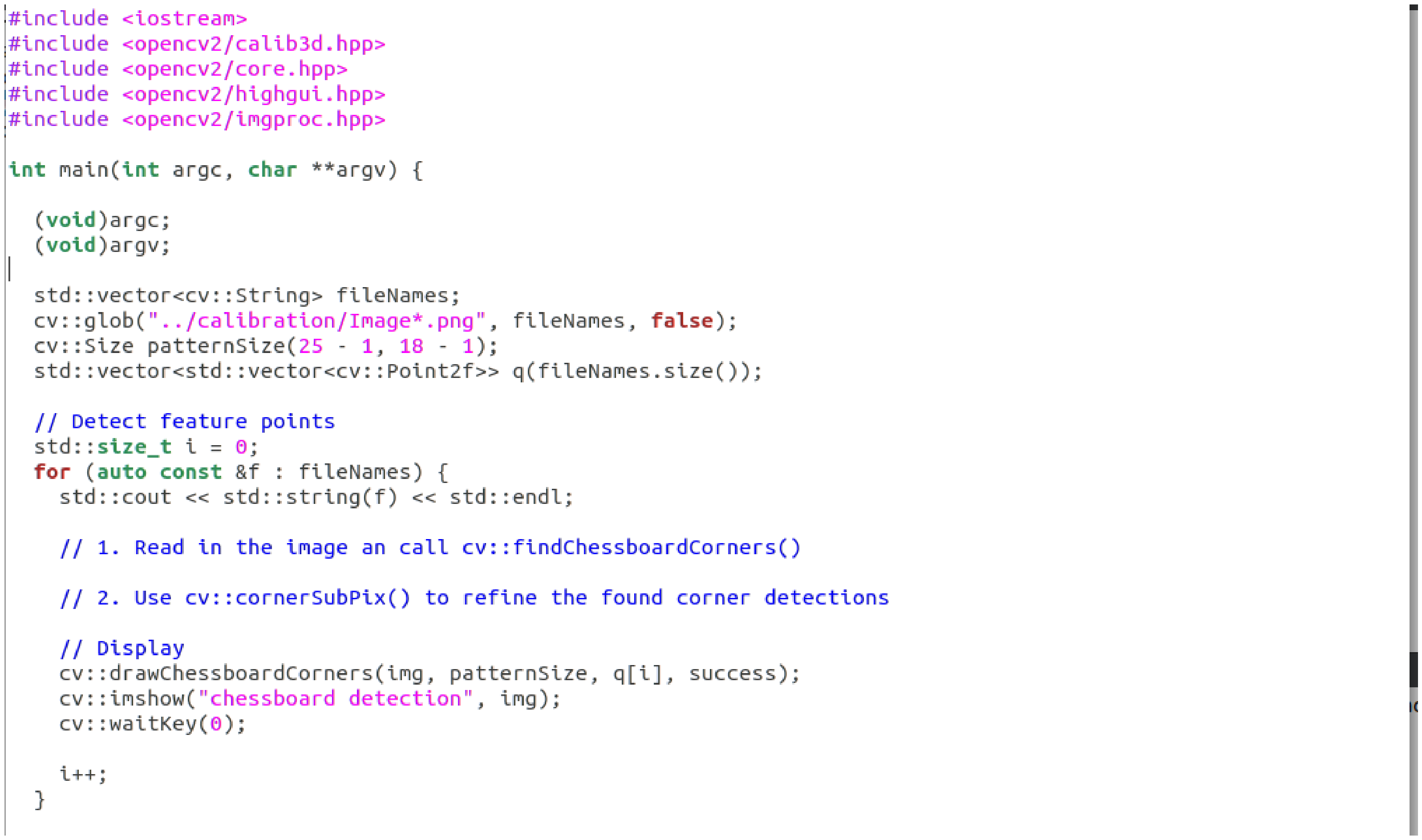
$$\hat{x} = (A^\top A)^{-1} A^\top b = A^\dagger b$$

Homography exercise



Part 1 of calibration code

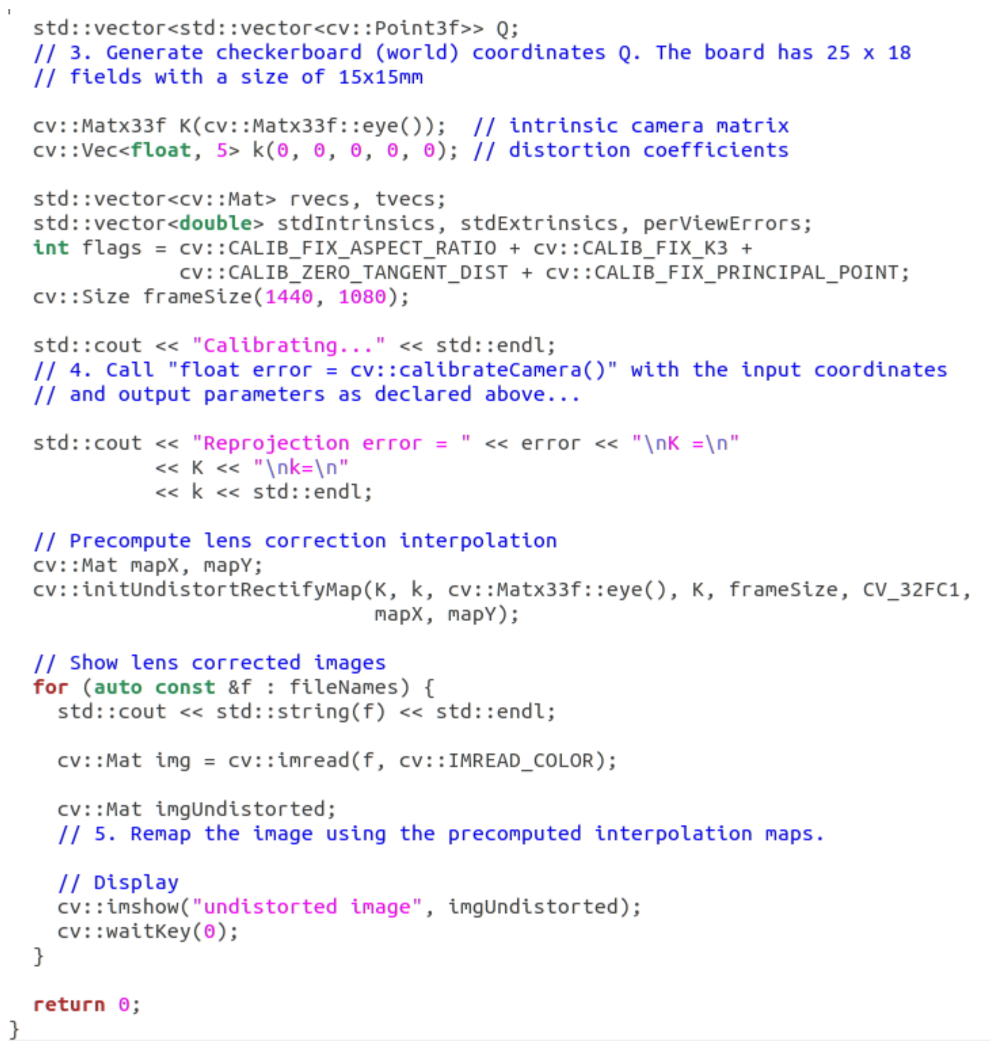
Load images and locate chessboard corners with sub-pixel precision



```cpp
#include <iostream>
#include <opencv2/calib3d.hpp>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>

int main(int argc, char **argv) {

  (void)argc;
  (void)argv;

  std::vector<cv::String> fileNames;
  cv::glob("../calibration/Image*.png", fileNames, false);
  cv::Size patternSize(25 - 1, 18 - 1);
  std::vector<std::vector<cv::Point2f>> q(fileNames.size());

  // Detect feature points
  std::size_t i = 0;
  for (auto const &f : fileNames) {
    std::cout << std::string(f) << std::endl;

    // 1. Read in the image an call cv::findChessboardCorners()

    // 2. Use cv::cornerSubPix() to refine the found corner detections

    // Display
    cv::drawChessboardCorners(img, patternSize, q[i], success);
    cv::imshow("chessboard detection", img);
    cv::waitKey(0);

    i++;
  }
```

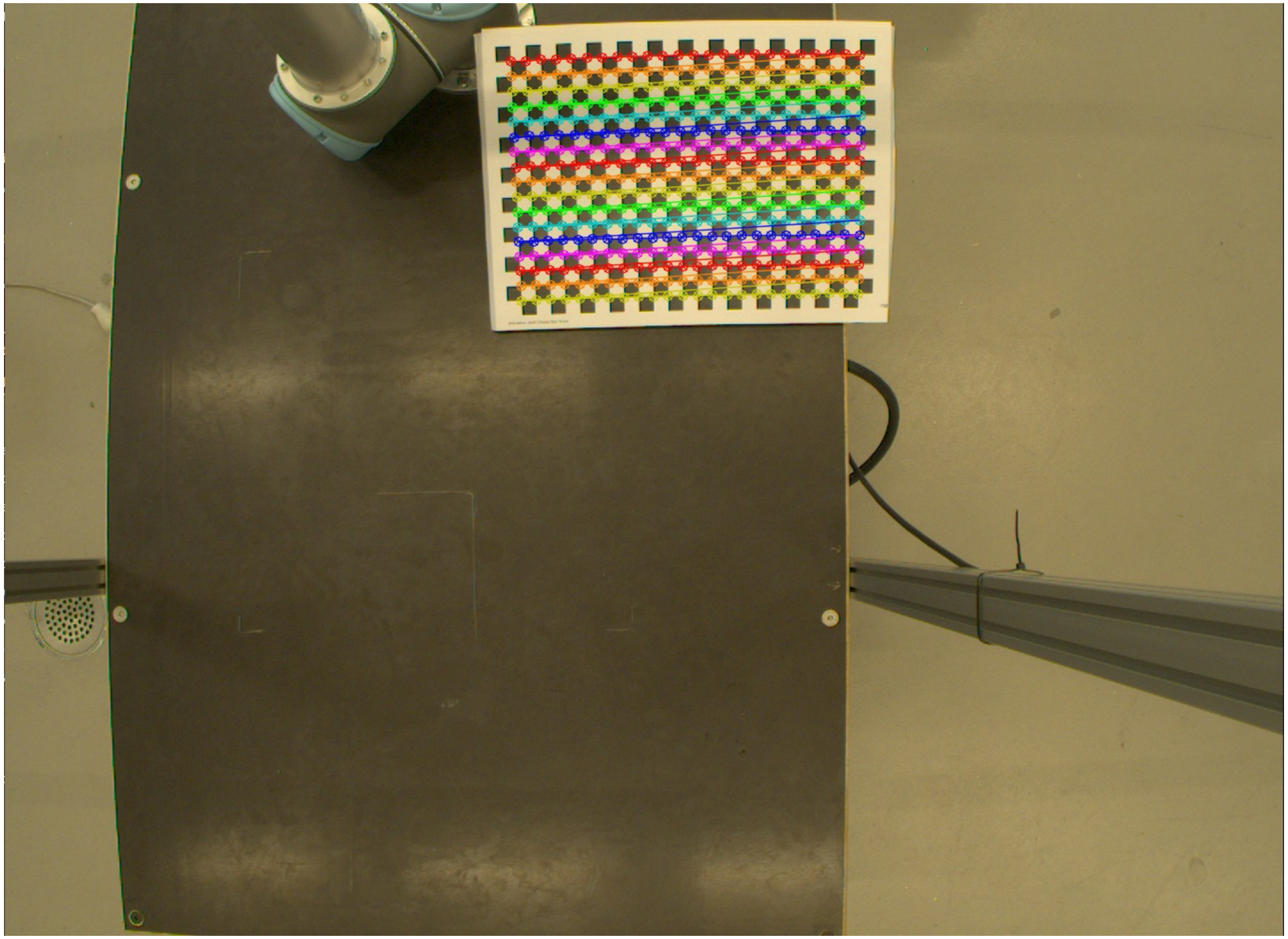
Part 2 of calibration code

Define chessboard coordinates real world coordinates, call the calibrateCamera method, and undistort the images using the computed camera matrix and distortion coefficients

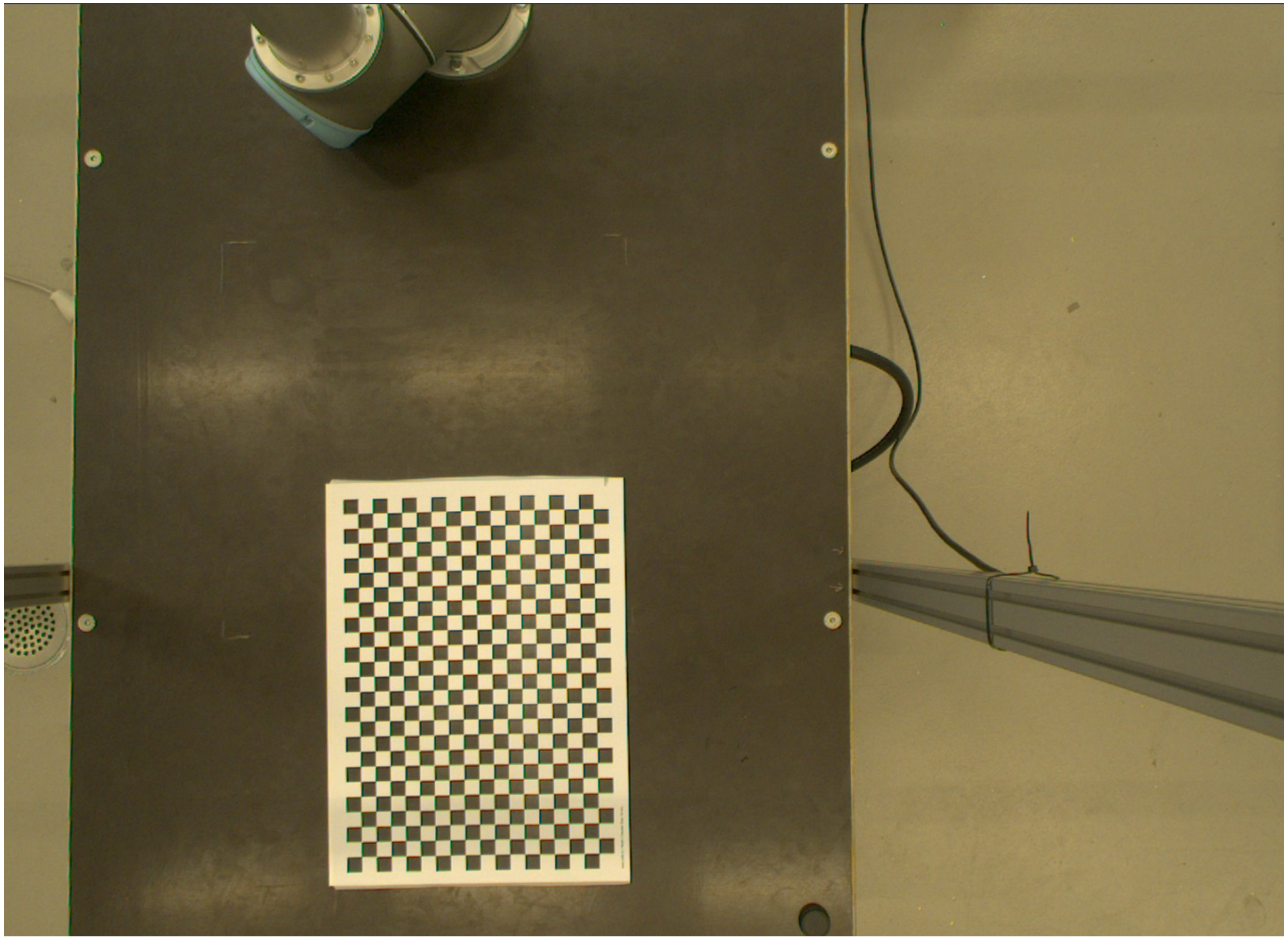


```
  std::vector<std::vector<cv::Point3f>> Q;
  // 3. Generate checkerboard (world) coordinates Q. The board has 25 x 18
  // fields with a size of 15x15mm

  cv::Matx33f K(cv::Matx33f::eye());  // intrinsic camera matrix
  cv::Vec<float, 5> k(0, 0, 0, 0, 0); // distortion coefficients

  std::vector<cv::Mat> rvecs, tvecs;
  std::vector<double> stdIntrinsics, stdExtrinsics, perViewErrors;
  int flags = cv::CALIB_FIX_ASPECT_RATIO + cv::CALIB_FIX_K3 +
              cv::CALIB_ZERO_TANGENT_DIST + cv::CALIB_FIX_PRINCIPAL_POINT;
  cv::Size frameSize(1440, 1080);

  std::cout << "Calibrating..." << std::endl;
  // 4. Call "float error = cv::calibrateCamera()" with the input coordinates
  // and output parameters as declared above...

  std::cout << "Reprojection error = " << error << "\nK =\n"
            << K << "\nk=\n"
            << k << std::endl;

  // Precompute lens correction interpolation
  cv::Mat mapX, mapY;
  cv::initUndistortRectifyMap(K, k, cv::Matx33f::eye(), K, frameSize, CV_32FC1,
                              mapX, mapY);

  // Show lens corrected images
  for (auto const &f : fileNames) {
    std::cout << std::string(f) << std::endl;

    cv::Mat img = cv::imread(f, cv::IMREAD_COLOR);

    cv::Mat imgUndistorted;
    // 5. Remap the image using the precomputed interpolation maps.

    // Display
    cv::imshow("undistorted image", imgUndistorted);
    cv::waitKey(0);
  }

  return 0;
}
```

Detected chessboard corners



Rectified image



Expected calibration output

```
Reprojection error = 0.327524

K =

[1178.7924, 0, 739.5282;

 0, 1178.7924, 583.0882;

 0, 0, 1]

k=

[-0.242507, 0.118744, 0.000705469, -0.000532835, -0.0381491]
```