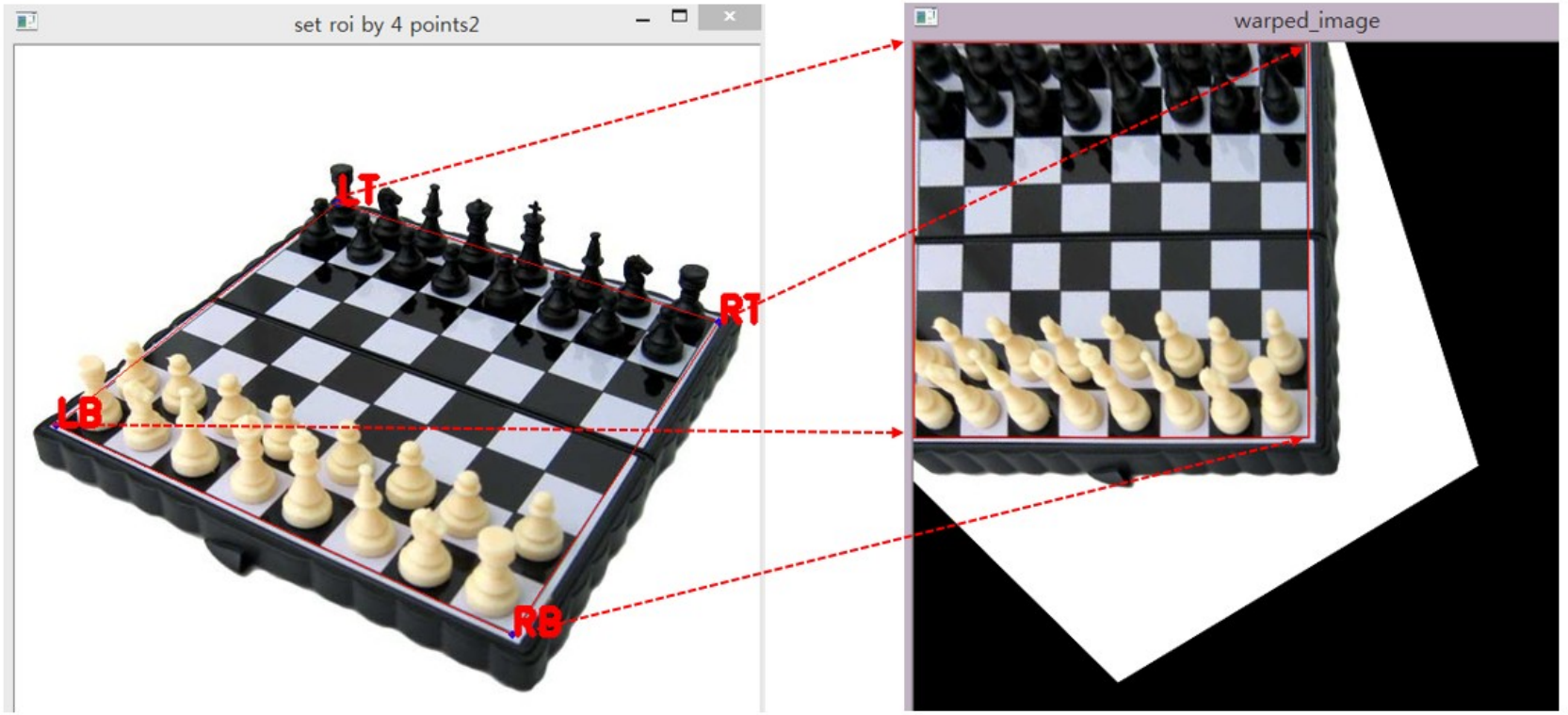
Exercise 4: Binary vision

In this exercise, you will get a theoretical understanding of the derivation of the perspective transform, apply a perspective transform to an image, project a point using the projection matrix and optionally calibrate a camera.

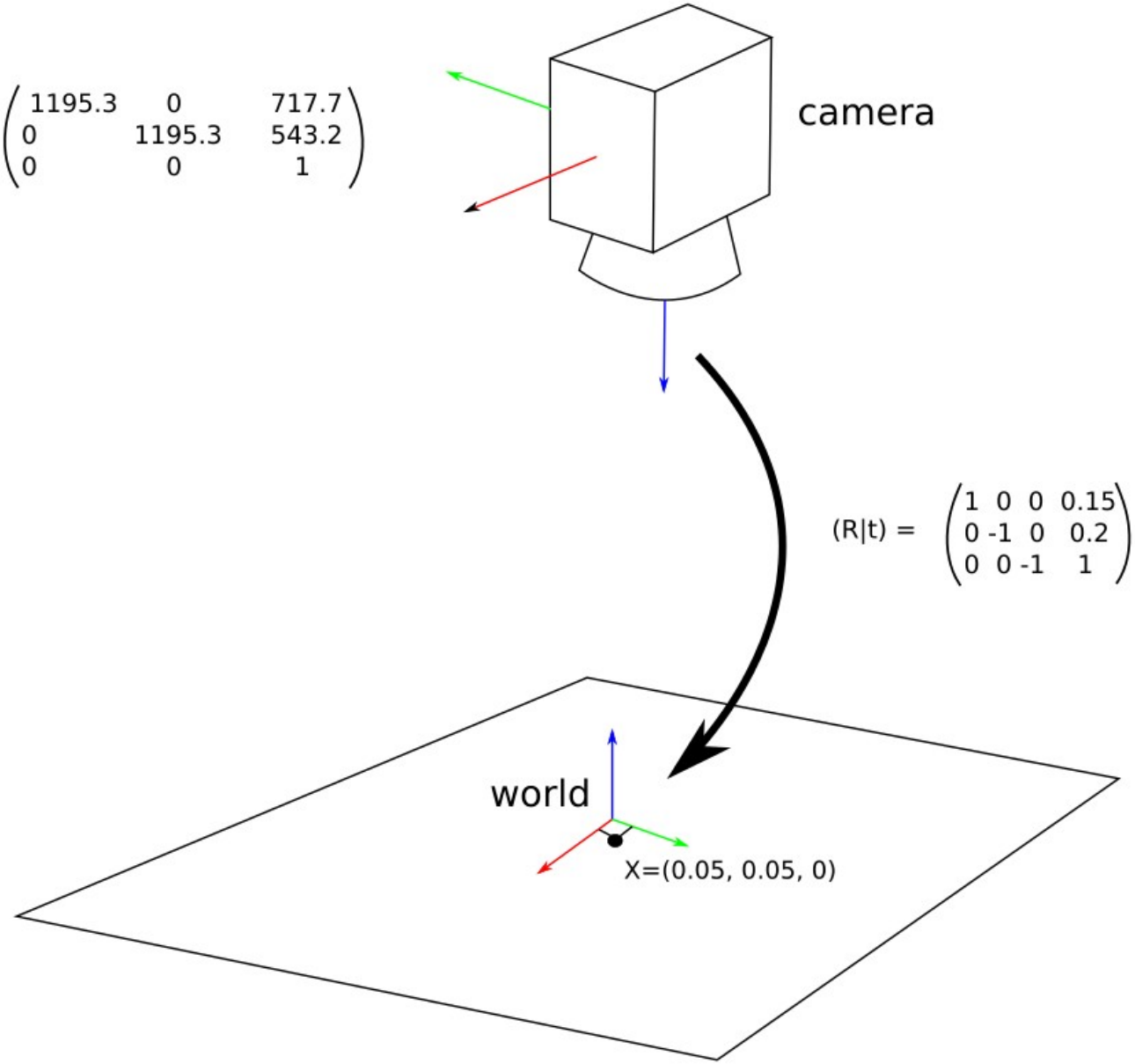
- Section 5.3 in Howe describes how to compute the perspective transform from four sets of image points by deriving a matrix equation (eq. 5.10) and solving it. Derive eq. 5.10 by hand following the procedure in the book. Optional: Implement a method which computes the perspective transform for an arbitrary number of points and solve the equation 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. Compute and apply the perspective transform which performs this operation
- Compute the projection of the point X onto the image plane (see image on the following page). Use the following:
  - X = (0.05m, 0.05m, 0.0m). The point's coordinate is provided in the world frame
  - The extrinsic camera parameters are R = (1 0 0; 0 -1 0; 0 0 -1) t=(0.15m, 0.2m, 1m)
  - The intrinsic camera parameters are fx=fy=1195.3 cx=717.7 cy=543.2
- Optional: Load the calibration/image*.png images and calibrate the camera. 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 almost the same code as provided for the semester project

Homography exercise



Setup for projection exercise

$$K = \begin{pmatrix} 1195.3 & 0 & 717.7 \\ 0 & 1195.3 & 543.2 \\ 0 & 0 & 1 \end{pmatrix}$$



Part 1 of calibration code

Load images and locate chessboard corners with sub-pixel precision

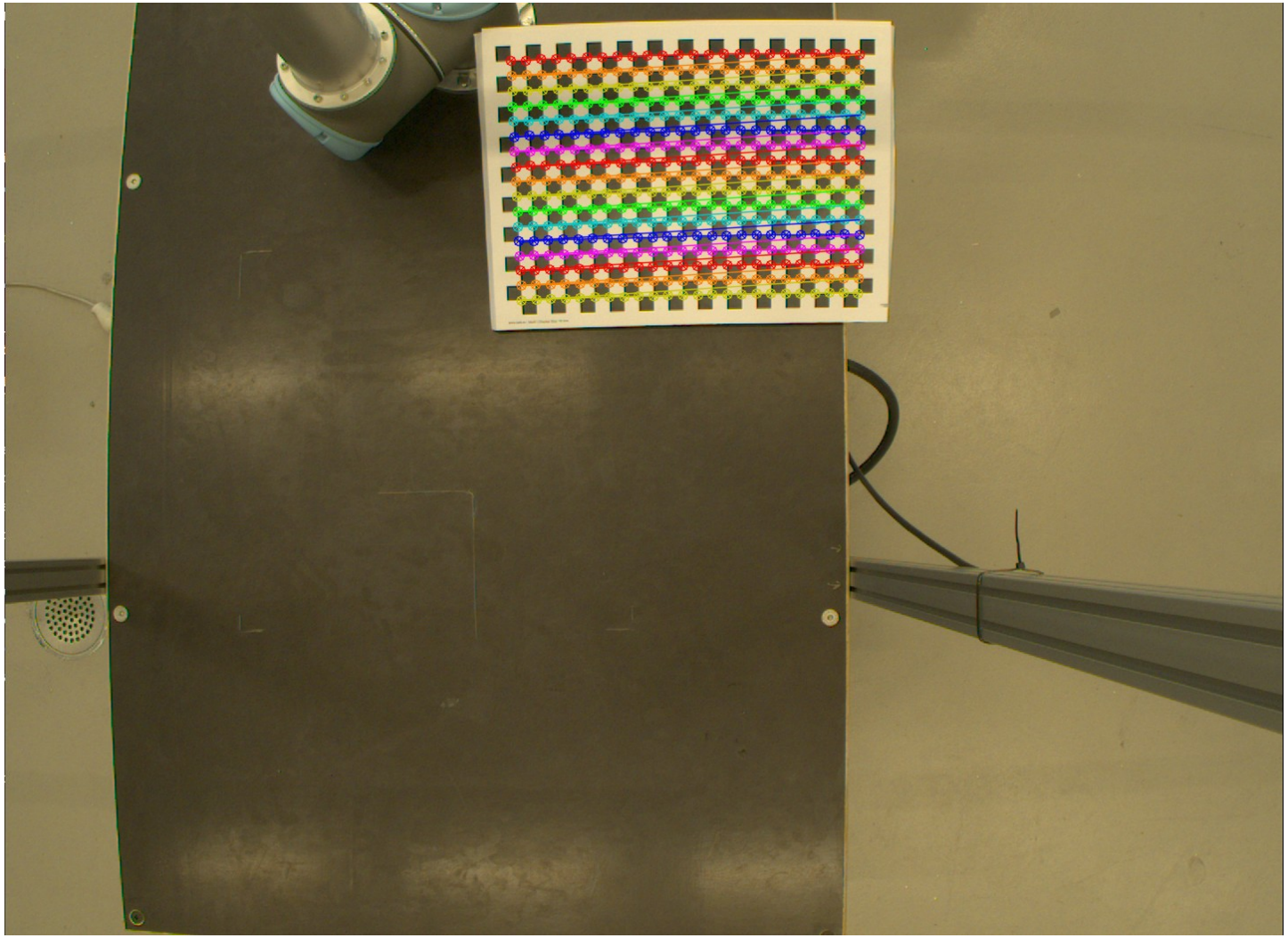
```cpp
#include <iostream>
#include <opencv2/calib3d.hpp>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>

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

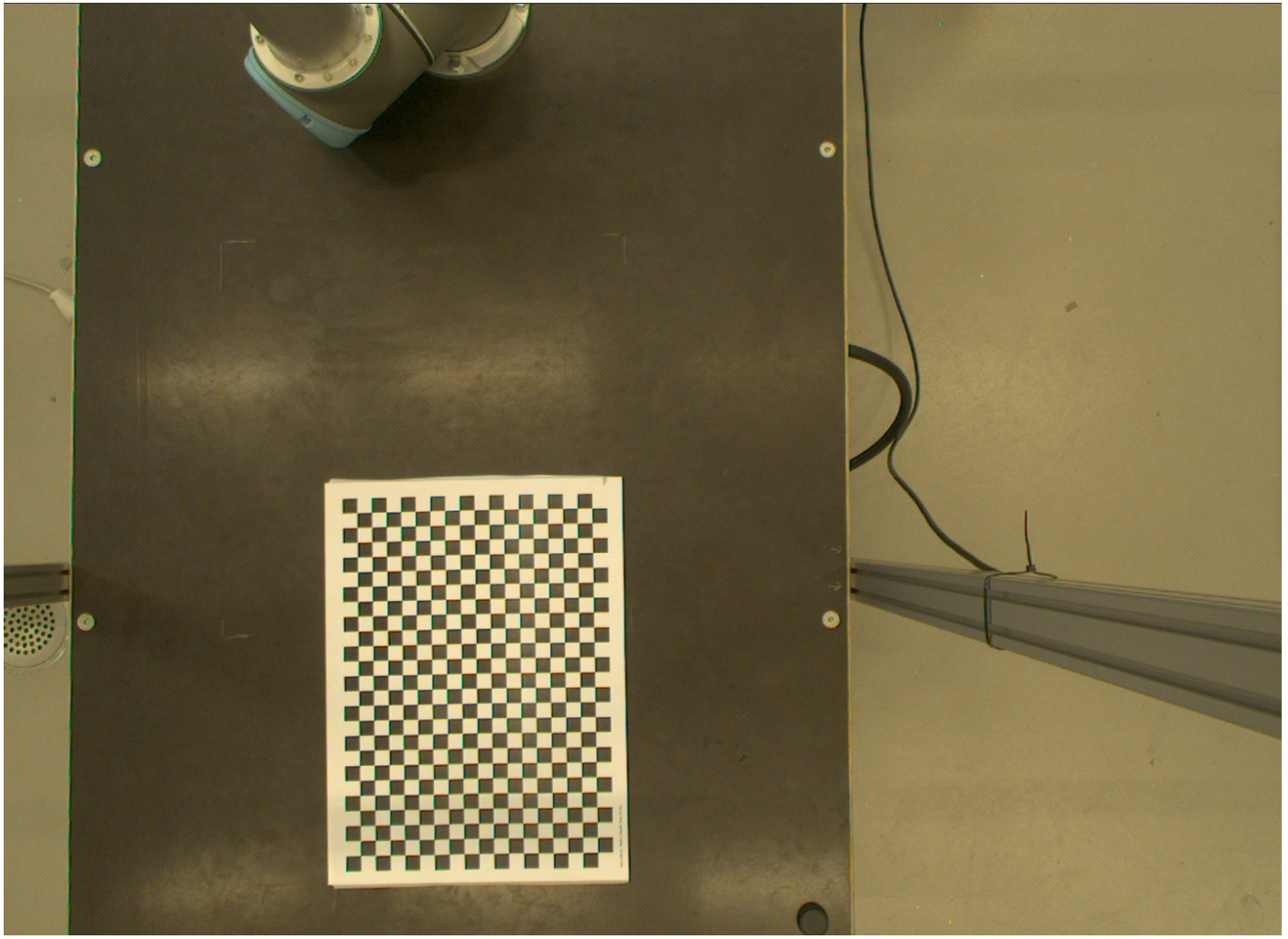
Define chessboard coordinates real world coordinates, call the calibrateCamera method, and undistort the images using the computed camera matrix and distortion coefficients

```
  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 = 0.364212
K =
[1198.9491, 0, 720;
 0, 1198.9491, 540;
 0, 0, 1]
k=
[-0.232877, 0.0852352, 0, 0, 0]
```