UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Laboratory 2 – Camera Calibration

Alberto Pretto

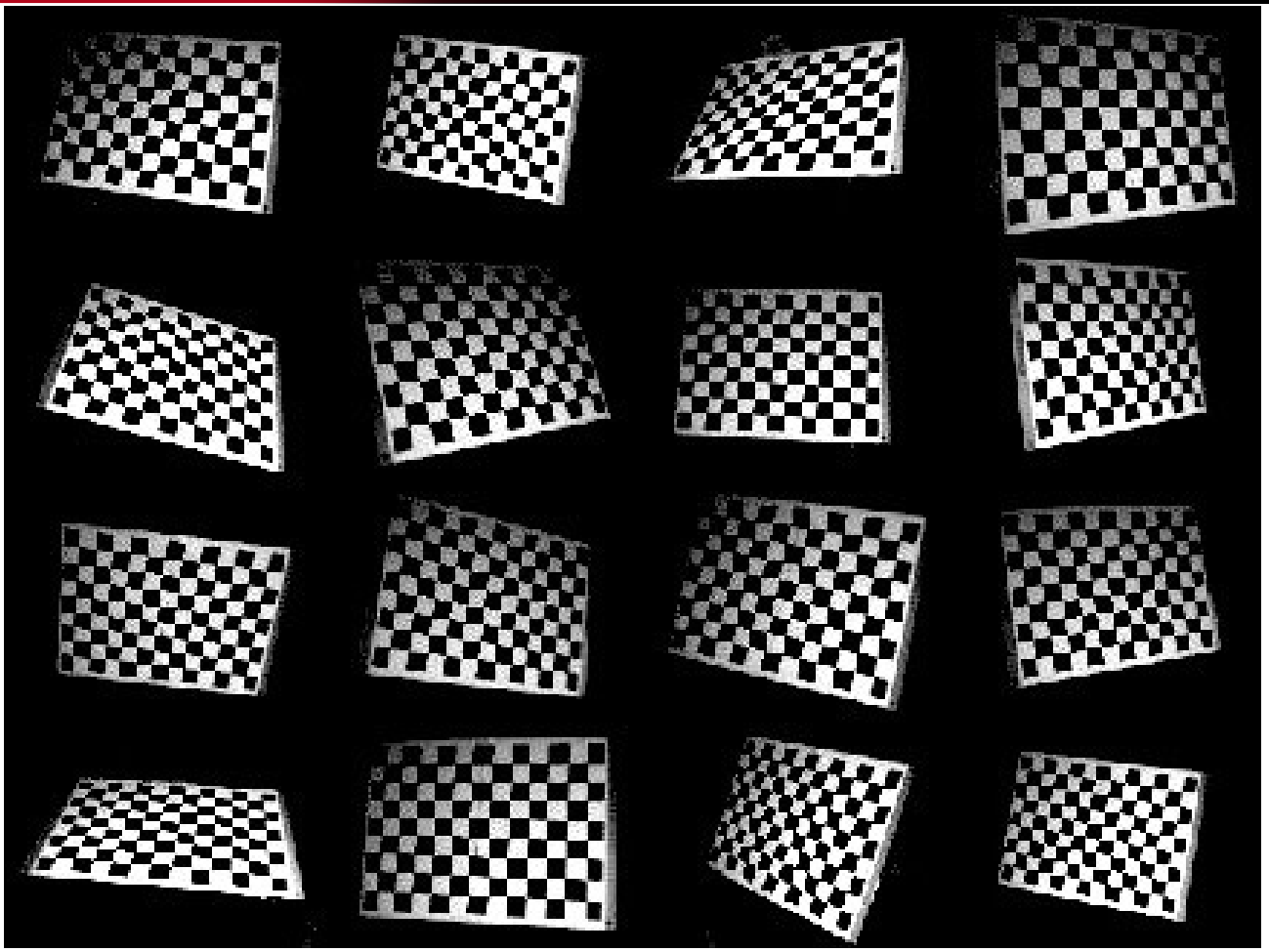
DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

INTELLIGENT AUTONOMOUS
SYSTEMS LAB

Lab Recap

- Acquire and load the dataset
- Extract the corners from each image and prepare the corresponding 3D – 2D vectors
- Compute the extrinsic and intrisic parameters
- Compute the mean reprojection error and chose the best and worst image
- Undistorts an image and show it

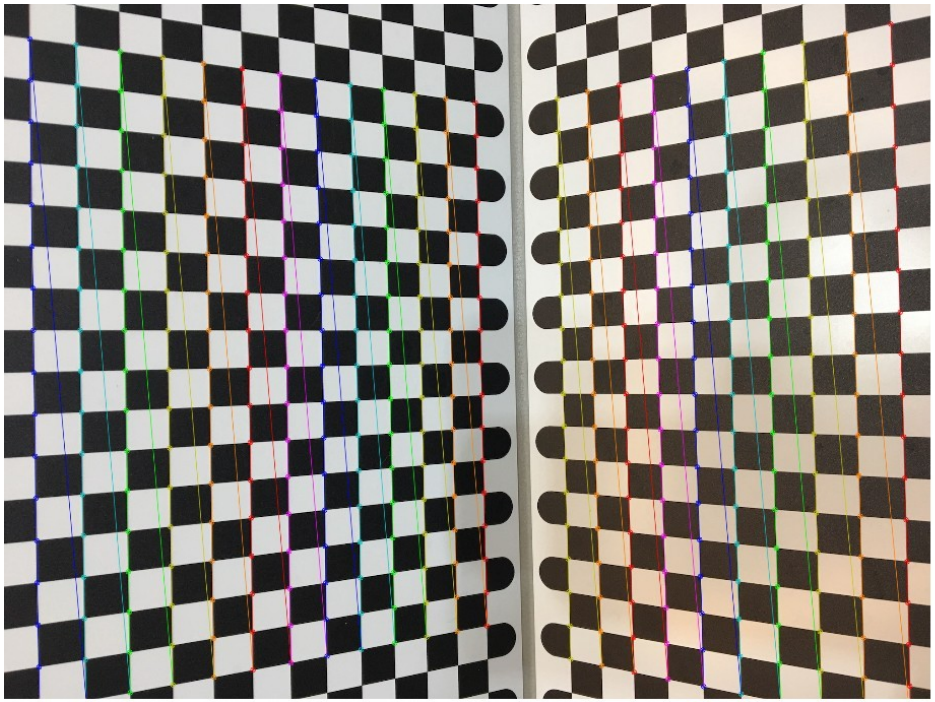
Acquire a Dataset

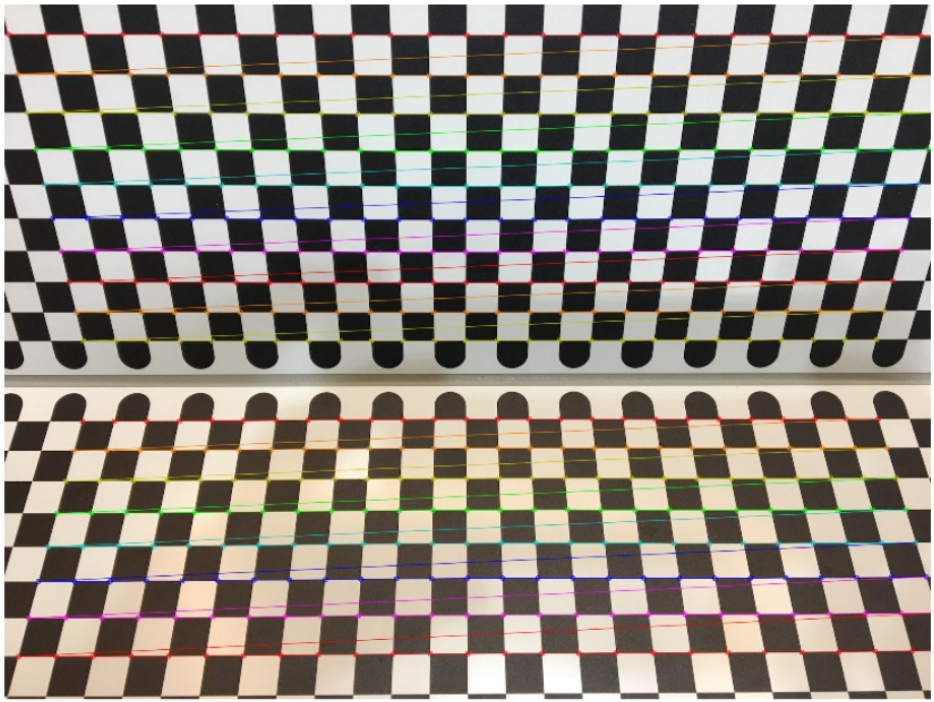


- ... or just load the provided dataset!

Acquire a Dataset

- Recent results report that only 4 "good" images (or just 2 with two patterns!) are sufficient to obtain good and repeatable results.





Load the dataset

- Suppose your images are stored in a directory “images/” with names “img0.jpg”, “img1.jpg”, ...
- Load their filenames into a std::vector< cv::String >, e.g., by using the OpenCV function:

```
cv::utils::fs::glob( const cv::String &directory,
                     const cv::String &pattern,
                     std::vector< cv::String > &result, ...);
```

- E.g.

```
#include <opencv2/opencv.hpp>
#include <opencv2/core/utils/filesystem.hpp>
...
cv::utils::fs::glob( cv::String(“images”),
                     cv::String(“*.png”),
                     filenames );
```

Load the dataset

- Now you can easily read the images:

```
for (const auto& fn: filenames)
{
  cv::Mat img = cv::imread(fn, cv::IMREAD_GRAYSCALE);
  ...
}
```

Points Vectors

- For each view, you should provide a vector of 3D pattern points (e.g., the 3D coordinates of the corners in the checkerboard reference frame) and a corresponding vector of the 2D corners (in pixels) extracted from such image.
- Since we have a set of views (images), we prepare two vectors of vectors of points

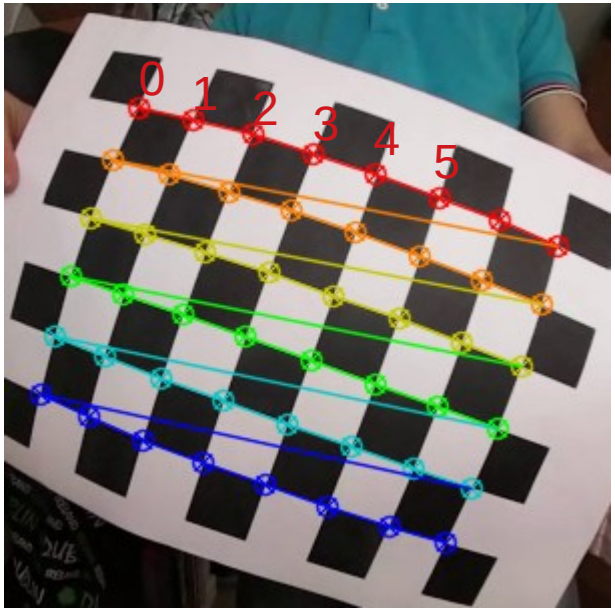
```
std::vector< std::vector< cv::Point3f> > points3d;
std::vector< std::vector< cv::Point2f> > points2d;
```

- Or equivalently:

```
std::vector< std::vector< cv::Vec3f> > points3d;
std::vector< std::vector< cv::Vec2f> > points2d;
```

3D Points

- For each view, list the same 3D points
- Take into account the checkerboard size and the square size (in meters)
- For the provided data:
  - 6 x 5 corners
  - 0.11 m square size
- Pay attention to the order of the points!



2D Points

- For each calibration image, find the checkerboard corners using the OpenCV function `cv::findChessboardCorners()`
  https://docs.opencv.org/4.5.1/d9/d0c/group__calib3d.html#ga93efa9b0aa890de240ca32b11253dd4a
  storing the found corners into a `std::vector<cv::Point2f>`.
- If successful, push back (`std::vector::push_back()` method) such vector into the `points2d` vector of vectors.
- Consequently push back a vector of 3D corners into the `points3d` vector of vectors.

2D Points

- If you want to visualize the extracted corners, use:
  `cv::drawChessboardCorners()`
  https://docs.opencv.org/4.5.1/d9/d0c/group__calib3d.html#ga6a10b0bb120c4907e5eabbcd22319022
  and visualize the image by using:
  `cv::imshow()`

Calibrate the camera

- Compute the camera calibration parameters along with the extrinsic parameters from `points3d` and `points2d` by using the `cv:: calibrateCamera()`

  https://docs.opencv.org/4.5.0/d9/d0c/group__calib3d.html#ga3207604e4b1a1758aa66acb6ed5aa65d

  function
- Print (with names) the parameters with the function `std::cout`.

Reprojection errors

- Compute the mean reprojection error by reprojecting the 3D corners into the image, and comparing the obtained 2D points with the positions of the extracted 2D corners.
- Choose the best and worst image based on the mean reprojection error
- (Is there is something better than the mean reprojection error)?

Reprojection errors

- To reproject the 3D corners, use che `cv::projectPoints()` function,

  https://docs.opencv.org/4.5.0/d9/d0c/group__calib3d.html#ga1019495a2c8d1743ed5cc23fa0daff8c

  with the camera positions computed by `cv::calibrateCamera()` (i.e.,. the extrinsic parameters).
- To compute the mean reprojection, compute the mean Euclidean distance between reprojected points and extracted corners.

Show a Undistorded Image

- Undistort a test image by using the `cv:: initUndistortRectifyMap()`

  https://docs.opencv.org/4.5.1/d9/d0c/group__calib3d.html#ga7dfb72c9cf9780a347fbe3d1c47e5d5a

  and `cv::remap()` functions.

  https://docs.opencv.org/4.5.1/da/d54/group__imgproc__transform.html#gab75ef31ce5cdfb5c44b6da5f3b908ea4

- Visualize the original and undistorted images in two different windows by using the `cv::imshow()` function

An Useful Improvement

- After detectin the corners in each image, you may use `cv::cornerSubPix()`

  https://docs.opencv.org/master/dd/d1a/group__imgproc__feature.html#ga354e0d7c86d0d9da75de9b9701a9a87e

  to refine their positions with sub pixel precision

Some programming hints

- Take small steps, testing each time your new code, e.g.:
  - Load the images name and print out them
  - Open each image and show it
  - For each image, extract corners and visualize them
  - …
- Divide your code into functions / classes, e.g.:
  - `visualizeCorners()`
  - `undistorImage()`
  - `...`

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Laboratory 2 – Camera Calibration

Alberto Pretto

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

INTELLIGENT AUTONOMOUS
SYSTEMS LAB