Geometric and 3D Computer Vision
Final Project

Prof. Filippo Bergamasco

a.a. 2023/2024

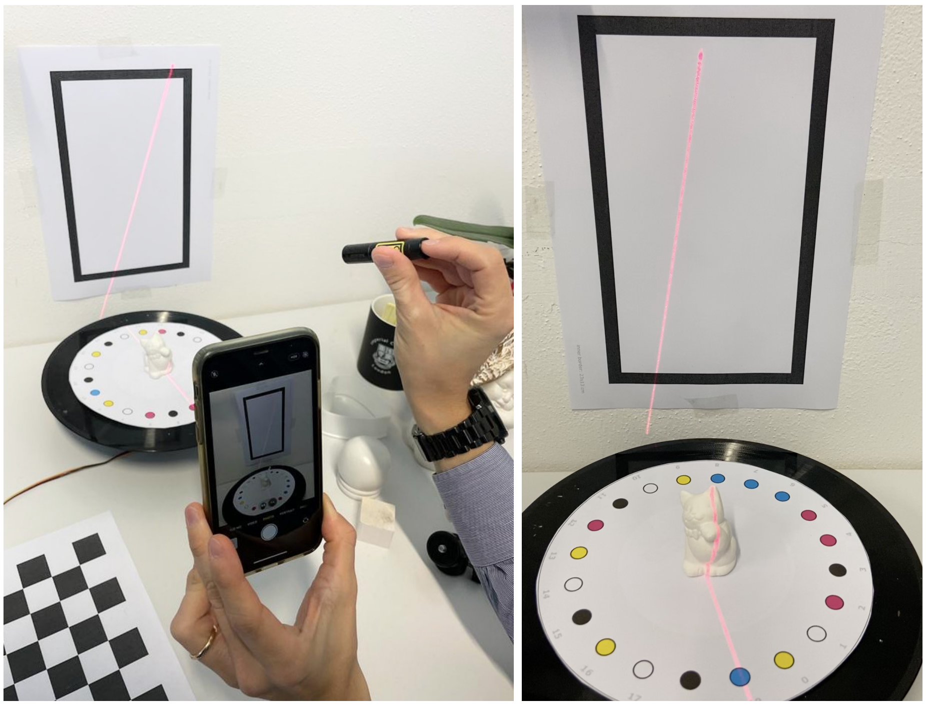

Figure 1: Left: Our simple scanner setup composed of a camera, a rotating turntable with a circular (colored dot) marker on top of it, and a rectangular marker on the background. The object is placed on the turntable and a laser line is projected to the scene. Right: An example frame captured by the camera.

1 Introduction

This year's final project[1] aims at the development of a simple 3D laser scanner using off-the-shelf components. The scanner is composed of a motorized turntable with a custom-designed *fiducial marker* placed on top showing a set of colored dots located at specific positions (see Sec.1.1). Such marker rotates together with the turntable thus offering a reference system attached (comoving) to the object to be scanned. An additional planar rectangular marker is placed vertically behind the turntable and remains fixed with respect to it.

The acquisition is performed by projecting a laser line on the target object ensuring that the same line is also visible inside the two markers. A camera acquires the whole scene several times during the rotation of the turntable (Fig.1).

[1]Last revision: December 12, 2023

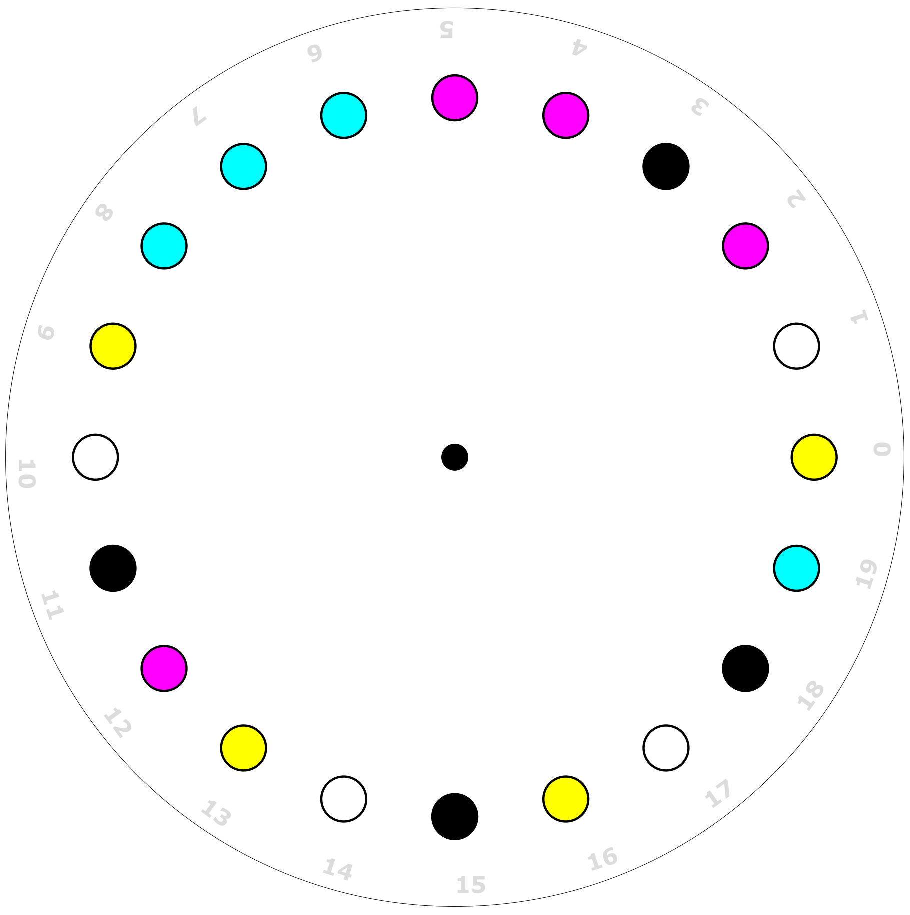

Figure 2: Circular marker attached to the turntable. It is composed of 20 dots colored in yellow, white, magenta, cyan, and black according to a given pattern. See the text for details.

Your task is to write an algorithm that, by analyzing the shape of the line in each frame, outputs a 3D point cloud consistent with the 3D geometry of the object.

1.1 Circular marker design

Figure 2 shows the marker attached to the turntable. It is composed of 20 dots distributed counterclockwise on a circle. The distance between the center of each colored dot and the center of the circle is 7.5 cm. The angle between two consecutive dots and the center is $360°/20 = 18°$.

Let's define a marker (world) reference system with the origin placed at the center of the circle, the x-axis pointing toward dot 0, the y-axis toward dot 5, and the z-axis exiting up from the marker plane. In such settings, dot 0 have coordinates $\begin{pmatrix} 7.5 & 0 & 0 \end{pmatrix}^T$, dot 5 have coordinates $\begin{pmatrix} 0.0 & 7.5 & 0 \end{pmatrix}^T$, dot 1 have coordinates $\begin{pmatrix} 7.1329 & 2.317 & 0 \end{pmatrix}$, and so on. Since the object is placed firmly on top of the marker, and rotates with it, it is possible to reconstruct the object's pose with respect to the camera by observing where each colored dot is projected onto the image plane. In doing that, consider that the sequence of dot colors is as follows: (Y,W,M,B,M,M,C,C,C,Y,W,B,M,Y,W,B,Y,W,B,C) where C, Y, M, W, and B represent color Cyan, Yellow, Magenta, White and Black respectively.

This particular color sequence forms an aperiodic Necklace (ie. a Lyndon word) with an alphabet size of 6 symbols (colors) and a string length of 20. It guarantees to be not ambiguous under rotation even when observing a substring of just 4 symbols over the sequence of 20. For more info see (`http://combos.org/necklace`)

Note: gray numbers near each colored dot are meant for debugging purposes and cannot be exploited to recognize the marker.

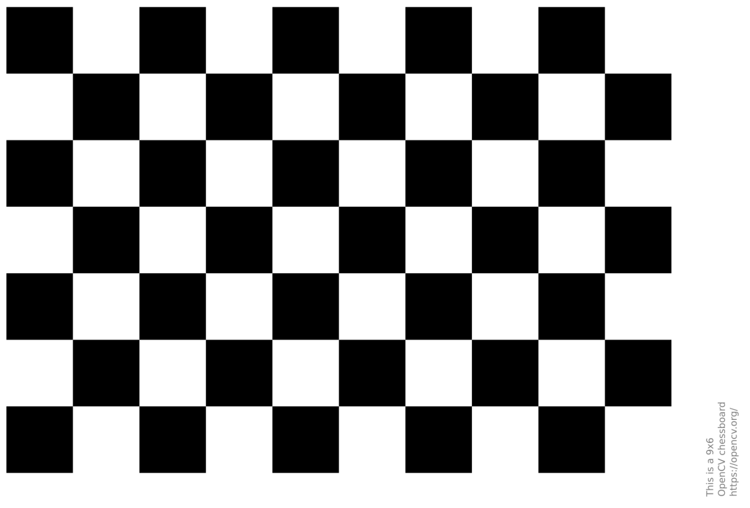

Figure 3: The standard OpenCV's calibration target.

1.2 Rectangular marker design

The rectangular marker placed behind the rotating turntable is very simple. It is composed of a thick black rectangle. The inner white rectangle measures 23×13 cm.

1.3 Camera calibration

The camera must be calibrated before usage to perform an accurate 3D reconstruction. Together with the 3D scanning software, you are also asked to create a program to calibrate a camera using multiple exposures of the standard OpenCV's chessboard calibration target (Fig. 3. I suggest you to refer the OpenCV camera calibration tutorial `https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html` for general information about the process.

1.4 Requirements

The project must contain two different programs:

1. A *camera calibrator* that loads the frames of the provided calibration video and computes the intrinsic parameters of the camera (the intrinsic matrix K and the lens distortion vector assuming a 5-parameters model) without any user intervention. This program should output the Root Mean Square (RMS) re-projection error together with the calibration output.
2. A *3D scanner* that reads one of the 3 videos provided (cat, cube, and ball) and processes it on a frame-by-frame basis. After processing all the frames (or during the processing), it computes a 3D point cloud with all the reconstructed 3D points of the scene that have been illuminated by the laser line. This program can optionally present an interactive interface to the user with the only requirement that the user cannot "help" the scanner to identify the laser line or the markers.

The two programs must be written in Python3 and should use NumPy, OpenCV, and all the additional libraries you consider useful for the project. Jupiter notebooks can also be used if necessary.

Point clouds must be saved as $N \times 3$ matrices stored in an ASCII encoded text file[2]. Such files can be visualized with the Open-source tool MeshLab `http://www.meshlab.net/`. Optionally, the Open3D library `http://www.open3d.org/` can be used to visualize the reconstructed data in real-time during the processing.

[2] Also known as XYZ format

1.5 Additional notes

There are no particular constraints on the number of function/classes/py files produced. You are highly encouraged to give any visual feedback to better understand each algorithm involved. For example, the program could show the boundary of each planar rectangle on the scene, the detected line points, the reconstructed 3D point cloud, etc. Any debug information useful to explain the operations involved is welcome and will contribute to the final evaluation of the project during the oral exam.

A reasonable real-time processing speed will be evaluated as a positive feature of your work (although not mandatory to pass the exam). You are invited to report the processing time of each frame and, if appropriate, of each different processing step.

Comment your code whenever possible. Since no additional report is required, comments are a good way to clarify what your code is supposed to do.

2 Exam Information

The final course exam consists of an oral discussion about the project. When ready to take your final exam, package the source code in a zip file named `<name>_<surname>_exam.zip` and submit it via Moodle. Then, notify me via mail (`filippo.bergamasco@unive.it`) so that we can arrange a date (usually within a couple of weeks from the submission). Please, do not submit any data related to the project (video files, images, calibration data, etc.).

During the exam, I will ask you to explain all the interesting parts of the source code, focusing on the implementation details and the algorithms involved. You are supposed to discuss the strengths and limitations of any choice you made during the development. During the discussion, you will also have to show a demo of your project preferably on your laptop.

I'll consider the exam passed with full marks if you demonstrate proficiency in the computer vision techniques we have seen during the course in relation to the project you have developed. This means that the actual final performance of the produced code is not critical if the behavior is properly justified and discussed.

For any questions regarding the project feel free to mail me at `filippo.bergamasco@unive.it`. **I'll be happy to give you useful advice.**

3 Data

Video files containing the acquisition of 3 different objects and the chessboard used for calibration can be downloaded here:

`https://www.dsi.unive.it/~bergamasco/teachingfiles/G3DCV_turntable_scanner_data.zip`

Appendix: useful OpenCV functions

What follows is a short list of OpenCV functions that you may find useful for the development of the project. You are allowed to use any other function other than the ones listed here.

cv.undistort Transforms an image to compensate for lens distortion. I suggest applying this function as soon as a video frame is loaded, and before any other processing.

cv.threshold Applies a fixed-level threshold to each array element.

cv.findContours Finds contours in a binary image.

cv.fitEllipse Fits an ellipse around a set of 2D points.

cv.findHomograpy Finds a perspective transformation between two planes.

cv.warpPerspective Applies a perspective transformation (ie. Homography) to an image.

cv.solvePnP Finds an object pose from 3D-2D point correspondences. Even if the object is planar, and therefore you could find the $\mathbf{R}, \mathbf{T}$ by decomposing and Homography, I suggest to use `cv.solvePnP` with the flag `SOLVEPNP_IPPE`. The result will be much more robust to noise.

cv.Rodrigues Converts a rotation matrix to a rotation vector or vice versa. Use it in conjunction with `cv.solvePnP`