- CHOICE: implement a way to enhance the input to reduce the number of input images that are not correctly processed by findChessboardCorners, for example by enhancing edges or getting rid of light reflections: 10
- CHOICE: produce a 3D plot with the locations of the camera relative to the chessboard when each of the training images was taken: 10
- CHOICE: use your creativity. Check with Ronald Poppe for eligibility and number of points (via Teams or email).

Submission: Submit a zip through Blackboard with

- Code (+ project files) but no libraries or images (max. submission size is 20MB)
- Report (1 page)

Deadline: Sunday, February 18, 2024, at 23.00. For questions about the assignment, use the INFOMCV 2024 Teams (Assignment channel). If you found that your assignment partner did not work properly, notify Ronald Poppe as soon as possible.

Assignment 2: Voxel-based 3D reconstruction

This document contains a detailed description of Assignment 2, the data and the provided Python code used to visualize a voxel reconstruction. Use the code as a basis of your own program. The main goals of this assignment are (1) to calibrate four cameras (intrinsics and extrinsics), (2) to do background subtraction for each view, and (3) to implement the silhouette-based voxel reconstruction algorithm.

Download data + example code

- Download data (51MB)
- Download example code framework: https://github.com/dmetehan/Computer-Vision-3D-Reconstruction.git

Data files

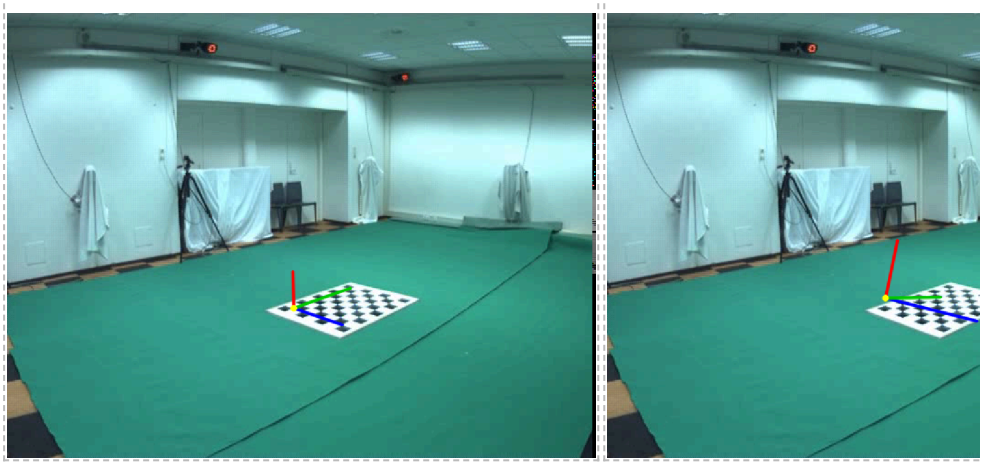
In the provided **data.zip** file you will find 4 directories named: cam1, cam2, cam3, cam4. In it there are 4 videos and 1 XML file. Unpack the file in the root folder of your project (the "data" folder should be in your project root folder).

1. **intrinsics.avi**
   - The video file contains recordings of a chessboard moving around each camera.
2. **checkerboard.avi**
   - This video file will provide your camera extrinsics.
3. **background.avi**
   - Use this video to pick a frame (or do more complex processing, see below) to create a **background** image.
4. **video.avi**
   - This is the video from which the data has to be reconstructed in 3D.

Task 1. Calibration

1. Obtain the intrinsic camera parameters with your code from Assignment 1, for every camera. Use the **intrinsics.avi** files of each view. You can (hand or automatically) pick a number of frames from each video. The number of rows/columns and the size of a tile can be found in **checkerboard.xml**.
2. We will now use the **checkerboard.avi** files to calculate the camera extrinsics. Check for a function to calculate the extrinsics provided that the intrinsics are already known. As **cv2.FindChessBoardCorners(...)** will not work, you should use the interface you built in Assignment 1 for getting chessboard corners. Do this precisely. Ways to automate or speed-up this process are eligible for **choice points**. To select the chessboard corners, start at the **same** chessboard corner

(e.g. left-upper corner; look at the color of the corner cell) for each camera. Keep in mind that the chessboard is not square. Do the extrinsics calibration using the selected checkerboard corners for each camera. This stackoverflow answer can give you some hints about how to do it: https://stackoverflow.com/a/55284535/3602047 After completing the intrinsics and extrinsics calibration process, visualize the 3d world coordinates on the starting corner of the checkerboard for each camera:



Good Bad

3. After finishing with all cameras, write one final **config** file containing all camera properties (including extrinsics and intrinsics). The designated directory is **data/camX/** (with X the camera number). This allows you to use a previously obtained calibration. Use the **intrinsics.xml** for an example how to store the intrinsics parameters (including distortion coefficients).

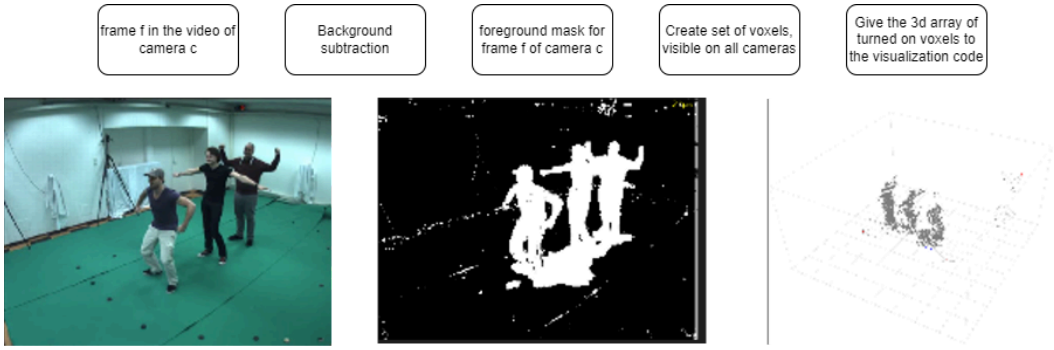
Task 2. Background subtraction

1. Use the **background.avi** files to create a background model. A single frame may not give the best results. Averaging frames may help to improve the background subtraction process somewhat. How to create the best possible background image is up to you. You can also replace this with a more sophisticated model, e.g. by modeling each pixel as a Gaussian (mixture) distribution. That will give more points.
2. Once a background model is made, you will be doing background subtraction on each frame of **video.avi**. This means you have to find all foreground pixels. Convert the foreground frame image of a given camera from BGR to HSV color space, and calculate the difference with your background model. Threshold the differences for the Hue, Saturation and Value channels. See how you combine the different channels to determine if a pixel is foreground (value 255) or background (value 0).
3. Setting the thresholds automatically is a **choice task**. Two suggestions, but there are many more viable solutions:
    1. Make a manual segmentation of a frame into foreground and background (e.g., in Paint). Then implement a function that finds the optimal thresholds by comparing the algorithm's output to the manual segmentation. The XOR function might be of use here. You can look over different combinations of thresholds, and simply select the combination that gave the best results.
    2. Assume that a good segmentation has little noise (few isolated white pixels, few isolated black pixels). Implement a function that tries out values and optimizes the amount of noise. Be creative in how you approach this. Perhaps the functions *erode* and *dilate* can help.
4. Also think about post-processing, e.g. using *erosion* and *dilation*. Other techniques such as *Blob detection* or *Graph cuts* (seam finding) could work. You don't need to implement these algorithms yourself, use the OpenCV API to find the relevant functions/methods. Motivate in your report why and how you use these.

Task 3: Voxel reconstruction

1. You will need to create the lookup-table for the valid voxel back-projections onto each of the camera's FoVs. So, given a position in R^3, we need a projection of it in R^2. Luckily this can be done by the OpenCV function **projectPoints**. The projectPoints-method requires (aside from input and output points) the calibration matrices, which have to be saved in **data/camX/config.xml**:
    1. Camera Matrix (3x3): Camera Intrinsics matrix
    2. Camera Distortion Coefficients Matrix/Vector (5x1): The cameras have fish-eye lenses, this needs correction.
    3. Rotation Matrix/Vector (3x1): Camera rotation vector from the origin
    4. Translation Matrix/Vector (3x1): Camera translation vector from the origin.
2. Now develop the voxel reconstruction algorithm such that the voxels that correspond to the person in the scene are "on", based on the four foreground images. The result of your work should be a 3D cubic half-space containing the voxel reconstruction of the provided input video (**video.avi**). Use the provided functions. Noise and voxels corresponding to shadow reduce the quality of your reconstruction. Think about ways to remove such voxels. Similarly, if there are (small) holes in your reconstruction, think of a way to fill those, without making unrealistic assumptions. A **choice task** is to "color" the voxels according to what you see in the video. Another **choice task** is to convert the voxel model into a surface mesh.

The basic drawing pipeline



Codebase: Install the relevant libraries following the installation instructions on github: https://github.com/dmetehan/Computer-Vision-3D-Reconstruction.git. When you run the executable.py you should see an empty scene with floor grid and 4 dummy cameras. You can press 'G' to generate random voxels and use 'W', 'A', 'S', 'D', and your mouse to traverse the 3d space. Overall you don't need to worry about the details of the visualization code. All the functions you need to edit are in assignment.py. You're free to create as many files/classes as necessary. The position and orientation of the cameras in world coordinates should be estimated using the cameras' extrinsics matrices. The 3D voxel space is a half-cube, with side lengths 128 and height 64. y is the up direction. You can set the voxel locations using list of lists. [[x1, y2, z1], [x2, y2, z2], ..., [xn, yn, zn]].

Report: Your report should be around 2 pages (you can use this Word template) and contain:

1. An explanation of your methods.
2. The extrinsic camera parameters (rotation matrix and translation) for each of the four cameras.
3. The thresholds for your background subtraction (or the description of other parameters in your approach), and how it is determined if a pixel is foreground or background. Include a foreground image of each of the cameras for one frame of the horseman video.
4. A brief summary of your choice tasks.
5. A **link to a video** (Youtube, Vimeo, Wetransfer, etc.) clearly showing the 3D reconstruction of the input videos. Make sure the link is accessible

reconstruction of the input videos. Make sure the link is accessible.

Submission: Through Blackboard, hand in (1) your **report** (2 pages, max 4). Include (2) your **source code** (do not include the resources, .jpg, .png, etc.) and (3) the calibration files of each camera (**no videos!**).

Grading: The maximum number of points for this assignment is 100. The assignment counts for 10% of the total grade. You can get up to 70 points for these tasks:

- Quality of calibration (and calculated/motivated properly): 20
- Ingenuity of background subtraction method: 15. More points are given for more robust techniques.
- Quality of background subtraction: 10
- Quality of voxel model (and sophistication of tricks): 15
- Report (motivate your choices, **indicate choice tasks**): 10

In addition, you can earn a maximum of 30 choice points by:

- CHOICE 1: Automating the detection of the four corner points of the chessboard: 10 (you need to fully automatically localize the chessboard, and subsequently the four points to mark; assistance in ordering the four points is allowed)
- CHOICE 2: Automating the way the thresholds are determined: 10 (eligible if your thresholds are optimal without manual supervision)
- CHOICE 3: Coloring the voxel model: 15. Importantly, take into account whether a voxel is visible and not occluded by a camera. Max 5 points if there is insufficient depth reasoning.
- CHOICE 4: Implementing the surface mesh: 10. Convert the voxels to a surface mesh with (this is the exception to the rule of no alien code) an implementation of Marching Cubes (or another implementation).
- CHOICE 5: Optimizing the construction of the voxel model: 10 (this task refers to the implementation of the frame-by-frame optimization as discussed in the lecture)