INFOMCV Assignment 2
Martino Fabiani, Taher Jarjanazi (Group number: 20)

Summary:
The background subtraction process was carried out as follows. Firstly, by implementing a "frame extraction" function, the ability to extract frames from each video was made available. Once these frames were obtained, the first step was to create a background model. Iterating through each frame and using the OpenCV function cv.accumulateWeighted(), a background model initially set to None and later equalized to the first frame read, start to be updated through averaging between the various frames read, adjusted in proportion to an alpha parameter determining the adaptation speed to subsequent frames. Once this was achieved, the development of the foreground mask followed. To obtain this, using the video frame division showing the subject in the room, and in our case using HSV reading, the average background was subtracted from each of these frames. Clearly, this subtraction was not entirely clean but containing noise and imperfection. To obtain the final foreground mask, dynamic thresholding and histogram simplification were performed, followed by contour detection functions to eliminate noise and imperfections. A more detailed explanation of these last two processes will be provided in the following paragraphs.
Continuing with voxel reconstruction, the first objective was the development of a lookup table. In our case, the process involved initially creating a three-dimensional space iterating over the height, width, and depth parameters passed to the function as input. Once obtained, for each of the 3D points, the iteration was done regarding the four cameras. For each of these, intrinsic parameters and the previously obtained foreground mask were loaded. Through calibration parameters and the cv.projectPoints function from openCv, points projected from the 3D space into a 2D array were obtained, and at this point, a check was made to determine if this point belongs to the dimensions of our foreground mask and, if so, whether the pixel value is white (255) or black (0). To store this, a boolean vector was used that, for each camera, keeps track and assigns a True value if the voxel belongs to the main figure of our foreground, and at the end of the camera iterations if all 4 cameras confirm that the 3d coordinates belongs to the foreground, it appends them to an empty list. This signifies that voxel will be initialized as “on". Once this process is completed, iteration continues for all the three-dimensional points in our space.
Despite 3D reconstruction not being exactly up to par with the expected outcome, many checks were made on the parameters. After numerous tests we consider our lookup table structure stable and functional, as it is generally possible to recognize the shape of the represented subject in multiple projections, even if it is not excellent. Therefore, we argue that the remaining incomplete part to work on concerns the finetuning of parameters and dimensions with which the coordinates are scaled and with which our figure is then reconstructed at higher accuracy. The positioning of the cameras has proved to be quite accurate. Finally, regarding their rotation, it seems to be correct and consistent with their positioning.

Extrinsic parameters

Cam1:

$$rotation\_vector = \begin{bmatrix} \mathbf{-1.3342} \\ \mathbf{0.5520} \\ \mathbf{0.6331} \end{bmatrix} \qquad translation\_vector = \begin{bmatrix} \mathbf{239.8538} \\ \mathbf{731.1611} \\ \mathbf{4745.8329} \end{bmatrix}$$

Cam2:

$$rotation\_vector = \begin{bmatrix} -1.5904 \\ 0.0272 \\ 0.0333 \end{bmatrix} \qquad translation\_vector = \begin{bmatrix} -247.2400 \\ 1475.8601 \\ 3672.1339 \end{bmatrix}$$

Cam3:

$$rotation\_vector = \begin{bmatrix} -1.0684 \\ -1.3408 \\ -1.4215 \end{bmatrix} \qquad translation\_vector = \begin{bmatrix} -725.0120 \\ 1266.0944 \\ 2502.5538 \end{bmatrix}$$

Cam4:

$$rotation\_vector = \begin{bmatrix} -1.3479 \\ -0.6470 \\ -0.7093 \end{bmatrix} \qquad translation\_vector = \begin{bmatrix} 983.8127 \\ 967.5093 \\ 4273.7804 \end{bmatrix}$$

Background subtraction method:

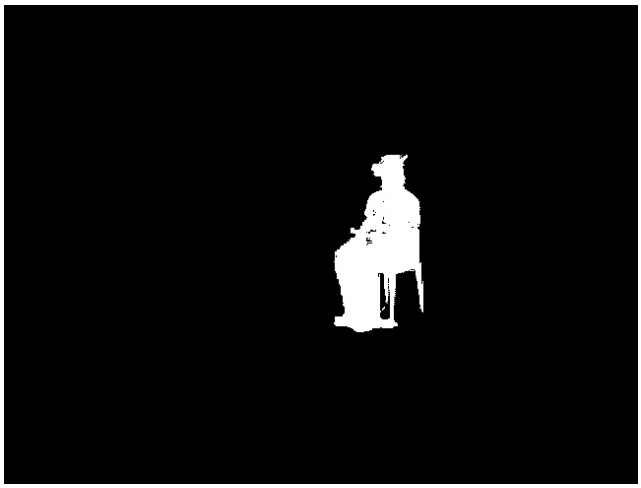
TRESHOLDS DYNAMIC SET:

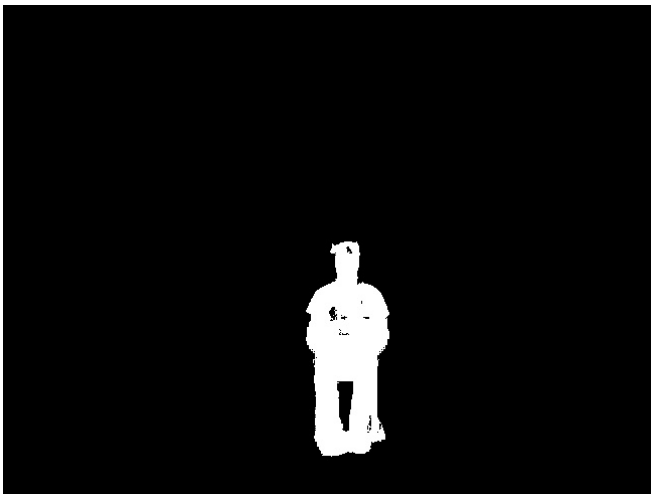
In the process of selecting thresholds for image segmentation, the histogram method paired with the Otsu method was used. Initially, the histogram of the image's HSV values was computed, providing an overview of the distribution of colors present. Subsequently, the Otsu method, a technique within the OpenCV library, was applied to automatically find optimal binarization thresholds for each HSV channel. Specifically, this method seeks the threshold that maximizes the separability between pixel classes in the image, enabling the separation of objects from the background. Finally, by combining the binary masks obtained from each channel, a foreground mask was generated, highlighting which regions belong to the background, setting the intensity value to 0 (black), and which regions belong to the main subject, setting the intensity value to 255 (white).

CONTOUR DETECTION AND NOISE FILTERING:

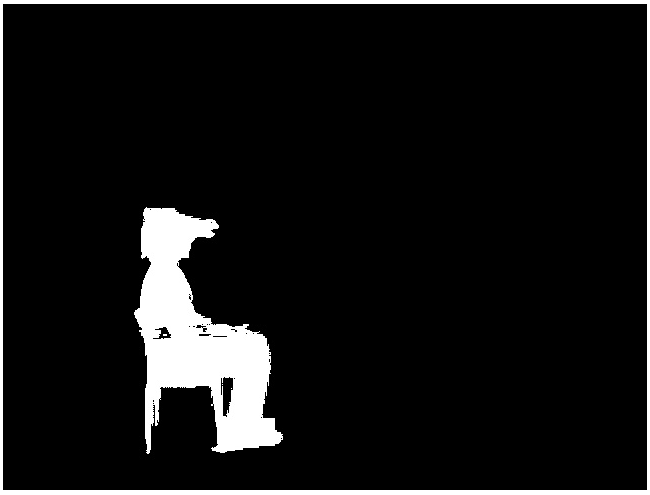
Bob detection is a method that utilizes the cv.findContours function to automatically identify contours in an image. Grazie a questa funzione, impostando quindi un valore di area minima che si vuole evidenziare è possible isolare solo I contorni necessary. Assuming that in our foreground mask there are only the main subject with some noise, the process involves identifying the maximum contour among those detected and outlining a contour around the figure noise (in case of more than one subject it is not right to take only the maximum contour but it is needed to use the common way and find the area threshold in order to keep all the contours needed). However, instead of directly using it as a contour, a contour mask has been developed. This mask is an image of the same shape as our foreground mask but completely black, where the contour we want to retain (in our case, the main subject) is drawn and then filled with white pixels. Finally, using cv.bitwise_AND, an OpenCV function that takes two images and applies the AND operator to all pixels in the images, only the pixels of our foreground mask that are within the identified contour are retained in the final image.

CAM1: CAM2:

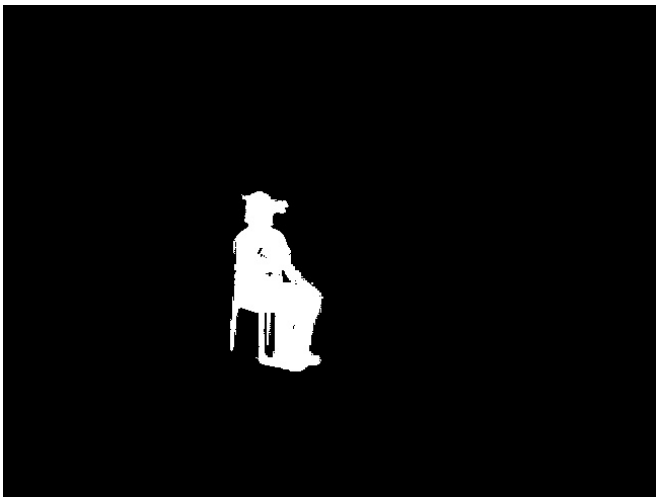




CAM:3



CAM:4



Choice tasks:

- **DYNAMIC TRESHOLDS:**

As already explained before we used an autamatic way to set the thresholds.

- **Automatic corner detection:**

We implement a method to find the four corners of the chessboard automatically using the HoughLinesP() in OpenCV in combination with the Canny() edge detection function and findContours(). Although the function only outputs the outer corners of the chessboard, it can be useful when the findChessboardCorners function of OpecnCV is not able to detect the chessboard corners.

- **IMPROVEMENT IN THE MANUAL CALIBRATION:**

Despite not being a true method for automating the corner recognition process, we have implemented a function called 'getMagnifiedImage' in our code. This function, provided with an input image, coordinates of a point, a magnification factor, and a radius to define an area to magnify, allows for a significant improvement in the accuracy of manually input coordinates during the calibration process. Firstly, the function verifies that the mouse coordinates are within the image boundaries. Subsequently, it defines a region of interest around the specified point, crops this region, and applies a magnification effect to it (adjusted by the input parameter). Ultimately, in the case of manual detection, this enables a clearer visualization of the chessboard corners, thus enhancing precision.

SURFACE MESH:

Inside our code, a function for surface mesh has been implemented. This function takes a 3-dimensional NumPy array initialized with False values as input. During the creation of the lookup table, when "on" is assigned to a voxel, the coordinates are stored by setting the value of the array at that position to True. At the end of the table development, this array is provided as input to the function with a threshold value of 0.5. This means that values above this threshold (like True) are considered ON, while others are considered OFF. Subsequently, using the OpenCV function measure.marching_cubes() and then Poly3DCollection(), mesh faces are computed and visualized, and finally saved. In our code, due to some errors in the voxel positioning, the represented image is consistent with the one reconstructed in 3D. An improvement in the shape of the reconstructed figure would consequently be reflected in the mesh image.

Link to video

3D Voxel Visualizer 2024-02-29 21-32-12.mp4