INFOMCV Assignment 2
Authors: Sotiris Zenios, Andreas Alexandrou (Group 23)

Summary

(Briefly explain your background subtraction method, your postprocessing and how you build the voxel model. Approx. half a page.)

In the development of our background subtraction method, we employed an approach that integrates Gaussian Mixture Models (GMM) for initial background modelling, dynamic threshold optimization for enhanced foreground segmentation, and morphological operations for postprocessing the segmentation masks. Firstly, the process begins with the creation of background model for each camera with the use of GMM-based algorithm that effectively differentiates between background and foreground elements within a scene. For the foreground Detection the video frames were converted into the HSV colour space and the calculation of the absolute difference between the frames and its corresponding background model was found, thereby highlighting potential foreground regions. To refine these initial detections, a dynamic threshold optimization procedure was introduced. This procedure iteratively adjusts the thresholds for hue, saturation, and value components of the HSV space, aiming to minimize the discrepancies between the generated foreground masks and the manually segmented frames. Additionally for the post-processing the masks undergo a series of morphological operations (erosion and dilation) to eliminate the noise and fill the gaps within the detected objects in the foreground. To build the voxel model the voxels grid was generated by defining a 3D space, discretized into voxels. Specifically, the grid spans a space defined by np.linspace(-1024, 1024, num=50) for the x and y axes, and np.linspace(0, 2048, num=50) for the z-axis. The choice of 50 points along each axis offers a balance between resolution and computational efficiency. For the creation of the lookup table, camera configurations were parsed to facilitate the accurate projection of 3D voxel coordinates into 2D image planes for each camera setup, utilizing OpenCV's projectPoints function. For each of the voxel the visibility across the silhouette masks from each camera was checked. This involves determining whether the projected 2D point of a voxel falls within the foreground region and if the voxel is visible in all the camera views, it is then marked. It's colour is determined by averaging the colours from all cameras where its visible. To further refine this model, we implemented colour correction techniques, including white balance adjustment and color distribution matching, prior to the reconstruction process. The visible voxels are then compiled into a list with their 3D coordinates. Those coordinated were scaled down (divided by 16) to accommodate for the simulation's grid .The voxels are saved in a txt file along with their RGB colours, which is then read and used by the *assignment.py* file to simulate the 3D space of the grid with the voxel model and the 4 cameras.

Extrinsic parameters

(Include rotation matrix and translation for each of the four cameras. Approx. one third of a page.)

Camera 1: Rotation Matrix: 0.8260566041886146 1.9040779030869435 -1.6069595685773714
Translation Matrix: 209.90592577999203 699.966014139633 4560.913446461205

Camera 2: Rotation Matrix: 1.181097558142964 1.236209083212354 -1.2351660745938506
Translation Matrix: -245.26064790644395 1466.3225571702687 3580.9739390124114

Camera 3: Rotation Matrix: 1.6162136360247792 -0.19390614166352468 0.1726141986914336
Translation Matrix: -715.3113814234764 1248.4148090094413 2428.7855644941

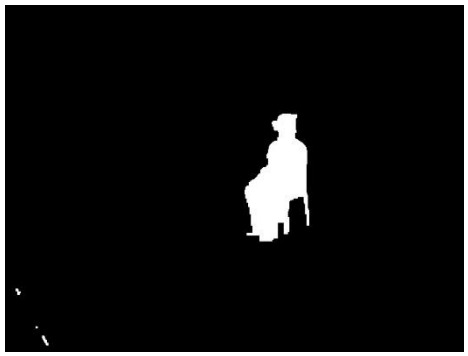
Camera 4: Rotation Matrix: 1.6063863475289146 0.5809177601466187 -0.4965647983034189
Translation Matrix: -972.4843838499992 941.882760566528 4073.9394692974024

Background subtraction

(Mention how you set 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. If you use an approach with training a background model, explain how that works. Also include a foreground image for each of the four cameras. Approx. half a page.)

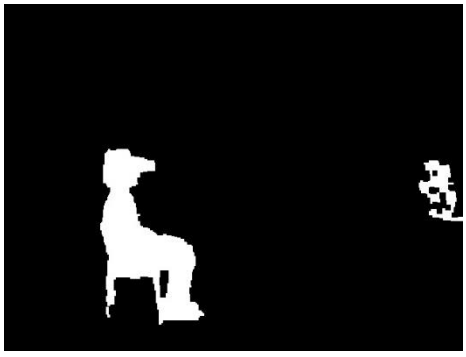
For setting the thresholds for the background subtraction a dynamic approach was used. That was achieved through a search from a range of different values of hue, saturation and value components, aiming to minimize the discrepancy between the generated foreground mask and manually segmented images. More specifically, for each combination of HSV, the generated mask undergoes morphological operations to reduce noise and improve segmentation quality, and then is compared with the manually segmented image using the XOR gate. The evaluation defines the thresholds that result in the most accurate foreground segmentation. The application of morphological operation consists of morphological opening with a 2x2 kernel which is an erosion followed by dilation. This step aims to remove noise form the foreground mask. After that morphological closing with a 7x7 kernel was used, that is basically dilation followed by erosion, to fill the gaps and small hole of the image. Finally, another morphological opening with a kernel 3x3 was use to clean the remain noise and smoothens the edges. The kernel size for each morphological operation was determined based on extensive experimentation to optimally refine the foreground mask. To determine if a pixel is foreground or background in a video sequence a method cv2.absdiff was used, utilizing the absolute difference between each frame's HSV representation and a pre-established background model in HSV space.

Camera 1 : HSV thresholds : (2, 14, 21)



Camera 2 : HSV thresholds : (0, 7, 25)

Camera 3 : HSV thresholds : (3, 7, 21)



Camera 4 : HSV thresholds : (1, 0, 27)

Choice tasks

(Indicate which ones you did, and how you did them; Approx. one third of a page.)

Choice 1: Automating the way the thresholds are determined:

The **optimize_thresholds** function was created to determine the optimal thresholds for hue, saturation, and value in the context of foreground segmentation. The function explores a predefined range of threshold values for each HSV component to find the best mask that segments the foreground from the background. For each combination of HSV the function applies morphological operation to reduce the noise and improve mask quality and then evaluates the mask's accuracy by comparing it with the manually segmented frame with the XOR gate. The combination of HSV values that yields the lowest discrepancy (closest match with the manually segmented frame), is defined as the optimal set of thresholds for each camera. This automated process for the thresholds ensures that the segmentation algorithm is finely tuned to get the most accurate foreground representation possible, boosting the overall performance of background subtraction.

Choice 2: Colouring the voxel model:

We wanted to enhance the quality of 3D reconstruction from multiple camera views by colouring the voxels. Two image preprocessing steps were employed: white balancing and color matching. White balancing was applied to correct disparities in color representation across images caused by varying lighting conditions. This was achieved by adjusting the images so that neutral colors, which should appear white or grey were correctly rendered as such, therefore ensuring a consistent baseline across all images. After white balancing, color matching was performed using the first camera's image as a reference. This step adjusted the color distribution of images from the other cameras to match that of the reference image, ensuring uniformity in color representation across all cameras. Then, when checking and saving the voxels that are visible in all scenes in *check_voxel_visibility*, the position as well as the mean colour representation at the found position of the 4 colour corrected images were saved. This uniform color representation resulted in a coherent and more accurate 3D model, as it ensures that the color of each voxel in the reconstruction accurately reflects the scene's true appearance.

Choice 3: : Speeding-up the creation of the look-up table and background subtraction using Parallelization:

The *create_lut_parallel* and *background_subtraction_parallel* functions were parallelized using Python's concurrent.futures.ThreadPoolExecutor, optimizing performance by processing camera data concurrently across four threads, therefore assigning the task of each camera to a different thread. This method efficiently handles independent tasks, processing data from multiple cameras, by executing them in parallel rather than sequentially. The ThreadPoolExecutor accelerates execution, making the system more efficient.

	Sequential	**Parallel**
create_lut	13.30 sec	6.14 sec
background_substraction	173.46 sec	73.97 sec

Link to video

https://www.youtube.com/watch?v=L-qcKw5hcrI