Francesco SANTAGIUSTINA Computer Vision

PROJECT 2 REPORT :

People Counting with Color and Depth Data

Aim, working hypothesis and scope of the project:

In this project our aim is to develop a simple computer vision application that takes in input a color image together with the corresponding depth map and is able to count the number of people in the scene. We will focus more precisely on images produced by a fixed ceiling camera in an indoor environment. This will allow us to make reliable assumptions on the angle of view and on the kind of object that we can expect to see in the images. The provided test dataset only featured walking persons as moving objects from one frame to the other (no animals and vehicles nor objects and furniture changing position). In addition to this a first frame showing no people was provided so I felt legitimated to use it as background frame for background substraction (more details later). If we should consider the production of professional software embedded in the camera or coming along it, a setup phase to get the background frame could be proposed to the user, or more advanced mechanism taking in account a dynamic background gradually changing according to some permanence criterion may be implemented but are left out of the scope of this project. Finally we suppose this people counting system to be used in everyday life conditions for “statistical data gathering” and not for secutiy purposes or other critical issues.

Chosen approach and motivations:

I choosed to mainly focus on the depth images to perform the recognition. Firstly they are very informative regarding the shape of objects and people can be recognized more easely by their shape than the color of the light they reflect which may vary according clothing, type of hair… Obviously one may recover the shape by edge detection or segmentation techniques applied directly on the color image but the variability of the viewpoint, human pose, partial occlusions of the field of view etc. lead to many difficulties. Template or pattern matching technique coul have been used but a very adaptive model would have been necessary. Secondly even if machine learning and deep learning techniques could have been the best solution for a problem of this kind, the provided images were only a dozen and I wasn’t able to find a suitable dataset for training. Indeed most of the datasets contain images taken from an oblique or horizontal viewpoint and not from an almost vertical one as in our ceiling camera case. Last but not least, it was the first time I had the occasion to deal with depth images and I was very eager to gain some experience in their handling.

Descritpion of the processing algorithm:

From a global perspective (and apart from the many low-level processing techniques), my algorithm works by first extracting the foreground and then by applying to it blob detection and, in parallel, watershed segmentation. The correct detection of individuals leading to a correct counting.

- **Background substraction and normalization:**

As anticipated in the first point, I started by using the depth image 0 (with no people) as background in order to extract the foreground of each depth map. A simple substraction of frame 0 to each other frame, followed by thresholding and morphological operations, allowed me to identify the relevant regions of each frame (the foreground containing people in our case).

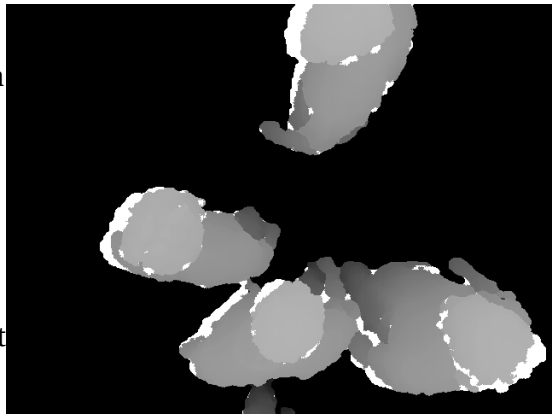

Figure 1: Normalized foreground

Then, I used the identified regions of interest as masks on the original depth maps, in order to get back the original and correct depth information (the main problem otherwise was for people placed betweeen the camera and walls, which are considerably closer than the ground to the camera, and hence caused a small difference in height between background and foreground). After normalizing the image to span the whole intensity range and applying a threshold to remove the small artifacts of the foreground image it was ready for the next stages: blob detection and watershed segmentation.

- **Blob detection:**

I actually choosed to try to apply blob detection algorithm only after having already finished the part of the code implementing watershed segmentation. What interested me was the possibility to easily set parameters to filter the found blobs according to criterias like : area, circularity, momentum, convexity, inertia etc… I was indeed worried that my code for watershed segmentation could provide to many false positives in conditions different from those of the training set. So I was happy to find a procedure that could be implemented very easely and may help me to filter out shapes which do not roughly correspond to human bodies.[1]

Furthermore in its essence the cv::SimpleBlobDetector class is very suited to detect bodies viewed form upside. The detector indeed works by slicing the image by intensity and searching for binary blobs (connected components) in each slide, finding their center and comparing it with those of other slices. In this way, binary blob present in different slices with close centers will be associated to form a blob. But in our case the intensity of the image is a proxy for the height of the object and thus all the binary blob centers will follow follow more or less the vertical axis of the body and allow a proper formation of blobs corresponding to human bodies.

- **Seed extraction and watershed segmentation:**

The watershed segmentation has been the most difficult part to implement effectively. I stress the fact that watershed segmentation require seed points to be applied. So it can be used only when you actually know how many (potential) people there are in the frame. Our counting problem must be almost solved before the watershed segmentation, which will only allow us to define the area occupied by the persons in the image. Elaborating a general algorithm which was able to find appropriate seedpoints even for picture where the depth-map “silhouettes” of different people were

1 As a starting point for the parameters of the blob detector I followed the experienced suggestions of Larry Li’s article on people detection (larrylisky.com/2016/03/03/people-tracking-using-a-depth-camera), but I tuned them to match the ceiling cam conditions.

over-lapping or contiguous has required me to adapt the standard “thresholded distance-map” procedure.[2]

First I choosed to slice the “foreground” depth-map in order to keep the more relevant part of the body (head and torso) and I applied morphological opening to remove small artifacts. Then I had to compute the distance transform of the resulting image and to normalize it. But normalizing it directly reaveled to be a bad solution because higher values are assigned to the connected components formed by more than one person (when present), thus leading to comparatively small responses on single-person connected components. For these reasons I decided to first divide the image in connected components and then normalize the depth-map of each connected component individually. Then the distance transform can be succcessfully thresholded and after applying some morphological operators to “clean” it we get our seed regions.

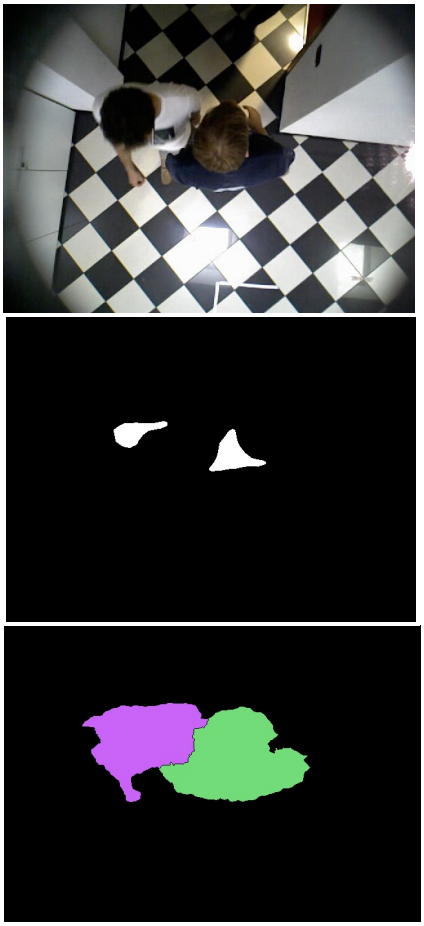

Figure 2: Color image, seeds from the depth image and segmentation

These seed regions can be passed to the cv::watershed() method to perform the segmentation of the foreground depth image. The whole point of the operation was to identify the proper area corresponding to each person (or other transient object), especially in the cases where two or more of them are contiguous or overlapping. But I noticed that the watershed algorithm applied on the depth map actually failed when the shape was too ambigous (like the “hug” picture”) even if the seed points were corrects (placed at the level of the head or neck of each person). If the shape data was not helpful to distinguish them successfully I thought that maybe the color data could help me to exit from this impasse as very often people were wearing clothes of different colors. Even if the depth and the color image do not perfectly match I decided, for sake of simplicity, to perform a pixel-wise multiplication of the two, thus obtaining a color image where only the persons were highlighted, and the luminance was proportional to the highness. The watershed segmentation algorithms handled very well this input, and was able to separate correctly all the bodies (except for the arms in some rare cases).

Results:

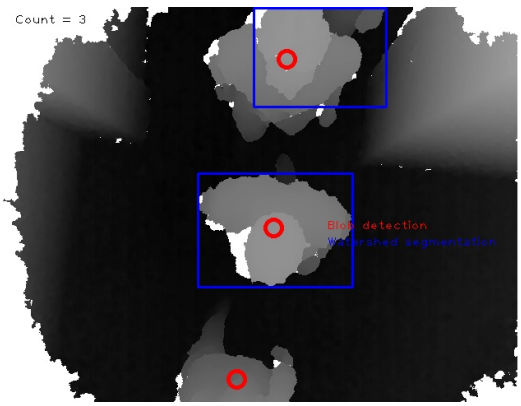

Figure 3: Final result

This algorithm perform quite well on the provided dataset but may fail to generalize to more complex situations : presence of kids or animals, people crawling or sitting etc… Obviously one of the main problems is represented by people which are only partially inside the field of view. In general if part of their torso or head is inside of the image they get detected but not if only legs are present. The following table show the result of the detections. Blob detection generally performs better than watershed segmentation for people not entirely in the field but the idea is to use both of them combined (the simplest way being to pick the maximum of the two results.)

2 Exposed in docs.opencv.org/3.1.0/d2/dbd/tutorial_distance_transform.html for instance.

Image n°	Ground truth		Blob detection	Watershed segmentation	Combined
	Entire	Partial			
0	0	0	0	0	0
1	3	0	2	3	3
2	1	1	0	2	2
3	2	0	1	2	2
4	3	0	2	3	3
5	2	1	1	2	2
6	1	0	1	1	1
7	3	0	3	3	3
8	3	0	3	3	3
9	2	1	2	2	2
10	2	1	2	2	2
11	2	1	2	2	2
12	3	0	3	3	3
13	4	0	4	4	4
14	2	0	2	2	2
15	2	2	4	3	4
16	2	2	3	2	3
17	4	1	4	4	4

Table 1: Results of the counting procedure on the given dataset

Further possible enhancements:

- One could check the correspondance between blob detection markers and watershed segmentation rectangles in order to avoid situations where for instance two blob are founds as well as two segments but only on fo them is the same so there are actually three peoples but max(2,2) = 2 and not 3.
- The segmentation could be applied to an image containing both color and depth data but merged in a more intelligent way than multiplication (a simple approach could be to use a 4-dimensional vector and then some clustering techniques).
- To avoid false positives in more challenging situations one could add a check on some other criteria once the region of interest where there is possibly a person has been found. For instance one could try to fit a head shape by using Hough transform, or use face detection if the person is turned toward the camera in order to have almost sure detections.
- In could be interesting to reproject the depth-map values into the 3D space. I have seen that this is possible and only need the calibration parameters of the camera. Once we have a cloud of points one may perform template matching with a 3D model, or compute the statistics of the distribution of heights difference (with respect to the top of the head) which are pretty characteristic (think of two main distributions, one for the head the other for the shoulder).

Additional notes:

The main file that need to be compiled and run is “source.cpp”. The shell scipt “test.sh” does it and load the images from the folder “dataset” in the current working directory. By running the program manually another path can be specified as execution argument in the following form : "/home/francesco/Desktop/Computer Vision/Projects/T2/dataset/*.png". Figures will appear quickly press any keystroke to pass to the successive one and in the end, to terminate the program.