Report lab3: People counting in Depth images

Clément Amate

January 21, 2025

1 Introduction

The ability to accurately detect and count people in various environments is an essential task in numerous applications, like surveillance, crowd management, and human-computer interaction. Traditional methods of people detection often rely on visual information from standard cameras. However, these approaches can struggle in complex conditions, such as low light, cluttered backgrounds, or occlusions. To overcome these limitations, depth images obtained from Time of Flight (ToF) sensors offer an alternative. These sensors capture the distance information of objects in the scene, allowing for more robust detection of people based on their physical dimensions and distance from the sensor.

This lab aims to develop an application for detecting and counting people using depth data from a ToF sensor. The process involves understanding the depth data through visualization, analyzing its characteristics, and then proposing an effective approach for people detection and counting. The approach will be validated by applying it to test depth images from the TIMo Dataset and evaluated with manual analysis.

2 Experiment materials

This section describes the tools, datasets, and methodologies used to perform the experiments for people counting using depth images. All code and dataset used are available on GitHub [1].

2.1 Depth data

The depth images used in this lab were obtained from a Time of Flight (ToF) sensor, which measures the distance between the sensor and objects in the environment by calculating the time it takes for emitted light to bounce back. These depth images are grayscale, where pixel intensity corresponds to the distance of the object from the sensor. Typically, the closer objects have higher intensity values, while distant objects have lower intensity values. Each image provides depth information for every pixel, which is crucial to distinguishing objects based on their spatial properties rather than color or texture. A sample depth image is shown in 1, where the varying intensities represent different depths in the scene. The raw images are rather dark for the human eye and need to be processed to be more explicit.

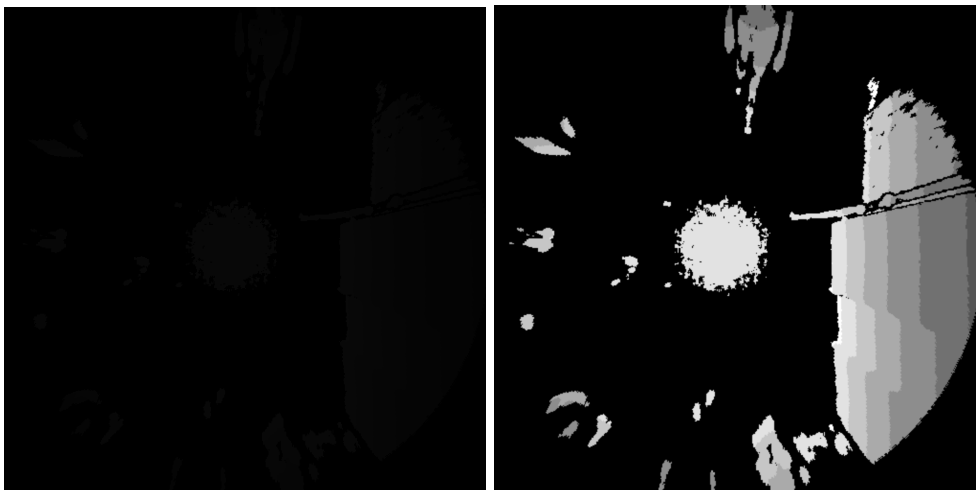

Figure 1: On the left the raw picture and on the right processed picture showing explicitly the shapes.

[1] GitHub link

2.2 Preprocessing Steps

To begin the analysis, the raw depth images were preprocessed to normalize the intensity values and eliminate unnecessary artifacts:

- **Normalization.** The depth images were first normalized to a range of [0, 1.0] using OpenCV's *cv2.normalize* function. This step ensures that the depth values are uniformly distributed, improving the stability of further calculation.

```
img_normalized = cv2.normalize(frame, None, 0, 1.0, cv2.NORM_MINMAX, dtype=cv2.CV_32F)
```

- **Masking Bright Regions.** Bright regions in the depth image, such as reflections from external light sources, were masked out. A threshold value of 0.7 was chosen, meaning any pixel with a normalized intensity above this value was excluded from further processing.

```
mask_bright = np.where(img_normalized < bright_threshold, 1, 0).astype(np.uint8)
img_no_bright = cv2.bitwise_and(img_normalized, img_normalized, mask=mask_bright)
```

2.3 Thresholding to Isolate People

After masking out overly bright regions, thresholding was applied to segment the image and isolate potential people. This was done by applying a binary threshold to the normalized depth image. Two threshold values were used:

- Low threshold: 0.3 to capture pixels representing relatively close objects.
- High threshold: 0.6 to exclude distant objects or background noise.

The result was a binary mask where pixels corresponding to people were separated from the background: As an output we would get a mask where the pixels corresponding to people are isolated from the background ones.

```
_, people_mask = cv2.threshold(img_no_bright, low_thresh, high_thresh, cv2.THRESH_BINARY)
```

2.4 Morphological Operations

To improve the quality of the thresholded mask and remove noise and small artifacts, morphological operations were applied. **Closing operation** was used to fill small holes in the detected people areas. **Opening operation** was applied to remove small isolated regions that were not relevant. We chose to use a 7x7 kernel, the kernel size is a cleaning parameter. The bigger the kernel is the heavier will be the smoothing. 7x7 kernel size is good compromise since our images does not present excessive noise.

```
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7, 7))
cleaned_mask = cv2.morphologyEx(people_mask, cv2.MORPH_CLOSE, kernel)
cleaned_mask = cv2.morphologyEx(cleaned_mask, cv2.MORPH_OPEN, kernel)
```

2.5 Contour detection and filtering

Next, contours were detected in the cleaned mask. A contour represents a boundary of connected pixels that share the same intensity value. These contours were found using OpenCV's *cv2.findContours* function, which retrieves external contours from a binary mask.

```
contours, _ = cv2.findContours((cleaned_mask * 255).astype(np.uint8), cv2.
    RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

After detecting the contours, we apply a filters to ensure that only valid contours were retained. A valid contours has most likely a **minimum Area** of 500 pixels. All contours under this value were discarded. Depending on the context and images we can imagine adding more filters such as aspect ratio or maximum size.

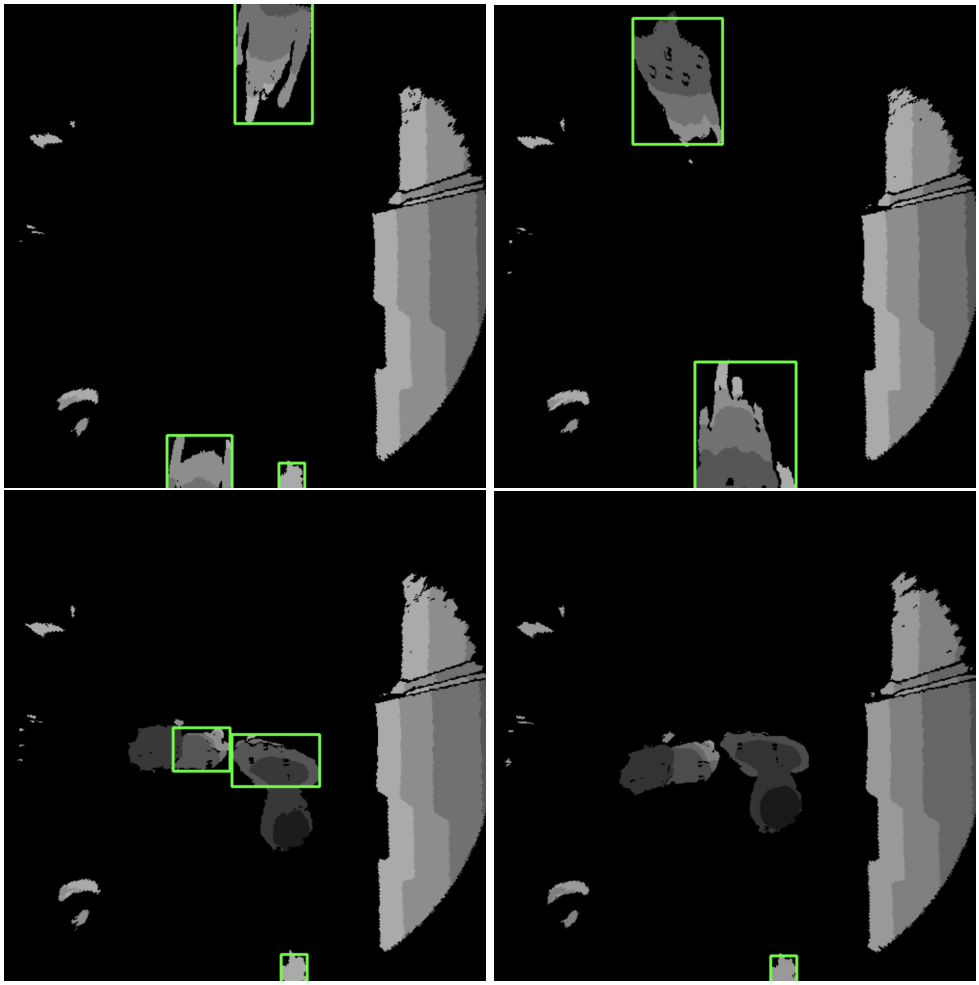

Figure 2: Picture from the first dataset where 3 people were detected

```
min_area = 500 # Minimum area for valid contours
valid_contours = []
for contour in contours:
    x, y, w, h = cv2.boundingRect(contour)
    aspect_ratio = w / h
    if min_area < cv2.contourArea(contour):
        valid_contours.append(contour)
        cv2.rectangle(output, (x, y), (x + w, y + h), (0, 255, 0), 2)
```

Finally the number of people detected is given by the length of *valid contours*.

3 Results

We test our programm with 3 different datasets to capture various context and measure the accuracy. For the first dataset, from *depth_cross*. These images are rather simple and our program manages to recognize people quite effectively. Figure 2 shows the 4 examples. The scene describes 2 people crossing the image. The scene begins by the upper-left frame and finishes with the upper-right frame. In both upper images, the people are well detected. The people are tracked during the entire scene except when crossing the exact middle for 2 frames. Bottom-right picture shows one of the two frames with undetected persons. The bottom-left image is the exact following frame. In this example, we can see that when the object is too close to the camera (the head), it is no longer detected. We can mitigate this issue by changing the lower threshold. In figure 3, we change the low threshold to 0. On the left side, the problem has been solved and we detect closer object from the camera. On the other hand, on the right side, a new problem arise where we are not able to detect previously detected person.

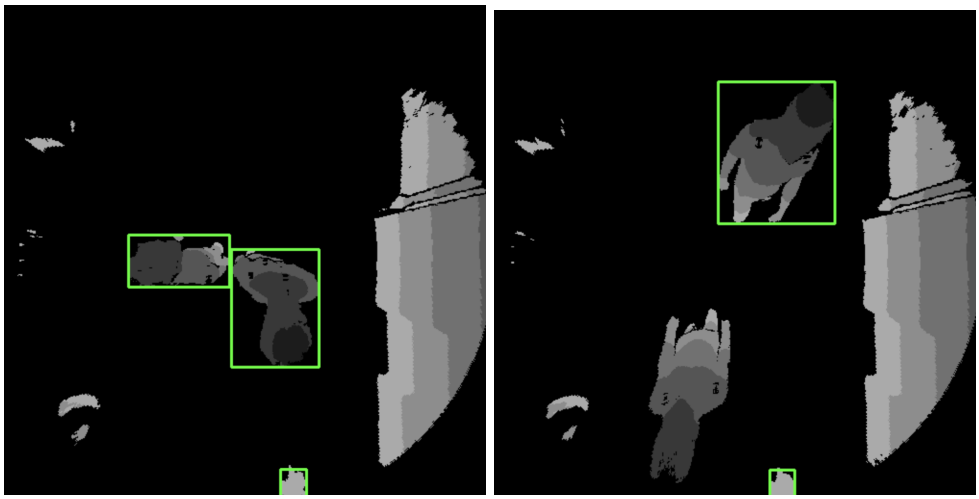

Figure 3: Test with *lower_thresh*=0

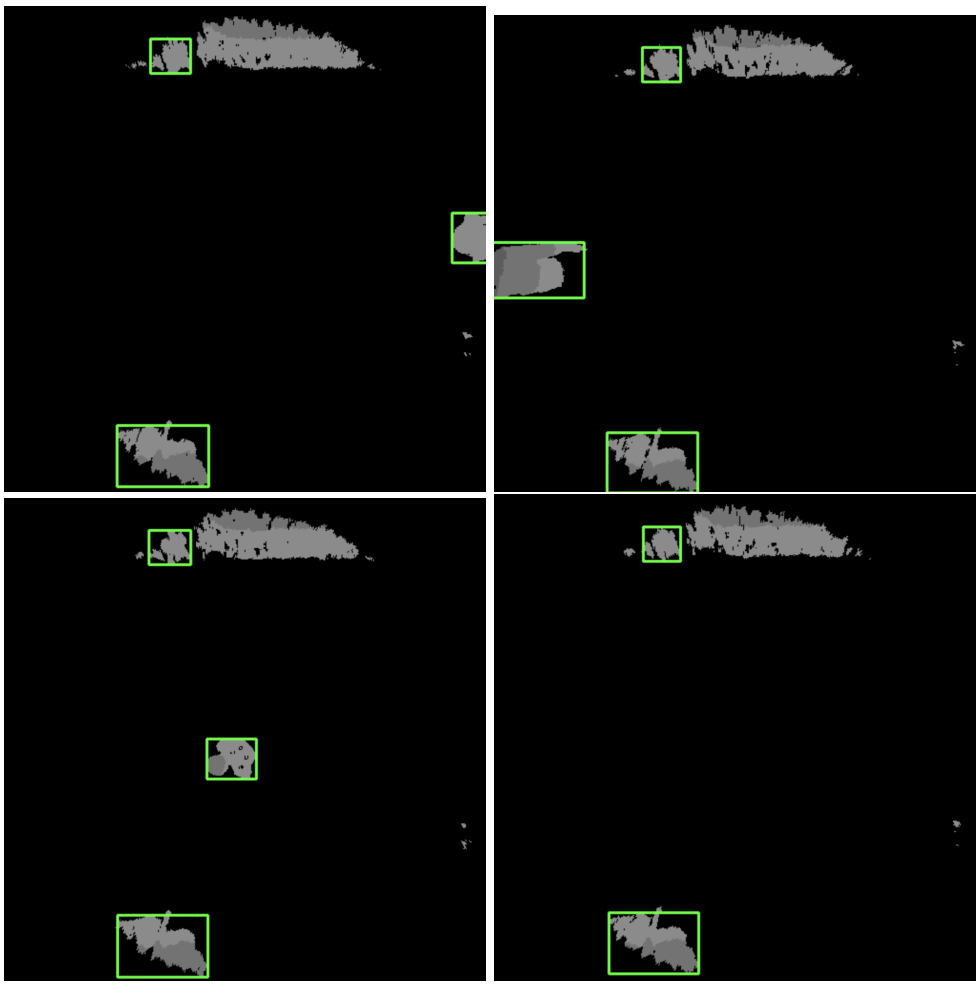

Figure 4: Test of the program with the second dataset with the first test sequence.

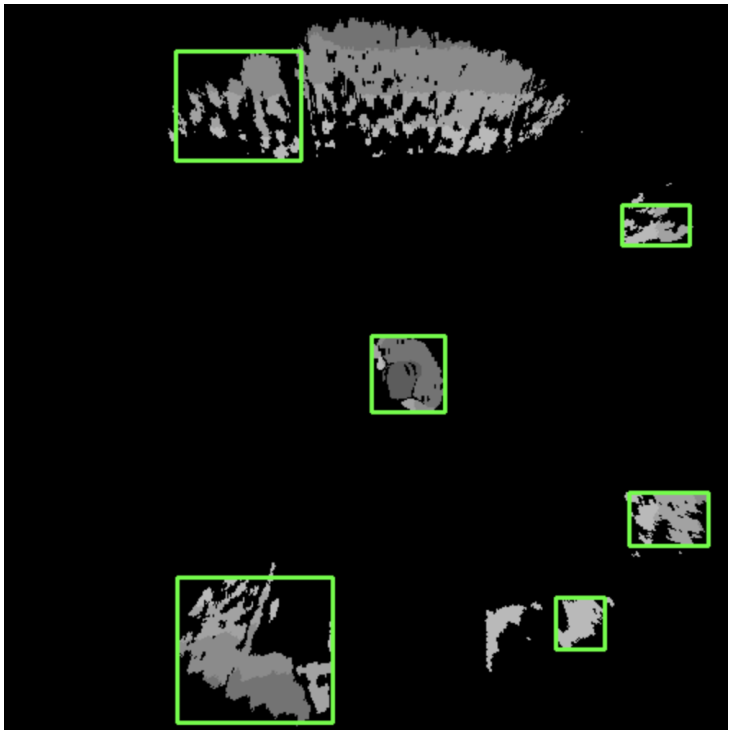

Figure 5: Test to detect the sat person with a higher bright threshold

Moreover, we observe a small detected object in all example presented in figure 2 and 3. This residual has the same color as a person and a similar height. Changing the upper threshold is not resolving the issue and provoke other miss detection to happen. We could mitigate this issue by reducing the size filter but applying this modification will over fits the data. We tested our parameters on the second dataset. Figure 4 display the results. The context is a single person crossing the scene from right to left. The top part of the figure shows the results. From the beginning to the end, the program detect the person. The person stops in the middle and sit on the floor. The bottom-left picture shows the start of the procedure. On the bottom-right, the person is sitting on the floor. As we can see, the program does not detect it since the object does not have a sufficient height. Aside from this issue, the program detects other non-human entities. We could fix this issue by adding more precise constraints on the morphological shape but it would over fits our data. In order to detect the sitting person we increase *bright_threshold*.

Figure 5 shows the result with $bright\_threshold = 0.8$. On the other hand, by increasing this parameter we consider other non relevant object. This parameter could be consider if we add more morphological constraint. In this way, we would detect only object with a human shape. As a last experimental test, we propose to try the program on a last dataset. This dataset is more complex and contain more people. The scene includes persons moving all over the room and sometimes overlapping each other. The result of our program is demonstrated by figure 6. This last test gives satisfying result. The top side of figure 6 shows well-recognized human interactions. In top-left corner, two person are stick to each other, our model count them as a single entity. As soon as they separate (top-right corner), our model starts to detect them as two different entities. The bottom-left image shows a good detection by the program, identifying the two people perfectly. The bottom-right picture is a complex situation to analyze for the program since the humans are overlapping with the light shape on the top. Those humans are not detected. To solve the problem, we would need to remove this light shape from the image using a precise threshold adapted to our context.

While all situation presented are probable to happen in real life context, some mistakes are more important than others. In a context of camera surveillance where the environment is stagnant and the only important information is the presence of a single or few human different threshold values could be used. This situation implies that a single detection is enough to recognize the potential threat. Since false positives are more important than false negatives, having an incorrect count of detected people is less critical than ensuring everyone is detected. In a strict people counting context, false negative are also important which can lead to design sharper threshold.

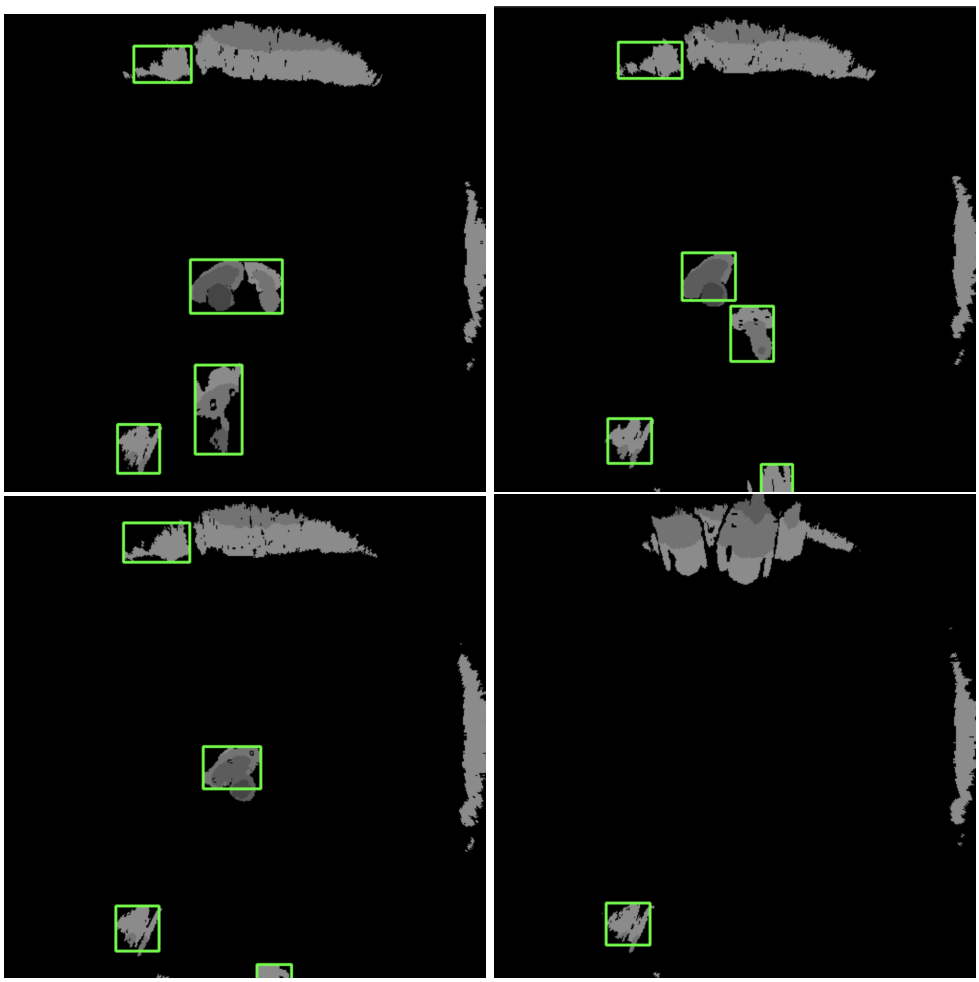

Figure 6: Result with the second test sequence. We see human interaction and more complex movement

4 Conclusion

In this lab, we developed a methodology for detecting and counting people using depth images obtained from Time of Flight (ToF) sensors. The implemented approach involved preprocessing depth data, thresholding, applying morphological operations, and filtering contours to isolate and count humans. The performance of the program was evaluated using multiple datasets with varying levels of complexity, revealing both strengths and limitations. Our method demonstrated robust detection in simpler scenarios, effectively recognizing individuals under normal conditions. However, challenges such as overlapping objects, proximity to the sensor, and non-human entities being misclassified as people highlight the need for further refinement. Adjusting thresholds and morphological constraints can mitigate some of these issues but may lead to overfitting specific datasets. The results suggest that while the system can perform well in controlled environments, real-world applications would benefit from dynamic thresholding, improved contour filtering, and incorporating additional shape-based or machine learning techniques for better generalization. Depending on the application context—such as surveillance or strict people counting—the prioritization of false positives or false negatives can guide parameter tuning. Future work could focus on integrating more advanced filtering methods or implementing deep learning approaches to improve detection accuracy in complex scenes. Despite its limitations, this project highlights the potential of depth imaging as a robust tool for people detection in challenging environments.