Sébastien BOIS - Maxime BOUET - E5AIC

People counting report

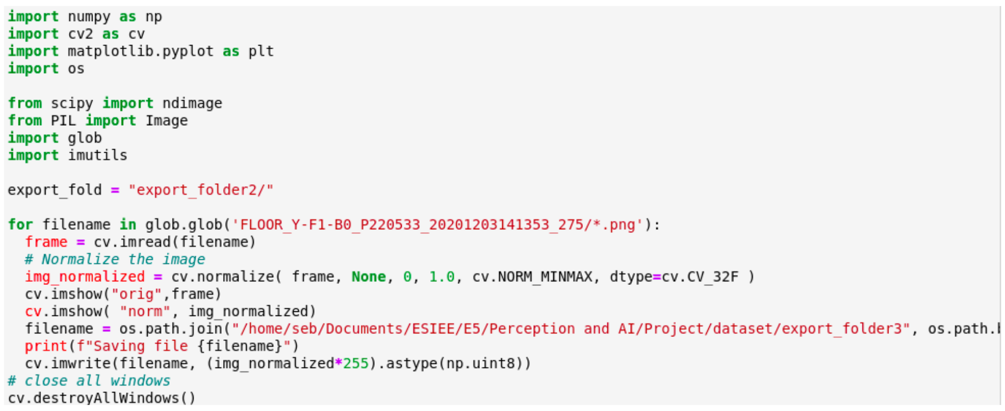
1. Data visualization and preprocessing

All the data are taken from the TiMo dataset on https://vizta-tof.dfki.de. The images are pictures taken from Depth Camera. The pictures are not sensible to the colors but only to the shapes and depth of the picture. The further the object is from the camera, the lighter he will appear on the image.

An example of the pictures taken by the camera can be found below. It is a raw image, without any treatment.

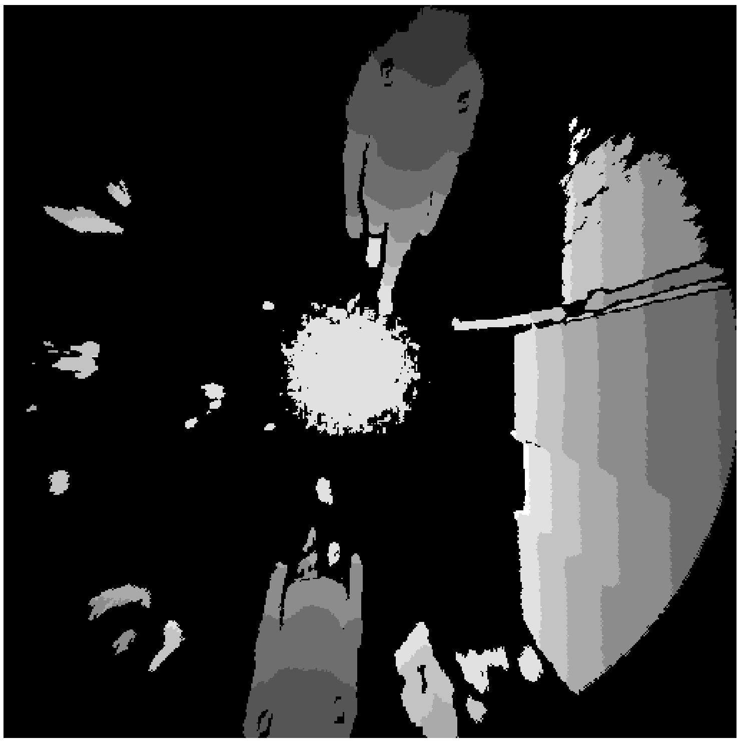
Raw CROSS_X-F1-B1_P880043_20200625111459_225_cs001_00142.png

We can see some lighter spots in the picture but it is hard to distinguish things clearly. To be able to work on the pictures, we must first normalize all of them with the following provided snippet. It loads all the images from a source folder, applies a normalizing function and exports them to a target folder.



```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
import os

from scipy import ndimage
from PIL import Image
import glob
import imutils

export_fold = "export_folder2/"

for filename in glob.glob('FLOOR_Y-F1-B0_P220533_20201203141353_275/*.png'):
  frame = cv.imread(filename)
  # Normalize the image
  img_normalized = cv.normalize( frame, None, 0, 1.0, cv.NORM_MINMAX, dtype=cv.CV_32F )
  cv.imshow("orig",frame)
  cv.imshow( "norm", img_normalized)
  filename = os.path.join("/home/seb/Documents/ESIEE/E5/Perception and AI/Project/dataset/export_folder3", os.path.
  print(f"Saving file {filename}")
  cv.imwrite(filename, (img_normalized*255).astype(np.uint8))
# close all windows
cv.destroyAllWindows()
```

Once the normalization done, the pictures are more readable and looks like the following:



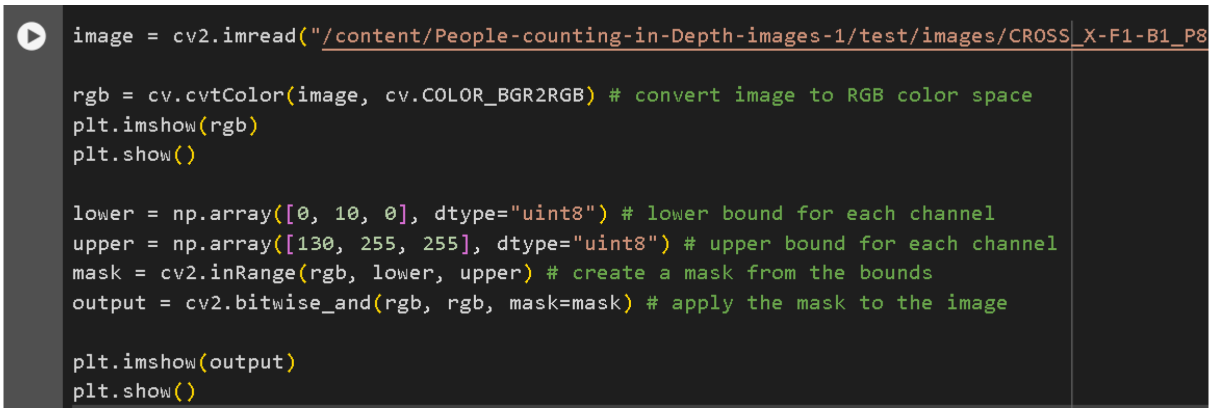
CROSS_X-F1-B1_P880043_20200625111459_225_cs001_00142.png with normalization

We can clearly distinguish the people from the ambient objects. Also, the depth differences are clearly more visible than in the raw images. For the datasets, we use three different vidéos which were decomposed in frames.

The first video represents two people walking in a room. It is used for training and validation of the model. Two other videos are used to test the model. They represent more complex scenes. The first represents three people walking in a room, sometimes people are stacked and sometimes they are distant and clearly distinguishable. The second video represents a person walking and then falling on the floor, as if he is collapsing. From this information, we can suppose that the model would struggle on detecting people and counting them because it is trained on only one specific sequence, which is very different from those used as test samples.

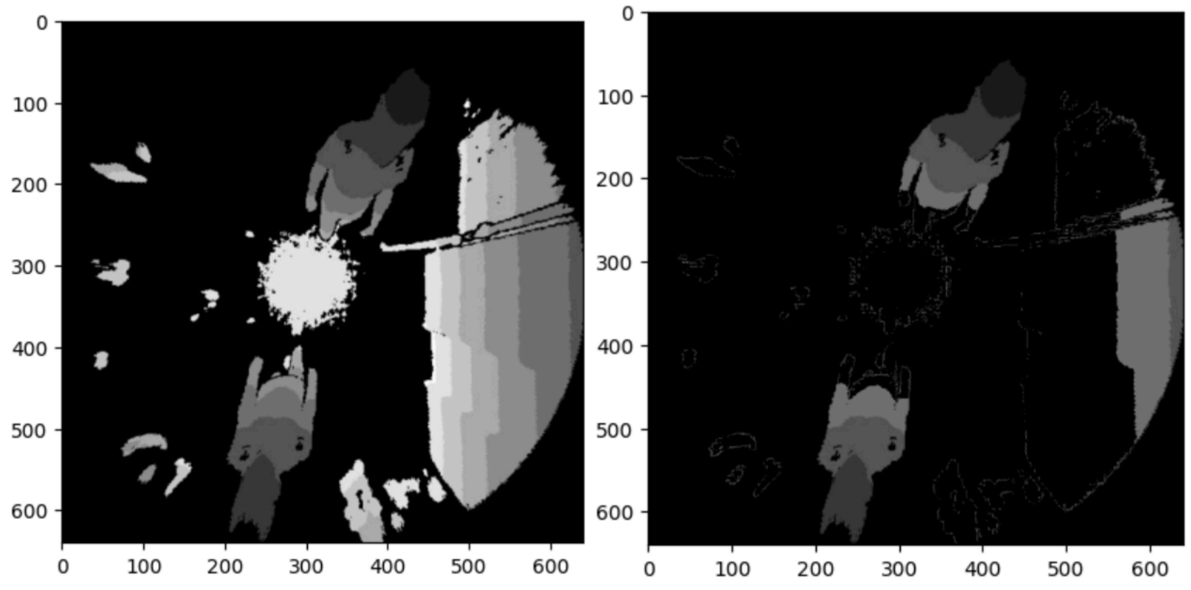
2. Conventional approach

For this part, we decided to do it in 2 stages. First, we use the same technique as the first 2 TPs where we isolate the desired parts according to the colors, only by applying a mask.



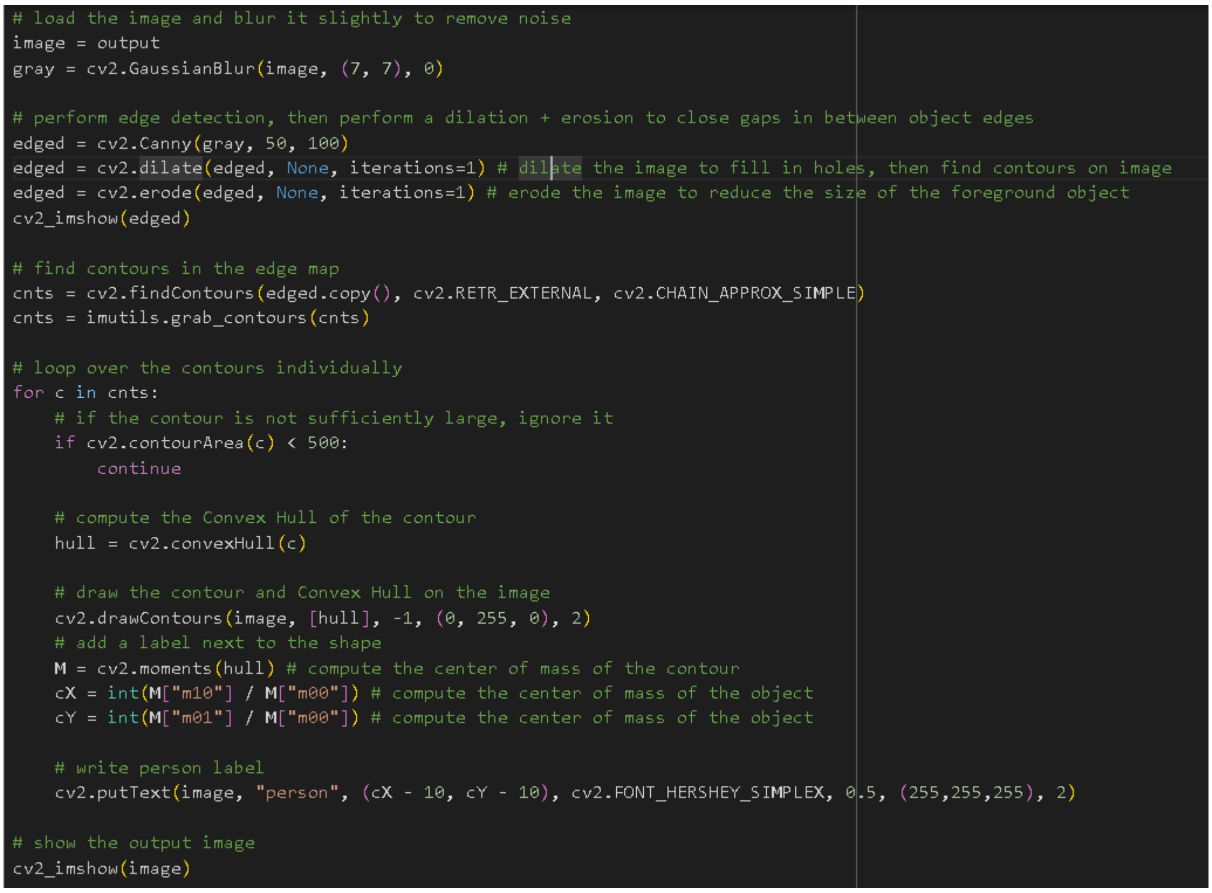
```
image = cv2.imread("/content/People-counting-in-Depth-images-1/test/images/CROSS_X-F1-B1_P8

rgb = cv.cvtColor(image, cv.COLOR_BGR2RGB) # convert image to RGB color space
plt.imshow(rgb)
plt.show()

lower = np.array([0, 10, 0], dtype="uint8") # lower bound for each channel
upper = np.array([130, 255, 255], dtype="uint8") # upper bound for each channel
mask = cv2.inRange(rgb, lower, upper) # create a mask from the bounds
output = cv2.bitwise_and(rgb, rgb, mask=mask) # apply the mask to the image

plt.imshow(output)
plt.show()
```

This mask allows you to select only black and dark gray. This allows us to apply a big filter before our second step. Here are the results:



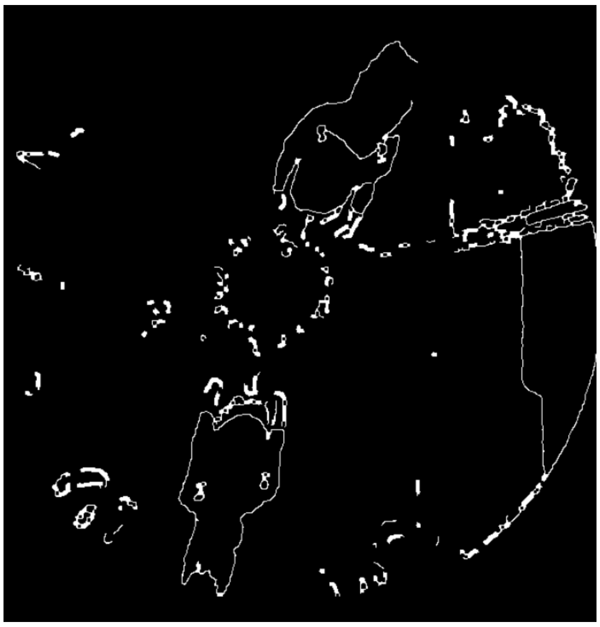
We observe that the 2 people remain but without their legs. We are starting to see the limits of this approach.

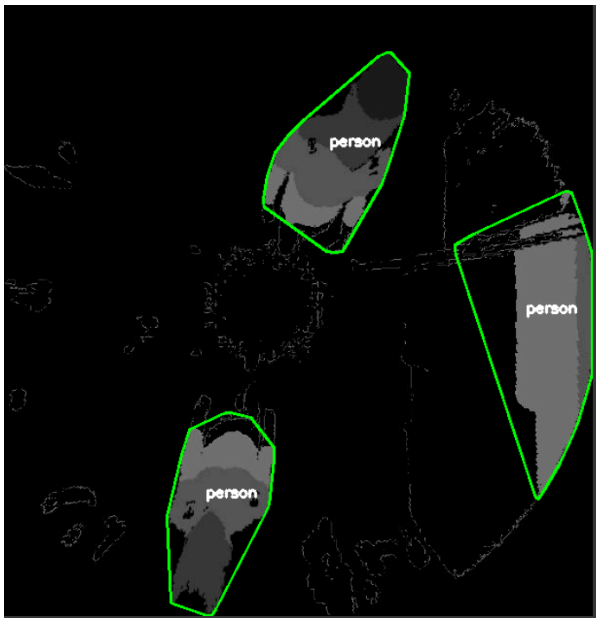
Secondly, we tried to perform detection based on the shapes on the image. We start by blurring a bit the image to remove the noise. Then we perform edge detection, dilation, and erosion to fill gaps between object edges. After that, the program has to find the contours and go through them to decide whether to keep the object or not. If the shape is too small, it's probably noise and the program should discard it. A threshold is defined to make the program able to decide if the object should be kept. Then, the last thing to do is to draw the contours for the remaining objects.



```
# load the image and blur it slightly to remove noise
image = output
gray = cv2.GaussianBlur(image, (7, 7), 0)

# perform edge detection, then perform a dilation + erosion to close gaps in between object edges
edged = cv2.Canny(gray, 50, 100)
edged = cv2.dilate(edged, None, iterations=1) # dilate the image to fill in holes, then find contours on image
edged = cv2.erode(edged, None, iterations=1) # erode the image to reduce the size of the foreground object
cv2_imshow(edged)

# find contours in the edge map
cnts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)

# loop over the contours individually
for c in cnts:
    # if the contour is not sufficiently large, ignore it
    if cv2.contourArea(c) < 500:
        continue

    # compute the Convex Hull of the contour
    hull = cv2.convexHull(c)

    # draw the contour and Convex Hull on the image
    cv2.drawContours(image, [hull], -1, (0, 255, 0), 2)
    # add a label next to the shape
    M = cv2.moments(hull) # compute the center of mass of the contour
    cX = int(M["m10"] / M["m00"]) # compute the center of mass of the object
    cY = int(M["m01"] / M["m00"]) # compute the center of mass of the object

    # write person label
    cv2.putText(image, "person", (cX - 10, cY - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255,255,255), 2)

# show the output image
cv2_imshow(image)
```

You can find the results of this conventional approach below:





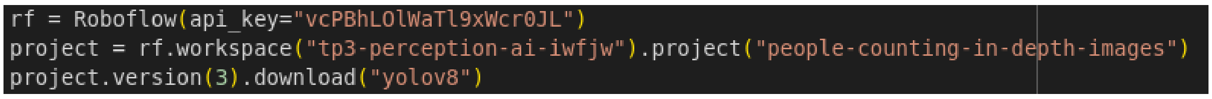
The result is quite good because we are in an easy case. If people are lying down, nothing will be detected because the people will not pass the first filter based on colors. It shows the bottleneck of the detection using colors. Indeed, sometimes people may be further to the camera whereas they can be closer. Also, an object is not necessarily a person if it passes the size threshold. In this example, we clearly notice a wall detected as a person. Lastly, if two people are colliding or very close, the algorithm will not be able to detect whether it is one person or two people.

3. AI-based approach

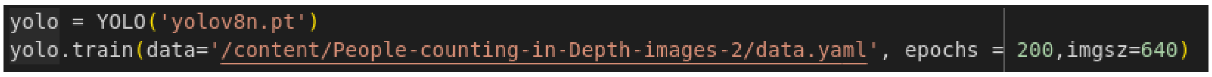
For this part, we choose to use the YoloV8 deep neural network to detect the people in the different pictures. From these predictions, we can count the number of people and evaluate the distances between them.

RobotFlow is used to label the different images and to augment the data of the training set. There is a total of 78 training images including augmentation with different image rotations, from -180° to 180°. There are also different brightness from -30% to 30% and noise up to 1.45% of the pixels. For the images to be usable by the YoloV8, it must be resized to 640x640.

Once the preprocessing and augmentation steps are done, we can import the data with the RobotFlow API in python. The YoloV8 is imported using the Ultranalytics API and is trained with the training dataset.

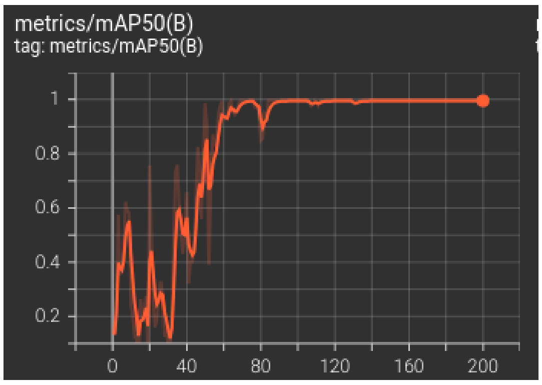


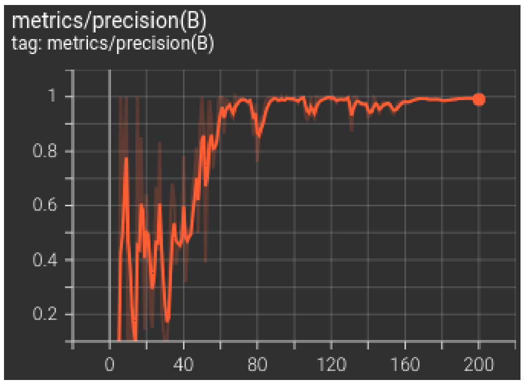
```
rf = Roboflow(api_key="vcPBhLOlWaTl9xWcr0JL")
project = rf.workspace("tp3-perception-ai-iwfjw").project("people-counting-in-depth-images")
project.version(3).download("yolov8")
```

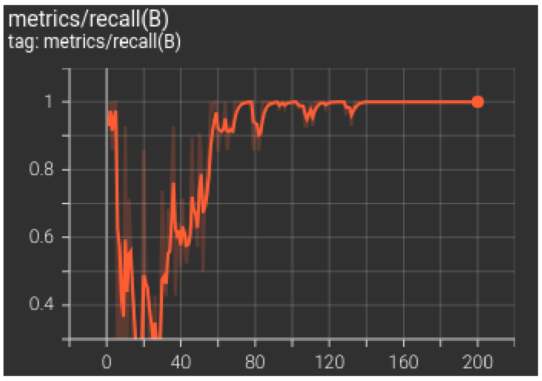


```
yolo = YOLO('yolov8n.pt')
yolo.train(data='/content/People-counting-in-Depth-images-2/data.yaml', epochs = 200,imgsz=640)
```

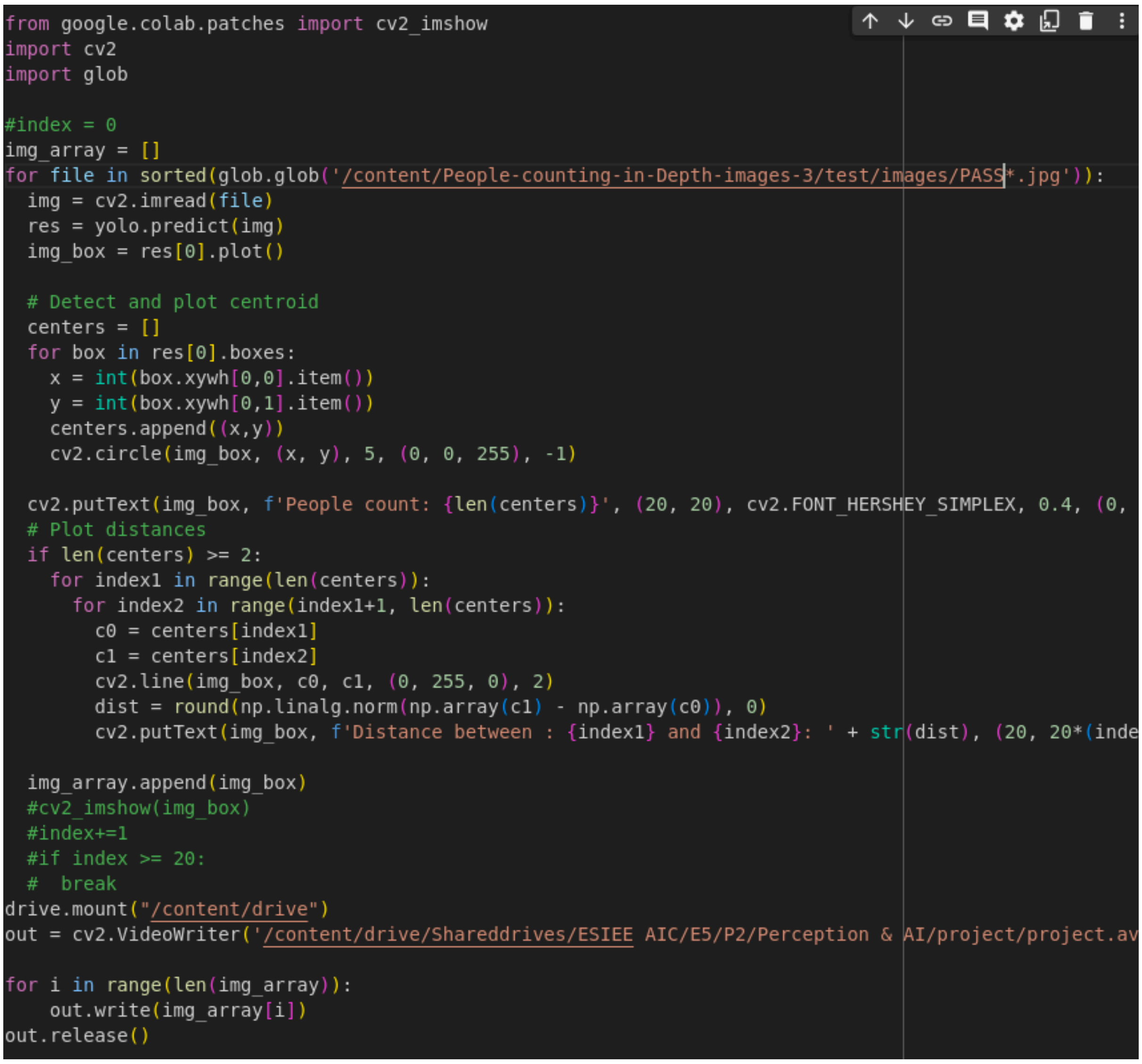
The model is trained over 200 epochs and 78 images. The following metrics are retrieved using tensorboard. We notice that the training is very efficient by looking at the curves converging around 100% for the mAP50, the precision and the recall. The results are very good since the validation samples are from the same video sequence as the training samples. To have a real idea of the effectiveness of the predictions, we must test on the test samples which are from different videos.





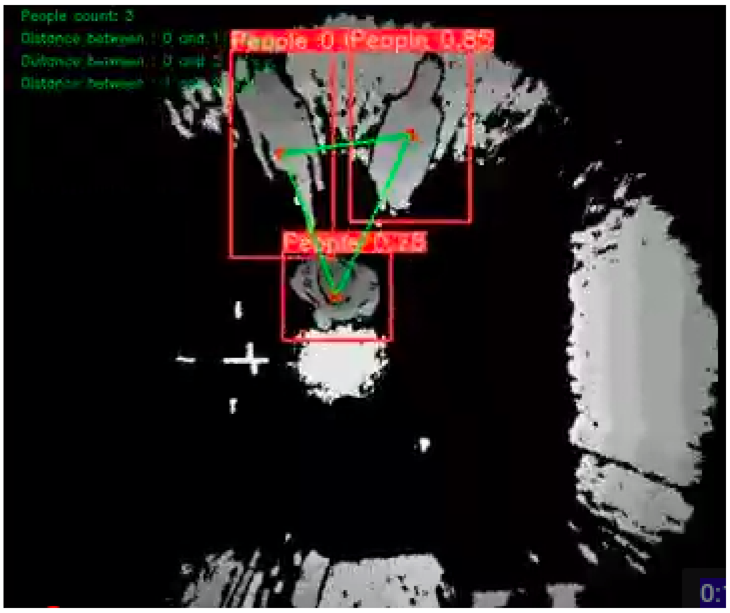


We created a code sample which detects people and uses a box to contour their shape. It displays the score of the detection, the center of the box and draws lines between each center of boxes. Distances are the length of these lines. Lastly, we display the number of people in each picture. All the images are then reassembled to form the original video with the detection features.



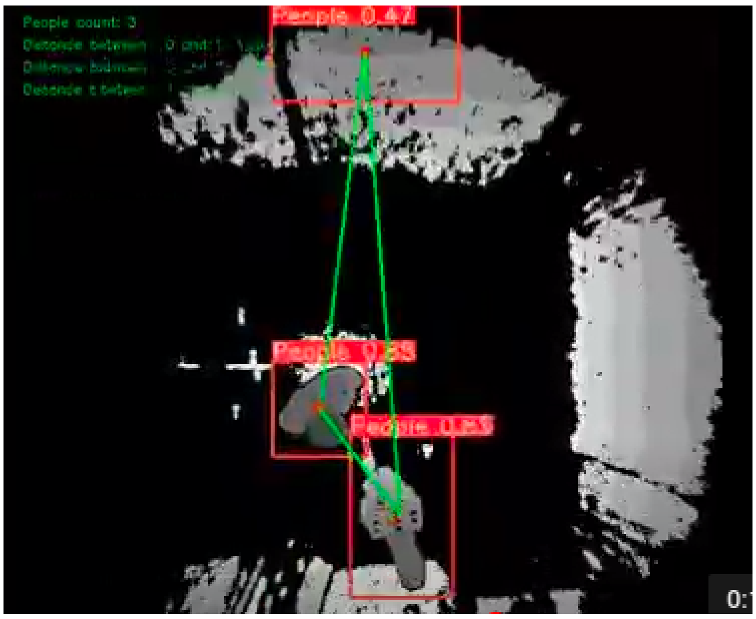
```
from google.colab.patches import cv2_imshow
import cv2
import glob

#index = 0
img_array = []
for file in sorted(glob.glob('/content/People-counting-in-Depth-images-3/test/images/PASS*.jpg')):
  img = cv2.imread(file)
  res = yolo.predict(img)
  img_box = res[0].plot()

  # Detect and plot centroid
  centers = []
  for box in res[0].boxes:
    x = int(box.xywh[0,0].item())
    y = int(box.xywh[0,1].item())
    centers.append((x,y))
    cv2.circle(img_box, (x, y), 5, (0, 0, 255), -1)

  cv2.putText(img_box, f'People count: {len(centers)}', (20, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.4, (0,
  # Plot distances
  if len(centers) >= 2:
    for index1 in range(len(centers)):
      for index2 in range(index1+1, len(centers)):
        c0 = centers[index1]
        c1 = centers[index2]
        cv2.line(img_box, c0, c1, (0, 255, 0), 2)
        dist = round(np.linalg.norm(np.array(c1) - np.array(c0)), 0)
        cv2.putText(img_box, f'Distance between : {index1} and {index2}: ' + str(dist), (20, 20*(inde

  img_array.append(img_box)
  #cv2_imshow(img_box)
  #index+=1
  #if index >= 20:
  #  break
drive.mount("/content/drive")
out = cv2.VideoWriter('/content/drive/Shareddrives/ESIEE AIC/E5/P2/Perception & AI/project/project.av

for i in range(len(img_array)):
    out.write(img_array[i])
out.release()
```

Globally, we obtain good results if we consider that the YoloV8 model has been fine tuned on only one video sequence, which is pretty classic with two people walking. Indeed, the model is able to detect the right number of people and to generate box fitting well to their hitbox.



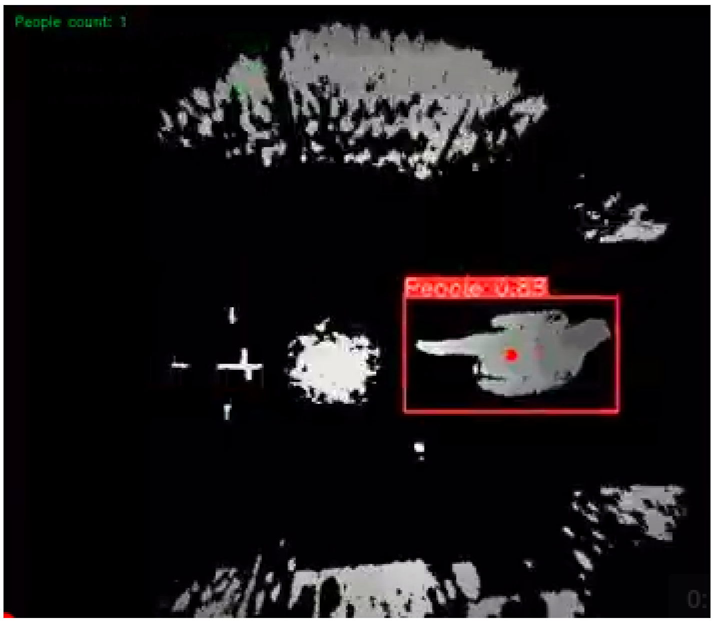
Good detection on the PASS video sequence

However, there are also some frames on which we notice a problem. For example, since the model has only been trained over a video sequence on which only the right wall appears and people are coming from the top, the model tends to detect people at the emplacement of the wall at the top of the picture.



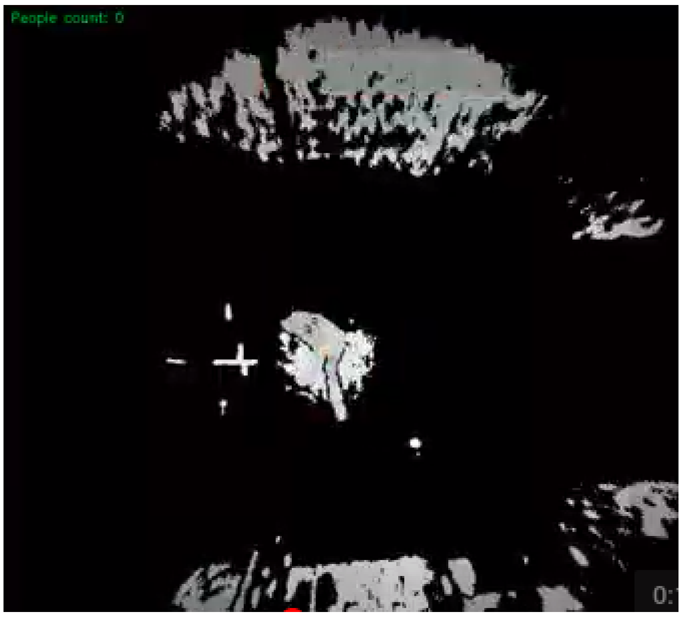
Bad detection with a well misclassified as a person on the PASS video sequence

On the other hand, we also tried to use the same training to detect a person who decides to lie on the floor in the middle of the video. Because the model is used to detect people standing and walking, it detects pretty well the unique person at the beginning of the video.



Good detection in the FLOOR video sequence

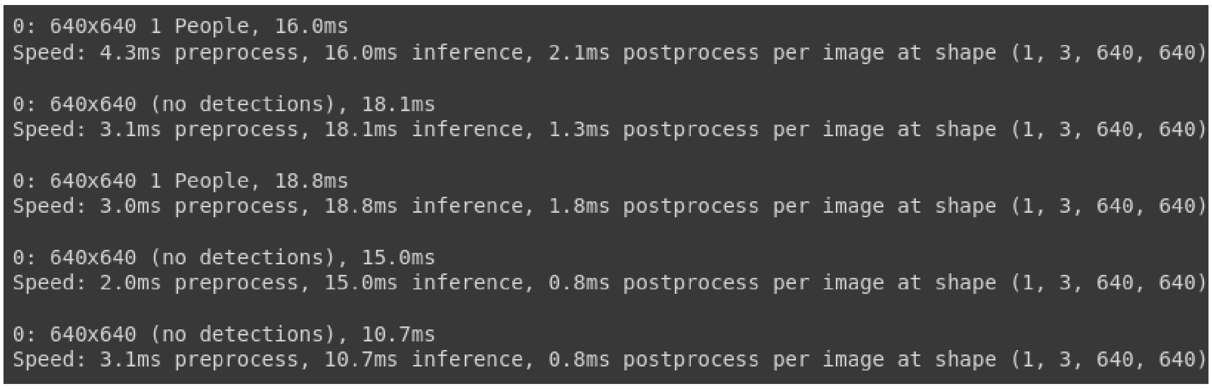
However, the model struggles to detect people lying on the floor because he has never seen any example during the training stage. Sometimes, it succeeds in detecting the person but most of the time it does not recognize anything.



Bad detection with a person lying on the floor in the FLOOR video sequence

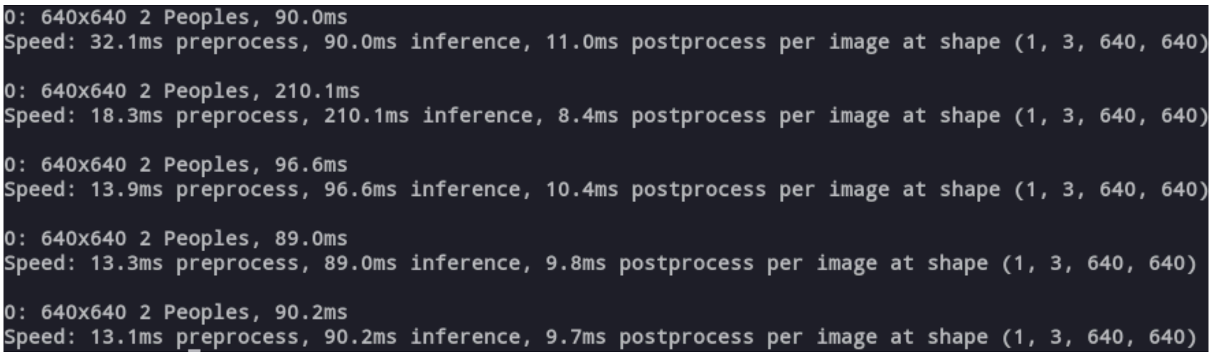
4. Embedded program on Nvidia Jetson Nano

In the laptop version of the program, detection on test samples takes at most 15ms. On the other hand, if we test the same program on the Nvidia Jetson Nano microcontroller, we obtain a far higher computation time. Indeed, detection on an embedded device often takes more time than in our modern computers because of the difference in components and consumption. Here, the detections are performed using the torchscript format because it is the only one working on the Jetson Nano currently.



```
0: 640x640 1 People, 16.0ms
Speed: 4.3ms preprocess, 16.0ms inference, 2.1ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 (no detections), 18.1ms
Speed: 3.1ms preprocess, 18.1ms inference, 1.3ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 1 People, 18.8ms
Speed: 3.0ms preprocess, 18.8ms inference, 1.8ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 (no detections), 15.0ms
Speed: 2.0ms preprocess, 15.0ms inference, 0.8ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 (no detections), 10.7ms
Speed: 3.1ms preprocess, 10.7ms inference, 0.8ms postprocess per image at shape (1, 3, 640, 640)
```

Computation times on Google collab environment using a T4 GPU runtime

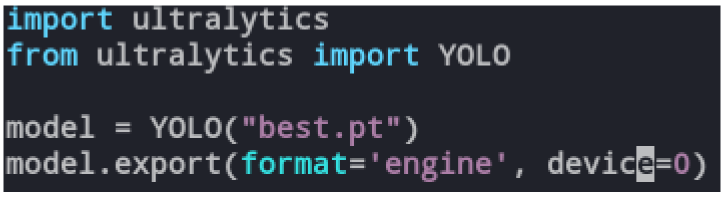


```
0: 640x640 2 Peoples, 90.0ms
Speed: 32.1ms preprocess, 90.0ms inference, 11.0ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 2 Peoples, 210.1ms
Speed: 18.3ms preprocess, 210.1ms inference, 8.4ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 2 Peoples, 96.6ms
Speed: 13.9ms preprocess, 96.6ms inference, 10.4ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 2 Peoples, 89.0ms
Speed: 13.3ms preprocess, 89.0ms inference, 9.8ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 2 Peoples, 90.2ms
Speed: 13.1ms preprocess, 90.2ms inference, 9.7ms postprocess per image at shape (1, 3, 640, 640)
```

Computation times on the Nvidia Jetson Nano using the embedded GPU

We notice that the computation time is almost ten times greater than for the google collab environment. An idea to improve the performance and make the model embedded-friendly would be to use a more optimized version of the YoloV8 model. The TensorRT model for example, is optimized to perform operations on small microcontroller cards. To convert our base model to the optimized one, we must first download the ‘best.pt’ model from the training environment, upload it to the embedded card and export the script in this new environment by specifying the format and the type of device. We declare device to 0 to let the API know that the model should compute over the GPU and not the CPUs.

The following piece of code summarizes the conversion explanations:



```
import ultralytics
from ultralytics import YOLO

model = YOLO("best.pt")
model.export(format='engine', device=0)
```

Sadly, because of some import issues on the ultralytics.utils library, we can't export it as an optimized version. In theory, we should expect better performance using this model.

Source code

All the source codes can be found at the following github:
https://gitlab.com/Swixiz/people-counting