Human Detection and Segmentation

Marco Ferri - 13th October 2020

The purpose of this report is to compare different approaches for background removal.

Human Segmentation

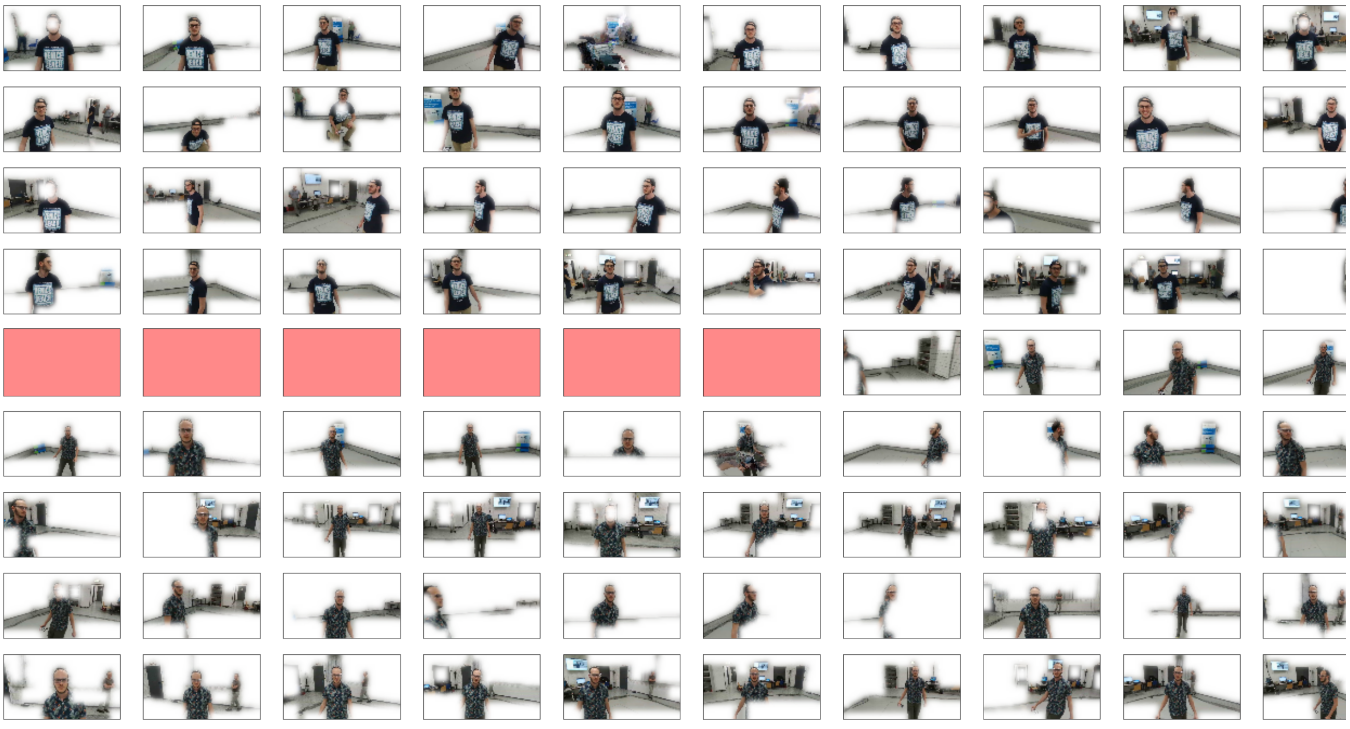
Canny Edge Detection

Simple version

The first algorithm that has been tried is based on Canny Edge Detection for finding the edges of the image and so the contours. These are used for creating a mask around the subject in the image, for isolating it from the background. A core aspect of the Canny function is the correct choice of the parameters `minVal` and `maxVal`, since it is essential for distinguishing the subject edges from the rest and later keeping the right contours.
Several experiments have been made with different parameters, but **no combination of `minVal` and `maxVal` is optimal** with respect to the given dataset. Some scenes would require more filtering, while others may disappear completely if strong filtering is chosen; only deep human dataset inspection can lead to the optimal background removal, which may require a lot of time. Up to now, the whole dataset has been filtered with the same `minVal` and `maxVal` parameters, respectively set to 100 and 400.

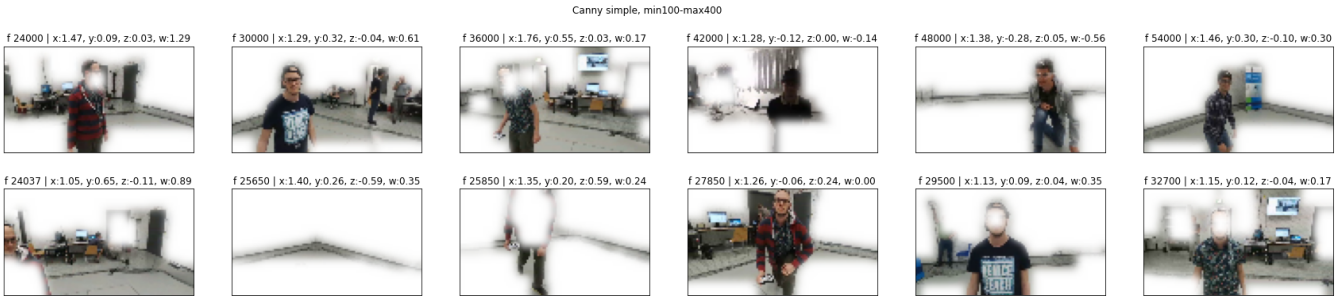
Most of the time the person is well detected, while other times it completely disappears or even results in a **fatal error** (red frames below). Room baseboard (line between the floor and the wall) is often still present in the image, but it becomes almost invisible when merged with replacement backgrounds. However, many examples just keep **too much background** or it even happens that the body of the person is present in the image while its **face disappears**.

Example below shows how background removal performed on a subset of the dataset.



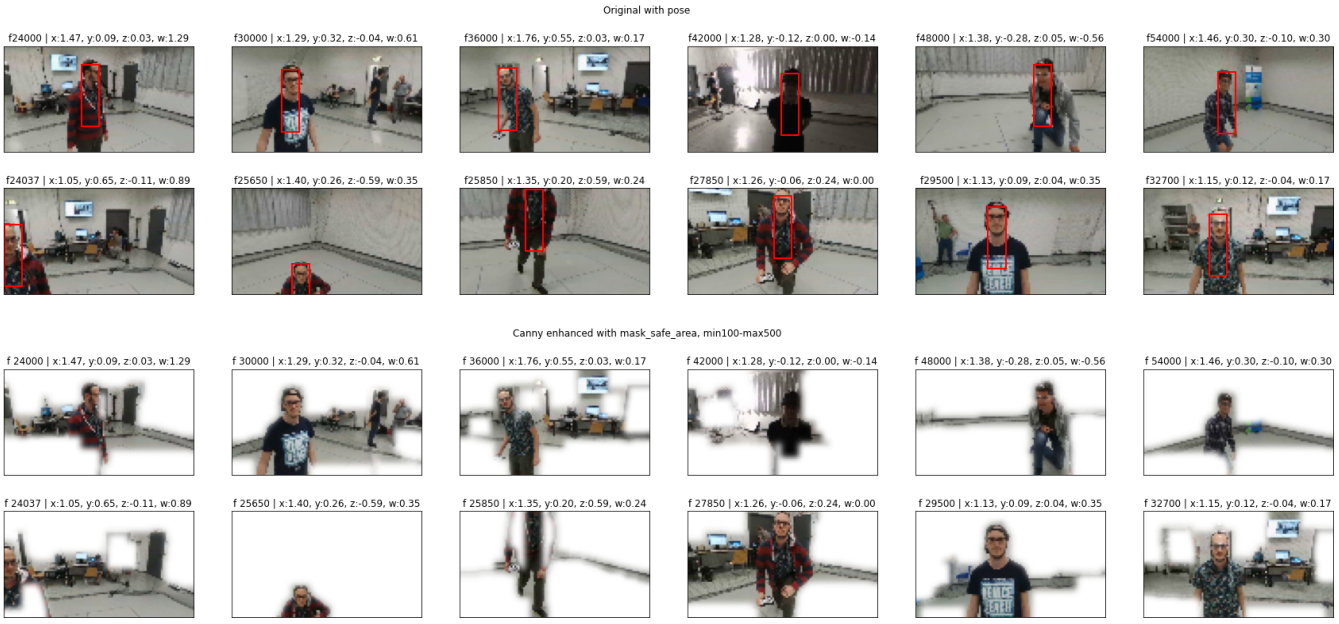
Enhanced version

The algorithm shows two main issues: too much background is often kept and sometimes the face disappears while the body of the person is still present in the image. Since **the face is crucial** for the model to work properly, we propose a very simple solution: the face area will always be kept.



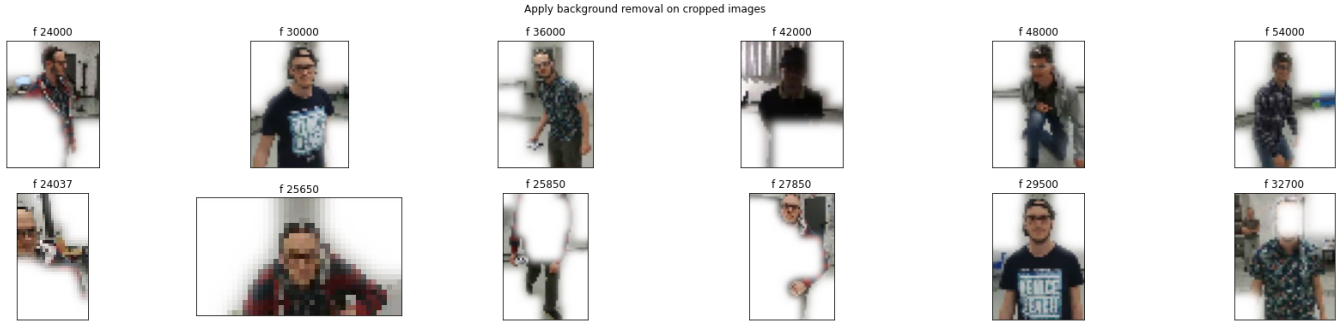
Consider the examples above, a subset of the dataset where the face or the body disappears. For mitigating the problem, let's **integrate the person position** inside the image. Firstly, a pose-to-image coordinates conversion has been tried but, since this requires a very precise face position detection and the data are too noisy, the approach has been discarded.

For now, let's **manually insert the face coordinates** and expand the area for including part of the body which is shown to disappear sometimes. Running the algorithm with such an improvement, we obtain better images that are shown below, but still we do not solve the problem of having too much background into the resulting images.



Alternative version

In order to further mitigate the problem of having too much background, we also tried to run the algorithm on **cropped images**, to see if it works on such a case. Since the **results are very poor**, Canny's possibilities stop here. It might be that another approach is better for background removal.



GrabCut Algorithm

Given the Canny Edge Detection limits in removing the background from the images, the GrabCut algorithm has been tried. It operates using the subject position in the image and some statistical inference for labeling each pixel of the image as *background* or *foreground*. See the example below.

The algorithm is the following (source):

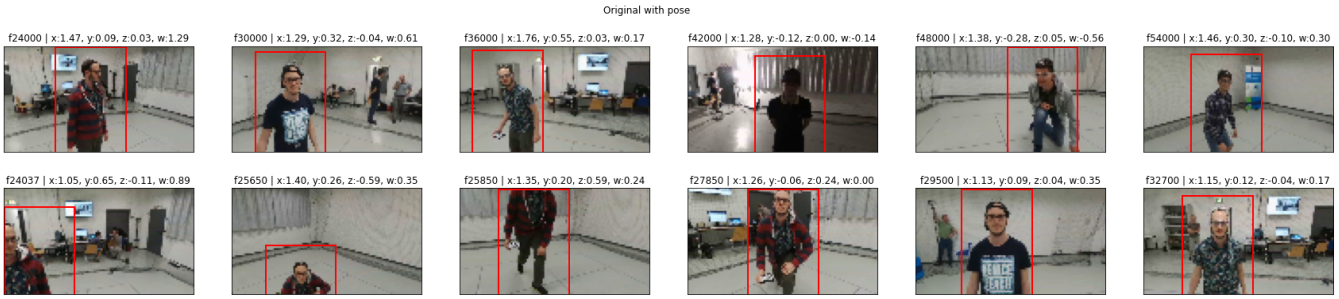
- User inputs the rectangle. Everything **outside this rectangle will be taken as *sure background***. Everything **inside the rectangle is *unknown***. Similarly any user input specifying foreground and background are considered as hard-labelling which won't change in the process.
- Computer does an initial labelling depending on the data we gave. It labels the foreground and background pixels (or it hard-labels).
- Now a **Gaussian Mixture Model** (GMM) is used to model the foreground and background.
- Depending on the data we gave, GMM learns and creates a new pixel distribution. That is, the **unknown pixels are labelled either *probable foreground* or *probable background* depending on its relation with the other hard-labelled pixels** in terms of color statistics (like clustering).
- **A graph is built** from this pixel distribution. Nodes in the graphs are pixels. Additional two nodes are added, Source node and Sink node. Every foreground pixel is connected to the Source node and every background pixel is connected to the Sink node.
- The **weights of edges connecting pixels to source node/end node are defined by the probability of a pixel being foreground/background**. The weights between the pixels are defined by the edge information or pixel similarity. If there is a large difference in pixel color, the edge between them will get a low weight.
- Then a **mincut algorithm is used to segment the graph**. It cuts the graph into two, separating the source node and the sink node with minimum cost function. The cost function is the sum of all weights of the edges that are cut. After the cut, all the **pixels connected to the Source node become *foreground*** and **those connected to the Sink node become *background***.
- The process is continued until the **classification converges**.

OpenCV GrabCut function has two initialization modalities. You can **pass the rectangle**, as described above, or a **mask of the image** in which you specify whether a certain pixel is *sure-background*, *probable-background*, *probable-foreground* or *sure-foreground*. These classes are also used by the library during the algorithm itself.

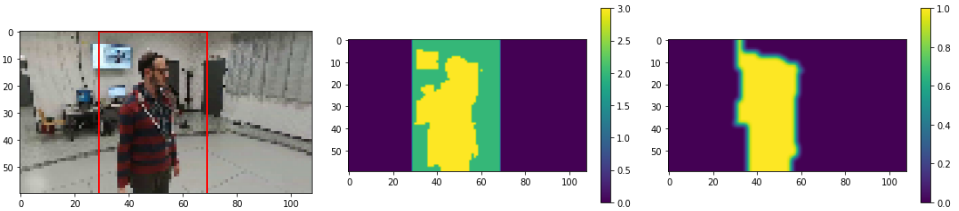
Both the approaches require a previous knowledge about the **subject position in the image**. For now, let's analyze both by **manually inserting** this information for the example images.

Rectangle initialization

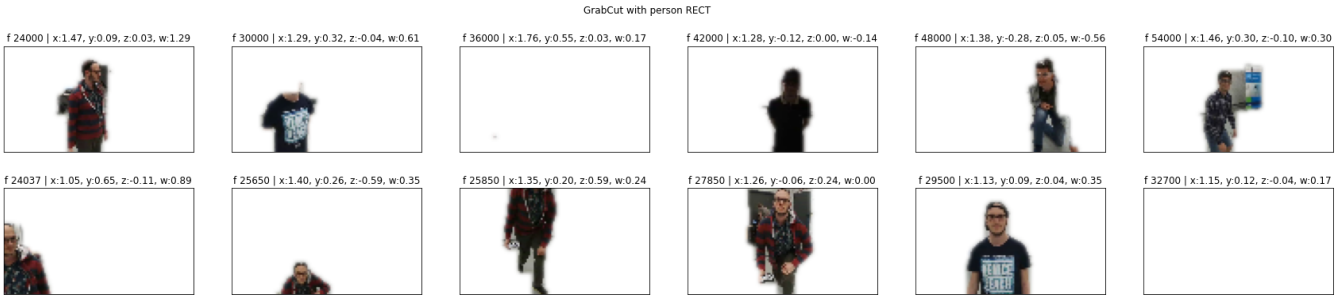
This approach requires that, together with the image to process, also a rectangle is given which must **entirely contain the subject** to isolate from the background. The pose information already used before, which corresponds to the face+body position, has been used to approximately derive also the person position in the image. This has been passed to the algorithm.



GrabCut proceeds as follows. Area inside the rectangle is marked as *probable-background* (green), while the pixels outside are *sure-background* (blue). As the algorithm keeps going, it **finds pixels inside the rectangle which can be foreground**, marking them as *probable-foreground* (yellow). Later, we **binarize and smooth** the mask, finally removing the background from the original image.



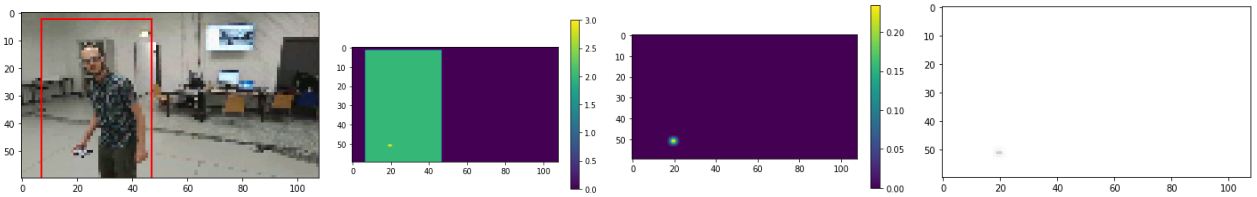
The same examples of before are shown here. Results seem better than the ones produced by Canny Edge Detection. However, it happens that **the face or the entire person is filtered out** of the image.



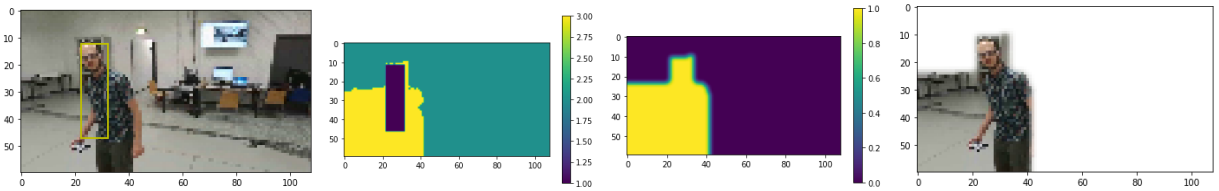
Mask initialization

In order to improve the previous technique we try mask initialization, which allows us to initially classify the pixels in the image as we prefer. This time, we do consider the pose information for telling GrabCut we already know that some part of the image is *sure-foreground*: face and part of the body.

Let's consider one of the failures shown above, for which the algorithm completely misses the person. Notice that *sure-background* is blue, *probable-background* is green and *probable-foreground* is yellow.

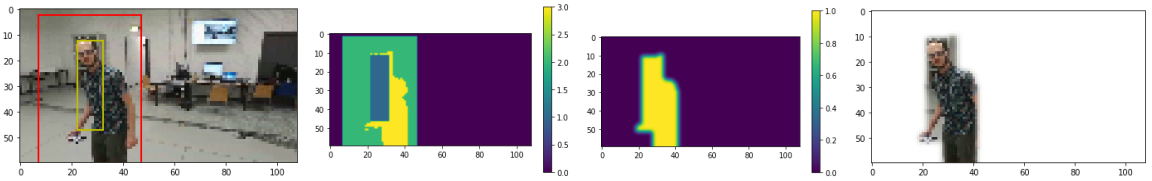


Now, instead of choosing rectangle initialization, we opt for mask initialization and specify *sure-foreground* pixels. In the second image below, we see manually assigned *sure-foreground* pixels in blue, while GrabCut inferred *probable-foreground* in yellow and *probable-background* in green. Results are undoubtedly better, but we notice that the left-most background has been kept in the final image, while we could easily identify it as *sure-background* using the person pose of before.



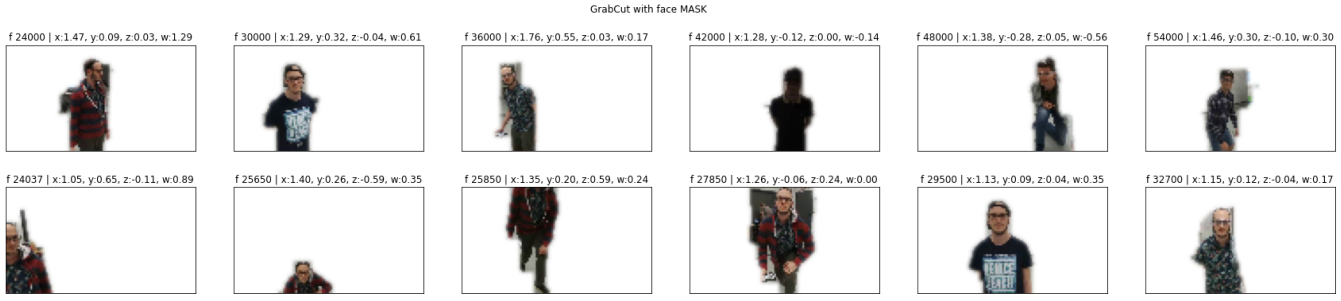
Hybrid initialization

We finally try a mixed approach between rectangle and mask initialization, both **specifying person and face positions** in the image. It allows us to initially set *sure-background*, *probable-background* and *sure-foreground* pixels. **Only *probable-foreground* pixels have to be found** by the GrabCut algorithm.



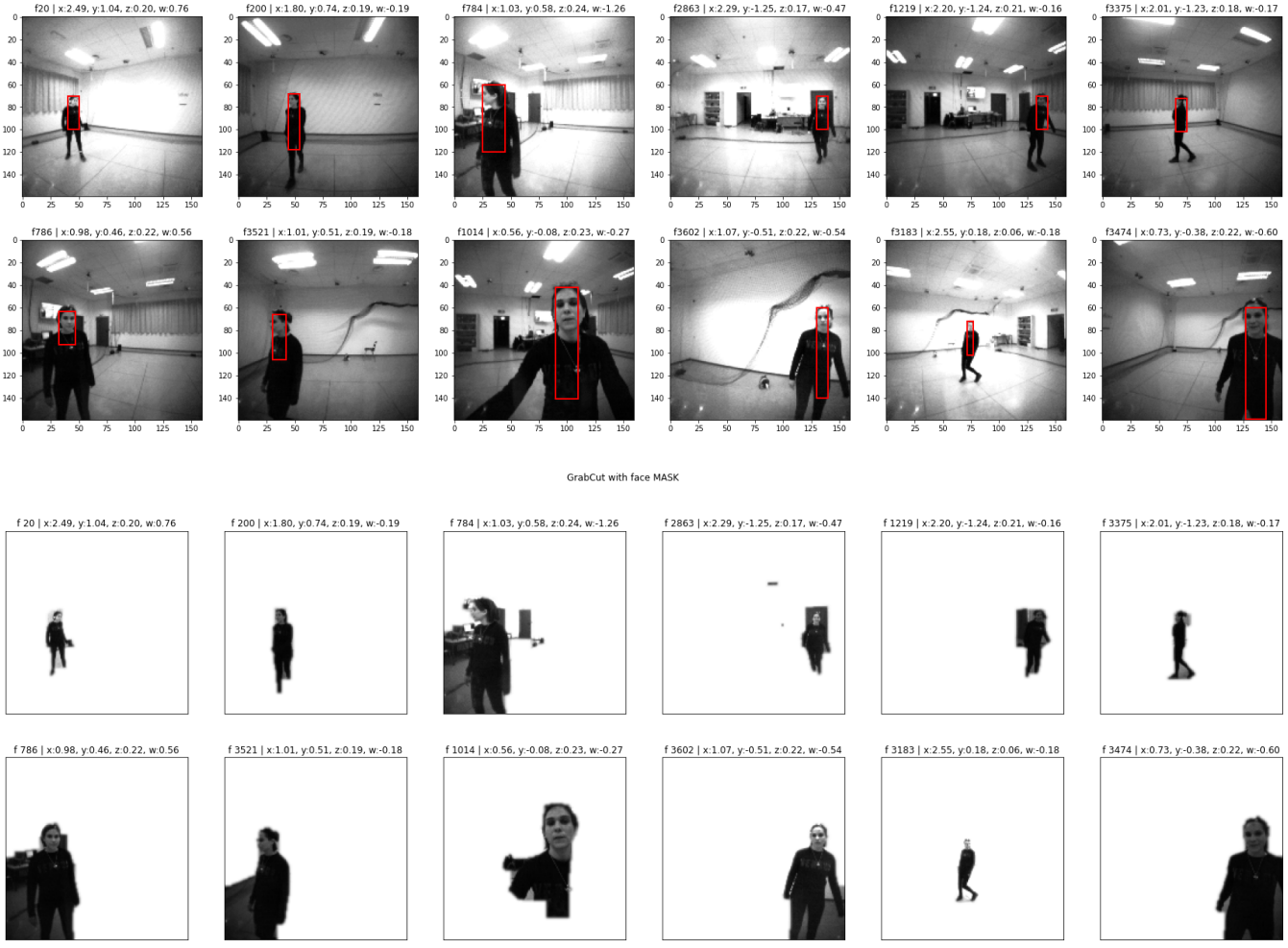
Please notice that *sure-background* is dark blue, *probable-background* is green, *probable-foreground* is yellow and *sure-foreground* is light blue.

Results are sub-optimal on the previous samples.



Test on Nicky's dataset

Since the real work will be done on Nicky's dataset, let's check that the algorithm is able to also remove the background on those images. They are grayscale with a different size, but GrabCut is still able to achieve a good work on them, as it is shown below. Please notice that this dataset contains a larger variety in user's pose, especially for the x variable. This means that the size of the face and the body may differ in different frames, something to keep in mind when drawing the rectangle manually.



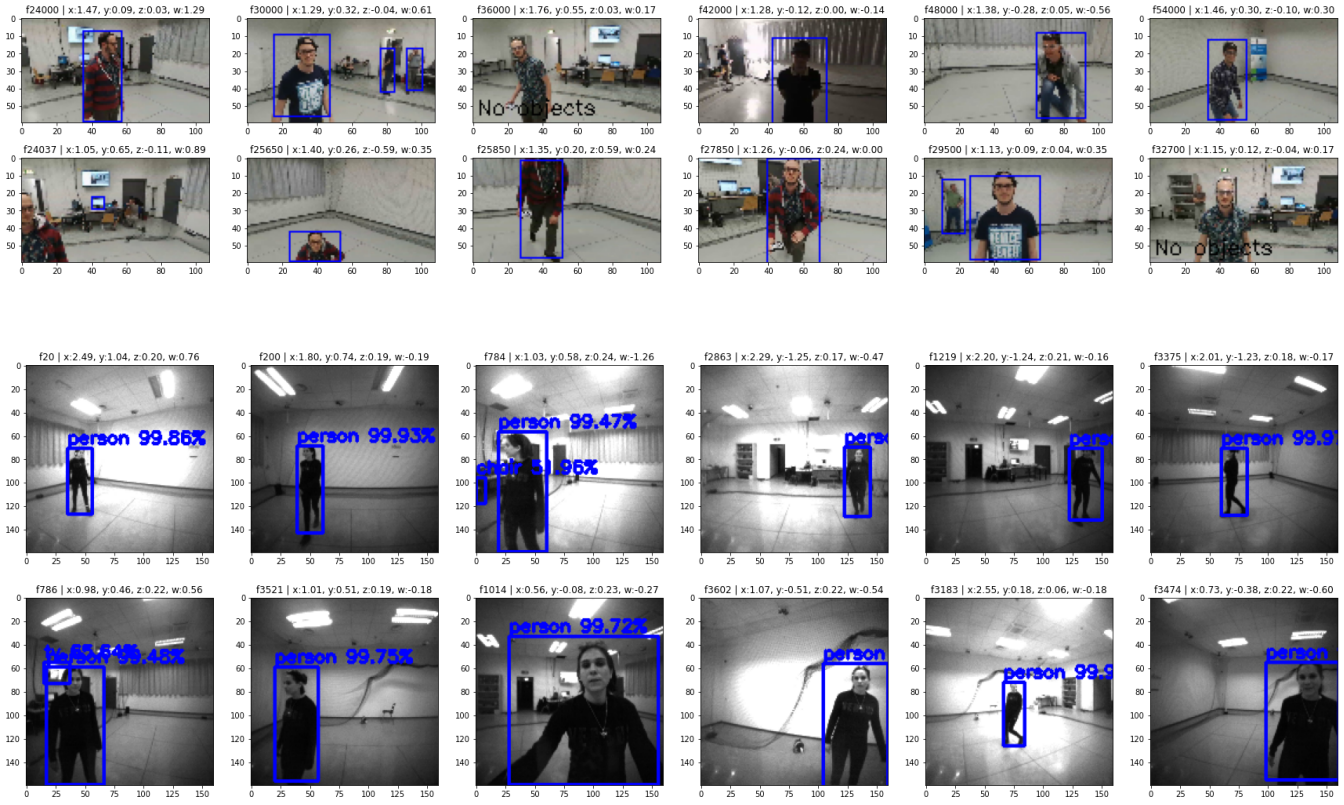
Human Detection

Now that we have a promising background removal algorithm, to let it work we need to infer person and face position from the image. As we already said, pose data is not sufficiently precise for providing such information, so we try other SOTA object detection techniques.

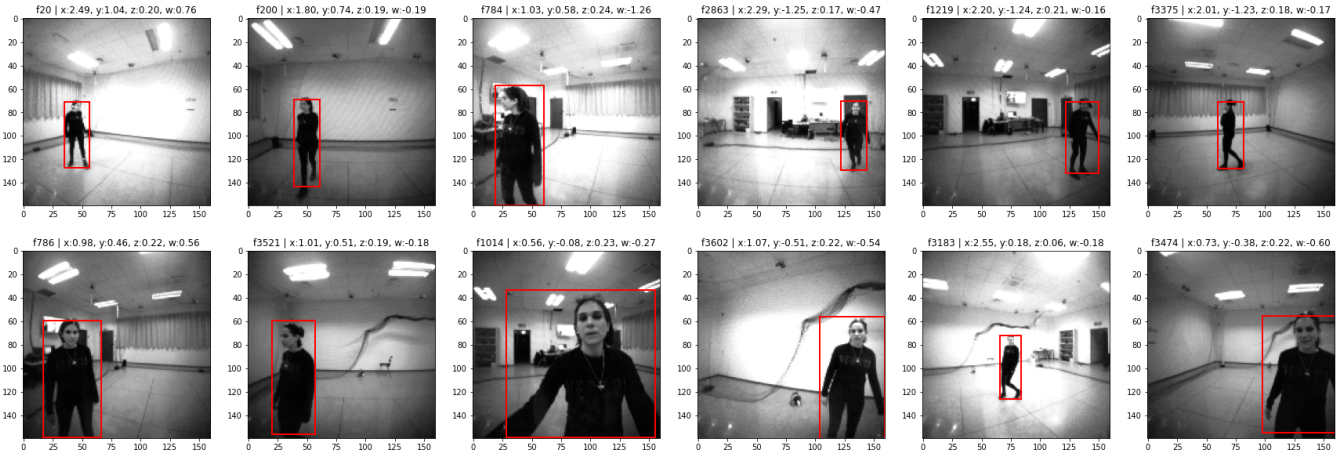
CVlib for person detection

Open source library (article, code and doc) that applies a YOLOv3 model trained on the COCO dataset, capable of detecting 80 common objects in context. Underneath it uses the OpenCV dnn module.

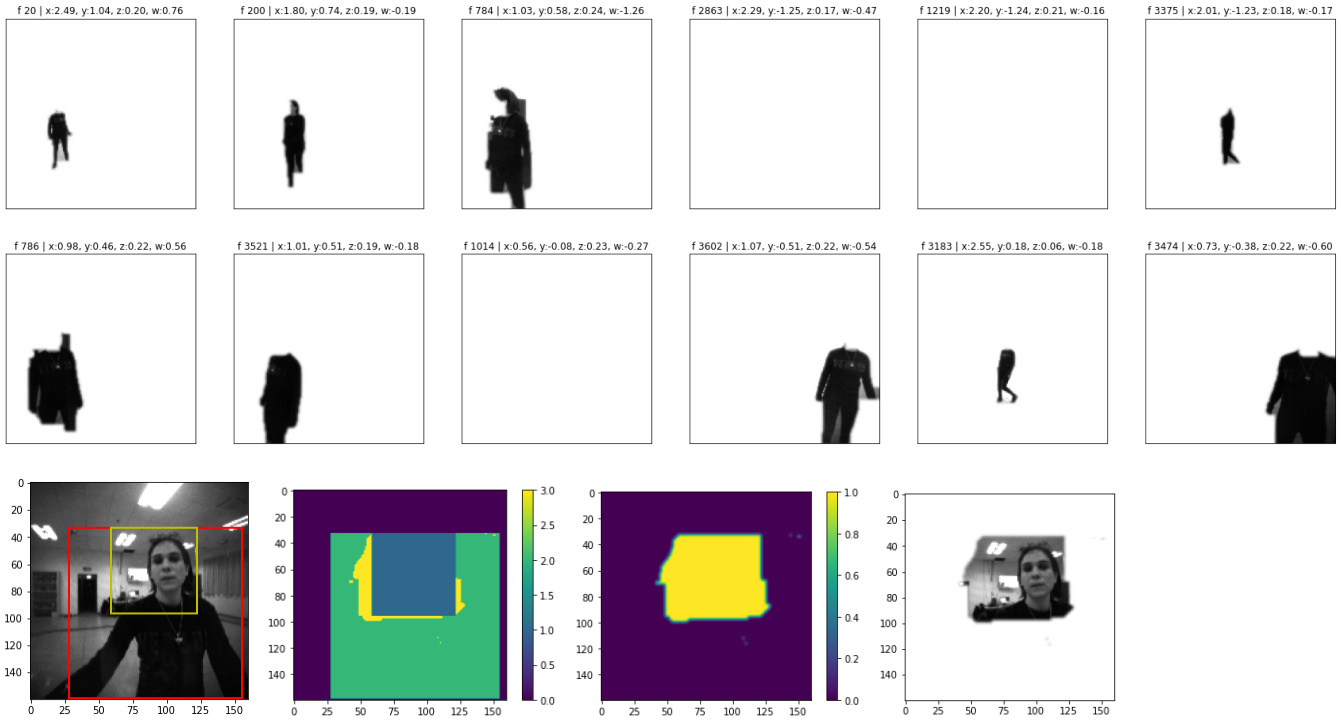
As shown below, the algorithm is able to detect **multiple objects**. However, Nicky's dataset seems to be easier to process compared with the Dario's one, which sometimes gives errors (no person is detected). If multiple objects are present, **the largest is chosen** at the moment.



An example of **YOLO + GrabCut** application is shown below. As said before, the hybrid initialization is the most powerful, but since we have very small images, **no face detection is possible**. Thus we simulate such an approach by **inferring the head + body position heuristically**. We draw a rectangle that starts at the top of the person, centered, with half the size of the person both in width and height. It often reveals to be a **good approximation** of the real head position.



GrabCut with person RECT

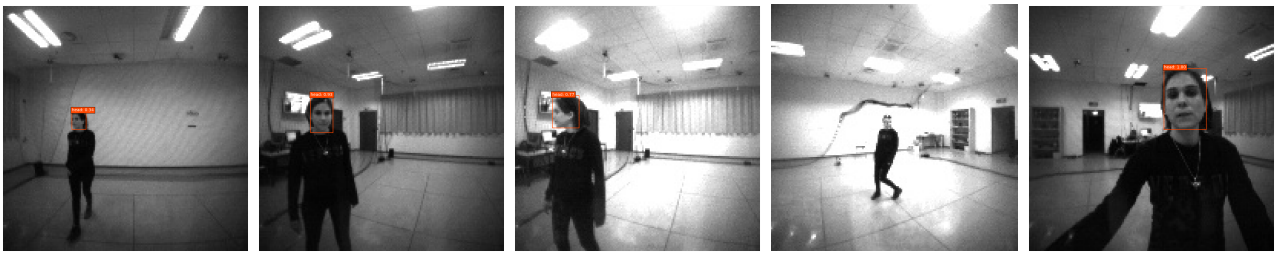


GrabCut with face MASK

SSD-based head detector

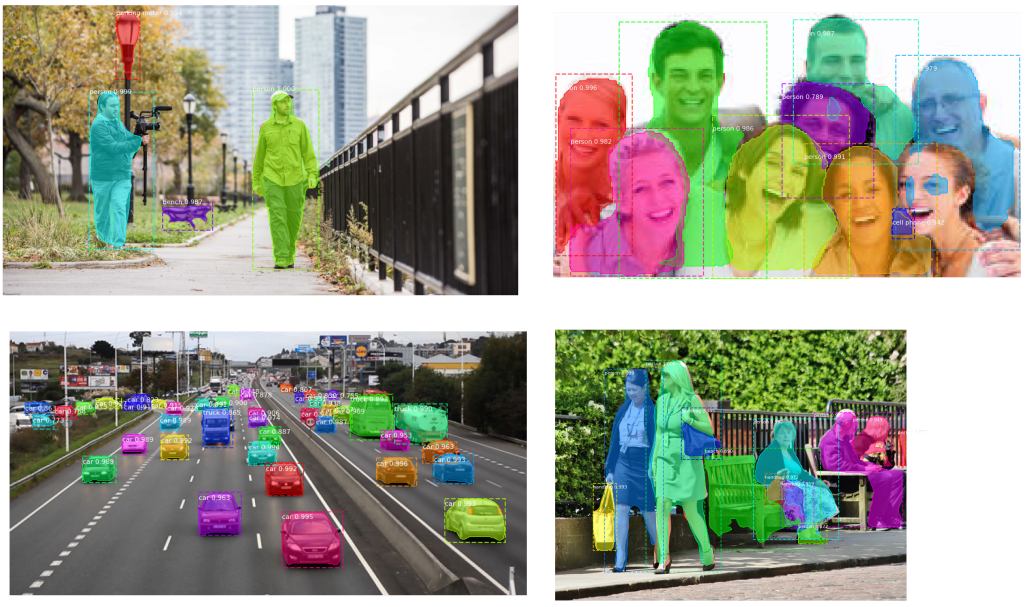
Open source library (code) based on the Single Shot Multibox Detector (SSD) for head detection. The model has been trained using the Hollywood Heads dataset as positive samples, and a subsample of the EgoHands dataset as negative samples. This model has been developed using Pierluigi Ferarri's Keras implementation of SSD as primary source (of which we provide some essential code), and replicates the original Matconvnet version of our model.

See this notebook for a demonstration. At first sight, it seems the approach does not work on very low quality images, so it fails on the Dario's dataset. Below are some examples on Nicky's dataset. Please note that the model fails on the fourth image since the head is too far and the image too small.

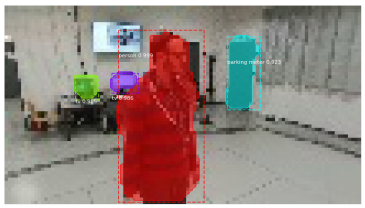


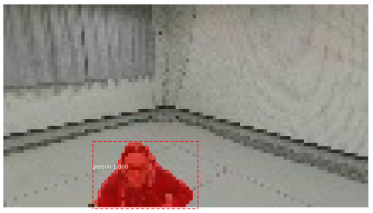
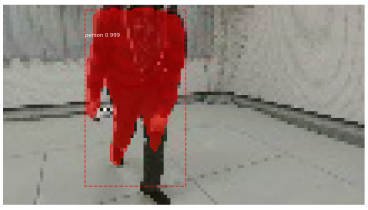
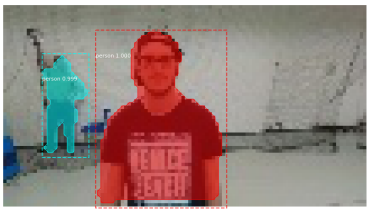
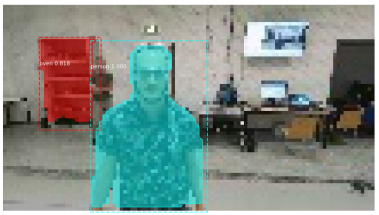
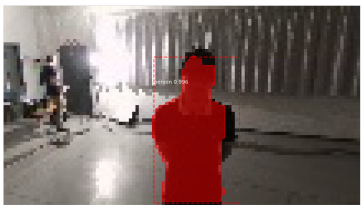
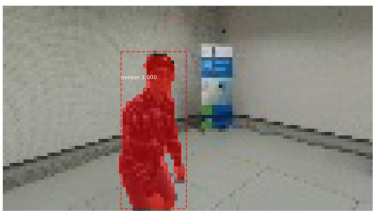
Mask R-CNN for Object Detection and Segmentation

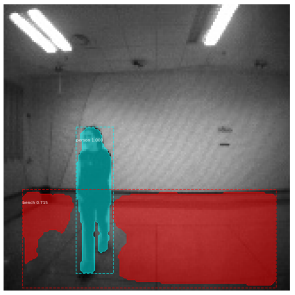
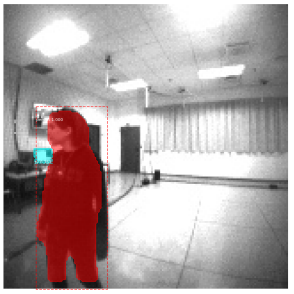
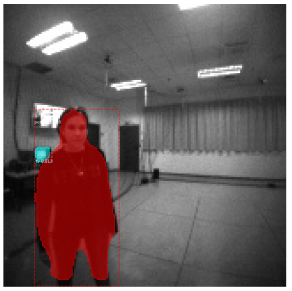
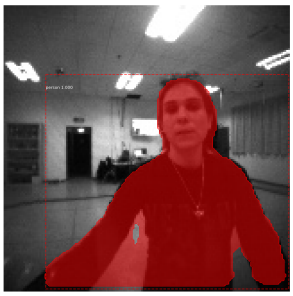
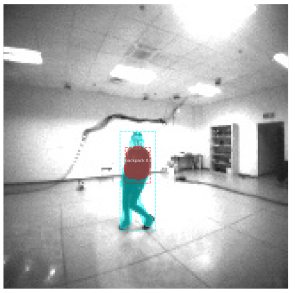
Open source library (code) that is an implementation of Mask R-CNN on Python 3, Keras, and TensorFlow. The model generates bounding boxes and segmentation masks for each instance of an object in the image. It's based on Feature Pyramid Network (FPN) and a ResNet101 backbone.

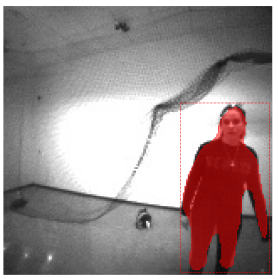


See the examples below for a demo on our datasets.
It performs extremely well on both Dario's and Nicky's dataset.





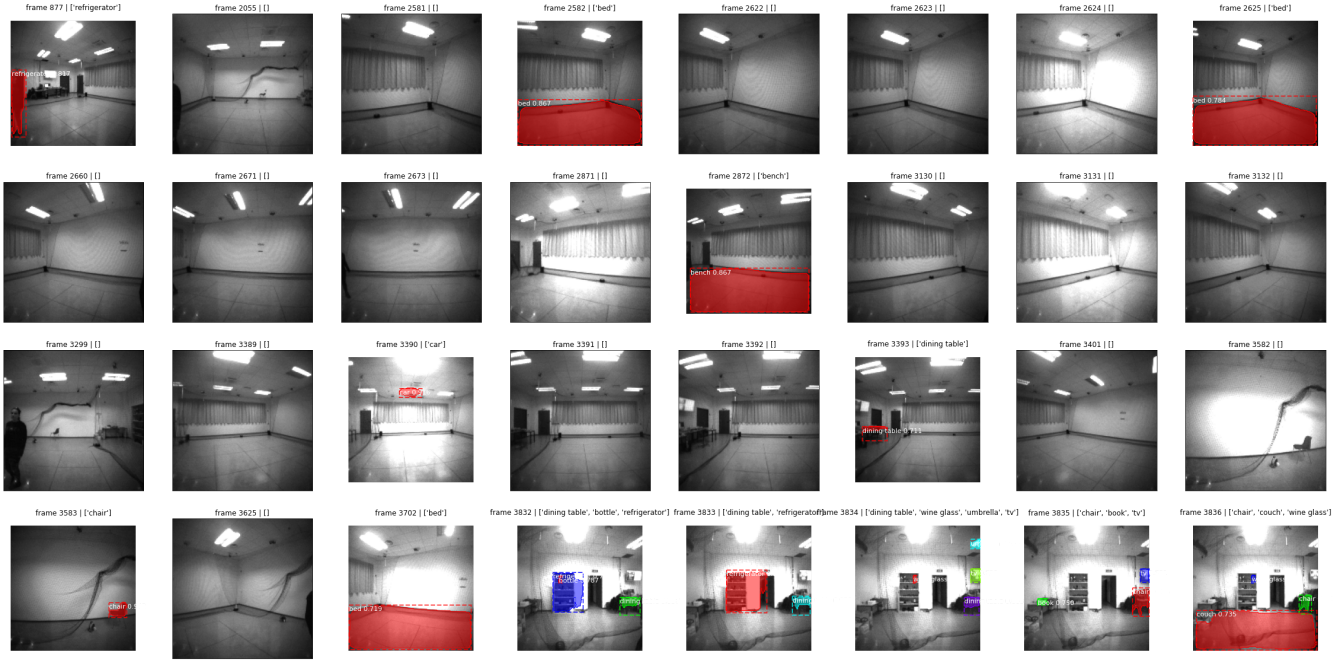
Mask R-CNN Inference

Nicky’s dataset

This section provides an overview about the model inference on Nicky's dataset, composed of a **total of 4120 images** taken inside the Drone Arena and just featuring Nicky.

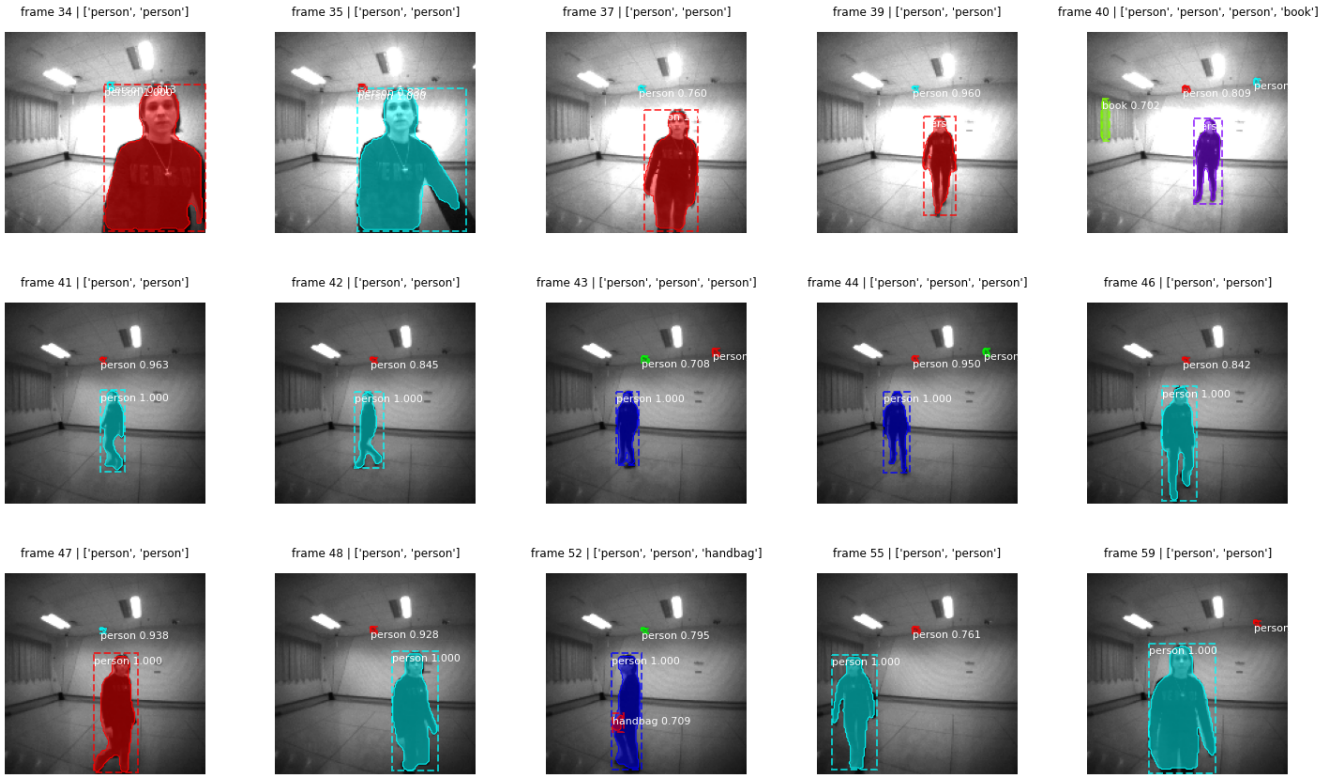
No person detected

There are **32 images** in the dataset for which the model does not detect any person.

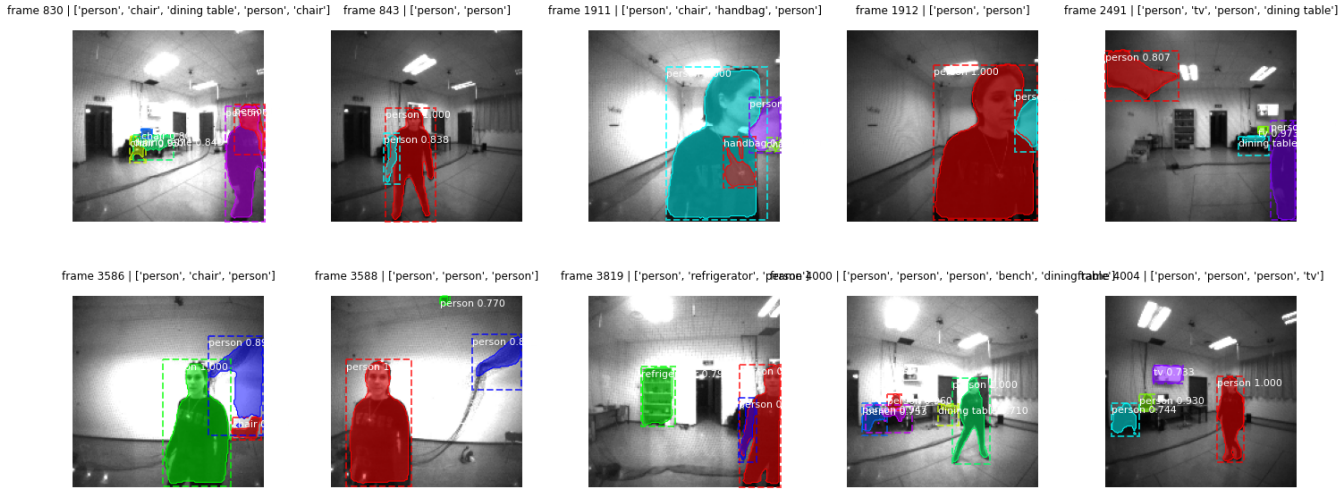


Multiple people detected

The model detects multiple people in **517 images**, but most of them are just detection errors. As you can see below, it often happens that a person is detected as a very small part of the ceiling.

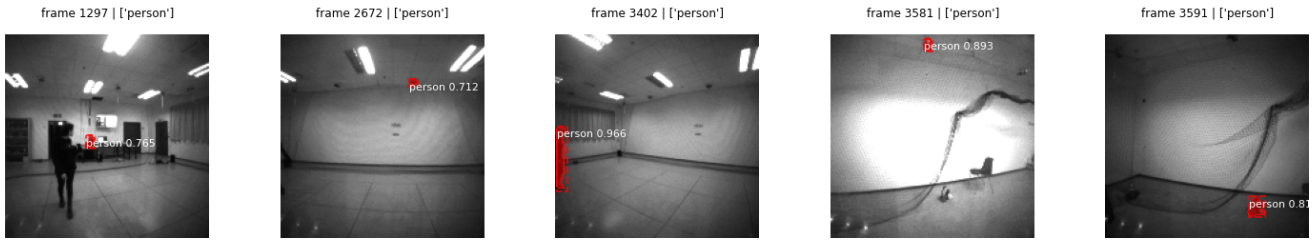


Filtering out cases in which a detected **person area is smaller than 500**, we just got these 10 results of multiple people detection. Analyzing them we see that they are false positives, so we conclude that **no image with multiple people** is present in Nicky's dataset.



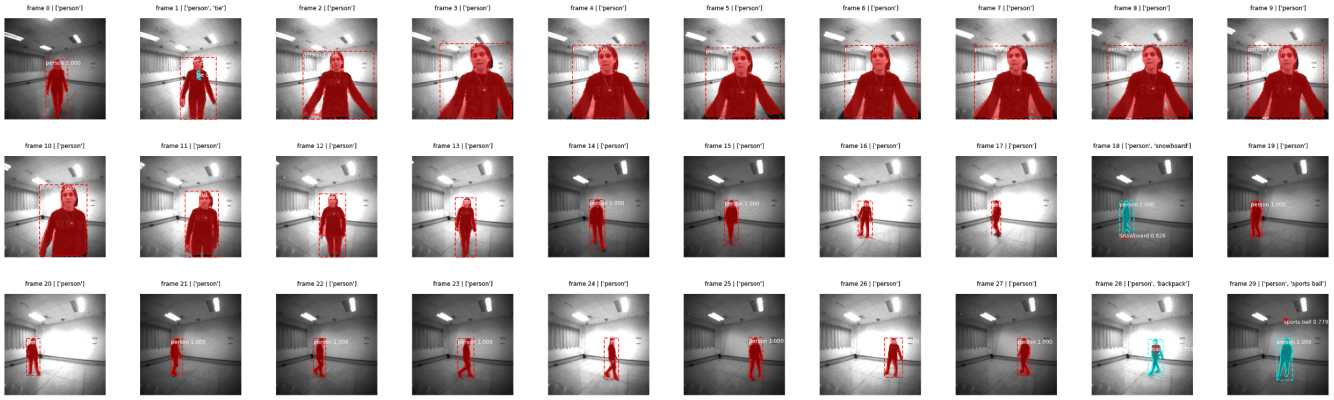
False positives

There are just **4 images** in which a person is detected while none is actually present in the image.



Correct detection

There are about **4070 images** in which there is a single person, regardless of the number of people actually detected (we are filtering out small people detected, so the false positives).



Centroid

The bounding box can also be used for computing the centroid of the largest person in the image. This information can be used for estimating the person's yaw with respect to the drone. An example is shown below.