IU International University of Applied Sciences
Course of Study: Applied Artificial Intelligence
Course: Project Computer Vision

Case Study
Project Computer Vision: Task 1

written by

Laurenz Lattermann – 4239646

https://github.com/l-lattermann/realtime-mask-detection.git

https://www.youtube.com/watch?v=zNIX0ZhWMbA

Introduction 1

1 Concept 2

1.1 Object detection basics 2

1.2 Mask and person detection 2

1.2.1 Suitable models 3

1.3 Distance measurement 3

2 YOLO – You only look once 5

2.1 Net structure 5

2.2 Training 6

2.2.1 Training YOLOv8 and YOLOv11 6

2.2.2 Data sets used 6

2.2.3 Hardware requirements 7

2.2.4 Cloud solutions 7

2.2.5 Training results 7

2.3 Repository content and structure 8

3 Additional functionalities 9

4 Conclusion 10

Literature List 1

Appendix 1

Introduction

COVID-19 has had a major influence on society, on our health system and on the way we view social interaction. The challenges that the pandemic posed to our modern world were multifaceted. One of them was implementing strict social behaviour rules to midigate the spread of the virus. Changing social behaviors and human interaction in just a few weeks under big pressure and risk of high losses of social integrity and even human lives raises the question for efficient technical solutions, frameworks, or interventions to mitigate negative consequences and ensure stability. This casestudy aims at exploring computer vision based technical aids that could have been useful during the COVID-19 pandemic and even can be useful in future applications of similar kind. Automatic computer vision based human surveillance is a ethically highly sensitive topic. There is a big variaty of questions to ask and risks to consider, i.e. pricavcy, efficacy, individual freedom etc. before implementig such systems in a real word scenario. Nevertheless, this casestudy will purely focus on the technical aspects of such systems, explore the theoretical background to computervision and present a exemplary prototype.

The overarching goal of this study is to implement real-time mask detection in a live video feed coupled with a basic distance measuring method, to automatically detect whether people comply with COVID-19 social distancing rules. The computer vision task, that is the automatic detection of masks and persons (object detection), is accomplished by leveraging state-of-the-art (SoA) computer vision algorithms like YOLOv8n (You Only Look Once, version 8 nano). The conceptual phase also includes considerations about different algorithms and highlights differences between them. During training, the generic open-source models are further trained or fine-tuned for the actual purpose, and the training results as well as the training parameters are explored.

Object detection and deep neural networks present the main focus, but this study focuses also on the implementation of distance mesuring in images and the theoretical basis behind it. Although distance measuring sounds straight forward at first, there are some key challenges to overcome. These include deducing object sizes and distances from raw image data, without having any refernce points and inferencing physical camera parameters, in scenarios where they are not known.

The conlclusion will reflect on the model choice and the training process. It highlights some of the problems that were encountered during this project and highlight learnings made underway. It will also give a future perspective on the topic and point a direction how the foundation, that was layed here can be used and improved.

1 Concept

The task of mask and distance control in a video live feed can be split in two distinct steps. One: The detection whether a person wears a mask or not. And two: Calculating the distance between the detected persons. Taking it further, these two steps can be broken up even more. Whereas distance measurements are purely algebraic calculations, that are rather intuitively, detecting person in image data can requires more profound operations, or one could say “algorithms”. The following outlines the theoretical basis of this process.

1.1 Object detection basics

Object detection is the called the process of detecting multiple instances of an object in an image and drawing a bounding box around them (Michelucci, 2019, p. 198). This seems easy to use humans but is complex to describe in mathematic operations. The most common choice are CNN’s (Convolutional Neural Networks). CNN’s consist of different layers, stacked on top of each other, that take a image as input and perform many mathematical operations on the image. Convolution is one of them, hence the name. The decision making is located in so called fully connected layers, which consist of neurons each liked to each other (Michelucci, 2019, p. 105). Each layer has weight, in the convolutional layers these are the filters itself, in the connected layer each neuron has a weight (Michelucci, 2019, pp. 109). These weights are the target of the training process, during which they get adjusted until the output is satisfying.

There are many prebuilt models available under a open source license. Most of them are pretrained and can already detect many different classes of objects.

1.2 Mask and person detection

Trying to distinguish between individuals that wear a mask and those who don’t, brings up two solutions. One is to detect a person and then perform a second check whether this person is wearing a mask. In relation to this project, this approach offers some advantages. Person is a class that is pretrained in many models, so the accuracy will be high without further training effort. Another advantage is – since distance measuring has to be performed afterwards – that the measuring part could be based on the person bounding boxes and can function even without masks in the image. There are considerable disadvantages to this approach. At first, detecting persons and then masks, takes two separate passes through a network, which would increase the computational cost immensely. This is hard to justify for a real time application. Training a model on person and mask simultaneously is also challenging, because it is very hard to find free, high quality training data containing *person* and *mask.*

The second solution is training a model on two classes – *mask* and *no mask.* This much simpler and turned out to be even better suited for distance calculation. The dataset availability great as well.

1.2.1 Suitable models

For real-time object detection applications, the choice of model depends on the trade-off between speed, accuracy, and computational efficiency. The model taken into consideration for the object detection task were YOLOv8 (You Only Look Once), SSD (Single Shot Detection) and Faster RCNN (Region-based Convolutional Neural Network).

As Kaliappan et al. (2023) state, YOLOv8 uses an anchor-free grid-based detection system, making it fast and efficient for real-time applications. SSD detects objects at multiple feature map scales, balancing speed and accuracy better than Faster R-CNN. Works well for mobile and embedded systems, but less accurate than YOLOv8 for small objects. Faster R-CNN uses a Region Proposal Network (RPN) for high-accuracy detection but at the cost of slow inference speed. Narrative: Kaliappan et al. (2023) examined the performance of those models and concluded that YOLOv8 has the best balance between accuracy and speed. So YOLOv8n and YOLOv11n were chosen for this project.

1.3 Distance measurement

Measuring the distance between two objects in theory is fairly simple. We can use the formula for the Euclidian Distance to do so.

$$\Delta d = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}$$

$\Delta d: distance\ total$
$\Delta x: distance\ in\ x-direction$
$\Delta y: distance\ in\ y-direction$
$\Delta z: distance\ in\ z-direction$

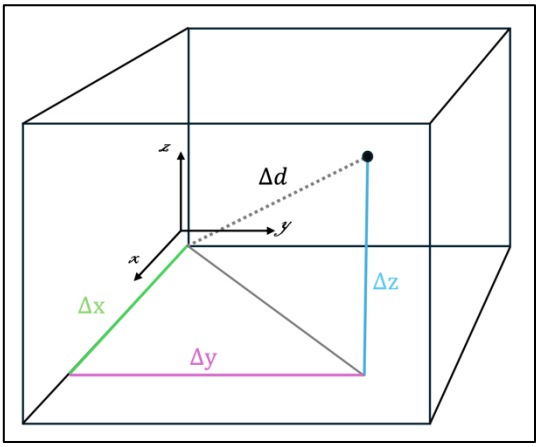


Figure 1 - Euclidean distance in 3D, created by the Author

The x- and y-coordinates can be read out easily, when a reference object with a distinct size is given. I the case of this project, we can use the size of the face bounding boxes as reference objects, since face have a fairly uniform width. Therefore we can obtain Δx (in pixels) and Δy (in pixels) by simply reading out the image data. These lengths can then be converted to real lengths by using the estimated face size:

$$\Delta X = \Delta x * \kappa$$

$$\kappa = \frac{w_{face}}{W_{face}}$$

$\Delta(X,Y,Z): Length\ in\ cm, \quad \Delta(x,y,z): length\ in\ pxl, \quad \kappa: Conversion\ factor$
$w_{face}: Bounding\ box\ width\ in\ pxl, \quad W_{face}: estimated\ average\ face\ width\ in\ cm$

Estimating the distance of an object to the camera with a single image or frame is not trivial. This process is often called *Monocular Depth Estimation.* It requires cues such as perspective, object sizes, and atmospheric effects. This makes it an ill-posed problem, as multiple depth solutions may exist for a given image (Eigen, Puhrsch, & Fergus, 2014). There are many approaches of achieving

distance estimations in image data. Most common are special sensors or techniques such as LIDAR (Light Detection and Ranging). When there is only one camera available – like in this project - the problem of depth estimation becomes even more challenging. *Monocular Depth Estimation* without any precise reference objects or scales is a task that can be accomplished by CNN's and deep learning algorithms but not by simple calculus (Chanduri et al., 2021).

Besides the difficulties in monocular depth estimation, there are some less accurate ways of estimating distances based on approximated camera specs. Since depth estimation plays only a minor role in this project, this less accurate solution is sufficient. For this approach we will assume the camera to be a pinhole camera. The pinhole camera model describes the geometry of perspective projection, where a pinhole selects light rays, forming an inverted image on the image plane. To simplify computations, an idealized virtual image plane is placed in front of the pinhole, making the image appear upright. As Tomasi (2016) explains in his work, following the projection geometrics, the image coordinates realate to the real world coordinates as follows:

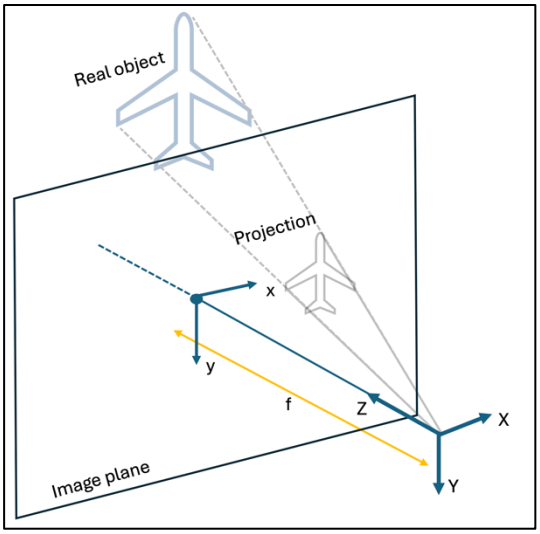


Figure 2 - Pinhole camera perspective projection. (x,y): image coordinates, (X,Y,Z): Projection coordinates, (f): focal length. (Derived from Tomasi, 2016)

$$x = \frac{fX}{Z} \quad , \quad y = \frac{fY}{Z}$$

$$x, y : image\ coordinates, \qquad X, X, Z : real\ coordinates, \qquad f : focal\ length$$

Based on this realtion we can derive the width and therfore the objects distance to the camera as follows:

$$w = x_2 - x_1 \Rightarrow w = \frac{fX_2}{Z} - \frac{fX_1}{Z} \Leftrightarrow w = \frac{f(X_2 - X_1)}{Z}$$

$$with\ W = X_2 - X_1 \Rightarrow Z = \frac{fW}{w}$$

$$w : image\ width, \qquad x_2, x_1 : image\ coordinates, \qquad W : real\ width, \qquad X_2, X_1 : real\ coordinates,$$

$$Z : real\ distance\ to\ camera, \qquad f : focal\ length$$

The distance to the camera can therefor be approximated with the focal length, the width of the object in the image plane and the real world witdth of the object. The real world object width can well be estimated in our use case, since masks or faces have a fairly uniform size and hard coded as average mask size. The image object width can be measured exactly. To determine the focal length of the camera, we can use the equation above like follows:

$$Z = \frac{fW}{w} \quad \Leftrightarrow \quad f = \frac{Zw}{W}$$

The real world distance can be obtained in a calibration process during programm startup. Therefor we hard code a claibration distance (i.e. 100cm) and record a single mask detection frame. The focal length can then be calculated like follows:

$$f = \frac{100cm * w_0}{W_0}$$

w_0: $bounding\ box\ width\ at\ 100cm$, W_0: $average\ face\ size$

By combining the formula for the Euclidean Distance and the formula for the Monocular Distance estimation we obtain:

$$\Delta D = \sqrt{\Delta x^2 + \Delta y^2 + \left(\frac{fW}{w}\right)^2}, \quad f = \frac{Zw_0}{W_0}$$

This Formula was used as a approximational distance measure in this project.

2 YOLO – You only look once

YOLO is a real-time object detection framework that frames detection as a single regression problem rather than repurposing classifiers. Unlike traditional methods, YOLO predicts bounding boxes and class probabilities directly from an entire image in one forward pass. This unified architecture allows real-time processing. While YOLO may introduce more localization errors, it reduces false positives and generalizes well across domains, outperforming traditional methods like R-CNN (Redmon, Divvala, Girshick, & Farhadi, 2016).

2.1 Net structure

The image below exemplary depicts the architecture of YOLOv5 to convey the basic architecture of YOLO networks. It takes a 448×448 RGB image as input and performs a series of convolutional and max-pooling operations to extract hierarchical features. The initial layers apply 7×7 and 3×3 convolutions to capture spatial and edge details, followed by multiple 1×1 and 3×3 convolutions to refine features. As the network deepens, the spatial dimensions decrease while the depth (number of filters) increases. The final layers flatten the extracted features into a 4096-dimensional vector, followed by fully connected layers, which outputs a 7 × 7 ×30 tensor of predictions (Redmon, Divvala, Girshick, & Farhadi, 2016).

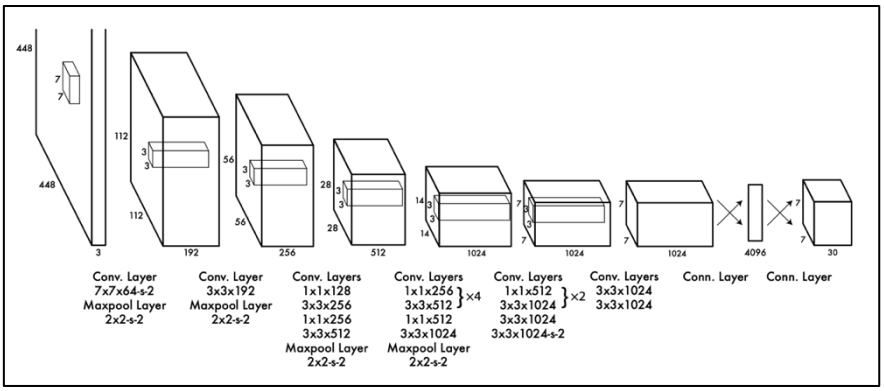


Figure 3 – Net architecture of YOLOv5 (Redmon, Divvala, Girshick, & Farhadi, 2016)

2.2 Training

Training a Convolutional Neural Network (CNN) requires a labeled training dataset. This dataset consists of images with ground truth labels, which are passed through the network. The difference between the predictions and the ground truth is evaluated using a loss function. Based on the loss function's output, the model's parameters are updated via backpropagation. In the case of CNNs, these parameters are weights in the fully connected layers and convolutional kernels in the convolutional layers. An optimization algorithm determines how much and in which direction the parameters will be changed to achieve efficient training and convergence (Yamashita et al., 2018).

2.2.1 Training YOLOv8 and YOLOv11

YOLOv8 and higher is distributed as a Python package and command-line interface (CLI), streamlining the process of model training, validation, and deployment (Yaseen, 2024). For this project the Ultralytics package was used. This package is the official library for training and deploying YOLO models. It provides a streamlined training process and supports custom training data sets in various formats.

For this project, YOLOv8n and YOLOv11n were trained with equal training parameters each on two datasets. One mask dataset and one person dataset. Although, the person-detection model was not used in the end due to low efficiency.

2.2.2 Data sets used

The choice of a high-quality dataset is one of the most crucial aspects of the training process. The dataset must represent real-world variations in class appearance, shape, angle, and scale. It should also include noise, rotations, and other distortions that mimic low image quality. Additionally, the number of images in the dataset must be sufficiently high to capture real-world variance and prevent excessive bias due to random image selection. Finding the right dataset was also a major challenge for the previously mentioned approach, where both persons and masks needed to be detected. No available dataset met the required quality criteria mentioned above. Two datasets were used for this project. One containing the classes: mask, no_mask and improper_mask. The other dataset contained only the class person. Both of the datasets were obtained from the roboflow universe. Links to the datasets can be found in the Appendix.

Augmentation	Person	Mask
Image count	2545	1716
Crop (Min-Max Zoom)	N/A	0% Min, 15% Max
Shear (Horizontal/Vertical)	N/A	±5° Horizontal, ±5° Vertical
Hue	N/A	-25° to +25°
Saturation	N/A	-25% to +25%
Brightness	N/A	-10% to +10%
Exposure	-20% to +20%	-5% to +5%
Blur	N/A	Up to 1px
Noise	Up to 8% of pixels	Up to 15% of pixels
90° Rotate	Clockwise, Counter-Clockwise	N/A

Figure 4 - Augmentations of the training data sets

2.2.3 Hardware requirements

The training process of the models posed a significant challenge to the available hardware. Initially, training was performed on an Apple Silicon M4 with MPS (Metal Performance Shader) and on a NVIDIA GTX 1060 with CUDA (Compute Unified Device Architecture) support. Nevertheless, training time for 100 epochs was well about 20 hours. This was very impractical for comparing training results on different training parameters.

Feature	Apple M4 GPU	NVIDIA GTX 1060
Architecture	Apple Custom GPU (Metal)	Pascal
Process Node	3nm (TSMC)	16nm (TSMC)
CUDA Cores	N/A (Uses GPU Compute Cores)	1,280
Tensor Cores	No (Relies on GPU ALUs)	No
Ray Tracing Cores	No	No
FP32 Performance	~3.5 TFLOPS (estimated)	~4.4 TFLOPS
FP16 Performance	~7.0 TFLOPS (estimated)	No FP16 support

Figure 5 - Apple Silicon M4 and NVIDIA GTX 1060 specifications

2.2.4 Cloud solutions

As Yaseen (2024), mentioned that YOLOv8 utilizes mixed precision training, a technique that allows the model to leverage 16-bit floating-point precision during training and inference and specifically mentioned compatibility with NVIDIA's A100 and T esla T4 GPU's, adopting a cloud-based solution offering these GPUs would effectively address the training time bottleneck and enable faster model iteration.

And indeed, Google Colab offered A100's and T4's. Google Colab is a cloud-based Jupyter Notebook environment that allows executing Python code without requiring local installation. It runs entirely in the cloud. One of its key advantages is the ability to leverage GPU and TPU acceleration, significantly improving the performance of machine learning and deep learning tasks. Since it comes preloaded with popular libraries like TensorFlow, PyTorch, and NumPy, users can get started immediately without worrying about setting up dependencies (Michelucci, 2019). The training time reduced to two hours for the T4 GPU and under an hour with the A100 GPU.

2.2.5 Training results

The comparison between YOLOv8 and YOLOv11 shows notable differences in performance and training dynamics. YOLOv8 completed training significantly faster, taking only 25% of the time compared to YOLOv11, making YOLOv8 much more efficient in terms of training time. However, YOLOv11 demonstrates superior optimization, achieving lower training losses across all metrics. Its box loss (0.87311 vs. 0.90888), classification loss (0.4944 vs. 0.53694), and DFL loss (1.0921 vs. 1.12003) are all lower than those of YOLOv8, indicating more refined bounding box placements and classification accuracy.

In terms of precision, YOLOv11 outperforms YOLOv8 with a value of 0.92212 compared to 0.88533, meaning it is better at minimizing false positives. However, YOLOv8 achieves slightly higher recall (0.88879) and mAP50-95 (0.62624 vs. YOLOv11's value not provided in this set), suggesting that it detects a wider range of objects and may generalize slightly better at different IoU thresholds. Both models show overfitting during the training process, which can be seen in a increase in combined validation loss. YOLOv8 shows a much higher increase in combined validation loss. Overall,

YOLOv8 achieves competitive performance, making it a strong choice for time-sensitive applications. YOLOv11, on the other hand, provides better precision and lower losses, suggesting that it could generalize more effectively and be more reliable in real-world scenarios. The training parameters can be found in a table in the Appendix.

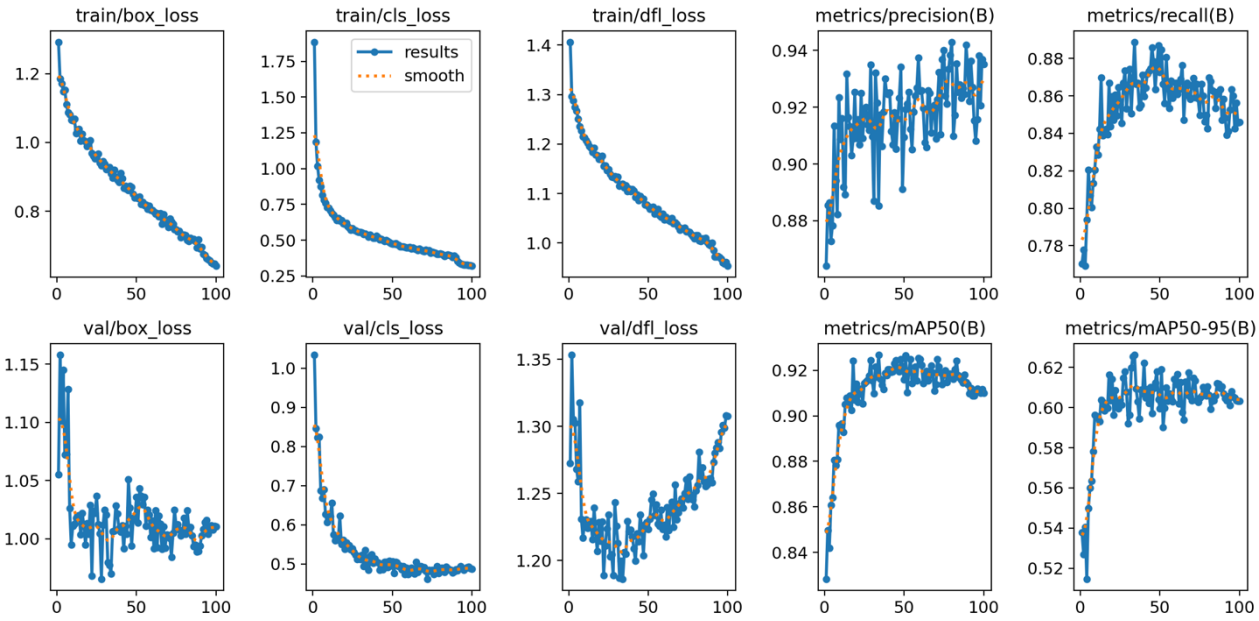


Figure 6 - YOLOv8n training progress

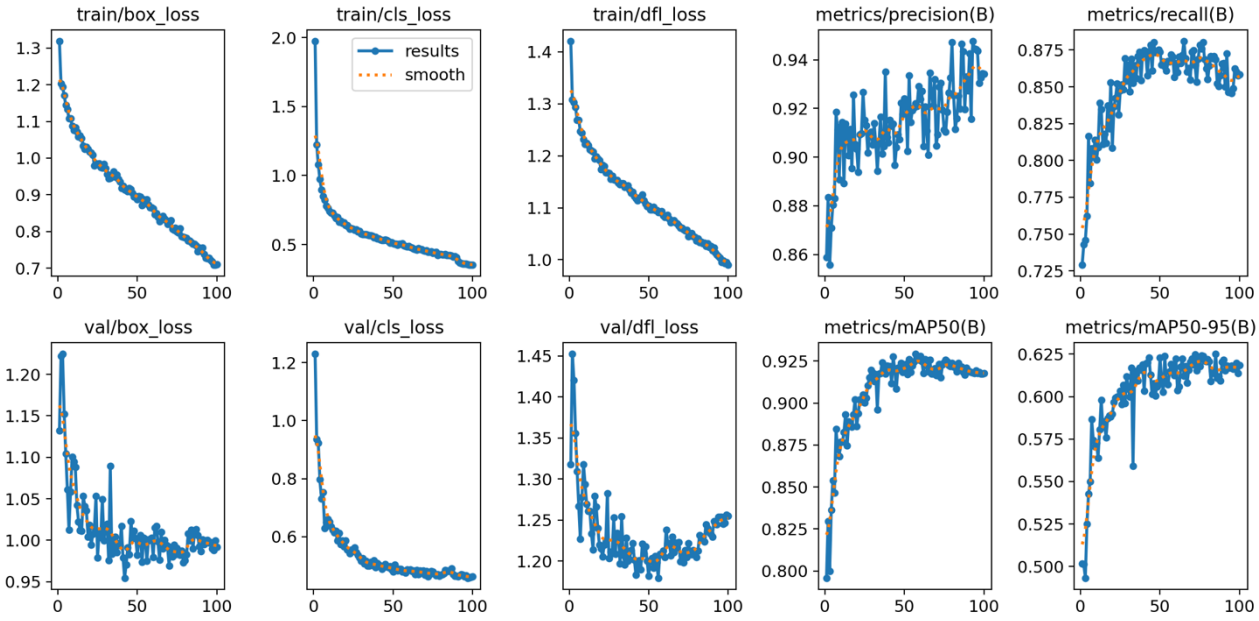


Figure 7 - YOLOv11n training progress

2.3 Repository content and structure

The following describes the structure and the contents of the project repository. Additionally to the README.md file on GitHub this aims to provide more detailed information and in depth explanation.

/data_sets

This folder contains imgage data and video data for the testing notebooks.

/models

This folder contains the best models from the training process: yolov11n_mask.pt, yolov11n_person.pt, yolov8n_mask.pt, yolov8n_person.pt. The person models where ultimately not used in the project due to efficiency limitations of two separate forward passes per frame and less acurate distance calculation based on person bounding box size.

/scripts

This folder contains the scripts used by main.py. The cam_calibration.py claculates the focal length in pixels based on a hardcoded distance of 100cm and a hardcoded average face width of 17cm. The cv2_functions.py contains all sorts of functions related to cv2, like all the bounding box overlays and the status bar displayed in the bottom of the frame. The frame_fetcher.py contains the class FrameFetcher which initializes a frame capture object in a separate deamon thread. This cuts the frame process time in half, since the frame fetching is very time intensive.

/tests

The testfolder contains:

• test_mp4.ipynb: Tests model predictions with video files from /data_sets/video_data

• test_path_handling.py: Tests if the dynamic file path handling from config.py is working

• test_webcam.ipynb: Tests model prediction with webcam as capture source

• unsterstand_yolo_tensor_structure.ipynb: A quick output analysis of the yolo output tensor

/training

This folder contains all training runs. The subfolders were automatically created by the Ulrtalytics framework. They inlcude plots of the training loss progression, the training arguments etc.

config.py

This file stores file paths using the constants and paths configuration pattern, making them dynamically available across different operating systems and preventing hardcoded path issues.

main.py

This file contains the main programm.

3 Additional functionalities

In addition to the basic detection and distance mesure functions, some practical tweeks were implemented. The application includes a little UI (User Interface) created with OpenCV functions that allow to change detection parameters like IOU and detection confidence during runtime. Also the model can be switched during runtime and it is possible to toggle on a test mode for the distance mesuring, so only one person is required to show off the functions. On top of that it facillitates a blurr mode, to ensure privacy when recording the video material.

Screenshots of the running application can be found in the Appendix, GIFs of testing on stock video and realtime testing footage can be found in README file on GitHub.

4 Conclusion

This project presented a journey through the implementation process of real-time mask detection using state-of-the-art computer vision models. With YOLOv8 and YOLOv11, the study demonstrated the trade-offs between training speed and accuracy in object detection models. YOLOv8 provided a significantly faster training time, making it a strong candidate for real-time applications, while YOLOv11 exhibited superior optimization and precision. This underlines and confirms the improvemts that YOLOv11 presents compared to YOLOv8.

The implementation of distance measurement presented challenges, particularly in monocular depth estimation. The approximation approach using bounding box sizes as reference objects and applying the pinhole camera model allowed for an functional solution. The mesurments were quiet accurate on small distances, but decreased in accuracy with greater distances. Despite those limitations of single-camera depth estimation, this method proved sufficient for the project's objectives.

Hardware constraints initially posed a major bottleneck in training time. The transition to cloud-based solutions, specifically Google Colab with NVIDIA A100 and T4 GPUs, significantly reduced training durations and enabled more efficient model iteration. This was a great learning and once more proved the importance of cloud solution nowadays.

A key takeaway from this study is that while model selection and computational efficiency are crucial, dataset quality remains a limiting factor in creating custom models. The challenge of obtaining high-quality, annotated datasets was the main limiting factor for some conceptual solutions proposed at the beginning of this project. This highlights the need for efficient or even automated creation of data sets and ground truth data.

Overall, the project successfully implemented a proof-of-concept system for real-time mask detection and distance measurement. The final solution included some functionalites like model switching, prediction frame rate adjustment and face blurring – all during runtime.

The UI functions offered a practical handling of the application and made it easy to debug.

Future work could involve integrating stereo vision or depth sensors to improve distance estimation accuracy. Also more models could be added to the model catalog to compare their performance live during runtime. Concerning the detection accuray, better datasets could be used to train the model.

Literature List

Chanduri, S. S., Suri, Z. K., Vozniak, I., & Müller, C. (2021). *CamLessMonoDepth: Monocular depth estimation with unknown camera parameters*. IEEE. https://doi.org/10.48550/arXiv.2110.14347

Eigen, D., Puhrsch, C., & Fergus, R. (2014). *Depth map prediction from a single image using a multi-scale deep network*. https://doi.org/10.48550/arXiv.1406.2283

Islam, S. U., Ferraioli, G., Pascazio, V., Vitale, S., & Amin, M. (2024). *Performance analysis of YOLOv3, YOLOv4 and MobileNet SSD for real-time object detection. The Sciencetech*, 5(2), 38-49. https://www.researchgate.net/publication/381851712

Kaliappan, V. K., Manjusree, S. V., Shanmugasundaram, K., Ravikumar, L., & Hiremath, G. B. (2023). *Performance analysis of YOLOv8, RCNN, and SSD object detection models for precision poultry farming management.* 2023 IEEE 3rd International Conference on Applied Electromagnetics, Signal Processing, & Communication (AESPC). IEEE. https://doi.org/10.1109/AESPC59761.2023.10389906

Michelucci, U. (2019). *Advanced applied deep learning: Convolutional neural networks and object detection.* Apress. https://doi.org/10.1007/978-1-4842-4976-5

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). *You Only Look Once: Unified, real-time object detection*. IEEE. Retrieved from http://pjreddie.com/yolo/

Tomasi, C. (2016). *A simple camera model*. Duke University. Retrieved February 9, 2025, from https://cs.duke.edu

Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). *Convolutional neural networks: An overview and application in radiology.* Insights into Imaging, 9(4), 611-629. https://doi.org/10.1007/s13244-018-0639-9

Yaseen, M. (2024). *What is YOLOv8: An in-depth exploration of the internal features of the next-generation object detector.* National University of Computer and Emerging Sciences. Retrieved from https://arxiv.org/abs/2408.15857

Appendix

Links to training data

https://universe.roboflow.com/group-tbd/real-time-face-mask-detection-and-validation-system-dataset/dataset/4#

https://universe.roboflow.com/titulacin/person-detection-9a6mk/dataset/16#

GitHub-Link

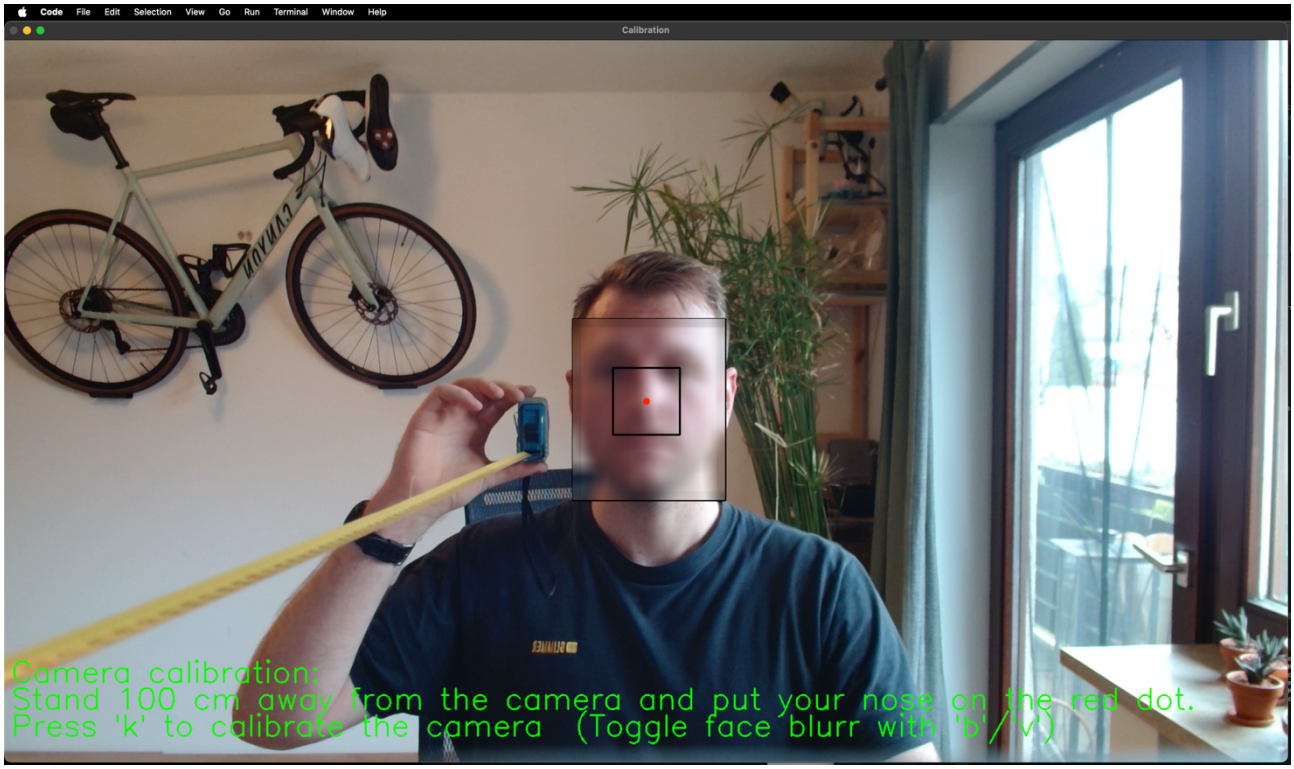
https://github.com/l-lattermann/realtime-mask-detection.git

Comparison of the training parameters

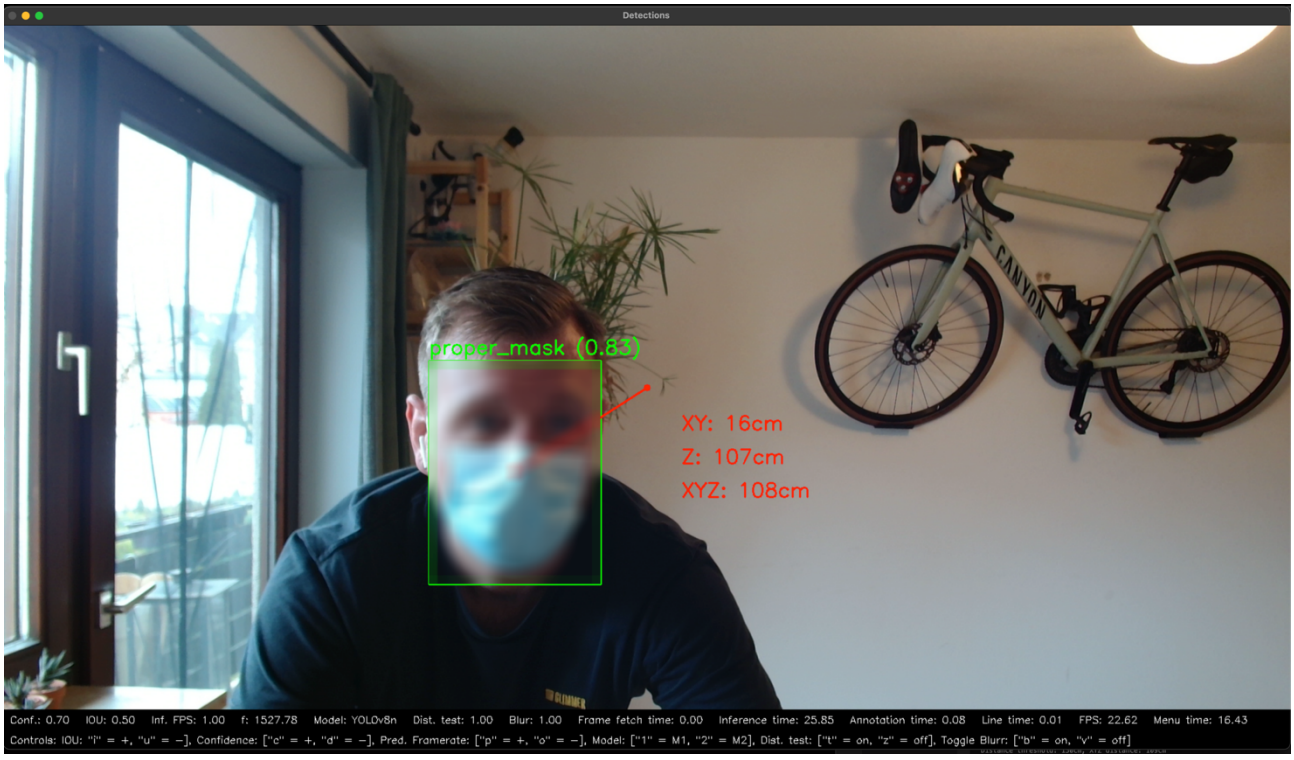
parameters	v8_mask	v8_person	v11_mask	v11_person
task	detect	detect	detect	detect
mode	train	train	train	train
model	yolo11n.yaml	yolov8n.yaml	yolo11n.yaml	yolo11n.yaml
data	data.yaml	data.yaml	data.yaml	data.yaml
epochs	100	100	100	100
time				
patience	100	100	100	100
batch	16	16	16	16
imgsz	640	640	640	640
save	TRUE	TRUE	TRUE	TRUE
save_period	-1	-1	-1	-1
cache	FALSE	FALSE	FALSE	FALSE
device	0	0	0	0
workers	8	8	8	8
project				
name	train	train	train	train
exist_ok	FALSE	FALSE	FALSE	FALSE
pretrained	yolo11n.pt	yolov8n.pt	yolo11n.pt	yolo11n.pt
optimizer	auto	auto	auto	auto
verbose	TRUE	TRUE	TRUE	TRUE
seed	0	0	0	0
deterministic	TRUE	TRUE	TRUE	TRUE
single_cls	FALSE	FALSE	FALSE	FALSE
rect	FALSE	FALSE	FALSE	FALSE
cos_lr	FALSE	FALSE	FALSE	FALSE
close_mosaic	10	10	10	10
resume	FALSE	FALSE	FALSE	FALSE
amp	TRUE	TRUE	TRUE	TRUE
fraction	1.0	1.0	1.0	1.0
profile	FALSE	FALSE	FALSE	FALSE
freeze				
multi_scale	FALSE	FALSE	FALSE	FALSE
overlap_mask	TRUE	TRUE	TRUE	TRUE
mask_ratio	4	4	4	4
dropout	0.0	0.0	0.0	0.0
val	TRUE	TRUE	TRUE	TRUE
split	val	val	val	val
save_json	FALSE	FALSE	FALSE	FALSE
save_hybrid	FALSE	FALSE	FALSE	FALSE
conf				
iou	0.7	0.7	0.7	0.7
max_det	300	300	300	300
half	FALSE	FALSE	FALSE	FALSE
dnn	FALSE	FALSE	FALSE	FALSE
plots	TRUE	TRUE	TRUE	TRUE
source				
vid_stride	1	1	1	1
stream_buffer	FALSE	FALSE	FALSE	FALSE
visualize	FALSE	FALSE	FALSE	FALSE
augment	FALSE	FALSE	FALSE	FALSE
agnostic_nms	FALSE	FALSE	FALSE	FALSE
classes				
re	FALSE	FALSE	FALSE	FALSE

embed				
show	FALSE	FALSE	FALSE	FALSE
save_frames	FALSE	FALSE	FALSE	FALSE
save_txt	FALSE	FALSE	FALSE	FALSE
save_conf	FALSE	FALSE	FALSE	FALSE
save_crop	FALSE	FALSE	FALSE	FALSE
show_labels	TRUE	TRUE	TRUE	TRUE
show_conf	TRUE	TRUE	TRUE	TRUE
show_boxes	TRUE	TRUE	TRUE	TRUE
line_width				
format	torchscript	torchscript	torchscript	torchscript
keras	FALSE	FALSE	FALSE	FALSE
optimize	FALSE	FALSE	FALSE	FALSE
int8	FALSE	FALSE	FALSE	FALSE
dynamic	FALSE	FALSE	FALSE	FALSE
simplify	TRUE	TRUE	TRUE	TRUE
opset				
workspace				
nms	FALSE	FALSE	FALSE	FALSE
lr0	0.01	0.01	0.01	0.01
lrf	0.01	0.01	0.01	0.01
momentum	0.937	0.937	0.937	0.937
weight_decay	0.0005	0.0005	0.0005	0.0005
warmup_epochs	3.0	3.0	3.0	3.0
warmup_momentum	0.8	0.8	0.8	0.8
warmup_bias_lr	0.1	0.1	0.1	0.1
box	7.5	7.5	7.5	7.5
cls	0.5	0.5	0.5	0.5
dfl	1.5	1.5	1.5	1.5
pose	12.0	12.0	12.0	12.0
kobj	1.0	1.0	1.0	1.0
nbs	64	64	64	64
hsv_h	0.015	0.015	0.015	0.015
hsv_s	0.7	0.7	0.7	0.7
hsv_v	0.4	0.4	0.4	0.4
degrees	0.0	0.0	0.0	0.0
translate	0.1	0.1	0.1	0.1
scale	0.5	0.5	0.5	0.5
shear	0.0	0.0	0.0	0.0
perspective	0.0	0.0	0.0	0.0
flipud	0.0	0.0	0.0	0.0
fliplr	0.5	0.5	0.5	0.5
bgr	0.0	0.0	0.0	0.0
mosaic	1.0	1.0	1.0	1.0
mixup	0.0	0.0	0.0	0.0
copy_paste	0.0	0.0	0.0	0.0
copy_paste_mode	flip	flip	flip	flip
auto_augment	randaug-ment	randaugment	randaugment	randaugment
erasing	0.4	0.4	0.4	0.4
crop_fraction	1.0	1.0	1.0	1.0
cfg				
tracker	botsort.yaml	botsort.yaml	botsort.yaml	botsort.yaml
save_dir	runs/de-tect/train	runs/de-tect/train	runs/de-tect/train	runs/detect/train

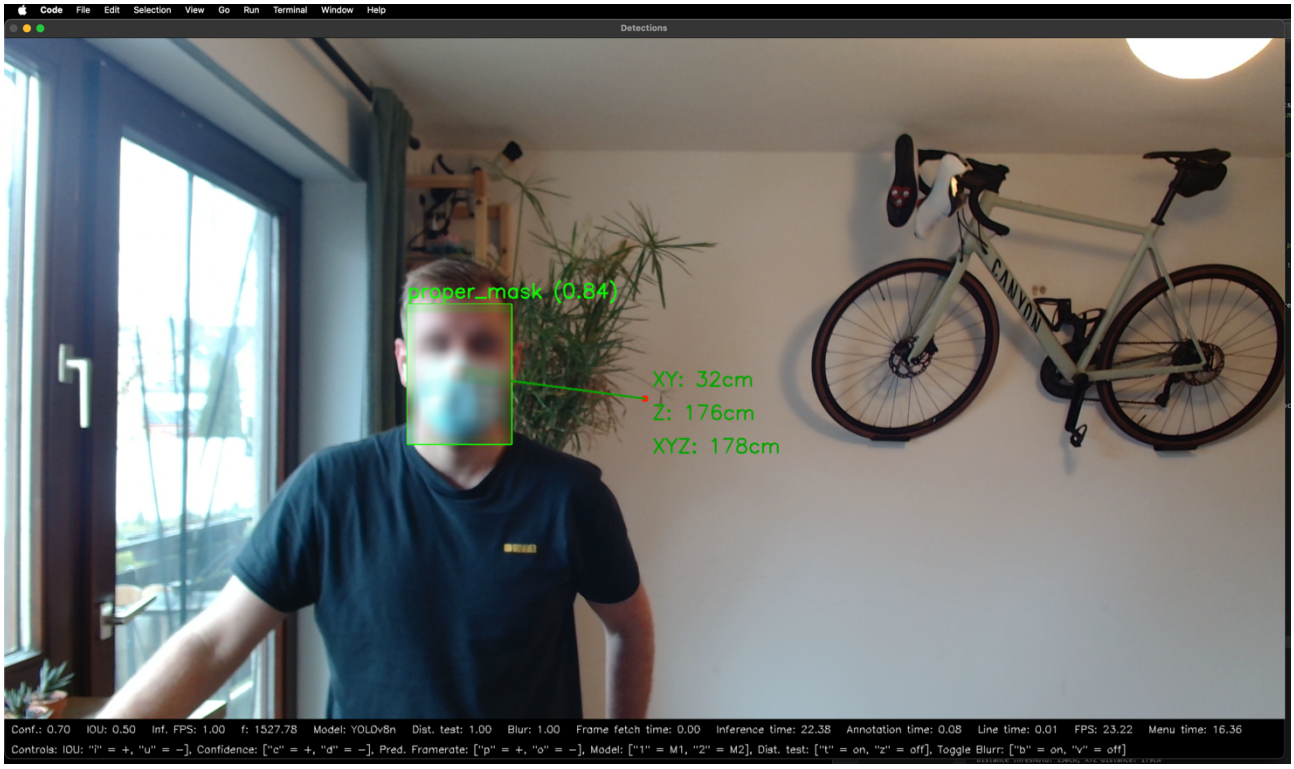
Camera calibration



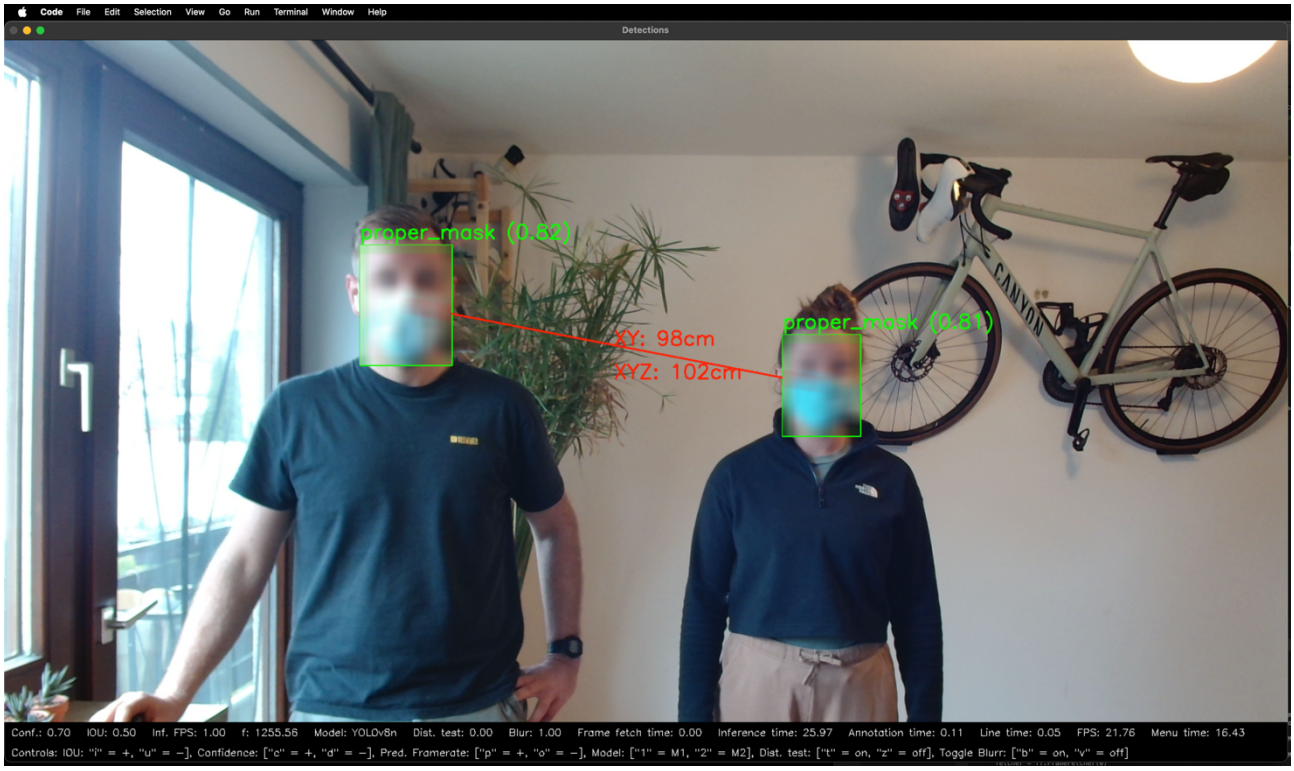
Distance measure test function – distance too small



Distance measure test function – distance great enough

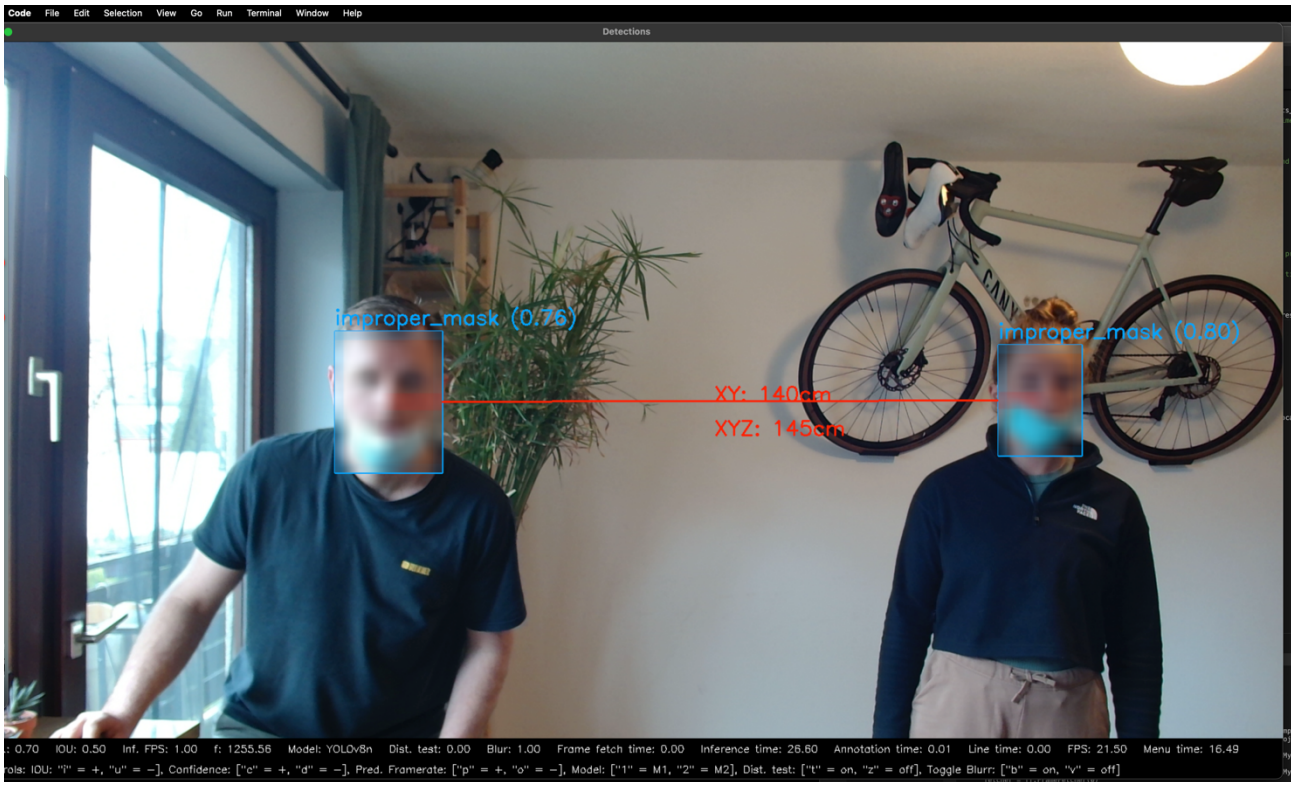


Multiple Masks detection – proper mask, distance too small

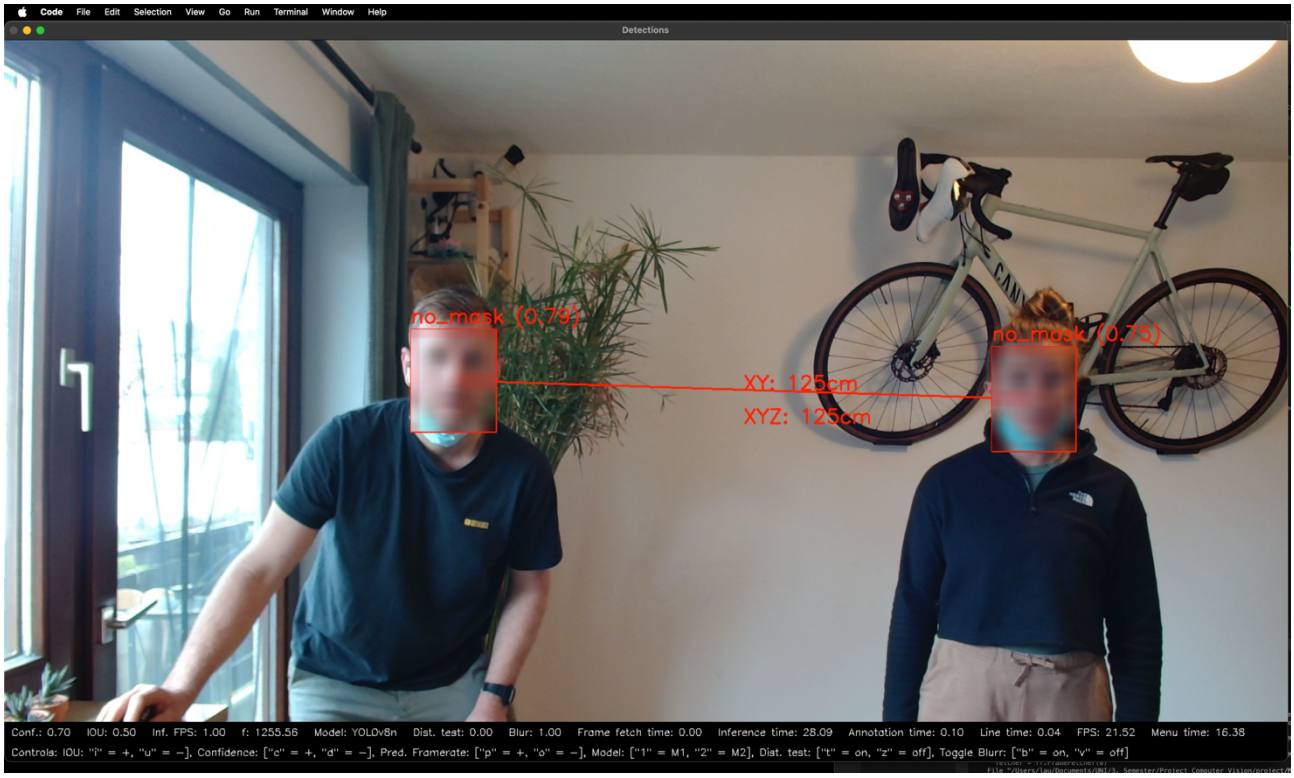


Multiple Masks detection – proper mask, distance great enough

Multiple Masks detection – improper mask



Multiple Masks detection – no mask, distance too small



Multiple Masks detection – no mask, distance great enough